Master Thesis

Design and Control of a Quad-Rotor: application to Autonomous Drone Refereeing

CST 2017.083

P.L.M. Rooijakkers

Technische Universiteit Eindhoven University of Technology

Eindhoven University of Technology
Department of Mechanical Engineering
Control Systems Technology

Supervisor:
dr. D.J. Guerreiro Tomé Antunes

Committee:
prof.dr.ir. W.P.M.H. Heemels
dr.ir. M.J.G. van de Molengraft
dr.ir. R. Tóth

Eindhoven, August 2017

Abstract

Most of the current commercially available and affordable multi-rotors have limited computational power, which limits their ability to perform complex autonomous tasks. Moreover, few of the available multi-rotors offer the possibility to integrate new components such as sensors. In this work, a computationally powerful multi-rotor platform meant to be used in the future as an autonomous drone referee (or in other applications requiring intensive computations) is designed. This design includes the mechanical and software design, as well as the control algorithms. At the heart of the proposed solution lies a modular on-board control-box which accommodates a Pixhawk flight-controller and an Intel Nuc i7 PC. While the multi-rotor is equipped with several sensors appropriate for drone refereeing, such as an 180 fish-eye camera and a RGB-Depth camera, it allows for easily adding different sensors in the future. In order to allow users to develop and deploy new control algorithms, a Simulink interface is designed for MAVLink, a widely spread communication protocol for multi-rotors. Furthermore, computer vision algorithms are implemented determining the position of the multi-rotor, on-board, with respect to a soccer-field. The position can also be obtained in other fields using, e.g., Aruco markers. In order to guide and control the multi-rotor along a trajectory, a new path-following control law with guaranteed convergence properties is proposed. Besides the convergence guarantees, one of the novelties in this solution is the use of a computationally efficient method based on a second order Taylor polynomial estimation in order to analytically determine the position dependent reference point on the path (required by the path following control algorithm), even for complex, intersecting, and overlapping paths. In the context of the drone refereeing application, an algorithm is developed based on FIFA guidelines and an experienced human referee. The algorithm is based on scale-space theory and relies on the idea of following the 'bubble' of active play. The mechanical, software, and control modules are integrated and the multi-rotor platform is successfully tested in practice, performing an autonomous flight.

Contents

1 Introduction

1.1 Background

In the last few years, unmanned aerial vehicles (UAV's), or drones, have earned a more and more prominent role in society and this trend is expected to continue. In fact, in a recent article published by Gartner [1] the perceptual smart machine age is announced, and UAVs will play an important role in it. This article defines the Gartner's hype cycle of emerging technologies which is shown in Figure 1. This graph shows that the popularity of UAVs is rising. On the other hand, this graph also indicates that after a peak of inflated popularity, often based on some successful stories, a period of disillusion can occur. However, ultimately the platform will be mainstream and will occupy its own niche in the market. Based on this article, a positive future for the (commercial) UAVs can be expected.

Many industries have already embraced the functionalities of UAVs. Applications of UAVs can be found for example in the video industry (for shooting videos from appealing perspectives), agriculture (in order to monitor the growth of crops), sports (drone-races are becoming more and more popular), the housing market (where UAV's are used to capture promotion material), civil industry (in order to carry out inspections), and the transportation industry (where potentially UAVs can be used for package delivery). Besides end-user applications, the multi-rotor vehicle is an attractive platform for testing new control strategies among researchers. In technical education, multi-rotors are often used as an attractive learning and experimental platform. Noteworthy is, for example, the work of D' Andrea and his team at ETH. In their work, drones perform complex autonomous tasks such as ball juggling [2], and cooperate in order to build structures [3]. Also the work of Vijay Kumar and his team at the University of Pennsylvania has shown the capabilities of drones flying in formation [4].

Each application has its own specifications and therefore UAVs arise in many different forms. In Figure 2 a range of various UAV types is shown. Figure 2.A shows a fixed wing plane and Figure 2.B shows an unmanned helicopter. A multi-rotor is shown in Figure 2.C. Because of its four rotors, this multi-rotor is also denoted by quad-copter or quad-rotor. Since this configuration is so popular, the word 'drone' often refers to this type of UAV. In Figure 2.D an unmanned vertical take off and land (VTOL) vehicle is shown (Amazon patented [5], prototype of a package delivery drone.). This type of vehicle combines the advantages of the flexibility of a multi-rotor with the energy efficiency of a fixed-wing plane [6]. In Figure 2.E the biologically inspired 'Air Penguin' [7] is shown, which is basically a balloon or airship. This type of (conceptual) UAV uses fins or rotors [8] in order to change speed and direction. In Figure 2.F the Robird [9] is shown which is a remotely controlled ornithopter using wings in order to fly.

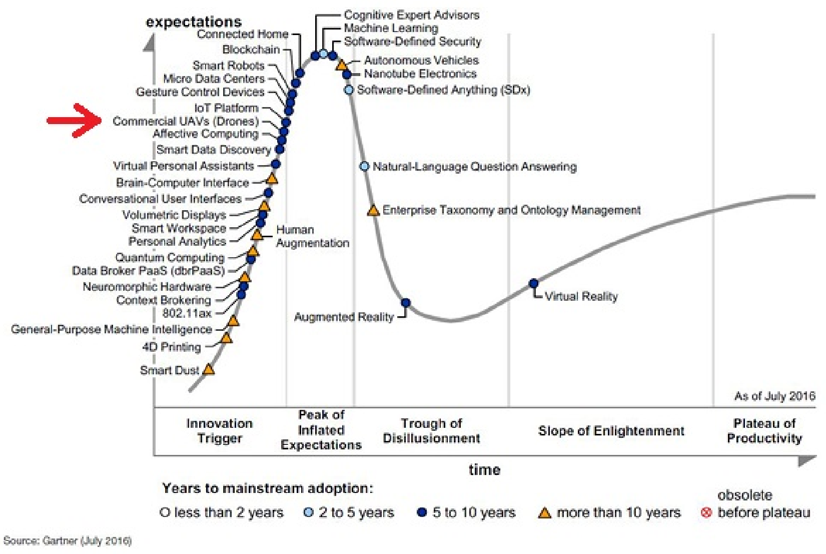


Figure 1: Gartner's Hype Cycle for Emerging Technologies July 2016. Shown is that Commercial UAVs (Drones) are entering soon the Peak of Inflated Expectations.

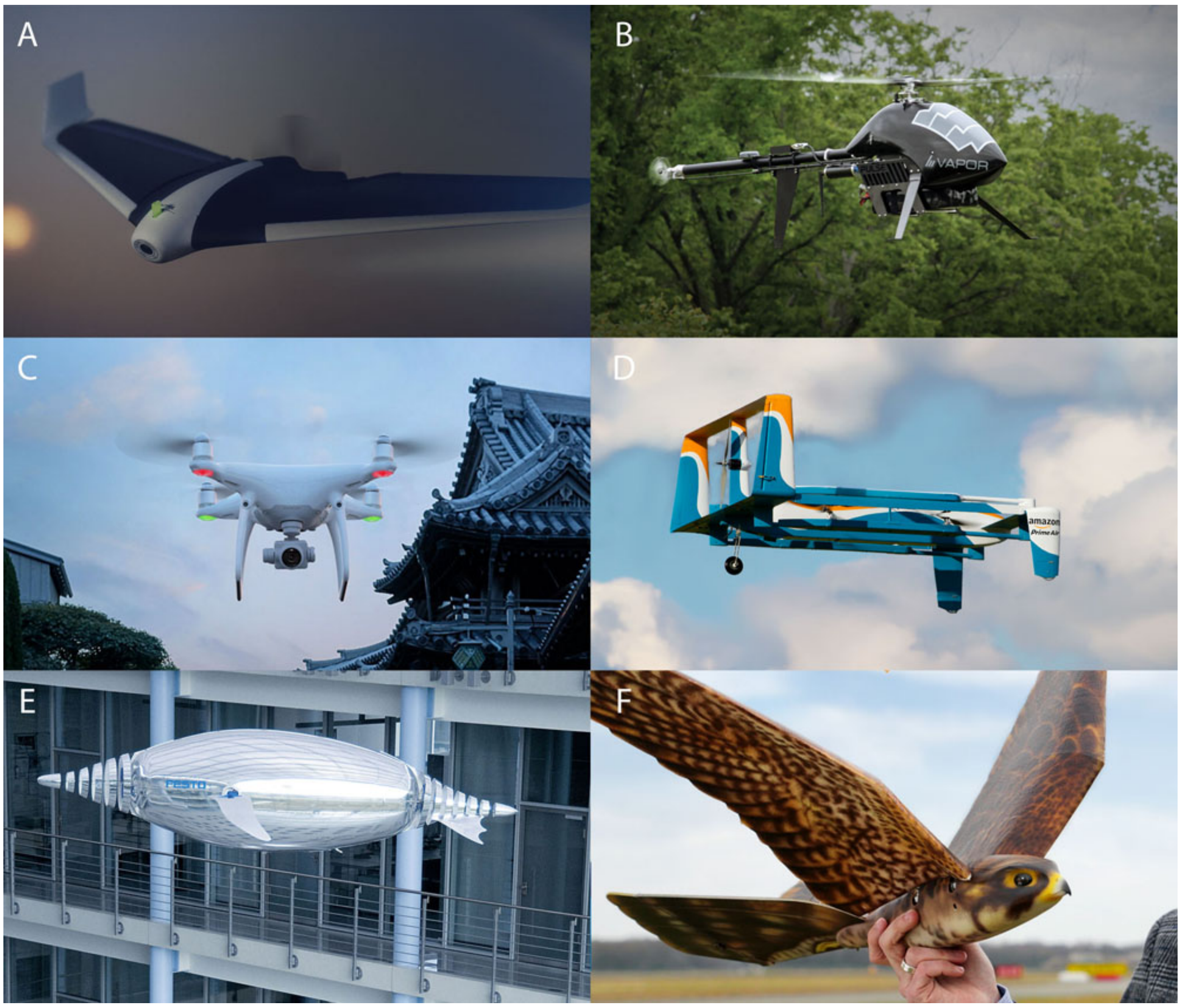


Figure 2: A selection of UAV types. A: Parrot Disco FPV, B: Pulse Aerospace HeliSynth, C: DJI Phantom 4 Pro, D: Amazon Prime Air, E: Festo AirPenguin, F: Clear Flight Solutions Robird

In this thesis, the focus is on multi-rotors. Multi-rotors are in general versatile and cost-effective. An important advantage of the multi-rotor is its capability of hovering at a fixed location, and its relative small form factor allowing it to fly in indoor environments. The focus in this work is especially on multi-rotors that can be used for academic purposes, i.e., the user is able to deploy their own control algorithms on the multi-rotor. The current commercially available and affordable multi-rotors that are qualified for research, such as the Parrot AR.Drone 2.0 [10][11] and the Crazyflie 2.0 [12], have often limited computationally power. This limitation was for example experienced in the PDEng project at TU/e where a prototype for an autonomous drone referee system is developed [13][14]. For autonomous flights, often significant computational on-board power is required in order to execute for example computer vision algorithms. An alternative solution is to perform the computer vision algorithms off-board on a remote system. However, wirelessly sending video at a high frame-rate leads to large delays as noticed in the context of the PDEng project. Another concern is that the current commercially available multi-rotors often require platform dependent source-code and rather complex build procedures. Moreover, new control algorithms are often first simulated in Matlab Simulink. One of the pitfalls is that promising control algorithms developed and simulated on a regular PC appear to run much slower or not run at all on-board of the multi-rotor. In other academic settings the control of a multi-rotor often (heavily) relies on an external, high performance, indoor positioning system, providing an accurate, nearly real-time, high frequency, state-estimation of the multi-rotor. Depending on how autonomy is defined, it is debatable to what extent the multi-rotor, in this type of setup, is actually performing an autonomous flight, since state estimation is an essential part of the complete control loop.

1.2 Contents of the thesis

The limitations of current multi-rotor solutions mentioned above, the experience from previous projects, and the requirements stated in previous work motivated the main goal of this work, which can be stated as follows:

> *Design of a modular unmanned air vehicle prototype platform on which it is possible to test and verify control algorithms for autonomic flights in an academic environment, run computationally intensive algorithms, and which can easily be extended in the future with state-of-the-art sensors. The selected use case is the multi-rotor being able to perform the task of a referee during a (robot) soccer game.*

In this thesis a computational powerful multi-rotor platform meeting these requirements is developed which is capable of performing an autonomous flight. This report describes all the engineering steps from defining the requirements up to a working platform. Based on the system requirements and a study on 20 commercially available multi-rotors, 6 flight-controllers, and 6 potentially companion-computers the hardware components for a new multi-rotor are selected. The design and production of a carbon frame is described. This frame keeps all system components together, such as the propulsion system, sensors, and a modular control-box. The control-box accommodates a Pixhawk flight-controller, and an Intel Nuc i7 PC, extending the computational power of the flight controller and basically transforms the multi-rotor into a 'flying laptop'. With a normal PC, on-board, it is possible to run all regular PC software on the multi-rotor. Furthermore, the multi-rotor is equipped with an optical-flow sensor, ultrasonic height sensor, an 4K 180° fish-eye camera, and a RGB-Depth camera. Due to the modular design it is possible to replace, remove, or extend the capabilities of the system fairly easily.

Matlab [15] is the standard development tool for experiments during the systems-and-control courses at TU/e [16], [17]. Often, TU/e students develop and simulate new control algorithms in Matlab Simulink. In this work, the software essentials are developed in order to deploy the Simulink-based control algorithms in real-time on the multi-rotor. For the communication with a Pixhawk flight-controller the widely spread MAVLink protocol is used. Via MAVLink, state-updates and control instructions can be communicated between a flight controller and a companion-computer or external ground-station. An appropriate implementation of the MAVLink protocol is currently not available for Simulink. In this work a complete, bidirectional, configurable, real-time implementation of the MAVLink protocol for Matlab and Matlab Simulink is developed. The software features a clear separation between protocol layer and transport layer allowing the MAVLink-Simulink interface to work also with different applications. The developed MAVLink-Simulink interface already found its way to a professional industrial drone company (Avular).

In order to guide a multi-rotor along a planned trajectory, path-following is considered. Differently from trajectory tracking, where the vehicle is required to track a time-parameterized trajectory specifying where the vehicle's should be at each time step, in path following the distance to the path is driven to zero while a velocity reference defined along the path is followed. This strategy has therefore soft, rather than hard, time requirements, typically enabling better behavior in the sense the position errors are smaller and smoother actuation is obtained. A new path-following control law with guaranteed convergence properties is proposed relying on model simplification and singular perturbation theory. In this work, the control objective, which is often expressed in the Frenet-Serret frame, is expressed directly in the inertial frame, providing some computational benefits. In the provided path-following strategy, the position of the multi-rotor determines the reference-position on the path, which intuitively corresponds to the point with the shortest distance to the vehicle, although this will be formally defined in Chapter 4. For complex or overlapping paths, an analytical solution to the problem of finding the closest point to the path often does not exist or is not unique. Furthermore, with traditional methods, it is not always guaranteed that it will provide a continuous reference along the path, i.e., based on a straight forward least distance policy it can occur that hopping between paths will occur when paths cross or are almost parallel. In this work a computationally efficient method is proposed using a local second order Taylor polynomial estimation in order to obtain an analytically solution while maintaining the continuity of the reference along the path. The proposed control law is extended to include the path depending heading of the multi-rotor, for which its stability is verified based on a Lyapunov convergence proof. The method is implemented in simulation.

Since the intended application for the platform is drone refereeing, an online motion planning algorithm for autonomous drone refereeing in soccer is proposed. Guidelines from FIFA and an experienced human referee are transformed to a computer algorithm that attempts to follow the 'bubble' of active play. The algorithm is based on biologically vision inspired scale-space theory. In order to test this method, a simulation environment is designed for the drone referee based on real-data originating from robot soccer.

Finally, the multi-rotor is tested in practice in order to perform a proof-of-concept. The proof-of-concept accounts to performing an autonomous flight. In this work, autonomy is understood as executing all the computations on board of the multi-rotor, i.e., no external position reference or control signals are received. To this end, a computer vision algorithm is developed that uses the lines on a soccer-field as reference. Although this method is tested in simulation, because of time considerations, another method based on the detection of Aruco markers is implemented in practice. The fact that the multi-rotor is able to successfully perform an autonomous flight, provides a solid evidence that the complete control chain of, sensing by computer vision, real-time control using Simulink, the communication using our MAVLink-Simulink interface, and the hardware actuation is functioning.

1.3 Contributions

The main contributions of the thesis are the following:

- A modular drone incorporating a companion powerful computer was designed and produced, accomplishing the goal of having a 'flying laptop' for research at TU/e. To the best of our knowledge there is no commercially available quadcopter which meets these requirements. This will enable the students at TU/e to test their simulated controllers and algorithms designed in their laptops directly in the drone, without computational concerns and avoiding the typically cumbersome path of deploying the code in a target. Besides this, the modularity of the drone can be exploited to easily add different state-of-the-art sensors for future research problems.

- A complete, bidirectional, real-time, MAVLink-Simulink interface is provided, in order to communicate with the flight controller via the widely spread MAVLink protocol. This software, which can be used as an interface between Simulink and any drone supporting MAVLink (e.g. drones using the Pixhawk flight-control firmware, or ground control station) will be made available for the general public and already found its way to a professionally industrial drone company (Avular).

- A new path-following control law with guaranteed convergence properties is proposed. Besides the convergence guarantees, this solution introduces a novel method for keeping track of the vehicle dependent reference point on the path. This resorts to a second order Taylor polynomial estimation which copes with complex, intersecting, and overlapping paths.

- A simple, and somewhat conceptual, method was proposed to guide an autonomous drone referee during a robot soccer match. This method relies on scale-space theory and is based on experience of a human referee.

Besides these main contributions, the thesis also proposes other ideas which can also be seen as contributions. For example, in the Appendix it is proposed an open-source low-cost indoor positioning system using Raspberry Pi. Another example is the adaption of a Tech United algorithm for detecting the robots position in a soccer field, to work, at least in simulation, with a multi-rotor in 3D space. This report might be seen as a manual for the use, and eventually further development, of the platform as obtained in this work.

1.4 Organization of the report

The remainder of the report is organized as follows:

Chapter 2, Hardware Design of a Computationally Powerful Multi-Rotor Platform.
This chapter describes the hardware design of a computationally powerful multi-rotor. First, the system requirements are stated, and the general layout of a multi-rotor system is described. Based on a study on commercially available multi-rotors, flight-controllers, and potentially companion components, a hardware selection is made. The design of a modular control box is described, and the components for the propulsion system are selected. Several conceptual frame designs are discussed. Finally the design and

production of a first prototype frame and an improved frame design are discussed.

Chapter 3, Interface and Software Design for a Multi-Rotor Platform.
This section describes the software essentials that allows future users to deploy new control algorithm on the multi-rotor. In order to enable the communication between a Matlab Simulink model and the flight-controller in real-time using the standardized MAVLink protocol, a MAVLink-Simulink interface is designed. In the second part of this chapter the computer vision algorithms that estimate the position of the multi-rotor on-board are discussed. Finally, in order to perform a proof-of-concept, which is an autonomous flight, the integration of the separate software components is discussed.

Chapter 4, Path Following Control for a Multi-Rotor.
In this chapter path-following control is discussed. First, a cascaded controlled system structure allowing system reduction by the separation of fast and slow dynamics is discussed. A control law for path-following is proposed, derived from the Frenet-Serret frame, however, directly expressed in the inertial frame. The stability of the controlled system is checked using a convergence proof. In order to follow complex and overlapping paths, a computationally efficient method based on a local second order Taylor polynomial estimation is proposed which calculates the point with shortest distance on the path. Finally, a control law for guiding the heading of the multi-rotor along the path is proposed. The method is verified by numerical simulations.

Chapter 5, Drone Referee System.
In this chapter an online motion planning algorithm for soccer drone referee is proposed, which potentially can be used in an autonomous referee system. Furthermore the design of a simulation environment for drone referee is described.

Chapter 6, Experimental Results.
This chapter describes the experimental results of the proof-of-concept, which is performing an autonomous flight.

Chapter 7, Conclusions, Remarks and Future work.
This final chapter summarizes the conclusions, remarks and future work.

2 Hardware Design of a Computationally Powerful Multi-Rotor Platform

2.1 Introduction

In this Section we will describe the design of a computationally powerful multi-rotor that is able to perform computationally complex tasks. A design method is followed consisting of three phases: analysis, synthesis, and evaluation (ASE). In general, the design of an engineering product is an iterative process. However, here we will discuss each phase only once in a gradually overlapping form. Section 2.2 starts by stating the system requirements, followed by a description of the general structure of a multi-rotor in Section 2.3. Section 2.4 summarizes a market search which analyzed 20 commercially available multi-rotors, 6 flight-controllers, 6 potential companion-computers, and several sensors. Section 2.5 describes the design of a new multi-rotor, based on the market research. Since the multi-rotor is produced in-house also the production process is briefly discussed. In Section 2.5.1 the component selection is described. Special attention is payed to the selection of an Intel Nuc PC as companion-computer which will transform our multi-rotor into a 'flying laptop'. A fish-eye camera and RGB-D camera are selected and the connection to the visual-fields of birds is discussed. In Section 2.5.2 the modular design of a control-box is discussed, and the design of a printed circuit board (PCB) for the on-board power conditioning is described. Furthermore, the implementation of a remote emergency stop and an on-board safety switch are discussed. In Section 2.5.3 the experimental results of different propeller configurations are discussed, and the components for the propulsion system are selected. In Section 2.5.4 several concepts for the design of a frame are proposed. Based on one of the concepts, the final design, and the production of the multi-rotor is discussed in Section 2.5.5. An improved design of the frame is described in Section 2.5.5. Finally in Section 2.6 a summary is given.

2.2 System Requirements

We start by recalling one of the goals of this work, already mentioned in Chapter 1:

> *Design of a modular unmanned air vehicle prototype platform on which it is possible to test and verify control algorithms for autonomic flights in an academic environment, run computationally intensive algorithms, and which can easily be extended in the future with state-of-the-art sensors. The selected use case is the multi-rotor being able to perform the task of a referee during a (robot) soccer game.*

The system requirements that emerge from the above description can be divided in a more detailed set of functionalities as it is listed next.

The vehicle should be an unmanned air vehicle (UAV)

1.a **Propulsion**. The vehicle should be able to lift its own weight from the ground and be able to change its position in three dimensions in order to perform a flight.

1.b **Time of flight**. The vehicle should be able to fly for a considerable amount of time, preferably half of the duration of a robot soccer match (approximately 15 minutes).

1.c **Producible**, The vehicle must be producible within the constrains of the available production techniques and budget.

1.d **Maintainable**. It should be possible to repair the vehicle.

The vehicle should be an academic prototype platform

2.a **Flexibility**. Academic prototyping requires that it should be possible to expand, replace, or reduce system functionalities on a regular base. Flexibility of the system can for example be achieved by a modular-system approach with a clear separation of functionalities.

2.b **Accessibility**. In an academic environment multiple users with different backgrounds and levels of knowledge may use the platform. The platform should be accessible for a large group of users.

2.c **Safety**. On the platform unverified controllers will be tested, this can result in potentially unsafe situations caused by unpredictable system behavior. Some level of safety must be achieved in order to reduce the risk on injuring the user or damaging the environment.

The platform should be able to test and verify control algorithms for autonomous flights:

3.a **Control**. It must be possible to control the physical components that propel the system.

3.b **Sensing**. Sufficient sensors should be available and accessible to determine the systems state and optionally the vehicles environment onboard.

3.c **Processing unit**. Modern control algorithms require a digital processing unit to be executed on. Since this is a prototype platform, it must be easy for the user to change the program/algorithm. In addition, significant computational power should be available. Here we define significant as comparable with a TU/e laptop [18], because the platform is mainly meant to run algorithms which students at TU/e will develop and test in their laptops.

The vehicle should fulfill the function of a referee during a (robot) soccer game.

4.a **Communication**. It should be possible to wirelessly communicate with the vehicle when the vehicle is part of a complete autonomous referee system [14].

4.b **Sensors**. Sufficient sensors with sufficient update rates should be available to fulfill the role of referee.

4.c **Dimensions**. The vehicle should fit in the environment of a (robot) soccer-field.

2.3 General structure of a multi-rotor

In general a multi-rotor consists of several components. Each component has its own function and only when all functions work seamlessly together the vehicle will be able to perform a flight. In Figure 3 an abstract diagram of the common components used in a multi-rotor is shown.

From bottom up, the often rigid-body vehicle-frame holds all components together, brings the required stiffness, and protects the components against crashes or weather conditions. The propulsion system of the vehicle, indicated in the bottom of the diagram, is critical. The rotors, in general placed in a single plane, generate thrust and drag performing the forces and torques required to lift and control the vehicle in space. Often brush-less three-phase electric-motors drive the rotors, with or without intermediate mechanical transmission. The electric speed controllers (ESC's) regulate the rotational speed of the motor according to a given reference. This is done by modulating the high current electric motor power. The ESC's are in general directly powered by the battery. The micro-controllers within the ESC's can often be programed for optimal acceleration and braking of the blades. The appropriate reference speed is calculated by the flight-controller. In traditional systems the ESC reference speed is given in the form of a pulse width modulation (PWM) signal. The flight-controller compares the current state of the vehicle with its prescribed target state and acts accordingly in order to bring the system to its target state. The flight-controller bases its current state estimate on sensor measurements. The main sensor for measuring the vehicles attitude, roll and pitch, is often an inertial measurement unit (IMU) sensor, typically consisting of a (micro-electromechanical system (MEMS)) gyroscope and an accelerometer. On the other hand, for measuring the vehicle's position, a wide range of different sensors can be used. The most common sensors for measuring the multi-rotors position or velocity in space are: cameras, ultrasonic sensors, laser sensors, GPS, air-pressure sensors, air-speed sensors, and compass sensors. For autonomous flights, the flight-controller forms the heart of the system. Besides stabilizing and positioning the vehicle, the flight-controller communicates with its environment. The communication can, for example, account to receiving instructions or target states from a ground-station or companion-computer. Most commercially available multi-rotors can be remotely controlled via a radio control (RC) transmitter receiver link. In turn, the flight-controller can send messages back to the remote operator, including, for example, the battery state or errors. Furthermore, on-board of a multi-rotor there is often room for a payload. The payload can in principle be anything such as cargo, a camera, or a companion-computer. A companion-computer is often used to extend the computational capabilities of the flight-controller. Often the flight-controller performs the time critical control tasks of the multi-rotor such as stabilization, whereas the companion computer performs the decision making, the more high-level tasks, the computationally intensive tasks, and instructs the flight-controller.

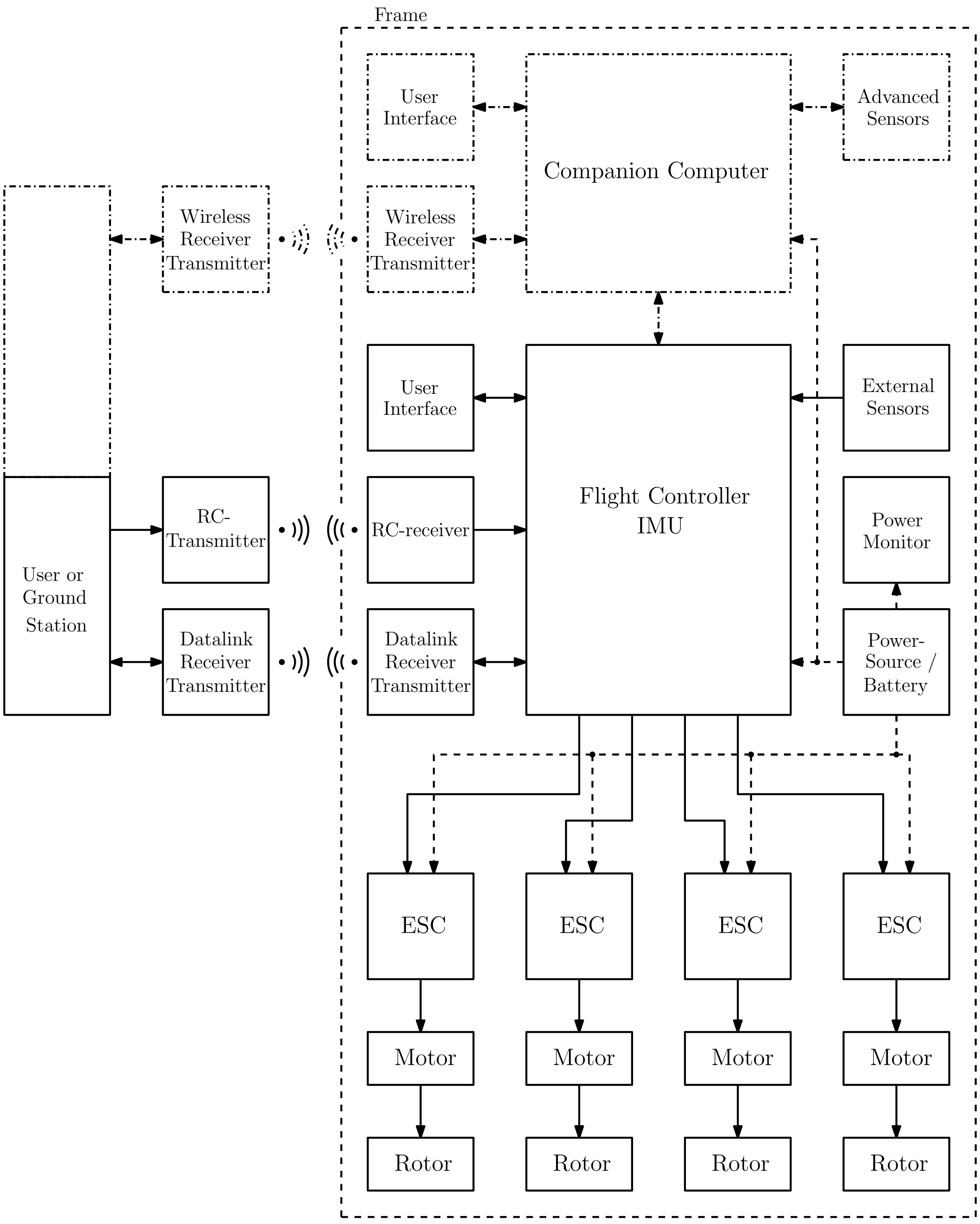


Figure 3: Schematic sketch of the general structure of a multi-rotor. The companion-computer is often not present in a commercially available multi-rotor platform and therefore indicated by dashed lines.

2.4 Market Research

In the last decade a large variety of commercial available multi-rotors have been introduced in the market. In this section we will summarize the analysis of 20 commercially available commercially available multi-rotors, 6 flight-controllers, 6 potential companion-computers, and a set of commercially available sensors. The described data forms the basis for the component selection in Section 2.5.1.

2.4.1 Commercially available multi-rotors

In this section the results of a study on the properties of 20 commercially available multi-rotors are described. The system properties are obtained from the data-sheets of the multi-rotor producers. In Table A.1, in the appendix, the 20 commercially available multi-rotors are listed. Below, the frame-material, frame-shape, energy-system, propulsion-system, and a normalized ratio for energy efficiency are summarized.

Frame material and shape. In general the rotors of a multi-rotor are symmetrically placed around the center of the multi-rotor. Often the rotors are supported by arms, the arms are connected in the center resembling an X or a star form. Except for the rotors, all other components, such as batteries, controllers, and payload, are placed at the center of the multi-rotor. Some multi-rotors have blade protectors, protecting the blades and environment during collision. The multi-rotors with a vehicle mass $m_v > 1.5kg$ often have a frame made of carbon fiber reinforced polymer (CFRP) tubes, whereas the lighter multi-rotors have a frame made of plastic, 2D CFRP plates, or even the PCB itself.

Energy and propulsion system Design of a propulsion system for a multi-rotor requires knowledge about the relations between, power, thrust, torque, vehicle weight, and energy storage. A propulsion system of a multi-rotor consist, in general, of several rotors driven by electro motors powered by a battery. The properties such as thrust, torque, and energy consumption of a rotor can be, approximately, calculated from first principle physics. Examples of such calculations can be found in the field of aerospace applications. Methods used are for example blade element theory or numerical simulations. However, these calculations require often specific knowledge about parameters and coefficients. The intention of this work is to use commercially available components, from which detailed technical aerodynamically parameters are often not available. Therefore, the theoretical approximation of rotor dynamics in this work is limited to the energy equations, as are described, in detail, in [19]. Figure 4 illustrates, a simplified representation of the air-flow around the rotor. As is derived in [19], based on momentum theory, the theoretical power P_h required for hovering is given by the relations

$$T_h = 2\rho A v^2 = m_v g \quad (1)$$

$$P_h = T_h v = 2\rho A v^3 \quad (2)$$

where T_h represents the vertical thrust required to hover, m_v the vehicle mass, g the gravitationally constant, ρ the air density, A the rotor disk area, and v represents the rotor induced velocity. From this relation it can be concluded that it is interesting to have a low induced-velocity and a large rotor disk area. This might be the reason why the flight-time world-record multi-rotor has relative large propellers. As can be observed from the Table A.1, in the appendix, the majority of the multi-rotors uses lithium polymer (LiPo) batteries. LiPo batteries are based on lithium-ion technology [20], the electrolyte is a polymer. The advantage of a LiPo battery is that it has a relative high energy density and is able to deliver a relative large electrical-current. The energy E_b in a LiPo battery pack can be estimated by the relation

$$E_b = 3.7 s_{cell} Q_b \quad (3)$$

where 3.7 is the nominal voltage per battery cell, s_{cell} represents the number of cells in the battery pack, and Q_b the battery capacity in Ah, such that E_b is expressed in Wh.

Hover lift efficiency In order to compare the multi-rotors with each-other, the hover lift efficiency can be calculated. This efficiency represents the amount of mass a multi-rotor is able to lift, during hover, normalized by the energy consumption. The hover lift efficiency is given by

$$\eta_h = 1000 \frac{m_v t_f}{E_b} \quad (4)$$

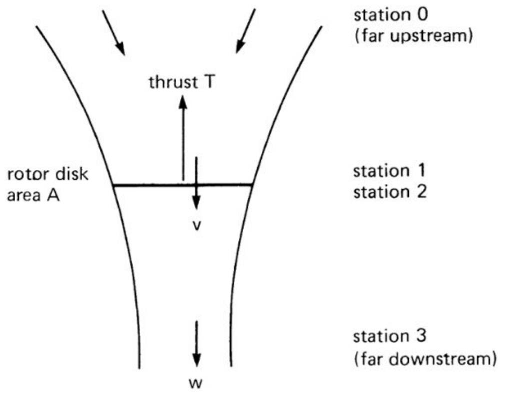


Figure 4: Schematic representation of the airflow around a rotor, supporting the explanation of the, on momentum theory based, flow model for hover. As derived in [19], the area of the wake far downstream is assumed $\frac{1}{2}A$ and $w = 2v$.

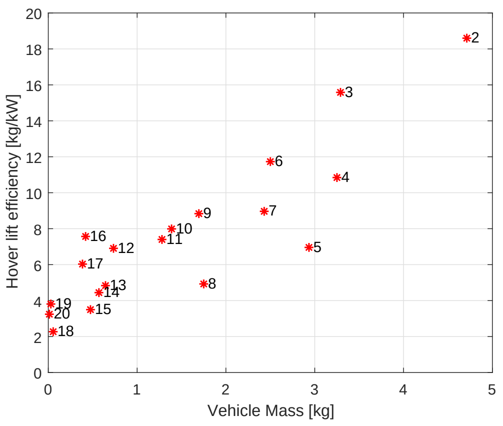


Figure 5: Hover lift efficiency of commercially available multi-rotors with a mass $m_v < 5kg$. Indicated in the graph are the id-numbers as defined in Table A.1 of the appendix. For example, id. 2 is the world-record multi-rotor.

where t_f is the flight time of the multi-rotor in hours, such that the hover lift efficiency η_h is expressed in $\frac{kg}{kW}$. The numerical results for the 20 commercially available multi-rotors are graphically shown in Figure 5. As it can be observed, more heavier multi-rotors tend to be energy more efficient.

2.4.2 Commercially available flight-controllers

In general the time-critical control task within a multi-rotor are executed by an on-board flight-controller. Nowadays, a wide range of flight-controllers are commercially available. Since our intention is to have an option to deploy our own control algorithms on the flight-controller, an open-source flight-controller is preferred. In Table A.5, in the appendix, a set of popular open-source flight-controllers are given together with some of their properties.

The **Ardupilot APM 2.8** is a 8-bit flight-controller based on the ATmega 2560 processor, the platform is originally based on the Arduino Mega 2560. The **Naze32** and **CC3D** are very basic small 32-bit flight-controllers, mostly meant for racing drones. The more advanced **Pixhawk** is an open-hardware and open-software platform. It is a popular platform with a large active (academic) community. The Pixhawk and the newer Pixhawk 2.1 are equipped with numerous connectivity options for sensors and external communication. The Pixhawk runs the NuttX real-time operating system (RTOS). A considerably more advanced platform is the **Qualcomm Snapdragon Flight** with computer vision capabilities. It has, with its on-board optical-flow sensor, 4K front camera, and on-board Wifi, already most of the features desired from a companion-computer. However, the price of the board is similar to the price of an advanced companion-computer.

2.4.3 Commercially available companion-computers

For the more advanced (high level) control, often, an on-board companion-computer is required. This companion-computer extends the capabilities of the low-level flight-controller. On a companion-computer it is possible to deploy, for example, computer vision algorithms [21]. The allowable payload mass on a multi-rotor is often limited, therefore the mass of the companion computer is an important selection criteria. In Appendix A.6 a set of lightweight potential companion-computers is listed.

The **Raspberry Pi** is a well known single-board computer which is often used for development and prototyping. The Raspberry Pi has multiple interface options. The platform is widely supported. For example, Matlab Simulink models can be deployed and run on the device. The **Odroid** platforms have similar or more computational power compared to the Raspberry Pi. Also the interface options are comparable to the options available on the Raspberry Pi. The **Intel Aero Compute Board** is a single-board computer special developed to be used in multi-rotors. The board is part of Intel's Aero, ready to fly (RTF), multi-rotor platform. This development platform is already equipped with a Real-Sense stereo vision camera system. The **Intel Nuc** is a full desktop PC in a small form factor. Since the Intel Nuc is a small desktop PC, it performs similar to a laptop or desktop. The bigest advantage of the Intel Nuc is that all the applications that run on a regular PC will also run on the Intel Nuc. By removing all unnecessary components, an enclosure, in our case, a mass of only 190 gram is obtained (originally 600 gram). The **Nvidea Jetson TX1** is a credited-card sized single-board computer. The main advantage of this board is the high performing graphic processing unit (GPU). The GPU contains 256 CUDA cores which makes advanced parallel processing possible. Due to the ARM architecture, not all applications can run on the Jetson TX1 as it is the case for the Intel Nuc. In order to take advantage of the GPU the user has to use special libraries or needs to learn to program CUDA.

2.4.4 Commercially available sensors and positioning systems

In systems and control applications, sensing is at least as important as actuating. In autonomous systems, sensors are the essential instruments providing the current state of the system. In Appendix A.7, it is indicated which states can be measured by a specific sensor.

Often, in indoor environments, an indoor position system (IPS) is used to provide the absolute position of the multi-rotor with respect to the world. Besides providing the real-time position of the multi-rotor, which can be used for feedback-control, an IPS can be used as bench-mark or ground truth for evaluating control algorithms. Popular commercially available systems are the optical tracking system, such as a Vicon and Optitrack, and the ultra wide band (UWB) systems such as DecaWave. The optical tracking systems are more accurate and have a higher sampling-rate than the currently available ultra wide band (UWB) systems. However, a professional optical tracking system is, in general, much more expensive than a UWB system.

For proper control it is essential to have a good estimate of the current system-state. Indoor positioning systems (IPS) are designed to provide this system's state. Another function of an IPS is to provide a benchmark or ground-truth in order to evaluate control algorithms. The current, reliable, commercially available indoor positioning systems are expensive. For the control of multi-rotors, currently, no IPS is available in the lab. The current system, a top camera tracking the horizontal position and heading of the multi-rotor, is not reliable. According to the experience obtained by [14], the system suffers from signal drops, a coarse accuracy, and signal lag. In Appendix C we propose 3 alternative, relative cheap, concepts for a future IPS. Although all 3 methods are only conceptually tested in simulation and in a proof-of-principle experimental setup they might be of inspiration for further work.

2.5 Design of a Multi-Rotor

After the studies presented in the previous sections, this section addresses the synthesis of all system components. Based on the system-requirements, in Subsection 2.5.1 the system components are selected. In Subsection 2.5.2 a modular design concept for the control-box is introduced. Subsection 2.5.3 describes the selection of the propulsion system, based on some experimental results with a mock-up multi-rotor. In Subsection 2.5.4, several conceptual designs are discussed. Section 2.5.5 describes the design and production process of a first full-scale prototype, and Section 2.5.6 describes the design of an improved second prototype.

2.5.1 Selection of system components

In this section the commercially available components are selected. The selection is based on the criteria originating from the system requirements stated in Section 2.2. Because the selection of the propulsion system is mainly based on the vehicle mass m_v, first the components that count as payload are selected in order to determine the mass. In this work, the payload of the multi-rotor are the control electronics and the sensors. In the sequel, the companion-computer, the flight-controller and sensors are detailed separately.

Companion-computer Based on the accessibility and processing requirements a companion-computer can be selected from the table in Appendix A.6. The processing capacity of the companion-computer should be preferably comparable to a TU/e laptop. This narrows down the options to the Intel Aero Compute board, the Intel Nuc and the Nvidia Jetson TX1. The Intel Nuc is a normal PC and therefore most convenient to most users. It is possible to install different operating systems on the Intel Nuc such as Linux Ubuntu, and Windows and it is therefore a flexible platform. In addition, all programs that run on a normal laptop will also run on the Intel Nuc. On the other hand, the other two systems are equipped with dedicated hardware for image processing, which can be an advantage. However the flexibility, interchangeability of programs, and the flat learning curve of using the Intel Nuc are considered to be more important advantages. The Intel Nuc is therefore selected as companion-computer for this multi-rotor platform.

Flight-controller The control, flexibility and sensing requirements can be used as criteria for the selection of a flight-controller from the table in Appendix A.5. It is important that different types of sensors can be attached to the flight-controller in order to allow different (future) configurations. Furthermore, it is essential that a form of communication between the companion-computer and the flight-controller is possible. This narrows down the selection to the Pixhawk and the Qualcomm Snapdragon Flight. The Pixhawk is a very flexible flight-controller with a large and active (academic) community which makes the platform very accessible. Furthermore the Qualcomm Snapdragon Flight board is significantly more expensive than the Pixhawk which is a consideration according the budget. Therefore the Pixhawk is selected.

Sensors The sensors are required to measure the state of the system. The chosen flight-controller, the Pixhawk, has a built-in IMU which is used to estimate the attitude of the multi-rotor. Besides stability, the system requirements state that the system must be able to perform an autonomous flight. This requires information about the position of the multi-rotor with respect to the environment. In addition, also the applications running on the multi-rotor require often information about the environment. With computer-vision algorithms it is for example possible to obtain information about the environment, in a contact less sense. In general, using computer-vision algorithms and cameras is considered as a challenging task. However in nature, it seems that vision is widely spread without problems. For example a bird is able to fly autonomously. Before selecting a camera, it might be wise to study what, for example, birds see and how they use vision. At least a bird is able to fly autonomously.

Biological vision Article [22] describes the visual fields and their function in birds. In Figure 6, an example of the visual field-of-view in birds is shown. The variations of visual fields in birds can be roughly divided into three categories: a group with a binocular-field with a width of $10-30deg$ of overlap, a group

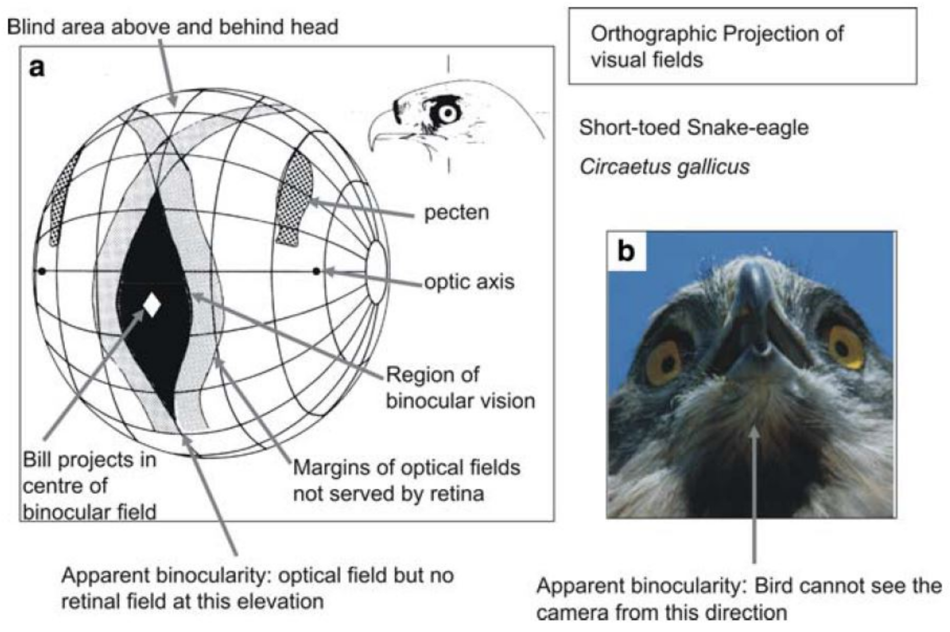

Blind area above and behind head
a
Orthographic Projection of visual fields
Short-toed Snake-eagle
Circaetus gallicus
pecten
optic axis
Region of binocular vision
Bill projects in centre of binocular field
Margins of optical fields not served by retina
Apparent binocularity: optical field but no retinal field at this elevation
b
Apparent binocularity: Bird cannot see the camera from this direction

Figure 6: Example of orthographic projection of visual fields in birds [22]. Birds have in general a wide visual field of view.

with no or $< 10deg$ binocular-field width, and a group with a wide binocular-field. The binocular-field is the field where the visual fields of both eyes overlap. A binocular-field results in stereopsis and thus in the perception of depth. This can be useful during foraging, or provisioning chicks. However, it is doubtful whether stereoscopic view is used during for example locomotion in wood-stock, most likely, using stereopsis for locomotion is too slow. Instead, it is expected that motion-parallax (objects closer seem to move faster than objects far away) is an important source of depth perception. The presence of a binocular field is for this purpose most likely a consequence of obtaining a balanced optical flow field. In addition, it is doubtful whether, for example, owls, with a relative wide binocular field use stereopsis for hunting. Possibly, their eyes are placed more to the front to make room for their advanced ears. On the other hand, for example, herons have a large vertical binocular field, in order to observe a large area while standing still and waiting for preys to pass by. The wood-cock is a bird with comprehensive visual coverage of the hemisphere [23]. This bird has (nearly) 360 deg view, and has binocular-fields in front, above, and behind its head, possibly to observe, for example, approaching predators from all around without having to move its head. Furthermore, in order to avoid sun-rays from directly entering the eye, birds are often equipped with advanced eyebrows.

Fish-eye camera Compared to the visual field of view in birds, regular cameras have, with an angle of around 70°, a rather narrow field of view. In addition, one of the experimental conclusions in the Autonomous Referee System [14], was that it is difficult to track a fast moving ball with a narrow angle down-facing camera. In order to solve this problem, in this work, a fish-eye camera with an field of view of 180°is selected. In Figure 7 an example image captured with the fish-eye camera is shown, and in the table in Appendix A.8 the technical specifications are given. A effect of a wide field of view might be that the ball can be tracked for a larger period of time without actively having to control the multi-rotor. This then, as was discussed in the autonomous referee system project, might eventually lead to a multi-rotor doing nothing more than necessary, i.e., making it lazy, as is for example described in [24].

RGB-Depth camera In order to obtain a perception of depth, in addition to a wide field of view, a RGB-D camera is selected. Besides a regular color image a RGB-D camera has a depth channel. In this work the light-weight Intel RealSense R200 RGB-D camera module is selected. The camera provides a color-image, stereoscopic ir-images, and a depth image. The depth map is computed from the disparity between the two ir-cameras. An embedded ir-projector helps with obtaining a depth map in indoor short-range environments. For objects at a larger distance and in outdoor environments only disparity is used. From experiment, it is concluded that only objects with a sufficient rough surface structure are detected at a large distance. In Figure 8 an example image is shown and the technical specifications are given in the table in Appendix A.9.

Optical-flow and height sensor In order to measure the fly height and the horizontal velocity, the PX4Flow [25] optical flow unit is selected. This sensor, especially designed for the Pixhawk, uses a high-

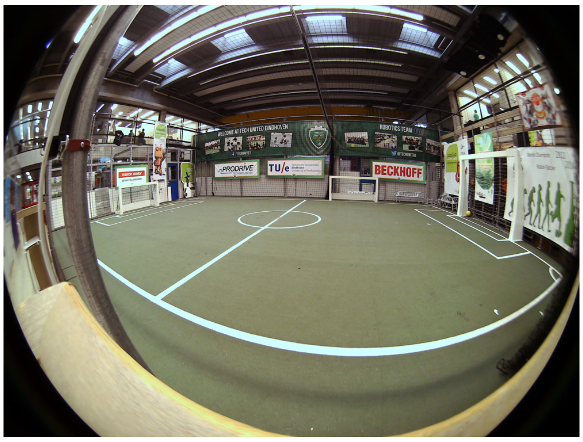


Figure 7: Example image of the Tech United soccer-field captured with the ELP-USB8MP02G-L180 fish-eye camera.

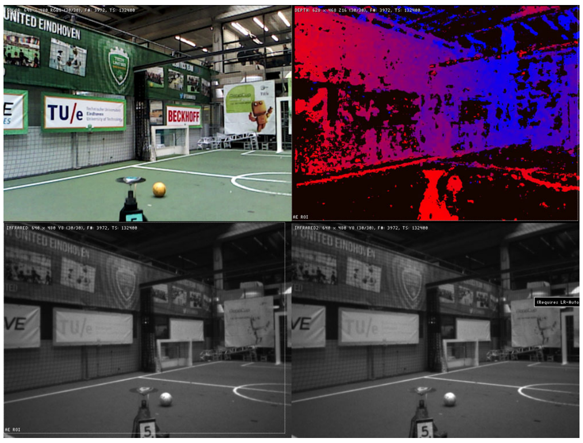


Figure 8: Example image of the Tech United soccer-field captured with the Intel RealSense R200 RGB-D camera.

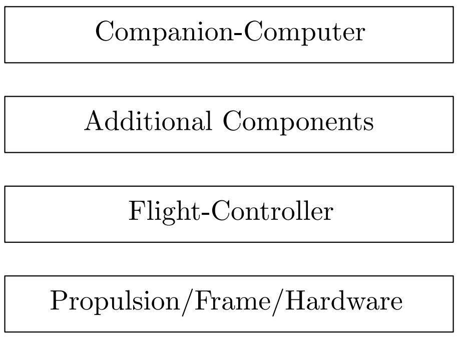


Figure 9: Schematic representation of the modular design of the multi-rotor by structuring the components in layers.

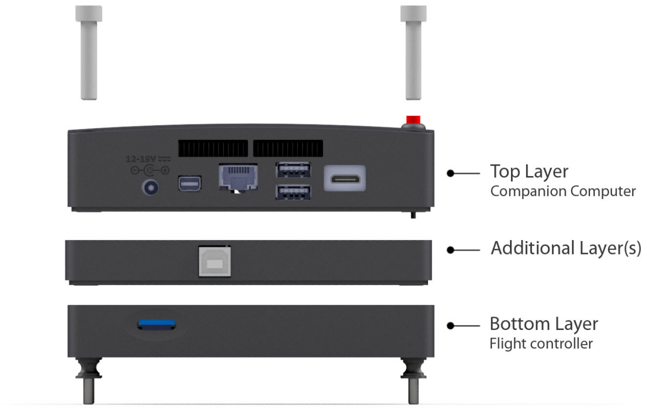


Figure 10: Realization of the layered structure of the control-box. Additional layers can be added without having to change other layers.

speed down-facing camera to measure the velocity/displacement of the multi-rotor with respect to the ground. With an ultrasonic sensor the distance to the ground is measured.

Indoor Positioning System (IPS) An indoor position system (IPS) can be used for determining the position of a multi-rotor. Besides that the obtained position can be used for real-time feedback control, the system can also operate as benchmark or ground-truth. Currently, no accurate and reliable IPS is available in the lab. With the current top-camera eventually the position of the multi-rotor in the horizontal plane can be measured. In this case, LED markers mounted on the vehicle are detected from which te position and heading of the multi-rotor can be computed as is described in [26]. However, according to [14], the current obtained data is erroneous, with a signal drop rate up to 25%. In order to reliably measure the absolute position and heading of the multi-rotor, an IPS based on Ultra Wideband (UWB) technology was ordered. However, at the time of writing the system is unfortunately not operational nor delivered. Therefore another method, based on computer-vision, is used as will be discussed in Section 3.4.

2.5.2 Modular design of a control-box

In this section the concept of a modular control box is introduced. In Figure 9 the modular design structure is schematically visualized. In this layered structure, functionalities and hardware that belong together are organized in a single layer. The strength of this approach is that modules/layers can be replaced while the rest of the system can remain intact. As a consequence the interfaces between the layers must follow a form of standardization. For example, between the propulsion system and the flight-controller we use the standard pwm interface, between the flight-controller and the companion-computer we use USB. In principle, it is even possible to add (unforeseen) *additional components* in the future without having to change the rest of the system. This modular feature matches with the requirements maintainability and flexibility. The realization of the modular layered control-box is shown in Figure 11.

Design considerations The function of this control box is more than covering the electronics. Among others the function of the control-box

- Provides protection of the electronics during crashes, and the user during handling.
- Holds the components in place and allowing a form of vibration isolation between frame and electronics.
- Provides the human-interfaces such as buttons and light signals.
- Allows access to the connectors of the flight-controller and the companion-computer.

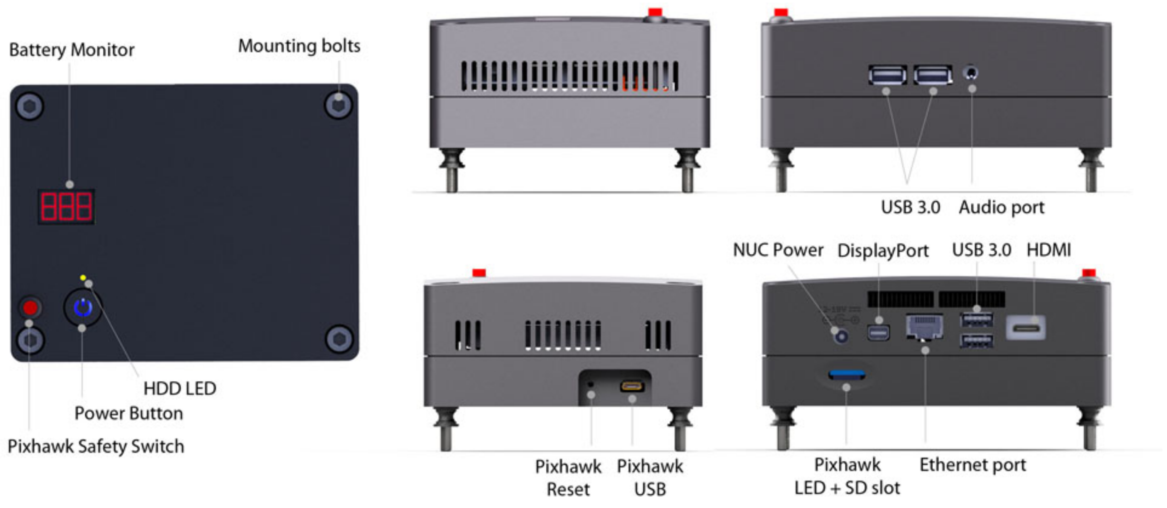


Figure 11: Overview and explanation of functionalities and components of the control-box. All the connectors are accessible from the outside in order to avoid that the user has to open the cover.

- Gives the multi-rotor a reliable, robust, professional and appealing look.

In this design it is chosen to leave the connectors of the Intel Nuc accessible from the outside, as is shown in Figure 11. This is done to avoid that the user requires to open the control box for, debugging, resetting or unplugging one of the components. In this way it is also avoided that the multi-rotor electronic components fly unprotected without a cover.

Preparing the Intel Nuc As it is shown in Figure 13.E the Intel Nuc is delivered in an aluminum housing. The originally mass of the Intel Nuc is 600 gram. In order to reduce the mass, all unnecessary components are removed such as the aluminum enclosure and the steel hard disk drive holder. Since the Wifi and Blue-tooth antenna are permanently mounted to the enclosure, the antenna is replaced. The stripped Intel Nuc is shown in Figure 13.A, including the 275GB ssd storage and 8GB memory, the remaining mass is 190 gram.

Triple redundant power supply The control box can be powered from 3 different sources. The complete control box can be powered using a four cell LiPo battery pack via one of the two JST-XH (balance plug) connectors. Another option to power the complete control box is to use the original Intel Nuc's power adapter and connect it via an adapter to the JST-XH power connector. The third method is to power the companion-computer and flight-controller separately. The Pixhawk flight-controller can be powered with $5VDC$ via its usb-port. The Intel Nuc can be powered using the original power adapter via the dc-plug. By using the default connectors, the chance of connecting a wrong voltage is small. For example the balance plug of a non four-cell battery does not fit in a four-cell battery balance plug. The highest voltage that can occur in the control box is the voltage produced by the original Intel Nuc power adapter which is $19VDC$.

PCB Design Changing batteries or power sources is possible without discontinuing the power to the electronics, i.e., the Intel Nuc can remain on during a battery change. A similar battery changing system is used in the Turtle robots of the Tech United soccer team. Instead of one, two battery connectors are present. The connectors are separated with Schottky diodes in order to avoid commutation of the batteries. As long as at least one battery is connected the electronics are powered. The highest voltage of the two batteries is used. Compared to the regular silicon diodes showing a voltage drop of $\approx 0.7V$, the voltage drop over Schottky diodes is only $0.2V$. In order to place all electrical components in a small form-factor, a PCB is designed. Besides components for conditioning of the power also a voltage divider for sensing the battery state is placed on the PCB. The circuit is schematically shown in Figure 12. In order to avoid possible misunderstandings, we want to stress that only the control box is powered via the

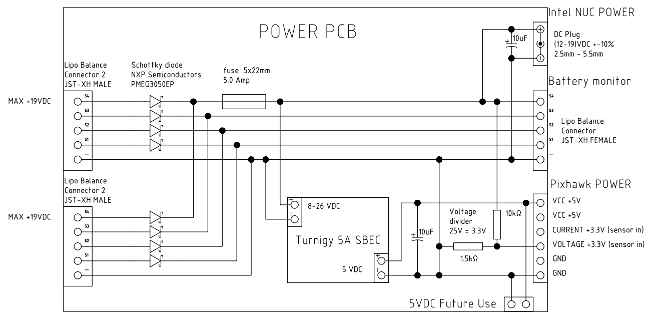

POWER PCB
Lipo Balance Connector 2 JST-XH MALE
Schottky diode NXP Semiconductors PMEG3050EP
fuse 5x22mm 5.0 Amp
MAX +19VDC
S4
S3
S2
S1
−
Lipo Balance Connector 2 JST-XH MALE
MAX +19VDC
Intel NUC POWER
10uF
DC Plug (12-19)VDC +-10% 2.5mm - 5.5mm
Battery monitor
Lipo Balance Connector JST-XH FEMALE
Pixhawk POWER
VCC +5V
VCC +5V
CURRENT +3.3V (sensor in)
VOLTAGE +3.3V (sensor in)
GND
GND
8-26 VDC
Turnigy 5A SBEC
5 VDC
10uF
Voltage divider 25V = 3.3V
10kΩ
1.5kΩ
5VDC Future Use

Figure 12: Detail overview of the electrical wiring diagram of the power conditioning PCB for decoupling the two LiPo battery packages.

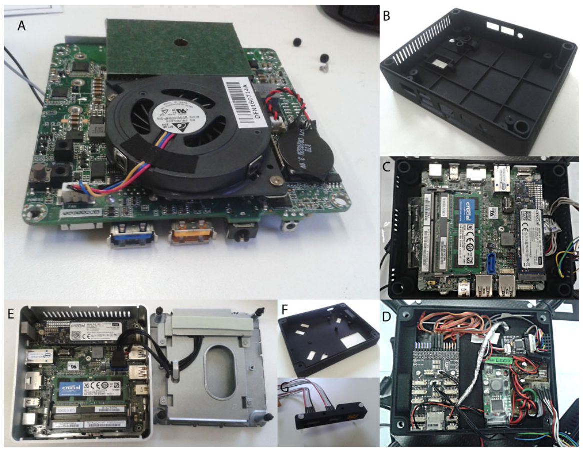


Figure 13: Overview of the electronics in the control-box. A. Intel Nuc, B,F,G. 3D-printed parts, E. Intel Nuc in original housing, C,D. electronics in 3D printed housing.

Schottky diodes, i.e., the large current to the propulsion system follows a different circuit. The complete electrical circuit is explained in more depth by the electrical diagrams which can be found in Appendix H.

Additve Manufacturing The selected production method for this control-box is additive manufacturing, also known as 3D-printing. An advantage of this production method is that a high degree of functional-density can be reached. Due to the form freedom and available materials a highly optimized, light-weight, product can be obtained. In Figure 14 the designs of the housing of the companion-computer-layer (left), and the housing of the flight-controller-layer (right) are indicated. The spring suspended power-on-button, of the Intel Nuc, is integrated in the 3D-printed product in the form of a leaf spring, as is indicated in Figure 15. The selected material is polyamide (PA) and produced using the selective laser sinter (SLS) process. The symbols indicating the maximum input-power, voltage and polarity, are directly printed on the cover.

Emergency stop To immediately stop the rotors from spinning, in case of emergency, a failsafe emergency stop is implemented. The default kill-switch procedure in the Pixhawk is triggered as soon as the radio-receiver looses connection with the radio-transmitter. Or more precisely when one of the control channels sent by the radio-transmitter drop signal, the kill-switch is triggered. This means that

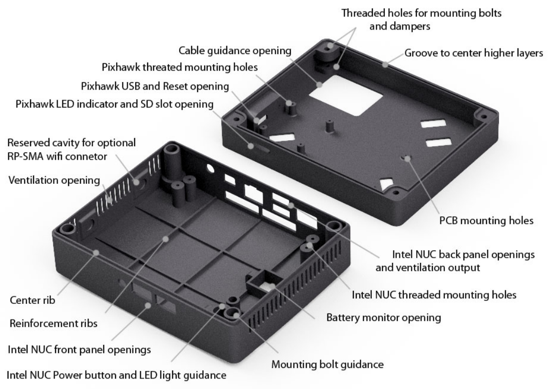

Threaded holes for mounting bolts and dampers
Cable guidance opening
Groove to center higher layers
Pixhawk threated mounting holes
Pixhawk USB and Reset opening
Pixhawk LED indicator and SD slot opening
Reserved cavity for optional RP-SMA wifi connetor
Ventilation opening
PCB mounting holes
Intel NUC back panel openings and ventilation output
Intel NUC threaded mounting holes
Center rib
Battery monitor opening
Reinforcement ribs
Intel NUC front panel openings
Mounting bolt guidance
Intel NUC Power button and LED light guidance

Figure 14: Top and bottom part of the layered control-box. This image emphasizes the high feature density of the design.

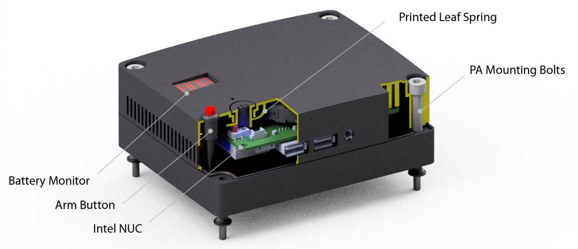


Figure 15: Cross section of the designed cover. Indicated is the 3D printed leaf spring for the Intel Nuc power on button. The button is such designed that if the spring breaks the button remains working.

the kill-switch will be activated by switching off the radio-transmitter. Theoretically, this is still a software kill-switch since it is implemented in the default Pixhawk code. Alternatively, one can decide to use/design a hardware battery-circuit-breaker that is activated on radio-signal drop.

2.5.3 Selection of a propulsion system

The previous section described the design of the control-box. The control-box represents, besides the cameras and sensors, the payload. Based on the mass of the payload a propulsion system can be selected. In general it is a challenging task to find an optimal combination of propeller-blades, electric-motor, and ESC. It is even more challenging since the required technical specifications are often not specified for commercially available components.

Experimental tests propeller configurations In order to gain some experience with the design of a multi-rotor, two very basic multi-rotors were built. The first multi-rotor uses a commercial frame as is shown in Figure 16.B, and in the second prototype the frame is replaced with our own designed frame as is shown in Figure 16.D. The performance, in terms of hover flight-time, with different types of propellers and vehicle mass configurations are tested on these prototypes. The results are shown in Table 1.
The following lessons were learned. The flight time decreases almost linearly with the mass, in the tested configuration space. For our frame a propeller with 3 blades gives the longest flight time. Propellers with the same specifications, but from different producers can have significantly different performance.

Propulsion system Based on the obtained experience and the collected data in the market research, a propulsion system can be selected. As mentioned in the system requirements in Section 2.2 the purpose of this multi-rotor is to fulfill the function of a drone referee during, for example, a RoboCup MSL [27] soccer game. The playtime of a game is 15 minutes, therefore we aim for a flight-time beyond 15 minutes. Together with the payload mass, an initial guess of the required battery capacity can be based on one of the commercially available multi-rotors in Appendix A.1. In general the further selection of the propulsion system is an iterative process. Finally, the propulsion system, as is specified in Table A.10, is selected. The selected battery is a 4 cell 4500 mA LiPo. It can be observed that the propulsion-system is comparable to the commercially available multi-rotor indicated with id. 11 in Table A.1. However, a shorter flight time is exchanged for a higher vehicle mass. If the current propulsion system, for some reason, does not satisfy the future requirements, it can be easily replaced by another system due to our modular design.

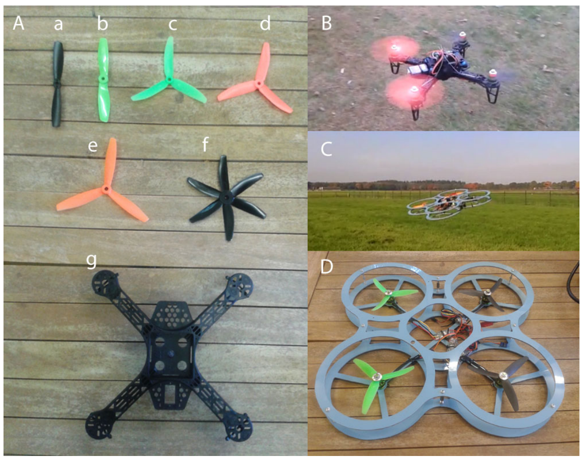


Figure 16: First simple prototype used for gaining insight and performing measurements. In A, the different tested propeller configurations are shown. In B the assambled original frame, performing a flighttest, is shown. In C and D our first produced prototype is shown.

Frame	Blade	Producer	Mass	Flight-time
B	(d) 3 × 5040	Diatone	385	9.5
B	(d) 3 × 5040	Diatone	430	8.0
B	(d) 3 × 5040	Diatone	505	5.0
D	(d) 3 × 5040	Diatone	530	3.1
D	(a) 2 × 5045	Kinkong	530	-
D	(b) 2 × 5045	Dalprop	530	3.3
D	(c) 3 × 5030	HobbyKing	530	5.0
D	(e) 3 × 6045	Gemfan	530	3.7
D	(f) 6 × 5040	Gemfan	530	1.5

Table 1: Experimentally measured flight-times of several different propeller and frame configurations. The flight-time is based on a fully charged 3 cell 1000mA LiPo battery. The blade configuration is indicated according to the following example: (d) 3 × 5040 where (d) is the blade as shown in Figure 16.A.d, 3 indicates the number of blades per propeller, 50 indicates a diameter of 5.0 inch and 40 a pitch of 4.0 inch. Frame B and D corresponds to the frame as shown in Figure 16.B and 16.D respectively. Due to an unbalance it was not possible to fly with the Kingkong blades.

2.5.4 Design concepts

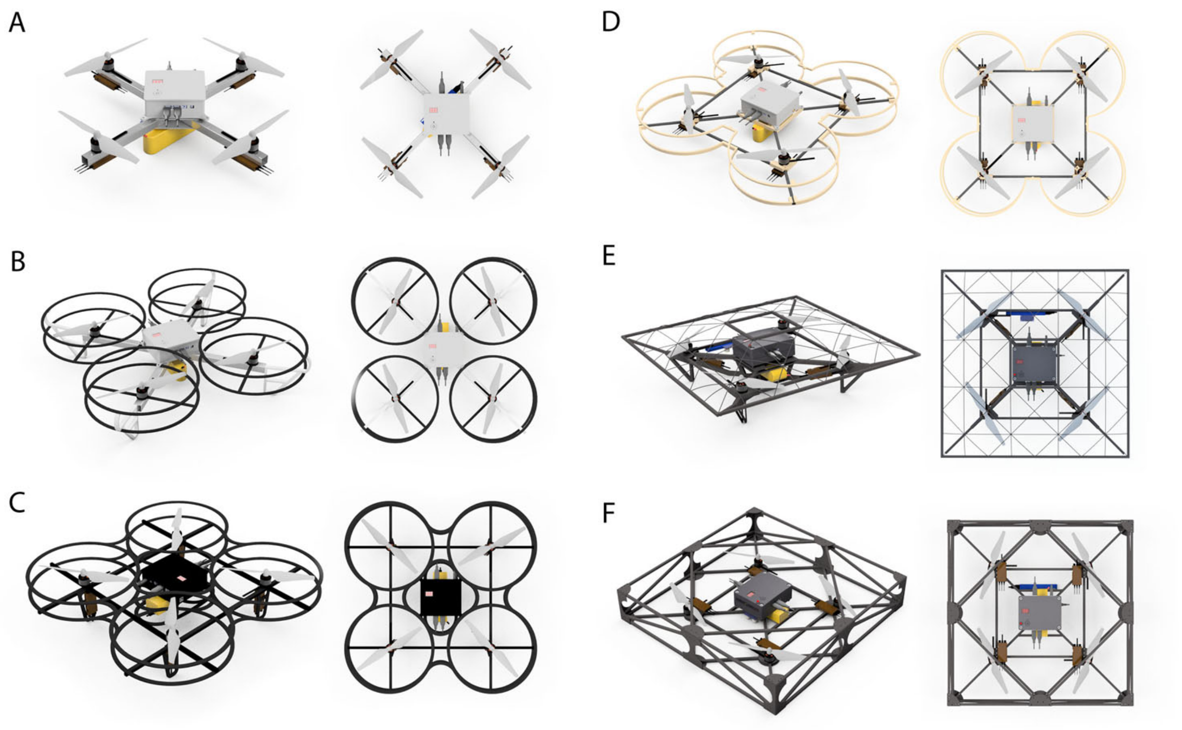


Figure 17: Several conceptual designs.

Frame design In Section 2.5.2 the design of the control-box is described and in Section 2.5.3 the propulsion system is selected. In this section the design of the frame is described. First several conceptual designs are described, from which a final concept is selected and elaborated in more detail in Section 2.5.5 and in Section 2.5.6.

Discussion of concepts In Figure 17 several of our conceptual designs are shown. In Figure 17.A the probably simplest possible design of a quad-rotor is shown. The design consists of a cross-arm construction with the rotors at the tip of each arm. In this concept, no blade protection is available around the propellers. Since safety is an important requirement, in the succeeding concepts rotor protectors are present. Figure 17.B shows a similar concept as 17.A, however, extended with blade protectors and integrated landing gears. In Figure 17.C the frame is constructed from two parallel plates that in-cooperate the function of frame and the function of blade protector. This concept is realized in the form of a (scaled) prototype and is demonstrated in Figure 16.C and 16.D. For this prototype, (blue) HDPE plate material is used. The plates are separated by spacers and screws. The prototype is tested in practice. However, a disadvantage of this concept is that the 2D structure of the frame is not very strong and stiff. Some propellers produce resonances and during a crash the plates bend and the propeller blades break. Performing the design in another material such as CFRP makes it too expensive. Figure 17.D shows a thin X-frame of round CFRP tubes with cross-links between the motors. A disadvantage of this model are the relatively complex form-transitions between flat plates and round tubes. In Figure 17.E a more serious concept is shown. In Figure 18 a more detailed image of the design is given. With a little bit of imagination, the Evoluon and the meshed structure of the Blob can be recognized, which are iconic buildings located in Eindhoven. The intended material is CFRP, and consist of pultruded lightweight square tubes, plates, and thin rods. The mesh of thin rods, in potential, protects the user from unexpected contact with the rotors, during for example a collision. However, producing the mesh dome requires significant labor and is therefore not realized. In figure 16.F a more square shaped design is proposed which is more simple to produce. In Section 2.5.5, the concept in Figure 16.F is further discussed.

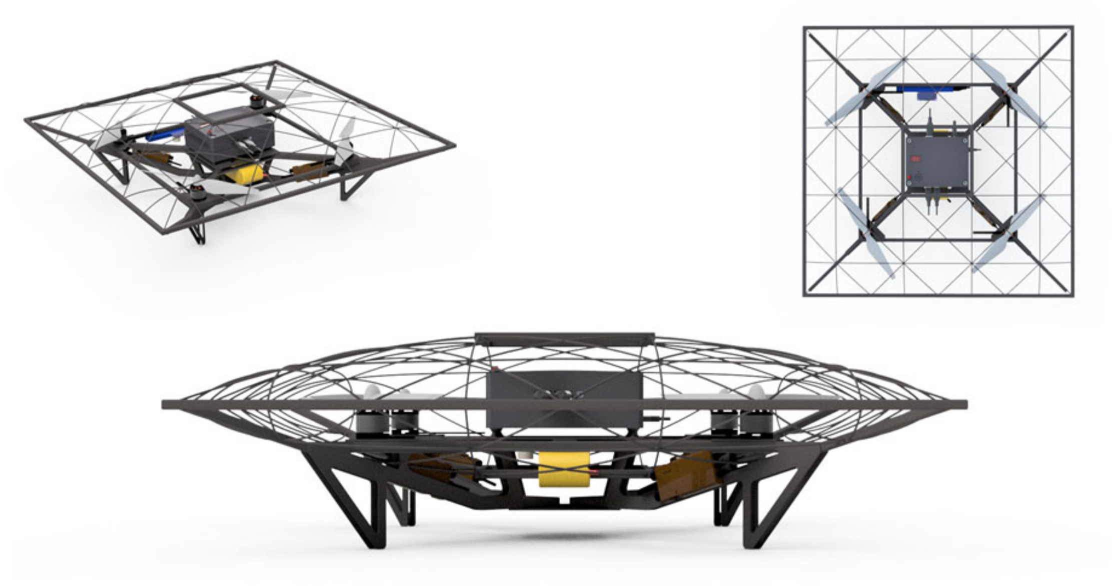


Figure 18: Concept of a multi-rotor with blade protection in the form of a mesh cover in order to avoid direct contact between rotors and environment. The mesh can potentially made of thin CFRP rods.

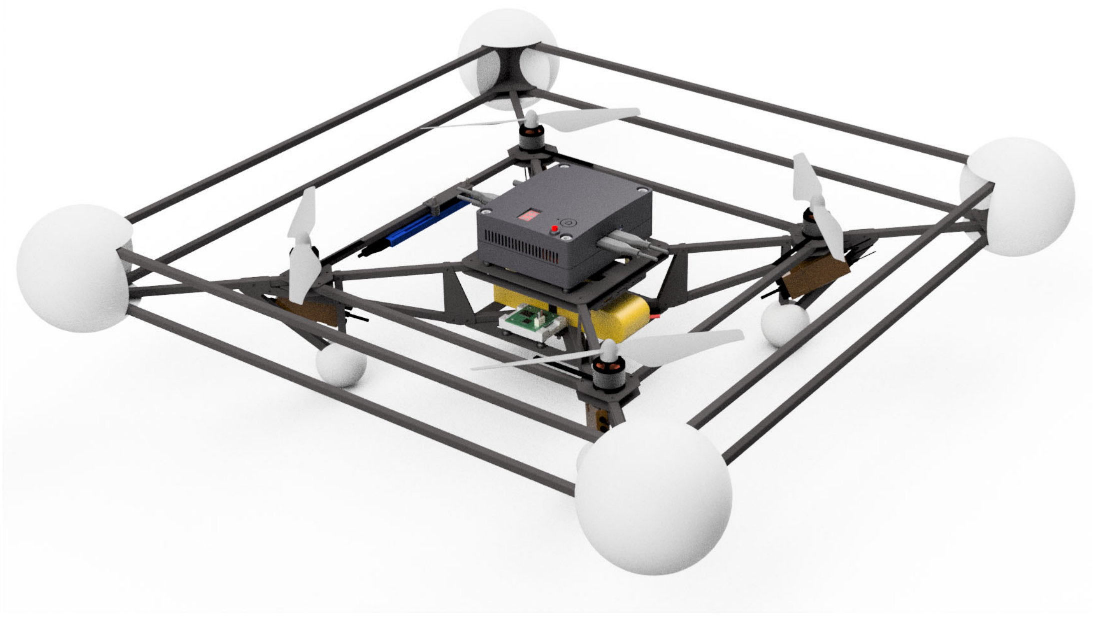


Figure 19: Detail view of prototype frame 1. The border around the rotors is made of square pultruded CFRP tubes and should protect the rotors and environment during, for example, crashes. The EPS balls at the corners have the function of absorbing energy during impact and smoothing the potentially ‘sharp’ corners.

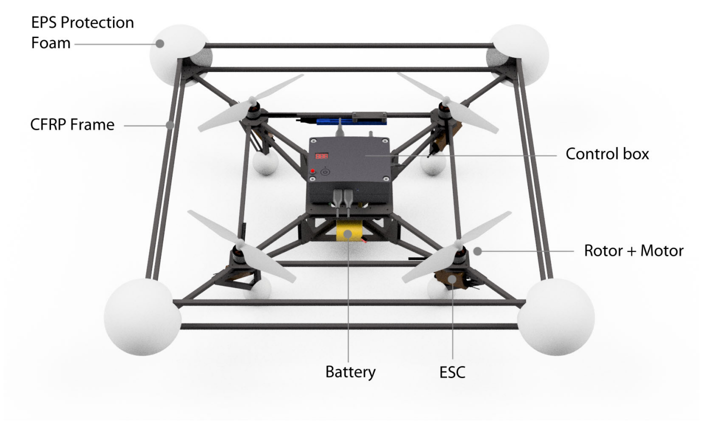


Figure 20: Indication of some of the components and functions of prototype frame 1.

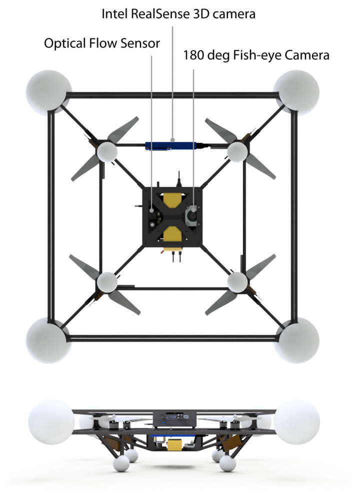


Figure 21: Bottom and side view indicating the positions of the cameras and sensors.

2.5.5 Design of prototype frame 1

In this section, the design of the first complete prototype is described. This frame is based on concept E and F as is shown in Figure 17. The intention of this frame is to hold the sensors, control-box, and propulsion system as described in Sections 2.5.1, 2.5.2, and 2.5.3 respectively. On top of this, we try to implement the safety requirements as specified in Section 2.2. The final design is shown in Figure 19, and its specific components are indicated in Figure 20, and 21.

Design Compared to most multi-rotor frames, as are listed in Appendix A.1, this frame design is not purely based on the cross shape approach. Instead, the design is based on a truss construction, where the constructional stiffness is reached by introducing triangles. Furthermore, all joints are based on combining 2D connections, which are relatively simple to produce in the form of gusset plates, as is shown in Figure 22.C. The form complexity is placed in the gusset plates, leaving the tubes almost unprocessed, which reduces the production costs. The purpose of the border is protecting the blades and human during collision. Although it is possible to calculate the tensions in the frame material using finite element analysis, as is shown in Figure 22.A-B, the forces during a crash can be high and unpredictable. In addition, the strength calculation is only valid with well defined material properties, including the anisotropic behavior of the pultrusion tubes. In order to save time, it was decided to limit the calculations to a comparison of the second-moments-of-area of different beam shapes. The expanded polystyrene (EPS) helps protecting the frame and environment during crashes.

Material The selected material for the complete frame is CFRP. This material is selected because of its high stiffness and relative low mass-density [28]. The frame consist of thin square tubes and plates. The square tubes are produced using the pultrusion process, which is a relatively cost efficient process. A disadvantage of this production process is that the material properties are highly anisotropic. Due to a crash, we learned the hard way that while the pultrusion material is strong in some load cases, it is brittle in the other load cases. The intersection dimensions of the square tube are $8 \times 8 \times 0.5mm$. According to the supplier the mass per meter is $22gram$ and the real measured mass is $27gram$ meter. This resulted in a heavier frame than in the 3D model was expected.

Production We decided to produce the prototype and components ourselves. The gusset plates as shown in Figure 22, are cut from a prefabricated plate of $1mm$ and $1.5mm$ CFRP, using a conventional milling machine and a band saw. As can be observed in Figure 22 most of the parts are designed such that they can be nested, resulting in a minimum waste of material. In order to assemble the components, as it is shown in Figure 23.A, in first place, screws were used, resulting in the frame as shown in Figure 23.C. In order to reduce the vehicle mass, at a later stage, instead of screws, epoxy is used to assemble

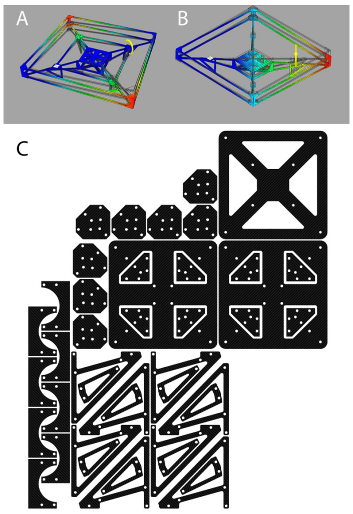


Figure 22: Example of finite element calculations in A and B. In C the nest-ability of the gusset plates is shown. The complexity of the form transitions is integrated in the 2D plates such that the frame tubes only need to be cut at length.

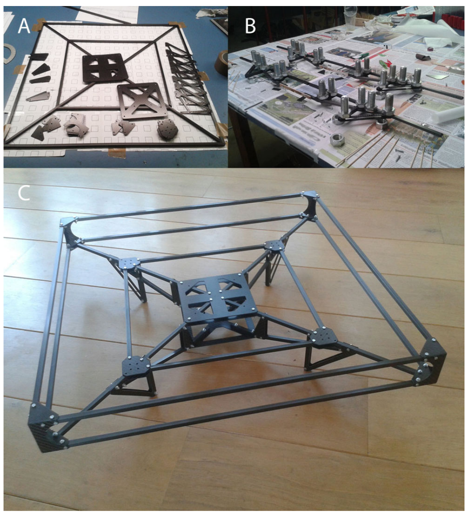


Figure 23: Production process of the multi-rotor frame 1. In A the milled loose parts are shown. In B the assemblage process using epoxy is shown. Figure C shows the assembled frame before the screws were replaced by epoxy.

the multi-rotor. The parts were connected in several stages as shown in Figure 23.B. An advantage of using epoxy compared to using screws is that the frame stiffness is significantly increased.

Intrinsic safety For safety reasons the multi-rotor should only be used in a shielded area, with for example nets, and operated by a capable user. The rotors are only protected from the side. Due to weight and performance issues, the inlet and exhaust areas of the rotor are unprotected. In order to promote safe handling the power connections are placed relatively far and below the rotors. The power connector is placed such that the connections are not visible from the top, as shown in Figure 24. In this way the user has to rotate the multi-rotor in order to connect the power. This will minimize the change that the user puts their hands between the rotors. Disconnecting, in case of emergency, is possible from all positions by pulling on the power cable. However, with a normal functioning system, the rotors are by default off and are, remotely, armed after the Pixhawk safety button on top of the control box is released.

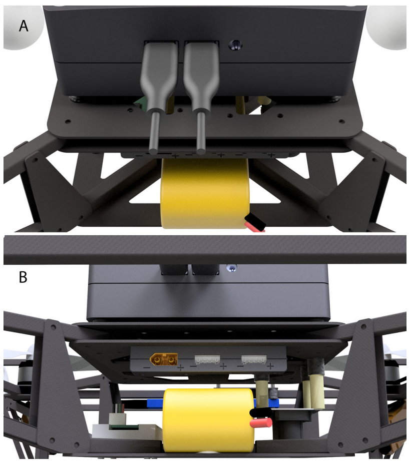


Figure 24: Power connectors, the yellow left connector is for the main power and two white balance plugs are for powering the controllers and accessory. The connectors are not visible from the top view, forcing the user to approach the connectors from below, which is considered to be more safe.

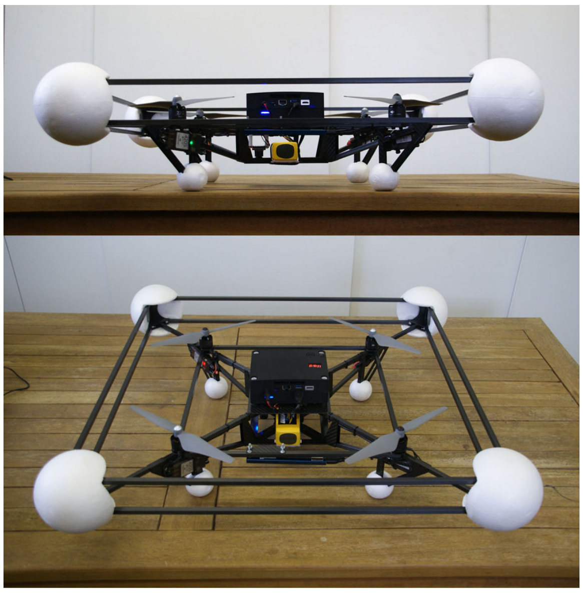


Figure 25: Overview of the finished product.

Final Product In Figure 25 the complete assembled multi-rotor is shown. The size of the frame is $640 \times 640 \times 125mm$ with a diagonal motor to motor distance of $500mm$. The mass of the carbon frame is $440g$ and the complete mass is $1950g$. Flight tests using the remote control show that the multi-rotor is able to fly. With the current 4 cell $4500mAh$ LiPo battery package the flight time is approximately 11.0 minutes. Hovering requires 61% of the maximum thrust that the rotors are able to deliver.

2.5.6 Design of prototype frame 2

Due to the crash as will be mentioned in Section 6.2.3, an improved frame for the multi-rotor is designed. In the design of this frame, more attention is payed on surviving crashes. It can be expected that crashes will occur more often with a multi-rotor running experimental control algorithms. Due to the modular design of our multi-rotor, changing the frame appeared to be relatively simple.

Form As is shown in Figure 26, in this design, two CFRP tubes form the basis for the structure of the multi-rotor. The tubes do not cross in the same plane, therefore they do not need to be cut in the center and preserve their originally strength. The tubes are longer than the diagonal distance between the motors. The protruded tube will protect the rotor during crashes. At the tip of the tubes, shock-absorbers are placed that also function as landing suspension. Since the landing suspension is placed far from the center they do not cover the field of view of the fish-eye camera. The fish-eye camera is placed on the bottom plate in such a way that when the bottom hits the flat floor the lens will not be damaged. Besides the cameras, the bottom plate holds and protects the battery. Together with the top plate and spacers/pillars between the plates the tubes are hold in cross form. The top and bottom plate are best observed in Figure 28, and the different components are indicated in Figure 27. By carving-in two of the

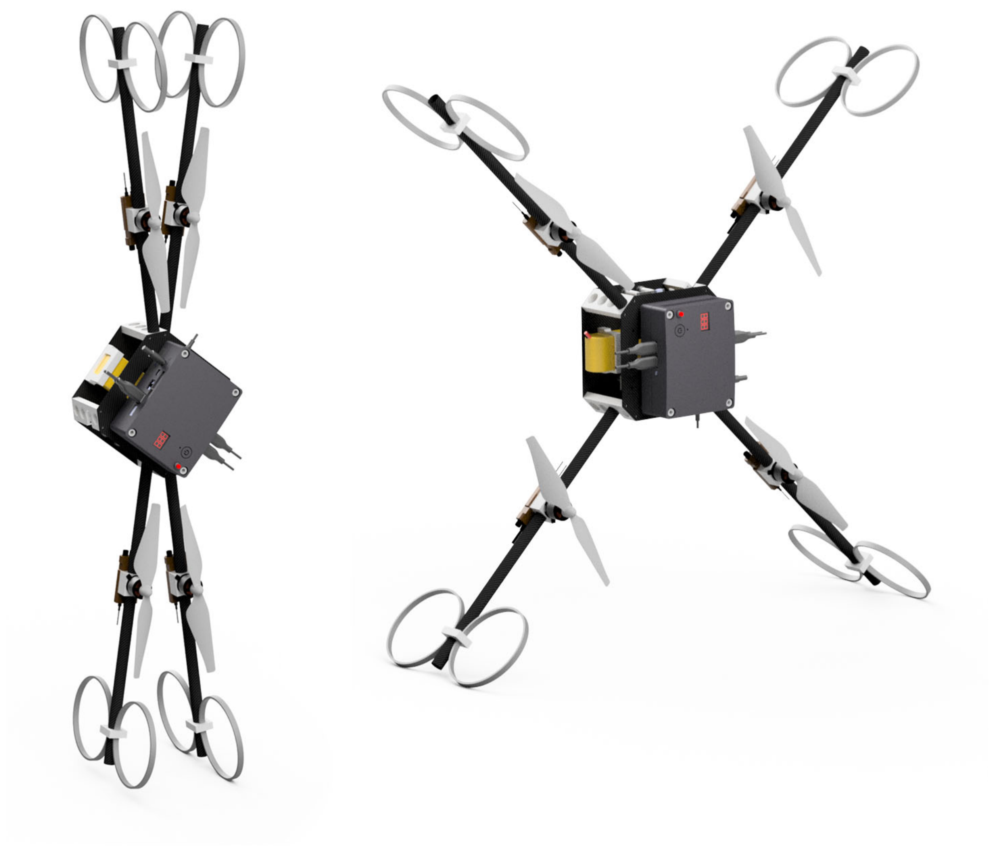


Figure 26: Design overview of the second prototype. The arms of the multi-rotor can be folded, which allows compact transport and storage. This multi-rotor is more simple to produce than the first prototype. In this prototype no blade protectors are present, and therefore extra safety measures have to be taken into account such as flying only in a shielded area. The extra long tubes protect the rotors during crashes.

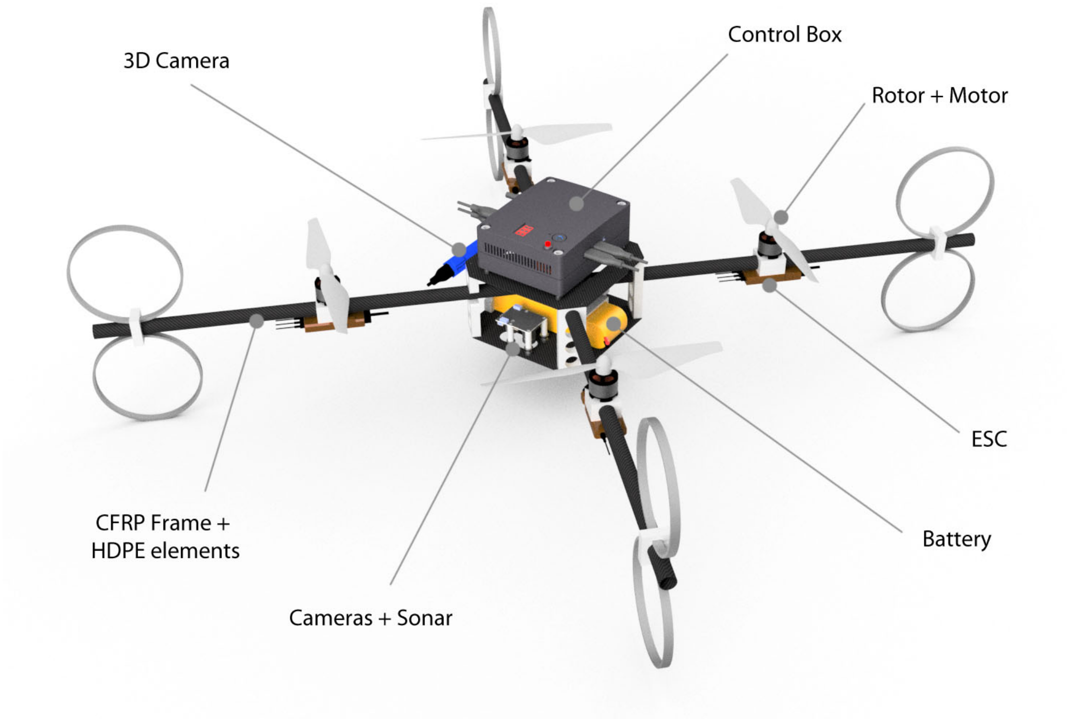


Figure 27: Indication of components and functions. Clearly visible are the PVC suspension springs at the ends of the frame tubes.

holes holding the tubes a snap connection is created. In this way, the multi-rotor can be folded up, as it is shown in Figure 26. The folded multi-rotor is easier to transport and store.

Material and Production We produced this multi-rotor frame using standard metal work machines. The $\varnothing 16 \times 1mm$ CFRP tubes are cut at length and the holes for the screws are drilled during montage with a hand-held drilling machine. The two $1mm$ thick CFRP mounting plates are produced using a band-saw and milling machine. The HDPE plastic parts, indicated in white in Figure 26, 27, and 28, are produced from a $10mm$ HDPE plate using conventional milling. The shock absorber suspension is made of $110 \times 3.2mm$ PVC pipe with a length of $8mm$. All components are mounted with small metal screws.

Final product The realization of the multi-rotor is shown in Figure 29. Figure 30, shows the accessory components belonging to the multi-rotor:

A Battery charger

B Multi-rotor

C Remote radio-transmitter

D Original Intel Nuc power adapter

E LiPo battery packages

F Wifi router/access-point for ground-station

The dimensions of the multi-rotor are $700 \times 700 \times 240mm$ and when folded $1070 \times 180 \times 240mm$. The diagonal distance between the motors is $500mm$. The total mass of the multi-rotor is $1910g$. Flight tests using the remote control show that the multi-rotor's flight time is approximately 11.5 minutes with a 4 cell $4500mAh$ LiPo battery package. For hovering, 61% of the maximum available thrust is required. All required components and materials for building this multi-rotor are listed in the bill of material list in Appendix B. The overall cost, including all the accessory, sensors, and the most expensive component (the Intel Nuc processor (≈€690)) was about €1700, excluding labor. Compared to existing options in the market, that do not provide the flexibility and modularity of the present solution, this price was found to be acceptable.

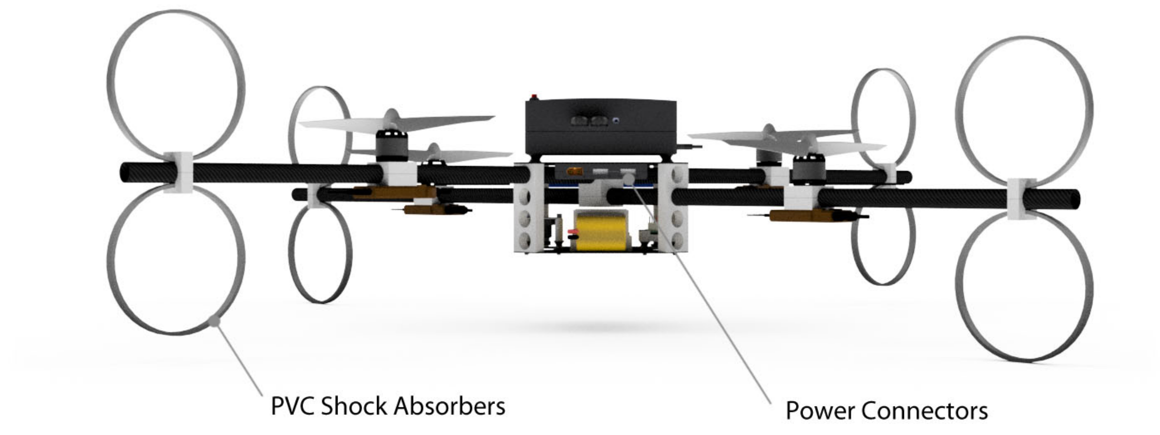


Figure 28: Side-view, showing that the rotors are not in the same plane, and that the cameras are protected by the frame.

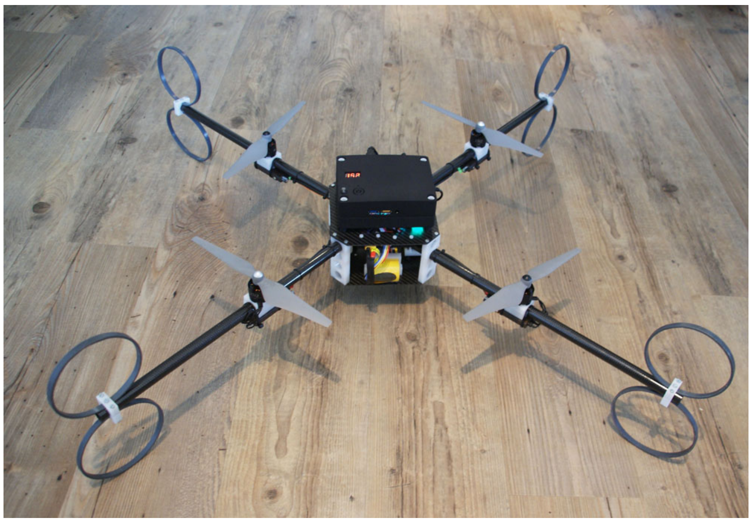


Figure 29: Overview of the produced finished product. The frame consist of CFRP tubes and plates. The frame components are connected by (white) HDPE blocks containing all the difficult form transitions and are easy to produce using milling techniques.

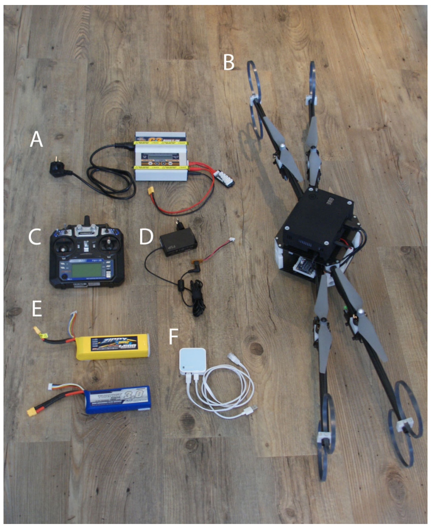


Figure 30: The accessory belonging to the multi-rotor. A. battery-charger, B. multi-rotor, C. remote-controller, D. power-adapter, E. batteries, F. Wifi access-point.

2.6 Summary and evaluation

This section described the complete design process, from orientation to realization, of a computationally powerful multi-rotor. In Section 2.2 the system-requirements were listed, Section 2.3 summarized the general structure of a multi-rotor, and Section 2.4 described the analysis of 20 commercially available multi-rotors, 6 flight-controllers, 6 potential companion-computers, and several sensors in a market research. In Section 2.5, a selection of components was described. The Pixhawk flight-controller and the on-board Intel Nuc PC transformed our multi-rotor into an 'flying laptop'. In Section 2.5.2 the modular design of a control-box was discussed, and based on the experimental results with different propeller configurations a propulsion system was selected which was described in Section 2.5.3. Several concepts for the design of the multi-rotor were discussed in Section 2.5.4. The final complete design was described in Section 2.5.5 and an improved frame was described in Section 2.5.6. The final multi-rotor has multiple sensors on-board, among others an fish-eye camera, a RGB-D camera, an ultrasonic height sensor, and an optical flow sensor. The dimensions of the final multi-rotor are $700 \times 700 \times 240mm$. The diagonal distance between the motors is $500mm$. The total mass of the multi-rotor is $1910g$.

As a final note, let us compare and contrast the system requirements with the final product. The multi-rotor is naturally capable of providing the required **propulsion** to lift its weight of about 1910 g. The **time of flight** is nearly 15 minutes and the **dimensions** are 700x700x240mm meeting the requirements needed for the intended drone refereeing application. The overall cost did not exceed €2000, which makes it feasible to **produce** more units in the future if needed. The first prototype that crashed was fixed but in order to avoid similar problems in the future a second frame was designed and produced. The simplicity of the second design and the modularity of the overall system, makes us believe it will be easy to **maintain**. On the other hand the second design is not as **safe** as the first one, since the propellers are less protected. The choices for having an open-source flight controller and an Intel Nuc PC assure that the **flexibility**, **accessibility**, **control**, **communication**, and **processing unit** requirements are met, and also allowing to incorporate already very interesting **sensors**, such as a fish-eye camera, but most importantly leave the door open for whichever sensors are needed in the future.

3 Interface and Software Design for a Multi-Rotor Platform

In this chapter, the essential software peripherals are developed in order to allow users to design and deploy software on the multi-rotor using Matlab Simulink. Section 3.1 provides a brief motivation for using Matlab/Simulink and summarizes some of the special features of our software. In Section 3.2 the current system layout in terms of software is described. Section 3.3 discusses a full implementation in Matlab and Matlab Simulink of the MAVLink protocol, a widely spread communication protocol for multi-rotors. Section 3.4 describes the implemented on-board computer vision algorithms which will later be used for indoor positioning. In Section 3.5 an overview of the integration of all software components is given. Finally, in Section 3.6, a summary and some remarks are given.

3.1 Introduction

For many years, Matlab [15] has been the standard development tool for the experiments during the systems-and-control courses worldwide and in particular at TU/e [16], [17]. In many courses, Matlab is used to simulate the impact of control algorithms on dynamical models. In some of the courses real-world experiments on hardware are provided and again Matlab is the default design tool. During the courses students become experienced Matlab users. Therefore, it is a logic step to use Matlab for the designing of the control software for a multi-rotor. However, in many of the current commercially available multi-rotor systems, the software is programmed in c-code and executed on a flight-controller with relatively low computational power. These setups suffer from two disadvantages:

1. programming in c-code is often time consuming and debugging requires a significant effort compared to developing control-software in a graphical environment such as Matlab Simulink;
2. it is often problematic to deploy software, which was tested in simulation on a regular laptop, on a flight-controller with limited computational power.

In order to solve these problems, our multi-rotor is equipped with, besides of the flight-controller, an Intel Nuc i7 companion-computer, as was discussed in Section 2.5.1. The performance of the Intel Nuc is comparable to the performance of a TU/e laptop [29]. In our case, the Matlab development environment runs directly on the companion-computer. The companion-computer communicates via a wired connection with the Pixhawk flight-controller. The communication between the two consists of instructions and state updates. In order to perform the communication in a standardized structure, the MAVLink protocol [30] is used. For Matlab Simulink, currently, there does not exist a complete, real-time implementation of the MAVLink protocol, in Matlab Simulink, although, on the Internet, there are a limited number of open-source initiatives. However, provided implementations of the MAVLink protocol communicate via (non-real-time) libraries and others implement only a limited, incomplete, hard-coded, or unidirectional part of the protocol. Besides the MAVLink protocol itself, it is often unclear how these setups guarantee that the MAVLink communication messages arrive in time. Another concern is how to guarantee the sample-time of the Simulink model.

In the sequel, a full implementation of the MAVLink protocol for Matlab and Matlab Simulink is given. A clear distinction between protocol, transport-layer, and real-time implementation is highlighted. Our MAVLink protocol implementation consist of pure Matlab code and is auto-generated from the 'common.xml' file containing the original protocol-definitions. This will guarantee the independence on libraries, operating-system, and robustness to future changes. The protocol can be used in Simulink 'Normal mode' (where the software is executed within Simulink itself) as well as in 'External mode' (where the software is compiled and executed outside Simulink). Noteworthy is that a professional drone company (Avular) is now using our implementation. Since the messages in the MAVLink protocol do not use specific start-stop indicators nor are of fixed byte-length, a dedicated serial-port S-function for Simulink is developed. An S-function allows to execute c-code within Matlab and Simulink. This function is able to receive and send data of variable length in Linux OS as well as in MS Windows. To our best knowledge, the available default serial-port functions within Simulink use a fixed byte size, and release their data only when the buffer is full. Due to this, MAVLink messages arrive with significant delays. Furthermore, in order to guarantee real-time execution, a Simulink pacer S-function is developed which synchronizes the sample-time with the real-time system-clock, for Linux and MS Windows OS.

In order to later perform a proof-of-concept, to show that the multi-rotor setup is able to perform a completely autonomous flight, several computer vision algorithms are studied and implemented in order to determine the multi-rotor's position. We want to stress that here with autonomous we mean that all the computations are executed on-board without the requirement to communicate with an external system. The multi-rotor determines its position, on-board, purely based on vision, which is in contrast to many other applications using an external system (top-camera) for position and heading feedback. In one method, the position of the multi-rotor is determined based on the detection of lines on a soccer-field. In a more simplified setting, the position of the multi-rotors is determined based on the detection of Aruco markers which are spread in the environment. Aruco markers are 2D black-white markers that can be detected with a high confidence using a computer-vision algorithm. A desire and advice of the autonomous referee system [14] was not to use Matlab Simulink for developing computer vision algorithms, because of speed considerations. Therefore, the computer vision algorithm is not developed in Matlab, but in c++ and is using the OpenCV library for computer-vision.

Although in this work we do not strictly rely on the system-architecture as is described in the autonomous referee system [14] the following model can be kept in mind. A Simulink model running in real-time on the companion-computer instructs the flight-controller via the MAVLink protocol. A computer vision algorithm determining the position of the multi-rotor runs also, in parallel, on the companion-computer and broadcasts its results to the real-time Simulink model. The computer vision algorithm, basically, functions as an advanced sensor.

3.2 System Layout in Terms of Software

In this section is discussed the system-layout in terms of physical-connections, configuration-freedom, operating-system, and software development platforms.

3.2.1 Physical connections between system-elements

In Figure 31 an overview of the physical connections between the different system components is graphically visualized. The multi-rotor is equipped with a Pixhawk flight-controller. The flight-controller provides the angular-velocity references for the four rotors which are part of the propulsion system. In addition, the flight-controller receives measurement data from the optical flow ground velocity sensor and ultrasonic height sensor. There is a radio-control receiver which receives a six-channel signal from a remote control. Furthermore, the safety button is connected to the Pixhawk. The Pixhawk gives visual and audio feedback to the user by using LEDs and a buzzer. Besides the flight-controller, a companion-computer is present on the multi-rotor. The companion-computer is an Intel Nuc i7, with a computational power comparable to a TU/e laptop [29]. The Intel Nuc enables running the computationally intensive tasks on-board, such as computer-vision algorithms. Two cameras are connected to the companion computer via a USB connection. A wide angle fish-eye camera provides 180° bottom facing omni-vision. The other camera, an Intel Real-Sense R200, provides color and depth vision in down/forward direction. The companion-computer is equipped with a Wifi (and Bluetooth) connection, in order to communicate wirelessly with an external system. The communication between the companion-computer and flight-controller is via a serial connection over USB, following the MAVLink protocol [31]. It is important to mention that both Wifi and the remote controller use the 2.4GHz bandwidth. However, during experiments, no problems were experienced.

3.2.2 Configuration freedom

Before describing the software layout used in this work, we want to stress that it is possible to use other software configurations. For example, a wide range of open-source flight-controller firmware is available for the Pixhawk. In Table 2, vertically, a set of flight-controller software is listed. The PX4 firmware is often used in academic use. The user can use the default firmware, or change or extend the software to its own desires. Alternatively a low level flight-controller can be designed using the Simulink PX4 Support package. On the companion-computer, any software that is available for a regular PC can be installed. The operating system can be selected freely. Also for the companion-computer, different default software is available in order to communicate with a flight-controller. In Table 2, in the horizontal direction, a selected set of software is shown. The MAVLink-Simulink interface is developed in this work. It is noteworthy that, in principle, our MAVLink-Simulink interface also can be used as a ground-station. In that case there is no companion-computer on the multi-rotor, but the communication with the flight-controller is by a wireless data-link.

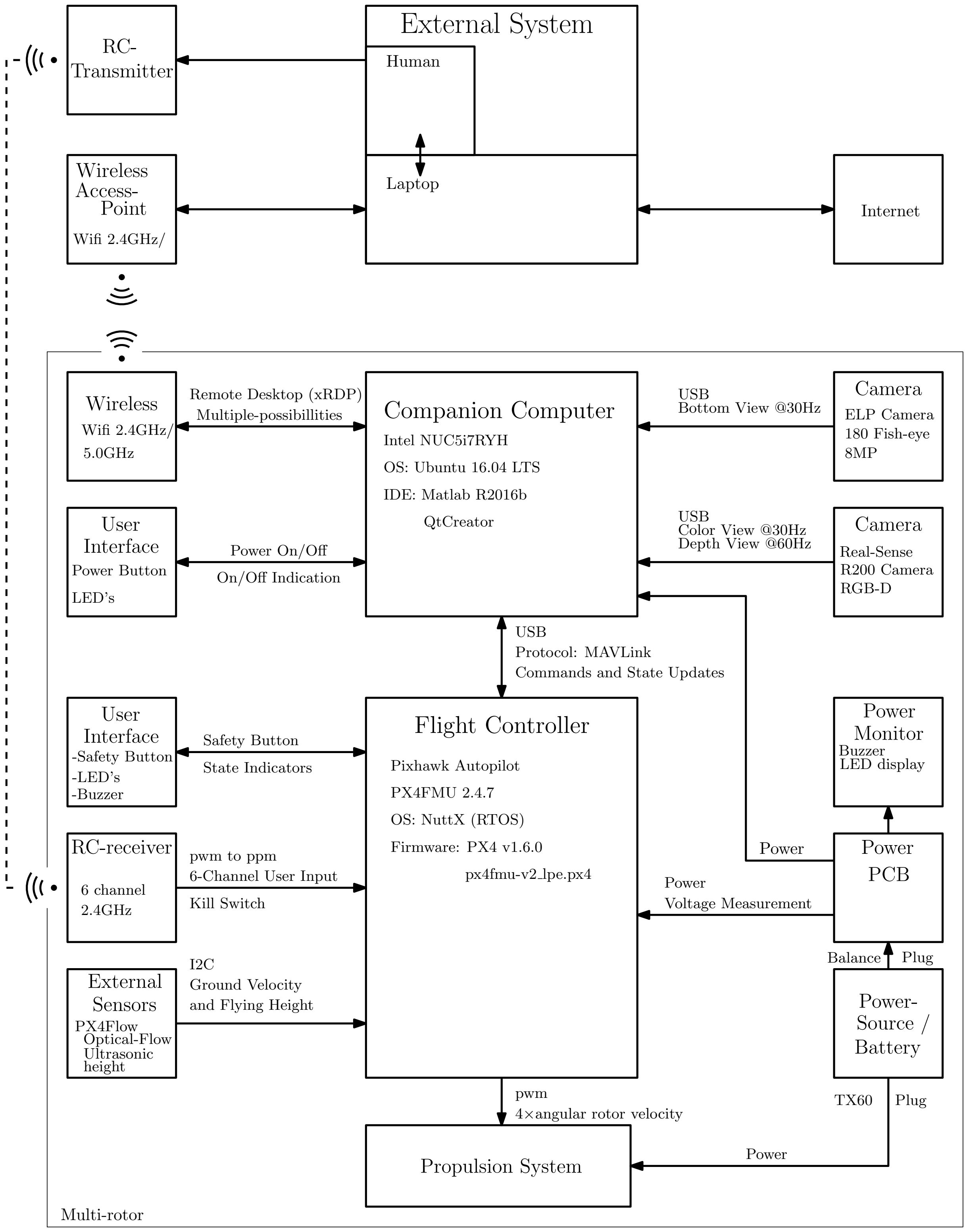


Figure 31: Schematic sketch of the structure of the multi-rotor.

	QGroundControl	(MAV)ROS	MAVLink Simulink Interface
PX4 default			×
PX4 default + custom code			
Simulink PX4 Support			
ArduPilot			

Table 2: Selection of the available software for the Pixhawk flight-controller (vertical), and the companion-computer/ground-station (horizontally). In this work the default PX4 firmware is used on the flight-controller and the in the sequel developed MAVLink-Simulink interface is used on the Intel Nuc companion-computer.

3.2.3 Operating-Systems, Development Platforms, and Remote Desktop

In this section a compressive overview of the current system-setup is given. In Figure 31, the layout of the system is schematically shown. As is indicated in Table 2 with a × symbol, in our case, the default PX4 flight controller (with local-position-estimator (lpe) [32]) is installed on the Pixhawk side. On the companion-computer side, the Linux distribution Ubuntu 16.04 LTS operating system is installed [33]. The companion-computer is running Matlab R2016b [15] and for the design of c++ code the QtCreator IDE [34] is installed. Furthermore the OpenCV library [35] is installed which is known for its efficient computer-vision algorithms. The communication between companion-computer and an external system is via a Wifi connection. In our proof-of-concept our goal is to let the multi-rotor perform an autonomous flight. Therefore, no external continuous control signals are communicated between the external-system and the companion-computer. The companion-computer is operated via remote-desktop. The xRDP server [36], [37] is used for the remote-desktop communication. The RDP protocol is a semantic protocol which is significantly faster compared to, for example, the in general graphical VNC protocol. The tutorial in Appendix F explains how to login to the multi-rotor, without having to install any additional software on your workstation, also is explained how files can be easily transfered between the multi-rotor and the users workstation.

3.3 A MAVLink Interface for Matlab Simulink

This section describes the MAVLink-Simulink interface as is developed in this work. First, in Section 3.3.1, the Simulink 'External mode' is described. In Section 3.3.2 the MAVLink protocol is discussed. MAVLink is a open-source protocol for the communication with and between Micro Air Vehicles (MAVs). In Section 3.3.3 our MAVLink-Simulink interface is discussed. This MAVLink-Simulink interface enables the use of the complete MAVLink protocol in Matlab and Matlab Simulink. Finally in Section 3.3.4, the implementation and peripherals for the real-time implementation in Simulink are discussed.

3.3.1 Simulink external mode

Traditionally, Matlab Simulink is known as a powerful simulation tool for the design and simulation of dynamical systems. Due to its intuitive graphical-interface, complex-systems can be fairly easily modeled, while a clear global system overview is perceived. This makes Simulink a popular design-tool among control-system-engineers. However, Simulink can do more. In fact it is possible with Simulink to control hardware in the loop. In this way, the graphical designed control diagram can be used for simulation as well as for real-time hardware control without having to change to a different development platform.

Besides running Simulink models in 'Normal mode' Simulink has an option to create and compile c-code and deploy it on an (external) target, this is known as 'External mode'. In this case, Simulink Coder will produce a native executable which can be executed on a target-machine (which can be the same machine as Simulink is running on). During execution it is possible to monitor and change the model-variable values via Simulink and log the obtained data. In general, a significantly higher execution performance can be achieved by compiling the Simulink diagram and executing it as a native application.

3.3.2 The MAVLink Protocol

Micro Air Vehicle Link (MAVLink) is an open-source communication-protocol designed for the communication with and between unmanned air vehicles [30]. The MAVLink protocol consist of a set of well

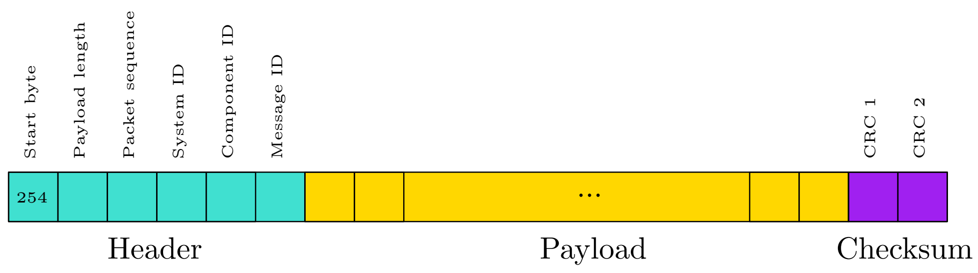


Figure 32: Schematic overview of a MAVLink message in byte form. Each MAVLink message has a header, payload, and checksum.

defined MAVLink messages. Each message is indicated with a name and a unique id-number. A messages is a small package of data and is designed to fulfill a specific purpose. A message can, for example, contain a state-update or a commando. MAVLink messages consist of an array of bytes. This array is divided in a header, a payload and a checksum as is shown in Figure 32. The payload contains the main information and follows a specific structure. The structure of the payload is declared in an .xml file. The message, in the form of an array of bytes, can be sent over many different communication channels, such as a serial port, USB, or Wifi channel. In Appendix D, the MAVLink protocol is described in detail. Not all technical details are described in the original online description, these details are directly derived from the c source-code in [31]. Below a summary of the protocol is given.

Message definitions The message-structure is defined in the extensible markup language (XML) format. The file(s), containing the message definitions, can be found on the MAVLink Github [31]. More specifically, the most used common messages and corresponding enums can be found in the 'common.xml' file. In addition, there are platform specific messages that are defined in separate files. In our setup, we only use the common messages. A message, in the XML-file, is typically indicated with a message id-number ($0 \ldots 255$), a description, and data fields of a specific type. Each field-type can occur in scalar-form or vector-form.

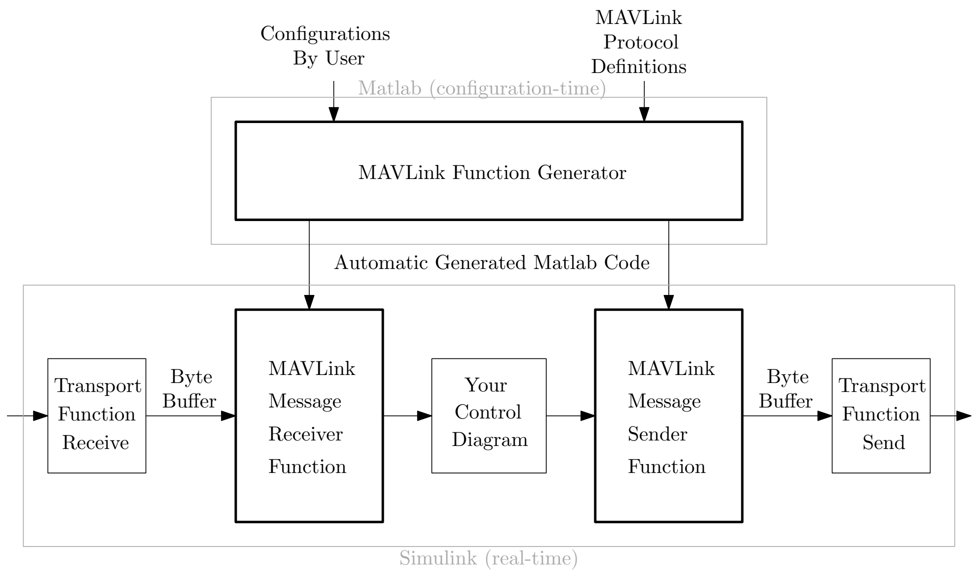
Message Construction A message, as defined above, must be 'packed' before it can be sent over a serial connection. A complete single MAVLink message is build up from an array of bytes. Each individual message contains a header (first 6 bytes), payload (n bytes), and a checksum value (2 bytes). The structure is visualized in Figure 32.

- **Header.** The header of a message is of fixed length (6 bytes). Each byte location has a specific function as is indicated in Figure 32.

- **Payload.** The payload contains the data-fields of the message. Because all data-fields are of fixed size, the payload has a fixed length for each message-id. Before the data-fields of a message are sent, they are sorted by field-type-byte-size as is explained in more detail in Appendix D.1.3.

- **Checksum.** The two checksum bytes on the end of the message are used for verifying that the message is correctly received. However, the checksum also verifies that the **semantics** of the messages is for both sides the same. In other words both sender and receiver use the same xml-file with the protocol definitions. This verification is accomplished by calculating the checksum also over an additional seed-number. The algorithm for calculating the checksum is described in [38], the Matlab equivalent is given in Appendix D.1.4.

- **Seed-number.** The seed-number is by itself a checksum calculated over the text in the message-definition in the xml-file. This means that when the message-id, a field-type, a field-name, or a field-vector-length changes the seed number for the specific message-id also changes. Since the seed number is accumulated in the checksum calculation for each sent or received message, both sender and receiver require to have the same xml file for proper communication. More details about the seed number calculation can be found in Appendix D.1.5.



Configurations By User
MAVLink Protocol Definitions
Matlab (configuration-time)
MAVLink Function Generator
Automatic Generated Matlab Code
Transport Function Receive
Byte Buffer
MAVLink Message Receiver Function
Your Control Diagram
MAVLink Message Sender Function
Byte Buffer
Transport Function Send
Simulink (real-time)

Figure 33: Schematic overview of a MAVLink function generation. Based on the MAVLink protocol definitions and user configurations the Matlab code for the real-time sender and receiver functions is generated. This configuration is required only once, thereafter the generated Simulink 'blocks' can be used in Simulink for real-time execution.

3.3.3 The MAVLink-Simulink interface

This section describes the MAVLink-Simulink interface as developed in this work. The interface consists of three components

1. MAVLink function generator
2. MAVLink message sender function
3. MAVLink message receiver function

where the two latter functions are automatically generated by the first one. The MAVLink message sender and receiver are the functions used for the real-time message handling, whereas the MAVLink function generator composes the code based on the user settings and message definitions and is only executed once. This process is schematically visualized in Figure 33.

Function Generator The MAVLink function generator as indicated in Figure 33, produces the Matlab code for the MAVLink Message Receiver and Sender functions. The code is generated based on the original MAVLink definitions and some specifications given by the user/designer. The MAVLink definitions are given in the form of the original 'common.xml' file. The required user input is a confined list of message-id numbers to be receive and to be sent in the form of a Matlab vector. Based on this input, the function generator composes the code for the receiver and the sender functions. This process is best explained by the following steps

1. The user specifies a list of MAVLink message-ids, of interest, that must be sent or received.
2. The MAVLink message definitions are read from the 'common.xml' file and converted to a Matlab struct.
3. For each message the seed-number is calculated, and the data-fields are sorted.
4. Thereafter, the code for the MAVLink receiver and sender is composed. The code is written as text to the Matlab function files, based on a set of if-then-else statements.
5. The final result are two regular, automatically generated, Matlab function files that can be used within Matlab or in Simulink.

Receiver Function The MAVLink receiver function consist of a piece of Matlab code generated by the MAVLink generator function. The receiver function recovers from a byte buffer the variable-values of the MAVLink messages. For example, when a message with an attitude update is received, this function transforms the message to, a roll, pitch, and yaw value that can be directly used within Simulink. The process is in detail described in Appendix D.4.

In short, during initialization the function output-variables are initialized with zero values. Thereafter the output-values are updated by the arriving messages. Between the arrival of messages the output variables are kept constant in the form of a zero-order hold function. Because messages can be divided over two or more time-samples, i.e., the first part of the message can be already received while the rest of the message is received at the next sampling time, a first in first out (FIFO) procedure is followed in an intermediate buffer. In order to detect a message in the buffer, the function searches for the message start indicator and verifies its checksum. Thereafter the payload is sorted and type-casted to the matching output-variable. Once the message is evaluated, the message, plus earlier data, is removed from the FIFO buffer.

Sender Function In order to send data via the MAVLink protocol, the MAVLink Sender function can be used. This function packs, constructs, and combines MAVLink messages, based on some configurable conditions. The different conditions for sending a MAVLink message are: send automatically, force a message to send, or disable sending. The send condition can be configured for each message separately by a send-mode variable. The following rules hold for the send mode

Mode	**Value**	**Action**
Forced	1	Send message at each execution
Automatic	0	Send when one or more values of the variables in a message changed
Disabled	-1	Do not send the message

In most situations the Automatic mode can be used. For example, when the Simulink diagram calculates the attitude-targets that must be followed by the flight-controller, this sender function checks if a target value is changed with respect to the previous time-sample. If the value is changed, in automatic send mode, the sender function will construct a MAVLink message, calculates its checksum, and provides it to the transport layer which will send it to the flight-controller. The internal structure of the sender function is explained, in detail, in Appendix D.4.

3.3.4 Simulink model

Figure 34 shows an example of how our MAVLink interface looks like in Simulink. The communication-interface is build from three blocks, the MAVLink-Sender, the serial-read-write port, and the MAVLink-Receiver, indicated by 34.A. 34.B. and 34.C respectively. In addition, to guarantee the real-time sample-frequency, a pacer function is used, indicated with 34.D. As can be observed there is a strict separation between protocol, transport-layer, and real-time implementation in Simulink. In principle all the functions can work standalone.

In our setup the communication is performed over a serial over USB connection, and uses a special function to guarantee real-time execution, as discussed below. In principle, besides using this Simulink model on the companion-computer, on the multi-rotor, it can also be used as ground-station when the USB connection is replaced by a wireless connection. The functions are developed for the Linux OS and for the MS Windows platform.

Serial read-write block The serial_read_write block performs the communication between the Simulink model and the serial-port. To the best of our knowledge, all the currently available Simulink serial-communication-blocks read and write only a fixed number of bytes from and to the serial port, or require specific start and stop indicators. The disadvantage of this approach is that for reading data from the serial port we have to wait until the input buffer is completely full before the data becomes available to the connected block. Since MAVLink messages are not necessary sent with a fixed sample rate, nor are of fixed length and do not use default start and stop indicators, it occurs that messages are significant delayed (in the order of seconds) when using the default Simulink functions.

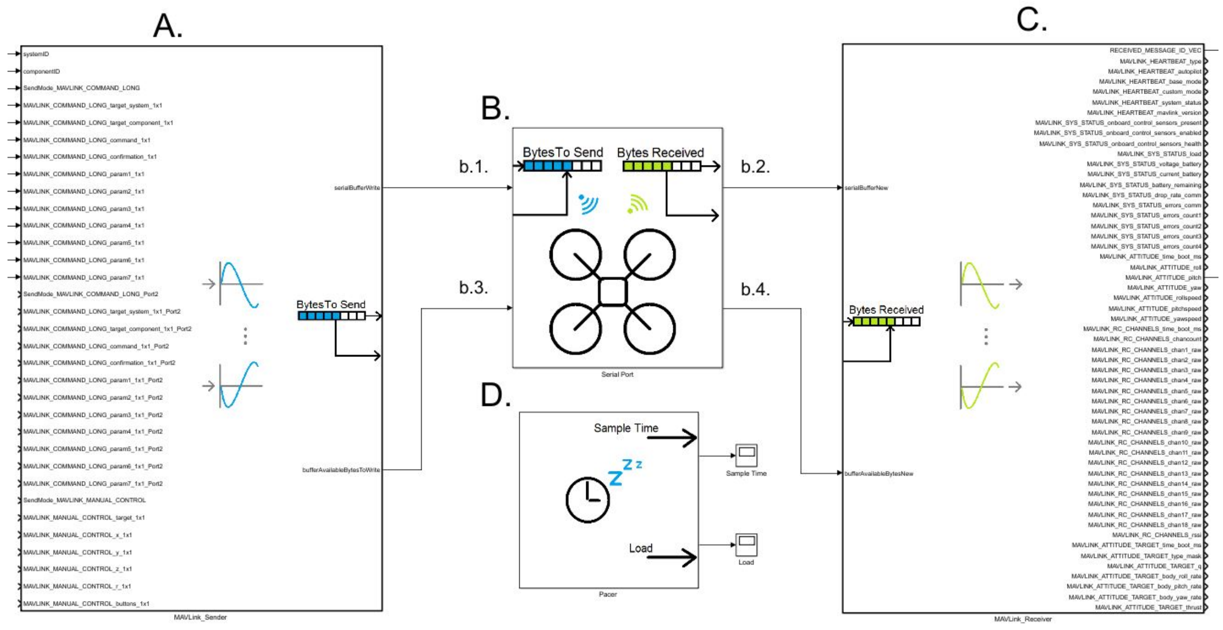


Figure 34: Example of the Simulink overview. A) Matlab-Function-Block to interact with the rest of the Simulink model and pack MAVLinkl messages. B). S-Function-block to send and receive data over a serial port. C). Matlab-Function-Block to unpack MAVLink messages and interface with the rest of the Simulink model. D. pacer block for controlling the sample rate.

In order to solve this problem and obtain a 'clean' real-time connection, we designed our own serial-port interface function in the form of a Simulink S-function written in c-code. This block receives all the current data available on the serial-port at each time-sample and sends all the created MAVLink messages at each time-sample. The detailed structure is given in Appendix D.4.

Pacer Block By default, Simulink executes the simulation-steps directly after each-other in order to show the simulation results as fast as possible to the user. However, for real-time execution of the model we want the simulation time to be synchronized with the real-time system clock. In principle, in Windows we can use the Real-Time Desktop Simulink target to achieve this. Since this target does not allow us to use the required 'windows.h' header/library in order to access the serial-port and there is, to our best knowledge, no default serial-port available with variable buffer length, this target can not be used. For Linux we have only the 'pace' block, from the Aerospace-Blockset, available for real-time model execution. A disadvantage of this 'pace' block is that it does not function when compiled in Simulink External mode.

For this reason we designed our own pacer-function in the form of a Matlab S-function in c-code, as is indicated by Figure 34.D. The pacer function is inspired by the 'libtimer_posix.c' library which is used in the E-Boxes [39] at TU/e. The pacer block is a wait function that puts the application into sleep mode until it is time for the next execution-iteration, such that the simulation-time matches the real-time system clock. A fixed sample-time is assumed. In order to guarantee that this function-block is executed after all other calculations during an iteration, the block priority is set to a lower priority compared to all other blocks in the Simulink model. More details are given in Appendix D.4.

3.4 Position From Computer Vision

In order to give the multi-rotor a notion of its position with respect to the world, computer-vision is used. In the last decades, computer-vision has gained more and more attention. In many disciplines computer-vision algorithms solved problems, examples can be found in medical scanners, the packaging industry, and person authentication using face recognition. Our interest is on algorithms determining the position of the camera with respect to the world. Impressive results are shown, for example, by the consumer products [40] and [41] where simultaneously a model of the world and the camera position is obtained on a mobile device. In this proof-of-concept we limited ourself to a relatively simple, yet

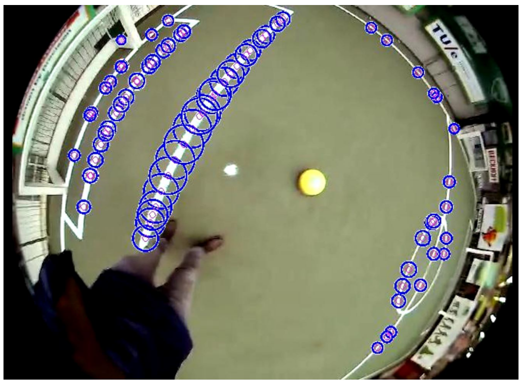


Figure 35: Detection of points on a line in a real-data video stream. Adaptive thresholding is used to seperate the white lines from the background. Based on a sparse grid sample-points are selected and checked for being on a line element. A point on a line is defined as the center of a circle having only two points divided by 180 degree on the circumference of the circle, as described in [43].

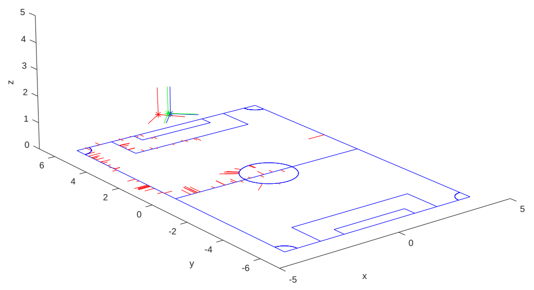


Figure 36: Simulation of localization on a soccerfield. In blue the field-lines are shown, in red the projection errors are indicated with respect to the nearest field-line. The red frame indicates the previous estimated position. The green frame indicates the estimated position using the levenberg-marquardt optimization method. The blue frame indicates the groundtruth.

robust, method for determining the multi-rotors position using computer-vision. One method relies on the detection of the lines on a soccer-field as is discussed in 3.4.1. The other method uses the detection of Aruco markers as is discussed in Section 3.4.2.

3.4.1 Localization on a soccer-field

In case the multi-rotor is used as soccer referee, the multi-rotor should be able to determine its position with respect to the soccer-field. Intentionally a high-performance deca-wave system should provide the position of the multi-rotor. However, due to circumstances this system is not operational at the time of writing. In an attempt to determine the position of the multi-rotor with respect to the soccer-field using computer vision, the fish-eye ground facing camera on the multi-rotor is used. The method is mainly derived from the source code of the soccer robots of the Tech United team [42]. In this case, in the image obtained from the camera, points on the lines of the field are detected, as is shown in Figure 35. Based on these line-points the position of the multi-rotor is determined. In short, the algorithm executes the following steps. First adaptive thresholding is used to separate the white lines from the background. Based on a sparse grid, sample-points are selected and checked for being on a line element. A point on a line is defined as the center of a circle having only two neighbor points divided by 180 degree on the circumference of the circle, which is known as the circle-check as is described in [43]. This method seems to show, at an abstract level, some parallels with the biologically inspired Combination Of Shifted FIlter REsponses (COSFIRE) which is proposed in [44]. After the detection of line-points, based on an initial estimate of the position, the line points are projected, in a virtual 3D environment, on a known map of the soccer field. Each line-point is matched to a field-line, based on its shortest distance, as is shown in Figure 36. Finally, iteratively, using a Levenberg-Marquardt optimization algorithm the projection error is minimized and a new estimation of the position is obtained. This approach is implemented in simulation on experimental data, however due to the risk that the development of a real-time implementation is too time consuming in this work, another method is implemented as is described in Section 3.4.2.

3.4.2 Position based on the detection of Aruco markers

Aruco markers [45] are square black-white 2D markers that can be detected by a computer vision algorithm with high confidence. The markers have a characteristic black on white border. An internal binary pattern is used to distinguish the markers from each-other and to determine its heading. In our setup, the Aruco markers are placed, in a pattern, on the floor of the flight test-area, as is shown in

Figure 37. The bottom facing 180° fish-eye camera mounted on the multi-rotor is used to detect the markers. Based on the position of the detected markers in the camera image, the horizontal position, heading, and even the height of the multi-rotor is determined. This method is already proved to work with a multi-rotor as is, for example, shown in [30]. Although Matlab and Simulink are powerful tools for prototyping computer-vision algorithms, for the real-time implementation of the (in general) computationally heavy vision algorithms other development platforms might be more qualified. It was noticed by the Autonomous Referee System [14] project that the implementation of computer-vision algorithms for real-time implementation in Simulink can be problematic. The real-time computer vision algorithms, in this work, are developed using c++ code together with the OpenCV library [35] containing computationally efficient computer vision algorithms. An important selection criteria is that the OpenCV library already provides an algorithm for detecting Aruco markers. In order to determine the position of the multi-rotor the following steps are required

1. **Camera calibration.** The method below, requires a model that describes how 3D object points appear in the 2D image obtained by the camera. The camera-model is described by a pinhole camera model with an additional radial correction in the form of a polynomial. During a calibration process the model parameters are estimated. More details about the camera model are for example given in [46].

2. **Detection of Aruco markers.** The Aruco markers, or more precisely the corners of the Aruco markers, are detected in the 2D image using the method as is described in [45].

3. **Pose estimation.** The real world 3D positions of the Aruco markers, as shown in Figure 37, are fixed and assumed to be known. Also the corresponding 2D positions of the Aruco markers in the image are known. Together with the calibrated camera model it is possible to estimate the position of the camera with respect to the markers. In general an iterative method is used that, based on an initial guess, minimizes the projection error and simultaneously obtains the position of the camera. This process is, for example, described in [46]. The method that is used in this work is part of the OpenCV library.

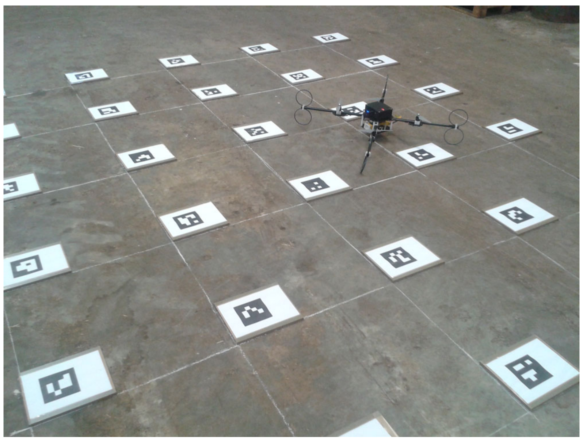


Figure 37: Field of Aruco markers on the floor of the test area. The multi-rotor in flight detects the markers with the bottom facing fish-eye camera and estimates its 3D position based on the markers. Noteworthy is that besides X-Y position and heading also the height is measured visually.

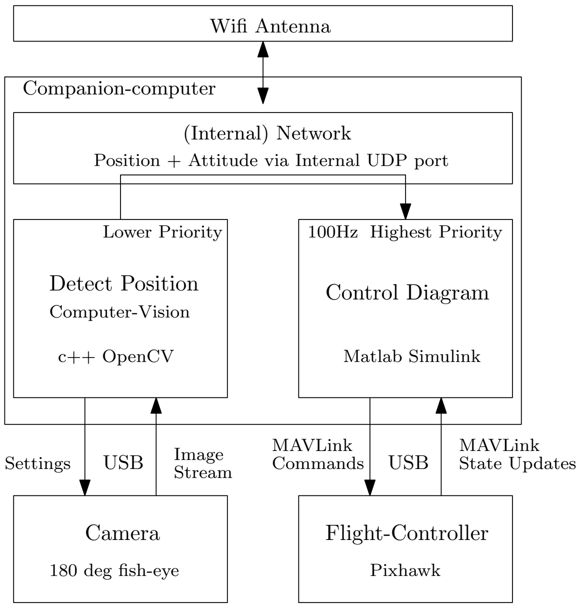


Figure 38: Schematic representation of the integration of the different software components. The structure is basically a feedback loop, where the computer vision algorithm performs as a sensor providing a position estimate to the Simulink control diagram.

3.5 System Integration

In this section, the integration of the different software components is described. In contrast to the autonomous referee system [14], where a system architecture is developed for among others a multi-rotor agent, in this work the focus is on performing a proof-of-concept. The proof-of-concept is to let the multi-rotor perform an autonomous flight. In this work the word autonomously is interpreted as receiving no control or sensor signals from an external system, i.e., all the sensing and control is executed on-board, thus no top-camera is for example used. Although we do not rely on a system-architecture for complex system behavior such as the task-skill-motion behavior model as is used in [14], the following model can be kept in mind. Schematically the structure is shown in Figure 38. The real-time Simulink model, running on the companion-computer, performs the communication with the flight-controller using the MAVLink protocol. This Simulink model calculates the target references for the the low-level flight-controller. In parallel, also on the companion-computer, the computer-vision algorithm runs, determining continuously the position of the multi-rotor with respect to the field of Aruco markers. In order to share the position results with the real-time Simulink model, a form of inter process communication is required. Although there are multiple solutions for inter process communication, in this proof-of-concept a relative simple, however flexible, method is used based on an internal UDP connection. The computer vision algorithm broadcasts its results to an internal port from which the real-time Simulink model is able to read. The computer vision algorithm can thus be considered as a advanced sensor providing the most recent position of the multi-rotor. In this form a feedback loop is created, where the real-time Simulink model is considered as the controller.

3.6 Summary, evaluation, and remarks

In this section the software peripherals allowing users to develop their own software were described. The communication between the Intel Nuc companion-computer and the Pixhawk flight-controller follows the MAVLink protocol. The MAVLink protocol is used to send and receive instructions and state-updates. A complete, real-time, and bidirectional implementation for Matlab Simulink was missing. In order to solve this problem, in this work, a MAVLink-Simulink interface is developed as is described in Section 3.3. The MAVLink-Simulink interface, at protocol level, consist of three functions: a generator-function, a MAVLink receiver function, and a MAVLink sender function. The latter two are automatically generated by the generator-function which produces 'clean' Matlab code based on the original MAVLink definitions and some user configurations. It is therefore robust to 'minor' future protocol changes. During the development process, a clear distinction is maintained between the protocol, transport layer, and real-time implementation. In order to perform the transport of the MAVLink messages and real-time implementation, special S-functions were developed. The MAVLink-Simulink interface is able to work with Linux as well as with MS Windows. The interface is proven to work in Linux up to at least 1000Hz and in MS Windows at least 50Hz can be reached. Noteworthy, due to the clear separation of protocol and transport-layer, a professional drone company is currently working with the MAVLink protocol layer developed in this work.

In order to perform a proof-of-concept, which is in our case the multi-rotor performing an autonomous flight, a position estimation algorithm is developed. Since we define autonomous, in this work, as receiving no control signals or position feedback from an external system, the position estimation is purely based on computer vision and executed on-board. As was discussed in Section 3.4 first the position is estimated by detecting the lines on a soccer-field using the 180° fish-eye camera. The algorithm is tested in simulation based on experimental data. Because we did expect that the real-time implementation of this algorithm would take too much time in this project, a more simple method is implemented based on a field of Aruco markers. Because of potential performance issues, the algorithm is written in c++ and is using the OpenCV library.

As is described in Section 3.5, the individual components are integrated in the form of a feedback loop. The Simulink model, performing the control calculations and communication with the flight-controller via the MAVLink-Simulink interface, runs in real-time in Simulink's external mode. In parallel, also on the companion-computer, runs the computer vision algorithm, estimating the multi-rotor's position and broadcasting the results via an internal UDP port to the Simulink model. The computer vision algorithm reaches a sample-frequency of $> 30Hz$ (limited by the camera), for an image of 1024×768 pixels. Worth noting is that, without significant changes, the same c++ code, including the detection, position estimation, and UDP socket is now/temporary being used in an application of the honors class

[47]. The focus of the vision algorithms, in this work, is to perform the proof-of-concept of an autonomous flight, future work might be to integrate the system-architecture developed in the autonomous referee system [14].

4 Path Following Control for a Multi-Rotor

In this section a path-following controller is proposed for a multi-rotor moving in a 3D space. In path-following a path is followed, in the sense that the vehicle's position coincides (or is close to) a point in the path with a, path depended, prescribed velocity along a path. Compared to trajectory tracking no hard time constraints are considered along the path. However, the fact that a velocity reference is given along the path allows to meet soft time constraints. In general, path following is often used for under actuated, non-linear systems such as an 2D unicycle-type of vehicle as (first) proposed in [48]. Later, in many other applications path following is applied in 3D space as is, for example, described for an underwater vehicle in [49]. In this chapter, we start by discussing the nonlinear equations of motion of the multi-rotor which suggest a natural cascaded control approach: the motor speeds, applied to the multi-rotors's propellers, control the magnitude of the 3D thrust/force vector of the multi-rotor and generate torques which control its direction. The 3D force vector controls the motion of the multi-rotor, which from Newton's law follows a simple double integrator model. With the aid of singular perturbation theory, the control problem is separated in a fast and a slow control problem (inner and outer-loop respectively). Although, Appendix E.6 describes the development of a low-level LQR controller, and Section 4.4 a feedback linearization controller for the stabilization of the attitude, the main focus in this chapter is on the slow control problem (path following), and therefore it considers a simple double integrator model. The proposed control law, in this work, is derived from the usual Frenet-Serret formulas. However, contrarily to previous works [49][50], in this work, the control law is directly expressed in the inertial frame. The path following control strategy is proved to have convergent behavior, in the sense that the vehicle's distance to the path converges to zero and the vehicle's velocity converges to the desired velocity along the path. A crucial step in path-following, where the reference-point on the path is specified based on the current position of the multi-rotor, is to keep track of the path arclength from the origin of the path which in many cases amounts to finding the point along the path with the shortest distance to the vehicle. Often, for complex paths, there is no analytical solution that determines the reference-point on the path. With straightforward methods it is not always guaranteed that the reference-point on the path is continuous. In this work a computationally efficient method is proposed based on a local second-order Taylor polynomial estimation for which an analytically solution exists, which provides the shortest distance to the path. Finally is described how the double integrator system for the outer-loop is derived, based on singular perturbation theory, and model simplification.

The remainder of this chapter is organized as follows. In Section 4.1.1, the nonlinear equations of motions derived from the Newton-Euler equations are presented. In Section 4.1.2 the cascaded control structure of the multi-rotor is described. The Frenet-Serret formulas are described in Section 4.2.3 and Section 4.2.4 provides some insights on the control law. The new method to keep track of the reference point along the path is discussed in Section 4.2.7. Furthermore, Section 4.2.8 describes a proposed control law for guiding the heading of the multi-rotor along the path. In addition, a cubic-spline is used in order to merge separate way-points into a continuous path, as is discussed in Section 4.3. The complete system is evaluated in simulation and the numerical results are graphically presented in Section 4.2.9 and Section 4.3.2. In Section 4.4 the inner-loop stabilization by means of feedback linearization control is described, and the numerical simulation results are presented. Furthermore, in Section 4.5 the stability properties of the combined control system with the inner and outer loop is discussed based on singular perturbation theory.

4.1 Physical Model and Control Structure

In Section 4.1.1 the nonlinear equations are described, and in Section 4.1.2 cascaded structure of the multi-rotor is revealed, leading to two control problems for the inner and outer-loop. As we shall see the outer-loop problem, accounts to controlling a double integrator system which we we will tackle with path following.

4.1.1 Equations of motion

The dynamical model of the multi-rotor can be derived from the Newton-Euler equations as is for example presented in [51]. The multi-rotor is modeled as a rigid-body with four rotors. A schematic representation of the multi-rotor model is shown in Figure 39 represented in the north east down (NED) frame convention. With only four actuators (4 rotors) and six degrees of freedom (3 spacial and 3 rotational) the system is under actuated. The non-linear dynamical equations are given by

$$\dot{p}_A = v \tag{5}$$

$$m\dot{v}_A = -R\,(\lambda)\,T\,e_3^B + m\,g\,e_3^A \tag{6}$$

$$\dot{\lambda} = Q\,(\lambda)\,\omega \tag{7}$$

$$J\dot{\omega} = \tau + \omega \times J\omega \tag{8}$$

where $p_A = [p_{A,x} \quad p_{A,y} \quad p_{A,z}]^T$, $v_A = [v_{A,x} \quad v_{A,y} \quad v_{A,z}]^T$, $\lambda = [\phi \quad \theta \quad \psi]^T$, $\omega = [\omega_x \quad \omega_y \quad \omega_z]^T$, and $\tau = [\tau_x \quad \tau_y \quad \tau_z]^T$ represents the systems position, velocity, Euler angles, and the body angular velocities, respectively. More specifically, ϕ, θ, and ψ represent the roll, pitch, and yaw angle respectively. The expressions for the rotation matrix $R(\lambda)$, the matrix $Q(\lambda)$, and inertia matrix J are given in Appendix E.1. Furthermore, g represents the gravitational acceleration constant, and m the multi-rotors's mass. The gyroscopic effects can be compensated by introducing an input $\tau = \bar{\tau} - \omega \times J\omega$ where $\bar{\tau} = [\bar{\tau}_x \quad \bar{\tau}_y \quad \bar{\tau}_z]^T$ is a new input. Four rotors propel the multi-rotor. The independent thrust T_i and torque τ_i of each rotor i is quadratically related to the rotor angular-velocity w_i in the from

$$T_i = cw_i^2 \qquad i \in \{1, 2, 3, 4\} \tag{9}$$

$$\tau_i = (-1)^i dw_i^2 \qquad i \in \{1, 2, 3, 4\} \tag{10}$$

where c and d are the thrust-constant and drag-constant respectively. The independent rotor contributions can be combined via a mixing matrix Γ such that

$$\hat{u} = \Gamma\,\hat{\omega}^2 \tag{11}$$

where $\hat{u} = \begin{bmatrix} T & \tau_x & \tau_y & \tau_z \end{bmatrix}^T$ are the combined body thrust and torques, $\hat{\omega} = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 \end{bmatrix}^T$ are the 'real' system inputs, and the invertible mixing matrix Γ is given in Appendix E.2. Because of the property that Γ is invertible the control inputs can be calculated in terms of body thrust and torques from which via

$$(\hat{\omega})_i = (\sqrt{\Gamma^{-1}\hat{u}})_i \qquad i \in \{1, 2, 3, 4\} \tag{12}$$

the independent rotor angular-velocities can be obtained, where $()_i$ denotes the ith component.

4.1.2 Separation of dynamics

The nonlinear equations of motion (5)-(8) reveal a cascaded structure. This structure appears more clearly by introducing a variable F_A representing the forces applied to the rigid body

$$F_A = -R(\lambda)Te_3^B + mge_3^A \tag{13}$$

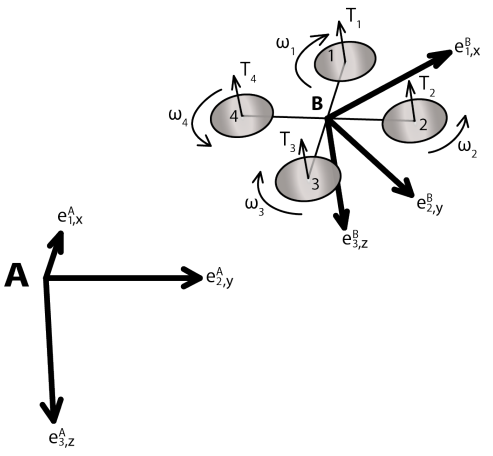


Figure 39: Schematic representation of the quadrotor. Indicated is the body-frame B and the inertial frame A. T_i represents the thrust and ω_i the rotation direction of rotor i.

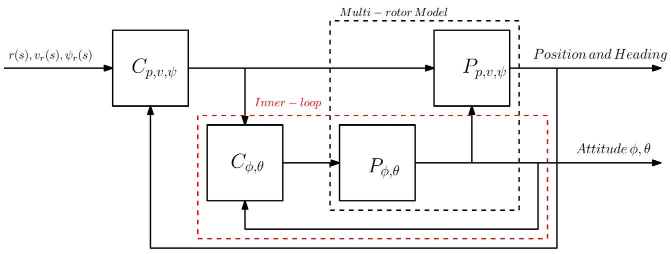


Figure 40: Abstract schematic representation of the control scheme. The roll ϕ and pitch θ stabilzation of the multirotor is concidered as the inner-loop. The inner-loop follows a reference specified by the outer-loop.

where $F_A = [F_x, F_y, F_z]^T$ such that the dynamics (5)-(8) can be separated in dynamics describing the position and velocity of the multi-rotor

$$\dot{p}_A = v \tag{14}$$

$$\dot{v}_A = \frac{1}{m} F_A \tag{15}$$

and the dynamics describing the attitude of the multi-rotor

$$\dot{\lambda} = Q(\lambda)\,\omega \tag{16}$$

$$\dot{\omega} = J^{-1}\bar{\tau} \tag{17}$$

The attitude dynamics are nonlinear, however, independent of the vehicle position and the velocity. The position dynamics depend, via F_A, on the attitude and reveal a linear double integrator structure between the position p and force F_A. Furthermore, the attitude dynamics are often considered as the fast dynamics and the position is considered as the slow dynamics. This cascaded structure and separation of time-scales is often exploited in the control of a multi-rotor in the form of sequential loop closing. The attitude dynamics, with relative low moment of inertia J, are often separately controlled by a high gain feedback controller. This feedback control-loop is often considered as the inner-loop. The much slower position and velocity dynamics are controlled by an outer-loop, as it is visualized in Figure 40. The purpose of the inner-loop is to track a force F_{ref} prescribed by the outer-loop with minimal error. By applying proper control on the inner-loop, under certain conditions, the fast inner-loop dynamics can be approximated by a direct feed-through $F_A = F_{ref}$, explained by singular value theory, as will be discussed in Section 4.5. The remaining dynamics in the outer-loop, including the inner-loop, can then be approximated by a double integrator structure. The outer-loop will be stabilized by means of path following as is discussed in Section 4.2.

4.2 Path following

4.2.1 Cascaded control structure

The control design for a multi-rotor often exploits the cascaded structure of the equations of motion (14)-(17) as is described in Section 4.1, and consists of two separate loops: a fast inner and a slow outer-loop. The inner-loop, addresses the attitude control (roll, and pitch) of the multi-rotor, and the outer-loop addresses to the position, velocity, and heading control of the vehicle.This section focuses on the outer-loop velocity and position control. Assuming the inner-loop has a sufficient high bandwidth the dynamical model including the inner loop is assumed to be

$$\dot{p}(t) = v(t) \tag{18}$$

$$\dot{v}(t) = u(t) \tag{19}$$

where $p = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T$ represents the position of the multi-rotor, $v = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T$ the body velocity of the multi-rotor, and $u = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix}^T = \frac{1}{m} F_{ref}$ the control input in the form of a scaled reference force.

4.2.2 Pathfollowing

In path-following the vehicle is required to follow a path

$$r(\xi) = \begin{bmatrix} r_x & r_y & r_z \end{bmatrix}^T \tag{20}$$

without a hard time constraint. More specifically, the path has to be followed in the sense that the vehicle's position should coincide with one of the points in the path with a specified velocity profile $v_r(\xi)$ along the path. This contrasts with trajectory-tracking, where, the reference on the path is directly specified by time. In general this works very well when the vehicle is close to its reference. However, when the distance to the reference is increasing, problems can occur due to for example wind gusts, input saturation, or any other distortion. In Figure 41 an example of such a potential situation is given where the vehicle has to move through a door. Figure 41.A shows the vehicle lagging behind the reference resulting in a collision when the path is not re-planned in time. In Figure 41.B the path-following approach is illustrated. In this case, the reference is a function of the vehicle's position. Instead of a time constraint,

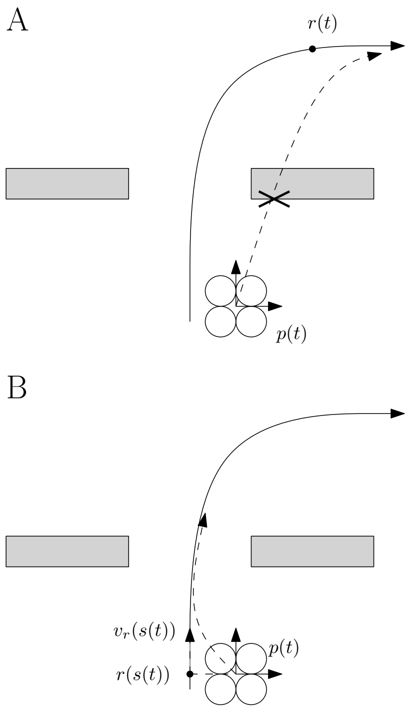


Figure 41: Illustration of multi-rotor passing a door. In A the consequence of lagging behind the reference in the case of trajectory tracking is sketched. In B the solution of path-following is sketched.

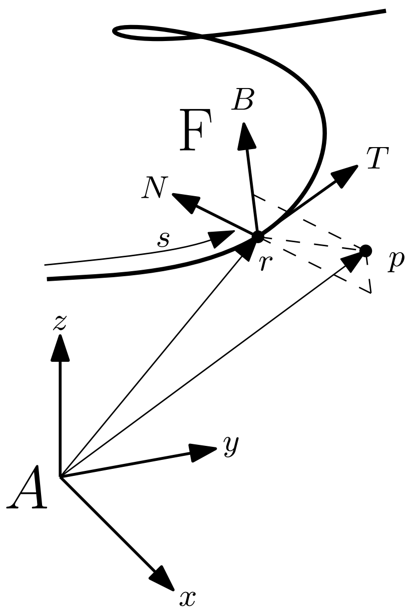


Figure 42: Example image of Frenet-Serret frame F positioned on a three-dimensional curved path.

a velocity reference tangent to the path is given, and the vehicle attempts to move to the point on the path with shortest distance to the vehicle (or more formally to the vehicle position dependent reference point on the path parameterized by the arclength along the path with respect to the path's origin as discussed below). This reduces the position error to the path preventing collisions in this illustrative example. Another advantage of path-following, compared to trajectory-tracking, is that when the vehicle is in front of the reference the vehicle does not have to move backwards, a potential forward/backward oscillation will be avoided. A disadvantage of path-planning is that, without adaptation of the velocity along the path, it can not be guaranteed that the vehicle arrives at a specific location at a specific time. An everyday intuitive example of path-following can be found in traffic. In this context the roads specify the path and the speed-limit signs specify the velocity profile along the path assuming the vehicle wishes to travel at the maximum allowable speed. In case of for example a traffic-jam the vehicle can continue its way with maximum allowed speed once the disturbance is vanished, and the only consequence is its delay in time of arrival.

4.2.3 Frenet - Serret formulas

A path can be specified by a general parameterization $\bar{r}(\xi)$ where ξ belongs to an interval $\xi \in [\underline{\xi}, \bar{\xi}]$. A natural parameterization of the path is in terms of the arc-length $s = f(\xi)$, which can be obtained by

$$s(\xi) = \int_{\underline{\xi}}^{\xi} \| \frac{\partial \bar{r}(a)}{\partial \xi} \| da \tag{21}$$

which is an invertible map, provided that $\frac{\partial \bar{r}}{\partial \xi} \neq 0$ in non-degenerate ξ paths. Note that $s \in [0, \bar{s}]$, where

$$\bar{s} = \int_{\underline{\xi}}^{\bar{\xi}} \| \frac{\partial \bar{r}(a)}{\partial \xi} \| da. \tag{22}$$

For non-degenerate curves $r\,(s)$, where r is such that $\bar{r}(\xi) = r(s(\xi))$, without zero curvature, the following right handed orthogonal frame F at position $r(s)$ can be defined

$$T(s) = \frac{\frac{\partial r(s)}{\partial s}}{\|\frac{\partial r(s)}{\partial s}\|} \tag{23}$$

$$N(s) = \frac{\frac{\partial T(s)}{\partial s}}{\|\frac{\partial T(s)}{\partial s}\|} \tag{24}$$

$$B(s) = T(s) \times N(s) \tag{25}$$

where $T(s)$, $N(s)$, and $B(s)$ are known as the normalized tangent, normal, and bi-normal vectors respectively, and the $\times$ operator represents the vector cross-product. This frame, known as the Frenet-Serret frame, is visualized in Figure 42. The orientation of the Frenet-Serret frame with respect to the world-frame A is specified by the rotation matrix

$$R_F(s) = \begin{bmatrix} T(s) & N(s) & B(s) \end{bmatrix} \tag{26}$$

We can then write the position of the vehicle expressed in A, denoted by p, in terms of the position of the frame F denoted by $r(s)$ where s is a function of p as specified in the sequel as

$$p(t) = R_F(s(t))d(t) + r(s(t)). \tag{27}$$

where d is the position of the vehicle expressed in the frame F. The Frenet-Serret formulas specifying the frame F are

$$\frac{\partial T(s)}{\partial s} = \kappa(s)N(s) \tag{28}$$

$$\frac{\partial N(s)}{\partial s} = -\kappa(s)T(s) + \tau(s)B(s) \tag{29}$$

$$\frac{\partial B(s)}{\partial s} = -\tau(s)N(s) \tag{30}$$

where $\kappa(s) = 1/r_\kappa(s)$ represents the curvature and $r_\kappa(s)$ the radius, and $\tau(s)$ represents the torsion.

Position on the path For path-following it is crucial to define a reference point on the path, denoted by $r(s(t))$, where $s(t)$ is a parameter representing a vehicle position dependent path arclength, which will be defined shortly, and is in general a function of $p(a)$, $a \in [0, t]$. For clarity of notation the dependence of $s(t)$ on $p(a)$, $a \in [0, t]$ is omitted. However in many cases, the parameter $s(t)$ is determined immediately by the *current* position $p(t)$ of the vehicle and corresponds to the point on the path with the shortest distance to the vehicle. In general, this is where the euclidean distance between $p(t)$ and $r(s(t))$ is minimal, yielding the formula

$$s\,(t) = \arg\min_{a \in S} \frac{1}{2}\|p\,(t) - r(a)\|^2 \tag{31}$$

where $\|x\|^2 = x^T x$ is the squared euclidean norm and $S \subseteq [0, \bar{s}]$. This equation (31) can be used to define and compute $s(t)$ for paths for which (31) has a unique solution in this interval, such as straight lines, in which case we can take $S = [0, \bar{s}]$. However, for general paths, (31) can have multiple solutions and the desired parameter $s(t)$ will correspond to a local solution, i.e., (31) will hold locally for $S = [s(t) - \epsilon, s(t) + \epsilon]$ for a small positive ϵ. Still, this local condition will allows us to formally define $s(t)$ for general paths by computing its vehicle's position dependent derivative $\dot{s}(t) = g(p(t), \dot{p}(t), s(t))$ and formally defining $s(t)$ in terms of g as

$$s(t) = s(0) + \int_0^t \dot{s}(a)da = s(0) + \int_0^t g(p(a), \dot{p}(a), s(a))db$$

We assume that $s(0)$ is fully determined by $p(0)$ and satisfies

$$s\,(0) = \frac{1}{2}\|p\,(0) - r(s(0))\|^2 \tag{32}$$

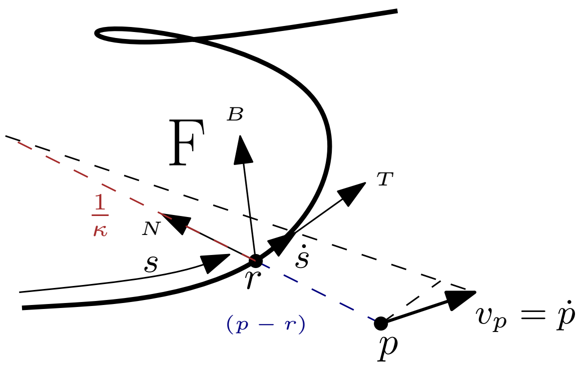


Figure 43: Graphical representation of the relation (37) between $\dot{p}$ and $\dot{s}$. This relation describes how $s(t)$ chances under the influence of a change in $p(t)$.

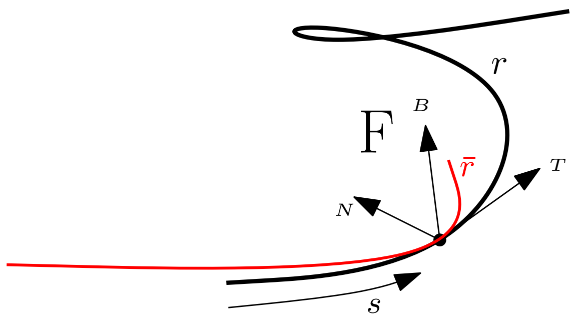


Figure 44: Schematic illustratio of the second order Taylor polynominal approximation $\bar{r}$ of the path r.

in such a way that typically $r(s(0))$ is the point along the path with shortest distance to $p(0)$. To compute $\dot{s}$ let us start by noticing that, since this is a semi-positive-definite quadratic function the minimum can be found in the point where its derivative is zero. By taking the derivative with respect to s the relation

$$0 = \Big(p(t) - r(s(t))\Big) \cdot \frac{\partial r(s(t))}{\partial s} \tag{33}$$

is obtained where $\cdot$ represents the dot product. The above equation states that the shortest distance to the path is where the projection of vector $p(t) - r(s(t))$ onto the tangent to the path is zero. Using (23) yields

$$0 = \Big(p(t) - r(s(t))\Big) \cdot T(s(t)). \tag{34}$$

In general this equation is difficult to solve analytically. In Section 4.2.7 a more practical approximate solving method is proposed. However, from this equation the dynamical relation between $\dot{p}(t)$ and $\dot{s}(t)$ can be obtained by taking its time derivative

$$0 = \dot{p}(t) \cdot T(s(t)) + p(t) \cdot \frac{\partial T(s(t))}{\partial s(t)} \dot{s}(t) - \frac{\partial r(s(t))}{\partial s(t)} \cdot T(s(t))\, \dot{s}(t) - r(s(t)) \cdot \frac{\partial T(s(t))}{\partial s(t)} \dot{s}(t) \tag{35}$$

$$0 = \dot{p}(t) \cdot T(s(t)) + p(t) \cdot \kappa N(s(t))\, \dot{s}(t) - \dot{s}(t) - r(s(t)) \cdot \kappa(s(t))\, N(s(t))\, \dot{s}(t) \tag{36}$$

$$\dot{s}(t) = \frac{\dot{p}(t) \cdot T(s(t))}{1 - \kappa(s(t)) \Big(p(t) - r(s(t))\Big) \cdot N(s(t))} \tag{37}$$

where $N(s(t))$ is pointing towards the center of curvature. For clarity this relation is graphically illustrated in Figure 43. It can be observed that there is a singularity when $p(t)$ is at the center of rotation. This singular point must be avoided. Note that at points on the curve where $\kappa(s(t)) = 0$, which is for example the case at an inflection point or a straight line, the normal $N(s(t))$ is undefined, and (37) boils down to

$$\dot{s}(t) = \dot{p}(t) \cdot T(s(t)). \tag{38}$$

Note that, for the sake of completeness, we assume that when $\kappa(s(t)) = 0$, $N(s(t))$ is orthogonal to $T(s)$ and $\begin{bmatrix}0 & 0 & 1\end{bmatrix}^T$, i.e.,

$$N(s) = T(s) \times \begin{bmatrix}0\\0\\1\end{bmatrix} \tag{39}$$

when $\kappa = 0$.

4.2.4 Defining the control objective

Following the path means that the vehicle's position coincides with a point on the path and the vehicle's velocity coincides with a prescribed velocity in tangent direction to the path. The following control-errors are introduced

$$e_p(t) = p(t) - r(s(t)) \tag{40}$$
$$e_v(t) = v(t) - v_r(s(t))T(s(t)) \tag{41}$$

where $e_p = \begin{bmatrix} e_{p,x} & e_{p,y} & e_{p,z} \end{bmatrix}^T$ is the position error, and $e_v = \begin{bmatrix} e_{v,x} & e_{v,y} & e_{v,z} \end{bmatrix}^T$ is the velocity error, and $v_r(s)$ is the reference velocity profile specified along the path expressed in the inertial frame A. By transforming the position errors to the Frenet-Serret frame it can be observed that by definition the position error in the tangent direction is $e_{p,T}(t) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} R_F^T(s(t))e_p(t) = 0$ which is a result of (34). By transforming e_v to the Frenet-Serret frame we define $e_{v,T} = v_T - v_r(s(t))$, $e_{v,N} = v_N$, and $e_{v,B} = v_B$ where

$$\begin{bmatrix} v_T \\ v_N \\ v_B \end{bmatrix} = R_F^T v. \tag{42}$$

The control objective is to minimize (40) and (41).

Conceptually the parameter $s(t)$ can be determined by running (37) on-line. However, in practice, due to noise and model disturbances, running (37) would lead to inaccurate results since it would entail integrating noise. Therefore, keeping track of approximations of $s(t)$ and computing locally (31) will yield much more accurate results from a practical point of view and this motivates the new proposed method to compute (31) based on local Taylor expansion which will be discussed in the sequel.

4.2.5 Control law

The following control rule is proposed to stabilize the system

$$u = -k_1\Big(v(t) - v_r(s(t))T(s(t))\Big) - k_2\Big(p(t) - r(s(t))\Big) + \frac{\partial v_r(s(t))}{\partial s}T(s(t))\dot{s}(t) + v_r(s(t))\frac{\partial T(s(t))}{\partial s}\dot{s}(t) \tag{43}$$

which can be rewritten using the Frenet-Serret formulas as

$$u = -k_1\Big(v(t) - v_r(s(t))T(s(t))\Big) - k_2\Big(p(t) - r(s(t))\Big) + \frac{\partial v_r(s(t))}{\partial s}T(s)\dot{s}(t) + v_r(s(t))\kappa N(s(t))\dot{s}(t) \tag{44}$$

where $\frac{\partial v_r(s(t))}{\partial s}T(s)\dot{s}(t)$ represents the feed-forward term for the change in the velocity profile along the path, and $v_r(s(t))\kappa N(s(t))\dot{s}(t)$ represents the feed-forward term for the centrifugal force. The two other feed-back terms are better interpreted when expressed in the Frenet-Serret frame. The first term

$$-k_1 R^T(s(t))\Big(v(t) - v_r(s(t))T(s(t))\Big) = -k_1 \begin{bmatrix} v_t - v_r(s(t)) \\ v_N \\ v_B \end{bmatrix} \tag{45}$$

is a feedback term to make the tangential velocity converge to v_r and the normal and bi-normal velocity to zero. The term

$$-k_2 R^T(s(t))\Big(p(t) - r(s(t))\Big) = -k_2 \begin{bmatrix} 0 \\ d_N \\ d_B \end{bmatrix} \tag{46}$$

is a feed-back term to drive the distance to the path to zero. The control gains have values $k_1 > 0$, and $k_2 > 0$.

4.2.6 Convergence proof

Lyapunov Consider a Lyapunov function of the form

$$V(e_v(t), e_p(t)) = \frac{1}{2}\|e_v(t)\|^2 + k_2\frac{1}{2}\|e_p(t)\|^2 \tag{47}$$

which can be written as

$$V(e_v(t), e_p(t)) = \frac{1}{2}\|v(t) - v_r(s(t))T(s(t))\|^2 + k_2\frac{1}{2}\|p(t) - r(s(t))\|^2 \tag{48}$$

Taking the time derivative of V we obtain

$$\begin{aligned}\dot{V}(e_v(t), e_p(t)) = &\Big(v(t) - v_r(s(t))T(s(t))\Big) \cdot \Big(\ddot{p}(t) - \frac{\partial v_r(s(t))}{\partial s}T(s(t))\dot{s}(t) - v_r(s(t))\frac{\partial T(s(t))}{\partial s}\dot{s}(t)\Big) \\ &+ k_2\Big(p(t) - r(s(t))\Big) \cdot \Big(\dot{p}(t) - \frac{\partial r(s(t))}{\partial s}\dot{s}(t)\Big)\end{aligned} \tag{49}$$

Substitution of the system-dynamics (18) with the control law (43) and using (37) yields

$$\dot{V}(t) = -k_1\|v(t) - v_r(s(t))T(s(t))\|^2 \tag{50}$$

$$= -k_1\|e_v(t)\|^2 \tag{51}$$

which is only semi-negative definite. Therefore to establish convergence we resort to La Salle's theorem [52].

La Salle The error dynamics can be written as

$$\dot{e}_v = \ddot{p}(t) - \frac{\partial v_r(s(t))}{\partial s}T(s)\dot{s}(t) - v_r(s(t))\frac{\partial T(s(t))}{\partial s}\dot{s}(t) \tag{52}$$

Substitution of the system-dynamics (18) with the control law (43) and using (37) yields

$$\dot{e}_v(t) = -k_1\Big(v(t) - v_r(s(t))T(s(t))\Big) - k_2\Big(p(t) - r(s(t))\Big) \tag{53}$$

$$= -k_1 e_v(t) - k_2 e_p(t) \tag{54}$$

from which can be concluded that $e_v(t)$ remains only zero when $e_p(t)$ is zero. Therefore the controlled system is globally asymptotically stable [52].

4.2.7 Path Position by Taylor Polynomial Approximation

Solving (34) to obtain $s(t)$ is in many cases impractical. In turn, using numerical methods in order to detect zero crossings is often a time consuming task, which is not ideal in real-time control applications. In addition, we can use the information that $s(t)$ is close to $s(t - t_s)$, where t_s is a small time step. In other words the method should avoid hopping between trajectories, as can be the case, for example, for a spiral form path with low pitch, a self intersecting path, or an overlapping path. As discussed before, for obtaining $s(t)$ integration equation (37), is also an option, and at least conceptually works in simulation. However due to inevitable disturbances occurring in practice, integration errors can be expected to result in a biased estimate of $s(t)$.

Taylor polynomial approximation In this work we propose a method that solves locally equation (34) for $s(t)$ by the approximation of a Taylor polynomial around $s(t - t_s)$. Based on this Taylor polynomial approximation an analytic solution for $s(t)$ close to $s(t - t_s)$ can be calculated in a short period of time. Consider the Taylor polynomial approximation of $r(s(t))$ around the point $s(t - t_s)$

$$r(s(t)) = r(s(t - t_s)) + \sum_{n=1}^{\infty}\frac{1}{n!}\frac{\partial^n r(s(t - t_s))}{\partial s^n}\Big(s(t) - s(t - t_s)\Big)^n \tag{55}$$

Here only the first three terms are taken into account, yielding the quadratic equation

$$\bar{r}(s(t)) = a_r + b_r\Delta s + \frac{1}{2}c_r(\Delta s)^2 \tag{56}$$

where

$$a_r = r(s(t - t_s)) \tag{57}$$

$$b_r = \frac{\partial r(s(t - t_s))}{\partial s} \tag{58}$$

$$c_r = \frac{\partial^2 r(s(t - t_s))}{\partial s^2} \tag{59}$$

$$\Delta s = \Big(s(t) - s(t - t_s)\Big). \tag{60}$$

and the dependence on t by Δs is omitted for clarity of notation. Substitution of (56) into (33) leads to the cubic function

$$0 = A_{rt}(\Delta s)^3 + B_{rt}(\Delta s)^2 + C_{rt}\Delta s + D_{rt} \tag{61}$$

where

$$A_{rt} = -\frac{1}{2}c_r \cdot c_r \tag{62}$$

$$B_{rt} = -\frac{3}{2}b_r \cdot c_r \tag{63}$$

$$C_{rt} = -c_r \cdot p(t) - a_r \cdot c_r - b_r \cdot b_r \tag{64}$$

$$D_{rt} = b_r \cdot p(t) - a_r \cdot b_r \tag{65}$$

are scalar values.

Vieta's Substitution The analytical solution of a cubic function can be obtained using for example Vieta's substitution method. Three roots can be found, for which there is at least one real root. Based on the value of the determinant, not discussed here, one is able to determine whether the other roots are complex or real. The general solution requires the calculation of

$$\Delta_0 = B_{rt} - 3A_{rt}C_{rt} \tag{66}$$

$$\Delta_1 = 2B_{rt}^3 - 9A_{rt}B_{rt}C_{rt} + 27A_{rt}^2 D_{rt} \tag{67}$$

$$C_d = \sqrt[3]{\frac{\Delta_1 \pm \sqrt{\Delta_1^2 - 4\Delta_0^3}}{2}}, \tag{68}$$

the general solution is then

$$\Delta s_k = -\frac{1}{3A_{rt}}\Big(B_{rt} + \zeta^k C_{rt} + \frac{\Delta_0}{\zeta^k C_{rt}}\Big), \qquad k \in \{0 \quad 1 \quad 2\} \tag{69}$$

where $\zeta = -\frac{1}{2} + i\frac{1}{2}\sqrt{3}$. Then, to recover s_k the function $s_k(t) = s(t - t_s) + \Delta s_k$ can be used. Often the single real solution determines the $s_k(t)$ that indicates the shortest distance to the path. However, from experience by implementation in Matlab, numerical errors occur such that it is not possible to distinguish which of the $s_k(t)$ is exactly real. Therefore, for each $s_k(t)$ the euclidean distance of the vehicle to the path is evaluated

$$d_k = \sqrt{\Big(p(t) - \bar{r}(s_k(t))\Big)^2}, \qquad k \in \{0 \quad 1 \quad 2\} \tag{70}$$

then the selected solution, corresponding to the minimal distance, is

$$k = \arg \min_{k \in \{0,1,2\}} d_k \tag{71}$$

such that $s(t) = s_k(t)$.

Lines When the path is a straight line, the term c_r equals $c_r = [0, 0, 0]^T$ in the Taylor polynomial approximation in (56). In this case (73) becomes

$$0 = C_{rt}\Delta s + D_{rt} \tag{72}$$

with the solution

$$s(t) = \frac{D_{rt}}{C_{rt}} + s(t - t_s) \tag{73}$$

4.2.8 Heading

Besides position control, the heading ψ of the multi-rotor should also be controlled. In order to control the heading while the multi-rotor is following the path, a reference heading $\psi_r(s(t))$ as a function of $s(t)$ can be specified. The heading dynamics can be approximated by a double integrator system (119) as discussed in Section 4.4

$$\dot{\psi}(t) = \omega_\psi(t) \tag{74}$$

$$\dot{\omega}_\psi(t) = \nu_\psi(t) \tag{75}$$

where $\nu_\psi(t)$ is taken as the control input. The control objective is to minimize the errors

$$e_\psi(t) = \psi(t) - \psi_r(s(t)) \tag{76}$$

$$e_{\omega_\psi}(t) = \omega_\psi(t) - \dot{\psi}_r(s(t)) \tag{77}$$

The proposed control law is

$$\nu_\psi = -k_3\Big(\psi(t) - \psi_r(s(t)\Big) - k_4\Big(\omega_\psi(t) - \frac{\partial\psi_r(s(t))}{\partial s}\dot{s}(t)\Big) + \frac{\partial^2\psi_r(s(t))}{\partial s^2}\dot{s}^2(t) + \frac{\partial\psi_r(s(t))}{\partial s}\ddot{s}(t) \tag{78}$$

where k_3 and k_4 are positive control gains, $\ddot{s}(t)$ is obtained by taking the time derivative of (37).

Lyapunov Consider the Lyapunov function

$$V_\psi(e_\psi(t), e_{\omega_z}(t)) = \frac{1}{2}k_3 e_\psi^2(t) + \frac{1}{2}e_{\omega_z}^2(t) \tag{79}$$

$$= \frac{1}{2}k_3\Big(\psi(t) - \psi_r(s(t)\Big)^2 + \frac{1}{2}\Big(\omega_\psi(t) - \frac{\partial\psi_r(s(t))}{\partial s}\dot{s}\Big)^2 \tag{80}$$

taking the time derivative and substitution of the heading-dynamics, yields

$$\dot{V}_\psi(e_\psi(t), e_{\omega_\psi}(t)) = -k_4\Big(\omega_z(t) - \frac{\partial\psi_r(s(t))}{\partial s}\dot{s}(t)\Big)^2 \tag{81}$$

$$= -k_4 e_{\omega_\psi}^2(t) \tag{82}$$

which is only semi-negative definite.

La Salle It can be determined from the error dynamics

$$\dot{e}_{\omega_\psi}(t) = -k_3\Big(\psi(t) - \psi_r(s(t)\Big) - k_4\Big(\omega_\psi(t) - \frac{\partial\psi_r(s(t))}{\partial s}\dot{s}(t)\Big) \tag{83}$$

$$= -k_3 e_\psi(t) - k_4 e_{\omega_\psi}(t) \tag{84}$$

that $e_{\omega_\psi}(t)$ remains only zero when $e_\psi(t)$ is zero. Therefore the controlled system is global asymptotically stable.

4.2.9 Simulation

Consider the path in the form of a Lissajous curve

$$r(\xi) = \begin{bmatrix} a\cos(\omega_a\xi) \\ b\sin(\omega_b\xi) \\ c\sin(\omega_c\xi) \end{bmatrix} \tag{85}$$

where $a = 10$, $a = 8$, $a = 4$, $\omega_a = 1$, $\omega_b = 2$, $\omega_c = 3$. This is a three-dimensional curve. The specified velocity along the path is $v_r(s) = 5m/s$. By applying the above procedure, the numerical results for the second order system as shown in Figure 45 are obtained. In black the Lissajous curve is shown and on the curve the Frenet-Serret frame is indicated. In green the second-order Taylor polynomial approximation of the Lissajous curve around $s(t)$ is shown. In red the traveled path of the multi-rotor is shown. It can be observed that while the vehicle's initial position is located relatively far away from the start of the path, the vehicle starts tracking the path close to its starting point $s(t_0)$, indicated by the green marker. In Figure 46, the tracking errors, and the value of the Lyapunov function (47) are shown.

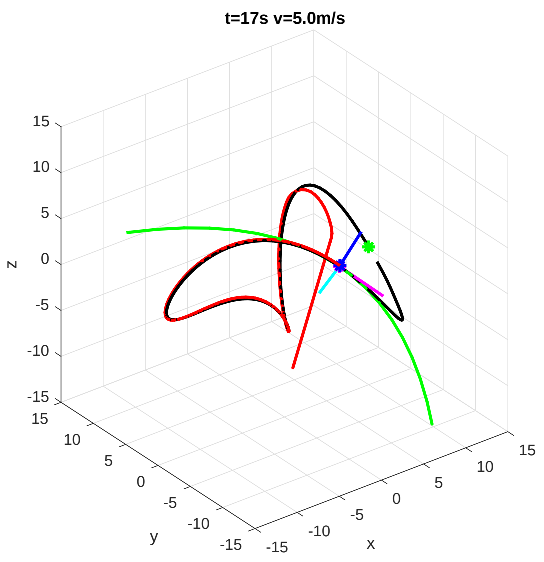


Figure 45: Numerical results of a simulation where the vehicle, indicated as a blue dot, is following a path. The path is indicated in black. At the closest distance to the vehicle on the path the Frenet-Serret frame is indicated. In green the second order Taylor polinominal approximation of the path is shown. In red is shown the traveled trajectory, and the start of the path is indicated with a green dot.

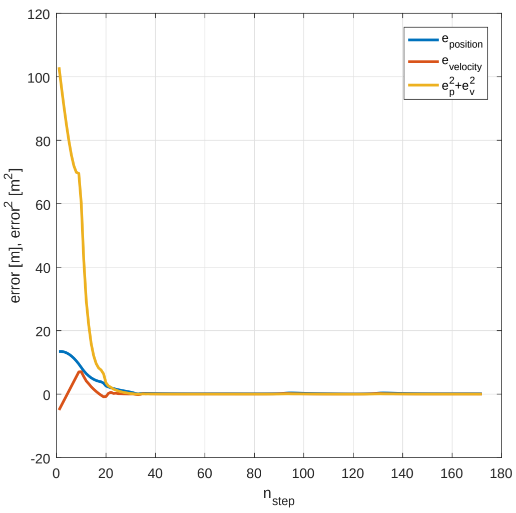


Figure 46: Control errors and the value of the Lyapunov function.

4.3 Path combinations

4.3.1 Path combinations and smoothing

Often a complete path is built of several smaller segments. These segments are for example, lines, circles, or other types of curves. In many cases the end of one segment coincides with the start of the next segment

$$r_{k-1}(\xi_k) = r_k(\xi_k) \qquad k \in \mathbb{N} \tag{86}$$

where k is the segment number and ξ_k the parameter of the connecting point between segment k and $k-1$. In addition, when the transitions are smooth, we have

$$r_{k-1}^{(n)}(\xi_k) = r_k^{(n)}(\xi_k) \qquad k \in \mathbb{N}, n \in \mathbb{N} \tag{87}$$

where (n) represents the n-th derivative with respect to ξ. On the other hand, in many cases, the end of one path segment does not coincide with the start of the next segment. In order to solve this, for example, an additional segment can be placed between the two non-coinciding ends. Instead of defining curve segment, it is also possible to specify a path in the form of a sequence of way-points.

$$q_k = \begin{bmatrix} q_{kx} & q_{ky} & q_{kz} \end{bmatrix}^T \qquad k \in \mathbb{N} \tag{88}$$

In this case, the way-points can be connected with curve segments. Estimation of the centrifugal forces requires a smooth curved path. Therefore, in this approach a cubic spline is used to describe a second order smooth path through the way-points. The segments of a cubic spline are cubic functions

$$r_k(\xi) = a_k + b_k\xi + c_k\xi^2 + d_k\xi^3 \qquad \xi_k \le \xi < \xi_{k+1} \tag{89}$$

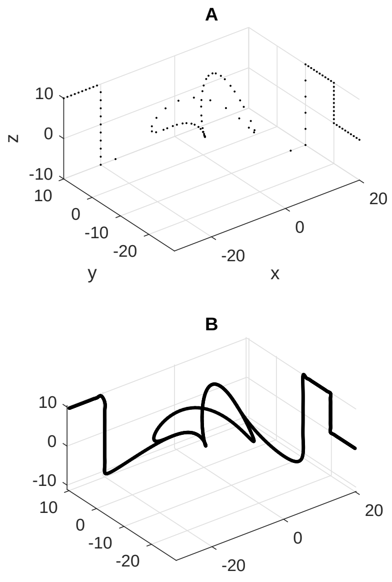


Figure 47: In A are shown the way-points of a combined path. In B the way-points are connected with seperate cubic functions, forming together a cubic spline.

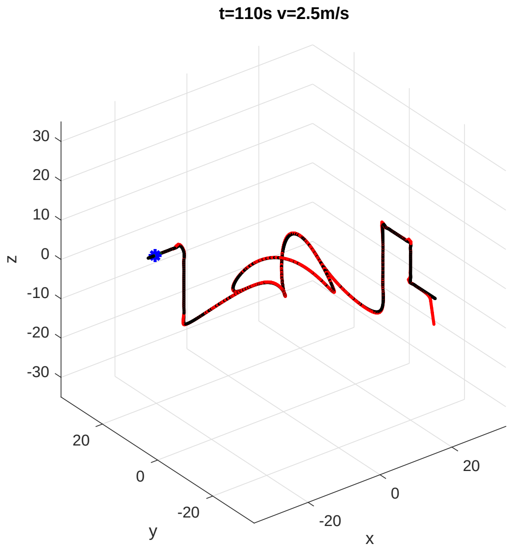


Figure 48: Shown are the numerical results of a simulation where the vehicle is following a path. In black is indicated the path, in red the traveled path is indicated, and the blue dot represents the vehicle.

the constrains for a smooth transition in the way-points q_k are

$$r^{(2)}_{k-1}(\xi_k) - r^{(2)}_k(\xi_k) = 0 \tag{90}$$

$$r^{(1)}_{k-1}(\xi_k) - r^{(1)}_k(\xi_k) = 0 \tag{91}$$

$$r_k(\xi_k) = q_k(\xi_k) \tag{92}$$

$$r_k(\xi_k + 1) = q_{k+1}(\xi_k + 1) \tag{93}$$

$$r^{(2)}_k(\xi_{k+1}) - r^{(2)}_{k+1}(\xi_{k+1}) = 0 \tag{94}$$

$$r^{(1)}_k(\xi_{k+1}) - r^{(1)}_{k+1}(\xi_{k+1}) = 0 \tag{95}$$

for $k_{min} < k < k_{max}$, for the begin and the end of the spline holds that the derivatives are $r^{(2)} = 0$ and $r^{(1)} = 0$. All the constraints can be put in matrix form and solved for a_k, b_k, c_k, and d_k. Further, one can keep track of the relation that indicates on which value of ξ curve k is active.

4.3.2 Simulation Combined Path

In Figure 47.A a set of way-points is shown. These way-points are obtained by sampling points on straight lines and a Lissajous curve. it can be observed that the Lissajous curve is traveled one and a half oscillations, resulting in an partly overlapping path. By fitting a cubic spline through the way-points, the path as shown in Figure 47.B is obtained. The path-following approach as described in Section 4.2-4.2.7 is applied. The Taylor polynomial method as described in Section 4.2.7 can be used to determine the nearest point on the path. Hopping back and forth the separate spline segments is not a problem since the relation $\xi \implies k$ is known. The numerical results of following the path are shown in Figure 48.

4.4 Stabilization of the inner-loop dynamics

The main goal of this section is to provide a stabilizing fast inner-loop controller in order to justify the choice of focusing on the control of the outer-loop in this thesis which amounted to using a double integrator system approximation for path following. Another justification is that in practice the default Pixhawk flight-controller is used in order to stabilize the attitude dynamics. The control objective is to ensure the inner-loop provides a force F_A to the outer-loop which is close to the reference force F_{ref} prescribed by the path following controller. In other words the goal is to minimize the tracking error

$$E_F = F_A - F_{ref} \tag{96}$$

where $F_A = [F_{x\,ref}, F_{y\,ref}, F_{z\,ref}]^T$ according to the dynamical system (13), (16), and (17) which are repeated below

$$F_A = -R(\lambda)Te_3^B + mge_3^A \tag{97}$$
$$\dot{\lambda} = Q(\lambda)\,\omega \tag{98}$$
$$\dot{\omega} = J^{-1}\bar{\tau} \tag{99}$$

The attitude dynamics are non-linear. Control theory provides multiple methods for stabilizing non-linear systems. A popular method is to linearize the system around a certain state and propose a linear control law that locally stabilizes the system, which is for example used in [53]. Another method is back-stepping in which recursively a special class of non-linear systems can be stabilized, this method is for example applied in [26]. In the remaining of this section we rely on feedback linearization in which a state transformation is applied, and a linearizing feedback term, such that the input-output behavior of a new system appears linear. The method as is described in [52] will be followed, in the following a concise description of the method is given, for a more compressive analyses is referred to the literature.

4.4.1 Feedback linearization

In addition to the non-linear equation (13), the thrust T of the multi-rotor is constrained to be positive $T \geq 0$. Therefore we propose to use $T = \hat{T}^2$ where $\hat{T}$ is a new control input. In order to allow a feasible state transform, as will be discussed later, we propose to use a dynamical extension of the input thrust in the form

$$\dot{\hat{T}} = \hat{T}_d \tag{100}$$
$$\dot{\hat{T}}_d = \bar{T} \tag{101}$$

where $\bar{T}$ is a new system input. First the system is brought into standard form according to [52]

$$\dot{x} = f(x) + g(x)u \tag{102}$$
$$y = h(x) \tag{103}$$

where

$$x = \begin{bmatrix} \phi \\ \theta \\ \psi \\ \omega_x \\ \omega_y \\ \omega_z \\ \hat{T} \\ \hat{T}_d \end{bmatrix} \quad f(x) = \begin{bmatrix} \omega_x + \omega_z \sin(\phi) tan(\theta) + \omega_z \cos(\phi)\tan(\theta) \\ \omega_y \cos(\phi) - \omega_z \sin(\phi) \\ \omega_y \sin(\phi)\sec(\theta) + \omega_z \cos(\phi)\sec(\theta) \\ 0 \\ 0 \\ 0 \\ \hat{T}_d \\ 0 \end{bmatrix} \quad g(x) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{104}$$

$$u = \begin{bmatrix} \bar{T} \\ \bar{\tau}_x \\ \bar{\tau}_y \\ \bar{\tau}_z \end{bmatrix} \quad h(x) = \begin{bmatrix} -\left(-\sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta\right)\hat{T}^2 \\ -\left(cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi\right)\hat{T}^2 \\ -\left(cos\theta\cos\phi\right)\hat{T}^2 + mg \\ \psi \end{bmatrix} \quad y = \begin{bmatrix} F_x \\ F_y \\ F_z \\ \psi \end{bmatrix} \tag{105}$$

where besides the force F_A also the heading ψ dynamics are incorporated. The goal of feedback linearization is to find an input

$$u = \alpha(x) + \beta(x)\nu \tag{106}$$

that renders the system behavior between y and ν linear by a change of variables

$$z = T(x) \tag{107}$$

which is invertible ($\partial T/\partial x$ is non-singular for every x). In the following approach the time derivative of the output y is taken in order to get insight in how the system inputs impact the system's output. After a double differentiation of the state with respect to time, for our system, all the inputs appear for the first time in the equations and therefore the relative degree of each output is $r_i = 2$ where $i \in \{1, 2, 3, 4\}$. This second derivative can be written as

$$\ddot{y} = N(x) + A(x)u \tag{108}$$

where, in Lie derivatives notation,

$$A(x) = \begin{bmatrix} L_{g1}L_f^{r_1-1}h_1(x) & L_{g2}L_f^{r_1-1}h_1(x) & L_{g3}L_f^{r_1-1}h_1(x) & L_{g4}L_f^{r_1-1}h_1(x) \\ L_{g1}L_f^{r_2-1}h_2(x) & L_{g2}L_f^{r_2-1}h_2(x) & L_{g3}L_f^{r_2-1}h_2(x) & L_{g4}L_f^{r_2-1}h_2(x) \\ L_{g1}L_f^{r_3-1}h_3(x) & L_{g2}L_f^{r_3-1}h_3(x) & L_{g3}L_f^{r_3-1}h_3(x) & L_{g4}L_f^{r_3-1}h_3(x) \\ L_{g1}L_f^{r_4-1}h_4(x) & L_{g2}L_f^{r_4-1}h_4(x) & L_{g3}L_f^{r_4-1}h_4(x) & L_{g4}L_f^{r_4-1}h_4(x) \end{bmatrix} \quad N(x) = \begin{bmatrix} L_f^2 h_1(x) \\ L_f^2 h_2(x) \\ L_f^2 h_3(x) \\ L_f^2 h_4(x) \end{bmatrix} \tag{109}$$

where $A(x)$ is known as the decoupling matrix. The expression for $A(x)$ is given in Appendix E.3. The state transformation is chosen as

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} h(x) \\ L_f h(x) \end{bmatrix}. \tag{110}$$

The proposed feedback linearization law is

$$u = A(x)^{-1}(-N(x) + \nu). \tag{111}$$

In order to apply this rule $A(x)$ must be non-singular which is by verification the case when starting and remaining in

$$\hat{T} > 0 \tag{112}$$

$$-\frac{1}{2}\pi < \phi < \frac{1}{2}\pi \tag{113}$$

$$-\frac{1}{2}\pi < \theta < \frac{1}{2}\pi \tag{114}$$

loosely speaking a consequence is that the produced force is restricted to $F_z < mg$. Because the rotation matrix is parameterized by Euler angles gimbal locks occur outside this range. An alternative would be to use another parameterization, for example quaternions, which can be considered in future work. By substitution of u in system (102)-(103) and applying state transformation (110) the linear decoupled system can be written as

$$y = z_1 \tag{115}$$

$$\dot{z_1} = z_2 \tag{116}$$

$$\dot{z_2} = \nu \tag{117}$$

expressed in the new states is obtained under the constraints (112)-(114). This decoupled system can be rewritten as

$$\begin{bmatrix} F_A \\ \psi \end{bmatrix} = \begin{bmatrix} I_{4\times4} & 0_{4\times4} \end{bmatrix} z \tag{118}$$

$$\dot{z} = \begin{bmatrix} 0_{4\times4} & I_{4\times4} \\ 0_{4\times4} & 0_{4\times4} \end{bmatrix} z + \begin{bmatrix} 0_{4\times4} \\ I_{4\times4} \end{bmatrix} \begin{bmatrix} \nu_{F_A} \\ \nu_\psi \end{bmatrix} \tag{119}$$

where ν_{F_A}, and ν_ψ are new system inputs that influence independently the force F_A and the heading ψ of the system respectively. The system reveals a double integrator chain. For the heading dynamics, expressed in the z states, a path depended stabilizing control law was proposed in Section 4.2.8. The force dynamics are stabilized in the sequel.

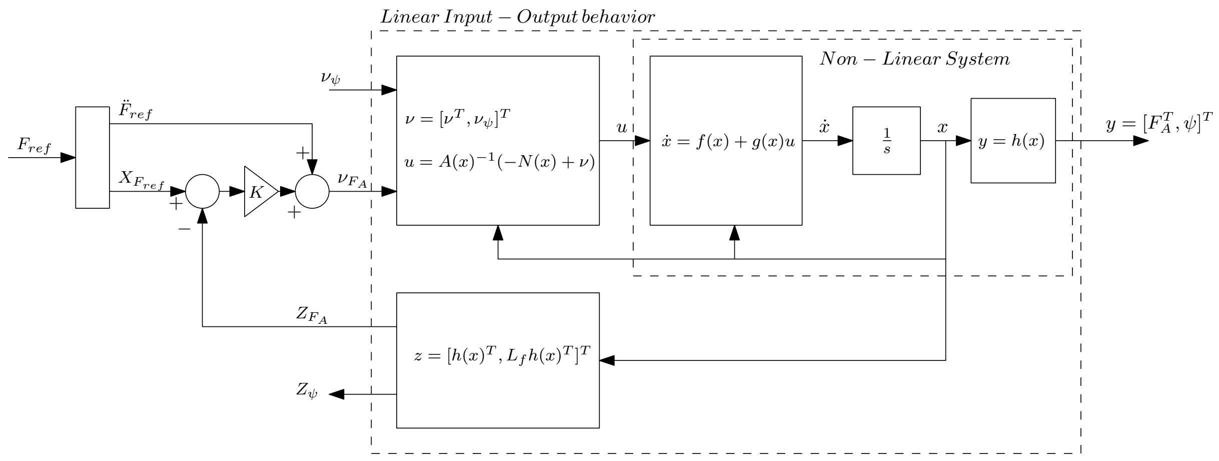


Figure 49: Schematic representation of the control scheme where feedback linearization is used to obtain a linear system input-output behavior and a tracking controller is used in order to stabilize the system.

4.4.2 Stabilizing the inner-loop dynamics

Corresponding to the control objective (96) the following tracking error was defined

$$E_F = F_A - F_{ref} \tag{120}$$

with the double integrator attitude dynamics

$$F_A = C_z Z \tag{121}$$

$$\dot{Z} = A_z Z + B_z \nu_{F_A} \tag{122}$$

where $Z = \left[(z_1)_{1-3}^T, (z_2)_{1-3}^T\right]^T$ with $()_{i-j}$ indicating the vector elements i up to j, and C_z, A_z, B_z can be derived from (118)-(119). In this form a regular linear tracking problem is obtained. The following control law is proposed

$$\nu_{F_A} = -K(Z - X_{F_{ref}}) + \ddot{F}_{ref} \tag{123}$$

where state $\left[F_{ref}^T, \dot{F}_{ref}^T\right]^T$ and $\ddot{F}_{ref}$ are assumed to be known or estimated by an appropriate state observer, K is a matrix with control gains which can be obtained by following the LQR method, as is elaborated in Appendix E.6 for the linearized system dynamics, or following the pole-placement method [54] yielding the system

$$E_F = C_z X_e \tag{124}$$

$$\dot{X}_e = (A_z - B_z K)X_e \tag{125}$$

The stability of the system can be checked by calculating the eigenvalues of $A_z - B_z K$ which should have negative real part. Limited by the constraints (112)-(114), and the saturation of the 'real' non-linear input (12), the control gains can be selected relative high yielding a fast converge of the error dynamics (125).

Simulation The approach of feedback linearization and the stabilization of the remaining linear system behavior is verified by simulation. In Figure 49 schematically the control scheme is presented. Realistic system parameters are used obtained form the Parrot AR.Drone 2.0 in [53]. In Figure 50, the numerical results of the force F_A following the reference force F_{ref} are shown. It can be observed that the reference is tracked with zero error. In Figure 51 the nonlinear system states x and the real system inputs in terms of body torque $\bar{\tau}$ and thrust T are given. A remark is that this control law only holds under the constraints (112)-(114) which loosely means that the multi-rotor is not flying upside-down, by simulation was verified that indeed these constraints must mold.

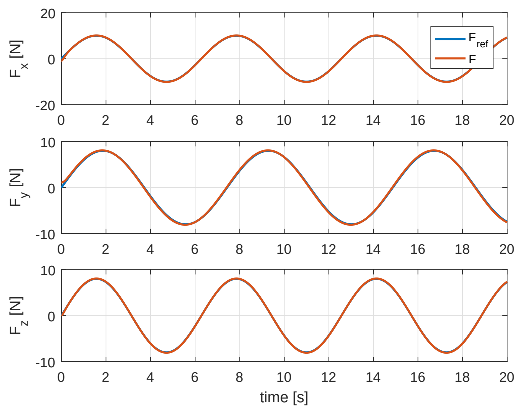


Figure 50: Numerical results showing that the reference force is tracked with zero error. The error converges relative fast to zero using feedback linearization and state feedback.

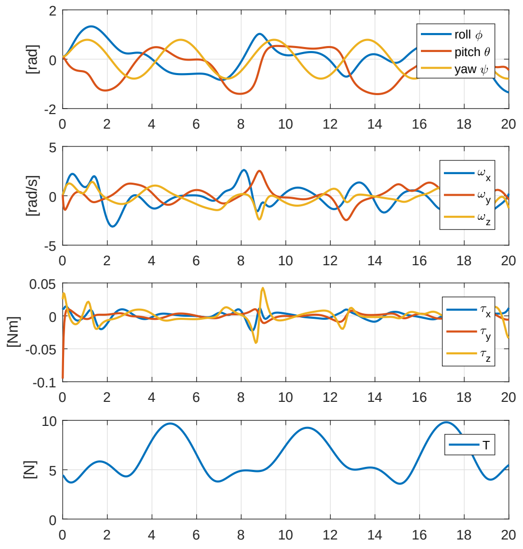


Figure 51: Numerical results representing the original system states and inputs corresponding to the tracking results as is shown in Figure 50.

4.5 Model reduction

4.5.1 Singular perturbation theory

In this section based on singular perturbation theory [52] is verified that the outer-loop dynamics can be approximated by a double integrator system for the position dynamics under the assumption that the attitude dynamics are much faster than the position dynamics. First the system is written in the standard form according to [52]. The position and velocity errors (40)-(41) defined in the the path following section, are here defined by e_{PF}. Furthermore there is the force tracking error E_F (96). The system dynamics can be described in the form

$$\dot{e}_{PF} = g(e_{PF}, F_A(X_e)) \tag{126}$$

$$\varepsilon \dot{X}_e = f(X_e, F_{ref}(e_{PF})) \tag{127}$$

where ε is assumed to be a small value indicating that the force tracking dynamics by the inner-loop are much faster than the slow outer-loop position and velocity dynamics. By relying on real-data obtained in [53] where a Parrot AR.Drone 2.0 is stabilized by a linear controller in y direction with a gain $K_\phi = 2Nm/rad$ and with a gain $K_y = 1N/m$ the following dimensionless ratio is obtained

$$\varepsilon = \frac{I_x K_x}{m K_\phi} \tag{128}$$

where $I_x = 0.0033kgm^2$ is the moment of inertia over the x-axis involved in the attitude dynamics, and $m = 0456kg$ the mass of the vehicle involved in the position dynamics. Evaluation of this equation yields a value in the order of $3.6 \cdot 10^{-3}$ which can be considered as small. Assumed is that for this multi-rotor a value of the same order can be expected and therefore we assume that the inner-loop is much faster than the outer-loop. According to singular perturbation theory [52], by letting ε approach zero the following root can be found

$$0 = f(X_e, F_{ref}(e_{PF})) \tag{129}$$

$$0 = (A_z - B_z K)(Z - X_{ref}) \tag{130}$$

$$Z = X_{ref} \tag{131}$$

since $F_A = C_z Z = F_{ref}$ the slow position and velocity dynamics can be approximated by

$$\dot{e}_{PF} = g(e_{PF}, F_{ref}) \tag{132}$$

for which in Section 4.2 an exponentially stable path following controller was developed. In Section 4.4.2 a controller that exponentially stabilizes the force tracking error (127) was developed. Then according to [55, Theorem 5] there exist a $\varepsilon^* > 0$ such that for all $0 < \varepsilon < \varepsilon^*$, the origin of the dynamical feedback system (126)-(127) is exponentially stable. Loosely speaking a controller that exponentially stabilizes the approximated system (132) will also stabilize the complete system (126)-(127) around its origin, when ε is sufficiently small and (127) is exponentially stable. Another conclusion is that the position dynamics

$$\dot{p} = v \tag{133}$$

$$\dot{v} = \frac{1}{m}F_A = \frac{1}{m}C_z Z \tag{134}$$

are then approximated by

$$\dot{p} = v \tag{135}$$

$$\dot{v} = \frac{1}{m}F_{ref} \tag{136}$$

which is a double integrator system as was used in Section 4.2.

4.6 Summary and remarks

In this section a path-following controller for a multi-rotor was discussed. In a first step the nonlinear equations of a multi-rotor were separated yielding a cascades system with a linear model for the position dynamics and a nonlinear model for the attitude dynamics. In general the position dynamics of a multi-rotor are considered much slower than the attitude dynamics. The fast attitude dynamics are controlled in an inner-loop, tracking a reference defined by the outer-loop. For the inner-loop a feedback linearization control law is considered, and evaluated in simulation. Based on singular perturbation theory the dynamical model for the outer-loop, including the inner-loop, can be approximated by a double integrator system. Path-following is considered for the outer-loop which is concerned with the relative slow position and velocity dynamics. For path-following, a control law is developed which is expressed directly in the inertial frame. The stability of the system is checked using a convergence proof, based on Lyapunov's and La Salle's theories. In addition to position and velocity control, a control law for guiding the heading of the multi-rotor along the path is proposed. In order to calculate the reference-point on the path, a computationally efficient method based on a local second-order Taylor polynomial approximation is proposed, for which an analytically solution exists. The proposed method is evaluated in simulation on several complex paths, the numerical results are graphically presented. Furthermore, a cubic-spline through a set of way-points is used in order to obtain a smooth path. Noteworthy is that a similar method, based on path-following, is recently successfully tested in practice in the honors project at TU/e [47].

5 Drone Referee System

This chapter describes the design of a conceptual on-line motion planner for a drone referee. Guidelines formulated by FIFA and an experienced referee are used to provide a computer algorithm determining the drone's movements. This algorithm also relies on scale-space theory which is used to obtain a point-of-interest during the game. The algorithm is evaluated in a specially designed 3D simulation environment for drone refereeing based on real-data from a robot-soccer game.

The remainder of the chapter is organized as follows. Section 5.1 provides some motivation and briefly describes previous work. The problem formulation is given in Section 5.2 and an online motion planner for drone refereeing is proposed in Section 5.3. Section 5.4 summarizes the development of a simulation environment with real-data and Section 5.5 provides simulation results. Conclusion and remarks are given in Section 5.6

5.1 Motivation and previous work

In many sports, especially those with fast dynamics, refereeing and commenting a game is difficult. In both cases, the decisions of the referee and the statements of the commentator influence the game and the public-atmosphere. A prominent example is soccer, which is a sport with fast dynamics. Due to this fast dynamical behavior, a significant effort from the referee is required to guarantee fair-play. In order to assist the referee, a video referee, and goal-line technology has been recently introduced in soccer. Recently, former professional footballer and manager Franz Beckenbauer stated that in the future assistance of an automatic-referee system will be carried out using drones [56]. Already, in the field of robot soccer, the idea and research on an automatic referee and commentating system is adopted for more than one and a half decade. In [57] an autonomous commenting system is proposed using the information from both a top camera and live observing the game and public with a humanoid robot. In the Robocup [27] middle-size league (MSL) an automatic referee system is proposed in [58] [59], based on information from a top camera. Also for the small-size league (SSL) an automatic referee system is developed in [60]. Again this system is checking the rules of the game using information from a top-camera. In order to place the ball at the correct place for a free-kick, without intervention of a human, a referee-robot is proposed [60]. In [13] an autonomous referee is proposed in the form of one or multiple cooperating drones. In [14] a more generic system-architecture is proposed. This system considers, besides drones, also other agents such as ground-robots, and fixed camera setups. The proof-of-concept in [14] relies on the direct implementation of the architecture on the hardware in the environment of a MSL soccer-field. Moreover, one of the agents in this referee-system is a multi-rotor, an important finding in this work was that the use of a down-facing camera on a drone gives a too narrow field of view. Another reported problem of a multi-rotor hovering above a soccer ball is that the ball is blown away.

In this work we try to reduce the design time by providing a simulation environment for drone refereeing based on real-data. In addition, we make a preliminary proposal for an on-line motion planner for drone refereeing. Herein, the insights of an experienced human referee are translated to a computer algorithm.

5.2 Problem Setting

5.2.1 Assumptions

In this work a RoboCup MSL soccer game is considered. However, the drone referee motion planner is inspired by soccer games with humans due to the strong similarities between robot and human games (e.g. indoor five against five human games). A soccer game with real humans is currently considered as being too complex. Furthermore, we assume that the current state of the game is known at each time-sample, i.e., the position and velocity of the players and the ball is known. In our case, only one multi-rotor equipped with a camera is considered. In addition, we assume a non-linear multi-rotor model (5)-(8), and an unconstrained flight area.

5.2.2 Guidelines

According to FIFA, *'The best position is one from which the referee can make the right decision.'* [61]. In practice the best position can only be verified afterwards. However, a recommended position can be based on experience and knowledge about the players. Often the diagonal system of control is advised. The referee follows then, approximately, a diagonal path across the field such that the assistant referee

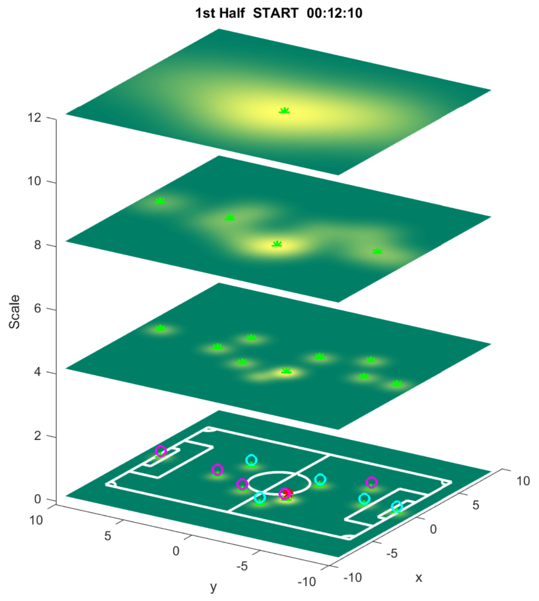


Figure 52: Scale-space representation of the players and ball on the field. The players are indicated by the small circles, the ball by a red star-point, and the points of interest by the green star-points which occur in the maxima at each scale.

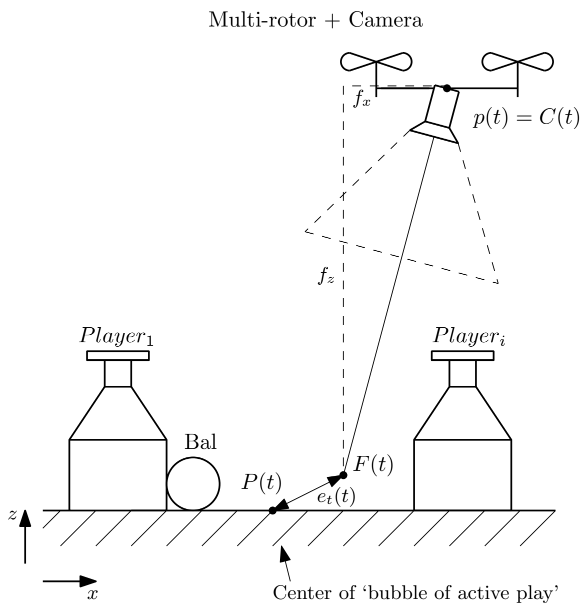


Figure 53: Schematically representation of the multi-rotor following the center of the 'bubble of active play' $P(t)$, in order to keep the 'bubble of active play' in the field of view of the camera.

is visible, and the overview is shared between the center- and the assistant-referee. Since we assume here only one observing multi-rotor the diagonal rule is neglected. In order to not influence, the game while monitoring the game, a distance is kept to the *'bubble of active play'* [62]. The ball by itself cannot commit a foul, therefore the referees attention must be divided over the ball as well as over the players.

In this work we try to translate the guidelines given by an experienced human center referee [62] to a computer algorithm. Conceptually, this algorithm can be used for drone refereeing or commentating. The following guidelines will be mimicked by the algorithm

- The game may not be influenced by the position of the referee.
- It is advised that the referee keeps at a distance from the 'bubble of active play'.
- The referee should keep track of the 'bubble of active play'.

A first clear advantage of a drone referee over a human referee is that while enforcing the first statement above might be difficult for a human referee it is trivial for a drone referee. In fact, it is often the case that the referee involuntarily intervenes in the system by colliding with other players or by blocking the ball. In turn as long as the drone referee flies sufficiently high such that the noise and air flow are not noticeable by the players, it will hardly have influence in the game (except in cases where a player shoots a ball in the direction of the drone).

We will focus on enforcing the two other statements above. As can be derived from the second statement above the referee should remain at a distance from the 'bubble of active play'. The main question is here: what is the bubble of active play? In general this is a relative simple question to answer for humans, however for machines this can be difficult. Subquestions are for example, what is active, and what is considered as the bubble, which will be addresses in the sequel.

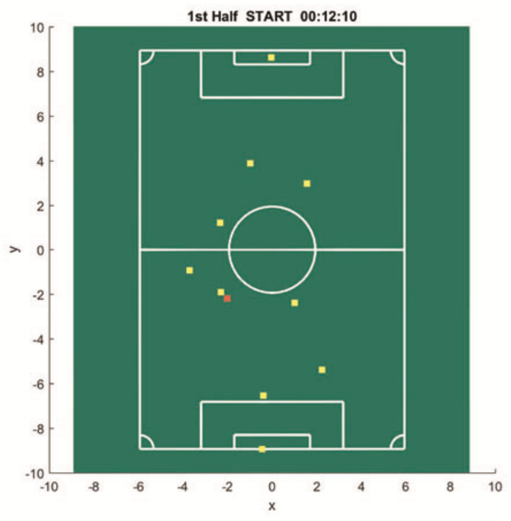


Figure 54: Projection of field-lines, players and ball on the image representing the game at a original scale ($\sigma = 0$). The players are indicated with a pixel having the weight value w_i (yellow). The ball is indicated by a pixel with a higher weight value w_b (Orange).

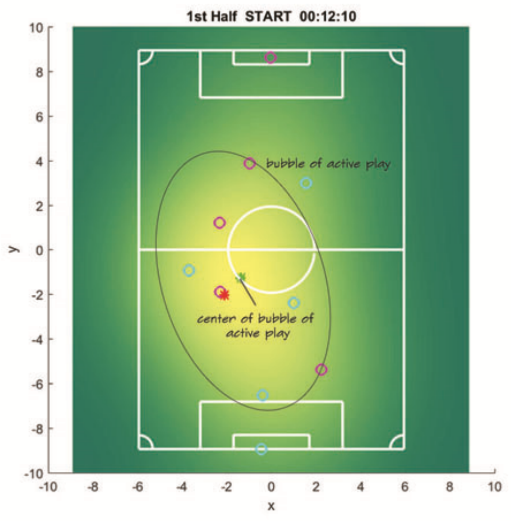


Figure 55: The same state of the game as in Figure 54. However the image in Figure 54 is now observed with an larger scale ($\sigma > 0$). The separate pixels in Figure 54 are merged together such that the pixel with highest value (green star-point) represents the center of the 'bubble of active play'.

5.3 Online Motion Planner for Drone Refereeing

5.3.1 Algorithm

In order to answer the questions raised at the end of Section 5.2, our solution is inspired by the biological model of vision which is present in the human-visual-system. The human-visual-system perceives the world at different scales simultaneously [63]. In this section we try to simulate how the game is perceived at a larger scale. In order to start simple, a state of the game at a single time step is considered, i.e., the dynamical behavior of the players and of the ball is initially neglected. In the sequel the focus is on defining what can be considered as the 'bubble of active play', by perceiving the game from a larger scale. The concept of scale is described by scale-space theory.

Scale space theory Scale-space theory is a special case of biologically motivated computer vision [63]. Often the zero scale in a digital image is at pixel level. At higher scales the interconnections with the surrounding pixels are going to play a role. In this work we considered the 2D linear scale-space. In this case higher scales are obtained by the convolution of an image with a Gaussian kernel

$$G_{2D}(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{137}$$

where x and y represent the 2D image directions, and σ is the scale. The obtained image at a higher scale ($\sigma > 0$) is then obtained from the convolution between the kernel $G_{2D}(x, y, \sigma)$ and the original image $L(x, y, 0)$

$$L(x, y, \sigma) = G_{2D}(x, y, \sigma) * L(x, y, 0) \tag{138}$$

where $*$ is the convolution operator. Note that when the scale parameter is zero $\sigma = 0$, the kernel boils down to the Dirac delta function, and convolution does not affect the image. Another observation is that, when the scale parameter is replaced by time, the Gaussian function equals the analytical solution of an impulse-response for the (thermal) diffusion equation. When the scale parameter approaches infinity a constant value is obtained for all the pixels.

Solution In our proposed solution the soccer-field is first discretized in order to obtain a high-resolution image consisting of small pixels. All the pixels have the same default value which is zero. Each player is modeled in the image as a single pixel having a value larger than zero representing an importance weight. The ball is also represented by a pixel where the pixel-value represents a specific weight. Then we define

$$L(x, y, 0) = b(x, y) + \sum_{i=1}^{n} p_i(x, y) \tag{139}$$

where $p_i(x, y)$ represents the weight function of each player i at each pixel position, and $b(x, y)$ represents the weight function of the ball at each pixel position. More precisely, if (x_i, y_i) is the center-pixel of player i in the image then

$$p_i(x, y) = \begin{cases} w_i & \text{if } (x, y) = (x_i, y_i) \\ 0 & \text{otherwise} \end{cases} \tag{140}$$

and if (x_b, y_b) is the center-pixel of the ball in the image then

$$b(x, y) = \begin{cases} w_b & \text{if } (x, y) = (x_b, y_b) \\ 0 & \text{otherwise} \end{cases} \tag{141}$$

where w_i and w_b are constant weights. In Figure 54, an example of the image $L(x, y, 0)$ at its original scale is given. However, from this image, representing the players and ball positions and weights, it is still not possible to determine a meaning in terms of point-of-attention or 'bubble of active play'. In order to see the connections of the players, we observe the image with another scale using (138). At different scales different local maxima appear, as it is shown in Figure 52. We define these points as the points of interest. In our case we select a scale σ_p such that only one global maxima is left, at the pixel location $(x_p(t), y_p(t))$, which we define as the *center of the 'bubble of active play'*, which is

$$(x_p(t), y_p(t)) = \arg \max_{x \in X, y \in Y} L(x, y, \sigma_p)(t) \tag{142}$$

An example of the 'bubble of active play' and center of the 'bubble of active play' is indicated in Figure 55. In vector form the center of the 'bubble of active play' is given by $P(t) = [x_p(t), y_p(t), 0]$. Since we have, currently, no (natural) rule that defines what exactly can be considered as inside and what as outside the 'bubble of active play', we take the center of the 'bubble of active play' as reference.

Tracking the 'bubble of active play' with a drone. Assume the camera is located at $C(t) = [x_c(t), y_c(t), z_c(t)]$ and the camera is observing an area around point $F(t) = C(t) - [f_x, f_y, f_z]$ representing the point-of-focus, where f_x,f_y, and f_z are offset constants with respect to the camera, as is schematically sketched in Figure 53. The control objective is to minimize the tracking error

$$e_t(t) = \hat{P}(t) - F(t) \tag{143}$$

where $\hat{P}(t)$ is the low-pass filtered version of $P(t)$ in order to guarantee a smooth tracking target, a first order low-pass filter is used with time constant τ_P, and $F(t) = [x_f(t), y_f(t), z_f(t)]$ is the point-of-focus of the camera mounted on the multi-rotor. The position of the multi-rotor is denoted by $p(t)$, which is equal to the camera position $p(t) = C(t)$. In this work the heading of the multi-rotor is kept constant. When assuming the double integrator dynamical system (14)(15) a regular PD feedback control-law will stabilize the system, as is for example described in [54].

Note that this tracking law enforces that the two desired conditions based on the insights of an experienced referee mentioned above are met. In fact the drone will try to track the 'bubble of active play' while keeping a certain distance, determined by f_x, f_y, f_z, from this bubble.

5.3.2 Practical Implementation

In general, convolution of an image with a large 2D kernel is an computational intensive task. However in our case we can apply some properties which simplify the calculations significantly. Since the players and the ball are modeled as known single points (impulse functions), the separate impulse responses can be summed. The convolution of an impulse with another function/kernel is equal to this function/kernel,

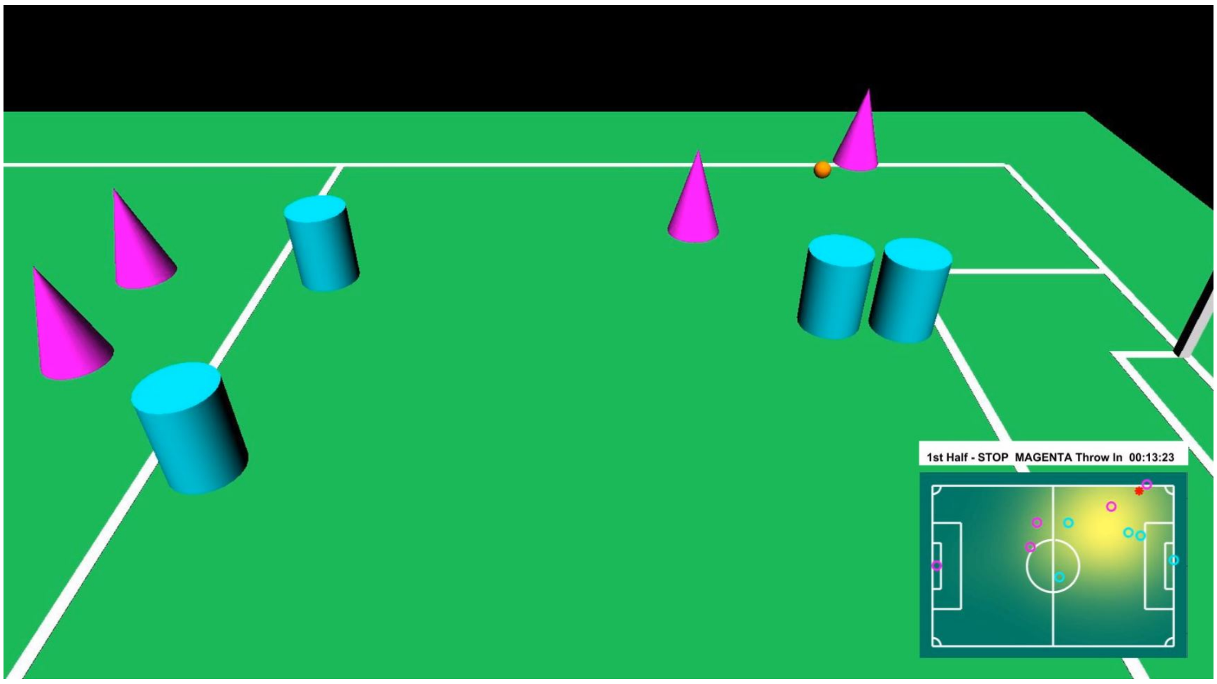


Figure 56: Example image showing the drone perspective in the 3D Simulink simulation enviroment. The simulation is based on real-data from a RoboCup MSL game.

which in our case is a Gaussian function. In addition, the separability property of the Gaussian function can be used such that a 2D Gaussian function can be constructed from two 1D Gaussian function.

$$e^{-\frac{x^2+y^2}{2\sigma^2}} = e^{-\frac{x^2}{2\sigma^2}} e^{-\frac{y^2}{2\sigma^2}} \tag{144}$$

In this way, the game can be observed from a larger scale in a single step, without the need of applying convolution. Finally, the center of the 'bubble of active play' is obtained by scanning the image for the highest pixel-value.

5.4 Real-data Simulation Environment

In order to evaluate the algorithms for the autonomous-referee system, we propose the use of a simulation environment. This will omit the time required for implementing each proposed algorithm in hardware and therefore drastically reduce the development cost. Many RoboCup (MSL) games have been played in the last years, a considerable number of these games are recorded in the form of log-files. These log-files contain, among others, the player and ball position at each time instant. Besides the positions also events such as kickoff, scored goals, and freekicks are logged. These logs provide the experimental data in order to test new algorithms on.

An existing simulation environment is for example the simulator of the Tech United team [64]. More recently adopted, are the log-files from the MSL logging database [65]. In this work the latter files are used. The MSL logging database files have a (human readable) json file structure and are therefore relative simple to interpret and implement in an simulation environment. The only, currently, missing information in the log-files are the field-dimensions which are obtained from the Tech United data-base [66].

The simulation environment is build in Matlab Simulink, an example image is shown in Figure 56. After loading and transforming the log-files to a time sorted position sequence, the sequence is played in Simulink. Currently the players and ball are modeled as colored basic geometric elements. A non-linear controlled multi-rotor model, as described in Chapter 4, determines the position of the camera/viewport.

5.5 Simulation Results

The algorithm as described in Section 5.3 is implemented in the simulation environment as described in Section 5.4.

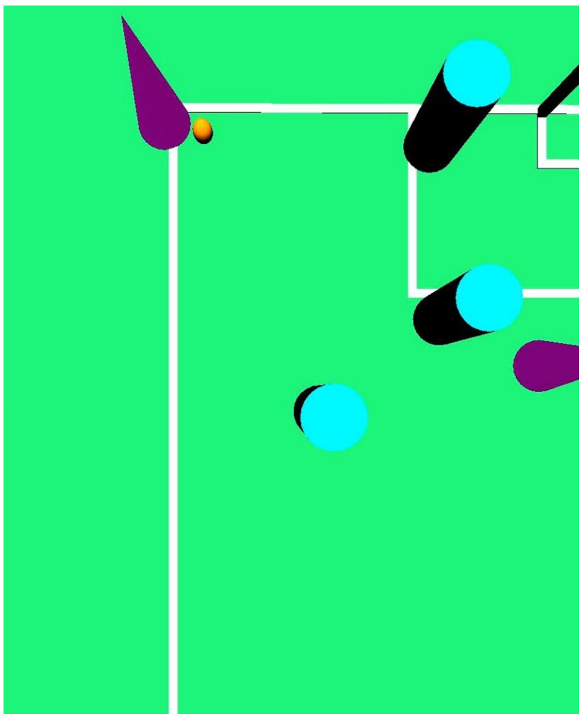


Figure 57: Wide angle view of a multi-rotor hovering above the 'bubble of active play' with down facing camera setup. Although the ball is here in the field of view, after kick-off the ball will leave the field of view rapidly.

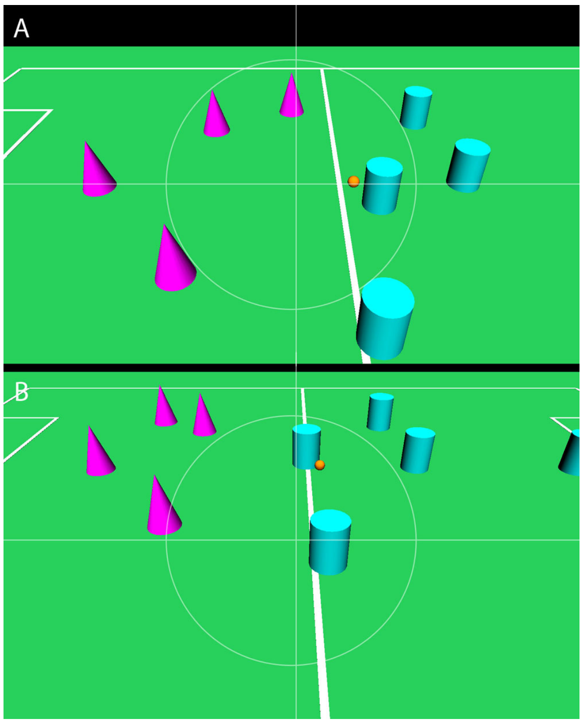


Figure 58: View of the multi-rotor following the 'bubble of active play' and an additional offset. In both images the ball is at the same position. Based on the position of the player with respect to the ball the camera position is adapted.

Bottom Facing Camera Position As a first approach, which will serve as a comparison to the proposed algorithm, the multi-rotor hovers above the point-of-interest, neglecting the guideline of keeping a distance from the 'bubble of active play' for a moment, which is $\sqrt{f_x^2 + f_y^2} = 0$ (which in turn imposes $f_x = F_y = 0$). A wide angle camera is, in simulation, mounted in bottom facing direction on the multi-rotor. In Figure 57, a captured image of the simulation is shown. Although the ball and players are here in the field of view, due to the high velocity, the ball after corner-kick will leave the field of view soon. It can be concluded that this camera configuration is not ideal.

Distance to the Ball In a second approach, the 'bubble of active play' is tracked by the camera while keeping the multi-rotor at a fixed distance from the center of the 'bubble of active play', which is $\sqrt{f_x^2 + f_y^2} > 0$. The weight w_b assigned to the ball is relative high compared to the weight w_i assigned to the players. Therefore the position of center of the 'bubble of active play' will be close to the position of the ball. The result is that the multi-rotor does not hover above the ball, and therefore not influences the game by blowing the ball away.

Although this method works in simulation very well, by visual inspection, the algorithm can be improved in order to obtain a more appealing perspective on the game. The following rules are added to the algorithm

- **Increase distance when the ball moves fast.** The distance to the center of the 'bubble of active play' $\|f\| = \left(\sqrt{f_x^2 + f_y^2 + f_z^2}\right)$ is increased proportionally with the velocity of the ball, such that
$$F(t) = C(t) - [f_x, f_y, f_z] \left(1 + c_v \|\dot{b}(t)\|\right)$$
where c_v is a constant, and $\dot{b}(t)$ is the velocity of the soccer ball. The result is that the field of view obtained by the camera on the multi-rotor is increased when the ball moves faster.
- **Take into account the potentially shooting direction.** The multi-rotor should move with the potential shooting direction of the ball. The potential shooting direction d_b is the pointing direction

of the ball with respect to the position of player k, denoted by p_k who is in possession of the ball

$$d_b(t) = \frac{b(t) - p_k(t)}{\|b(t) - p_k(t)\|}$$

where b is the position of the ball. The control objective changes then to

$$e_t(t) = \hat{P}(t) + \hat{D}_d(t) d_b(t) - F(t) \tag{145}$$

where $\hat{D}_d$ is a first order low-pass filtered version, with time constant τ_d of

$$D_d(t) = \begin{cases} c_d & \text{if player on the ball} \\ 0 & \text{otherwise} \end{cases} \tag{146}$$

in order to guarantee a smooth transition, where c_d is a constant offset. The effect of this approach is that the multi-rotor moves already in the direction of the ball's potentially shooting direction.

In Figure 58, the effect of the potential shooting direction offset on the position of the multi-rotor is visualized. In both, Figure 58.A and 58.B, the ball is placed at the same field location, and in both cases the center of the 'bubble of active play' is equal. The shooting direction is determined based on the position of the ball with respect to the position of the player on the ball. The dynamical behavior is best visualized by the following video: `https://youtu.be/3PHVnQjGKdM`. For the numerical simulations the following values were used , $\sigma = 5$, $w_i = 1$, $w_b = 4$, $\tau_P = 0.1$, $fx = -8$, $fy = 0$, $fz = 4$, $c_v = 0.5$, $c_v = 0.5$, $\tau_d = 0.5$, and $c_d = 2$.

5.6 Summary and remarks

We proposed an on-line motion planner for drone refereeing or commenting a soccer game. The solution is obtained by transforming the guidelines of a human referee into a computer algorithm. This algorithm incorporates the scale-space model which is inspired by the biological-visual-system. The algorithm is evaluated, by visual inspection, in a 3D simulation environment based on real-data from a RoboCup MSL soccer game. The results are shown in a video. The method must be considered as conceptual, and we want to stress that this control law is not a finished product. Many situations that occure during a soccer game are not covered by this algorithm, therefore future research is suggested before applying it in an autonomous referee system, where this algorithm might be of inspiration. We want to emphasize, corresponding to the statements in [61], that experience and insight can improve the 'best position' of the referee, i.e., a better estimation of the behavior of the players, physics, or environment may improve the performance of the referee drone.

6 Experimental Results

According to the system requirements, presented in Section 2.2, the multi-rotor should be able to perform an autonomous flight in an academic setting. In this chapter, after a design and production phase, the experimental results of an autonomous flight are discussed. Besides the essential requirement that the multi-rotor is able to fly, also, requirements such as flexibility, accessibility, and maintainability are of importance. In this chapter thus also the 'soft' requirements are evaluated. The main focus of this chapter is on the proof-of-concept, which is performing an autonomous flight. Autonomous is here defined as that all the control steps are executed on-board of the multi-rotor. We want to verify with this proof-of-concept that all the system components work properly, in order to allow future users to deploy their own control algorithms on the multi-rotor. At the time of writing, path-following, and the online motion planner for drone referee are not (yet) implemented on this multi-rotor. The focus of this experiment is thus on evaluating the functionality of the complete created tool-chain, rather than on performing (acrobatic) flight maneuvers. In Figure 59, the system components that all have to work in order to perform the proof-of-concept are schematically shown.

The remainder of the chapter is organized as follows. In Section 6.1 the systems's setup is described. In Section 6.2 the results of experimental flights are discussed. Section 6.3 provides concluding remarks.

6.1 System Setup

6.1.1 General layout

In the scope of this work a high performance multi-rotor is developed and produced. The multi-rotor is equiped with a modular control-box. This control box contains a Pixhawk flight-controller and an Intel Nuc i7 PC as companion-computer. Furthermore, the multi-rotor is equipped with an optical-flow and ultrasonic sensor, a 180° fish-eye camera, and a RGB-Depth camera. A remote transmitter can be used in order to control the multi-rotor remotely by a human pilot. The remote controller also functions as a failsafe kill-switch, i.e., when the remote-control signal drops, the kill-switch will be activated.

6.1.2 System layout for the proof-of-concept

For the proof-of-concept the algorithm as explained in Section 3 is used. Two programs are running in parallel on the companion-computer on the multi-rotor: the Simulink model which runs accurately at a frequency of $100Hz$ (up to $> 1000Hz$), and the c++ computer vision algorithm running at $30Hz$ restricted by the frame-rate of the camera. The computer vision algorithm determines the position of the multi-rotor with respect to the field of Aruco markers, this result is shared with the Simulink model via an internal UDP port. The Simulink model receives the results from the computer-vision algorithm and processes the data. Simulink, in turn, communicates over USB with the flight-controller using the MAVLink-Simulink interface. The feedback loop is schematically visualized in Figure 62. The processes are running on the companion-computer on-board of the multi-rotor and are started via remote desktop, an example screen-shot is shown in Figure 61.A.

6.2 Experimental Flights

Several experimental flights were executed during the project. A first test is carried out to assert whether the multi-rotor is able to fly using the remote-control, operated by a human pilot. Thereafter, the optical flow sensor is tested. In the final experiment the proof-of-concept is performed, which is performing an autonomous flight. This in cooperates computer vision, inter process communication, real-time control in Simulink, and the implementation of the MAVLink protocol.

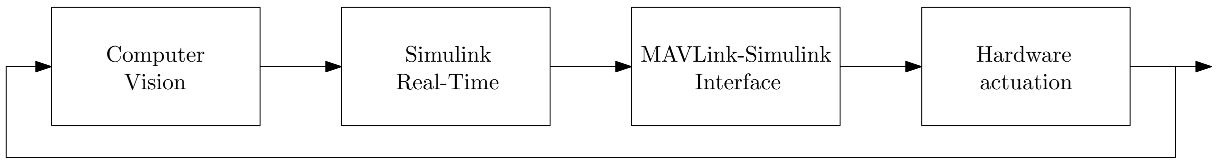


Figure 59: Closing the feedback loop. Schematic overview indicating all system components that are tested in the proof-of-concept.

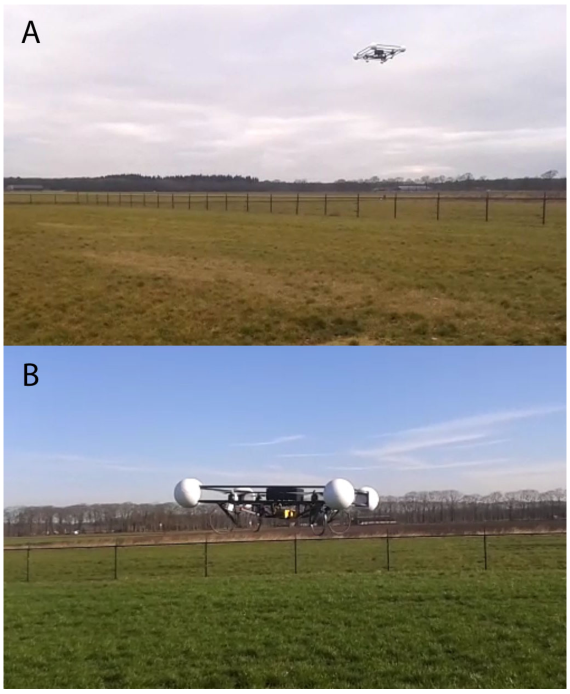


Figure 60: Prototype frame 1 performing a test flight. In A. the multi-rotor is controlled remotely by a human pilot, and in B. the multi-rotor is holding its position using the optical flow sensor.

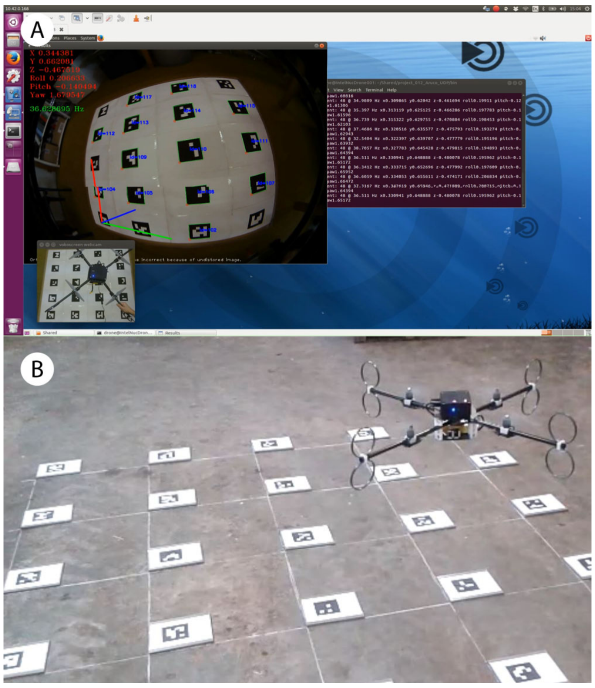


Figure 61: In A is shown a print-screen when logged in the multi-rotor via remote desktop. In B the is shown the multi-rotor performing the proof-of-concept.

6.2.1 Remotely controlled test flight

From first experiments, it can be concluded that the multi-rotor as designed in Section 2 is able to fly. The multi-rotor is in this case controlled manually via the remote-transmitter by a human pilot, as shown in Figure 60.A. in which multi-rotor prototype 1 is shown performing a test flight. With a 4 cell LiPo battery pack with a capacity of 4500mAh, a flight time of approximately 11.5 minutes is reached.

6.2.2 Optical Flow

In a second test, the optical flow sensor is used for position feedback control. The ultrasonic height sensor is used for measuring the flying-height. The default control algorithm available for the Pixhawk is used, holding the multi-rotor fixed in position. The test is executed at about a height of 1.5m as is for example shown in Figure 60.B. Although some tests were successful, it appeared that the signal from the optical flow-sensor often fails, due to bad light conditions and/or other problems.

6.2.3 Crash resistance

The failing optical-flow sensor produced a crash of the vehicle. The expected source of the problem is that when the position updates from the optical flow sensor fail or drop, the internal state estimator starts to predict the wrong states. The consequence is improper control and eventually this led to a crash.

Due to the crash, experimentally it was determined that the frame designed in Section 2.5.5 is not resistant enough against crashes. Although the frame was not extremely damaged and was repaired, in order to avoid future problems a new improved frame is designed as is described in Section 2.5.6. Because of our modular design, only the frame required replacement. The propulsion system and control box remained the same.

From the experiments, was concluded that the new frame is much more robust and maintenance is more simple. In order to emphasize this statement, the new frame survived a crash of about 3 meter on a concrete floor without much damage. Although this crash, in this particular case, caused no damage, we

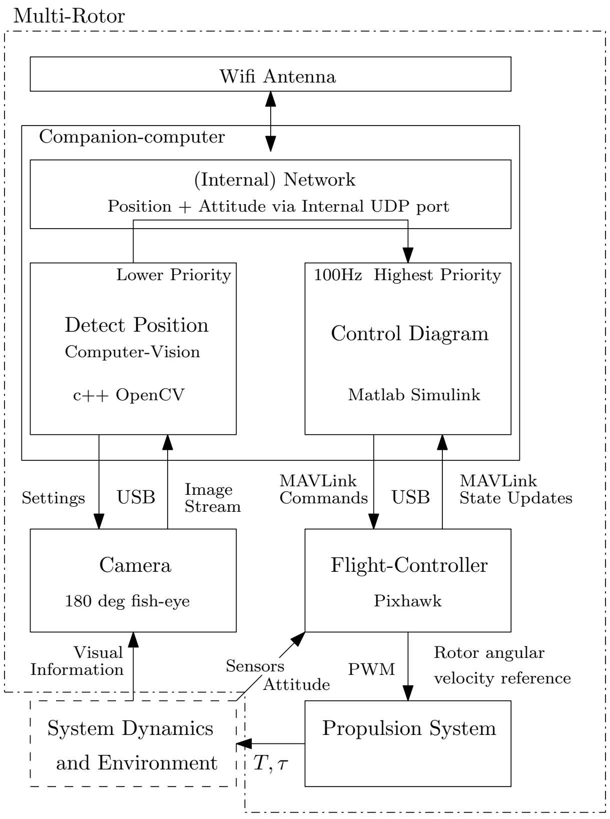


Figure 62: Schematic representation of the integration of the different software components. The scheme presents a feedback loop, where a camera captures images of the environment, a computer vision algorithm running in parallel on the on-board Intel Nuc estimates the multi-rotor position and functions as a sensor charing the results with the in real-time running Simulink control diagram. The Simulink model communicates via the MAVLink protocol with the Pixhawk flight-controller and performs the control actions. The Pixhawk, in turn, controls the inner-loop (attitude) and provides reference signals to the propulsion system.

do not promote to handle the multi-rotor careless, nor do we guarantee that the multi-rotor is able to survive all such crashes in the future.

6.2.4 Experiment: proof-of-concept

In this experimental flight the position of the multi-rotor is obtained from the bottom facing fish-eye camera while the multi-rotor is hovering above a field of Aruco markers. The optical-flow sensor and ultrasonic height sensor are disabled, i.e., also the height of the multi-rotor is estimated by computer vision. In Figure 61.B an image is shown of the multi-rotor during an experimental flight. The recorded experimental flight-data as logged in Simulink via the MAVLink-Simulink interface is shown in Figure 63. In blue the data obtained from the computer vision algorithm is shown. It can be observed from the data that at about 34 seconds after starting the Simulink model the multi-rotor takes off which is indicated with an A in Figure 63. This is a manual step using the remote control. Indicated by time B, the automatic position control is activated. In this stage the multi-rotor flies completely autonomous, i.e., the feedback loop as described in Section 3.5 is activated. The camera captures frames from the environment, the computer vision algorithm determines the multi-rotors position, and the Simulink model instructs the flight-controller via the MAVLink-Simulink interface. At time C in Figure 63, the system is switched back to manual control, and at time D the vehicle lands.

6.2.5 Industial Application

Recently, an industrial drone company (Avular) has showed interest in our implementation of the MAVLink protocol in Matlab Simulink. At the time of writing, in cooperation with TU/e, they are implementing the MAVLink-Simulink interface as developed in this work in their system. When succeeding, experimental results of an industrial applications using the MAVLink-Simulink interface can be expected.

6.2.6 Honors program at TU/e

Besides the MAVLink implementation in Simulink, also the computer-vision algorithm and UDP interface in c++ code appeared to be directly applicable in a different application. The code is at the moment of writing used for an experimental study on sensor fusion in a honors project [47].

6.3 Summary and remarks

In this chapter it was experimentally proven that the developed multi-rotor in this work is able to perform an autonomous flight. With the current battery capacity, a flight time of 11.5 minutes can be reached. It was empirically concluded that the second prototype frame is more resistant to crashes than the first frame. It was also concluded that an optical-flow sensor can be used for position feedback. However, it must be noted that the current optical-flow sensor is not very reliable. A down facing fish-eye camera was used in order to determine the position of the multi-rotor. The position is obtained based on the detection of Aruco markers using computer vision on-board of the multi-rotor. The horizontal position, heading and the fly-height are determined based on the Aruco markers. The Simulink model runs in real-time and is able to communicate with the flight-controller using the MAVLink-Simulink interface. Computer-vision algorithms can run on board of the multi-rotor and they are able to share their results with the Simulink model. The flight results can be logged and real-time visualized via Simulink External mode and remote-desktop. Both, an industrial drone company, and students of the honors program at TU/e have shown interest in parts of the complete design and are currently implementing and experimenting with some of the software components developed in this work.

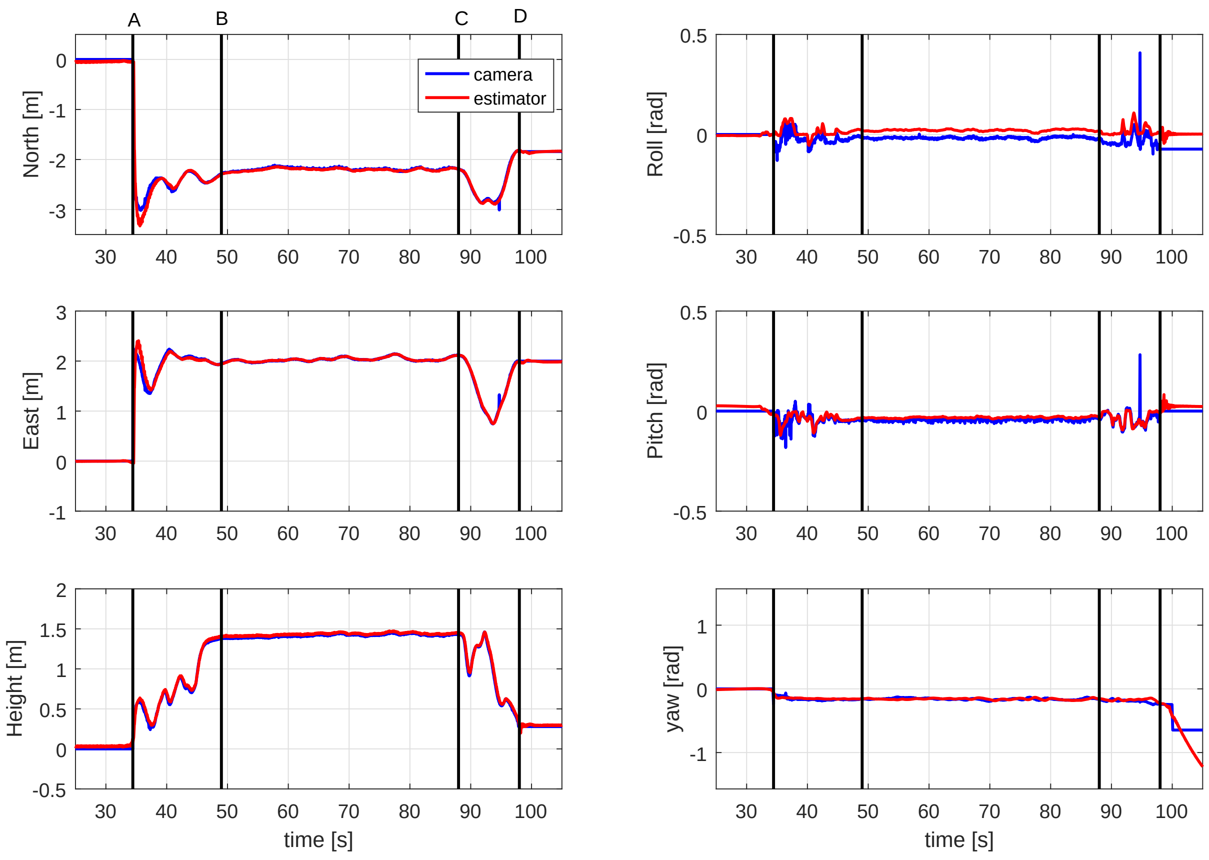


Figure 63: Numerical results recorded during the proof-of-concept. Between time B and C the multi-rotor is flying completely autonomous.

7 Conclusions, Remarks and Future work.

The main goal of this thesis was to develop a computationally powerful multi-rotor platform which is able to perform an autonomous flight and which can be used in an academic context. This means that, for example, the user should be able to deploy their own control algorithms on the multi-rotor. The current commercially available and affordable multi-rotor platforms that can be used in an academic setup have often limited computationally power. In this thesis a computationally powerful multi-rotor was developed and successfully tested in practice. With an on-board powerful companion-computer, basically, a 'flying laptop' was created.

7.1 Conclusions

Hardware design In first place the requirements and sub-requirements were stated for a new multi-rotor design, that also can be used as referee drone. A general system layout of a multi-rotor was described, and a market research was conducted. In this market survey, among others, the system components, energy efficiencies, and frame materials of 20 commercially available multi-rotors were studied. Furthermore, the specifications of 6 commercially available flight-controllers and 6 potential companion-computers were compared. Modularity formed an important guideline for the design of the multi-rotor. At the heart of the design is a modular control-box, accommodating a Pixhawk flight-controller, Intel Nuc i7 PC, and a custom designed PCB for power conditioning. The Intel Nuc i7 is a regular PC with comparable computationally power to a TU/e laptop. The layered modular control-box was designed for, and produced with, 3D printing. In order to obtain insight in the parameters affecting the propulsion of a multi-rotor, experiments with different propeller configurations were executed on a simple, yet working, mock-up multi-rotor model. It was concluded that without knowledge about the exact rotor parameters, which are often not specified for relative cheap commercially available rotor-blades, it is not easy to predict the performance. Therefore, a propulsion system is selected for which the specifications where clearly specified by the producer. Numerous concepts for the design of a frame were proposed in order to find a balance between the system requirements such as reproducibility, weight, safety, and robustness. One of the concepts was selected and worked out in detail, produced, and tested in practice. A conclusion was that during a crash, which can be considered as inevitable when running experimental control software, the pultruded CFRP frame breaks. It can be concluded that the anisotropic properties of CFRP and especially putltruded material can not be neglected. Although the frame was repaired, in order to avoid future problems, a second more robust frame was designed and produced. It was empirically determined that replacing the frame, due to the modular design, was relatively simple. Motivated by conclusions from earlier projects, for this setup, a 180° fish-eye camera was selected, instead of a regular camera with a limited field of view. Furthermore, the multi-rotor is equipped with an optical-flow ground speed sensor, ultrasonic height sensor, and a Intel RealSense R200 RGB-Depth camera. In this work only the fish-eye camera was used, for position feedback.

Software design Matlab Simulink is often used as design and simulation tool for new control algorithms at TU/e. Due to the limited computationally power in most commercially available and affordable multi-rotor platforms, often promising computationally intensive control applications that run without problems on a regular PC, will run much slower or not on the a multi-rotor. Since this multi-rotor is equipped with an on-board normal PC, all the regular software running on a PC is able to run on-board. An important contribution of this work is the development of a MAVLink-Simulink interface allowing Simulink models to communicate with the Pixhawk flight-controller. MAVLink is a widely spread communication protocol for multi-rotors. The communication consist of, for example, control instructions sent from Simulink to the flight-controller and state-updates received by Simulink which were sent by the flight-controller. A clear separation between protocol-layer, transport-layer, and real-time implementation is preserved. The MAVLink-Simulink interface already has found its way to a professional, industrial, drone company, who is currently implementing our protocol layer in their software. On-board computer vision is used in order to give the multi-rotor a perception of its position with respect to the world. The position of the multi-rotor, with respect to a soccer-field, was in simulation, based on real-data, obtained by the detection of lines on the field using data from the fish-eye camera. Because of time considerations, another method is implemented based on Aruco markers. The computer-vision algorithm is developed in c++ using OpenCV and executed on-board. Noteworthy is that the same program is currently/temporary used in a honors project at TU/e.

Path following control Path-following is considered in order to guide a multi-rotor along a planned trajectory. Compared to trajectory tracking, where the vehicle is required to track a time-parameterized trajectory specifying where the vehicle's should be at each time step, in path following the distance to the path is driven to zero while a velocity reference defined along the path is followed. This strategy has rather soft than hard time requirements. The nonlinear equations of motion for a multi-rotor model reveal a natural cascaded structure. Based on singular perturbation theory, the control problem can be separated in: a fast attitude control problem, and a slow position control problem. Two control loops can be constructed: a fast inner-loop for the attitude control and a slow outer-loop for the position and velocity control. Path-following is applied to the relative slow outer-loop dynamics, which reveal a double integrator structure. The control law is directly expressed in the inertial frame compared to the usual used Frenet-Serret frame. The stability of the controlled system is checked using the theories of Lapunov and La Salle. In order to calculate the reference point on the path which is closest to the multi-rotor, and near to the previous position, the use of a local Taylor polynomial approximation is proposed for which an analytical solution exists. This method allows to follow complex, intersecting and overlapping paths. A path generation algorithm is used based on a cubic spline in order to produce a smooth path through a set of way-points. For the attitude control a nonlinear control law based on feedback linearization is proposed in order to track a force which is specified by the path following controller.

Motion planner for drone refereeing In an autonomous referee system in soccer, a multi-rotor can potentially be used for observing the game. In soccer, a ball can not commit a foul by itself, the best position of a drone referee is therefore probably not directly above the ball. According to FIFA ' the best position is one from which the referee can make the right decision', an experienced human referee suggests that the referee should follow the 'bubble of active play' from a distance. In this thesis a concept for a computer algorithm is developed that based on the guidelines of a human referee is able to determine the center of the 'bubble of active play'. The interconnections between players and ball are determined based on scale-space theory, which is a model for the biological visual system. It is proposed to follow with an on-board camera the center of the 'bubble of active play'. From visual inspection, in a special for drone referee designed simulation environment, based on real-data, it can be expected that this method might work quite well in practice.

Experimental results In a final experimental proof-of-concept, a successful autonomous flight was performed. Meant with autonomous, in this work, is that all the sensing, control, and actuation is executed on-board of the multi-rotor, i.e., the multi-rotor does not receive control or state updates from an external system. In the proof-of-concept all the system components have to work seamless together in order to perform a successful flight. Because of time considerations path-following and the conceptual idea for tracking the 'bubble of active play' are not implemented. Basically a feedback loop is created in which the fish-eye camera captures images from the environment. Based on these images, a computer-vision algorithm estimates the position and heading of the multi-rotor. This pose-estimation is shared with the Simulink control algorithm which is performing the high-level/outer-loop control. The Simulink control diagram instructs the low-level/inner-loop flight-controller via the MAVLink-Simulink interface. Finally, the flight-controller performs the attitude-control, by specifying angular rotor velocities for the individual rotors. The rotors provide the forces and torques in order to move the vehicle. Noteworthy is that while the vision algorithm was tested with a regular TU/e laptop, the algorithm was directly deployed on the Intel Nuc companion-computer without adjustments, which was on of the requirements for the new multi-rotor design.

7.2 Recommendations and Future work

An interesting topic for future research might be the system identification of this multi-rotor. The model in this work relies on first principles physics. Effects such as blade flapping, drag, and rotor accelerations are not yet covered and it might be worth it to determine them experimentally in the feature. In this thesis a fairly simple computer-vision algorithm for determining the multi-rotor position in space based on the detection of Aruco markers was used. A disadvantage of this approach is that the system is bounded to fly in an environment with this specific type of markers. Although this method is useful in a prototype phase from a future system might be expected that it is able to determine its position in a random environment. A first step might be to further develop an algorithm that determines its position on a (robot) soccer field. Instead of determining the position of the multi-rotor on-board, an external (indoor) position system might be useful, which also can be used as benchmark. One might decide to further develop the indoor positioning system based on Raspberry Pi's as proposed in the appendix, or rely on another (commercially) system. In the modular control-box, space, and a 5VDC auxiliary connection, is reserved for an indoor positioning receiver, or LED markers. Although the current flight-time might be considered as sufficient during the prototype phase, for a final applications the flight time may need to be extended. This can be, for example, achieved by the selection of a battery with more capacity. When the payload increases or decreases, in the future, one might want to upgrade the propulsion system accordingly, while, probably, the same frame can be used. On the software side, a system architecture as is, for example, proposed in the PDEng project [14] might be implemented. The applications on the PC in the multi-rotor are currently started via remote desktop, optionally this can be automated in the future. In this thesis, the main focus was on the outer-loop control in the form of path-following since for the inner-loop the default Pixhawk flight-controller is considered. Therefore only briefly the design of a nonlinear attitude controller in the form of feedback linearization was discussed. The attitude is parameterized by Euler angles. An disadvantage is that Euler angles suffer from gimbal locks and produce rather complex feedback linearization terms. It can be of interest to study the properties of other parameterizations such as, for example, quaternions. The proposed path following method using the local Taylor polynomial is currently not verified by practical experiments. It might be interesting to discover how this method is working-out in practice. The MAVLink-Simulink interface is currently used by a professional drone company, it might be of interest to follow their experiences and potential improvements. A final, somewhat more general, recommendation is to keep sharing knowledge, collaborating, and informing each-other in the form of, for example, the regularly planned drone meetings at TU/e, which were of great value for this thesis.

References

[1] Inc. (NYSE: IT) Gartner. Gartner's 2016 hype cycle for emerging technologies identifies three key trends that organizations must track to gain competitive advantage. *STAMFORD, Conn., August 16, 2016* `http://www.gartner.com/newsroom/id/3412017`, 2016.

[2] R. Ritz, M. Mueller, and R. D'Andrea. Cooperative quadrocopter ball throwing and catching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4972–4978. IEEE, 2012.

[3] Fabio Gramazio Matthias Kohler Federico Augugliaro, Ammar Mirjan and Raffaello DAndrea. Building tensile structures with flying machines. *International journal of architectural computing*, 2013.

[4] Alex Kushleyev, Daniel Mellinger, and Vijay Kumar. Towards a swarm of agile micro quadrotors. In *Robotics: Science and Systems*, July 2012.

[5] Seattle WA (US) Amazon Technologies, Inc. Us2016375997, unmanned aerial vehicle with a tri-wing configuration. *United States Patent Application Publication*, 2016.

[6] S. Verling, B. Weibel, M. Boosfeld, K. Alexis, M. Burri, and R. Siegwart. Full attitude control of a vtol tailsitter uav. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3006–3012, May 2016.

[7] FESTO. Airpenguin, a group of autonomously flying penguins. *Festo AG & Co. KG*, 2006-2009.

[8] FESTO. emotionspheres, collision-free motion of autonomous systems in a space. *Festo AG & Co. KG*, 2013-2014.

[9] G. A. Folkertsma, W. Straatman, N. Nijenhuis, C. H. Venner, and S. Stramigioli. Robird: A robotic bird of prey. *IEEE Robotics Automation Magazine*, PP(99):1–1, 2017.

[10] Ar.drone 2.0 elite `www.parrot.com/fr/drones/parrot-ardrone-20-elite-edition`. *Parrot*, 2017.

[11] Pierre-Jean Bristeau, Franois Callou, David Vissire, and Nicolas Petit. The navigation and control technology inside the ar.drone micro uav. *IFAC World Congress Milano*, 2011.

[12] Crazyflie 2.0 `www.bitcraze.io`. *Bitcraze*, 2017.

[13] Arash Roomi Tom Zwijgers Cyrano Vaseur, Nestor Hernandez. Robotic drone referee. *2015 MSD PDEng program* `http://cstwiki.wtb.tue.nl/index.php?title=Robotic_Drone_Referee`, 2016.

[14] Joep Wolken Jordy Senden Sa Wang Tim Verdonschot Tuncay Uurlu ler Akarsh Sinha, Farzad Mobini. Autonomous referee system. *2016 MSD PDEng program* `http://cstwiki.wtb.tue.nl/index.php?title=Autonomous_Referee_System`, 2017.

[15] Mathworks, matlab, `www.mathworks.com`. 2016.

[16] R. van de Molengraft, M. Steinbuch, and B. de Kraker. Integrating experimentation into control courses. *IEEE Control Systems*, 25(1):40–44, Feb 2005.

[17] Technische Universiteit Eindhoven Duarte Guerreiro Tom Antunes. Automating grading of assignments in a matlab programming course, `https://mathworks.com/company/newsletters/articles/automating-grading-of-assignments-in-a-matlab-programming-course.html`. *MathWorks Technical Articles and Newsletters*, 2017.

[18] Tu/e notebook `www.tue.nl/studeren/studeren-aan-de-tue/faciliteiten-op-de-campus/notebook/`.

[19] Wayne Johnson. Helicopter theory. *Dover Publications Inc.*, 1994.

[20] Thomas B. Reddy David Linden. Handbook of batteries, third edition. *McGraw-Hill*, 2002.

[21] Jon Verbeke Dries Hulens and Toon Goedemé. How to choose the best embedded processing platform for on-board uav image processing ? *10th International Conference on Computer Vision Theory and Applications*, 2015.

[22] Graham R. Martin. Visual fields and their function in birds. *Centre for Ornithology, School of Biosciences, University of Birmingham*, 2007.

[23] Graham R. Martin. Visual fields in woodcocks scolopax rusticola (scolopacidae; charadriiformes). *Journal of Comparative Physiology June 1994*, 1994.

[24] Herman Bruyninckx Alessandro Saccon Maarten Steinbuch Alex Andriën, René van de Molengraft. Lazy motion planning for robotic manipulators. *Eindhoven University of technology, CST 2016.114*, 2016.

[25] D. Honegger, L. Meier, P. Tanskanen, and M. Pollefeys. An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. In *2013 IEEE International Conference on Robotics and Automation*, pages 1736–1741, May 2013.

[26] B.C.M. van Aert. Control and coordination algorithms for autonomous multi-agent quadrotor systems. *Eindhoven University of Technology, Master thesis, CST 2016.070*, 2016.

[27] Robocup `www.robocup.org`. 2017.

[28] Michael F. Ashby. Materials selection in mechanical design, third edition. *Elsevier Butterworth-Heinemann*, 2005.

[29] Cpu benchmark. *http : / / cpu . userbenchmark . com / Compare / Intel-Core-i7-5557U-vs-Intel-Core-i7-3630QM/ m27386vs626* .

[30] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Pixhawk: A system for autonomous flight using onboard computer vision. *Robotics and Automation (ICRA), 2011 IEEE International Conference*, 2011.

[31] Mavlink source code. *https://github.com/mavlink*, 2017.

[32] Default px4 firmware with lpe position estimator,`https://github.com/PX4/Firmware/releases`/px2fmu-v2_lpe.px4. 2017.

[33] Ubuntu 16.04 lts, `https://www.ubuntu.com/`. 2016.

[34] The qt company, qt creator, `https://www.qt.io/ide`. 2016.

[35] Open cv, `http://opencv.org/`. 2017.

[36] An open source remote desktop protocol(rdp) server, `http://www.xrdp.org/`. 2016.

[37] Griffon. How to install xrdp on ubuntu 16.04 easy way, `http://c-nergy.be/blog`. 2016.

[38] As5669a. *SAE Aerospace Standard*, 2009.

[39] Arthur Ketels Rene van de Molengraft. Ebox, 'libtimer_posix.c' library. *http://cstwiki.wtb.tue.nl/index.php?title=E-box*, 2012.

[40] Google tango, mobile computer vision. *https://get.google.com/tango/*.

[41] Microsoft hololens. *https://www.microsoft.com/en-us/hololens*.

[42] Tech united `http://www.techunited.nl`. *Eindhoven University of Technology*, 2017.

[43] B.A.C. van de Schoot. Robust visual-based detection under changing illuminations. *Traineeship report, TU/e, Mechanical Engineering, Dynamics and Control*, 2016.

[44] Nicolai Petkov George Azzopardi. Trainable cosfire filters for keypoint detection and pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(2):490–503, Feb 2013.

[45] S. Garrido-Jurado, R. Mu noz Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280 – 2292, 2014.

[46] Richard Szeliski. *Computer Vision: Algorithms and Applications.* http://szeliski.org/Book/, 2010.

[47] Honors students from Eindhoven University of Technology. Firefly eindhoven, `http://fireflyeindhoven.nl`. 2017.

[48] Claude Samson Alain Micaelli. Trajectory tracking for unicycle-type and two-steering-wheels mobile robots. *Technical Report 2097 INRIA*, 1993.

[49] P. Encarnacao and A. Pascoal. 3d path following for autonomous underwater vehicle. In *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No.00CH37187)*, volume 3, pages 2977–2982 vol.3, 2000.

[50] Pedro Gomes Carlos Silvestre Rita Cunha, Duarte Antunes. A path-following preview controller for autonomous air vehicles. pages p. 1–21. Proceedings of the AIAA Guidance, Navigation, and Control Conference 2006, 21-24 August 2006, Keystone, Colorado., 2006.

[51] Peter Corke Robert Mahony, Vijay Kumar. Multirotor aerial vehicles, modeling, estimation and control of a quadrotor. *IEEE Robotics & Automation Magazine*, September 2012.

[52] Hassan K. Khalil. Nonlinear systems, third edition. *Pearson Education Limited*, 2014.

[53] P.L.M. Rooijakkers D.J. Guerreiro Tom'e Antunes. System identification and control of a quadcopter. (internship report). *Technische Universiteit Eindhoven*, 2016.

[54] Abbas Emami-Naeini Gene F. Franklin, J. David Powell. *Feedback Control of Dynamic Systems Sixth Edition*. Pearson, 2010.

[55] Claude Lobry Tewfik Sari. Singular perturbations methods in control theory, stabilization of cascade systems, `http://www.math.uha.fr/sari/publi.html`. *Contrle non linaire et Applications, Travaux en Cours no. 64, Hermann, Paris*, 2005.

[56] Sean Gallagher. Franz beckenbauer believes referees will be replaced by drones as the use of technology in football continues to evolve. *MailOnline www.dailymail.co.uk*, 2014.

[57] Tanaka-Ishii K. Okuno H.G. Akita-J. Nakagawa Y. Maeda K. Nakadai K. Kitano H. Frank, I. And the fans are going wild! sig plus mike. *Stone, P., Balch, T., Kraetzschmar, G.K. K. (eds.) RoboCup 2000. Springer, Heidelberg*, 2001.

[58] C.A. Lopez Martinez M. Briegel H.H.C.M. van Wesel J.P.J. Groenen O. Hendriks O.F.C. Klooster R.P.T. Soetens M.J.G. van de Molengraft F.B.F. Schoenmakers, G. Koudijs. Tech united eindhoven team description. 2013.

[59] G.J.L. Naus M. Briegel M.J.G. van de Molengraft M. Steinbuch Gerald Koudijs, F.B.F. Schoenmakers. Autoref, an automatic referee for robot football. *Bachelor final project*, 2013.

[60] Danny Zhu, Joydeep Biswas, and Manuela Veloso. *AutoRef: Towards Real-Robot Soccer Complete Automated Refereeing*, pages 419–430. Springer International Publishing, Cham, 2015.

[61] Law 5, the referee. *FIFA https://www.fifa.com/mm/document/afdeveloping/refereeing/law_5_the_referee_en_47411.pdf*, 2004.

[62] Jim Gordon. How do referees position themselves in a football match? *Quara https://www.quora.com/How-do-referees-position-themselves-in-a-football-match*, 2016.

[63] Bart M. Haar Romeny. Front-end vision and multi-scale image analysis. *Kluwer Academic Publishers*, 2003.

[64] Tech united simulator, `http://www.techunited.nl/wiki/index.php?title=Simulator`.

[65] RoboCup. Msl logging database `http://wiki.robocup.org/MSL_logging_database`. 2016.

[66] Tech united data, soccer field dimensions, `http://robocup.wtb.tue.nl/svn/techunited/trunk/src/Turtle2/`.

[67] Linux manual. *https://linux.die.net*.

[68] Ms windows manual. *https://msdn.microsoft.com*.

[69] T. J. Bridges H. Alemi Ardakani. Review of the 3-2-1 euler angles: a yawpitchroll sequence. *Department of Mathematics, University of Surrey, Guildford GU2 7XH UK*, April 2010.

[70] L. M. Argentim, W. C. Rezende, P. E. Santos, and R. A. Aguiar. Pid, lqr and lqr-pid on a quadcopter platform. In *2013 International Conference on Informatics, Electronics and Vision (ICIEV)*, pages 1–6, May 2013.

[71] Francesco Sabatino. Quadrotor control: modeling, nonlinear control design, and simulation. Master's thesis, KTH Electrical Engineering, 2015.

[72] J.P. Hespanha. Linear systems theory. *Princeton University Press*, 2009.

[73] Bernard Friedland. Control system design, an introduction to state-space methods. *Dover Publications, Inc.*, 1986.

A List of tables

A.1 Table commercially available multi-rotors

id	Product name	Type	Mass [gram]	Flight-Time [minute]	Battery
1	Walkera QR X900 Hexacopter	Industry/Aplication	9950*	13*	LiPo 6s 16000mAh
2	Spiderdrone Security	World record	4712	126,17	LiPo 6s 24000mAh
3	dji Inspire 2	Filmmaking	3290	27	LiPo 6s 4280mAh
4	Walkera VOYAGER 4	Industry	3250	20*	LiPo 6s 4500mAh
5	dji Inspire 1	Filmmaking	2935	18	LiPo 6s 5700mAh
6	TBS endurance pro long range	Film rig	2500*	25	LiPo 6s 4000mAh
7	dji Matrice 100	Flight platform	2431	28	LiPo 6s 5700mAh
8	TBS discovery pro	Film rig	1750*	10*	LiPo 4s 4000mAh
9	Typhoon H	Hexacopter	1695	25	LiPo 4s 5400mAh
10	dji Phantom 4 pro	Video	1388	30	LiPo 4s 5870mAh
11	dji Phantom 3 pro	Video	1280	23	LiPo 4s 4480mAh
12	dji Mavic	FPV	734	24	LiPo 3s 3830mAh
13	Walkera Runner 250 PRO	Racing drone	644*	11*	LiPo 3s 2200mAh
14	Walkera AiBao	AR drone	570	18	LiPo 2s 5200mAh
15	Walkera Furious 215	Racing drone	475*	8,5*	LiPo 4s 1300mAh
16	Parrot AR.Drone 2.0	AR drone	420	12	LiPo 3s 1000mAh
17	Breeze	Flying camera	385	12	LiPo 3s 1150mAh
18	TBS XRACER Micro	FPV Racer	55	5,5	LiPo 1s 600mAh
19	Crazyflie 2.0	Development	29	7	LiPo 1s 240mAh
20	Cheerson CX-10	Nano quadcopter	12	6*	Li-ion 1s 100mAh

Basic charateristics of commercial available multirotors. Showing the type, mass, measurements, and frame material types. Entries indicated with * are derived or interped from a range or multiple sources and may differ from the real value.

A.2 Table commercially available multi-rotors (continued 1)

id	Product name	Type	Mass gram	Diagonal mm	Material arm
1	Walkera QR X900 Hexacopter	Industry/Aplication	9950*	900	CFRP tubes
2	Spiderdrone Security	World record	4712	1200	CFRP tubes
3	dji Inspire 2	Filmmaking	3290	605	CFRP tubes
4	Walkera VOYAGER 4	Industry	3250	657*	CFRP tubes
5	dji Inspire 1	Filmmaking	2935	581	CFRP tubes
6	TBS endurance pro long range	Film rig	2500*	500*	CFRP tubes
7	dji Matrice 100	Flight platform	2431	650	CFRP tubes
8	TBS discovery pro	Film rig	1750*	450*	CFRP tubes
9	Typhoon H	Hexacopter	1695		CFRP tubes
10	dji Phantom 4 pro	Video	1388	350	Plastic*
11	dji Phantom 3 pro	Video	1280	350	Plastic*
12	dji Mavic	FPV	734	335	Plastic*
13	Walkera Runner 250 PRO	Racing drone	644*	250	CFRP plate
14	Walkera AiBao	AR drone	570	325*	Plastic
15	Walkera Furious 215	Racing drone	475*	215	CFRP plate
16	Parrot AR.Drone 2.0	AR drone	420	360	CFRP tube
17	Breeze	Flying camera	385	280*	Plastic*
18	TBS XRACER Micro	FPV Racer	55	120	Plastic
19	Crazyflie 2.0	Development	29	92	PCB + Plastic
20	Cheerson CX-10	Nano quadcopter	12	45	PCB + Plastic

Basic characteristics of commercial available multi-rotors. Showing the type, mass, measurements, and frame material types. Entries indicated with * are derived or interpreted from a range or multiple sources and may differ from the real value.

A.3 Table commercially available multi-rotors (continued 2)

id	Product name	Time of Flight minutes	Battery	Motor	Rotor size
1	Walkera QR X900 Hexacopter	13*	LiPo 6s 16000mAh		17x5,5 inch
2	Spiderdrone Security	126,17	LiPo 6s 24000mAh	100KV	28x9 inch
3	dji Inspire 2	27	LiPo 6s 4280mAh		15x5 inch
4	Walkera VOYAGER 4	20*	LiPo 6s 4500mAh		382 mm
5	dji Inspire 1	18	LiPo 6s 5700mAh	3510H/350KV	13x4,5 inch
6	TBS endurance pro long range	25	LiPo 6s 4000mAh	400kV	15x5 inch
7	dji Matrice 100	28	LiPo 6s 5700mAh	3510	13x4,5 inch
8	TBS discovery pro	10*	LiPo 4s 4000mAh	2216 900KV	9x5 inch
9	Typhoon H	25	LiPo 4s 5400mAh		
10	dji Phantom 4 pro	30	LiPo 4s 5870mAh	2312/??	9,4x5 inch
11	dji Phantom 3 pro	23	LiPo 4s 4480mAh	2312/960KV	9,4x5 inch
12	dji Mavic	24	LiPo 3s 3830mAh		8,3x3 inch
13	Walkera Runner 250 PRO	11*	LiPo 3s 2200mAh		143 mm
14	Walkera AiBao	18	LiPo 2s 5200mAh		186 mm
15	Walkera Furious 215	8,5*	LiPo 4s 1300mAh	2500KV	127 mm
16	Parrot AR.Drone 2.0	12	LiPo 3s 1000mAh		9 inch*
17	Breeze	12	LiPo 3s 1150mAh		
18	TBS XRACER Micro	5,5	LiPo 1s 600mAh	20x8,5* brushed	56 mm
19	Crazyflie 2.0	7	LiPo 1s 240mAh	4x7* brushed	45 mm
20	Cheerson CX-10	6*	Li-ion 1s 100mAh		30mm

Basic charateristics of commercial available multirotors. Entries indicated with * are derived or interped from a range or multiple sources and may differ from the real value.

A.4 Table commercially available multi-rotors (continued 3)

id	Product name	Source
1	Walkera QR X900	`http://www.walkera.com/index.php/Goods/canshu/id/22.html`
2	Spiderdrone Security	`http://spiderdronesecurity.com`
3	dji Inspire 2	`http://www.dji.com/inspire-2/info`
4	Walkera VOYAGER 4	`http://www.walkera.com/v4/#v41`
5	dji Inspire 1	`http://www.dji.com/inspire-1/info`
6	TBS LONG RANGE SET	`http://team-blacksheep.com/products/product:802`
7	dji Matrice 100	`http://www.dji.com/matrice100/info`
8	TBS DISCOVERY PRO	`http://www.team-blacksheep.com/tbs-discovery-pro-manual.pdf`
9	Typhoon H	`http://us.yuneec.com/typhoon-h-specs`
10	dji Phantom 4 pro	`https://www.dji.com/phantom-4-pro/info`
11	dji Phantom 3 pro	`https://www.dji.com/phantom-3-pro/info`
12	dji Mavic	`http://www.dji.com/mavic/info`
13	Walkera Runner 250 PRO	`http://www.walkera.com/index.php/Goods/canshu/id/43.html`
14	Walkera AiBao	`http://www.walkera.com/index.php/Goods/info/id/42.html`
15	Walkera Furious 215	`http://www.walkera.com/index.php/Goods/canshu/id/45.html`
16	Parrot AR.Drone 2.0	`https://www.parrot.com/us/drones/parrot-ardrone-20-elite-edition#special-editions`
17	Breeze	`http://us.yuneec.com/breeze-specs`
18	TBS XRACER Micro FPV	`http://team-blacksheep.com/tbs-xracer-manual.pdf`
19	Crazyflie 2.0	`https://www.bitcraze.io/crazyflie-2/`
20	Cheerson CX-10	`http://www.cheerson.com`

Data-sheet sources of the commercially available multi-rotors

A.5 Table commercially available flight-controllers

id	Flight controller	Firmware	(main)Processor	Memory/ Storage	Sensors	Connec-tivity	Price
1	Ardupilot APM 2.8	Ardupilot	ATmega 2560 8-bit 16MHz	8KB/ 4MB	+	++	€€
2	Naze32 Full	Cleanflight Baseflight	ARM Cortex M3 32-bit 72MHz	20KB/ 16MB	+	+	€
3	CC3D	Libre Pilot	ARM Cortex M3 32-bit 72MHz	20KB/ 4MB	+	+	€
4	Pixhawk	PX4 Ardupilot Simulink	ARM Cortex M4F 32-bit 168MHz	256KB/ 8GB	++	+++	€€€
4	Pixhawk 2.1	PX4 Ardupilot Simulink	ARM Cortex M4F 32-bit 168MHz	256KB/ 8GB	+++	+++	€€€
4	Qualcomm Snap-dragon Flight	PX4 Linaro +OpenCV	Quad-core Krait 2.26GHz	2GB/ 32GB	++++	++++	€€€€

Selected set of commercial available open-source flight-controllers.

A.6 Table commercially available companion-computers

id	Name	(Main) Processor	Core/ Bit-size/ Frequency	Memory/ Storage	Graphical	Mass gram	Connec-tivity	Price
1	Raspberry Pi 3B	ARM Cortex-A53	4/64/1.2GHz	1GB/SD	Broadcom VideloCore IV	45	++++	€
2	Odroid C2	ARM Cortex-A53	4/64/1.5GHz	2GB/SD	Mali-450	56	++	€
3	Odroid XU4	ARM Cortex-A15 ARM Cortex-A7	4/32/2GHz 4/32/1.4GHz	2GB/SD	ARM Mali-T628	38+	++	€€
4	Intel Aero Compute	Intel Atom x7-Z8750	4/64/2.56GHz	4GB/32GB	Altera Max 10 FPGA	60	++++	€€€
5	Intel Nuc	Intel Core i7-5557U	2/64/3.4GHz	16GB/SSD	Iris graphics 6100	190	++++	€€€€
6	Nvidia Jetson TX1	ARM Cortex-A53	4/64/1.73GHz	4GB/16GB	256-core CUDA	88+	++++	€€€€

Selection of commercial available single-board computers that can be used as companion computer on-board of a multirotor.

A.7 Table commercially available types of sensors

Sensor	$\dot{\phi}$, $\dot{\theta}$	$\dot{\psi}$	ϕ, θ	ψ	$\ddot{x}$,$\ddot{y}$	$\ddot{z}$	$\dot{x}$,$\dot{y}$	$\dot{z}$	x,y	z
IMU acceleration			+		+	+				
IMU gyroscope	+	+								
Magnetometer				+						
Barometric sensor										+
Airspeed sensor (Pitot tube)							+	+		
GPS									+	+
Laser sensor (on-board, single beam)									+	+
Laser range scanner (on-board)			+	+					+	+
Ultrasonic sensor (on-board)									+	+
Optical flow (on-board bottom facing camera)		+					+	+		
Feature recognizing camera (on-board)			+	+					+	+
3D/Depth camera (on-board)			+	+					+	+
Ultra-wide-band IPS									+	+
Optical tracking (off-board)			+	+					+	+
Top camera (off-board, indoor)				+					+	

Selection of commonly used sensors in multi-rotor applications. Indicated with a + symbol are the system states measured (directly) by the sensor. ψ, θ, and ϕ represent the Euler-angles roll, pitch, and yaw respectively, x, y, z represent north, east and fly-height respectively

A.8 Technical specifications of the fish-eye camera

Camera	ELP-USB8MP02G-L180
Sensor	Sony (1 / 3.2 ") IMX179
Lens	180 deg Fish-Eye
Format	MJPEG / YUY2 /
USB Protocol	USB2.0 HS / FS
Performance	3264X2448 MJPEG 15fps YUY2 2fps 2592X1944 MJPEG 15fps YUY2 3fps 2048X1536 MJPEG 15fps YUY2 3fps 1600X1200 MJPEG 10fps YUY2 10fps 1280X960 MJPEG15fps YUY2 10fps 1024X768 MJPEG 30fps YUY2 10fps 800X600 MJPEG 30fps YUY2 30fps 640X480 MJPEG 30fps YUY2 30fp
Mass	12 gram
Power	DC 5V 150mA

Properties of 180 deg fish-eye camera.

A.9 Technical specifications of RGB-Depth camera

Intel RealSense R200		
Dimensions	130x20x7mm	
Mass	36gram	
Mass total (+cable)	60gram	
Connection	USB3	
Camera	**color**	**depth/IR**
Active Pixels	1920x1080	640x480
Aspect ratio	16:9	4:3
Frame rate	30 fps	60 fps
Field of view (D x V x H)	77°x43°x70°	70°x46°x59°

Properties of the RealSense R200 rgb-depth camera.

A.10 Technical specifications of the selected propulsion system

Property	**DJI E310**
Max Thrust [gram]	3200
Propeller Size [inch]	9.4×5.0
Motor	$23 \times 12\,960 KV$
Mass [gram]	464
Battery type	4 cell LiPo

Properties of the selected propulsion system.

A.11 Energy density LiPo batteries

Product	Cells	Capacity mAh	Energy Wh	Mass gram	Energy Density Wh/kg
Turnigy 1000mAh 25-35C	3	1000	11.1	80	138.75
Turnigy 3000mAh 30-40C	4	3000	44.4	306	145.1
Zippy 4500mAh 35C	4	4500	66.6	444	150
Average					144.6

Energy density of three lithium polymer (LiPo) batteries. The amount of energy in the battery is calculated using (3). The mass of the battery is measured and includes the mass of the cable connections.

B Bill Of Materials

Bill Of Material (BOM) list of materials and components used in the multi-rotor according to Prototype 2. All components are included, inclusive the accessory such as battery charger, Wifi router, remote control, cables, etc. The prices are without tax and shipping.

Qty	Description	Supplier	Total Price
1	Dji E310 (4*Motor/ESC; 4 pair props; Accessories pack)	store.dji.com	€148.-
1	Pixhawk 1	3DR	€*150.-
2	Carbon round tube 45deg, wound, 3k-PW (⌀16/⌀14)x1000mm	R&G, r-g.de	€46.10
1	Carbon Fibre Sheet ECOTECH 150x150x1.5 mm	R&G, r-g.de	€7.76
2	Carbon Fibre Sheet ECOTECH 150x150x1.0 mm	R&G, r-g.de	€5.26
1	HDPE White 300x300x10 mm	Hans Otten Staal	€*15.-
1	Intel RealSense R200	Intel	€*100.-
1	PX4Flow (Optical Flow Sensor)	Unmanned Techs	€100.81
1	DF13 4 Position Connector 30cm (pack of 5)	Unmanned Techs	€4.89
1	ELP Kamera Modul USB 8MP Fisheye mit 180 Grad Objektiv	Amazon.de	€68.99
1	ControlBox Nuc part	ShapeWays	€59.10
1	ControlBox Bottom	ShapeWays	€44.04
1	Power Connector Block	ShapeWays	€12.08
1	DLCK-89454 Twin Antenna WLAN MHF IV/HSC MPXHP32 Compatible	Onlinekabelshops	€12.99
1	Turnigy TGY-i6 AFHDS Transmitter and 6CH Receiver (Mode2)	HobbyKing	€44.08
1	HobbyKing C3 50W Charger/Discharger (AC/DC)(EU Plug)	HobbyKing	€31.65
1	Turnigy 5A (8-26v) SBEC for Lipo	HobbyKing	€4.81
1	ZIPPY Compact 4500mAh 4S 35C Lipo pack	HobbyKing	€43.48
1	Turnigy 3000mAh 4S 30C Lipo Pack	HobbyKing	€27.50
1	Charge Cable Male XT60-4mm Banana plug	HobbyKing	€3.43
1	Nylon XT60 Connectors Male/Female (5 pairs) GENUINE	HobbyKing	€4.80
1	JST-XH 4S Wire Extension 20cm (10pcs/bag)	HobbyKing	€5.03
1	2.5mm-5.5mm DC Plug Jack	HobbyKing	€0.29
1	Lipo Voltage Checker	HobbyKing	€2.60
1	Micro USB Cable	HobbyKing	€1.15
2	Screw Socket Head Hex M3x14mm Steel	HobbyKing	€0.68
1	Various Screws, bolts, nuts, M3 galvanized steel	Fabory	€*2.-
1	TP-Link WiFi router TL-WR802N N300	Conrad	€19.01
8	Skottky diode NXP PMEG3050EP,115	Conrad	€5.49
1	Various Plastic Screws, DIN912 M6x40, DIN84 M3x30, DIN84 M3x10, DIN934 M3	Conrad	€*4.-
1	Various Resistors, Capacitors, Cables, PCB-Board, Cable-binders, Etc.	Conrad	€*5.-
1	Intel Nuc NUC5i7RYH	Bol.com	€519.00
1	Storage: crucial CT275MX300SSD4 (275GB)	Bol.com	€83.50
1	Memory: crucial CT8G3S160BM (8GB)	Bol.com	€62.99
1	Mini HDMI to HDMI plug	Bol.com	€6.61
8	Vibration isolator 22.TP-A010X8-10M4	Verpas	€4.88
4	Vibration isolator 22.A010X08M4X10	Verpas	€3.05
8	Suspension Rings, PVC tube 110 x 3,2 mm, SN4/8,	Wildkamp	€*1
			€1721.44

where * indicates that the price is estimated.

C Alternatively proposed indoor positioning systems

C.1 Optical tracking using Raspberry Pi

In this concept, multiple, relative cheap, Raspberry Pi's with cameras are placed around the flight area, comparable to an optical tracking system, as is graphically shown in Figure 64.A. The only task of each Raspberry Pi is to detect markers in the 2D image provided by the connected camera. The markers can be for example LED's mounted on the multi-rotor. All Raspberry Pi's are connected to a central computer. Each Raspberry Pi sends only the detected 2D marker positions and a time-stamp to the central computer, i.e., no video needs to be transfered over the network and the central computer is not concerned by the image processing. The central computer only needs to determine the 3D marker position based on the 2D projections obtained by different cameras. Figure 64.B shows the simulation of this process. The central computer uses here an algorithm that re-projects the detected 2D markers in the form of (red-colored) lines in the 3D environment. A cluster of (near) line-crossings arises where the original marker was located. Potentially, advantage can be taken from the knowledge about where the markers are placed on the rigid multi-rotor frame, i.e., no separate markers need to be tracked only groups with known distance between each-other. In order to synchronize the time of all Raspberry Pi's potentially a flashing marker, visible by all cameras, functions as clock. It is Experimentally determined, in the simulation, that proper calibration of the cameras is essential. In this work, the system is not further elaborated, however it might be an inspiration for a future (open-source) project.

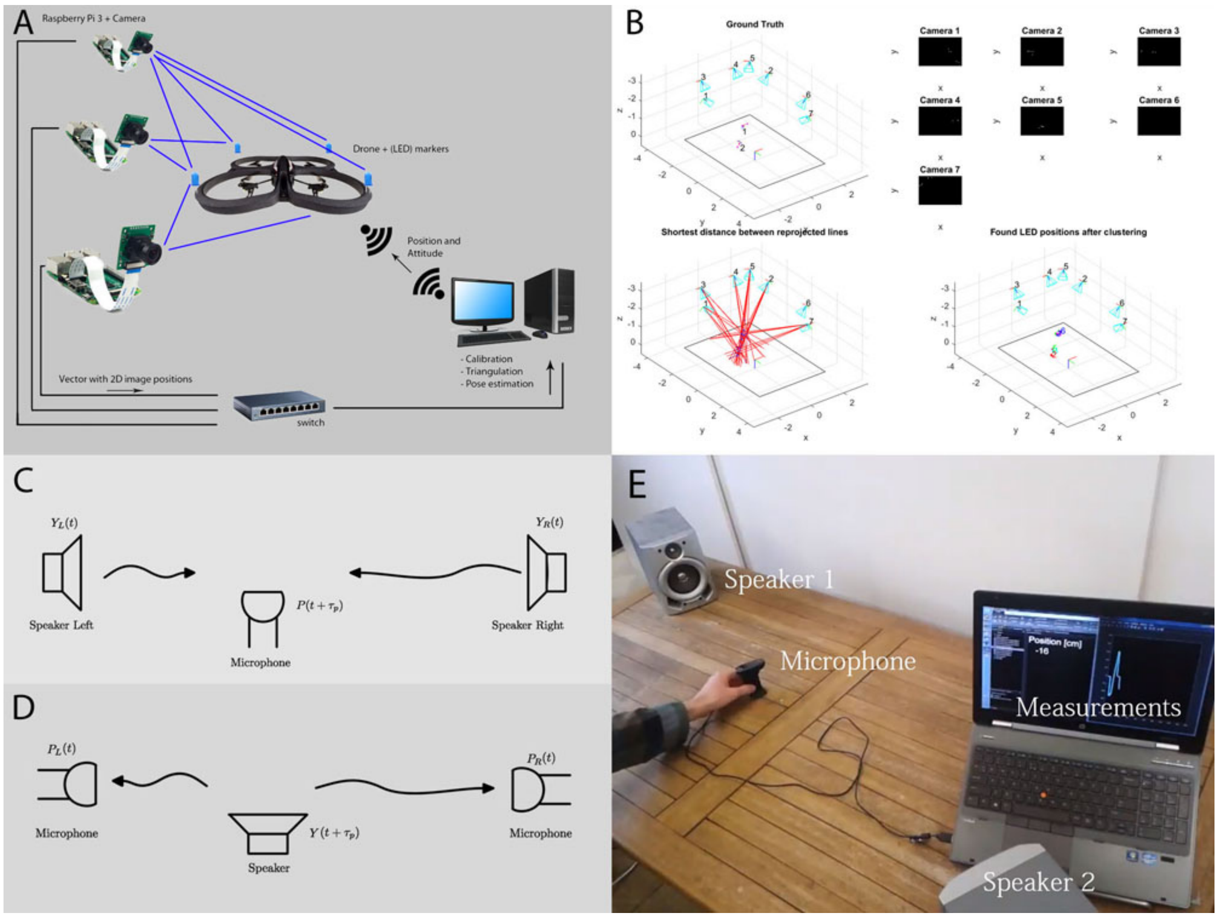


Figure 64: Several proposed alternatives for indoor position systems (IPS). In A is shown a schematic overview of a optical tracking system using several Raspberry Pi modules, in B the simulation results of system A are shown. In C a schematic overview of a proposed IPS system based on the difference of arrival of sound is shown. In D an approach is shown where the position of the sound source is estimated with multiple microphones. In E the experimental setup of concept C is shown.

C.2 Determining position based on sound

In this method we use the difference in time of arrival of sound propagating in air. As is schematically shown in Figure 64.C, time synchronized (ultrasonic) speakers at fixed positions broadcast on each speaker a distinct but known sound sequence. The microphone between the speakers, representing the multi-rotor, receives the sound waves of the different speakers. Besides receiving sound waves, the multi-rotor knows the distinct broadcasted sequences. The multi-rotor is thus able to detect the sound sequence produced by each speaker. Depending on the position of the multi-rotor, the sound sequence of each speaker is received with a different lag, assuming the constant propagation speed of sound waves in air. From this difference of arrival the multi-rotor will be able to calculates its position in 3D space.

Experiment In order to test this concept in practice, a regular stereo speaker set and microphone (embedded in a web-cam) with a sample rate of 44kHz is used in the experimental setup, as is shown in Figure 64.E. On both speakers a distinct white noise sequence is played. The microphone receives the sound waves and by using correlation with the two known white-noise sequences for each sequence a time-delay is obtained. The difference between these time-delays can be, via the constant propagation speed of sound, related to an off-center distance. It can be conclude from the experimental setup that an accuracy of about $1cm$ can be obtained, limited by the sample rate of the microphone (theoretically: $340ms^{-1}/44kHz = 0.77cm$). Furthermore a sufficient perturbing signal should be used, in order to detect the time-delay sufficiently fast.

C.3 Determining position based on sound (reversed)

Conceptually the measurement system, using the difference of time of arrival of sound, as described in Section C.2, can be reversed. In this case the multi-rotor is the sound source. Around the multi-rotor, multiple time synchronized microphones, are placed at fixed positions, as is graphical visualized in Figure 64.D. In this case it is not necessary to have knowledge about the sound sequence produced by the multi-rotor. However, a lag can be calculated between the different sequences received by the different microphones. With a constant propagation speed of sound in air, the position of the multi-rotor can be calculated. A multi-rotor by itself produces already a significant sound, however it must be studied whether this can be used. This conceptual method is not further tested by simulation or experiments, however the concept might be of inspiration for a future system.

D MAVLink Protocol

D.1 Message definitions

The MAVLink protocol consist of a set of MAVLink messages which can be send over a communication channel. The communication channel can be for example, wired via usb or ethernet, or wireless via wifi or bluetooth. The message-structure is defined in extensible markup language (XML) format. The file(s) containing the message definitions can be found on the MAVLink Github [31]. More specific, the most used common messages and corresponding enums can be found in the common.xml file. In addition, there are also platform specific messages that are defined in separate files. In our setup, we only use the common messages. Below is, in xml-format, an example message-definition shown

```
<message id="83" name="ATTITUDE_TARGET">
   <description>Reports the current commanded attitude of the vehicle as specified by
        the autopilot. This should match the commands sent in a SET_ATTITUDE_TARGET
       message if the vehicle is being controlled this way.</description>
   <field type="uint32_t" name="time_boot_ms">Timestamp in milliseconds since system
       boot</field>
   <field type="uint8_t" name="type_mask">  Mappings: If any of these bits are set,
       the corresponding input should be ignored: bit 1: body roll rate, bit 2: body
       pitch rate, bit 3: body yaw rate. bit 4-bit 7: reserved, bit 8: attitude</
       field>
   <field type="float[4]" name="q">Attitude quaternion (w, x, y, z order, zero-
       rotation is 1, 0, 0, 0)</field>
   <field type="float" name="body_roll_rate">Body roll rate in radians per second</
       field>
   <field type="float" name="body_pitch_rate">Body roll rate in radians per second</
       field>
   <field type="float" name="body_yaw_rate">Body roll rate in radians per second</
       field>
   <field type="float" name="thrust">Collective thrust, normalized to 0 .. 1 (-1 .. 1
        for vehicles capable of reverse trust)</field>
</message>
```

As can be observed, a message is typically described by a message id-number (0 . . . 255), a description, and data fields of a specific type. Each field-type can occur in scalar-form or vector-form. The possible field-types are indicated in Table 3. The number of bytes is explicitly indicated in this table. This will be of importance for field sorting described in Section D.1.3. Custom messages can be added by including the definition in the XML file. It is possible to select a free message id-number for this purpose. However, a drawback of the current protocol is that the number of messages-id's is limited due to the use of a single byte for the message-id. A solution to deal with this limitation is to make use of the general messages such as message-id `76 COMMAND_LONG` that sends more or less a message within a message. Message-id 76 is for example used for sending the ARM command as well as the CHANGE MODE command, and is further explained in Section D.3.

MAVLink Type	Description	Number of bytes	Matlab	Simulink
char	Characters / strings	1	char	char
uint8_t	Unsigned 8 bit	1	uint8	uint8
int8_t	Signed 8 bit	1	int8	int8
uint16_t	Unsigned 16 bit	2	uint16	uint16
int16_t	Signed 16 bit	2	int16	int16
uint32_t	Unsigned 32 bit	4	uint32	uint32
int32_t	Signed 32 bit	4	int32	int32
uint64_t	Unsigned 64 bit	8	uint64	2×uint32
int64_t	Signed 64 bit	8	int64	2×int64
float	IEEE 754 single precision floating point	4	single	single
double	IEEE 754 double precision floating point	8	double	2×single
uint8_t_mavlink_version	Unsigned 8 bit field	1	uint8	uint8

Table 3: List of possible field types in a MAVLink message. Note that the 64 bit types are mapped two times a 32 bit type because Simulink does not support 64 bit types.

In our work we read-in the XML file using Matlab and store the messages in a Matlab struct of the form

```
MAVLINK_MESSAGE{1...256}
    -id                     uint8           message-id
    -name                   String          message-name
    -description            String          message-description
    -field{1...n}
        --name              String          field-name
        --arrayLength       uint8           0 if scalar else array-length
        --type              String          field-type
        --enum              String          name of set of enums
        --description       String          field-description
        --bitSize           uint8           bits of the type
        --typeString        String          field-type used for calculate seed
    -sortOrder              uint8[1...n]    ordering of the data fields in payload
    -payloadLength          uint8           length of the payload in bytes
    -seed                   uint8           message seed number
```

D.1.1 MAVLink Message Construction

A message, as defined in the previous section, must be packed before it can be send over a serial connection. A complete single MAVLink message is build up from an array of bytes. Each individual message contains a header (first 6 bytes), payload (n bytes), and a checksum value (2 bytes). The structure is visualized in Figure 65. The header contains information about the sender, the type of message (message-id) and message length. The payload contains the data-fields as defined in the XML-file, however, the order of the fields is different from the definition in the xml-file, as will be discussed in Section D.1.3. Finally, the checksum is calculated over part of the header, the payload and a special seed number.

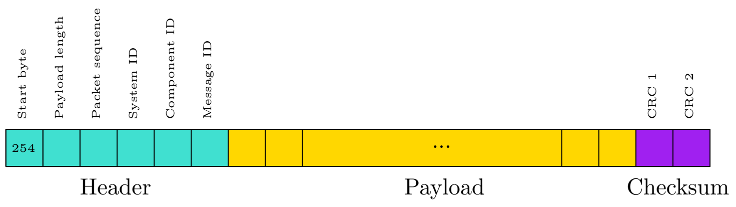


Figure 65: Schematic overview of a MAVLink message in byte form.

D.1.2 Header

The header of a message is of fixed length (6 bytes). Each byte location has a specific function as is shown in Table 4.

Byte	Description	Number of bytes
0	Start byte	Denotes the start of a message, this number is always 254 (or $0xFE$ hexadecimal)
1	Payload length	The length of the payload in bytes
2	Packet sequence	A number that cycles periodically from 0 to 255 intended to recognize packet drop
3	System ID	The number of the sending system (on our system: 1 for the flight controller, 255 for the companion computer)
4	Component ID	The number of the sending component on the sending system
5	Message ID	The message-id number of the message as defined in the xml-file

Table 4: Definition of the header part of a MAVLink message.

D.1.3 Payload

The payload contains the data-fields of a specific message as was defined in Section D.1. Because all data-fields are of fixed size the payload has a fixed length for each message-id. Before the data-fields of a message are send, they are sorted by field-type-byte-size in descending order. That some fields are of vector form is irrelevant, thus only the type of the scalar/vector-field is of importance, not the length of the vector. For fields with the same field-type size the xml-file ordering is preserved. The following Matlab code can be used for ordering the data-fields.

```
%% Determine field ordering, the fields are sorted by bit size in decend order
MAVLINK_MESSAGE{msgId}.sortOrder =[]; % clear order sequence vector
v = 1; % reset index
for p = 1:4 % for each possible bit size [ 64 32 16 8 ]
   for q = 1:length(MAVLINK_MESSAGE{msgId}.field) % for each field in a message(id)
      if MAVLINK_MESSAGE{msgId}.field{q}.bitSize == 2^(7-p)
         MAVLINK_MESSAGE{msgId}.sortOrder(v) = q;
         v = v + 1;
      end
   end
end
```

For the ATTITUDE_TARGET example message as defined in Section D.1 we have thus the reordered sequence of the fields as is shown in Figure 66.

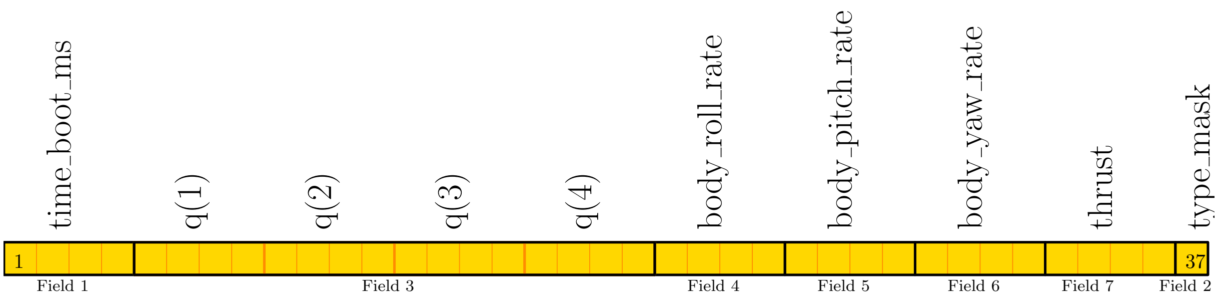


Figure 66: Schematic overview of a sorted payload for an ATTITUDE_TARGET message in byte form.

The payload can be constructed, in Matlab, by first typecasting the Simulink-doubles to MAVLink-types and thereafter to an array of uint8 bytes. This method works also for vectors, without any additional code. This can be for example observed in the code below for the MAVLINK_ATTITUDE_TARGET_q vector-field which is a vector-field with four entries.

```
MAVLINK_ATTITUDE_TARGET_msg_payload =
   [typecast(uint32(MAVLINK_ATTITUDE_TARGET_time_boot_ms),'uint8')   ,...
   typecast(single(MAVLINK_ATTITUDE_TARGET_q),'uint8')   ,...
   typecast(single(MAVLINK_ATTITUDE_TARGET_body_roll_rate),'uint8')   ,...
   typecast(single(MAVLINK_ATTITUDE_TARGET_body_pitch_rate),'uint8')   ,...
   typecast(single(MAVLINK_ATTITUDE_TARGET_body_yaw_rate),'uint8')   ,...
   typecast(single(MAVLINK_ATTITUDE_TARGET_thrust),'uint8')   ,...
   typecast(uint8(MAVLINK_ATTITUDE_TARGET_type_mask),'uint8')].';
```

D.1.4 Checksum

The two checksum bytes on the end of the message are used for verifying that the message is correctly received. However, in this case, the checksum also verifies that the semantics of the messages is for both sides the same. In other words both sender and receiver use the same xml-file. This verification is accomplished by calculating the checksum also over an additional seed-number. The short explanation of the seed-number is that it is a checksum over the content of the XML file, once calculated it is a one byte constant value. The detailed calculation of the seed-number is highlighted in Section D.1.5. The checksum in each message is calculated over the header without start bit, the payload and the temporary adjacent seed-number. In Figure 67 is visualized the array of bytes over which the checksum is calculated. The algorithm for calculating the checksum is described in [38]. The Matlab equivalent for calculating the checksum is given by

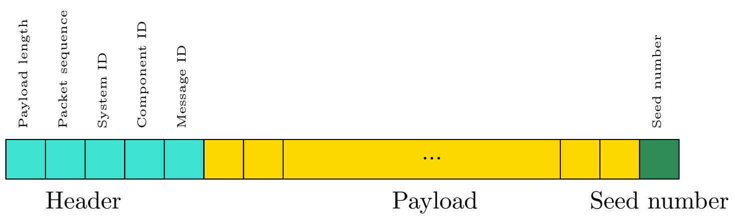


Figure 67: Schematic overview of the part of the message over which the checksum is calculated. Notice that the start byte is removed and the seed number is temporary added.

```
buf = uint8([header(2:end), payload, seednumber]);
function [crc1, crc2] = checksum(buf)
%% calculate check-sum according to the AEROSPACE STANDARD: AS5669A
   % initialize with all ones
   crcTmp = uint16(65535); % INIT_CRC = #ffff 16bit
   % Accumulate buf
   for i = 1:length(buf)
      tmp = bitxor( buf(i), uint8(bitand( crcTmp, hex2dec('00ff')))); % 8bit
      tmp1 = bitxor( tmp, bitsll(tmp, 4) ); % 8bit
      tmp2 = uint16(tmp1); % 16bit
      crcTmp = bitxor(bitxor(bitxor(bitsrl(crcTmp,8),bitsll(tmp2,8)),bitsll(tmp2,3)),
          bitsrl(tmp2,4)); % 16bit
   end
   crc1 = uint8(bitand(crcTmp, uint16(255))); % get low byte
   crc2 = uint8(bitsrl(crcTmp,8)); % get high byte
end
```

D.1.5 Seed number

The seed-number is by itself a checksum calculated over the text in the message-definition in the xml-file. This means that when the message-id, a field-type, a field-name, or a field-vector-length changes the seed number for the specific message-id also changes. Since the seed number is accumulated in the checksum calculation for each sent or received message, both sender and receiver require to have the same xml file for proper communication. The seed number is calculated with the same checksum algorithm as is used for runtime checksum-calculation as described in the previous section. The char (1 byte) array over which the seed-number is calculated is constructed as follows

```
buf = [char(MAVLINK_MESSAGE{msgId}.name),' '];
% add field type + space to the string
% add field name + space to the string
% if it is an array, then also add the array length
for q = 1:length(MAVLINK_MESSAGE{msgId}.field)
   fieldNr = MAVLINK_MESSAGE{msgId}.sortOrder(q);
   buf = [buf, char(MAVLINK_MESSAGE{msgId}.field{fieldNr}.typeAsChar), ' '];
   buf = [buf, char(MAVLINK_MESSAGE{msgId}.field{fieldNr}.name), ' '];
   if MAVLINK_MESSAGE{msgId}.field{fieldNr}.arrayLength > 0
      buf = [buf, char(MAVLINK_MESSAGE{msgId}.field{fieldNr}.arrayLength)];
   end
end
buf = uint8(buf)
```

In words this means, first add the message name plus a space (`32 = uint8('')`) then sort the fields as was described in Section D.1.3. Add for each sorted field the type and name separated by a space, if the field is an array add then the array-length as an uint8 to the buffer. When the buffer is complete, the seed-number can be calculated in the form

```
function [crc1, crc2] = checksum(buf)
seed = bitxor(crc1,crc2); % saved in struct
```

The buffer for our `ATTITUDE_TARGET` example is

```
'ATTITUDE_TARGET uint32_t time_boot_ms float q *float body_roll_rate float
    body_pitch_rate float body_yaw_rate float thrust uint8_t type_mask '
```

where * represents `*=uint8(4)`, for debug purpose the numerical variant looks like

```
[65 84 84 73 84 85 68 69 95 84 65 82 71 69 84 32 117 105 110 116 51 50 95 116 32 116
    105 109 101 95 98 111 111 116 95 109 115 32 102 108 111 97 116 32 113 32 4 102
    108 111 97 116 32 98 111 100 121 95 114 111 108 108 95 114 97 116 101 32 102 108
    111 97 116 32 98 111 100 121 95 112 105 116 99 104 95 114 97 116 101 32 102 108
    111 97 116 32 98 111 100 121 95 121 97 119 95 114 97 116 101 32 102 108 111 97
    116 32 116 104 114 117 115 116 32 117 105 110 116 56 95 116 32 116 121 112 101 95
     109 97 115 107 32].
```

The seed-number for this message-definition is 22.

D.2 MAVLink Receiving Messages

Receiving MAVLink messages is basically the inverse of constructing messages. In our case we scan, in the received byte-buffer, for the start byte (`254`), then we read the payload length and calculate the checksum over the potential message. If the checksum is correct then we are pretty confident that the start byte is the start of a message and not another random byte having the value `254`. Now that we are confident about that we have received a complete message, we can read the message-id and together with a switch statement we can transform the message payload back to the in the XML-file defined data-fields. An example of receiving the `ATTITUDE` message is shown in the listing below

```
switch messageID
   case 30
   % ATTITUDE
      % time_boot_ms: Timestamp (milliseconds since system boot)
      MAVLINK_ATTITUDE_time_boot_ms = typecast(serialBuffer(k+6:k+9), 'uint32');
      % roll: Roll angle (rad, -pi..+pi)
      MAVLINK_ATTITUDE_roll = typecast(serialBuffer(k+10:k+13), 'single');
      % pitch: Pitch angle (rad, -pi..+pi)
      MAVLINK_ATTITUDE_pitch = typecast(serialBuffer(k+14:k+17), 'single');
      % yaw: Yaw angle (rad, -pi..+pi)
      MAVLINK_ATTITUDE_yaw = typecast(serialBuffer(k+18:k+21), 'single');
      % rollspeed: Roll angular speed (rad/s)
      MAVLINK_ATTITUDE_rollspeed = typecast(serialBuffer(k+22:k+25), 'single');
      % pitchspeed: Pitch angular speed (rad/s)
      MAVLINK_ATTITUDE_pitchspeed = typecast(serialBuffer(k+26:k+29), 'single');
      % yawspeed: Yaw angular speed (rad/s)
      MAVLINK_ATTITUDE_yawspeed = typecast(serialBuffer(k+30:k+33), 'single');
   case ...
```

where `k` is the location of the message start-byte in the complete buffer.

D.3 MAVLink Command Message

Commands such as ARM or CHANGE MODE can be send using a specific command message. This is for example the the `COMMAND_LONG` message. The data-fields are in, this case, the target-system, target-component, command-number, confirmation and seven parameter-fields. This is thus a fairly universal message. Because the Arm/Disarm command is a popular command, we will use this command as an example for sending the `COMMAND_LONG` message. As specified by the enums in the XML-file, the command-number for the Arm/Disarm command is 400. For arming the target system we set the first parameter-field to the value 1 (0 for disarm). The target system is for this example 1 and the rest of the fields can left zero. The layout for this command is

```
MAVLINK_COMMAND_LONG_target_system = 1;
MAVLINK_COMMAND_LONG_target_component = 0;
MAVLINK_COMMAND_LONG_command = 400; % 400 =  Arm/Disarm command
MAVLINK_COMMAND_LONG_confirmation = 1;
MAVLINK_COMMAND_LONG_param1 = 1; % 1 = Arm, 0 = Disarm
MAVLINK_COMMAND_LONG_param2 = 0;
MAVLINK_COMMAND_LONG_param3 = 0;
MAVLINK_COMMAND_LONG_param4 = 0;
MAVLINK_COMMAND_LONG_param5 = 0;
MAVLINK_COMMAND_LONG_param6 = 0;
MAVLINK_COMMAND_LONG_param7 = 0;
```

After packing the message in byte-form it can be send to the target system, which reacts by arming the system.

D.4 MAVLink-Simulink Interface

Receiver Function The MAVLink receiver function is a piece of Matlab code generated by the MAVLink generator function. The function recovers from a byte buffer the variable-values of the selected MAVLink messages. The process is best explained by the following pseudo code

```
function [variables] = MAVLink_Receiver(newBuffer)
 Declare persistent variables
   if not exists
      variables_old = 0
      buffer = [0...]
      k = 0
   end
 Zero order hold function
   variables = variables_old

   % FIFO buffer
   n = length(newBuffer)
   buffer = [buffer(n:end) + newBuffer] % Remove old, add new
   k_start = k_start - n

   % Loop through buffer
   k = k_start
   while k <= length(buffer)
      if buffer(k) = 254 % Message Start Indicator
         tryToReadMessageHeader
         potentialMsgLength = buffer(k+msgLengthPos)
         if k+potentialMsgLength <= length(buffer)
            evaluateChecksum
            if Checksum = OK % Message verified
               msgID = buffer(k+IDPos)
               receivedMsgIDs = [receivedMsgIDs + msgID]
               msgPayload = buffer(k+PayloadArea)
               switch msgID % Unpack payload
                  case msgIDs_receive(1)
                     variables.msgID.var1 = typecast(msgPayload.var1)
                     variables.msgID.var2 = typecast(msgPayload.var2)
                     ...
                  case msgIDs_receive(2)
                     ...
                  otherwise
                     ignore message
               end
               k_start = k + msgLength
               k = k + msgLength
            end
         end
      end
      k = k + 1
   end

 Remember outputs for zero order hold function
   variables_old = variables
end
```

As can be derived from the listing above the output-variables are initialized with zero value. Thereafter the output-values are changed by the arriving messages. Between the messages the output variables are kept constant in the form of a zero-order hold function. Because messages can be divided over two readings of the serial buffer, i.e., the first part of the message can be already received while the rest of the message is received at the next sampling time. Therefore, internally, all fresh received data is first added to a FIFO buffer. In order to detect a message we search for the message start indicator and verify its checksum. Thereafter the payload is casted to the matching output-variable. Once the message is evaluated, the message plus earlier data is removed from the fifo buffer.

Sender Function In order to send data via the MAVLink protocol the MAVLink Sender function can be used. This function packs, constructs, and combines MAVLink messages, based on some configurable conditions. The different conditions for sending a MAVLink message can be, send autonomously, force a message to send, or disable sending. The condition can be configured for each message seperatly by a send-mode variable. The following rules hold for the send mode

Mode	Value	Action
Forced	1	Send message at each execution
Automatic	0	Send when one or more values of the variables in a message changed
Disabled	-1	Do not send the message

In most situations the Automatic mode can be used, however when twice the same message needs to be send, for example the ARM command, the other two modes can be used. The internal structure of the code is best explained by the following pseudo code

```
function [buffer] = MAVLink_Sender(sendModes, variables)
% Initialize persistent variables
if not exists
   payloads_old = [0 ....]
end

% Initialize variables
buffer = [0...]
k = 1

% for each user specified message
for msgIDs_send
payload.msgID = constructPayload(variables.msgID)
   % if buffer is not full (if buffer is full the messae will be send the next time)
      if k + messageLength <= length(buffer)
         % Send condition
         changed = notEqual(payload.msgID, payload_old.msgID)
         forced = isForced(sendModes.msgID)
         automatic = isAutomatic(sendModes.msgID)
         disabled = isDisabled(sendModes.msgID)
         if ( forced OR ( automatic AND changed )) AND NOT disabled
            % Construct the message
            header = creatHeader(msgID)
            checksumValue = calculateChecksum(header, payload.msgID, seedNumber.msgID
                )
            message = createMessage(header, payload.msgID, checksumValue)
            % Add message to buffer
            buffer = [buffer + message]
            k = k + messageLength
            % Old value to detect if variable has changed
            payloads_old.msgID = payloads.msgID
         end
      end
   end
end
```

In words the function constructs the payload by type casting the input variables. Then based on the send condition the message is constructed and added to the buffer.

Serial_read_write Block The serial_read_write block performs the communication between the Simulink model and the serial-port. To our best knowledge, all the current available Simulink serial-communication-blocks read and write only a fixed number of bytes from and to the serial port. The disadvantage of this approach is that for reading data from the serial port we have to wait until the input buffer is completely full before the data becomes available to the connected block. Since MAVLink messages are not per se send with a fixed sample rate, nor are of fixed length and do not use default start and stop indicators, it occurs that messages are significant delayed (in the order of seconds) when using a large buffer. Also in the other direction, for writing data to the serial-port, we want to send a message or set of messages direct. We do not want to wait until the buffer of fixed length is full, nor do we want to padding the buffer full with nonsense data to fake a full buffer.

In order to solve this and obtain a 'clean' real-time connection, we designed our own serial-port interface function in the form of a Simulink S-Function written in c-code. This block receives all the current data available on the serial-port and sends all the MAVLink messages at that time-step. The structure of the block is as follows. During the block initialization-step a serial port is opened. Then, at each time-step, all the available bytes at the serial-port are copyed to the first part of a fixed sized block-output-vector, indicated by Figure 34.b.3, and the number of received-bytes is given in another block-output-variable, which is indicated by Figure 34.b.4. A fixed length input-vector is chosen because it is much more simple to implement, compared to variable length, in Simulink due to memory allocation, etc. On the other side, for sending data, the first m bytes from a fixed length input-vector, indicated by Figure 34.b.1, are copied to the serial-port, where m is given by an input-variable as is indicated by 34.b.2. There is a block designed for the Linux Ubuntu OS and one for the MS Windows OS, which both follow approximately the same structure. Worth nothing is that the S-function is qualified for running Simulink in 'Normal mode' as well as in 'External mode'.

Pacer Block By default, Simulink executes the simulation-steps directly after each-other, in order to show the simulation results as fast as possible to the user. However, for real-time execution of the model we want the simulation time to be synchronized with the real-time system clock. In Windows we can use te Real-Time Desktop Simulink target to achieving this. Since this target does not allow us to use the required "windows.h" header/library in order to access the serial-port and there is no serial-port available with variable buffer length, this target can not be used. For Linux we have only the "pace" block, from the Aerospace-Blockset, available for real-time model execution. A disadvantage of this "pace" block is that it does not function when compiled in Simulink External mode. For this reason we designed our own pacer-function in the form of a Matlab S-Function in c-code, as is indicated by Figure 34.D. The pacer block is a wait function that puts the application into sleep mode until it is time for the next execution-iteration, such that the simulation-time matches the real-time system clock. A fixed sample-time is assumed. To guarantee that this function-block is executed after all other calculations during an iteration, the block priority to a high value, giving it a low priority. The used code inside the function is based on the "libtimer_posix.c" library which is used in the E-Boxes [39] at TU/e. To force the operating system to execute our simulation with highest priority, in Linux Ubuntu, the command *sched_setscheduler()* [67] is executed during the function initialization step. In MS Windows, *SetPriorityClass()* [68] is used.

After setting the application priority we use the *clock_gettime()* function in Linux and the *QueryPerformanceCounter()* function in MS Windows to obtain the real-time system time in nano-seconds and micro-seconds respectively. In every execution-iteration of the pacer block, the system-time is stored once, as the final command. This time is thus equal to the time in which the next execution iteration started. After all the other calculations are finished this function is called. The only thing this function has to do, is to put the application in sleep mode. According to [67] the application can only specify a minimum sleep time, the operating system decides when to return back to the application. In this approach, we use multiple short sleep intervals until the sample-time has past by, and it is time to start with the next execution iteration. In Linux Ubuntu the *usleep()* command is used whereas in MS Windows *Sleep()* is used. The process is explained by the following pseudo code

```
sampleTime = 0.01
deltaWaitTime = 0.00001

wait = TRUE;
while (wait){
   T = getSystemTime(); /* current time */
   if ( T < (Tprev + sampleTime )){
      sleep(deltaWaitTime); /* sleep mode */
   } else {
      wait = FALSE; /* quit loop */
   }
}
Tprev = getSystemTime(); /* store time */
```

Worth noting is that the function automatically takes over the discrete sample-time from the Simulink properties. Furthermore, the selected target for Simulink Coder is in our case *grt.tlc*.

E Equations, Control, and Complementary filter

E.1 Non-linear equations

Consider the rigid body model in vector notation

$$\begin{aligned} \dot{p}_A &= v && (147)\\ m\dot{v}_A &= -R\,(\lambda)\,T\,e_3^B + m\,g\,e_3^A && (148)\\ \dot{\lambda} &= Q\,(\lambda)\,\omega && (149)\\ J\dot{\omega} &= \tau + \omega \times J\omega && (150)\end{aligned}$$

where $p_A = [p_{A,x} \quad p_{A,y} \quad p_{A,z}]^T$, $v_A = [v_{A,x} \quad v_{A,y} \quad v_{A,z}]^T$, $\lambda = [\phi \quad \theta \quad \psi]^T$, and $\omega = [\omega_x \quad \omega_y \quad \omega_z]^T$ represents the system-state position, velocity, Euler angles, and the body angular velocities. More specific, ϕ, θ, and ψ represents the roll, pitch, and yaw angle respectively. The rotation matrix $R(\lambda)$ is defined as

$$R(\lambda) = R_z(\psi)R_y(\theta)R_x(\phi) \qquad (151)$$

$$R(\lambda) = \begin{bmatrix} \cos\theta\cos\psi & \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\theta\sin\phi\sin\psi & cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \qquad (152)$$

The time derivative of the Euler angels $\dot{\lambda}$ is related via matrix $Q(\lambda)$ to the angular body velocity's $\omega = [\omega_x \quad \omega_y \quad \omega_z]^T$. The derivation of matrix $Q(\lambda)$ is for example discussed in [69] and is given by

$$Q(\lambda) = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix}. \qquad (153)$$

Furthermore, g represents the gravitational acceleration constant, m, the quadrotor's mass, and J the inertia matrix of the form

$$J = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \qquad (154)$$

where I_{ii} is the angular moment of inertia over the i axis. In the sequel of this work, the gyroscopic effects $\omega \times J\omega$ are neglected since the angular body velocities are assumed to be relative low.
The control inputs are the total thrust T, in e_3^B direction, and the torques $\tau_B = [\tau_x \quad \tau_y \quad \tau_z]^T$.
The full written out equations are

$$\begin{aligned} \dot{x}_A &= u_A && (155)\\ \dot{y}_A &= v_A && (156)\\ \dot{z}_A &= w_A && (157)\\ \dot{\phi}_A &= \omega_x + \omega_y \sin\phi\tan\theta + \omega_z\cos\phi\tan\theta && (158)\\ \dot{\theta}_A &= \omega_y\cos\phi - \omega_z\sin\phi && (159)\\ \dot{\psi}_A &= \omega_y\sin\phi\sec\theta + \omega_z\cos\phi\sec\theta && (160)\\ \dot{u}_A &= -\frac{1}{m}\left(\sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta\right)T && (161)\\ \dot{v}_A &= -\frac{1}{m}\left(\cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi\right)T && (162)\\ \dot{w}_A &= -\frac{1}{m}\left(\cos\theta\cos\phi\right)T + g && (163)\\ \dot{\omega}_{xx} &= \frac{1}{I_x}\tau_x && (164)\\ \dot{\omega}_{yy} &= \frac{1}{I_y}\tau_y && (165)\\ \dot{\omega}_{zz} &= \frac{1}{I_z}\tau_z. && (166)\end{aligned}$$

E.2 Mixing matrix

The mixing-matrix Γ can be written as

$$\hat{u} = \Gamma\,\hat{\omega}^2 \tag{167}$$

$$\begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} c & c & c & c \\ d_r\,c_1 & -d_r\,c_2 & -d_r\,c_3 & d_r\,c_4 \\ d_r\,c_1 & d_r\,c_2 & -d_r\,c_3 & -d_r\,c_4 \\ -d_1 & d_2 & -d_3 & d_4 \end{bmatrix} \begin{bmatrix} w_1^2 \\ w_2^2 \\ w_3^2 \\ w_4^2 \end{bmatrix} \tag{168}$$

where d_r represents the off-center distance of the rotor with respect to the center-of-gravity, c and d are the thrust and torque constants respectively, $\hat{\omega}$ represents the vector with angular-velocity of each rotor, and $\hat{u} = [T, \tau_x, \tau_y, \tau_z]^T$ represents thrust and torque vector.

E.3 Feedback Linearization

The decoupling matrix can be expressed as

$$A(x) = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \tag{169}$$

where

$$a_{11} = (2\hat{T}(\sin(\phi)\sin(\psi) - \cos(\phi)\cos(\psi)\sin(\theta)))/m \tag{170}$$
$$a_{21} = (2\hat{T}(\cos(\psi)\sin(\phi) + \cos(\phi)\sin(\theta)\sin(\psi)))/m \tag{171}$$
$$a_{31} = -(2\hat{T}\cos(\theta)\cos(\phi))/m \tag{172}$$
$$a_{41} = 0 \tag{173}$$
$$a_{12} = (\hat{T}^2(\cos(\phi)\sin(\psi) + \cos(\psi)\sin(\theta)\sin(\phi)))/(I_x m) \tag{174}$$
$$a_{22} = (\hat{T}^2(\cos(\phi)\cos(\psi) - \sin(\theta)\sin(\phi)\sin(\psi)))/(I_x m) \tag{175}$$
$$a_{32} = (\hat{T}^2\cos(\theta)\sin(\phi))/(I_x m) \tag{176}$$
$$a_{42} = 0 \tag{177}$$
$$a_{13} = (\hat{T}^2(\cos(\psi)\sin(\phi)^2 - \cos(\theta)^2\cos(\phi)^2\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\phi)\sin(\psi)))/(I_y m\cos(\theta)) \tag{178}$$
$$a_{23} = (\hat{T}^2(\cos(\theta)^2\cos(\phi)^2\sin(\psi) - \sin(\phi)^2\sin(\psi) + \cos(\phi)\cos(\psi)\sin(\theta)\sin(\phi)))/(I_y m\cos(\theta)) \tag{179}$$
$$a_{33} = (\hat{T}^2\cos(\phi)^2\sin(\theta))/(I_y m) \tag{180}$$
$$a_{43} = \sin(\phi)/(I_y\cos(\theta)) \tag{181}$$
$$\begin{aligned} a_{14} = {} & (\hat{T}^2(\cos(\psi) - \cos(\theta)^2\cos(\psi) - \cos(\phi)^2\cos(\psi) + 2\cos(\phi)^2\sin(\theta)\sin(\psi) \\ & + \cos(\theta)^2\cos(\phi)^2\cos(\psi) + 2\cos(\phi)\cos(\psi)\sin(\phi) + \cos(\phi)\sin(\theta)\sin(\phi)\sin(\psi)))/(I_z m\cos(\theta)) \end{aligned} \tag{182}$$
$$\begin{aligned} a_{24} = {} & (\hat{T}^2(\cos(\theta)^2\sin(\psi) - \sin(\psi) + \cos(\phi)^2\sin(\psi) - 2\cos(\phi)\sin(\phi)\sin(\psi) \\ & + 2\cos(\phi)^2\cos(\psi)\sin(\theta) - \cos(\theta)^2\cos(\phi)^2\sin(\psi) + \cos(\phi)\cos(\psi)\sin(\theta)\sin(\phi)))/(I_z m\cos(\theta)) \end{aligned} \tag{183}$$
$$a_{34} = (\hat{T}^2\sin(\theta)\sin(\phi)^2)/(I_z m) \tag{184}$$
$$a_{44} = \cos(\phi)/(I_z\cos(\theta)) \tag{185}$$

E.4 Complementary filter

E.4.1 Attitude Estimation

Critical for proper control is the correct estimation of the attitude of the vehicle. Here we define the attitude as the roll ϕ, and pitch θ angles only. In general, the attitude is estimated on the base of measurement data from the accelerometer and the gyroscope. Assuming that the acceleration sensor is measuring the gravity vector the attitude $\lambda_a = [\phi_a \quad \theta_a]^T$ can be estimated by

$$\phi_a = \tan^{-1}\left(\frac{a_y}{a_z}\right), \tag{186}$$

$$\theta_a = -\sin^{-1}\left(\frac{a_x}{\|a\|}\right). \tag{187}$$

where $a = [a_x \quad a_y \quad a_z]^T$ the sensor reading. In addition, the gyroscope signal can be used to estimate the attitude $\lambda_g = [\phi_g \quad \theta_g]^T$ by integration in time of

$$\dot{\lambda}_g = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} Q(\lambda)\,\omega \tag{188}$$

Sensor Fusion However, in practice, the acceleration sensors measures besides the low frequent gravity vector also the more high-frequent body-accelerations and disturbance. On the other-hand, the gyroscope besides the relative high-frequent body angular-rotation velocities, a low-frequent bias. The bias causes drift when integrated. To reduce the effect of these disturbance, the following sensor fusion is rule is used

$$\dot{\hat{\lambda}} = \dot{\lambda}_g + c\left(\lambda_a - \hat{\lambda}\right) \tag{189}$$

where $\hat{\lambda} = [\hat{\phi} \quad \hat{\theta}]^T$ is the estimated attitude and c is a positive real scalar gain. The equivalent control scheme of (189) is shown in Figure 68. Laplace transform of (189) yields

$$\hat{\Lambda}(s) = \frac{s}{s+c}\Lambda_g(s) + \frac{1}{\frac{1}{c}s+1}\Lambda_a(s) \tag{190}$$

which is equivalent to a low-pass filter for the acceleration measurements and a high-pass filter for the gyroscope measurements, as is shown in Figure (69). This type of sensor fusion is known as a complementary filter.

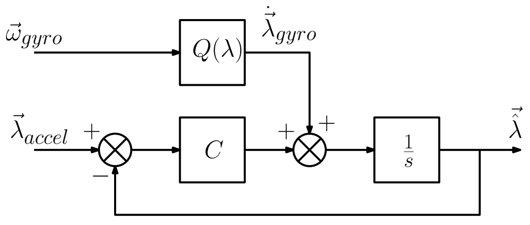


Figure 68: Cotrol sheme of sensor fusion.

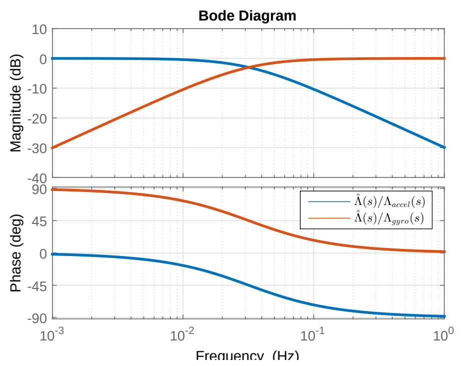


Figure 69: Complimentary filter for sensor fusion. Low-pass filter for acceleration sensor measurements and high-pass filter for integrated gyroscope measurements.

E.5 Linearization of the equations of motion.

In this case we assume near hovering behavior, therefore the non-linear equations of motion can be linearized around the point $\phi = \theta = \omega_x = \omega_y = \omega_z = 0$ and $T = T_0 = m\,g$. However, it can not be assumed that the heading ψ of the multi-rotor is approximately zero, therefore the current $\psi_c = \psi$ heading is evaluated. The following linearized dynamical system is obtained

$$\begin{aligned}
\dot{p}_{A,x} &= v_{A,x} & \dot{v}_{A,x} &= -\frac{T_0}{m}(\sin\psi_c\,\phi + \cos\psi_c\,\theta) & \dot{\phi} &= \omega_x & \dot{\omega}_x &= \frac{1}{I_x}\tau_x \\
\dot{p}_{A,y} &= v_{A,y} & \dot{v}_{A,y} &= \frac{T_0}{m}(-\cos\psi_c\,\phi + \sin\psi_c\,\theta) & \dot{\theta} &= \omega_y & \dot{\omega}_y &= \frac{1}{I_y}\tau_y \\
\dot{p}_{A,z} &= v_{A,z} & \dot{v}_{A,z} &= -\frac{1}{m}T & \dot{\psi} &= \omega_z & \dot{\omega}_z &= \frac{1}{I_z}\tau_z
\end{aligned}$$

where we assume

$$J = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}. \tag{191}$$

By introducing

$$u_{\phi,\theta,T} = \hat{R}_{\psi_c}(\psi_c) \begin{bmatrix} \phi \\ \theta \\ T \end{bmatrix} \tag{192}$$

where

$$\hat{R}_{\psi_c}(\psi_c) = \begin{bmatrix} -g & 0 & 0 \\ 0 & -g & 0 \\ 0 & 0 & -\frac{1}{m} \end{bmatrix} \begin{bmatrix} \sin\psi_c & \cos\psi_c & 0 \\ -\cos\psi_c & \sin\psi_c & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{193}$$

rotates the horizontal plane according to the current heading ψ_c, the system equations can be written as a cascaded double integrator system

$$\begin{bmatrix} \dot{\lambda} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} 0_{3\times3} & I_{3\times3} \\ 0_{3\times3} & 0_{3\times3} \end{bmatrix} \begin{bmatrix} \lambda \\ \omega \end{bmatrix} + \begin{bmatrix} 0_{3\times3} \\ J^{-1} \end{bmatrix} \tau \tag{194}$$

and

$$\begin{bmatrix} \dot{p_A} \\ \dot{v_A} \end{bmatrix} = \begin{bmatrix} 0_{3\times3} & I_{3\times3} \\ 0_{3\times3} & 0_{3\times3} \end{bmatrix} \begin{bmatrix} p_A \\ v_A \end{bmatrix} + \begin{bmatrix} 0_{3\times3} \\ I_{3\times3} \end{bmatrix} u_{\theta,\phi,T} \tag{195}$$

where $0_{3\times3}$ is a 3 by 3 zero matrix and $I_{3\times3}$ is a 3 by 3 identity matrix.

E.6 Low-level Linear Control

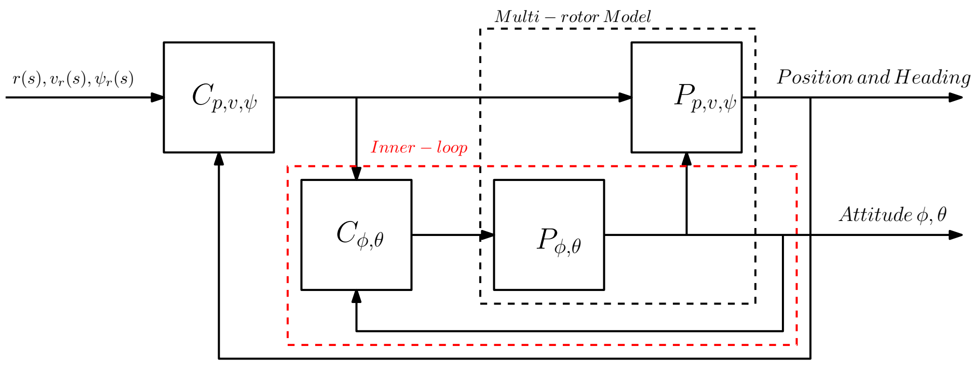


Figure 70: Abstract schematic representation of the control scheme. The roll ϕ and pitch θ stabilzation of the multirotor is concidered as the inner-loop. The inner-loop follows a reference specified by the outer-loop.

The low-level, attitude control, of a multi-rotor is already discussed in many papers. Popular proposed and validated control algorithms are for example linear control [70] using PID control and LQR, non-linear control in the form of feedback linearization [71], and a non-linear back-stepping approach [26]. Often, as in the Pixhawk, cascaded control is implemented consisting of an inner-loop, and an outer-loop, as sketched in Figure 70. The inner-loop controller is concerned with the low-level control, for stabilizing and controlling the attitude, and the outer-loop is concerned with the higher-level control, for controlling the position and velocity of the multi-rotor.

In this section the linear quadratic regulator (LQR) approach, as described in [72] and [73], is used to obtain a stabilizing controller for the roll and pitch angles of the multi-rotor. Consider the linear-system

$$\dot{x}_{\phi,\theta} = Ax + Bu_{\phi,\theta} \tag{196}$$
$$y_{\phi,\theta} = Cx + Du_{\phi,\theta} \tag{197}$$

as can be derived from (194), where

$$x_{\phi,\theta} \begin{bmatrix} \phi \\ \theta \\ \omega_x \\ \omega_y \end{bmatrix} \quad u_{\phi,\theta} \begin{bmatrix} \tau_x \\ \tau_y \end{bmatrix} \quad A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{1}{I_x} & 0 \\ 0 & \frac{1}{I_y} \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} D \quad = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{198}$$

The control objective is to minimize the quadratic function

$$J_{LQR} = \int_0^\infty (x - Lx_r)^T Q (x - Lx_r) + u^T R u \tag{199}$$

where Q and R are diagonal weighting matrices and $x_r = [\phi_r, \theta_r]^T$ is the reference attitude, the values are given by

$$Q = \begin{bmatrix} q_1 & 0 & 0 & 0 \\ 0 & q_2 & 0 & 0 \\ 0 & 0 & q_3 & 0 \\ 0 & 0 & 0 & q_4 \end{bmatrix} \qquad R = \begin{bmatrix} r_1 & 0 \\ 0 & r_2 \end{bmatrix} \qquad L = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{200}$$

where q_i and r_j are the individual weighting values. Full state feedback is assumed since all states are measured The proposed control law is

$$u_{\phi,\theta} = -K\Big(x_{\phi,\theta} - Lx_r\Big) + \begin{bmatrix} I_x & 0 \\ 0 & I_y \end{bmatrix} \ddot{x}_r \tag{201}$$

where $\ddot{x}_r$ is a feed-forward term, which is for this double integrator system the second time derivative of the reference x_r, and K is a matrix with feedback gains of the form

$$K = R^{-1} B^T P \tag{202}$$

where P is a constant, because of the infinite time case, symmetric matrix which is the solution of the Algebraic Riccati Equation

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{203}$$

for this relative simple system, the solution can be found analytically and is

$$K = \begin{bmatrix} \frac{1}{r_1}\sqrt{r_1 q_1} & 0 & \frac{1}{r_1}\sqrt{2r_1\sqrt{r_1 q_1} + r_1 q_3} & 0 \\ 0 & \frac{1}{r_2}\sqrt{r_2 q_2} & 0 & \frac{1}{r_2}\sqrt{2r_2\sqrt{r_2 q_2} + r_1 q_4} \end{bmatrix} \tag{204}$$

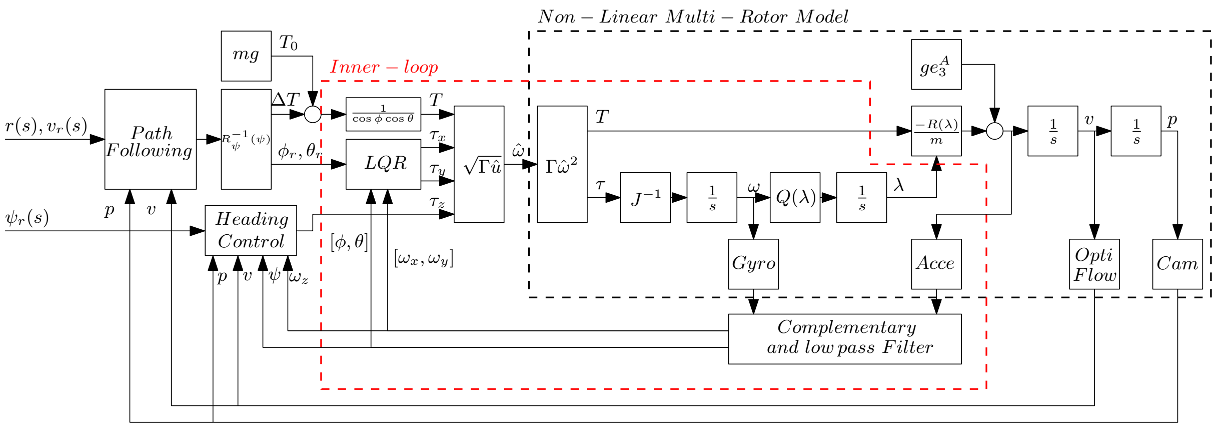
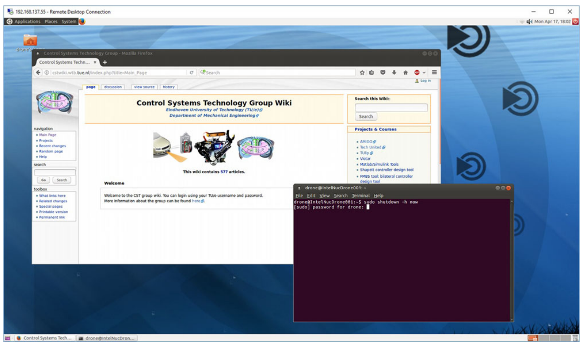


Figure 71: Schematic overview of an optional complete control scheme

F Remote connection to the on-board Intel Nuc PC

Since an Intel Nuc is on-board of the drone, the drone can be treated is a regular PC. There are multiple ways to interface with the drone. Obviously a monitor, keyboard, and mouse can be used to work with the drone. However, this is not always possible, for example during flight. A more applicable method is to work remote on the drone using the human-interface of your local machine (laptop). Basically, when working remote, the user-interface commands are send and received from the external-machine over a network connection. Modern operating systems, by default, include often one or more applications for remote access with an external-machines. Well known graphical or non-graphical communication protocols are for example rdp, ssh, and vnc. Where the last two are probably the best known protocols among Linux developers. However, the remote desktop protocol (RDP), originally introduced by Microsoft Windows, has some advantages compared to vnc. The RDP is a semantic protocol which means that for example text displayed on the screen is send as text and not as image. This reduces the network traffic significantly and results in a smooth interface without noticeable delays. From our experience we do not observe much difference between working remote and direct on the machine, as long no videos are played. Another advantage, since we are aiming for cross-platform communication between drone and local-machine, is that there already, by default, a rdp-client available is in all modern Windows versions. A disadvantage is that RDP is not available for Ubuntu, luckily there is xRDP [36] that works seamlessly together with Windows RDP. On Ubuntu the xRDP can be installed according to the steps listed in [37]]. Instead of the Ubuntu Unity environment the Mate environment is currently used to let xRDP work properly.

In order to establish a connection between the drone and the client-PC a network is required. The Intel Nuc in the drone is equipped with an Ethernet and a Wifi connection which both can be used for communication (the Blue-tooth port is available but not further discussed in this work). During flight, only wifi is used. Since users might want to work on different locations with the drone, for example at the lab, RoboCup soccer-field, outside, or at home, the network should be portable. Sometimes, it is practical to have Internet access on the drone, for example when downloading software, therefore the network configuration as shown in Figure 73 is used. Where the laptop is or is not connected to Internet, the Internet connection of the Laptop is shared over Ethernet with the Access-point. The access-point provides the wireless connection with the drone over wifi. In case of a wired connection the access-point is replaced by a direct Ethernet cable from drone to laptop. Modern network cards do not require a special crossover cable. Establishing a RDP connection is explained for Windows 10 in Appendix Section F.2, for Windows 7 in Appendix Section F.3, and for Ubuntu LT 16.04 in Appendix Section F.1, other operating systems are not tested. In Section F.4 is explained how establish a wired connection. Furthermore, in the digital appendix belonging to the multi-rotor a video explaining how to connect and use the MAVLink-Simulink interface is added. Finally in Section G an overview of the installed software on the multi-rotor is shown.



```
drone@IntelNucDrone001:~$ sudo shutdown -h now
[sudo] password for drone: 
```

Figure 72: Screen capture remote acces to the drone from windows local machine.

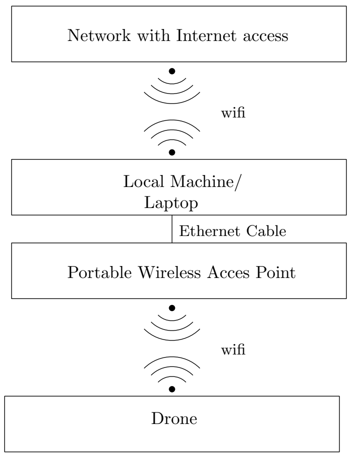


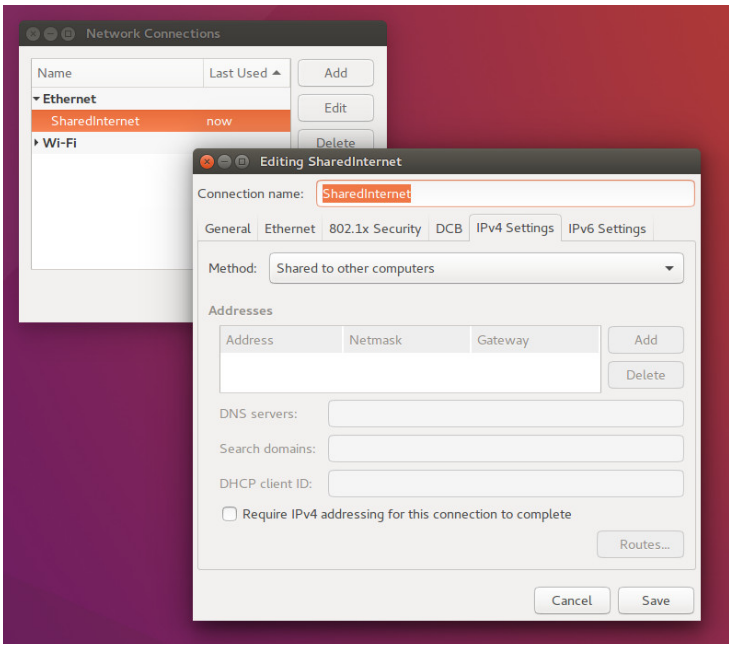
Figure 73: Example sketch of remote acces to the drone.

F.1 Ubuntu

F.1.1 Internet connection sharing (ICS)

Assuming that your laptop is connected via wifi to Internet, the Internet connection can be shared over the Ethernet/wireless port to the drone. The following steps should be taken:

1. Open the *Network Connections* dialog box, select the network you want to use for sharing, and click on *Edit*, a dialog box will pop up as is shown in the figure below.



Network Connections

Name	Last Used
Ethernet	
SharedInternet	now
Wi-Fi	

Editing SharedInternet

Connection name: SharedInternet

IPv4 Settings

Method: Shared to other computers

Addresses

Address	Netmask	Gateway

DNS servers:

Search domains:

DHCP client ID:

☐ Require IPv4 addressing for this connection to complete

2. On the *IPv4 Settings* tab select the *Method: Shared to other computers*

3. Eventually change the name of the connection and *Save* the settings.

4. Connect the access-point device physically to the Ethernet port of your laptop and start up the Intel Nuc on the drone.

5. The Intel Nuc on the drone will automatically connect to the access-point device as long as the access-point is configured to broadcast the SSID name: *DroneNetwork* with the password: *DroneNet*.

6. Alternatively the drone can directly connected to your laptop using a regular Ethernet cable. Modern Ethernet-cards do not require a special crossover Ethernet cable.

F.1.2 Remote Desktop

1. The first step is to find the ip-address of the drone. The drone has a dynamic ip-address since ICS has a fixed ip-range which is in Ubuntu (10.42.0.xxx) different from Windows (192.168.137.xxx). We want the drone, in this setup, to work with Ubuntu and Windows on the client-PC without installing additional software. Therefore, a dynamic ip-address is chosen. Open a terminal *ctrl+alt+t* and type

```
arp -n
```

. Although arp only shows the connections that are already known to the PC it seems to show all the ip-addresses and MAC-addresses of the connected devices on the shared connection.

2. The ip-address of the drone can be recognized by its "unique" MAC-address. The MAC-address of our drone is *78:92:9c:fe:52:09*. If the address does not show up directly, repeat the arp -n command several times. If also after reboot of the drone the mac-address does not show up you can plug the monitor, keyboard, and mouse into the drone and see what its current ip-address is with the command:

```
ifconfig
```

Alternatively you can install and use nmap to scan all ip-addresses on the network by entering the commands

```
sudo apt-get install nmap
sudo nmap -sP -n 10.42.0.0/24
```

respectively.

3. Once the ip-address is found, startup the remote desktop client *Remmina Remote Desktop Client* and create a new remote desktop file, with the following settings

   Name: **RDP Connection Intel Nuc Drone**

   Protocol: **RDP - Remote Desktop Protocol**

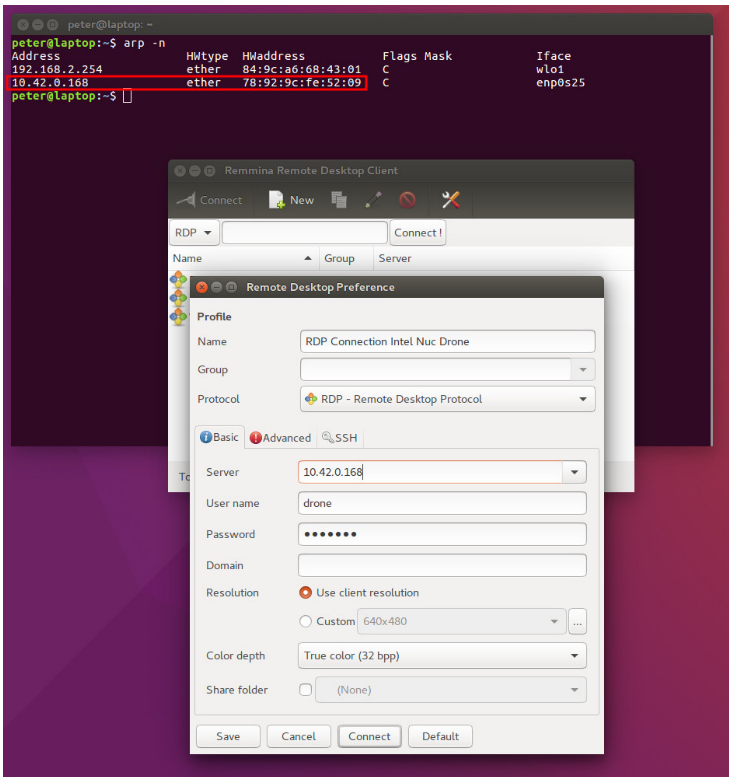
   Server: The ip-address of the Nuc for example (**10.42.0.xxx**)

   User name: **drone**

   Password: **i7drone**

   Domain: keep this field clear

   Colordepth: **High color (16 dpp)** for fast response (Matlab requires minimal 16 dpp), or **True color (32 dpp)** for true colors which is required to view the real colors from for example the camera. The image below shows an example.



```
peter@laptop:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.2.254            ether   84:9c:a6:68:43:01   C                     wlo1
10.42.0.168              ether   78:92:9c:fe:52:09   C                     enp0s25
peter@laptop:~$
```

Remote Desktop Preference

Profile

Name: RDP Connection Intel Nuc Drone
Group:
Protocol: RDP - Remote Desktop Protocol

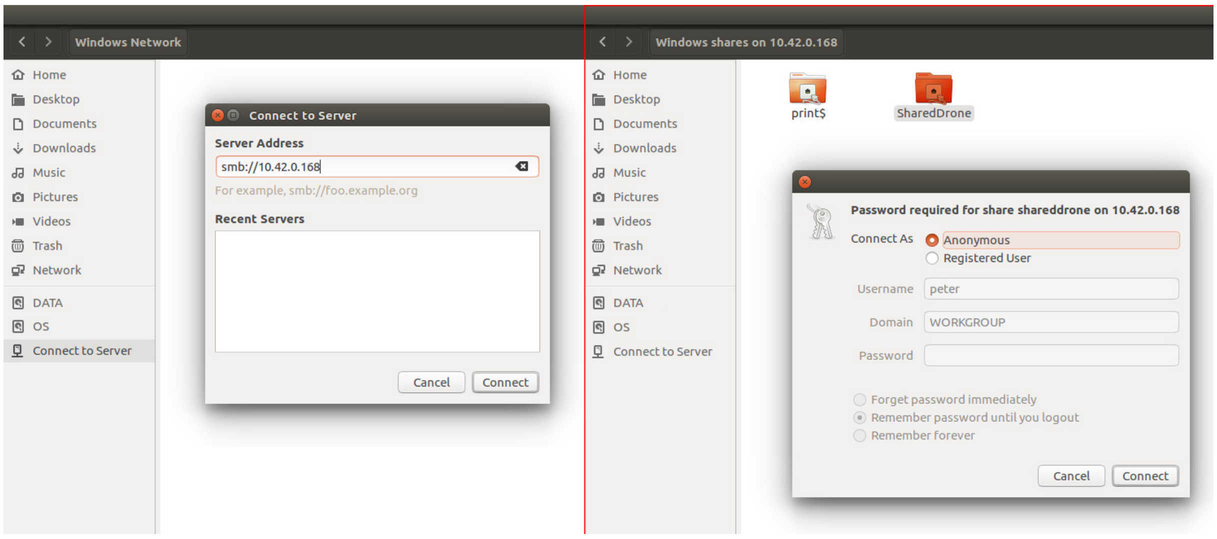
Server: 10.42.0.168
User name: drone
Password: •••••••
Domain:
Resolution: [✓ Use client resolution] Custom 640x480
Color depth: True color (32 bpp)
Share folder: (None)

4. Click Connect to log into the drone. If everything went well the graphical Mate environment on the drone is now visible. To make use of the shortcut *ctrl+alt+t*, to open a terminal in the Nuc, activate the *Grab all keyboard events* in the Remmina toolbar.

F.1.3 File sharing

1. On the drone Samba is installed which makes it possible to share files in a directory over the network with other devices. In the Ubuntu Unity file manager on your laptop click on *Connect to Server*, as is shown in the image below, and type in the ip-address *smb://10.42.0.xxx* which was found in one of the previous steps *smb://10.42.0.xxx*

2. Once connected, you can open the shared directory *SharedDrone*. If prompted connect as *Anonymous*.



Connect to Server

Server Address: smb://10.42.0.168

For example, smb://foo.example.org

Recent Servers

Password required for share shareddrone on 10.42.0.168

Connect As: [✓ Anonymous] Registered User

Username: peter

Domain: WORKGROUP

Password:

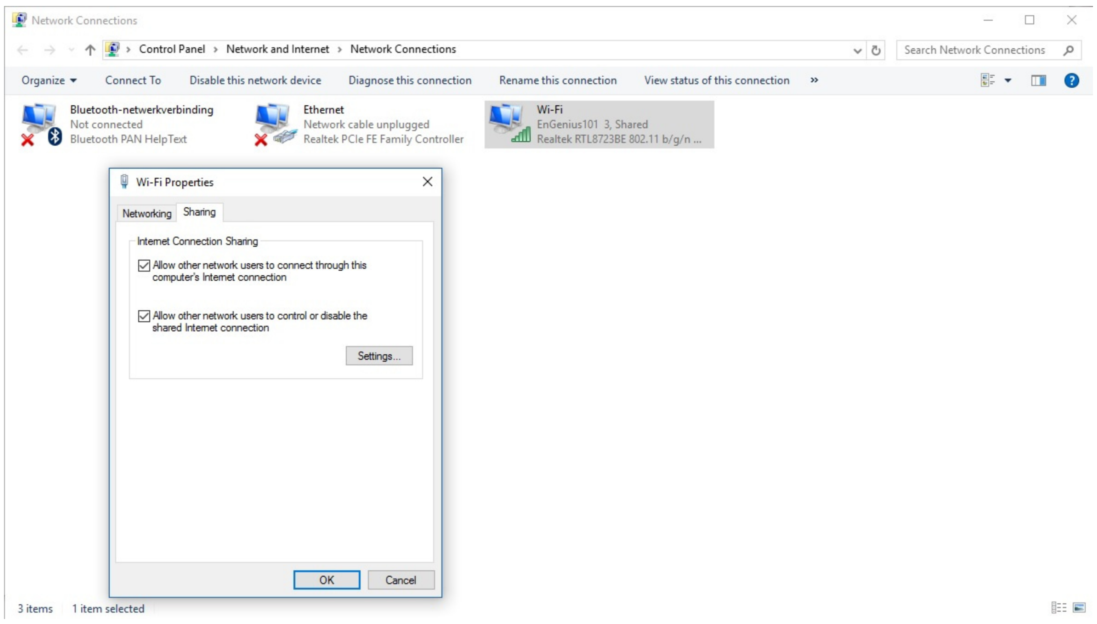
Forget password immediately [✓ Remember password until you logout] Remember forever

F.2 Windows 10

F.2.1 Internet connection sharing (ICS)

Assuming that your laptop is connected via wifi to Internet, the Internet connection can be shared over the Ethernet/wireless port to the drone. The following steps should be taken:

1. Open: *Control Panel* - *Network and Internet* - *Network Connections* as is shown in the image below. Oppose to Ubuntu, open the *properties* from the pull-down menu of the wireless *Wi-Fi* connection with Internet access



Bluetooth-netwerkverbinding
Not connected
Bluetooth PAN HelpText

Ethernet
Network cable unplugged
Realtek PCIe FE Family Controller

Wi-Fi
EnGenius101_3, Shared
Realtek RTL8723BE 802.11 b/g/n ...

Internet Connection Sharing
[✓] Allow other network users to connect through this computer's Internet connection
[✓] Allow other network users to control or disable the shared Internet connection

2. On the *Sharing* tab select *Allow other network users to connect through this computer's Internet connection* and click *OK* to confirm. This will change the IPv4 ip-address on the Ethernet port to 192.168.137.1.

3. Connect the access-point device physically to the Ethernet port of your laptop and start up the Intel Nuc on the drone.

4. The Intel Nuc on the drone will automatically connect to the access-point device as long as the access-point is configured to broadcast the SSID name: *DroneNetwork* with the password: *DroneNet*.

5. Alternatively the drone can directly connected to your laptop using a regular Ethernet cable. Modern Ethernet-cards do not require a special crossover Ethernet cable.

F.2.2 Remote Desktop

1. The first step is to find the ip-address of the drone. The drone has a dynamic ip-address since ICS has a fixed ip-range which is in Ubuntu (10.42.0.xxx) different from Windows (192.168.137.xxx). We want the drone, in this setup, to work with Ubuntu and Windows on the client-PC without installing additional software. Therefore, a dynamic ip-address is chosen. Open a command window *cmd* from the Windows start menu and type the command

```
arp -a
```

(note the use of -a compared to the -n used in Ubuntu). Although arp only shows the connections that are already known to the PC it seems to show all the ip-addresses and MAC-addresses of the connected devices on the shared connection.

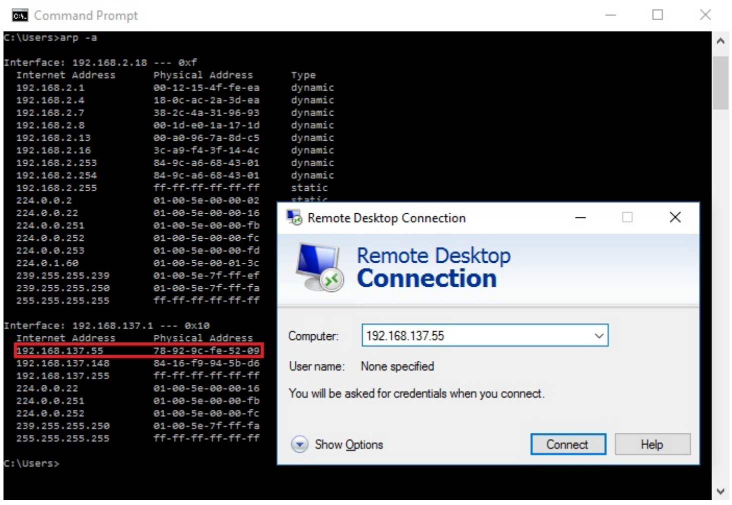
2. The ip-address of the drone can be recognized by its "unique" MAC-address. The MAC-address of our drone is *78:92:9c:fe:52:09*. If the address does not show up directly, repeat the arp -a command

several times. If also after reboot of the drone the mac-address does not show up you can plug the monitor, keyboard, and mouse into the drone and see what its current ip-address is with the command:

```
ipconfig
```

Alternatively you can install and use nmap for Windows to scan all ip-addresses on the network.

3. Once the ip-address is found, startup the remote desktop client *Remote Desktop Connection* from the Windows start menu. Specify the correct ip-address for example **192.168.137.xxx** as is shown in the image below.



```
C:\Users>arp -a

Interface: 192.168.2.18 --- 0xf
  Internet Address      Physical Address      Type
  192.168.2.1           00-12-15-4f-fe-ea     dynamic
  192.168.2.4           18-0c-ac-2a-3d-ea     dynamic
  192.168.2.7           38-2c-4a-31-96-93     dynamic
  192.168.2.8           00-1d-e0-1a-17-1d     dynamic
  192.168.2.13          00-a0-96-7a-8d-c5     dynamic
  192.168.2.16          3c-a9-f4-3f-14-4c     dynamic
  192.168.2.253         84-9c-a6-68-43-01     dynamic
  192.168.2.254         84-9c-a6-68-43-01     dynamic
  192.168.2.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.2             01-00-5e-00-00-02     static
  224.0.0.22            01-00-5e-00-00-16
  224.0.0.251           01-00-5e-00-00-fb
  224.0.0.252           01-00-5e-00-00-fc
  224.0.0.253           01-00-5e-00-00-fd
  224.0.1.60            01-00-5e-00-01-3c
  239.255.255.239       01-00-5e-7f-ff-ef
  239.255.255.250       01-00-5e-7f-ff-fa
  255.255.255.255       ff-ff-ff-ff-ff-ff

Interface: 192.168.137.1 --- 0x10
  Internet Address      Physical Address
  192.168.137.55        78-92-9c-fe-52-09
  192.168.137.148       84-16-f9-94-5b-d6
  192.168.137.255       ff-ff-ff-ff-ff-ff
  224.0.0.22            01-00-5e-00-00-16
  224.0.0.251           01-00-5e-00-00-fb
  224.0.0.252           01-00-5e-00-00-fc
  239.255.255.250       01-00-5e-7f-ff-fa
  255.255.255.255       ff-ff-ff-ff-ff-ff

C:\Users>
```

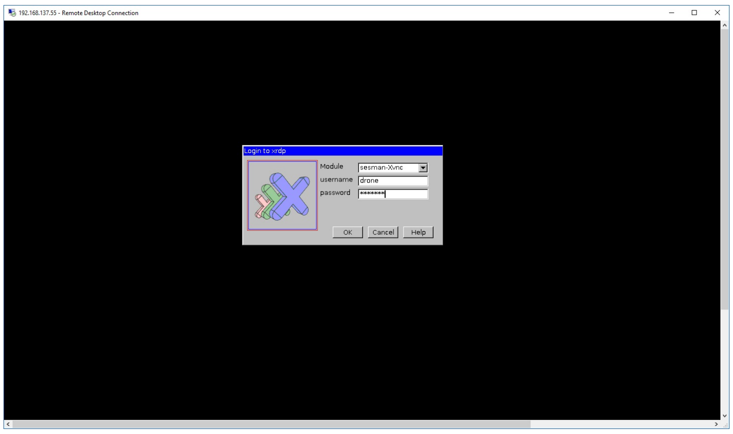
Computer: 192.168.137.55
User name: None specified
You will be asked for credentials when you connect.

4. When propted fill in the

   username: **drone**

   password: **i7drone**

   As shown in the image below.

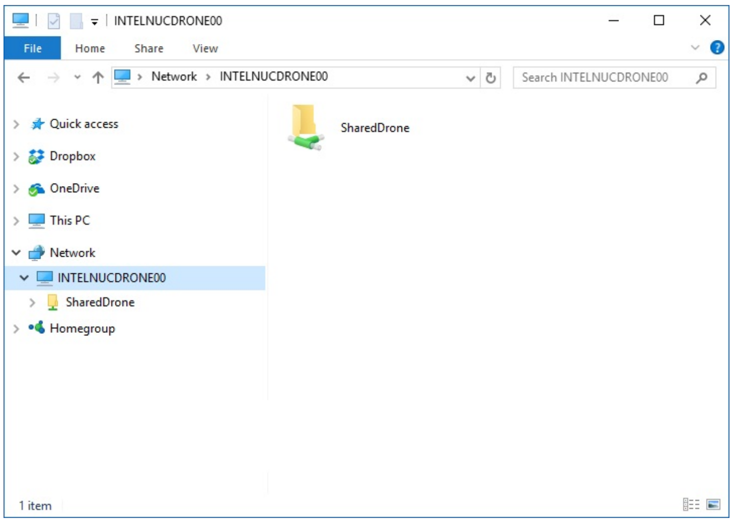


Module: sesman-Xvnc
username: drone
password: *******

F.2.3 File sharing

1. On the drone Samba is installed which makes it possible to share files in a directory over the network with other devices. Open in Windows *This Computer*

2. Since Samba works seamless together with Windows, the shared *SharedDrone* folder can already be found in the section *Network - INTELNUCDRONE00* as long as the drone is connected to the laptop. As is shown in the image below. (The Network/PC name of the drone is IntelNucDrone001)



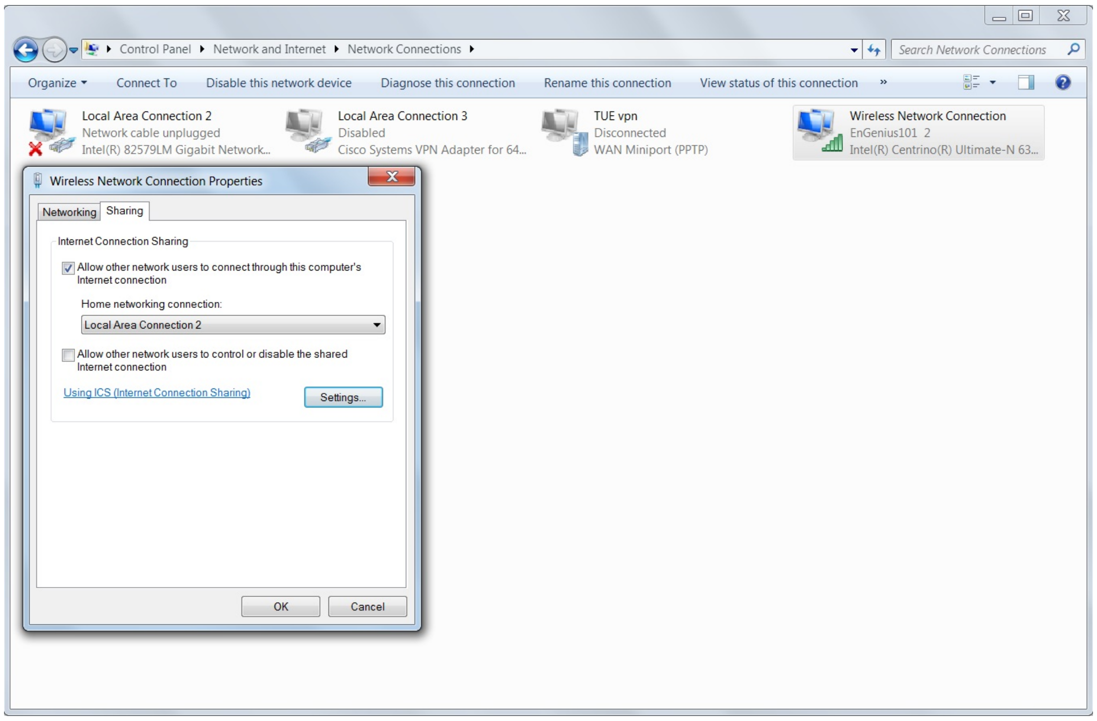
3. Alternatively you can view and download data by entering file://192.168.137.55/SharedDrone/ in your Internet browser

F.3 Windows 7

F.3.1 Internet connection sharing (ICS)

Assuming that your laptop is connected via wifi to Internet, the Internet connection can be shared over the Ethernet/wireless port to the drone. The following steps should be taken:

1. Open: *Control Panel* - *Network and Internet* - *Network Connections* as is shown in the image below. Oppose to Ubuntu, open the *properties* from the pull-down menu of the wireless *Wi-Fi* connection with Internet access


Control Panel ▸ Network and Internet ▸ Network Connections ▸
Search Network Connections
Organize ▾
Connect To
Disable this network device
Diagnose this connection
Rename this connection
View status of this connection
Wireless Network Connection Properties
Networking
Sharing
Using ICS (Internet Connection Sharing)
Settings...
OK
Cancel

Local Area Connection 2
Network cable unplugged
Intel(R) 82579LM Gigabit Network...

Local Area Connection 3
Disabled
Cisco Systems VPN Adapter for 64...

TUE vpn
Disconnected
WAN Miniport (PPTP)

Wireless Network Connection
EnGenius101 2
Intel(R) Centrino(R) Ultimate-N 63...

Internet Connection Sharing

[✓] Allow other network users to connect through this computer's Internet connection

Home networking connection:
Local Area Connection 2

[ ] Allow other network users to control or disable the shared Internet connection

2. On the *Sharing* tab select *Allow other network users to connect through this computer's Internet connection* and select your Ethernet port you want to share your Internet connection with. This will change the IPv4 ip-address on the Ethernet port to 192.168.137.1.

3. Connect the access-point device physically to the Ethernet port of your laptop and start up the Intel Nuc on the drone.

4. The Intel Nuc on the drone will automatically connect to the access-point device as long as the access-point is configured to broadcast the SSID name: *DroneNetwork* with the password: *DroneNet*.

5. Alternatively the drone can directly connected to your laptop using a regular Ethernet cable. Modern Ethernet-cards do not require a special crossover Ethernet cable.

F.3.2 Remote Desktop

1. The first step is to find the ip-address of the drone. The drone has a dynamic ip-address since ICS has a fixed ip-range which is in Ubuntu (10.42.0.xxx) different from Windows (192.168.137.xxx). We want the drone, in this setup, to work with Ubuntu and Windows on the client-PC without installing additional software. Therefore, a dynamic ip-address is chosen. Open a command window *cmd* from the Windows start menu and type the command

```
arp -a
```

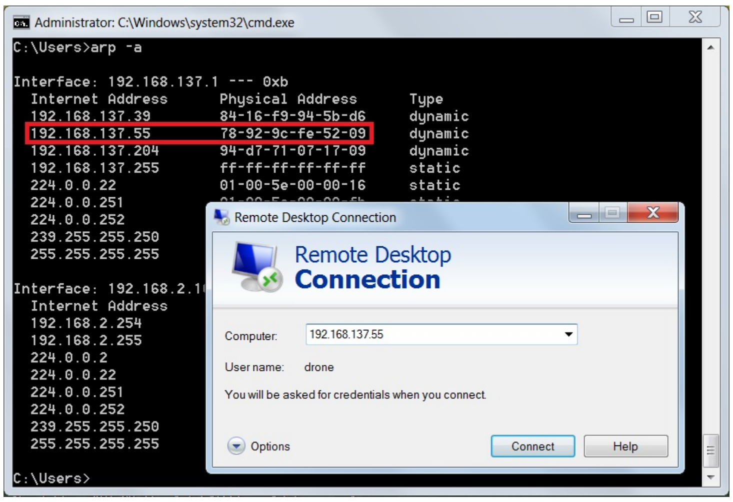
(note the use of -a compared to the -n used in Ubuntu). Although arp only shows the connections that are already known to the PC, it seems to show all the ip-addresses and MAC-addresses of the connected devices on the shared connection.

2. The ip-address of the drone can be recognized by its ”unique” MAC-address. The MAC-address of our drone is *78:92:9c:fe:52:09*. If the address does not show up directly, repeat the arp -a command several times. If also after reboot of the drone the mac-address does not show up, you can plug the monitor, keyboard, and mouse into the drone and see what its current ip-address is with the command:

   ```
   ipconfig
   ```

   Alternatively you can install and use nmap for Windows to scan all ip-addresses on the network.

3. Once the ip-address is found, startup the remote desktop client *Remote Desktop Connection* from the Windows start menu. Specify the correct ip-address for example **192.168.137.xxx** as is shown in the image below.



```
Administrator: C:\Windows\system32\cmd.exe
C:\Users>arp -a

Interface: 192.168.137.1 --- 0xb
  Internet Address      Physical Address      Type
  192.168.137.39        84-16-f9-94-5b-d6     dynamic
  192.168.137.55        78-92-9c-fe-52-09     dynamic
  192.168.137.204       94-d7-71-07-17-09     dynamic
  192.168.137.255       ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251
  224.0.0.252
  239.255.255.250
  255.255.255.255

Interface: 192.168.2.1
  Internet Address
  192.168.2.254
  192.168.2.255
  224.0.0.2
  224.0.0.22
  224.0.0.251
  224.0.0.252
  239.255.255.250
  255.255.255.255

C:\Users>
```

Remote Desktop Connection

Computer: 192.168.137.55

User name: drone

You will be asked for credentials when you connect.

4. When propted fill in the

   username: **drone**

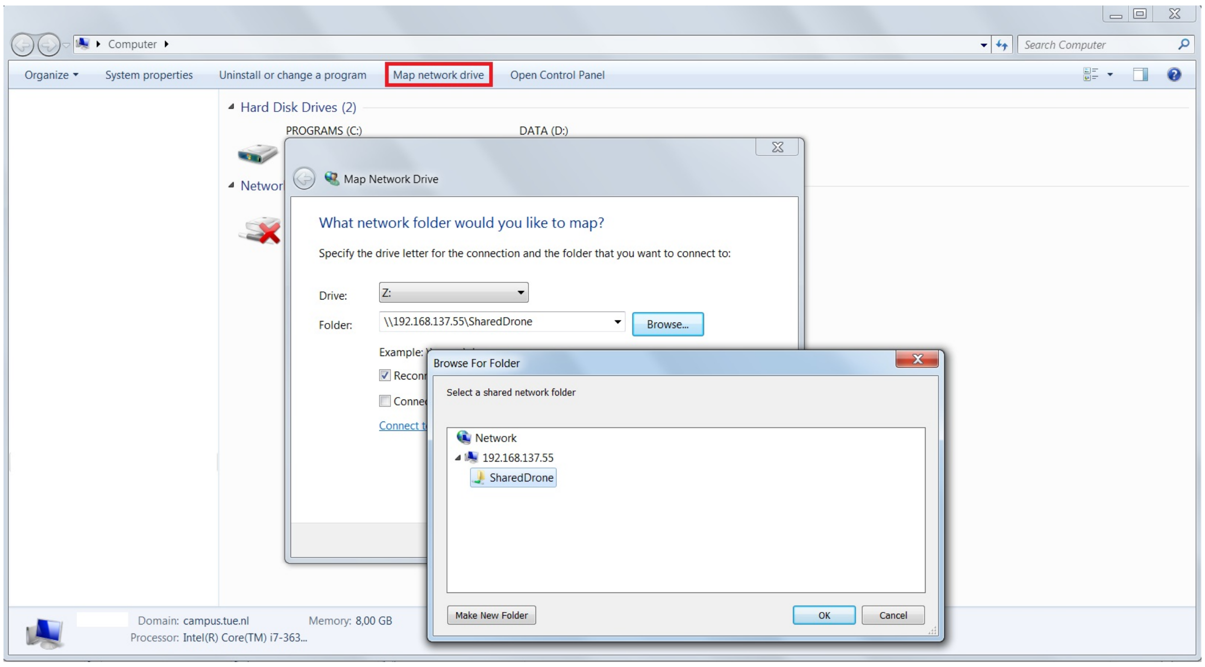
   password: **i7drone**

F.3.3 File sharing

1. On the drone Samba is installed which makes it possible to share files in a directory over the network with other devices. Open in Windows *This Computer*

2. Since Samba works seamless together with Windows, the shared *SharedDrone* folder can be mounted to your laptop, click in this computer on Map network drive.

3. Select a free drive and enter the ip-address in the form \\192.168.137.xxx and select browse to select the folder *SharedDrone* as is shown in the image below. (The Network/PC name of the drone is IntelNucDrone001)



What network folder would you like to map?

Specify the drive letter for the connection and the folder that you want to connect to:

Drive: Z:

Folder: \\192.168.137.55\SharedDrone

Select a shared network folder

Domain: campus.tue.nl Memory: 8,00 GB
Processor: Intel(R) Core(TM) i7-363...

4. The files can now be found as a mounted drive in *This Computer.*

5. Alternatively you can view and download data by entering file://192.168.137.55/SharedDrone/ in your Internet browser

F.4 Wired Ubuntu

- On your working laptop, activate **Shared to other computers** under IPv4 Settings for your ethernet connection (such that the Nuc is allowed to use Internet via your laptop).
- Connect the Nuc with your laptop using a regular Ethernet cable.
- If the Nuc is not already on, then start it using the on button on top of the drone (the blue light behind the button indicates that the Nuc is on).
- The default fixed ip-adress is: (If your not able to connect to the Nuc in the next step, then you can use)

  ```
  nmap -sP 10.42.0.0/24
  ```

  to search for the available ip-addresses on the (Ethernet) network. Often 10.42.0.1 is your own laptops addresses the other one is then the one of the Nuc drone. You can install nmap on your laptop using the command

  ```
  sudo apt-get install nmap
  ```

- Open Remmina Remote Desktop Client (by default installed on your Ubuntu system). Create a new remote desktop file, with the following settings

  Name: **Intel Nuc Drone Wired**

  Protocol: **RDP - Remote Desktop Protocol**

  Server: The ip-address of the Nuc for example (**10.42.0.217**)

  User name: **drone**

  Password: **i7drone**

  Domain: keep this field clear

  Colordepth:

  High color (16 dpp) for fast response (Matlab requires minimal 16 dpp?), or

  True color (32 dpp) for true colors which is required to view the real colors from the camera.

- Click Connect to log into the Nuc. If everything went well the graphical Mate environment is now visible. To make use of the shortcut Ctrl+t, in order to open a terminal in the Nuc, activate the *Grab all keyboard events* in the Remmina toolbar.
- In order to shutdown the Nuc 'safely' use

  ```
  sudo shutdown -h now
  ```

  Trying to shutdown the Nuc using the graphical interface will close the desktop session but not the Nuc.

G Installed software at the multi-rotor

G.1 General software

1. Install the desktop version of Ubuntu 16.04 LTS

    Computer name: IntelNucDrone001

    User: drone

    Password: i7drone

    Select do not ask for password at startup

    Set wireless internet settings to automatic connect to network with

    SSID: DroneNetwork

    Password: DroneNet

2. Install xRDP with Mate environment `http://www.xrdp.org/`
   according to the tutorial:
   How to install XRDP on Ubuntu 16.04 Easy Way, `http://c-nergy.be/blog`

3. Install Matlab with all its toolboxes

    user must be **root** (when there are license issues: remove the license fill and Matlab will ask for your license information the next time it is started)

    startup from terminal sudo matlab

    Tip: set shortcut keys as in windows

4. File sharing

    Create folder SharedDrone, via Samba share

5. guvcview for using camera instead of cheese, because cheese does not work in Mate

6. Qgroundcontrol (for installing flight controller in Pixhawk (not working in Mate environment)))

7. Install QTCreator for development of c++ code

8. Find more detailed instructions in the additional files in the digital appendix of this report

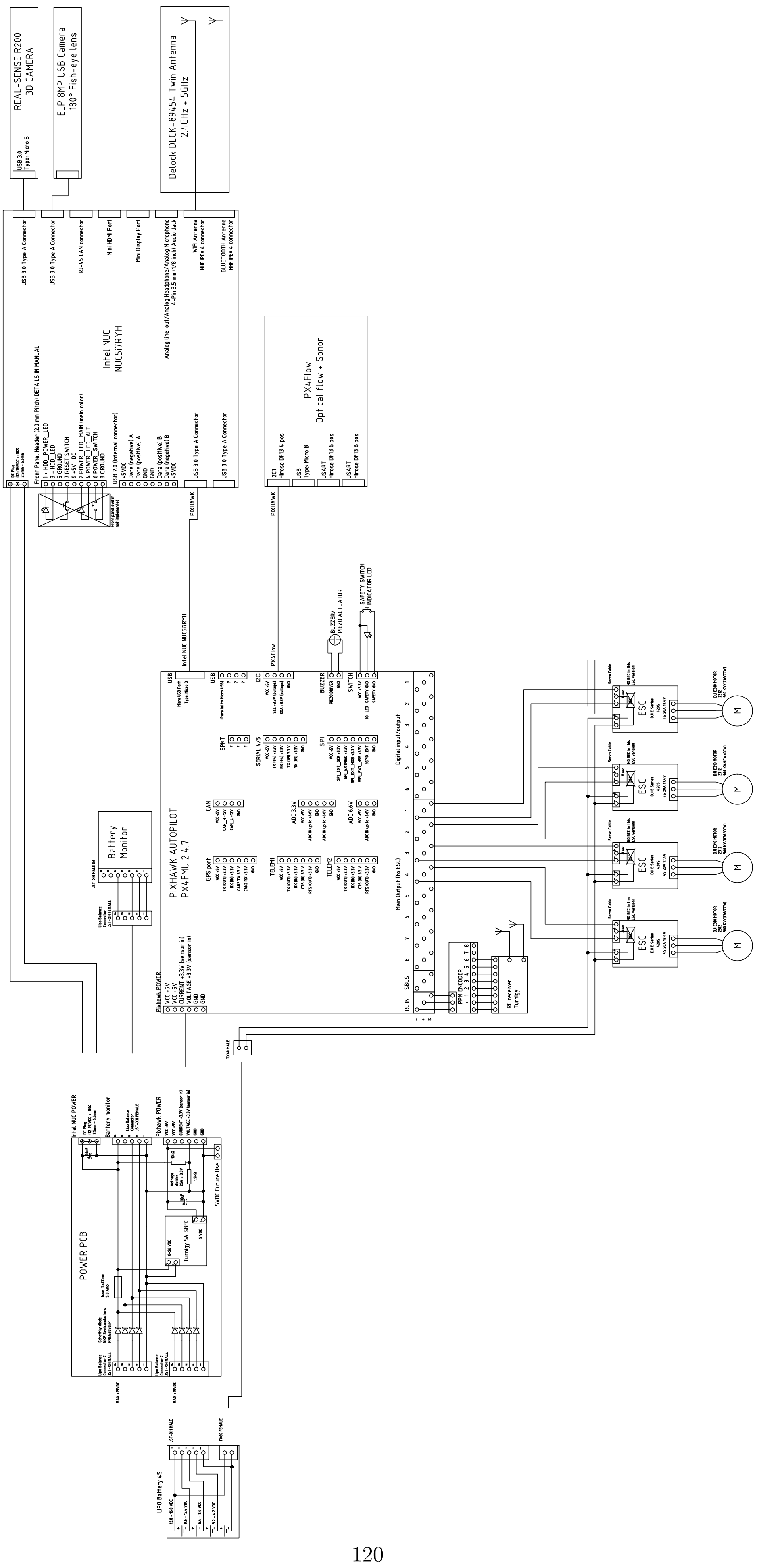
G.2 Install Opencv

Execute the following script in a terminal in order to install OpenCV (after # is comment):

```
# Follow the instructions on: http://docs.opencv.org/3.2.0/d7/d9f/
    tutorial_linux_install.html
sudo su # become root
cd / # change directory to root
sudo apt-get install build-essential # Install compiler if not already
sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-
    dev libswscale-dev # install required packages
sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-
    dev libtiff-dev libjasper-dev libdc1394-22-dev # install optional packages
cd opt/ # go to installation destination
git clone https://github.com/opencv/opencv.git # clone opencv from github
git clone https://github.com/opencv/opencv_contrib.git # clone opencv-contrib from
    github (required for Surf, Aruco markers, ect )
cd opencv # go to opencv folder
mkdir build # create a directory for building
cd build # go to the just created directory
cmake -D CMAKE_BUILD_TYPE=Release -D CMAKE_INSTALL_PREFIX=/usr/local -D
    OPENCV_EXTRA_MODULES_PATH=/opt/opencv_contrib/modules -D BUILD_EXAMPLES=ON .. #
    Make build files, see online manual for more options
make -j7 # build, runs 7 jobs in parallel
sudo make install # install libraries
```

H Electrical wiring diagram

Shown in the technical electrical wiring diagrams is how the different devices in the multi-rotor are connected. In Figure 74 is given a complete overview, in the PDF format the schematics are saved as vector format an can be enlarged. Alternatively, the appended original AutoDesk AutoCAD file can be used. In Figure 75, 76, 77, 79, 78, and 80 the enlarged detail views of the separate components is given. More details of the electrical circuit within the Pixhawk can be found in [`https://github.com/PX4/Hardware/blob/master/FMUv2/PX4FMUv2.4.5.pdf`], details about the PX4Flow module can be found in [`https://pixhawk.org/_media/modules/px4flow-schematic-v1.3.pdf`], [25] and [`https://pixhawk.org/_media/modules/px4flow-manual-v1.3.pdf`], details about the ESC's can be found in the user manual [`http://dl.djicdn.com/downloads/e310/en/E310_User_Manual_v1.0_en.pdf`], details about the radio receiver can be found in the user manual [`https://hobbyking.com/media/file/53173001X600367X17.pdf`] and more details about the Intel Nuc can be found in [`https://www.intel.com/content/dam/support/us/en/documents/boardsandkits/NUC5i7RYB_NUC5i7RYBR_TechProdSpec.pdf`].



REAL-SENSE R200 3D CAMERA
ELP 8MP USB Camera 180° Fish-eye lens
Delock DLCK-89454 Twin Antenna 2.4GHz + 5GHz
Intel NUC NUC5i7RYH
PX4Flow Optical flow + Sonor
PIXHAWK AUTOPILOT PX4FMU 2.4.7
Battery Monitor
ESC
M
BUZZER/ PIEZO ACTUATOR
SAFETY SWITCH INDICATOR LED
PPM ENCODER
RC receiver Turnigy
POWER PCB
Turnigy 5A SBEC
5VDC Future Use
LIPO Battery 4S

120

Figure 74: Complete overview electrical wiring diagram.

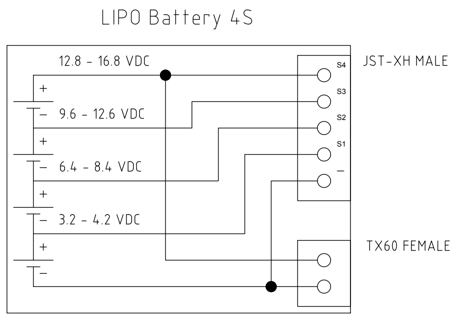


Figure 75: Detail electrical wiring diagram. Internal structure of battery package, shown are the in serie connected LiPo battery cells.

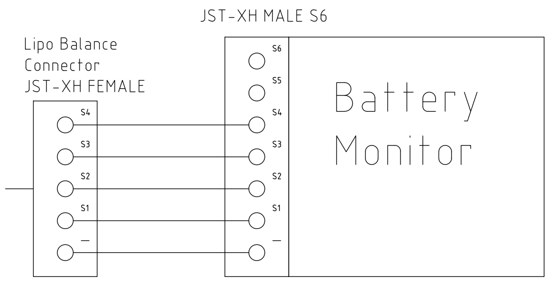


Figure 76: Detail overview electrical wiring diagram of the battery monitor module.

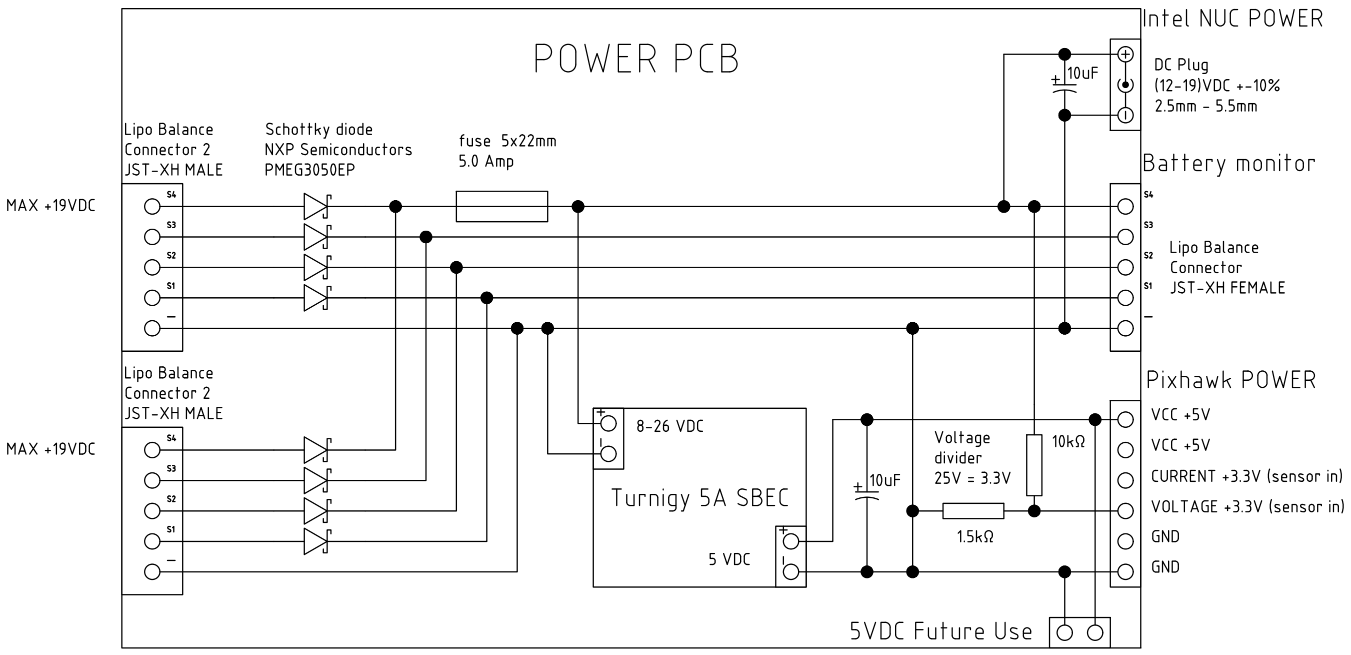

POWER PCB
Intel NUC POWER
DC Plug (12-19)VDC +-10% 2.5mm - 5.5mm
10uF
Battery monitor
Lipo Balance Connector JST-XH FEMALE
Lipo Balance Connector 2 JST-XH MALE
Schottky diode NXP Semiconductors PMEG3050EP
fuse 5x22mm 5.0 Amp
MAX +19VDC
MAX +19VDC
8-26 VDC
Turnigy 5A SBEC
5 VDC
Voltage divider 25V = 3.3V
10kΩ
1.5kΩ
Pixhawk POWER
VCC +5V
VCC +5V
CURRENT +3.3V (sensor in)
VOLTAGE +3.3V (sensor in)
GND
GND
5VDC Future Use

Figure 77: Detail overview electrical wiring diagram of the produced PCB for decoupling the inteference between two LiPo packages or a LiPo package and a wall power addapter.

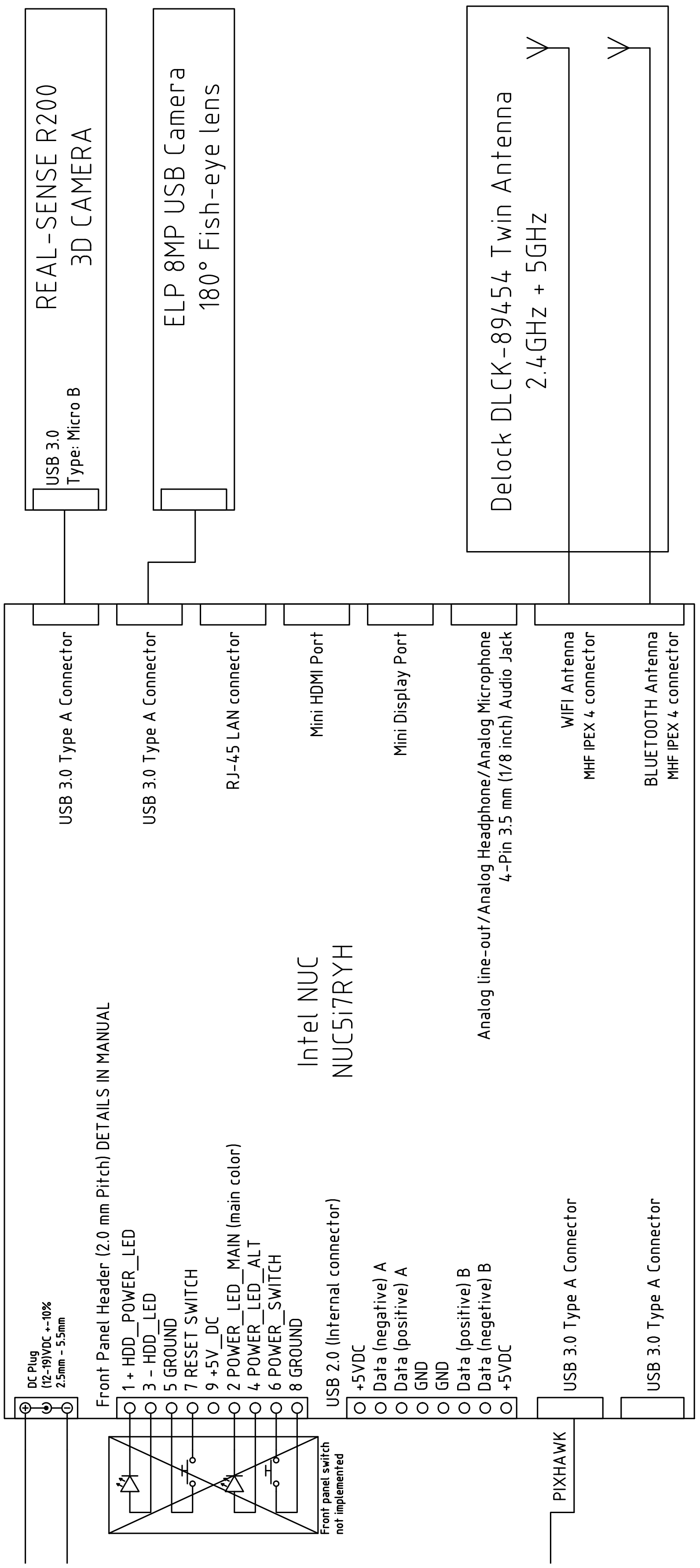


Figure 78: Detail overview electrical wiring diagram of the Intel Nuc, cameras, and wifi antenna. For possible future use the wiring af a wired remote on/off switch is indicated.

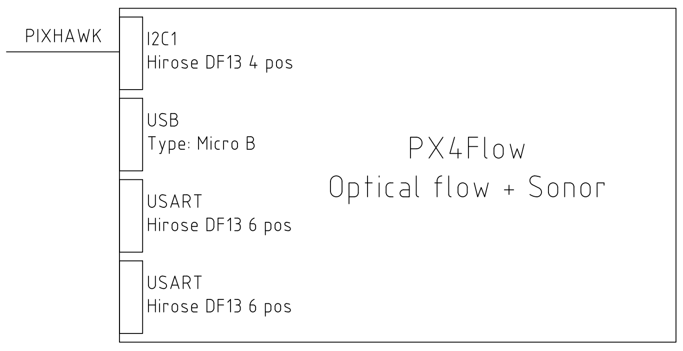


Figure 79: Detail overview electrical wiring diagram of the PX4Flow module with the Pixhawk.

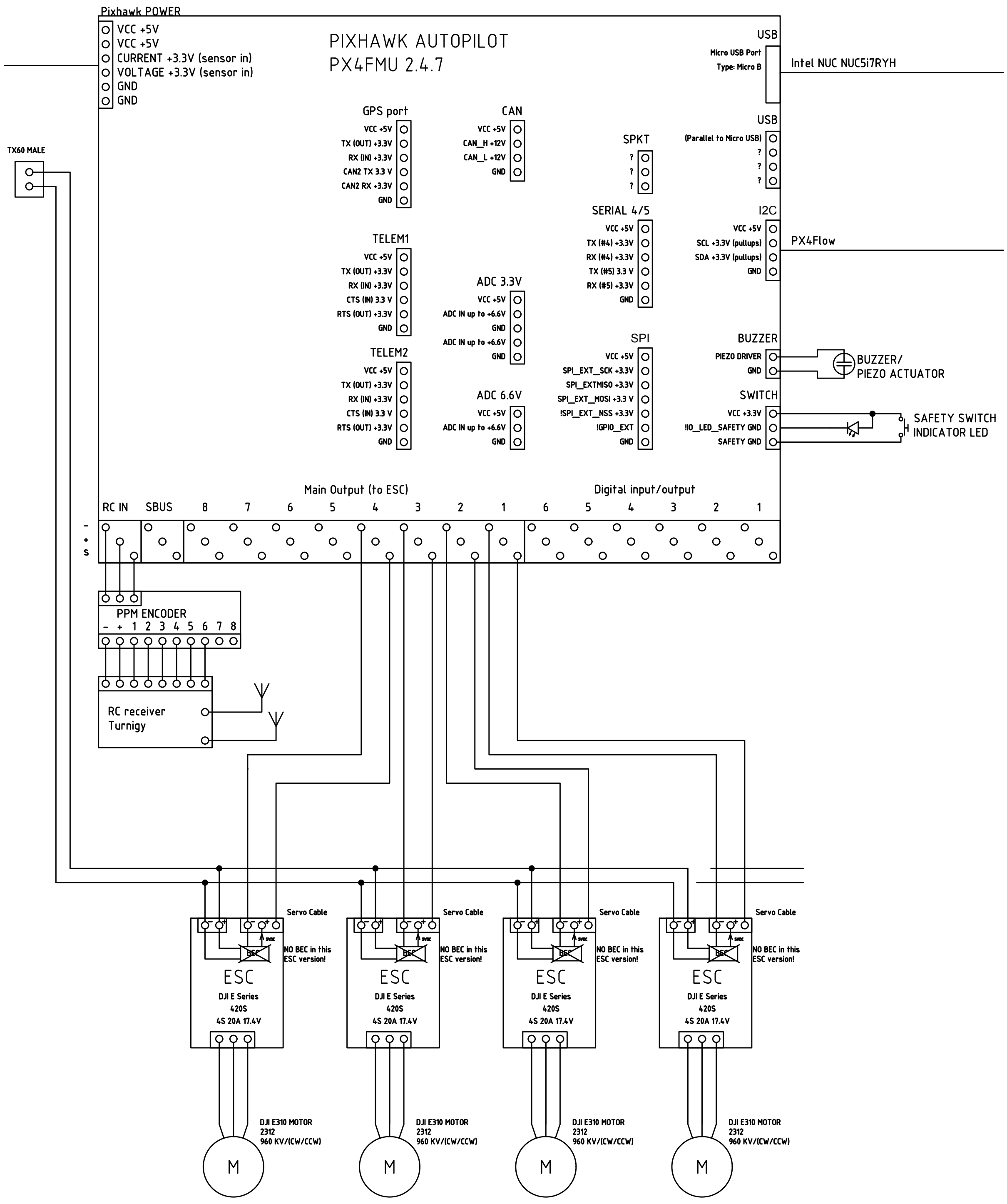


Figure 80: Detail overview electrical wiring diagram of the Pixhawk, electric speed controllers (ESC), and rotor motors.