FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Erstellung eines Modellbasierten Designprozesses zur Entwicklung von Cyber-Physischen Systemen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Bernhard Petschina, BSc
Matrikelnummer 1026486

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Mitwirkung: Univ.Ass. Dipl.-Ing. Bernhard Frömel, BSc

Wien, 15. November 2016

Bernhard Petschina	Radu Grosu

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Towards a Model-Based Design Flow for the Construction of Cyber-Physical Systems

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Bernhard Petschina, BSc
Registration Number 1026486

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Assistance: Univ.Ass. Dipl.-Ing. Bernhard Frömel, BSc

Vienna, 15th November, 2016

Bernhard Petschina

Radu Grosu

Erklärung zur Verfassung der Arbeit

Bernhard Petschina, BSc
Kreuzstraße 14, A-2443 Leithaprodersdorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 15. November 2016

Bernhard Petschina

Danksagung

Besonderer Dank geht an Univ.Prof. Dipl.-Ing. Dr.rer.nat. **Radu Grosu** von der Cyber-Physical Systems Group an der Technischen Universität Wien, weil er mir ermöglicht hat, an diesem Thema zu arbeiten, und für die Aufsicht der Arbeit und die Unterstützung bei der Verbesserung dieser. Ich möchte mich auch bei Univ.Ass. Dipl.-Ing. BSc **Bernhard Frömel** bedanken, weil er der erste Ansprechpartner bei Problemen war, und weil er wertvolle Tipps für diese Arbeite gegeben hat, und diese auch verbessert hat. Weiterer Dank geht an Projektass. Dipl.-Ing. BSc **Julian Grahsl** für seine Hilfe bei der Erstellung der Leiterplatten, und an Univ.Ass. Dipl.-Ing. BSc **Christian Hirsch** für seine Hilfe beim Erstellen der Bedienungsanleitungen. Ein besonderer Dank geht an Ing. **Leo Mayerhofer** für seine Hilfe bei den Bestellungen der Einzelteile, und die Unterstützung mit dem 3D Drucker. Ich möchte mich auch für seine Hilfe bei der Reparatur der Platinen mit der Heißluft-Lötstation bedanken.

Mein besonderer Dank geht an die Firma **PIU-Printex** für ihre schnelle Produktion und Lieferung, und für ihre Hilfe bei der Berechnung der Kosten für höhere Stückzahlen. Ich möchte mich bei ihnen auch für die Förderung dieser Arbeit bedanken, indem sie mir einen Preisnachlass auf die Herstellung der Platinen gegeben haben.

Zum Schluss möchte ich mich bei meinen Eltern bedanken, die mich bei meinem Studium in Wien unterstütz haben.

Acknowledgements

My greatest thanks go to Univ.Prof. Dipl.-Ing. Dr.rer.nat. **Radu Grosu** of the Cyber-Physical Systems Group at the Vienna University of Technology for enabling me to work on this thesis, and for supervising and reviewing this work. My thanks also go to Univ.Ass. Dipl.-Ing. BSc **Bernhard Frömel** for being the go-to person when problem arose, and also for giving tips for this thesis and for reviewing it. I also like to thank Projektass. Dipl.-Ing. BSc **Julian Grahsl** for his help in the design of the PCBs, and Univ.Ass. Dipl.-Ing. BSc **Christian Hirsch** for his help in the design of the manuals. I'd also like to give special thanks to Ing. **Leo Mayerhofer** for his help in the orders of the components, and for his assistance with the 3D printer. He also helped me repairing some of the boards with the hot air soldering station.

Special thanks go to **PIU-Printex** for their fast production and for their list of prices for higher quantities. I'd also like to thank them for sponsoring my thesis by giving us a special discount on the production of the circuit board.

Finally I'd like to thank my parents for enabling me to study in Vienna and for supporting me during my time as a student.

Kurzfassung

So genannte Cyber-Physical Systems (CPSs) sind Systeme, die nicht nur aus einer Software, oder einem mechanischen Teil bestehen, sondern aus der Kombination aller Teile, zusammen mit der Umwelt, in der sie eingesetzt werden. Ein Beispiel ist die Smart City, in der intelligente Verkehrsleitsysteme helfen, Staus und Unfälle zu vermeiden. In diesem Beispiel bilden die intelligenten Ampeln, die Sensoren auf den Straßen, und die Autos ein großes CPS.

Aber nicht nur solch großen Systeme werden CPS genannt, auch die Autos an sich sind solche Systeme. Heutige Autos enthalten viele Computer und Mikrocontroller, welche miteinander kommunizieren. Zum einen gibt es das System, welches den Motor mit seinen Komponenten steuert. Zum anderen gibt es Systeme, die dem Lenker und den Passagieren helfen, wie zum Beispiel ein integriertes Global Positioning System (GPS), oder Unterhaltungssysteme. Außerdem können Systeme vorhanden sein, die helfen, Unfälle zu verhindern, indem sie den Lenker warnen, oder das Auto bremsen. All diese Systeme, zusammen mit den mechanischen, elektrischen und elektronischen Teilen, bilden ein CPS.

Um solche Systeme zu bauen wird ein Designprozess verwendet, welcher hilft, die Arbeiten an die verschiedenen Abteilungen optimal zu verteilen, und Fehler früh zu finden oder gänzlich zu verhindern.

Diese Diplomarbeit beschäftigt sich mit der Definition eines neuen Designprozesses, welcher verwendet werden kann, um CPSs zu bauen. Dieser Designprozess sollte es den Abteilungen ermöglichen, parallel zueinander zu arbeiten. Ein kleiner Anwendungsfall sollte dann zeigen, dass der Designprozess praktisch angewendet werden kann. Zu diesem Zweck wurde ein kleiner Quadcopter gebaut, wobei die Kosten so gering wie möglich gehalten werden sollten, und außerdem sollte er in zukünftigen Projekten als Grundlage für Forschungsarbeiten verwendbar sein.

Um diese Ziele zu erreichen wurden folgende Schritte unternommen: Am Anfang wurden Informationen zu existierenden Designprozessen eingeholt. Danach wurde unser neuer Designablauf definiert und in einem Anwendungsfall getestet. Nach dem Erstellen der generischen Modelle wurden die Schaltpläne und das Board entworfen, welche anschließend an einen Leiterplattenhersteller geschickt wurden. Der letzte Schritt beinhaltete einen einfachen Test der Hardware, welcher zeigt, dass der Quadcopter seine Position während des Fliegens halten kann. Für uns hieß das, dass der Designprozess praktisch angewendet werden kann.

Abstract

Cyber-Physical Systems (CPSs) are systems, which not only consist of a software, or a mechanical part, but are a combination of the two parts, together with the environment in which they are used. One example of such a system are Smart Cities, which use intelligent traffic routing systems to route the cars in an optimal way to their destination, while also avoiding traffic jams. In this example, intelligent traffic lights, intelligent transportation systems, sensors on the roads, and the cars form a big CPS.

But not only such big systems are called a CPS, but a car as such is also such a system. Today's cars contain many computers and microcontrollers, which communicate with each other. For one, there is the system, which controls the motor and its components, such as the valves, the injection and exhaustion systems. And then there are the systems, which help the driver and the passengers, such as an onboard Global Positioning System (GPS), or an entertainment system. There are also systems, which help the driver in reducing accidents, by warning him of dangers, or by automatically braking the car. All these systems, together with the mechanical, electrical and electronical parts they control, form a CPS.

When creating such systems a design flow is used to help distributing the work to the different departments in an optimal way, and to find errors in the requirements as early as possible, or to avoid them altogether.

For this work we focused on the definition on a new design process, which can be used to create CPSs either from scratch, or from existing templates. This design process should enable the different departments to work in a parallel way. A small use case was then used to show that the defined design process can be used practically. This use case was the construction of a palm-sized quadcopter with some additional requirements. These requirements were that the quadcopter had to be as cheap as possible, and that it should be able to be used in future project as a research platform for formation flight.

To reach these goals the following steps were executed: At first information about existing design processes was gathered. After that we defined our model-based design process and used it in a simple use case. After defining the generic models we created the schematics and the board design files of the quadcopter, which were then sent to a manufacturer to produce. The last step included a simple test case, which shows that the quadcopter is able to steadily hold its position while being airborne. For our work this meant that the defined design process can be used in real world applications.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**
1.1 Problem Statement 1
1.2 Aim of the Work 2
1.3 Methodological Approach 3
1.4 Structure of the Work 4

2 Basic Concepts **5**
2.1 Design Processes 5
2.2 Formulary 10
2.3 Components of a Quadcopter 10

3 Requirements and Related Work **17**
3.1 Requirements 17
3.2 Current Research 18
3.3 Commercial Products and Open-Source Projects 20

4 Model-Based Design Flow **23**
4.1 Stage 1 - Preparation 26
4.2 Stage 2 - Implementation 27
4.3 Stage 3 - System Assembly 29

5 Use Case: Implementation **31**
5.1 Design Process 31
5.2 Component Selection 45

6 Critical Reflection **49**
6.1 Open Issues 50

7 Summary and Future Work **55**
7.1 Summary 55
7.2 Future Work 56

A Manuals **59**
A.1 Construction Manual 59
A.2 Programming Manual 62
A.3 Wireless Communication 64
A.4 Charging the Battery 65

B Design files **67**

List of Figures **73**

List of Tables **74**

Glossary **75**

Acronyms **77**

Bibliography **79**

CHAPTER 1

Introduction

1.1 Problem Statement

In the last few years there has been substantial progress in the field of autonomously flying quadcopters, also called drones or Unmanned Aerial Vehicles (UAVs)[1]. Much research is going in the direction of how to control these drones such that each one flies autonomously while maintaining a certain formation [BVD14][BFV09]. Some research focuses on protocols, which will enable a swarm of these drones to reach a given goal. One example of such a swarm application is distributed Simultaneous Localization And Mapping (SLAM) [CPD10][SK07]. Other examples are surveillance of an area using UAV swarms [JKG08], or wireless sensor networks [HGS+13].

Every application has a different set of requirements the drone has to fulfill, which means that either for different applications different drones are necessary, or the drones have to be easily configurable and extendable. For example, if one wants to implement a SLAM algorithm a high computing power is needed. This means that either every drone has to have a powerful microprocessor or a robust communication network to offload the computations to a stationary computer.
Irrespective of the exact application-specific requirements it is also desirable that the resulting quadcopter is as cost-effective as possible.

There exist some low-cost quadcopters, but most of these systems are not meant to be used as a research platform, because they either are only meant to be controlled manually or they lack the necessary sensors and cannot be extended (see Section 3.3). One big challenge in designing cost-effective drones is to find the right balance between size, capability of communication and computation resources.

[1] The words “quadcopter”, “quadrocopter”, “multicopter”, “drone”, ... are sometimes used synonymously in literature. In this thesis quadcopters are a certain type of multicopters, as they have 4 rotors. Multicopters are a type of drones and they all fall under the category “UAV”

For this work we focused on gathering requirements concerning a distributed quadcopter application, specifically formation flight, which we then used to design and build an extendable, low-cost quadcopter by adhering to a parallel design process.

1.2 Aim of the Work

The main result of this work is the proposal of a new design workflow for the parallel development of Cyber-Physical Systems (CPSs). To reach this goal we first had to gather information on current design processes, which was a challenging task, because most companies keep their design processes in secret. After that we adapted these processes to reduce waiting times between different departments, such as the mechanical design and the electrical or the software one. This step enables these departments to use a generic model, which is first refined for the respective department, and then used together with the general requirements as a base for the work of the departments. Using these models in the different departments then enables them to work in parallel, which should save some time, and therefore also money, in the whole design cycle.

This thesis further aims to validate the proposed design process by building a small, low-cost quadcopter, which can be used as a formation flight research platform. We want to be able to assemble this quadcopter in-house, as this reduces the dependence on external manufacturers. The drone has to be optimized for cost, which means that the sum of the costs of all required components had to be as little as possible. It also has to be as small as possible for this amount of money.

The quadcopter has to be usable in future projects, which means that it should feature sensors, which are required for typical swarm applications (for example communication modules and optionally Global Positioning System (GPS) modules). Finally we want the design of the whole project to be open, so that one will be able to enhance the quadcopter for future research. This means that if additional sensors or actors are needed, one will be able to include them in the design and build an extended version of the quadcopter just by following our proposed design approach.

1.3 Methodological Approach

The following plan was executed step-by-step to reach the goal of the thesis:

Literature Review / Requirements
At the very beginning some information about current design processes for CPSs as well as their respective requirements were gathered. For the use case, namely the model, design and construction of the quadcopter, the specific requirements for formation flight had to be investigated.

Design Process
At the beginning a new parallel design process was designed. This process had to be evaluated, which was done with the construction of the quadcopter while adhering to this process. The next steps are therefore substeps of the proposed design process.

Modeling / Design
The gathered requirements were taken as a starting point to create a model of the quadcopter.

Simulation
Using these requirements some simulations were made in suitable software, in our case Autodesk Inventor [aut]. For these simulations some constraints, e.g. a maximum size, had to be defined, and then a quadcopter, which fulfills these constraints, was simulated to test its properties. In this stage it was simple to see, which constraints would work in the real world and which would not. The outcomes of these simulations included a comparison between the maximum payload of the quadcopter versus its own size or weight, amongst others. Another outcome was a comparison between the maximum time of flight versus the weight or size.

Platform Implementation
The next step was the construction of the quadcopter from the design. This step included a list of vendors which can provide the respective parts for the specified cost. The circuit board was not built on our own, but we used a professional manufacturer for this. The result of this step was a working quadcopter, which was as small as possible for the specified costs.

Use Case
The main functional requirement of the quadcopter was hovering, because this shows that all relevant sensors can be read, the main controller works, and the motors can be controlled in software. After the future implementation of the centralized base station and the communication, one can then rely on the quadcopter to work in the expected way. In the last step a simple use case was therefore developed to show that the constructed quadcopter is indeed able to fly. The use case shows that the quadcopter is able to steadily hold its position while being airborne.

1.4 Structure of the Work

This thesis is structured as follows:

Chapter 2 introduces some basic knowledge, which is needed in the rest of this thesis. The first section shows an overview of current model-based design processes of software, mechanical, electrical and electronical systems. The second section introduces some formulas, which are used in the following chapters. In the last section the necessary and optional components of a basic quadcopter are summarized.

Chapter 3 gives an overview of available research papers and ready-to-fly quadcopters. The first section shows some research papers, which deal with the problems of designing a quadcopter from scratch. The last section gives an overview of currently available quadcopters, their features, advantages and disadvantages.

Chapter 4 shows our proposed model-based design process, which can be used to construct CPSs. First a flowchart of the overall design process is shown and the three sections then describe the respective stages of the design process.

Chapter 5 describes how the use case of the design process, namely the quadcopter, was built. The first section shows how we employed our new design process and describes the results of the different stages. The next section then describes the hardware of the created quadcopter, i.e. which components were used and why they were chosen. The last section gives an overview of the software, which controls the quadcopter to hold its position while being airborne.

Chapter 6 first compares the results of this work with other existing products. The second part of this chapter then gives an overview of unfinished parts of this work. As all requirements of this work have been fulfilled this section shows what still has to be done to use this work as a research platform for swarm applications.

Chapter 7 first gives a summary of the results of this thesis. After that some necessary future work on the proposed design flow and some future projects using our quadcopter are discussed.

Appendix A holds some manuals to better understand how to use the quadcopter. The first manual describes how the different mechanical parts have to be assembled on the Printed Circuit Board (PCB). The next one then shows how the assembled quadcopter can be programmed and which software is needed. After that the wireless capabilities of the quadcopter are introduced, and it is described how to connect the quadcopter to a PC to be able to receive debug messages or to control it. The last manual describes how to charge the battery of the quadcopter using the onboard charging controller.

Appendix B contains the 3D-model of the quadcopter, the schematics and the board design files, as well as the bills of materials.

CHAPTER 2

Basic Concepts

In the first section of this chapter we present traditional approaches to mechanical, electrical, electronical and software design processes and give a small overview on how these processes are being combined. The second section presents some formulas which are used in the next chapters. The last section then gives an overview of the necessary and optional components of a basic quadcopter.

2.1 Design Processes

2.1.1 Software Design Process

If the hardware system is finished one has to write software for it, which controls the hardware. Often some kind of controller has to be implemented if the hardware interacts with other systems or the environment. The traditional approach to writing this controller can be divided into 5 steps:

1. **Design**
   Design a block-based representation of the controller
2. **Implementation**
   Implement this design in a programming language
3. **Integration**
   Integrate the controller on hardware
4. **Testing**
   Adjust the parameters of the controller through testing
5. **Acceptance**
   If the result is not satisfactory, restart at step (1)

This so-called waterfall model was first proposed in 1970 and was reprinted in 1987. [Roy87]

As one can see this process is very time consuming and often tedious because sometimes the software can not be tested and adjusted on real hardware (think of controllers in planes). This is why more and more controllers are designed not on real hardware, but on models of the hardware and the environment. The key part of this so called Model-Based Design (MBD) approach is the model, which has to be created at the beginning. The design of the controller is then done using this model, which has to represent the physical system as close as possible (see [NM10] for more information on model-based designs for embedded systems):

1. **System identification**
   In this step an existing real world system is analyzed and a mathematical representation of the behaviour is built.
2. **Controller analysis and synthesis**
   Next the dynamic characteristics of the model are analyzed and an appropriate controller is designed.
3. **Offline simulation**
   The controller is then tested on the model of the system. No real hardware is used, which means that errors can be found early in this step.
4. **Deployment**
   If the controller behaves as expected it is implemented on real hardware.

This model-based approach only works if the hardware is finished and can be analyzed.

2.1.2 Mechanic Design Process

If the hardware also has to be built, another kind of model-based design, which works better on mechanical parts, can be used.

This mechanical approach works a bit different to the software design process (this is a short summary of the process described in [Ull10]):

1. **Model creation**
   A model of the mechanical part is created using suitable software. Very often this model is a 3D representation of a real component, which has yet to be created.
2. **Mechanical analyses**
   Stress tests and stability analyses are done on the model. This can be done on the model of one single component or on the model of the whole assembly. Often the software which was used to create the model can also be used in this step, but with limited functionality. If the model does not have the required fitness, one has to return to step (1) to change it to better perform at these tests.

Today the finished component is not produced using the final materials in the last step, but it is built using plastics. The reason for this is that the component could depend on other parts or could be part of a greater system itself. The engineers will then built a real model using plastics and do some real-world tests. One possible mistake in the design process could be that the component passes all tests, but it can not be used in the final system, because it can either not be produced, or it physically clashes with other parts.

This approach is today's standard if one has to built a new system, which consists of mechanical components. The most important reasons for this are cost and time. In the past if one had to construct a small mechanical component, they did not create a model for it, because computer-aided techniques were not widespread. A model and the mechanical stability analyses of small parts had therefore be created and done by hand, which was very tedious. That is why this was not done very often and therefore it could happen that the component did not fit in the greater system, or it broke too early. If that happened today the loss in time and money would be too much, and therefore a model representation of the whole system is often created at the beginning.

One big reason for the use of models in mechanical tests is that engineers do not have to manufacture a mechanical part if they have to perform stability tests on that part. Today's engineers can perform all kinds of mechanical tests on the digital model using appropriate software-suites.

Another reason for the creation of 3D models today is that suitable software has become widespread and easy to use. These software products very often also have plug-ins, which can perform the required mechanical analyses. Past engineers had to undergo special training if they wanted to do these analyses manually, whereas today the designers can do some preliminary tests themselves and alter the design if they find some mechanical flaws.

2.1.3 Electric / Electronic Design Process

The electric design process is a little bit different from the mechanical or software processes:

1. **Design**
   The circuit is designed on paper or on a PC. In this step components with ideal properties are assumed.

2. **Select components**
   In this step the components are replaced by real ones, which are selected to fit their specific requirements.

3. **Simulation**
   The result is simulated on a PC to see if the circuit fulfills the requirements and whether the selected components work in an acceptable way. If there is an error one has to return to step (2) or even step (1) to correct the circuit.

For more information on electronic design processes see [CH16].

The first step has not changed since the past, because there is no software where you can just input the requirements and it outputs the complete circuit. This is of course also true for the mechanical and software design processes. The second step has become easier in today's times, because there are software products which can assist in the selection of the components. The third step has changed the most, because in the past one had to derive and solve the equations of the circuit themselves. These equations are very hard to solve, because they are differential equations, which very often can not be solved analytically. Today one just has to input the circuit into the software, select the components, and the software simulates how the circuit and the components behave. Modern simulation software even shows if there are errors in the design, for example if the property of a component exceeds its limit, where in the real world it would break.

Pure electronic circuits are the simplest to create, because there is no loop in the design process:

1. **Select components**
   The requirements are analyzed and the components, which fulfill the requirements are selected.
2. **Draw schematic**
   The schematic is created using the information in the data sheets of the components.
3. **Draw board**
   The circuit board is then created from the schematic. This step determines the resulting size of the Printed Circuit Board (PCB), depending on other factors, such as the number of layers.

The simplicity of the whole design process renders the first step the most important one: The selection of the components determines if the resulting PCB fulfills the requirements or not. One wrongly selected component can only be detected by careful reviews, or only at the end when the whole board is tested, in which case the whole process has to be restarted at the first step. The second step is a very mechanical one, because all information one needs to create the schematic is written in the data sheets of the components. Very often the data sheets even contain examples of the required connections. The last step can be a very time consuming one, because if the schematic has many components the placement and the routing of the board can be very complex. There are software products, which can help in the routing or even the placement of the components, but very often the result is far from optimal and a human designer has to correct some wrong decisions of the software.

The argument of simplicity is only true on pure electronic circuits, which are very rare in real designs. It is very often the case that the electronic circuit controls some kind of electric process, in which case the complexity of the design increases.

Here is a selection of features ordered by increasing complexity of the design process:

1. **Pure electronic designs**
   Only digital signals like buttons, Light Emitting Diodes (LEDs), ...

2. **Mixed designs**
   Electronic/Electric or Electronic/Analog or other combinations

3. **Special function signals**
   Clock signals, Pulse-Width Modulation (PWM) signals, ...

4. **Serial buses**
   Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I^2C), ...

5. **Parallel buses** Peripheral Component Interconnect (PCI), Double Data Rate (DDR) interface, ...

Keep in mind that the first 4 features are pretty close in terms of design complexity[1], but there is a huge jump in complexity when designing parallel buses, especially when the bus is very wide or has a high data rate. The reason for this is that one has to keep an eye on the length of the signals to minimize the skew.

2.1.4 Combined Design Process

Please note that this section may not apply to all companies and may also be inaccurate, because most companies do not disclose their design processes to the public. The following section is a summary of some topics learned at university and some information from the internet, such as [PK05].

Long times the design process of a new product was highly serialized. At first the mechanical design was created, which dictated the overall measures of the product. After that the electrical and electronical schematics were built with respect to the mechanical design. The software was then created and tested on the real hardware at the end of the process. As one can easily see this serial workflow is highly inefficient, because every department has to wait for the other departments to finish their parts.

In recent years this design strategy was changed to a combined serial/parallel workflow, meaning that the mechanical design was either done at the beginning or at the end of the workflow. The electrical and electronical design were done in parallel and some independent parts of the software were also created at the same time. As one can see this workflow is vastly better than the completely serial process, but there is still room for improvement. For our quadcopter we therefore applied a completely parallel design workflow, which is described in detail in Chapter 4.

[1] We use the term *design complexity* as a synonym for cognitive complexity as defined by Kopetz [Kop08].

2.2 Formulary

2.2.1 Flight Time

To approximate the flight time of a quadcopter the following formula can be used.

$$t = \frac{C}{I_{tot}} \tag{2.1}$$

C is the nominal capacity of the battery, and I_{tot} is the total current, which is dominated by the current draw of the four motors.

2.2.2 Horizontal and Vertical Speed

According to [qua] the maximum horizontal speed of a quadcopter can be estimated with the formula

$$v_{hor} = \sqrt[4]{1 - \frac{1}{TR^2}} \cdot \sqrt{\frac{2mg_0}{\rho c_D A}} \cdot TR \tag{2.2}$$

where

$$TR = \frac{0.9T}{m} \tag{2.3}$$

and

$$A = \frac{1}{2} MTM^2 + 3\pi r_{prop}^2 \tag{2.4}$$

TR is the so-called *thrust ratio* and should only use 90% of the peak thrust T to allow the flight controller to keep the position of the quadcopter. MTM is the motor-to-motor distance, g_0 is the gravity of earth ($9.81\,\mathrm{m/s^2}$), ρ is the density of air ($1.2\,\mathrm{kg/m^3}$) and m is the estimated mass of the quadcopter. The calculation of the top area A uses r_{prop}, which is the radius of the propellers.

The maximum vertical speed, also known as the climb rate, can be calculated with the formula

$$v_{ver} = \sqrt{\frac{2mg_0}{\rho c_D A}} \cdot \sqrt{TR - 1} \tag{2.5}$$

2.3 Components of a Quadcopter

In this section we give an overview about which components are needed to let a quadcopter fly. Additionally some optional components which enhance the navigation abilities are also shown.

2.3.1 Microcontroller

The microcontroller is the most important digital part of a quadcopter, as it is used to read the sensors, control the actors, and optionally communicate with other systems. There are many types of microcontrollers available, from very small 8-bit ones with as little as 4 user-configurable pins, up to powerful 32-bit microprocessors with hundreds of pins.

When selecting a microcontroller for a quadcopter, one has to find the right balance between the capabilities and the power consumption of a microcontroller. It does not make sense to just use the most powerful microcontroller one can find to control such a small quadcopter, as the power consumption will be considerably higher.

It is usually best to first make a list of the required capabilities and then select a microcontroller based on that list. The first property every microcontroller has to fulfill is the number of user configurable pins. The microcontroller needs at least as many pins to interface with every sensor.

It is also an advantage if the microcontroller has some serial communication module, as this will considerably simplify the communication with the external sensors, such as the Inertial Measurement Unit (IMU).

As power consumption is a concern in small systems, such as quadcopters, one may want to use an 8-bit controller whenever possible, as they often use less power than the bigger ones. Of course, if highly complex algorithms are run on board, one may nevertheless want to use a 32-bit microcontroller to decrease the time needed to run the algorithm.

At last the personal familiarity may also play a role in the selection of a microcontroller. If two microcontrollers are up for selection, it is better to choose the one, one has more familiarity with to decrease the time needed to write and debug the software.

2.3.2 IMU

The IMU is the second most important part of a quadcopter, because it measures the attitude, and without it a quadcopter could not fly in a stable manner.

One of the most important properties for our application is the number of Degrees of Freedom (DOF). There are sensors which only have 3 DOF, and there are ones which have 9 or 10. Strictly speaking the ones with 3 DOF can't be called an IMU, because they can only measure one type of inertia, i.e. either orientation, acceleration or direction.

A 9 DOF IMU consists of a gyroscope, which measures the orientation, an accelerometer, which measures the acceleration, and a magnetometer, which measures the direction. Some types of IMUs also have a barometer integrated, which can be used to measure the height, and are then known as a 10 DOF IMU.

The reason why not just a gyroscope is used is that gyroscopes are very inaccurate and known to drift over time and temperature. An accelerometer can be used to compensate

Frequency	Range	Type of antenna
433 MHz	>1 km	wire aerial
868 MHz	>100 m	wire aerial; SMD
2.4 GHz	<50 m	SMD, PCB

Table 2.1: Properties of different RF modules

that drift, but they also become inaccurate if the sensor is tilted. The magnetometer is then used to compensate for the error when the sensor is tilted. The more advanced IMUs also have a temperature sensor integrated, which can be used to compensate for the drifts, which are caused by the changing temperature.

Most IMUs only provide the raw data of the three sensors, and leave the sensor fusion to the microcontroller, but there are more advanced types available, which have such an algorithm integrated.

2.3.3 Height sensor

A height sensor can be used to measure the height of the quadcopter while flying.

There are many ways to measure the height, but they can be grouped in two types of operation: relative sensors, such as infrared, or ultrasonic sensors, which only measure the distance to a known surface, and absolute ones, such as barometers, which can measure the absolute height in reference to the sea level.

Relative sensors work by measuring the distance to a known surface, and this brings them their great disadvantage: If the quadcopter is flying over a desk or some other unevenness of the surface, the sensor values are completely wrong.

Most of the time barometers are used, because they are independent of the composition of the surface. They work by measuring the barometric pressure of the air, which can then be converted to a height value. Barometers have two great disadvantages: they are pretty inaccurate when measuring heights in the mm range, and they are susceptible for changes in the barometric pressure, which can be caused by winds or indoors by opening or closing the door to the room.

2.3.4 Optional: Radio Frequency (RF)-module

If an RF module is used the quadcopter can send messages, such as its position and other debug outputs, to a base station, which could in turn send commands back to the quadcopter.

There are many types of small low-cost and low-power modules available, so the easiest classification is done by their frequency. Depending on the frequency of the modules they have different properties, as shown in Table 2.1.

Protocol	Frequency	Power consumption	Standard
ZigBee	433 MHz, 868 MHz, 2.4 GHz	small	IEEE 802.15.4
6LoWPAN	433 MHz, 868 MHz, 2.4 GHz	small	IEEE 802.15.4
BLE[2]	2.4 GHz	small	IEEE 802.15.1
Bluetooth	2.4 GHz	considerable	IEEE 802.15.1
Wi-Fi	2.4 GHz, 5 GHz	considerable	IEEE 802.11

Table 2.2: Some RF protocols sorted in ascending order based on their complexity

When selecting an RF module the working frequency is an important criterion, because it influences the required type of antenna. A frequency of 433 MHz may not be the best choice for small quadcopters, because it requires an external wire aerial, which could negatively impact the stability of the quadcopter. That means that for small quadcopters a frequency of 868 MHz or 2.4 GHz should be used, because they can be used with either SMD or PCB antennas. The best choice is a module in the 2.4 GHz band, because they use a PCB antenna, which is just a specially designed short copper trace on the board.

RF modules can also be categorized into the communication protocol. The most important and wide-spread protocols are shown in Table 2.2.

ZigBee is the most basic network of this list because it is very easy to implement, while it also offers advanced networking capabilities such as routing. 6LoWPAN is an advanced network for small sensors, because it uses IPv6 headers and addresses and can easily be integrated in existing networks via a router. Bluetooth features many networking capabilities and can easily be used to send messages and commands to modern smart phones or computers. BLE is part of the Bluetooth specification 4.0 and is specifically made for small low-power sensors. Wi-Fi, or Wireless LAN, is the most advanced wireless communication protocol in this list, and such a module can be used to easily integrate a system into an existing network. This complexity also brings a great disadvantage, which is the high power consumption of Wi-Fi modules.

When one wants to use a RF module in their project, they can either use a dedicated chip, which implements the protocol, or they can use an extra module. The advantage of the dedicated chip is that the application software can also be run on that chip, which means that an extra microcontroller is not needed. The disadvantage is that the antenna has to be routed oneself, which is a very complex task, which is often done by experts. People who are not well-versed in high frequency board design should therefore use dedicated modules, which already incorporate the antenna.

2.3.5 Optional: Global Positioning System (GPS) module

GPS modules can be used in bigger quadcopters to measure their absolute position while flying. They are often fused with the sensor values to reduce their inaccuracies to a

[2]Bluetooth Low Energy (BLE) is part of the Bluetooth specification 4.0, but is shown here as an extra entry, because it has several advantages for low power systems.

minimum. These modules are pretty big and most of the available modules use a ceramic antenna, which adds a considerable amount to the weight of the quadcopter.

More advanced sensors not only use the American GPS satellites, but also the Russian *GLONASS*, the Chinese *Beidou*, and the European *Galileo* system to increase their accuracy. Especially the Russian *GLONASS* system is many times more accurate than GPS for civilian uses.

Small quadcopters, such as ours, often don't use a GPS module because of their weight constraints.

2.3.6 Battery

The selection of the right battery can be a tedious process (see Section 5.1.1).

There are different types of rechargeable batteries available, such as Nickel-Metal Hydride (NiMH), Lithium-Ion (Li-ion), Lithium Polymer (LiPo) ones.

Most of the times LiPos[3] are used because they offer the highest capacity-to-weight ratio for their small size and are available in many capacities and forms. LiPos also offer the highest charge ratios of up to 10 C and discharge ratios of up to 100 C, which means that they can be discharged with up to 100 times their capacity in amperes. E.g. a 1 Ah battery with a discharge ratio of 100 C can provide continuous currents of up to 100 A until the battery is in danger of breaking down.

2.3.7 Frame

The frame is the most important mechanical part of a quadcopter and is used to hold all parts in position and to provide enough stability while also dampening the vibrations.

Bigger quadcopters use frames, which are built of iron bars or even fiber glass or carbon fiber. Smaller quadcopters often only use small plastic plates to offer additional stability, while reducing the weight as much as possible. The smallest quadcopters very often omit the frame and just use the PCB to hold the motors.

2.3.8 Motors

The selection of the right motors is very important when building a quadcopter. If the motors are too small the quadcopter won't fly and if they are too big and powerful the battery will be drained before the quadcopter could lift off.

There are many types available for bigger quadcopters, but most of the time brushless DC motors with external rotors are used. They are available in many sizes and torques, and the selection of the correct ones is the subject of many manuals on the internet.

[3] In literature the term "LiPo" is often used as a shorthand notation for "LiPo battery". In this thesis we also use this shorthand notation.

When building a small quadcopter the selection is reduced, because only standard coreless DC motors with internal rotors are available in these sizes. If the quadcopter is very small, every small motor with a diameter of about 7 mm can be used if their speed is high enough (more than 30 000 rpm and a rated current of about 1 A should be enough for palm-sized quadcopters).

2.3.9 Motor driver

The purpose of a motor driver is to provide high currents to the motors without straining the PWM outputs of the microcontroller too much.

The motor driver has to be adapted to the current rating of the motor, otherwise its efficiency is too low or it gets too hot and breaks down.

2.3.10 Propellers

The selection of the right propeller can be a very tedious process, because they have to be selected based on the size of the quadcopter, its weight, its maximum speed and most importantly the speed of the motors. There are many manuals and formulas on the internet which help in this process.

It is very important to use two clockwise propellers and two counter-clockwise ones in a diagonal fashion, because otherwise the quadcopter would spin around its axis and it won't be possible to stabilize it.

CHAPTER 3

Requirements and Related Work

The first section of this chapter shows the requirements of the proposed design process and the application requirements of the quadcopter. The second section then presents some available quadcopter projects and their relation to the requirements of our thesis.

3.1 Requirements

Before we present some related work we want to describe the requirements of our thesis. The design process had to have the following properties:

- **Allows for integrated development of physical, electrical, electronic and software departments**
  Cyber-Physical Systems (CPSs), as per their definition, consist of not only an independent electronic or software part, but they are defined as the integration and interaction of software and physical components. The design process therefore needs to be able to account for all departments.

- **Allows the design of CPSs, like quadcopters, taking application requirements into account**
  Even the best design process may be useless if it cannot be adapted for different applications. As each application has a different set of requirements and specifications the design process has to be defined in a generic way to be able to easily adapt it for each CPS.

- **Independent departments**
  The departments should be able to work with as much independence on other departments as possible.

- **Parallel workflow**
  When the work in the departments can be done independently, the departments can deploy a parallel workflow. This means that they can complete their work without having to wait on other departments.

- **Model-based design**
  The definition of a model enables the departments the formal verification of the results of their work against the model and the specification of the application. The design process should work for both, top-down and bottom-up methods for creating the model. In the top-down approach a model is created from a set of specifications and requirements, while in the bottom-up approach the model is created from an existing application.

- **Independent validation of departments**
  When a model and requirements are defined for each department, this allows independent tests to be performed on each part of the whole system. Of course, a positive test in a single department does not mean that the whole system will work as specified, but these independent tests enable the departments to find fundamental errors early in the design process.

The application requirements of the quadcopter were defined as follows:

1. Small size with a diameter of less than **150 mm**
2. Reasonable flight time of not less than **3 minutes**
3. Costs for a theoretical series production similar to that of other available quadcopters, i.e. the total costs for a series production should be less than **200 €**
4. **Open design** to allow future hardware modifications
5. **Open source** to allow future software enhancements
6. Suitable for **swarm applications**, i.e. possibility to exchange messages with a stationary base station
7. Ability of **autonomous flight** without the help of a base station

When we refer to item numbers in the next sections we refer to this definition of the application requirements.

3.2 Current Research

While there has been done some research on how to construct a low-cost quadcopter, almost none of them deal with our problem.

We will discuss some of the most important research papers on this topic in this section.

There is already a master's thesis on how to construct a low-cost quadcopter for research purposes [Bur10], but this thesis had a different goal than our thesis. Their goal was to use Commercial Off-The-Shelf (COTS) parts to construct a working quadcopter. This approach lead to a quadcopter with total costs of the components of $313.60. The size of the resulting quadcopter was also not a design criterion, which means that the diameter was about 600 mm, according to the used frame. This design therefore violates items 1 and 3 of our application requirements.

In the paper "Design and Assembling of a low-cost Mini UAV Quadcopter System" [PDC15] the authors used a commercial quadcopter, the *450 ARF* [con] and improved it by adding additional sensors. These sensors include a magnetic sensor, a Global Positioning System (GPS) receiver, a temperature and humidity sensor, and a Bluetooth module. Unfortunately the paper is only a technical report and the authors did not write about the total cost of their work.

The papers "Design considerations of a small UAV platform carrying medium payloads" [BGdRGP14] and "Multicopter UAV Design Optimization" [MHO14] propose a new way to configure UAV platforms from scratch. The authors of both papers present a tool, which can compose a UAV from different versions of the components. One just has to enter the optimization criteria, e.g. the payload or flight time, and the tools configure a quadcopter, which is ready to be built. Unfortunately the resulting configurations are far from perfect, because the controlling unit is the same in every configuration, which means that it features unnecessary parts, which may not be needed in a specific configuration. The downside, and the reason why these tools could not be used for our thesis was that they require different variations for each part of the quadcopter, i.e. different sizes and materials for the frame, different sets of batteries, motors and propellers. These tools are of use when one already has a wide range of these components available and wants to build a quadcopter which optimizes a certain property. They are also a good way to determine which components one has to use for each quadcopter when one wants to build different quadcopters with different requirements. Therefore we could not use these tools for our quadcopter, because we only have one set of requirements, and we also didn't have a large variation of different components. It would have been more work to search these components and input them into the tools than to just select the usable components right away.

Another paper, "Development of a Low Cost Quadrotor Platform for Swarm Experiments" [DNY13], shows another quadcopter with the total costs of less than $400. The diameter of this UAV is approximately 500 mm, which means that this is also quite large. The authors made a comparison of different commercial control systems and show how the resulting software was written. This project also violates the items 1 and 3 of our application requirements.

Of course, there are not only papers, which describe the mechanical aspect of quadcopters, but also how such a UAV can be controlled. For example, the papers [CTK11], [EA07],

[HGB10] and [ZWX12] describe how the whole control system works. The authors in the respective papers show how a Kalman filter can be implemented to filter the noisy signals. These filtered signals are then used by PI and PID controllers to control the motors of the drone.

3.3 Commercial Products and Open-Source Projects

There are many commercial drones available, but almost none of them could be used as a research platform, because they are not open and therefore can not be extended with additional sensors or actors. Most of these drones can only be controlled via Remote Control (RC) and are incapable of autonomous flight.

The prices of these products range from about 25 € [ama] to more than 1000 € [asc] [1]. The former product is very small, as its diameter is about 55 mm, but it can only be controlled via RC and it can not be extended in any way. The latter product is not so small, with a diameter of about 650 mm, but it is a research platform, which can host various sensors, like GPS modules, cameras, laser scanners, etc. This quadcopter can also be programmed by the user, which enables it to fly autonomously. As a summary these low-cost quadcopters violate the items 4, 5, 6 and 7 of our application requirements.

Another commercial product is the "Parrot AR Drone 2.0" [par], which costs 299 €. This drone has a diameter of about 550 mm, it can not be extended with additional sensors, but it can be controlled with a computer or a smartphone. This way one can implement the algorithm for autonomous flight on a computer and let it control the drone. One disadvantage is that the design is not open, so one can not alter the drone and build a replica to allow additional sensors. A big advantage is that the software Application Programming Interface (API) is well documented, which makes it easy to build a program which controls the drone. In fact if you search the internet with your favourite search provider you can find many open-source projects for this task. This product violates the items 1, 3 and 4 of our application requirements.

One of the most successful commercial quadcopters is the "MikroKopter", which was developed by a German company. The reason for its success is that it is an open platform, which means that everyone can download the schematics and source code and build the drone on their own. This also means that one can easily extend the quadcopter and let it fly autonomously. The smallest kit, which is currently available, costs about 680 € and has a diameter of about 480 mm [mik]. This quadcopter violates the items 1 and 3 of our application requirements.

There are also some open source projects, such as the "Paparazzi UAV"[pap], or the "LibrePilot"[lib], which are centered around tools to ease the development of own drones. Both projects introduce a software-stack, which can be used to create an autonomous flying drone. One just has to configure the stack for the used components (for example microcontroller, motor controller, sensors) and the software automatically creates a first

[1] All prices in this thesis are current as of 01 July 2016.

version of the firmware for the drone.
Both projects also introduce several ready-to-fly controller-boards, which can be used to build one's own drone.

At time of writing only one board had the right specifications to be useful for this thesis. The only downside of the Lisa/S[bitb] controller board is its prize of $300.0. Aside from that it has every component, which is useful in a swarm application: a powerful microcontroller, a built-in GPS module, built-in motor controllers and an optional RC module. The most important feature, and the reason why all other boards can not be used for this thesis, is that it can be powered by a single-cell lithium polymer battery. Because of the high cost of this board it violates item 3 of our application requirements.

The most promising project for this thesis is the "Crazyflie 2.0"[bita], which has almost every feature needed for swarm applications. The downsides are its high price of $180 for the complete quadcopter, and that it does not feature a GPS module, which is only a minor downside considering that these small drones are best used indoors. Some big advantages are that the design and the software are open, which means that one can easily adapt the schematics or the source code to meet one's needs. Please note that the design is not completely open, because the manufacturer only provides the schematics, and not the board files. So if one wants to build their own drone, they have to create the board files from scratch. We therefore decided to use the schematics of this project as a starting point to develop our own controller board, which will be cheaper and a better fit for our design. This quadcopter violates none of the items of our application requirements.

Table 3.1 gives a qualitative assessment of popular quadcopter projects. In that table the results can be in the range [−−, ∼, +, ++, +++], where −− means that the respective requirements is strongly violated, and +++ means that the respective requirement is strongly met.

Attribute	Low-cost toys	AR Drone 2.0	Crazyflie 2.0	TU Drone
Size	+++	−−	+++	+++
Open Design	−−	−−	∼	+++
Open Source	−−	+	+++	+++
Extensibility	−−	−−	++	+
Swarm applications	−−	++	++	+++
Autonomous flight	−−	+	++	++
Price	20 €	349 €	160 € [2]	135 €
Violates requirements	4, 5, 6, 7	1, 3, 4	−−	−−

Table 3.1: Comparison of quadcopters

[2] Euro-price as of 25 September 2016, given for comparison to other quadcopters. The original price was $180.

CHAPTER 4

Model-Based Design Flow

In this chapter we propose a workflow for designing Cyber-Physical Systems (CPSs), which require parts from different departments, such as the mechanical, electric, electronic and software department. One of the main focuses of this workflow is an independent workflow for each department such that they can do their work in parallel, which enables them to do their work in a very efficient way. This parallel workflow was furthermore tested on a real-world application, namely the design of a quadcopter.

A parallel design flow enables each department to start their work at the earliest time possible, and the departments are not dependent on each other. Although some extra work has to be done in the beginning, such as the generation of a generic model and the definition of test cases, this approach should nevertheless be more efficient than the traditional design flows where each department had to wait until the previous ones had finished their work.

As can be seen in Figures 4.1 and 4.2, our design flow can be divided into three stages, which have to be done one after the other. Nevertheless, the most time-consuming work, namely the implementation in the different departments can be done in parallel. This can be seen in our figure in the second stage.

Stage 1 consists of the preparatory work, such as the definition of requirements, the creation of generic models and the extraction of test cases. This work cannot be done in parallel to the other stages as all further stages rely on the results of these steps.

Stage 2 is the implementation stage, where the generic model is first turned into a specific one for each department. This can be seen in Figure 4.1 in the first four parallel steps. After that the models, which now consist of real components, are further refined with the help of the extracted test cases, as is shown in Figure 4.2. The last steps in this stage consist of a check to see if all requirements can be met by the combined results of the departments.

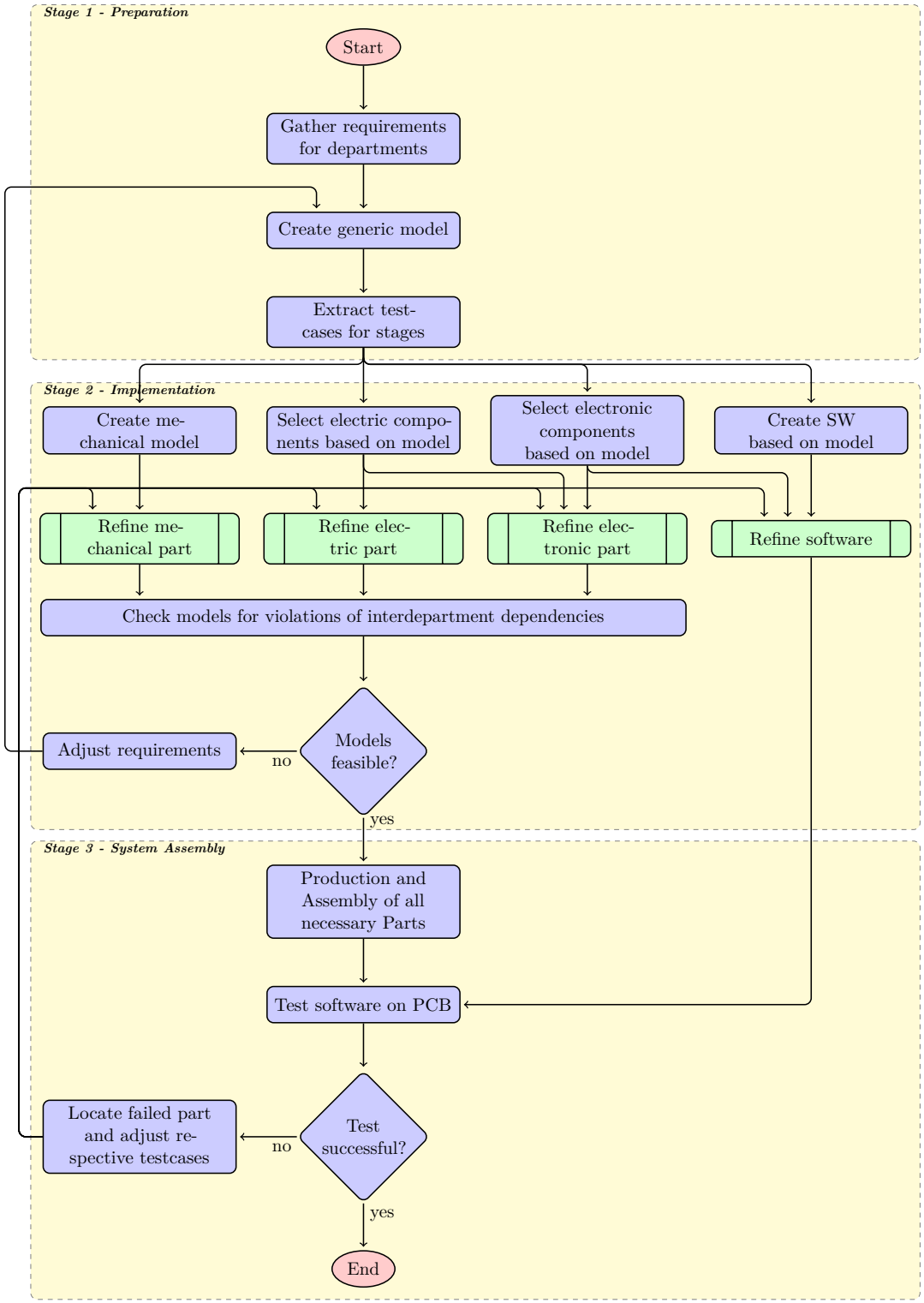


Figure 4.1: Flowchart of the design flow

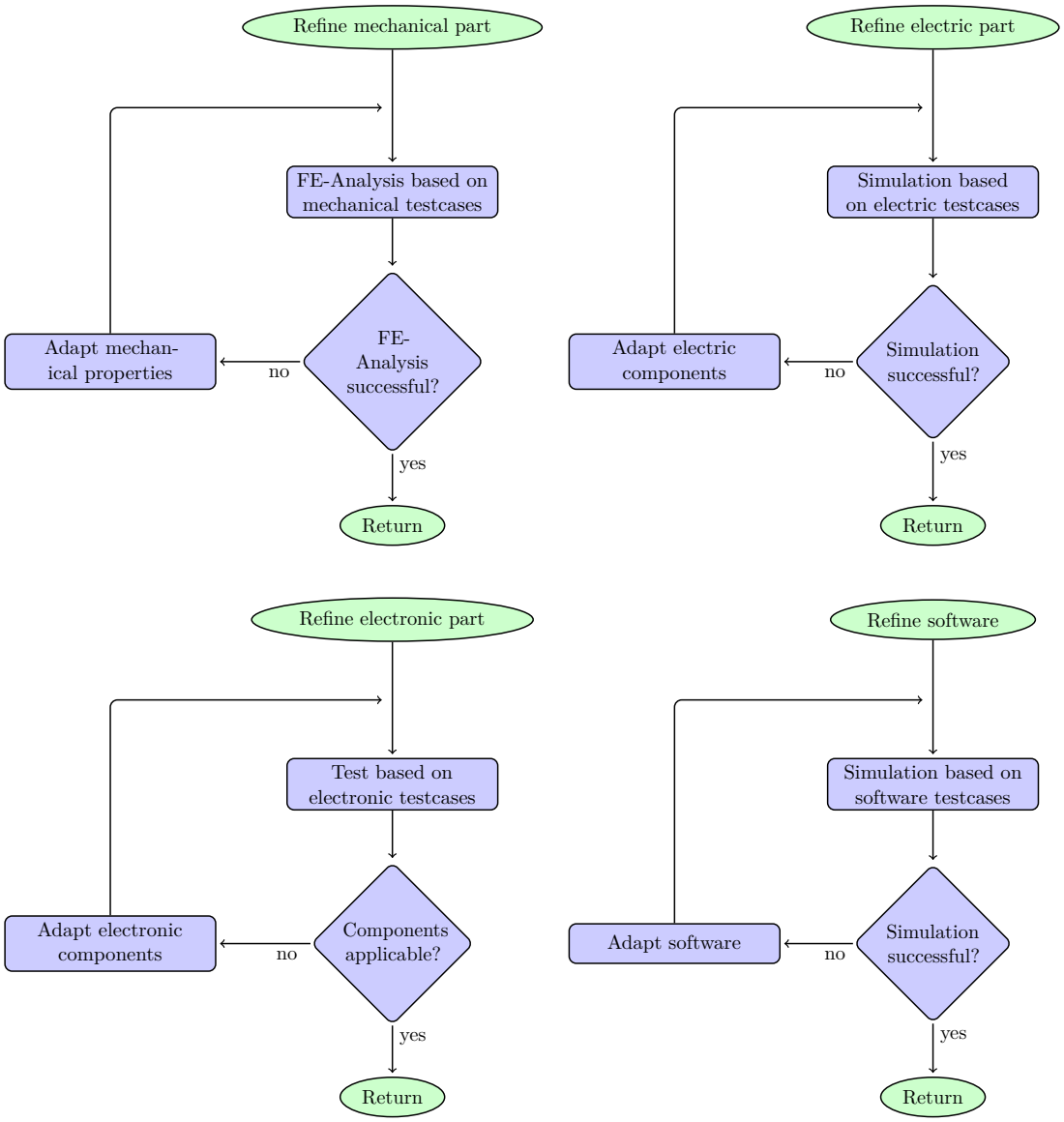


Figure 4.2: Flowchart of the design flow (cont.)

Stage 3 is the last stage, as this is where the results of the previous steps are gathered and the whole system is assembled and finally tested. At first all required mechanical, electric and electronic parts are produced and assembled. After that the whole system is tested using the software written in the previous stage. When all tests were successfully completed, this marks the end of the design flow and the system is ready for operation.

The next sections describe each step of the design flow in detail.

4.1 Stage 1 - Preparation

The very first step is to gather the requirements and to create the specification sheet. This procedure is of course the same regardless of which type of design flow is employed. The specifications have to be complete and accurate as this is the base of the subsequent work. An error in this step may render the final product useless as it either does not work or is not what the customer wanted. For detailed explanations of the required parts of a specification sheet one can refer to the norms [VDIb], [VDIa] and [IEE98].

The next step is the creation of a generic model, which acts as the base of the work for each department. One can think of this model as a summary and mathematical design of the specification sheet. The generic model consists of an abstract description of the CPS under design and its relevant environment. For the creation of this model, a top-down, a bottom-up or a mixed approach can be employed. In the top-down approach the generic model is created from the specifications of the application. This model will later help to check the results of the departments against the specifications. The bottom-up approach on the other side takes a finished application and creates a model based on the physical implementation. This model can then be checked against the requirements. The mixed approach can be used when a similar physical implementation, which fulfills some of the specifications, is available. In this case a model is created from the available implementation and then altered for the new specifications. Irrespective of the used approach the generic model will be refined by each department in the later steps of the design flow.

The generic model is then used to extract test cases for each department to test their solutions upon. E.g. for a particular product the outer dimensions may be extremely important. This means that one test case for the mechanical department may be that the product does not exceed some maximal size. As one can see, together with the generic model these test cases are what every department needs to begin its work.

Please note that formal methods are beneficial to describe the requirements and the subsequent generic model, as this enables one to compare the specific models in each department against the generic one and test that they are compatible among each other. These formal methods are not described in this thesis and are subject to future work (see Chapter 7).

4.2 Stage 2 - Implementation

This is where the design flow splits into the different departments and everything is done in parallel without having to wait for the other departments to finish.

The main work of the mechanical department is to create a mechanical design of the product. Depending on the particular product this can include a design of a case, or the design of some other mechanical parts. Every mechanical part which is important for the product to work is designed here. The particular design flow in this department may vary depending on the product. It may be important to start with a 3D model of a mechanical part, followed by some numerical simulations, for example a Finite Element-Analysis (FE-Analysis), or the department may decide to skip this step and just create a drawing of the part without any model or analysis.

After the mechanical model is created it is verified against the generic model and the extracted test cases. In Figure 4.2 this is shown exemplary with an FE-Analysis of the mechanical model. The requirements for this department may include that the mechanical part does not break or has the required safety margins, which can both be tested this way.

The main focus of the electric department is to design and test the electric parts of the product. This process can include the design of an electric schematic and the selection of the required components, such as resistors, capacitors, inductors and other parts. Depending on the particular product the workload increases or decreases. E.g. a quadcopter requires very few purely electric components, as most of the resistors or capacitors belong to the electronic design. This is in contrast to the workload of a big industrial system, where every motor has to be specifically selected.

The figure shows that the validation of the electric design can be done with a simulation. Of course the main requirement for the electric design is that the final product does what it is intended to do. Other requirements may include the maximum power consumption, or a minimum power output.

The electronic department deals with the design and validation of the digital parts. This includes the selection of microcontrollers, drivers and other digital chips, and their necessary electric parts. For example, most interface drivers which translate between different voltages need external capacitors for their charge pumps. Other extremely important parts are the decoupling capacitors, which should be included such that every chip has at least one on each power input, and which have to be placed as near as possible to the respective chip.

The electronic design can be tested and verified in parts with a simulation, but most of the validation work still has to be done by humans. This is because the whole digital design is very complex, and depends on the information in data sheets, which can not be validated by simulations. E.g. according to the data sheet of a digital chip a resistor with a particular value has to be placed between two pins. If the value of the resistor

depends on the whole design, or if it is used to select a particular operation mode of the chip, this design is impossible to verify with a computer program.

In the figure you can see an arrow going from the electric design to the electronic refinement. This expresses that often the electric schematic influences parts of the electronic design. For example, the selection of a particular high current driver depends on the electric circuit it is supposed to drive.

The fourth department is responsible for software development. This step includes not only the development, but also the tests, such as validation and simulation of the software. At first a software model of the environment has to be created to be able to simulate the software. This can be done using the generic model and the specification sheet. This procedure is called Software in the Loop (SIL) and has to be used because at the beginning of the process no information about the physical hardware is available. This does not mean that software development is useless at this stage, because even if the low-level drivers can not be created, the main algorithms can still be implemented. Instead of real low-level drivers, generic ones are used. This has the advantage that simulation, testing and debugging becomes easier, because these generic drivers can be adapted to print out as much information about the program as needed.

After the electronic department has fixed the hardware components, the software can be adapted to run on the hardware. In the figure this is shown with the arrow going from the selection of electronic components to the refinement of the software. This so called Hardware in the Loop (HIL) process is then better suited to simulate the software in real world environments. In this step the generic drivers are removed and replaced by real low-level drivers, which are adapted to the actual hardware. This process makes debugging and testing easier because in this stage the developers can be sure that the high-level algorithms work, because they have already been tested. That means that locating a potential error is easier because the developers know that the error can only be in the low-level drivers.

When the mechanical, electric and electronic designs are finished the last step of this stage can be performed, which is the check for inter-department design failures. In this step all models from the departments are verified against each other to make sure that they are compatible. Some requirements may be fulfilled by each department on their own, but when the models are brought together, they may not be realizable. One easy to understand example is the following: The application requirements for a new quadcopter state that it should weigh at most 30 g and have a flight time of at least 5 hours. These requirements are split for the different departments, which means that the mechanical department gets the requirement for the weight, among others, and the electric department has to fulfill the requirement for the flight time. On their own, both requirements are realizable, but when the results from both departments are brought together, they will contradict each other. I.e. the main problem is that the battery will be far too heavy to be able to fulfill the flight time. Currently there are no batteries commercially available, which have enough capacity for the required flight time, and still weigh less than the required 30 g.

At the end of this second stage the software, the electronic designs, some additional electric designs, and the design of the mechanical parts are finished and ready to be manufactured, which is done in the next stage.

4.3 Stage 3 - System Assembly

The first step of this stage is to manufacture all parts which were designed in the previous stage. This includes the production of all mechanical parts, all electrical designs, all Printed Circuit Boards (PCBs) and the subsequent assembly into the final product. Please note that the production of the parts can of course be done in parallel. This is not shown in the figure because it would have made the whole picture more complex.

When the product is finished the main software test is performed. As the software has already been tested and simulated numerous times before that test, the expectation is that this last test has to be successful. Reality is of course different, because very often this test is not 100 percent successful and small errors are still found. Sometimes these errors stem from inaccuracies in the mechanical parts, and sometimes there are errors in the electrical or electronical design. Software is also not perfect and some bugs may still linger. Minor errors may stem from inaccurate models of the environment such that the designed controllers are not adjusted correctly. Sometimes the controllers are not set at all and have to be optimized in the real world environment.

So if there are still some errors, the next step is to locate the relevant department and refine the relevant part. Sometimes only the relevant department has to do some adjustments, but it can also be the case that the modification is not so minor and other departments have to be involved to adjust their respective parts. Of course, this second case should not happen frequently, because these modifications, which involve more departments are very time consuming. If that occurs frequently in reality, more emphasis has to be put on the work in the first stage such that these errors are caught earlier in the design process where they can be fixed more easily.

At the end when all tests are completed successfully the product is finished and ready for series production.

CHAPTER 5

Use Case: Implementation

In this chapter we will first show you how we employed the design process introduced in the previous chapter. The next section then describes which components were selected for our quadcopter. The last part of that section then discusses the total costs of the production of our quadcopter.

5.1 Design Process

Please bear in mind that the quadcopter was built by a single person, namely the author of this thesis. It is therefore hard to speak of a parallel workflow, but the author nevertheless employed the underlying process described before.

5.1.1 Generic Model

At first some theoretical calculations were done to be able to approximate the resulting sizes, weight, flight time and maximum speed based on a few parameters.

The approximation of the resulting size was done the following way: At first some existing quadcopters (most prominently the "Crazyflie 2.0"[bita]) were taken as a base to study their properties and the relationships between them. For example, there is a complex relationship between the size of a quadcopter, the size of the battery, the size and thrust of the propellers, the weight of additional loads, and the flight time. The following relationships are the most obvious (please note that for the following items it holds that only the mentioned part of the quadcopter is taken into account, e.g. in the first item only the physical size of the quadcopter is changed, but not the battery):

- The larger a quadcopter is, the heavier it is, and the lower its flight time is.
- The larger the battery is, the higher its weight is, and the lower the flight time is.

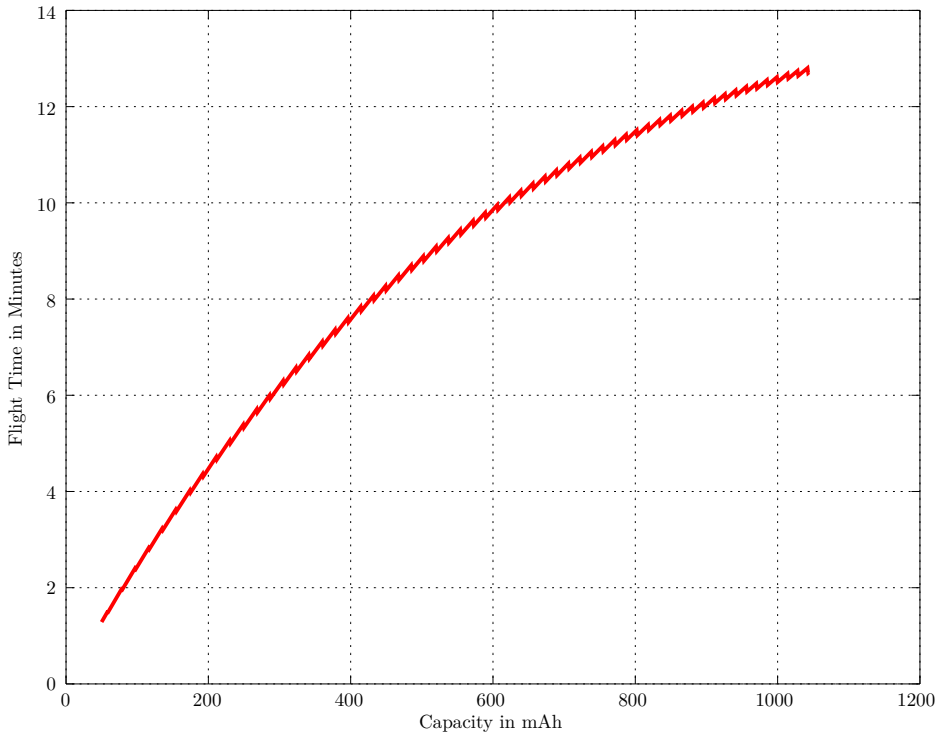


Figure 5.1: Plot of the flight time over the capacity of the battery

- But also, the larger the battery is, the higher its capacity is, and the higher the flight time is.
- All in all, the curve of the capacity rises faster than the weight of a battery, so the net outcome of a larger battery is, that the quadcopter can fly a longer time (see Figure 5.1).
- The larger the propellers are, the larger the thrust is, and the higher the speed of the quadcopter is.
- The larger the propellers are, the larger the motors need to be, the higher the weight is, and therefore the lower the flight time is.
- The higher the payload is, the higher the weight is, and the lower the flight time is (see Figure 5.2).

As one can easily see, because of these complex relationships it is not possible to design a quadcopter from scratch in one go, but one has to employ an iterative design process, such as the one in Figure 5.3, which we used to create a generic model.

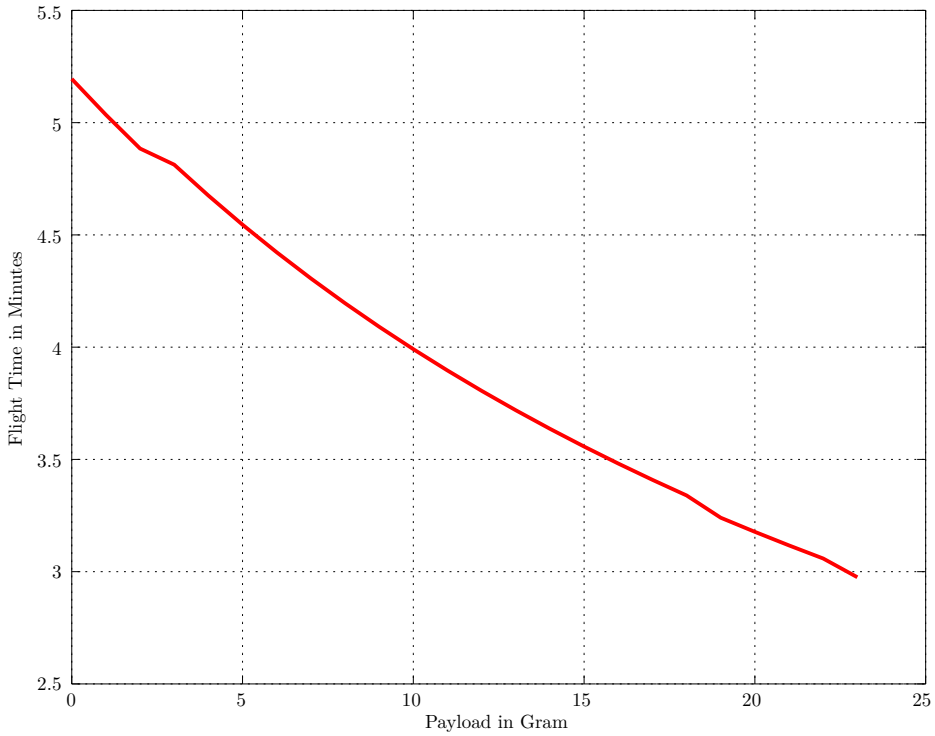


Figure 5.2: Plot of the flight time over the weight of the payload

At first the size of the quadcopter was defined and the motors, propellers, frame and the battery were selected to fit the size. After that the flight time was calculated and checked if it was high enough. If that was not the case a better battery, which has the same size but a higher capacity, was sought. If such a battery was found, the resulting flight time was checked again. If no such battery was available, the resulting size of the quadcopter was increased and the process began at the beginning. The end of the process resulted in the selection of the motors and propellers, which can be seen in Section 5.2.

To calculate the flight time the formula[1] 2.1 was used. Using a capacity of 240 mAh and a total maximum current of approximately 4.44 A (see [bitc]) we calculated a flight time of 3.2 min when all motors are running at full power.

As the approximate weight of the whole quadcopter was estimated to be about 35 g, we could also calculate the flight time when the quadcopter is hovering. Figure 5.4 shows the thrust characteristic of a similar quadcopter, the “Crazyflie 2.0” (see [bitc]). According to that figure the quadcopter draws about 2.8 A ($= 4 \cdot 0.7$A) when the quadcopter is

[1] All formulas are described in Chapter 2.2.

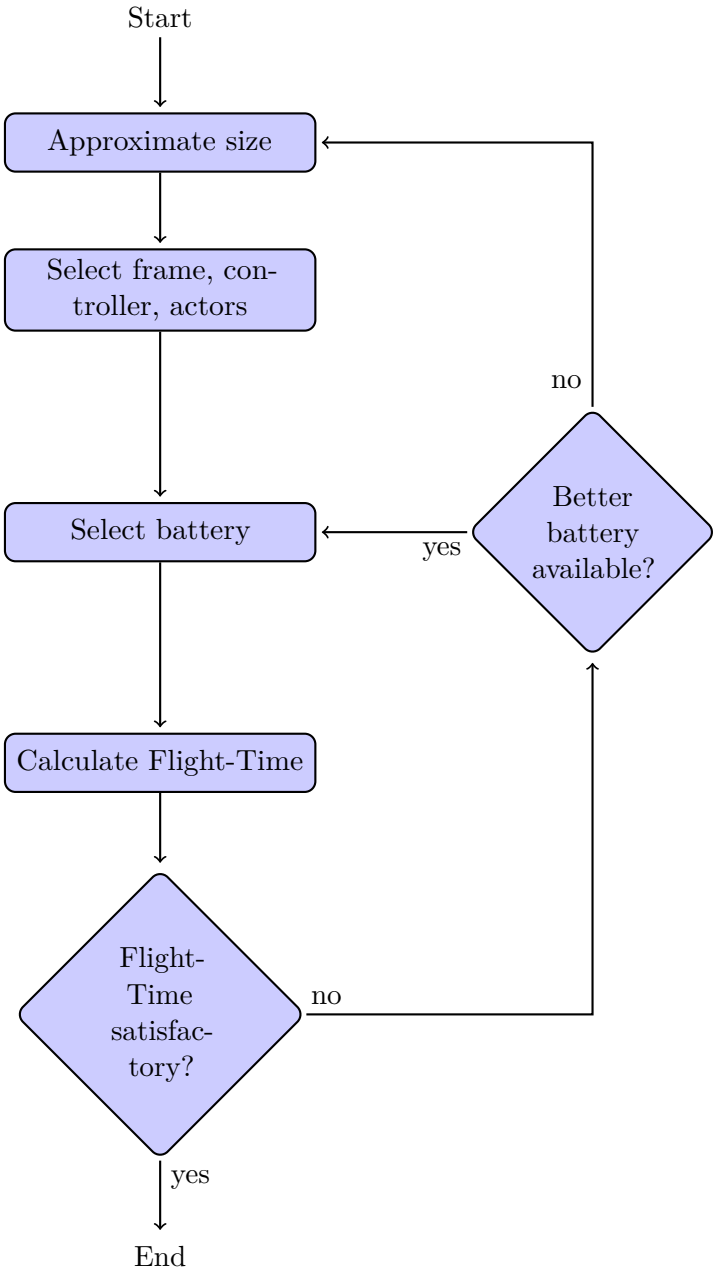


Figure 5.3: Flowchart of the process to create a generic model

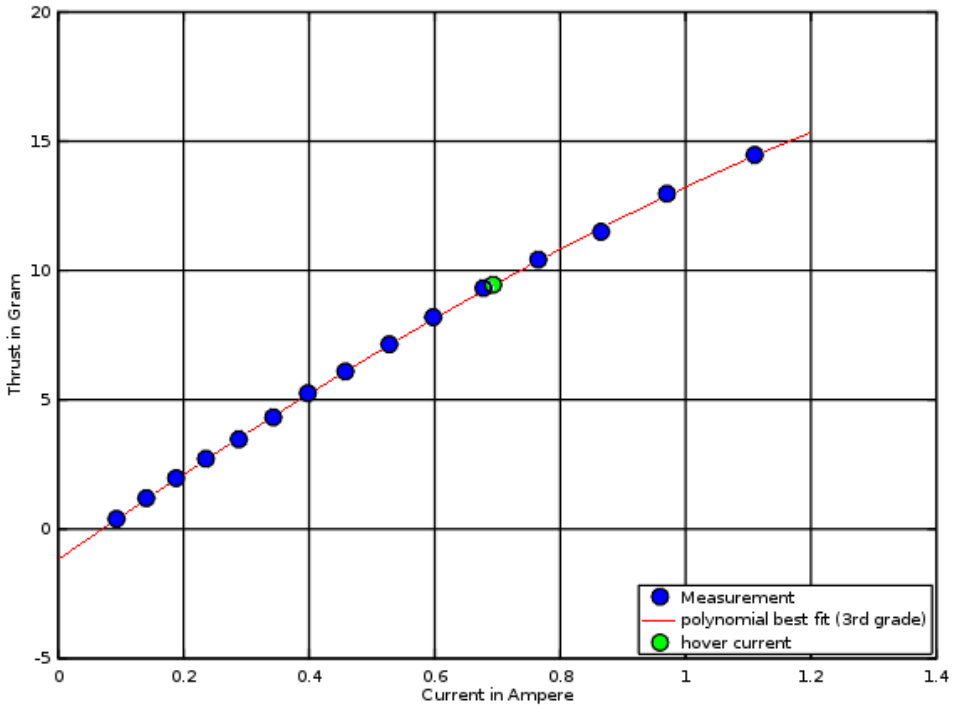


Figure 5.4: Plot of the thrust characteristic

hovering. Again using Formula 2.1 the flight time in this case is about 5.2 min, which can also be seen as the zero-payload case in Figure 5.2.

To get an estimate on how fast such a small quadcopter will be able to fly we used the Formulas 2.2 and 2.5.

Inserting the approximated values for our quadcopter, which are a peak thrust T of 57.9 g, a motor-to-motor distance MTM of 94 mm, and a radius of the propellers r_{prop} of 45 mm, gives a maximum horizontal speed of approximately 9 m/s and a maximum vertical speed of 5 m/s.

At the end of the preparatory work the generic model consisted of a rough sketch of the outline of the quadcopter, its resulting size and the selection of the most important components. This model was then used in the four departments to refine the sketch and to create the resulting hardware.

Here is a short summary of the most important results of this stage:

- Diameter similar to the Crazyflie [bita], in our case 100 ± 10 mm.
- Total weight of about 40 ± 10 g

- Motors, propellers and the battery are the same as on the Crazyflie [bita].
- Flight time of more than 3 min.

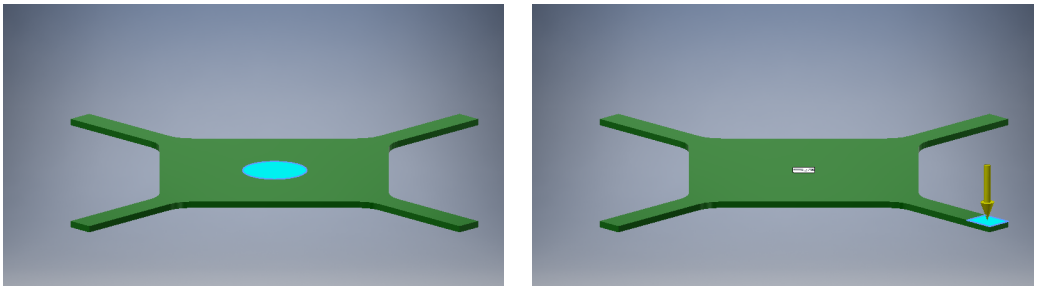
5.1.2 Mechanical Design

The main work in the mechanical department was to derive a concept of the mechanical structure of the quadcopter. This concept was then realized in a 3D model, which was used to simulate the mechanical stability of the system. The most important outcomes at this stage were the measurements of the Printed Circuit Board (PCB), including the thickness, which have a great influence on the stability, and the design of the motor mounts. After the 3D model was created a Finite Element-Analysis (FE-Analysis) was performed to calculate the maximum deformation of the board when the quadcopter is accelerating.

The 3D model, which can be seen in Appendix B.1, shows how the quadcopter will look like when it is assembled, including the main sizes, which are the diameter of 102.8 mm, and the height of 20.4 mm. The outcome of the FE-Analysis was that a thickness of 1 mm for the PCB should be enough to provide enough robustness for all reasonable use cases. This thickness also provides enough safety margins for the quadcopter to survive some crashes, which are inevitable when the software controllers are not optimized.

We will now briefly explain how the FE-Analysis was performed. For the design of the 3D model and the FE-Analysis **Autodesk Inventor** [aut] and its included tools were used. At first the reference area was defined to be at the bottom of the main body of the board (Figure 5.5a). This is the case because prior considerations showed that the arms would be the weak points and would be the first parts to break if the dimensions of the board were wrong. That means that the FE-Analysis can be reduced to only set the focus on them. This was then done by setting a load of 0.57 N onto the end of one arm (Figure 5.5b). This load was calculated using the data of the motor and propellers [bitc], which show that at full speed the thrust of one arm would be about 14.5 g. Then using the formula $F = m \cdot a$ and using the mean gravitation of 9.81 $\frac{\mathrm{m}}{\mathrm{s}^2}$ we got a force of 0.14 N. In the analysis we then used four times the maximum thrust, which should give enough margin to also include minor crashes. As one can easily see, the board is symmetrical, which means that only one arm has to be tested, because the results have to be the same for the others as well.

The next step was then to let the program simulate this use case, and to verify if the results were plausible. After some iterations a thickness of 1 mm, together with a width of the arms of 4 mm, were found to offer enough safety margins for our application. The length of the arms were fixed beforehand, because their minimum length depends on the radius of the propellers. The maximum length is not restricted, but one should leave them as short as possible to avoid instabilities because of vibrations or oscillations. In our case, as we used the same propellers as the Crazyflie [bita], which have a diameter of 45 mm, we chose a length of 20 mm for the arms. Figure 5.6 shows that the maximum mechanical stress would be 12.55 MPa and located on the transition from the arm to the



(a) Reference area (b) Simulation load

Figure 5.5: Constraints of the FE-Analysis

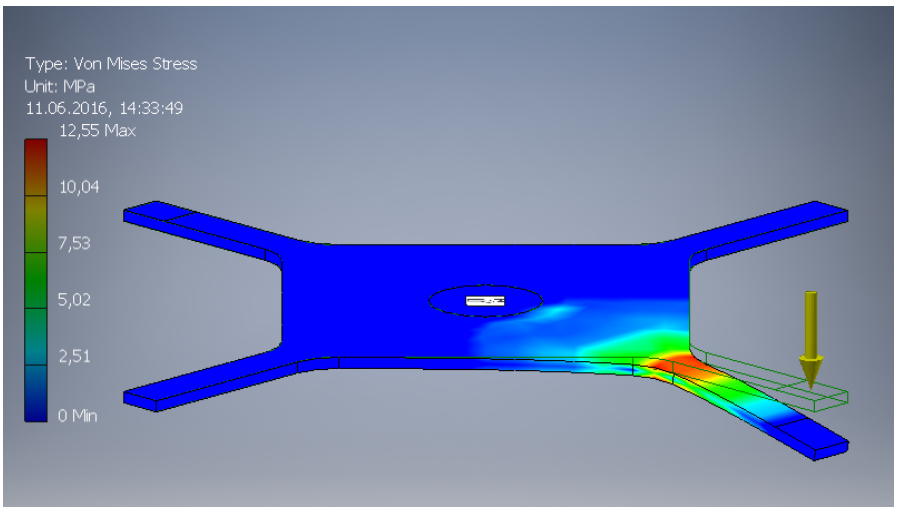


Figure 5.6: Maximum stress

main body. This is plausible, because that is the location where the arm would break in case of a heavy crash. Figure 5.7 shows that the deformation of the arm under the applied load would only be about 0.4 mm and its main location would be at the end of the arm. Figure 5.8 shows that the minimum safety factor would be about 9.2 in this situation. A safety margin of 9.2 means that the board can be burdened with 9.2 times the load of this simulation until it breaks.

At the end of the mechanical design process the model of the motor mounts was converted to a format for a 3D printer[2] to be able to print them. Figure 5.9 shows a comparison between the 3D model of the motor mounts and the printed ones. As one can easily see the differences between them are negligible, which means that 3D models on the computer are a good way to simulate mechanical properties of real parts.

[2]We used an **Ultimaker 2** [ult] with default settings to print our parts.

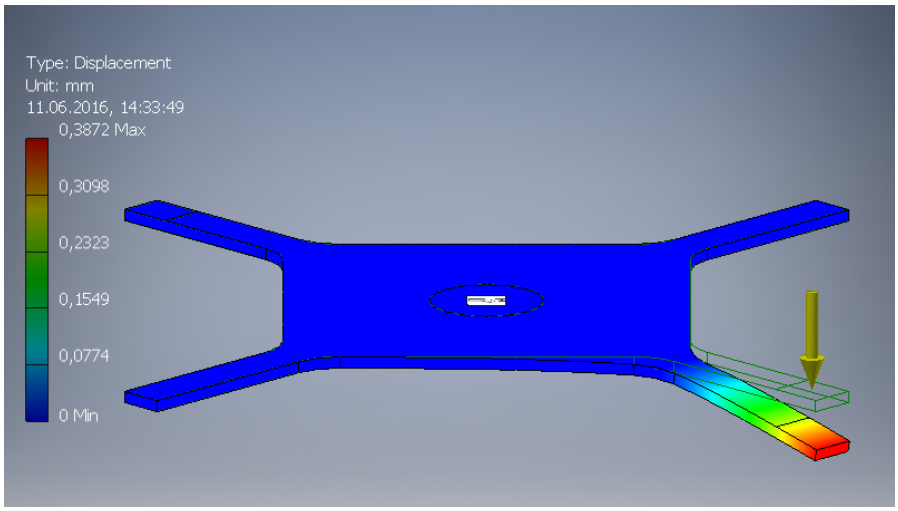


Figure 5.7: Maximum deformation

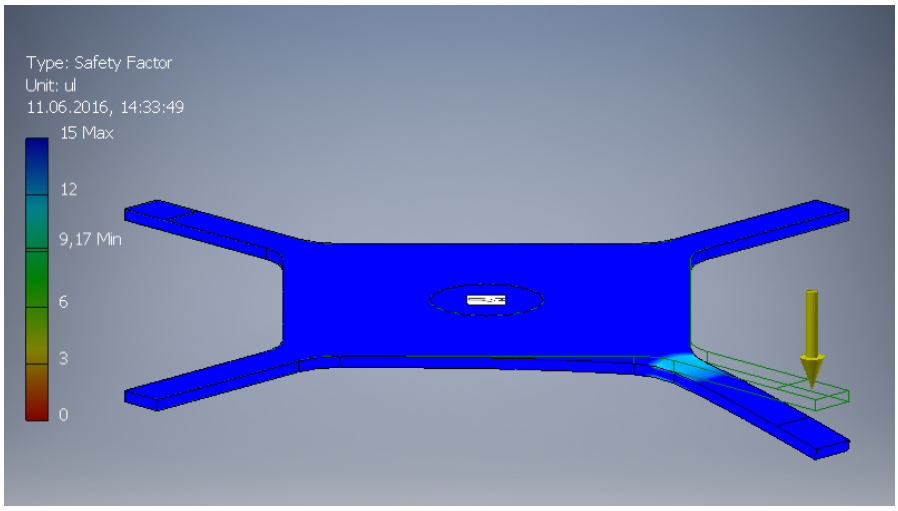
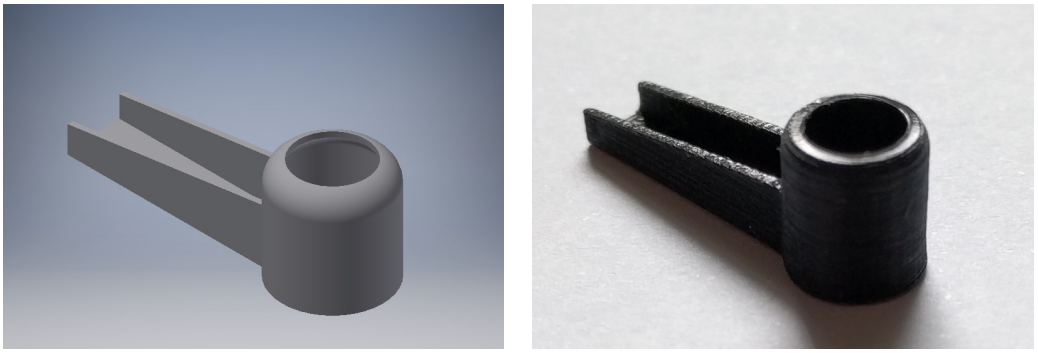


Figure 5.8: Minimum safety factor



(a) Model (b) Real

Figure 5.9: Comparison between the model and the real part

5.1.3 Electric Design

The electric design consisted of the design of the motor driver, and the design and simulation of the passive filter, which is used to filter the Pulse-Width Modulation (PWM) noise from the power supply of the sensors.

The motor driver, whose circuit can be seen in Figure 5.10, is needed because the motors draw currents of up to 1 A, but the microcontroller can not supply these currents on its pins. Here is an explanation on how one of the four drivers works (see the leftmost part of Figure 5.10): **P1** is the mechanical connector where the motor is connected. Its first connector holds the battery voltage and the other connector goes to the MOSFET. **D1** is a so-called *flyback diode* and filters the high voltages, which the motor generates when the PWM has a transition from the nominal voltage back to 0. This diode protects the rest of the circuit from these inductive voltages. The MOSFET **T1** is the main driver and can supply continuous currents of up to 5.4 A. It gets driven by the microcontroller, which is connected to the gate via the resistor **R7**. This resistor is needed to protect the microcontroller from high currents when the gate is driven with a PWM. The resistor **R8** is used to pull the gate to ground when the microcontroller is not configured and its pins are set as inputs.

To protect the sensors from the PWM noise of the motors on the power supply a passive lowpass filter was implemented (see Figure 5.11). At first the cutoff frequency was calculated using the standard formula, and then the circuit was simulated in **Linear Technology LTSpice** [lin] to better see the frequency response. According to the standard formula for LC-filters, $f_g = \frac{1}{2\pi\sqrt{L \cdot C}}$, the cutoff frequency is at 8.9 kHz. The simulation, which can be seen in Figure 5.12, shows that the filter is useful even at low PWM frequencies of about 20 kHz.

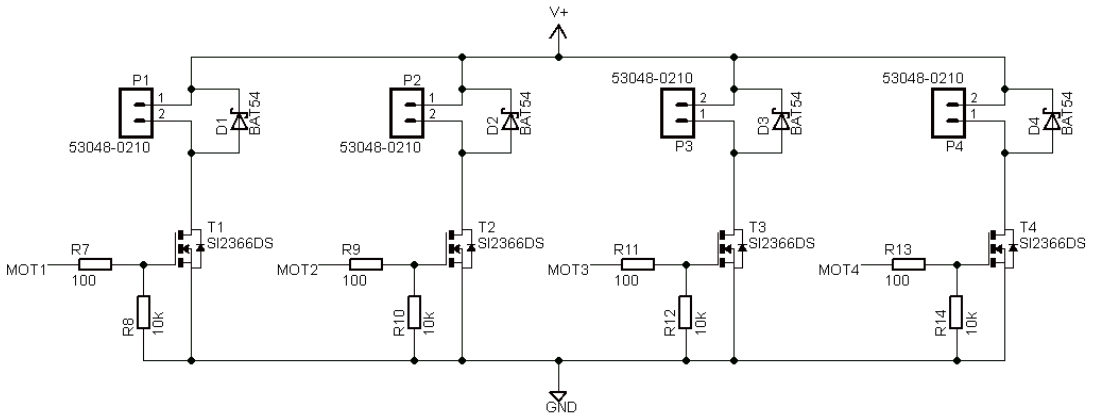


Figure 5.10: The four motor drivers

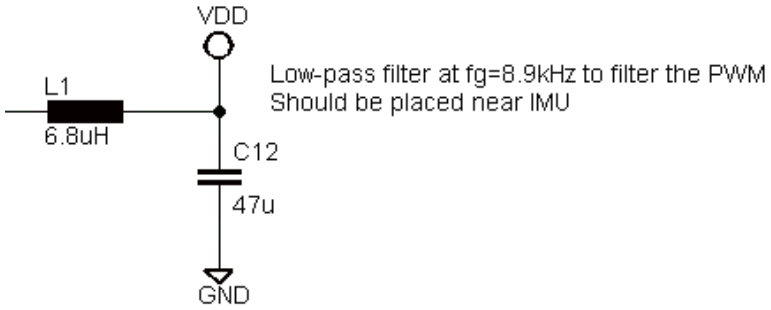


Figure 5.11: Passive lowpass filter

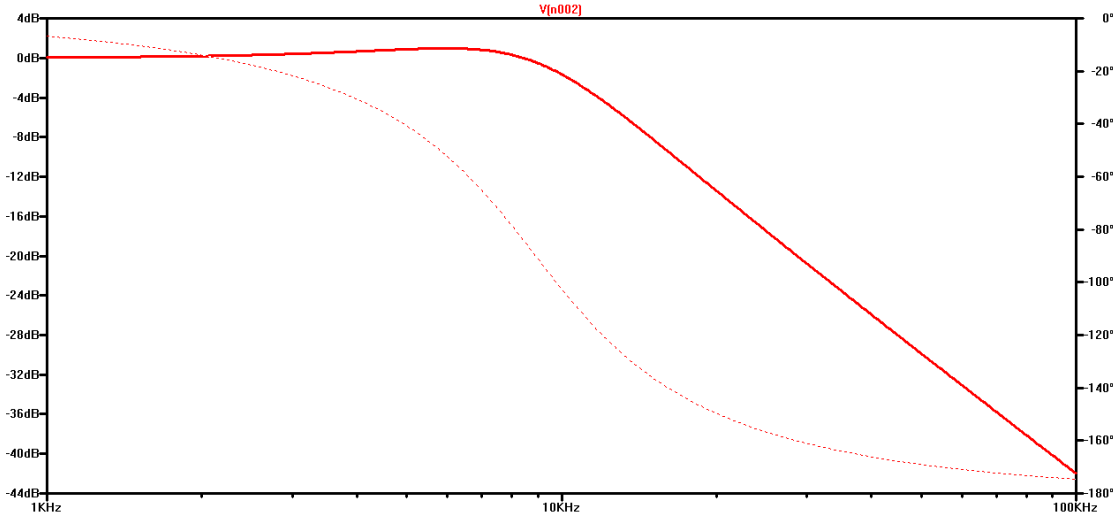


Figure 5.12: Simulation of the lowpass filter

5.1.4 Electronic Design

The electronic design of the quadcopter was a straightforward process. At first some research was performed to determine which components would be needed, whereupon the components were selected with the help of the online catalog of a distributor.

The results of this step was the selection of the following components:

- a **microcontroller** (*Atmel ATxmega32C4*), which controls the other sensors and actors
- an **Inertial Measurement Unit (IMU)** (*ST LSM9DS0*), which measures the attitude of the quadcopter
- a **barometer** (*ST LPS25H*), which can be used to determine the height of the quadcopter
- a **Radio Frequency (RF) module** (*Microchip MRF24J40MA*), which exchanges messages with other quadcopters or a computer
- some **passive elements**, such as a voltage regulator, resistors, inductors, capacitors, Light Emitting Diodes (LEDs), buttons, connectors
- the **motor driver**, which was developed in the electric department
- a **Li-Ion charger** (*TI BQ24075*), which is nice to have to be able to charge the battery without an external charger

After these components were chosen the creation of the schematic was a straightforward process, because most of the passive elements were described in the datasheets of the components. E.g. the datasheet of the IMU says that a 220 nF capacitor has to be placed between its *SETC* and *SETP* pins. For the creation of the schematic and the board design the free version of **CadSoft EAGLE** [cad] was used.

The passive parts, which were not mentioned in the datasheets were inferred from the following common electronic design rules:

- A capacitor should be placed near each power supply pin to filter the noise from the voltage. Normally a 100 nF capacitor is used.
- Every LED needs a resistor to limit the current flowing through it. Its value has to be calculated such that the current does not exceed the forward current from the datasheet of the LED.
- A flyback diode should be placed between the connectors of each inductive load to protect the rest of the system from inductive voltages.

- The ground should be the reference of the power supply, therefore it should be realized as a ground plane.

After the schematic, which can be seen in Appendix B.2, was created the board files were designed (Appendix B.3). Most of this work, such as the placement and routing, was done manually because this approach left us with the most freedom for the constrained size and form of the board.

5.1.5 Software Design

At the start of the software design a flowchart of the initialization routine and the main software was created. This was done to reflect the refinement of the general model and the creation of hardware independent routines of the general design process described in Chapter 4.

The resulting flowcharts can be seen in Figure 5.13 and 5.14.

After the hardware was finalized these flowcharts were realized in software. For this part **Atmel Studio** [atm] was used because it offers supreme debugging capabilities of its microcontrollers.

After power up the software initializes the microcontroller, such as the clock system, sleep modes, timers, and configures its pins. Directly afterwards the external sensors are set up. This includes the initialization and configuration of the IMU, barometer and the RF module. In the next step the charging controller for the battery is checked to see if the battery is currently being recharged. If this is the case the software enters an error state, which it only exits when the microcontroller is reset by powering it off and on again. This is an optional feature to make sure the quadcopter does not start flying when the USB cable is still connected. The next part of the initialization process is a loop, which is only exited when the start condition is detected. This start condition can be triggered either by a press of the onboard button or by sending the letter "g" via RF to the microcontroller. When the battery is not being recharged and the start condition has been detected the microcontroller waits for some time to make sure the user has removed the hands from the button. After that the internal timer is activated and the program enters its main loop.

The main loop sleeps until the internal timer has fired and a time of 5 ms has elapsed. When that happens the new attitude values from the IMU are refreshed and fed into a sensor fusion algorithm, namely the **Madgwick Quaternion Update** [MHV11], which fuses the values from the accelerometer, gyroscope and magnetometer into quaternions, which are then converted into the attitude values roll, pitch and yaw [3]. When the new attitude has been calculated it is fed into a Proportional-Integral-Derivative (PID) controller, which then calculates the PWM values for the four motors. At the end the program enters the sleep mode again, which is interrupted by the next timer interrupt.

[3] The author of the paper also provides an open source implementation of his algorithm, which can be downloaded at [mad]. This source code served as the base for our implementation.

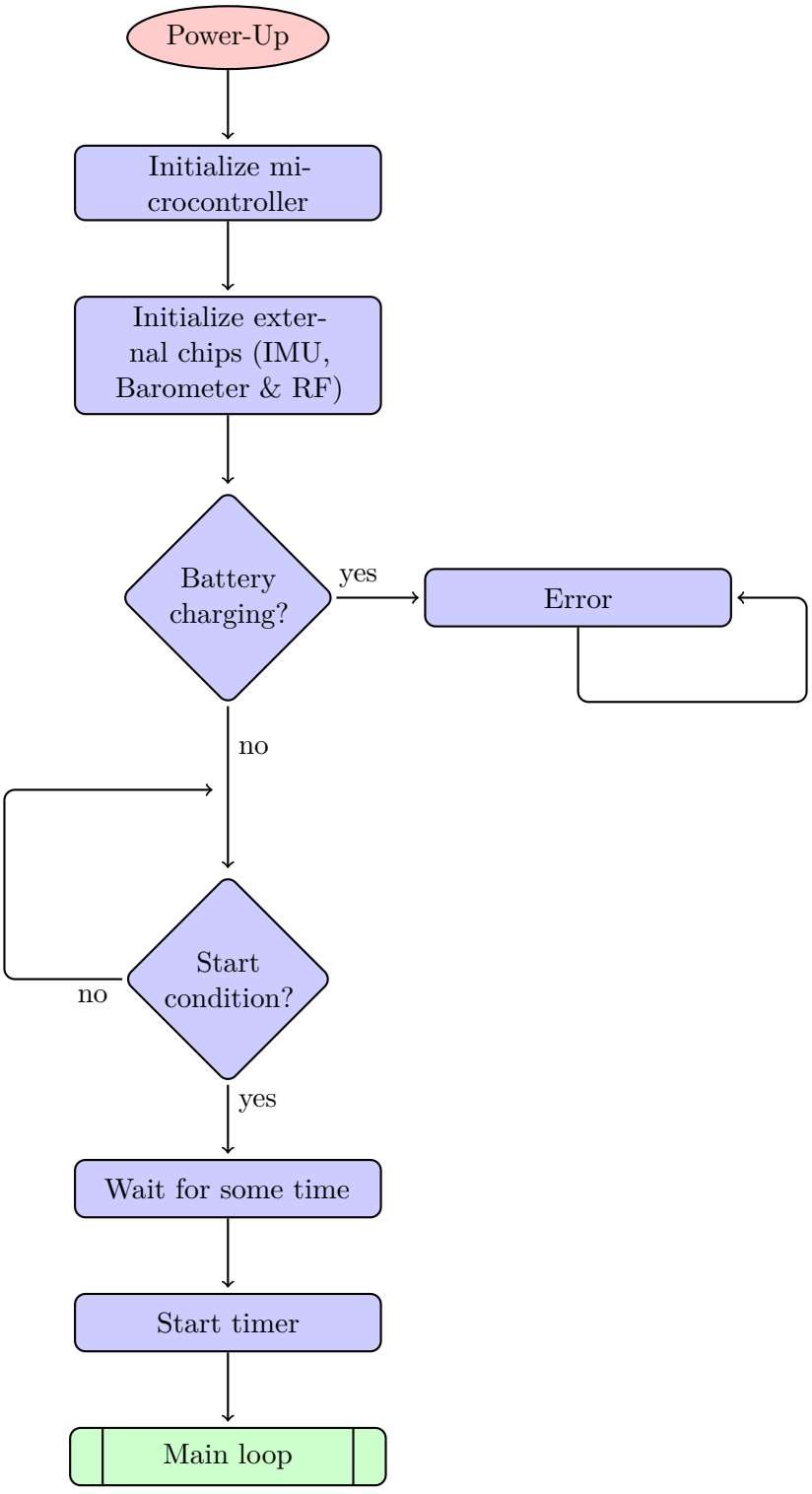
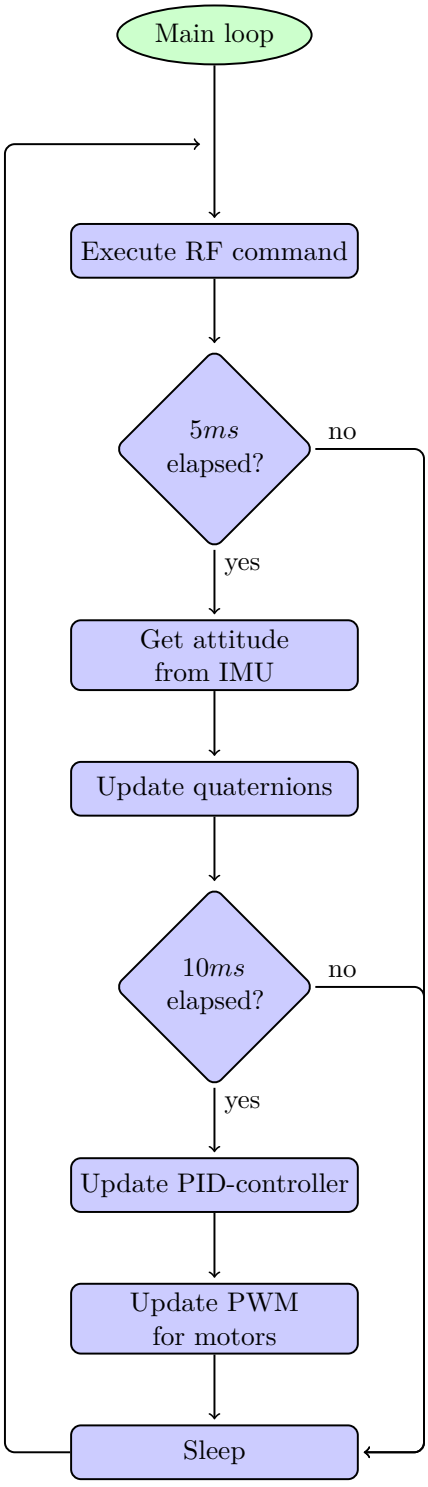
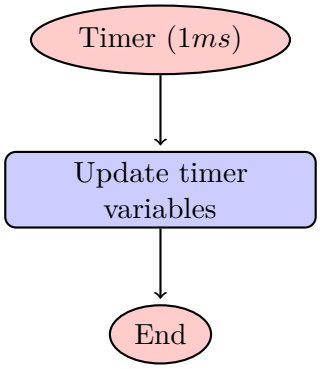


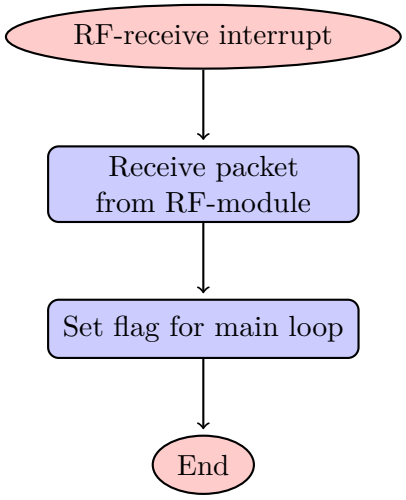
Figure 5.13: Flowchart of the software initialization



(a) Main loop



(b) Timer interrupt



(c) RF interrupt

Figure 5.14: Flowchart of the main program

Whenever the RF module triggers an interrupt the software copies the message from the module, decodes and sets a flag for the main loop, which then executes the command. Possible commands can be seen in Appendix A.3.

5.2 Component Selection

In this section we show which components were used for our quadcopter, and why. A small overview of the components was already given in Section 5.1.4.

As a microcontroller we selected an **ATxmega32C4** from **Atmel**, which is a small but powerful 8-bit microcontroller with a clock rate of up to 32 MHz. It has every required digital interface integrated, and is fast enough to perform sensor fusion on board.

The IMU is an **LSM9DS0** from **ST**, which has 9 Degrees of Freedom (DOF) and an integrated temperature sensor. It is known for its low drifts and it has an embedded buffer which can be used to perform a first filtering of the data.

To measure the current height of the quadcopter we use a barometer, more precisely an **LPS25H** from **ST**. This sensor is highly accurate in the mm range and also has an embedded data buffer, which is used as a mean filter.

As we have to send commands to the quadcopter, an RF module is required. For this we use a dedicated module, the **MRF24J40MA** from **Microchip**. It implements the ZigBee protocol in the 2.4 GHz range and uses very little power while providing a higher range (>10 m) than similar modules. The limited range is not a concern for our application, because our quadcopter should nevertheless only be used indoors.

The battery is the same as on the Crazyflie [bita], which is a small Lithium Polymer (LiPo) battery with a capacity of 240 mAh and a weight of 7.1 g.

Our quadcopter is quite small, which enables us to use the PCB as the frame and additional 3D-printed motor mounts to hold the motors.

The motors are also the same as on the Crazyflie [bita], which uses coreless DC motors with a diameter of 7 mm, a maximum speed of 58 800 rpm and a rated current of 1000 mA.

To drive these motors we built the motor driver ourselves and used a **SI2366DS** as a MOSFET for the high currents. It is rated for currents of up to 5.4 A, which means that it won't get too hot in our application.

For our quadcopter we used the same propellers as the Crazyflie [bita], because we could not find any others in that size, which also fit onto the shafts of the motors. They have a diameter of 45 mm and are available as clockwise and counter-clockwise versions.

5.2.1 Costs

The result of this design process and the selected components was a small quadcopter with a unit price of 370 € if one prototype is ordered, or a price of about 135 € if it is

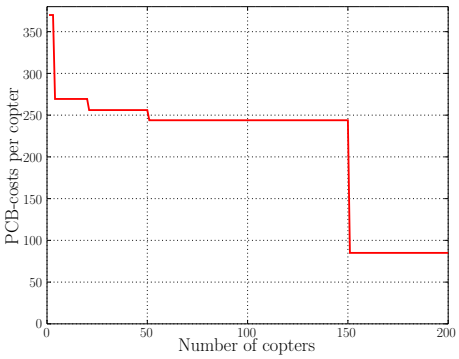
manufactured in large quantities. The bill of materials can be seen in the appendix in Table B.1 and B.2.

Most of the costs of the prototype are dictated by the manufacturing of the PCB and its assembly[4]. As can be seen in Figure 5.15a if only one board is manufactured and assembled, the costs are very high with 370 €. These costs decrease to about 250 € if 50 boards are manufactured. The only way to further decrease the costs is either to manufacture and assemble the boards on one's own, or to increase the quantities and order a series production of more than 1000 pieces. That way the costs should decrease to less than 100 € per PCB, which is depicted in the figure by the large step at the end.

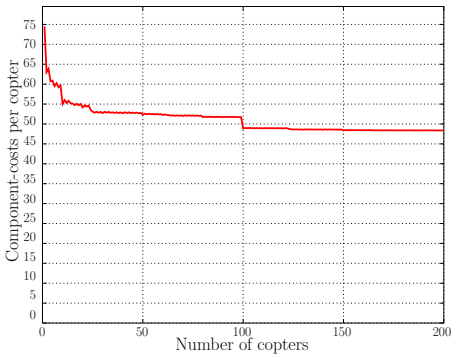
Of course, not only the production of the PCB amounts to the costs, but also the electrical and mechanical components of the quadcopter. Figure 5.15b shows the costs of all components, which were used to build the quadcopter, as a function of the number of quadcopters built. As can be easily seen, the more quadcopters are built, the less one has to pay for the parts of one quadcopter. The figure only shows the costs for up to 200 quadcopters, because this value is still reasonable. If one wants to buy many more quadcopters, the prices would further slowly fall until they converge to about 45 € per piece for 10000 quadcopters.

Figure 5.15c shows the sum of the costs as a function of the number of quadcopters. As one can see the total costs per quadcopter total to about 135 € per quadcopter if 200 quadcopters are built at once.

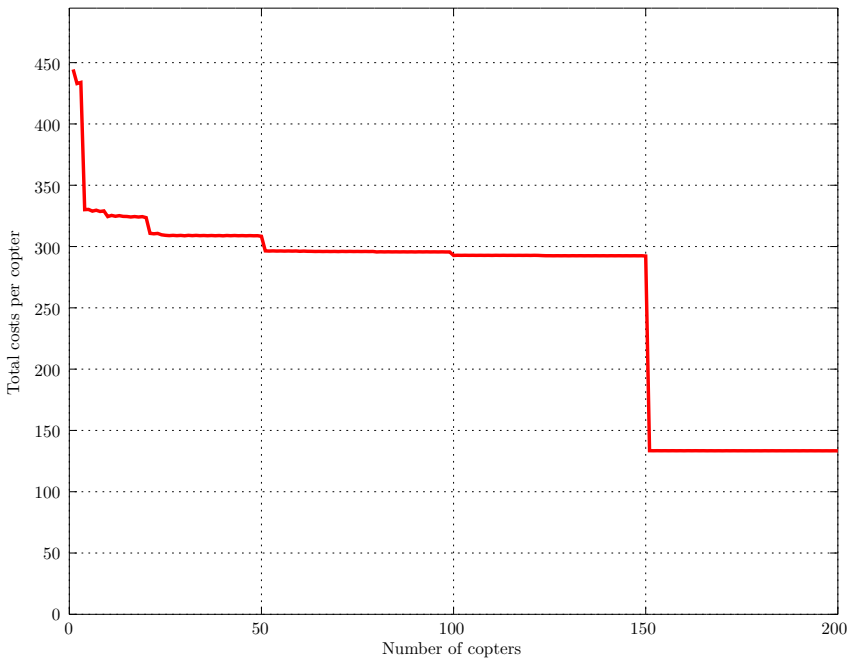
[4]We want to again give our thanks to **PIU-Printex** for their fast production and list of prices for higher quantities.



(a) PCB-costs



(b) Component costs



(c) Total costs

Figure 5.15: Different kinds of costs as a function of the number of quadcopters

CHAPTER 6

Critical Reflection

A comparison to existing design flows is not really feasible, because the available ones are very general and do not discuss the cases where more departments have to work together to create a new Cyber-Physical System (CPS). The reason for this is that most companies keep their internal processes in secret, as discussed in Chapter 2.1.4.

We are nevertheless confident that our proposed model based design flow can be used in the upcoming "Industry 4.0" to create CPSs either from scratch or from existing templates. To test this assumption we used this design flow to implement a simple use case, namely a palm-sized quadcopter. Of course, as the author was only a single person, it is very hard to speak of a "parallel" work flow. The obvious limitation was of course that the work in the departments could not be done in parallel, but the author nevertheless tried to adhere to the parallelism by not using results from previous departments when working on an other one. During this work the design flow was revised multiple times to be able to better react on problems, which arose during work. Some of these problems included inconsistent data sheets for our sensors and actors, such as misprints or missing but very important design constraints. Even at the end of this work the design flow was revised once more to account for mistakes, which did not happen in our case, such as inconsistent and therefore impossible to fulfill requirements. At the end of the work we are confident that our design flow can account for most of the errors, which can happen in practice.

When looking at the definition of our design flow, one will see that we did not include a formal definition of the general model, the requirements and the test cases. These formal definitions are very important and help to automatically test each of the models and results of the departments against the requirements. We did not include these in our work, because they were not in the scope of the thesis, as such definitions would have blown up the work on this thesis.

The first focus of our work was to adopt existing design flows to work into a fully parallel fashion. The second focus was then to show that this design flow works by creating a CPS ourselves, which was the quadcopter. Of course, even if there are similar products available on the internet we still had to develop our own quadcopter, because our specifications were different and could not be fully met with existing quadcopters.

The reason we used some components of the "Crazyflie 2.0"[bita], such as the motors, propellers and the battery, for our quadcopter is that these components have the lowest costs when compared to similar parts from other distributors. When one selects the motors, the requirements of the propellers will also be fixed, because they will have to be compatible with the motors. For example, when someone selects a motor with a shaft diameter of 1 mm and a maximum speed of 60 000 rpm[1], the propeller also must have a hole diameter of 1 mm and be able to operate at 60 000 rpm. When one searches the internet for such small motors, such as ours, they will see that there are not many that can be used based on the power requirements. When searching for usable propellers the variety decreases even more, because suitable propellers with a diameter of about 45 mm are pretty rare, because both kinds, clockwise and counter-clockwise, are needed. All in all, the reason why we used the parts of the Crazyflie, was that similar components were either not available, not fully suitable, or they were too expensive.

When one looks at the price of a series production, like the Crazyflie, they will see that the price of our quadcopter seems to be much higher than the price of the Crazyflie (180 $ compared to our 370 €). But this comparison is not suitable, because they are comparing the price of a series production to the price of a prototype. When comparing the price for a reasonably large series production of both quadcopters, one will come to the conclusion that ours is much cheaper with a total price of only about 135 € (see Chapter 5.2.1).

6.1 Open Issues

As already discussed, an open issue of the proposed design flow is that most of the work in the preparatory stage was not done in a formal way, which makes this a matter of future projects, see Section 7.2 for details.

Our requirements for the exemplary use case implementation to test the design flow were to implement a small, low-cost quadcopter, which can be used in future projects as a research platform for swarm applications. Additionally we had to implement a small sample software to show that our quadcopter is indeed able to hold its position while being airborne. For this the software had to hold the quadcopter as steady as possible in the air, without rising or falling too much.

Of course there is still room for improvement such as better sensor fusion algorithms, better smoothing of raw sensor values, calibration of sensors, but that would go beyond the scope of this work. Especially the magnetometer has to be calibrated to eliminate

[1] This is not a misprint. Very small motors, such as ours, typically have a very high revolution speed.

the drifts from hard iron distortions and soft iron distortions. Hard iron distortions stem from constant magnetic fields, which are added to the earth's field. They are produced by magnets in the vicinity of the sensor, in our case the motors are a small source of this type of distortion. Soft iron distortions stem from materials, which are not magnetic themselves such as iron and nickel, but nevertheless influence the earth's magnetic field in a non-constant manner. Hard iron distortions can easily be eliminated by determining the magnitude of the distortion and calculating a respective offset. Soft iron distortions require a more complex algorithm, because they are not constant with the orientation of the sensor. More information can be found at [mag].

When one wants to use our quadcopter for future research in swarm applications, one will have to solve the following problems first, which were not part of this work:

- Search for a way to reliably measure the position of the quadcopter
- Implement the desired application on a computer in a suitable environment, such as *Robot Operating System (ROS)*[ros] or *OpenCV*[ope]
- Adapt our software to implement the *Path Tracker* part of the advanced flight controller (see Figure 6.1)
- Optionally implement a calibration routine for the sensors
- Optionally implement some kind of notification when a charging process ends

While working on the software for our quadcopter we already devised a high level decomposition of the software architecture, which is able to achieve this feat. Our proposal can be seen in Figure 6.1.

The **reactive layer** is the only layer which has already been implemented by us and consists of the controller, which stabilizes the quadcopter. This is done by using a Proportional-Integral-Derivative (PID) controller and a set point of *0* to prevent the quadcopter from tilting and therefore from leaving the position in the air. Only small modifications have to be done to allow for arbitrary set points to let the quadcopter change its attitude and therefore its position.

The **State Estimation** is the part of the software which calculates the actual position of the quadcopter. This layer can be implemented as shown in the figure to only use the sensors of the quadcopter, such as the gyroscope and the accelerometer, to estimate the position. Of course, this estimation will not be usable for real applications as errors will quickly add up to falsify the calculated position. This estimation using only the onboard sensors can even be implemented on the quadcopter, because it does not need excessive computational power. If one wants better estimations of the position they will have to use external sensors, and offload this estimation to an external computer. In all cases this component has to send the estimated position back to the quadcopter to let it follow a set path.

The **Path Tracker** component can be implemented on the microcontroller of the quadcopter. It uses the set point from the Planner and the estimated position from the State Estimation to calculate the set points of the attitude, which are then fed back to the Reactive Layer to let the quadcopter change its course.

The **Planner** is the most complex part of the system because it enables the quadcopters to fly in a swarm. The main task of this component is to monitor the position of each flying quadcopter in a swarm and to calculate the next positions for all of them. This component gets the current positions from the State Estimation and may use some pre fixed path to let the swarm follow that path while also making sure that none of the quadcopters crash into each other or into an external obstacle. The output of this component is a position set point which is then sent to the Path Planner on board of the quadcopters.

A flowchart of a first version of this advanced swarm software can be seen in Figures 6.2 and 6.3. This software uses the onboard sensors to estimate the position of the quadcopter and sends that information to an external computer, which in turn returns the new position set point.

The initialization routine is the same as for the current software and can be seen in Figure 5.13. Most of the program which belongs to the Reactive Layer will not be described here as it is the same as in Section 5.1.5 and was already highlighted there.

The two most notable changes are the blocks highlighted in Figure 6.3. The first block uses the internal sensors, such as the Inertial Measurement Unit (IMU), to estimate the current position, which is then sent to the base station. This can be done by integrating the values from the accelerometer twice to end up with a first estimation of the position. A more advanced algorithm could also integrate the other sensor, such as the gyroscope, magnetometer and barometer in this calculation to reduce the errors of this estimation. Of course the errors will still be substantial, especially if this algorithm is performed for a longer time, as the small errors will quickly add up.

The second new block uses the position set point from the external base station to calculate the new set point of the attitude. This set point is then fed into the PID controllers to calculate new Pulse-Width Modulation (PWM) values for the motors.

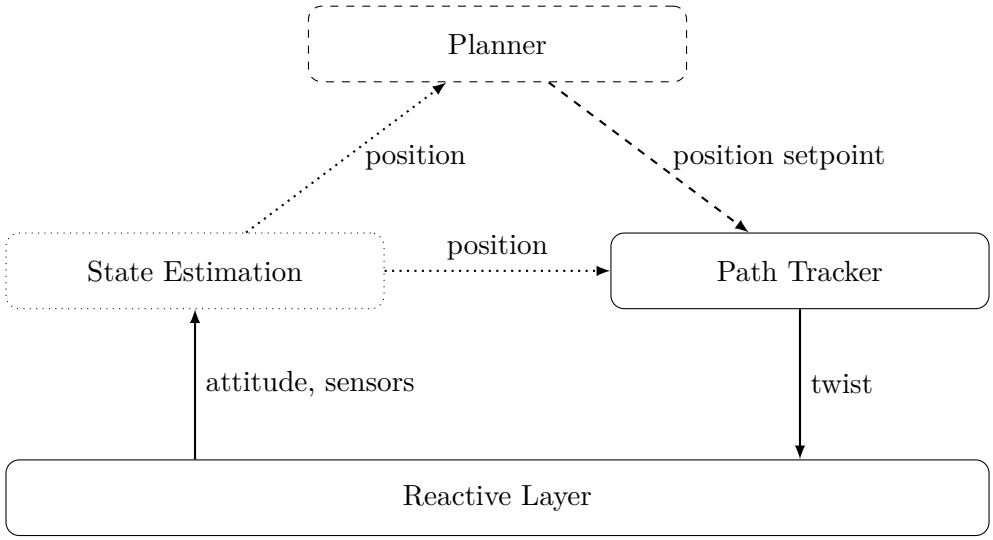
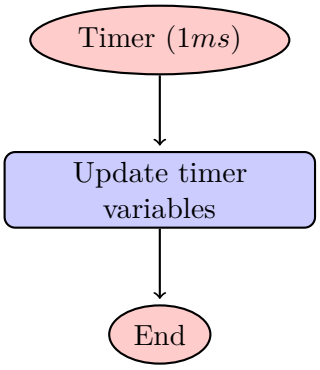
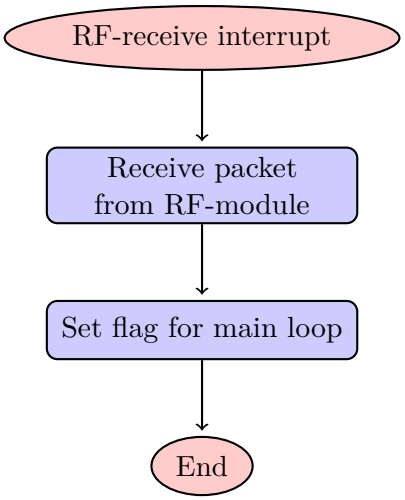


Figure 6.1: Proposed functional decomposition for the advanced flight controller



(a) Timer interrupt



(b) RF interrupt

Figure 6.2: Proposed flowchart for the advanced flight controller

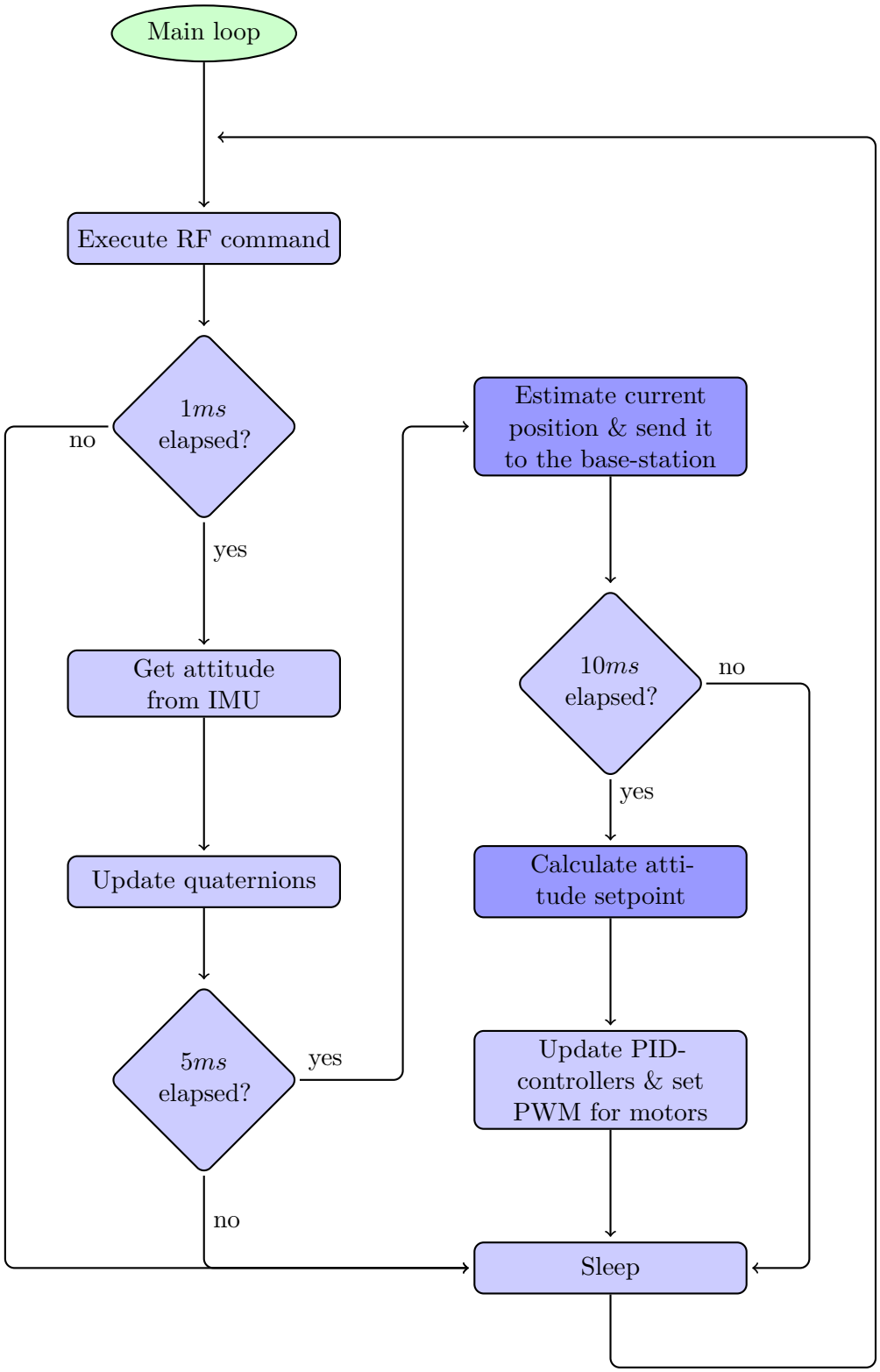


Figure 6.3: Proposed flowchart for the advanced flight controller (cont.)

CHAPTER 7

Summary and Future Work

7.1 Summary

The focus of this work was to create a design process, which is fully parallel in the departments. We therefore devised a flowchart of this new design flow (see Chapter 4), which was then tested with a practical application, namely the construction of a palm-sized, low-cost quadcopter. Because the author of this thesis was only a single person, a fully parallel workflow could not be applied, but the author nevertheless tried to adopt the new design flow. This was done by only working on one department at a time without using the results of prior departments.

The results of the use case implementation should then be a quadcopter and a small software to test if it is indeed able to fly. This exemplary use case should show that the design process is indeed suitable for creating Cyber-Physical Systems (CPSs) either from scratch or from existing templates.

At the end the result was indeed a small low-cost quadcopter, which can hold its position while being airborne.

The next items show the main features of our proposed design flow (see Chapter 3.1 for desriptions of the items):

- Allows for integrated development in different departments
- Allows the design of CPSs
- Independent and parallel workflow of departments
- Model-based design
- Independent validation of departments

Next are some of the main features of the designed quadcopter:

- Low-Cost design of only about 135 € if it is produced in large quantities
- Small size with a diameter of only 108 mm
- Created using existing components from a similar quadcopter (motors, propellers, battery), and newly designed parts (main board, motor mounts)
- Powerful 8-bit microcontroller running with up to 32 MHz
- 9-Degrees of Freedom (DOF) Inertial Measurement Unit (IMU) to measure the attitude
- Barometer to measure the height while flying
- Onboard Radio Frequency (RF) module running the *ZigBee* protocol to be able to exchange commands and messages with a computer
- Integrated charging controller and Micro-USB-port to be able to recharge the Lithium Polymer (LiPo) battery

7.2 Future Work

The focus of this thesis was to define a model based design flow for the construction of CPSs. As the focus did not lie in a complete formal description of the overall requirements and the generic model, this is left as a future work, to be able to formally proof the adherence to the requirements in the different departments. When such descriptions are ready the next step will be the implementation of a formal method to compare the models and test cases of the mechanical, electric, electronic and software departments against the generic model to show that they do not contradict each other. One further step in the software department would then be to formally proof that the created software adheres to the formal requirements.

Some future work also has to be done on the quadcopter because it is not yet suitable for swarm applications as it does not have the possibility to reliably measure its own position. It is only able to stay at a given position because the controller is configured with a set point of "0" for roll and pitch, which means that the quadcopter will not tilt in a direction, which enables it to stay at the starting position. If one would move the quadcopter to another position, it will only compensate for the change in roll and pitch, but not for the change in position. The next steps will therefore have to be the implementation of some kind of local positioning system to reliably measure the position of a quadcopter in the room. One possibility would be to use external cameras and image processing software, such as Robot Operating System (ROS) [ros] or OpenCV [ope], for this part.

When the problem of how to measure the position of the quadcopter has been solved, the next part will be to adapt the current software to accept commands from an external computer.

APPENDIX A

Manuals

A.1 Construction Manual

Before you begin building the quadcopter check if you have all needed parts (for more information on the parts, check the bill of materials):

- The assembled Printed Circuit Board (PCB)
- **One** black and **one** red cable of size AWG 28-24 (corresponds to 0.2 mm^2 - 0.08 mm^2)
- **Two** crimp terminals
- **One** housing for the crimp terminals
- **Four** 3D printed motor-mounts
- **Four** Motors
- **Two** Clockwise (CW) and **two** Counter Clockwise (CCW) propellers
- A short cable tie

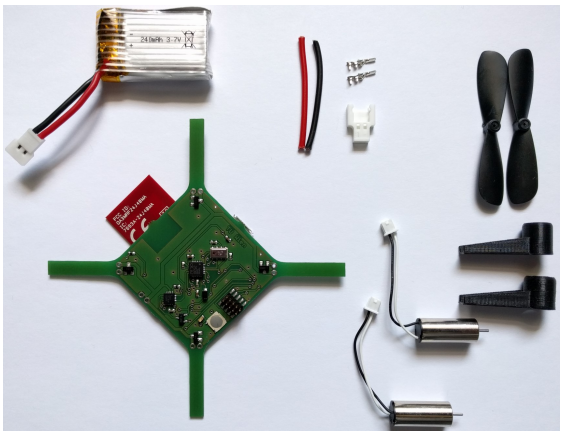


Figure A.1: All components

Further you will need the following tools:

- A tube superglue
- A soldering iron

- Optionally a hand crimp tool

Before we begin assembling the copter we define a the directions for easier references. As shown in Figure A.2 the Radio Frequency (RF)-module is on the bottom of the board and on the upper-left side. The side facing you now is the top side, with the left arm defining the left side of the copter.

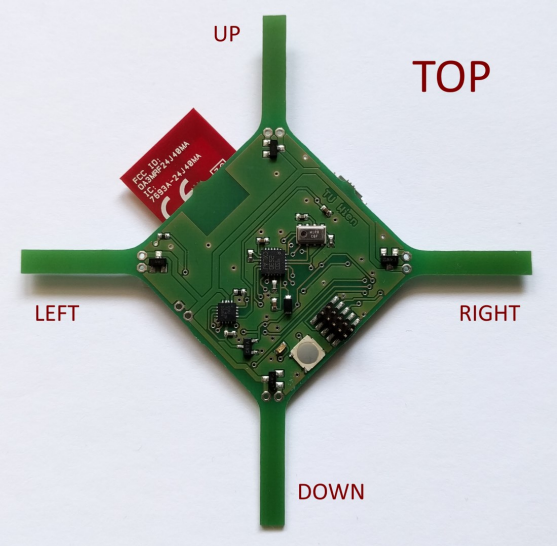


Figure A.2: Definition of directions

Start with the cable which connects the battery to the PCB: First cut the two cables to a length of about 20 mm and strip both ends. Using a hand crimp tool you have to put the crimp terminals onto one end of both cables. If you don't have such a tool you can also solder the crimp terminals onto the cables, in which case you have to be careful not to use too much tin-solder, otherwise the crimp terminals will not fit into the housing. Put the two crimp terminals into the housing such that the red cable is in position 1. Details can be seen in Figure A.3.

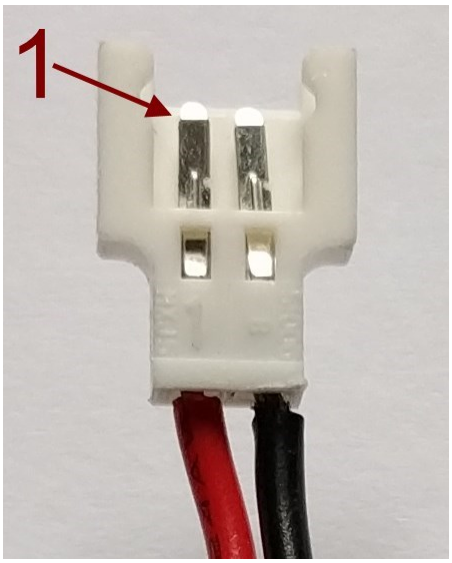


Figure A.3: Placement of the cables in the housing

The next step is to solder the cables onto the PCB. Put the cables through the holes on top of the board, such that you solder the cables on the bottom side. Be careful to put the cables into the correct holes! The red cable goes into the hole, which has two thick copper traces on the top side, and the black one goes into the hole, which connects directly to the ground plane on both sides. I.e. the black cable goes into the hole, which is closer to the RF-module. See Figure A.4 for further help.

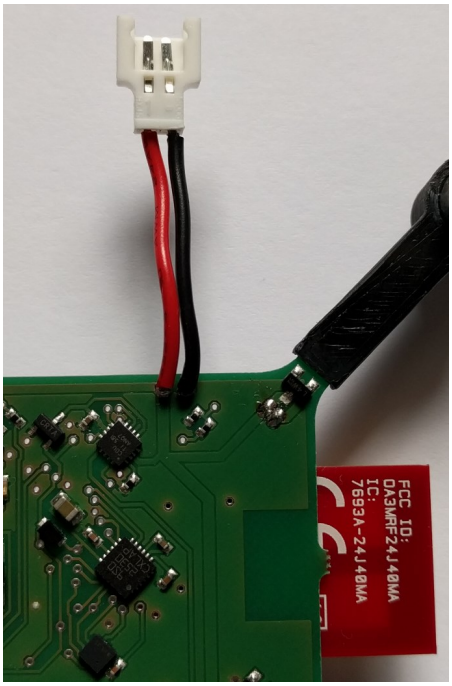


Figure A.4: Correct placement of the cables on the PCB

After that check if the 3D-printed motor-mounts have the correct sizes. First check if all motor-mounts fit onto the four arms of the board. If one does not fit you have to smoothen the inner walls of the mount until it fits. Next check if the motors fit into the round hole of the motor-mounts. Bear in mind that the cables of the motors have to go through the motor-mounts, i.e. the bottom halves of the motors are enclosed in the motor-mounts.

The next step is to glue the motor-mounts onto the PCB, but before that make sure to remove the motors from the motor-mounts. Put some superglue on one of the motor-mounts and press it on one arm of the PCB for a few seconds. Make sure to glue it to the correct side of the board: The open side of the arm of the motor-mount has to face the bottom side of the board. Figure A.5 shows the correct placement of the motor-mounts. Repeat this step until all four motor-mounts are properly attached.

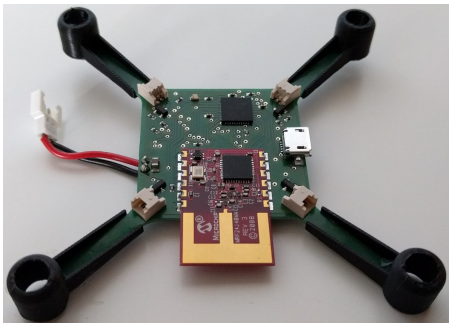


Figure A.5: Placement of the motor-mounts on the PCB

For the next procedures make sure you do not touch the bottom side of the motors, otherwise the superglue might not be working ideally. Put one motor into a motor-mount, such that the cables go through the mount. Pull out the motor a bit and put some superglue on the bottom half of the metal case, and be extra careful not to touch the glued side. Then press the motor into the motor-mount for a few seconds until it sits tight. Repeat this step for all four motors and then let the superglue dry for a few minutes before continuing the assembly. Figure A.6 shows how the result should look like. Continue with carefully twisting the cables of the motors a bit (3-5 windings) and connect the connectors with their respective counterpart on the board. Be careful not to rip the cables out of the motors!

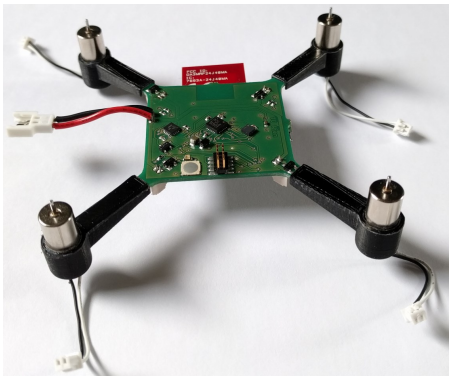


Figure A.6: Motors glued onto the motor-mounts

Then we have to attach the propellers to the motors: First take the CW propellers, i.e. the ones with the letter 'A' on them, and attach them to the two motors on the left and on the right side. Be careful not to put the propellers on the motors upside-down. All propellers have a small ring at the center, which indicates the top side, i.e. you should be able so see those rings when the propellers are properly mounted! Now take the CCW propellers, i.e. the ones without any letter on them, and attach them to the two remaining motors on the upper and lower side. The correct location of the propellers can be seen in Figure A.7.

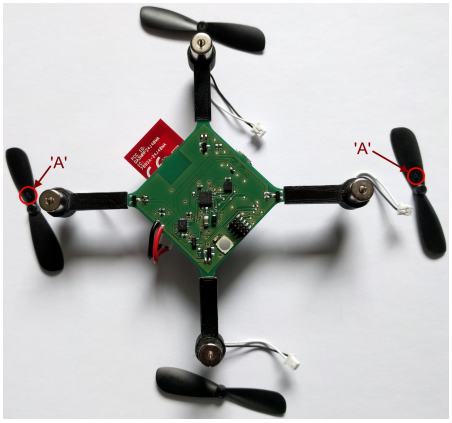


Figure A.7: Correct location of the propellers

At last bend the power cables to the bottom of the quadcopter, where the battery should be fixated using a short cable tie. For best performance you should pay attention to the balance of the quadcopter, i.e. place the battery in such a way that the balance point lies in the middle of the quadcopter.

A.2 Programming Manual

In this section we will show you how to program the quadcopter.

You will need the following tools:

- **Atmel Studio** (version 7.0 or greater) or any compiler which can compile to

XMEGA targets.[1]

- **JTAGICE3** or **Atmel-ICE** or any programmer which has a 10-pin 50-mil connector and the ability to program over the Program and Debug Interface (PDI).
- A micro-USB cable

First open Atmel Studio, load the respective project and compile it. Then connect the micro-USB cable and the programmer to the quadcopter as shown in Figure A.8. The micro-USB cable has to be connected to a computer as it is responsible for powering the board. Alternatively you can connect the battery to the board, but this setup will only work for a few minutes as the battery will also discharge when you program the microcontroller.

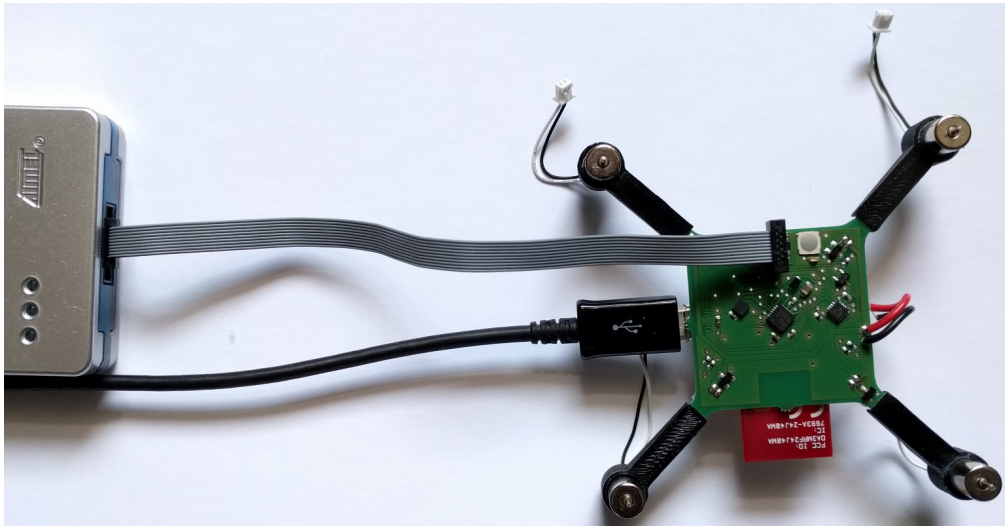


Figure A.8: Correct connections of the cables

In Atmel Studio you now have to open the *Device Programming* dialog and flash the device. Alternatively you can also click on *Start Debugging* as this will also flash the board. If you get an error you have to check if the correct programmer, microcontroller (**ATxmega32C4**), interface (**PDI**) and the correct interface speed ($\leq$1 MHz) are set! When the device is flashed you can disconnect the programmer and the USB cable and connect the battery to execute the program. You can also leave the programmer connected and debug the device. Be careful not to turn on the motors in this case if the propellers are already attached, as the cable may get entangled!

If you don't need debug outputs, you can disable them in the project settings to save some flash-space. To do this, open the file *defines.h* in Atmel Studio and comment out the line **#define DEBUG_OUTPUT**. Also open the project settings and select *Toolchain* on

[1] You can download Atmel Studio at Atmel's official homepage

the left side. Navigate to *AVR/GNU Linker* and disable the checkbox "**Use vprintf library**". These settings will save at least 3 kB of space in flash-memory, depending on the number of used debug-strings.

A.3 Wireless Communication

The copter has a ZigBee RF-module on-board, which can be used to control it or to receive status messages from the quadcopter. For the PC you will need a module with a USB-connector, such as the **Digi XBee XBP24-AWI-001** and the appropriate USB-dongle. Other ZigBee-modules may also work, but they must have at least the following specifications:

- Operating frequency of 2.4 GHz
- Standard ZigBee-protocol (**IEEE 802.15.4**), no additions (such as *ZigBee Mesh*)

The following steps assume you have the Digi XBee module.
Download *XCTU* from Digi's homepage [dig], install and open it. Plug in your XBee module and click on *Add a radio module*. In the next window confirm that the correct Serial/USB port is selected and set the correct interface settings. After clicking on *Finish* you have to select the module on the left side of the main window. The software now reads all configuration parameters from the device and displays them on the right side.

You have to correctly set the following parameters:

Parameter		Setting
CH	Channel[2]	C
ID	PAN ID	2200
DH	Destination Address High	0
DL	Destination Address Low	FFFF
MY	16-bit Source Address	2
MM	MAC Mode	802.15.4 no ACKs
CE	Coordinator Enable	Coordinator

After setting these parameters you can either use XCTU's integrated terminal or an external application to connect to the corresponding port and exchange commands and messages with the copter.

[2] Other modules may have a different naming style. In that case select the setting which corresponds to ZigBee channel 12.

The following commands are included in the standard version of the software:

Command	**Meaning**
"p <value>"	Set the p-value of the PID controller to <value>
"i <value>"	Set the i-value of the PID controller to <value>
"d <value>"	Set the d-value of the PID controller to <value>
"s"	Save the PID values in the internal EEPROM
"r"	Restore the PID values from the internal EEPROM
"g"	Start the main program and the motors when it is not running
	Stop the main program and the motors when it is running

A.4 Charging the Battery

The battery can be recharged directly by the quadcopter, without having to use a special external Lithium Polymer (LiPo) charger.

To charge the battery the following procedure should be used:

1. Connect the battery to the quadcopter
2. Do **not** press the button to start the program!
3. Connect a micro-USB cable to the respective connector and connect the other end to a suitable power supply, such as a phone's charging adapter. The USB port of a computer may also work, but the charging time will be substantially longer, as the computer only provides 100 mA in this situation.
4. The charging controller on board the quadcopter automatically begins the charging process. Now the button on the quadcopter may be pressed, but it has no function as the software does not start when the battery is being charged.
5. After at most 3 hours the battery is fully charged. Now you can disconnect the battery and the USB cable from the quadcopter.

APPENDIX B

Design files

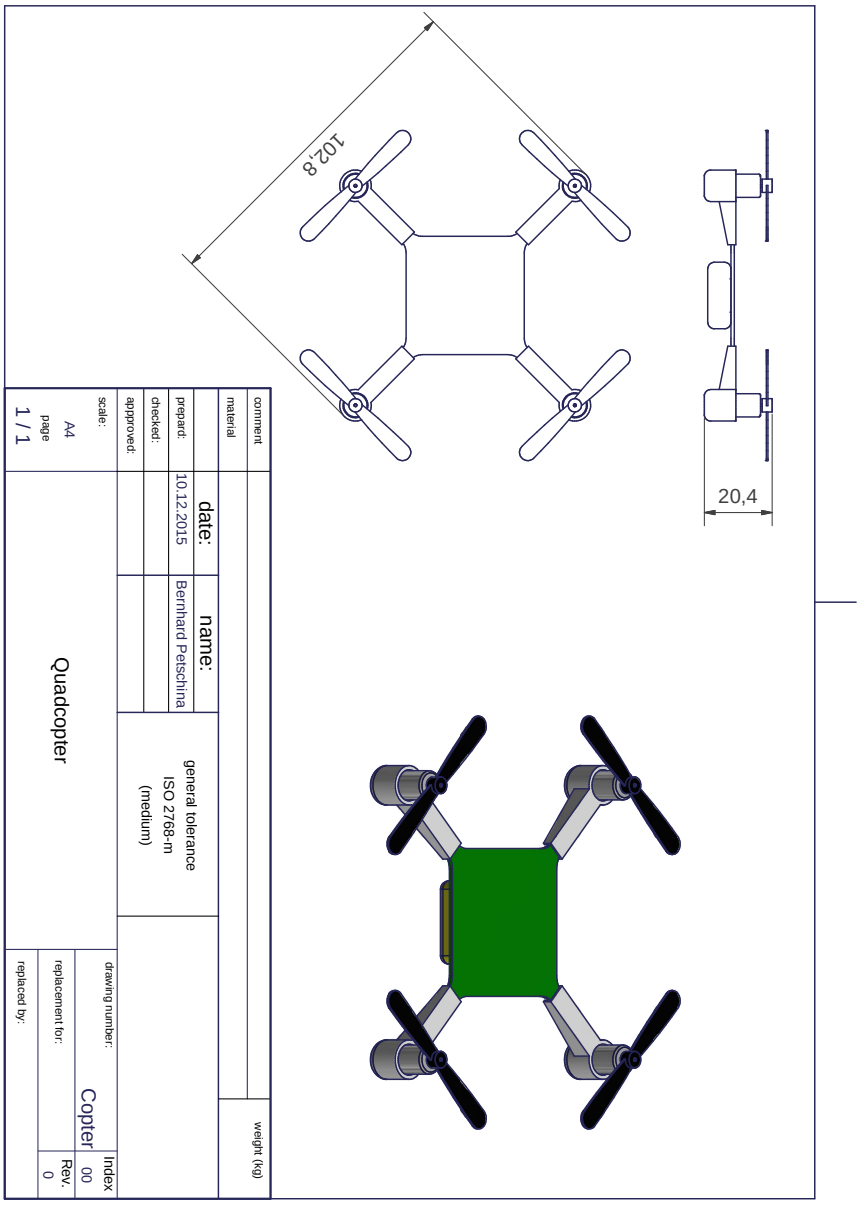


Figure B.1: 3D model of the quadcopter

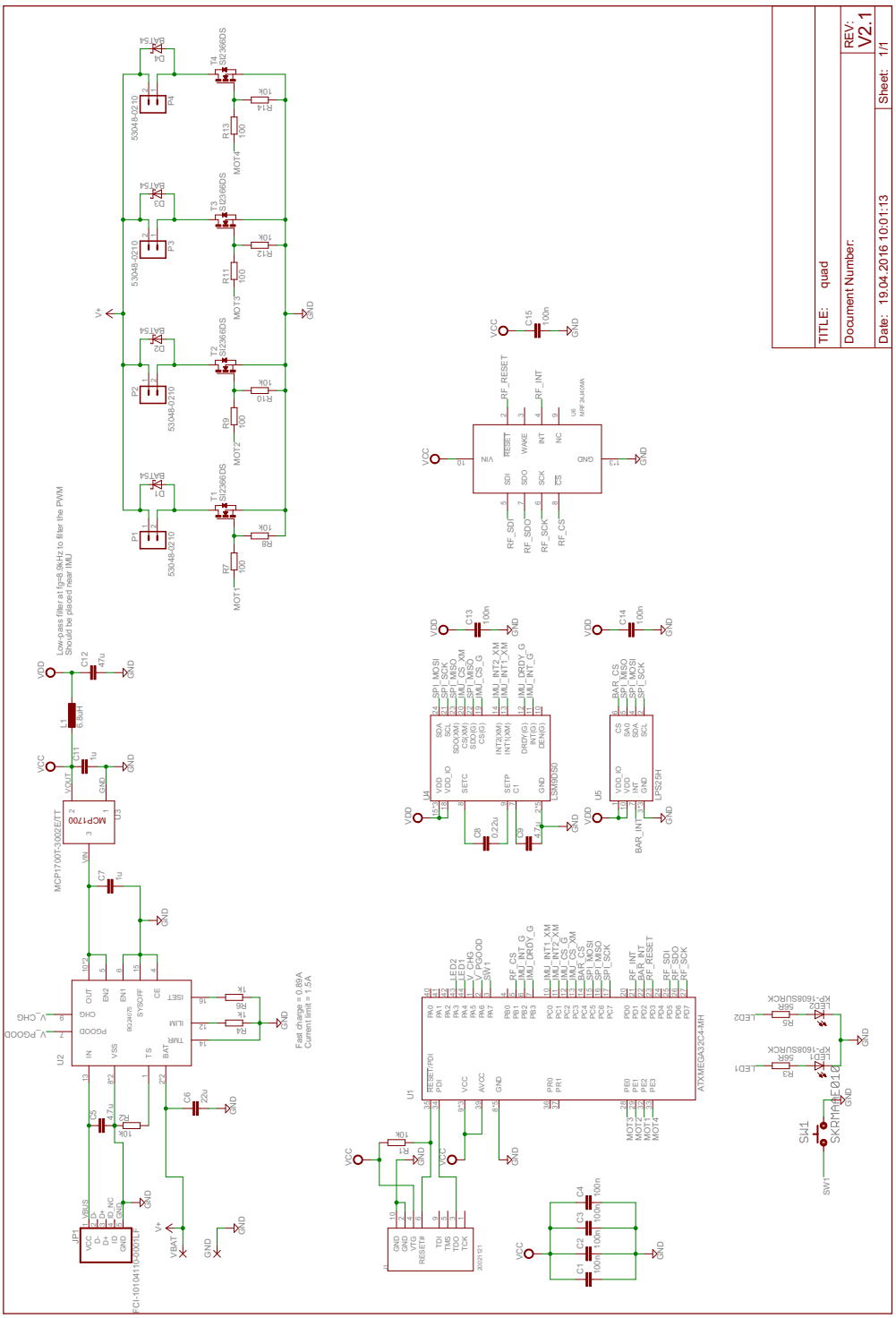
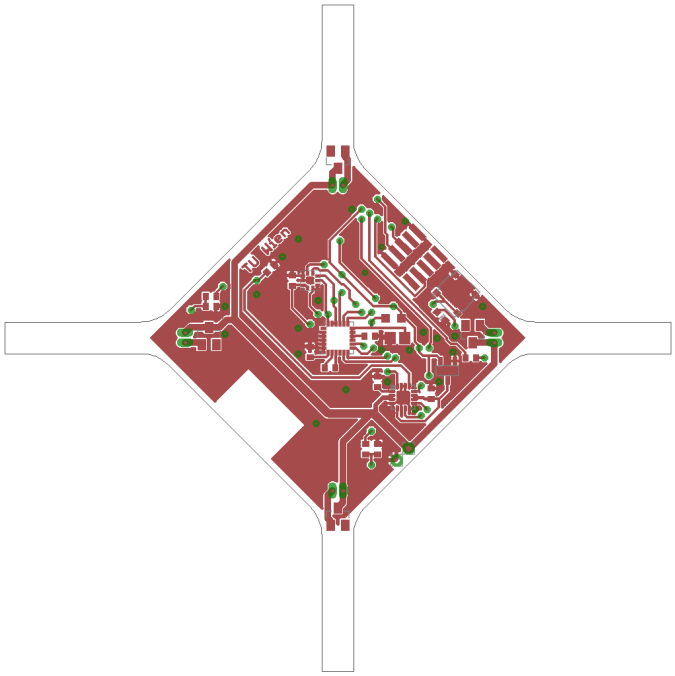
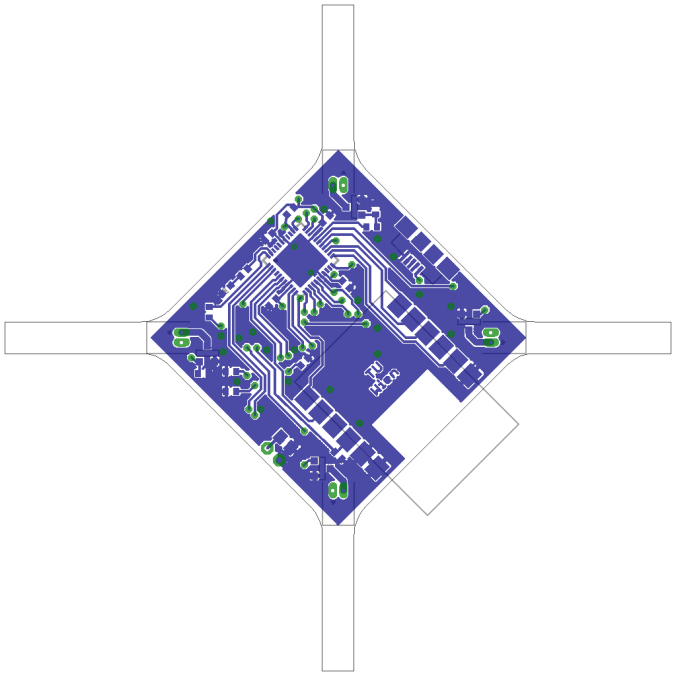


Figure B.2: Schematic of the quadcopter



(a) Top side



(b) Bottom side

Figure B.3: Board of the quadcopter

Part	Name	Ordernumber	Provider	Qty	Unit Price	Total Price
Microcontroller	ATxmega32C4-MH	2353906	Farnell	1	2.61€	2.61€
IMU	LSM9DS0	2433079	Farnell	1	10.6€	10.6€
Barometer	LPS25HBTR	829-6962	RS	1	3.6€	3.6€
RF-module	MRF24J40	1630202	Farnell	1	8.25€	8.25€
Charging controller	BQ2407RGTT	1754755	Farnell	1	2.75€	2.75€
Voltage regulator	MCP1700T-3002E/TT	1331482	Farnell	1	0.38€	0.38€
MOSFET	SI2366DS-T1-GE3	2056680	Farnell	4	0.29€	1.17€
Diode	BAT54	1081190	Farnell	4	0.19€	0.75€
USB connector	FCI-10104110-0001LF	2293753	Farnell	1	0.39€	0.39€
Debugging connector	FCI-20021121-00010C4LF	1865279	Farnell	1	0.71€	0.71€
Capacitor	100nF, 0402	1759388	Farnell	7	0.01€	0.04€
Capacitor	1uF, 0402	2496814	Farnell	2	0.01€	0.02€
Capacitor	220nF, 0402	2496815	Farnell	1	0.01€	0.01€
Capacitor	4.7uF, 0402	2469393	Farnell	2	0.21€	0.42€
Capacitor	22uF, 0805	1759415	Farnell	1	0.22€	0.22€
Capacitor	47uF, 0805	2362109	Farnell	1	0.42€	0.42€
Resistor	100R, 0402	2072515	Farnell	4	0.01€	0.05€
Resistor	10k, 0402	2072517	Farnell	6	0.01€	0.08€
Resistor	1k, 0402	2072516	Farnell	2	0.01€	0.03€
Inductance	6.8uH, 0805	2288749	Farnell	1	1.6€	1.6€
Resistor	56R, 0402	2059206	Farnell	2	0.01€	0.02€
LED	KP-1608SURCK	2290329	Farnell	2	0.1€	0.19€
Motor connector	Molex 53048-0210	1012259	Farnell	4	0.25€	1.01€
Switch	SKRMAAE010	2056854	Farnell	1	0.28€	0.28€

Table B.1: Bill of Materials for the PCB

Part	**Ordernumber**	**Provider**	**Quantity**	**Unit Price**	**Total Price**
Crimp terminal	670-6417	RS	1	0.05 €	0.05 €
Crimp housing	670-7000	RS	1	0.29 €	0.29 €
Lipo Battery for Crazyflie 2.0	-	ladroneshop.com	1	5.5 €	5.5 €
Crazyflie Nano Quadcopter Prop	-	ladroneshop.com	1	4.78 €	4.78 €
Crazyflie 2.0 - Spare 7x16mm DC motor	-	ladroneshop.com	4	3.5 €	14 €

Table B.2: Bill of Materials for the other components

List of Figures

4.1 Flowchart of the design flow 24
4.2 Flowchart of the design flow (cont.) 25

5.1 Plot of the flight time over the capacity of the battery 32
5.2 Plot of the flight time over the weight of the payload 33
5.3 Flowchart of the process to create a generic model 34
5.4 Plot of the thrust characteristic 35
5.5 Constraints of the Finite Element-Analysis (FE-Analysis) 37
5.6 Maximum stress 37
5.7 Maximum deformation 38
5.8 Minimum safety factor 38
5.9 Comparison between the model and the real part 39
5.10 The four motor drivers 40
5.11 Passive lowpass filter 40
5.12 Simulation of the lowpass filter 40
5.13 Flowchart of the software initialization 43
5.14 Flowchart of the main program 44
5.15 Different kinds of costs as a function of the number of quadcopters 47

6.1 Proposed functional decomposition for the advanced flight controller 53
6.2 Proposed flowchart for the advanced flight controller 53
6.3 Proposed flowchart for the advanced flight controller (cont.) 54

A.1 All components 59
A.2 Definition of directions 60
A.3 Placement of the cables in the housing 60
A.4 Correct placement of the cables on the PCB 61
A.5 Placement of the motor-mounts on the PCB 61
A.6 Motors glued onto the motor-mounts 62
A.7 Correct location of the propellers 62
A.8 Correct connections of the cables 63

B.1 3D model of the quadcopter 68
B.2 Schematic of the quadcopter 69

B.3 Board of the quadcopter 70

List of Tables

2.1 Properties of different RF modules 12
2.2 Some RF protocols sorted in ascending order based on their complexity . . . 13

3.1 Comparison of quadcopters 21

B.1 Bill of Materials for the PCB 71
B.2 Bill of Materials for the other components 72

Glossary

cognitive complexity In short, cognitive complexity refers to the cognitive effort needed to understand a model [Kop08]. In our thesis we use this term to describe the complexity of different hardware features.. 9

DDR A parallel bus operating with Double Data Rate uses both, the rising and falling edge of the clock signal to transfer data.. 9

FE-Analysis The finite element method can be used to numerically approximate a solution for partial differential equations. This method breaks a large problem down into smaller, simpler ones. These simple equations are then assembled into the large system and numerical methods are then used to solve these equations while minimizing the error. In mechanical systems this analysis can be used to compute the maximum mechanical stress and its location, amongst other features, for complex components.. 27, 36, 73

mechanical stress Mechanical stress, with the physical unit Pascal (Pa), is a quantity, which describes the forces the particles in a material exert on each other. In this thesis we use the term in the looser sense as a synonym for the internal force in a mechanical component. The following short summary is enough background knowledge for our thesis: Each mechanical component has a certain maximum stress, which it can handle until it deforms. This maximum stress depends on many factors, such as the material, physical forms and sizes, magnitude and direction of the external force. When an external force is exerted on the component, this creates a certain mechanical stress on it. Depending on the size of the stress the component may bend reversibly. If the mechanical stress is greater than a certain maximum mechanical stress the component deforms irreversibly and may even break.. 36

PDI The Program and Debug Interface (PDI) is a proprietary interface for programming and debugging of Atmel's XMEGA microcontrollers. PDI is a 2-wire interface and uses a 10-pin or a 6-pin physical interface.. 63

PID A Proportional-Integral-Derivative (PID) controller is a type of controller to calculate a set point from an error input. It uses a proportional part to calculate the overall set point. The integral part reduces the error from long-term drifts, and the derivative part dampens possible oscillations.. 42, 51

skew Skew describes the difference in arrival time of simultaneously transmitted bits in parallel busses.. 9

Acronyms

API Application Programming Interface. 20

BLE Bluetooth Low Energy. 13

CCW Counter Clockwise. 59, 62

COTS Commercial Off-The-Shelf. 19

CPS Cyber-Physical System. xi, xiii, 2–4, 17, 23, 26, 49, 50, 55, 56

CW Clockwise. 59, 62

DDR Double Data Rate. 9

DOF Degrees of Freedom. 11, 45, 56

FE finite element. 25

FE-Analysis Finite Element-Analysis. 27, 36, 37, 73

GPS Global Positioning System. xi, xiii, 2, 13, 14, 19–21

HIL Hardware in the Loop. 28

I^2C Inter-Integrated Circuit. 9

IMU Inertial Measurement Unit. 11, 12, 41–45, 52, 54, 56, 71

LED Light Emitting Diode. 9, 41, 71

Li-ion Lithium-Ion. 14

LiPo Lithium Polymer. 14, 45, 56, 65

MBD Model-Based Design. 6

NiMH Nickel-Metal Hydride. 14

PCB Printed Circuit Board. 4, 8, 12–14, 29, 36, 45–47, 59–61, 71, 73, 74

PCI Peripheral Component Interconnect. 9

PDI Program and Debug Interface. 63

PID Proportional-Integral-Derivative. 42, 51, 52, 54

PWM Pulse-Width Modulation. 9, 15, 39, 42, 44, 52, 54

RC Remote Control. 20, 21

RF Radio Frequency. 12, 13, 41–45, 53, 54, 56, 60, 61, 64, 71, 74

ROS Robot Operating System. 51, 56

SIL Software in the Loop. 28

SLAM Simultaneous Localization And Mapping. 1

SPI Serial Peripheral Interface. 9

UAV Unmanned Aerial Vehicle. 1

Bibliography

[ama] Quadrocopter x6. URL: https://www.amazon.de/Jamara-038320-Poky-Quadrocopter-2-4/dp/B00TJLS6LQ [cited 25 September 2016].

[asc] Pelican. URL: http://www.asctec.de/uav-uas-drohnen-produkte/asctec-pelican/ [cited 25 September 2016].

[atm] Atmel studio. URL: http://www.atmel.com/tools/atmelstudio.aspx [cited 25 September 2016].

[aut] Autodesk inventor. URL: http://www.autodesk.com/products/inventor/overview [cited 25 September 2016].

[BFV09] L.E. Barnes, M.A. Fields, and K.P. Valavanis. Swarm formation control utilizing elliptical surfaces and limiting functions. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 39(6):1434–1445, Dec 2009.

[BGdRGP14] J.A. Benito, G. Glez-de Rivera, J. Garrido, and R. Ponticelli. Design considerations of a small uav platform carrying medium payloads. In *Design of Circuits and Integrated Circuits (DCIS), 2014 Conference on*, pages 1–6, Nov 2014.

[bita] Crazyflie 2.0. URL: https://www.bitcraze.io/crazyflie-2/ [cited 25 September 2016].

[bitb] Lisa/s. URL: http://1bitsquared.com/collections/autopilots/products/lisa-s [cited 25 September 2016].

[bitc] Thrust investigation. URL: https://wiki.bitcraze.io/misc:investigations:thrust [cited 25 September 2016].

[Bur10] Leon Keith Burkamshaw. Towards a low-cost quadrotor research platform. Master's thesis, Naval Postgraduate School, 2010.

[BVD14] A.A. Bandala, R.R.P. Vicerra, and E.P. Dadios. Formation stabilization algorithm for swarm tracking in unmanned aerial vehicle (uav) quadrotors. In *TENCON 2014 - 2014 IEEE Region 10 Conference*, pages 1–6, Oct 2014.

[cad] Cadsoft eagle. URL: `https://cadsoft.io/` [cited 25 September 2016].

[CH16] Clyde F. Coombs Jr. and Happy T. Holden. *Printed Circuits Handbook*. McGraw-Hill, 2016.

[con] Quadrocopter 450 arf. URL: `http://www.conrad.at/ce/de/product/208000/QUADROCOPTER-450-ARF-35-MHz` [cited 25 September 2016].

[CPD10] A. Cunningham, M. Paluri, and F. Dellaert. Ddf-sam: Fully distributed slam using constrained factor graphs. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3025–3030, Oct 2010.

[CTK11] Ai-Ling Chan, Su-Lim Tan, and Chu-Lih Kwek. Sensor data fusion for attitude stabilization in a low cost quadrotor system. In *Consumer Electronics (ISCE), 2011 IEEE 15th International Symposium on*, pages 34–39, June 2011.

[dig] Digi xctu. URL: `http://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu` [cited 25 September 2016].

[DNY13] E. Davis, B.E. Nizette, and Changbin Yu. Development of a low cost quadrotor platform for swarm experiments. In *Control Conference (CCC), 2013 32nd Chinese*, pages 7072–7077, July 2013.

[EA07] B. Erginer and E. Altug. Modeling and pd control of a quadrotor vtol vehicle. In *Intelligent Vehicles Symposium, 2007 IEEE*, pages 894–899, June 2007.

[HGB10] F. Hoffmann, N. Goddemeier, and T. Bertram. Attitude estimation and control of a quadrocopter. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1072–1077, Oct 2010.

[HGS+13] Dac-Tu Ho, E.I. Grotli, P.B. Sujit, T.A. Johansen, and J. Borges de Sousa. Performance evaluation of cooperative relay and particle swarm optimization path planning for uav and wireless sensor network. In *Globecom Workshops (GC Wkshps), 2013 IEEE*, pages 1403–1408, Dec 2013.

[IEE98] IEEE. Software requirements specification. IEEE 830-1998, Institute of Electrical and Electronics Engineers, 1998.

[JKG08] A. Jaimes, S. Kota, and J. Gomez. An approach to surveillance an area using swarm of fixed wing and quad-rotor unmanned aerial vehicles uav(s). In *System of Systems Engineering, 2008. SoSE '08. IEEE International Conference on*, pages 1–6, June 2008.

[Kop08] H. Kopetz. The complexity challenge in embedded system design. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 3–12, May 2008.

[lib] Librepilot. URL: `http://www.librepilot.org/` [cited 25 September 2016].

[lin] Ltspice. URL: `http://www.linear.com/designtools/software/` [cited 25 September 2016].

[mad] Open source imu and ahrs algorithms. URL: `http://www.x-io.co.uk/open-source-imu-and-ahrs-algorithms/` [cited 25 September 2016].

[mag] Compensating for tilt, hard-iron, and soft-iron effects. URL: `http://www.sensorsmag.com/sensors/motion-velocity-displacement/compensating-tilt-hard-iron-and-soft-iron-effects-6475` [cited 25 September 2016].

[MHO14] O. Magnussen, G. Hovland, and M. Ottestad. Multicopter uav design optimization. In *Mechatronic and Embedded Systems and Applications (MESA), 2014 IEEE/ASME 10th International Conference on*, pages 1–6, Sept 2014.

[MHV11] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan. Estimation of imu and marg orientation using a gradient descent algorithm. In *2011 IEEE International Conference on Rehabilitation Robotics*, pages 1–7, June 2011.

[mik] Mikrokopter. URL: `https://www.mikrocontroller.com/index.php?main_page=product_info&cPath=80&products_id=898&zenid=q2o6lo7gjgdg2655m9j1s3niq5` [cited 25 September 2016].

[NM10] Gabriela Nicolescu and Pieter J. Mosterman. *Model-Based Design for Embedded Systems*. CRC Press, 2010.

[ope] Open source computer vision. URL: `http://opencv.org/` [cited 25 September 2016].

[pap] Paparazzi uav. URL: `http://wiki.paparazziuav.org/wiki/Main_Page` [cited 25 September 2016].

[par] Ar drone 2.0. URL: https://store.parrot.com/de/ar-drone-20/387-ardrone-power-edition-3520410014635.html [cited 01 July 2016].

[PDC15] Umberto Papa and Giuseppe Del Core. Design and assembling of a low-cost minu uav quadcopter system. Technical report, Department of Science and Technology, University of Naples, 2015.

[PK05] Education Transfer Plan and Seyyed Khandani. Engineering design process. 2005. URL: http://www.dphu.org/uploads/attachements/books/books_2547_0.pdf.

[qua] How fast can a quadcopter fly? URL: http://klsin.bpmsg.com/how-fast-can-a-quadcopter-fly/ [cited 25 September 2016].

[ros] Robot operating system. URL: http://www.ros.org/ [cited 25 September 2016].

[Roy87] W. W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, pages 328–338, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press. URL: http://dl.acm.org/citation.cfm?id=41765.41801.

[SK07] D.P. Stormont and A. Kutiyanawala. Localization using triangulation in swarms of autonomous rescue robots. In *Safety, Security and Rescue Robotics, 2007. SSRR 2007. IEEE International Workshop on*, pages 1–6, Sept 2007.

[Ull10] David G. Ullman. *The Mechanical Design Process*. McGraw-Hill, 2010.

[ult] Ultimaker 2. URL: https://ultimaker.com/en/products [cited 25 September 2016].

[VDIa] VDI. Lasten-/pflichtenheft für den einsatz von automatisierungssystemen. VDI 3694, Verein Deutscher Ingenieure.

[VDIb] VDI. Vorgehensweise bei der erstellung von lasten-/pflichtenheft. VDI 2519 Blatt 1, Verein Deutscher Ingenieure.

[ZWX12] Qiang Zhan, Junqing Wang, and Xi Xi. Control system design and experiments of a quadrotor. In *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*, pages 1152–1157, Dec 2012.