# An MPSoC based Autonomous Unmanned Aerial Vehicle

van Esch, G.F.J.E.A.

*Award date:*
2020

Department of Electrical Engineering
Electronics Systems Research Group

# An MPSoC based Autonomous Unmanned Aerial Vehicle

*Master Thesis*

Gijs van Esch

Supervisors:
Kees Goossens (TU/e)
D. van den Heuvel (Topic Embedded Systems)

Final version

Eindhoven, Thursday 6th February, 2020

# Abstract

Drones are being used more often in various industries for various applications. Drones these days have the capabilities to e.g. perform image processing and object detection. Many of these industry applications require increasingly more processing power. The drone industry could benefit from a small embedded platform that has all the hardware required for autonomous flight with extra computational power available.

This is where the Xilinx Drone Platform (XDP) platform can be applied. The XDP platform contains all the necessary sensors needed to fly. It contains a powerful Multi-Processor System-on-Chip (MPSoC) with a Field Programmable Gate Array (FPGA), lockstep real time processor, Graphics Processing Unit (GPU) and, an Application Processing Unit (APU).

In this master thesis different sensors for obstacle avoidance on a drone are investigated and the design flow to develop a fully autonomous drone platform is discussed.

Different autopilots are discussed and PX4 is decided most suitable for this project. Different sensors are available that can be used for obstacle avoidance. These are discussed and it is decided which is best for this project. For the design flow it is proposed to implement and test each sensor individually in the form of sprints. Hereby it is important that each component is tested individually and in small steps, otherwise this could lead to wrong conclusions. The trade-offs for different mappings of processes on the platforms are being discussed and the impact of the decisions are tested. The implementation of the obstacle avoidance has been discussed and the connection and reading of sensor has been implemented. Whether the drone avoids obstacles has not been verified.

As a final product a drone with the ability to fly is realised using this platform. All the individual sensors on the board have been tested and verified whether they work correctly. This is the case for all sensors except for the magnetometer. However, it has also been proven that the correctness of the magnetometer can not be guaranteed because of the magnetic interference from current leaving the battery.

# Preface

I would like to thank my TU/e supervisor, Kees Goossens, for providing high-level and critical feedback and guiding me throughout the entire project.

Furthermore I would like to thank all my colleagues within Topic Embedded Systems, within the HeCTiC team which showed interest in my project and helped me with the problems that I encountered. Especially I would like to thank Dirk van den Heuvel for asking critical questions and helping me throughout the project.

Finally, I would like a special thanks to all my friends and family for supporting me during the project, especially my girlfriend Kim Roosen.

# Contents

# List of Figures

# List of Tables

# Abbreviations

**AI** Artificial Intelligence. 1

**APU** Application Processing Unit. 23, 24

**AXI** Advance eXtensible Interface. 31, 32

**BEC** Battery Elimination Circuit. 36

**BSD** Berkeley Software Distribution. 6

**BSP** board support package. 24

**CPU** Central Processing Unit. 3, 7, 31, 50

**cReComp** creator for Reconfigurable Component. 7

**ESCs** Electronic Speed Controllers. 1, 36

**FIFO** First-In First-Out. 7

**FPGA** Field Programmable Gate Array. 1, 3, 6, 7, 9, 13, 20, 23–25, 31, 32, 36, 56

**GNSS** Global Navigation Satellite System. 17

**GPL** GNU General Public License. 6

**GPS** Global Positioning System. 17, 20, 23, 37, 38, 41, 44, 51, 52

**GPU** Graphics Processing Unit. 23, 25

**IIO** Industrial Input/Output. 32, 33

**IMO** Independently Moving Objects. 4

**IMU** Inertial Measurement Unit. 4, 16, 17, 20, 23, 37, 41, 42

**IO** Input/Output. 24, 32

**IP** Intellectual Property. 31, 32

**Lidar** LIght Detection And Ranging. 4, 8, 9, 11, 13, 27, 56

**MAVLink** Micro Air Vehicle Link. 26

**MEMS** Microelectromechanical systems. 4

**MPSoC** Multi-Processor System-on-Chip. 1, 2, 6, 14, 17, 23, 26, 55

**PC** Personal Computer. 3

**PID** Proportional-Integral-Derivative. 2, 13, 20, 34, 35, 55, 56

**PL** Programmable Logic. 23

**PS** Processing System. 23

**RISC** Reduced Instruction Set Computer. 16

**ROS** Robot Operating System. 5–7, 10, 13, 17, 22, 24–27, 36, 50, 55

**RTOS** Real-Time Operating System. 17, 25, 50

**SLAM** Simultaneous Localization And Mapping. 4, 13

**SLUGS** Santa Cruz Low-cost UAV GNC System. 5

**UART** Universal asynchronous receiver-transmitter. 10, 11, 22

**UAV** Unmanned Aerial Vehicle. 1–5, 8, 13, 15

**UORB** micro Object Request Brokers. 19, 33

**VTOL** Vertical Take-Off and Landing. 5

**XDP** Xilinx Drone Platform. 15, 16, 23, 37

# Chapter 1

# Introduction

## 1.1 Context

These days drones are being used increasingly more often [27, 28, 29, 30], in different fields as for instance, precision agriculture [31], emergency services [32, 33], construction inspection [34] and many more applications.

According to *Gartner Research* [35] critical components within drones are, Artificial Intelligence (AI), Algorithms, Software, and processing. When comparing an FPGA to a micro controller it can be observed that there are several fundamental differences. For instance, in a video processing application, an FPGA can perform massive parallelism for faster computation compared to a micro controller that needs to process each pixel sequentially. This also holds for processing data from, radar or laser-based systems, which makes an FPGA more suitable for the timing critical components. Additionally the use of programming logic on an FPGA makes it possible to control the motors and read sensors at a higher rate compared to a micro controller.

Topic wants to develop an Unmanned Aerial Vehicle (UAV) using a special MPSoC. The benefit of using an MPSoC, which has an FPGA is that there is much more computing power available. This can be used for, video processing, adding more sensors and having higher communication bandwidth.

An important factor in autonomous drone flight is to avoid the obstacles, which can be achieved using sensors. The research of this master thesis will be about, what kind of sensors could be used for autonomous drone navigation. What are the advantages and disadvantage of each sensor and what would be a suitable solution using an MPSoC. Since an MPSoC is used, more data can be processed compared to when using a micro controller. An example of this is the implementation of stereo vision on an FPGA [36].

The control of the actuators will be researched. In industry a drone is controlled via Electronic Speed Controllers (ESCs), the FPGA however can run a dedicated control loop with special hardware to achieve higher loop rates. It will be interesting to research what design flows have to be followed in order to develop an autonomous drone platform on the MPSoC.

## 1.2 Problem statement

Topic developed a platform which has all the necessary sensors available in order to fly. However, it has not been proven whether this platform actually has the capabilities to control a drone. A design flow has to be developed in order to implement and test software capable of flying autonomously. This thesis aims to give an overview of the design flow and problems encountered when developing an autonomous UAV on an MPSoC.

## 1.3 Research question

The main research question can be formulated as:

- On an MPSoC, what are the design flows and trade-offs for different sensors for an autonomous navigating drone?

The following sub research questions can be derived from the main question:

- What are the trade-offs for different mappings of the process on the MPSoC and what would be most suitable for this project?
- What currently available sensors can be used on a drone to detect obstacles?
- What software is available that offers the necessary functionality?

## 1.4 Approach

This work explains the research, design, implementation, and testing of an autonomous drone platform running on a MPSoC. Research will include finding the correct software to make it work, see how obstacle avoidance is implemented in other projects and which sensors are most suitable to use for navigation on a drone. Design will include the workings of the autopilot software and how the processes will be mapped onto the MPSoC. Implementation will include writing the drivers for the motors and sensors, tuning the Proportional-Integral-Derivative (PID) control loop and getting obstacle avoidance to work. Experiments and results will discuss the different tests that have been performed in order to verify the correct working. This work gives a good overview of what is necessary to develop and test an autonomous drone on a new platform.

## 1.5 Outline

In Chapter 2 publications regarding implementations of UAV on an MPSoC, trade-offs for different autopilots and sensors and the design flow will be discussed. Chapter 3 presents the necessary theory used within this thesis. In Chapter 4 the working of the autopilot and mapping of the processes is discussed. Chapter 5 describes the implementation of the autopilot and the obstacle avoidance on the MPSoC. In chapter 6 the experiments and results will be presented. Chapter 7 provides the conclusions and possible future work related to the thesis.

# Chapter 2

# State of the art

Unmanned aerial vehicles are being used more often. In 2017 the UAV market was valued at 18.14 billion dollars and it is projected to reach 52.30 billion dollars by 2025 [37]. It can therefore be classified as a rapidly growing market in which increasingly more companies show their interest. Multiple studies have been performed within the field of drones and FPGAs.

## 2.1 Sensors on an FPGA

An important aspect in developing a UAV are the sensors. These should detect the obstacles and perform localization in order for the drone to reach its goal without colliding. Multiple studies have been performed on using an FPGA in collaboration with sensors in terms of processing power.

*L. Schäffer et al.* [38] talk about combining measurements of different sensors with a Kalman Filter [39] to achieve sensor fusion on an FPGA. The usage of an FPGA has been compared to a Central Processing Unit (CPU) and has proven that it is 358 times faster than the CPU while consuming less power than the CPU. However, the algorithm on the Personal Computer (PC) was programmed in MATLAB which is not the most efficient way to program and was running on an operating system.

The paper of *M.E. Conde et al.* [40] is similar to the paper of *L. Schäffer et al.* [38]. It combines the data of a sonar sensor and an infrared sensor to get a more reliable result. This implementation is then compared with a 32-bit embedded-processor, the NIOS-II. The paper concluded that an FPGA has an acceleration factor of 524 times that of an embedded processor.

## 2.2 Obstacle avoidance in drones

As written in Section 2.1, sensors are an important aspect for this project since the obstacle avoidance will rely on them. It is researched which sensors are used for obstacle avoidance in drones.

*M. Itani et al.* [41] talk about the usage of ultrasonic sensors on a drone. It is investigated and tested whether it would be feasible and some solutions are proposed in the form of an algorithm. The paper concluded that using ultrasonic sensors is a cheap and feasible solution. Unfortunately, nothing was developed and it is thus all theoretical.

The paper of *K. Li et al.* [42] talk about a low-cost indoor navigation based on Ultra Wide Band radio and a 3D laser. The data of the different sensors is fused with a Kalman filter in order to achieve better position accuracy. It is concluded that with the data received from the 3D laser, when hovering, the accuracy of the drone can be considerably increased. However, the problem with their low-cost laser scanner is that it takes 1 second to measure a certain distance. The drone has to hover in order to calculate distances.

*C. Cigla et al.* [43] propose a forward-looking stereo camera solution fused with an egocentric cylindrical camera. Using the new Gaussian Mixture Models-base disparity image fusion algorithm with an extension they are able to handle Independently Moving Objects (IMO). The on-board implementation provides a visual map at 10Hz. One remark that is immediately clear from this paper is that a lot of processing power is required to process the stereo images, even at the low resolution of 752x480.

The paper of *N. Gageik et al.* [44] uses sensor fusion of ultrasonic and infrared sensors for obstacle avoidance within drones. Ultrasonic sensors sometimes fail to reliably detect a human. Using the fusion a low-cost yet reliable obstacle avoidance system can be made. Additionally, there is a remark that a low-cost laser scanner could replace the infrared sensors to increase the reliability. A big downside is, that the use of infrared sensors will be much less for flying outside than inside. This could be better when using a laser.

*R. Li et al.* [45] proposes a Simultaneous Localization And Mapping (SLAM) [46] algorithm based on the LIght Detection And Ranging (Lidar) and Microelectromechanical systems (MEMS) Inertial Measurement Unit (IMU) via a Kalman filter. Using this there is an autonomous integrated navigation technology. During the experiments it is found that the proposed algorithm can effectively improve Lidar's environmental feature extraction accuracy and decrease calculation amount of filtering algorithm.

## 2.3 Autopilots available

An autopilot is software which is used to fly a drone, there are several autopilots available which are discussed in this section.

### 2.3.1 Dronecode (PX4)

Dronecode is a platform that contains everything needed for a complete UAV solution: flight controller hardware, autopilot software, ground control station and developer's API for enhanced and advanced use cases [47]. Dronecode is structured around an open-source license, open design, development, and contribution model.

The main project of Dronecode is PX4. PX4 is an open-source flight control software for drones and other unmanned vehicles [48, 49]. It provides a flexible set of tools for drone developers to share technologies to create tailored solutions for drone applications. PX4 originated from the PIXHAWK project [50] at ETH Zurich. It was specifically designed to be a research platform for computer vision-based flight control. Currently the PX4 project contains more than 300 global contributors and is used by some of the world's biggest companies such as Sony, NXP, and Microsoft.

PX4 main features are a modular and extensible architecture both in terms of hardware and software.
`https://www.dronecode.org/` `https://PX4.io/`

### 2.3.2 ArduPilot

ArduPilot was originally developed by hobbyists to control model aircraft and rovers. It started with an arduino program, hence the name ArduPilot. Currently it is one of the most advanced, full featured and reliable open-source autopilot software available [51]. It is installed in over 1 million vehicles world-wide and is therefore one of the most tested and proven autopilot software. Since ArduPilot is open-source it is rapidly evolving and always at the cutting edge of technology development. Many peripheral supplies create interfaces for the platform. The community is as big as that of Dronecode/PX4. The big difference between ArduPilot and PX4 is the license which will be explained in section 2.3.10.

`http://ardupilot.org/`

### 2.3.3 Paparazzi project

Paparazzi UAV is an open-source drone hardware and software project which was founded in 2003 [52, 53]. It was designed primarily focused on autonomous flight. Several core developers are affiliated with universities and research institutions such as the MAVLAB of TU-Delft. The community of the Paparazzi project is much smaller than that of PX4 and ArduPilot. This can be observed in the difference in forum activity and activity on GitHub. The license for the Paparazzi project is the same as for ArduPilot.

`http://wiki.paparazziuav.org/wiki/Main_Page`

### 2.3.4 SLUGS

Santa Cruz Low-cost UAV GNC System (SLUGS) from the University of California Santa Cruz [54]. The open-source software suite contains everything needed to let airborne systems fly, although, not fully autonomous [55]. The project started in 2009 but unfortunately it is no longer active with the last update from 30th of August 2009. The license used is the MIT license which is a similar license as the one used for Dronecode.

`https://slugsuav.soe.ucsc.edu/`

### 2.3.5 LibrePilot

LibrePilot is relatively young compared to ArduPilot. It is founded in July 2015 and it focuses on research and development of software and hardware to be used in a variety of applications including unmanned autonomous vehicles [56]. Overall it offers roughly the same features as ArduPilot or PX4 but has a much smaller community. It uses the same license as ArduPilot.

`https://www.librepilot.org/`

### 2.3.6 ROS (Robot Operating System)

The Robot Operating System is a set of software libraries and tools that help to build robot applications [57, 58]. Robot Operating System (ROS) consists of packages which can be used and communicate with each other using so-called topics and nodes. This way each hardware component can be a specific package and work individually from other sources. These can thus be combined in order to get a complete working device. ROS can also communicate with PX4, sensors could be read using ROS and then the data could be sent to PX4 to use this data. ROSflight could be an option for autopilot. It is an autopilot designed from the ground up for integration with ROS [59].

`https://www.ros.org` `https://rosflight.org/`

### 2.3.7 ROS2

ROS2 is the successor of the Robot Operating System, ROS is being used by increasingly more people and companies. One of the problems was that since the beginning the API of ROS was compatible with the oldest versions. This is nice in terms of stability, but this also means that choices made at the beginning of ROS, which could have been better are still implemented [60]. That is why it was decided to create a new version of ROS which will also enhance several new technologies. Not all the packages available for ROS are available for ROS2 yet.

`https://index.ros.org/doc/ros2/`

### 2.3.8 Uaventure

Uaventure has a fully autonomous Hybrid Vertical Take-Off and Landing (VTOL) Flight Control System [61]. It is compatible with most Pixhawk (hardware which is also used for PX4) compatible autopilots. This software is mainly being used for the autonomous delivery of medicine in hard-to-reach areas or areas where faster transportation is necessary. Unfortunately, this is not an open-source project and will thus not be suitable for this project.

`http://uaventure.com/`

### 2.3.9 Develop from scratch

The final option could be to develop the complete algorithm from scratch. Building it from scratch would likely result in better utilization of the FPGA compared to when using an open-source autopilot. Additionally, it will probably be easier to add additional features within the autopilot since the exact working of the software is known. For an autopilot if these features are not implemented they must be extensively researched. However, this option will likely cost significantly more time than using an open-source autopilot. It requires much more research on how to exactly develop the algorithm and likely not have the same amount of safety features as when using an autopilot. There are several research papers about the control of the quadcopter [62, 63, 64] which worked on the control of the motors to make a quadcopter fly. This does not include running it on an FPGA or flying autonomously. Development of the entire autopilot will most likely not be completed within the project time frame.

### 2.3.10 License

The GNU General Public License (GPL) license allows free online distribution after purchase of the code [65]. The Berkeley Software Distribution (BSD) license does not allow this [66]. With BSD the author has more control about what happens with the code. Therefore it is preferred to use the BSD license and thus either PX4 or ROS(2).

### 2.3.11 Conclusion

In Table 2.1 the important criteria for each autopilot can be observed. The table shows that Dronecode (PX4) or ArduPilot will be the best solution especially since these have already been implemented on a similar platform using an MPSoC. ROS(2) could also be a solution because of its large community and many available packages.

An interesting remark to make is that PX4 can be used in combination with ROS. ROS can handle the detection of obstacles and avoidance while PX4 takes care of flight of the retrieved path.
In the end PX4 will be used for this project because of the license and the knowledge that it works in various configurations.

Table 2.1: Different autopilot software compared

| | License | Features | Implemented on MPSoC | Size of community |
|---|---|---|---|---|
| Dronecode (PX4) | open-source BSD | All available | Yes | Big community |
| ArduPilot | open-source GPL | All available | Yes | Big community |
| Paparazzi project | open-source GPL | All available | No | Small community |
| SLUGS | open-source MIT | Basic flying available | No | No community |
| LibrePilot | open-source GPL | All available | No | Small community |
| ROS | open-source BSD | Flying and obstacle avoidance available | No | Big community |
| ROS2 | open-source BSD | Flying and obstacle avoidance available | No | Big community |
| Uaventure | Closed | All available | No | No community |
| Develop from scratch | - | Necessary features can be implemented | - | - |

## 2.4 ROS on FPGA

Since PX4 will be used in collaboration with ROS, it is therefore essential to research about whether ROS can be implemented on the FPGA.

*K. Yamashina et al.* [67] proposes creator for Reconfigurable Component (cReComp) which is an automated design tool to improve productivity of ROS-compliant FPGA component. Xillinux [68] is used to form the connection between the ARM processor and the FPGA logic. It can read and write to the FPGA as a device file that corresponds to First-In First-Out (FIFO) buffers. Developers can connect user logic to the FIFO buffers of Xillinux. Using a special file format to describe the connection between the ARM cores and Programmable Logic the connection is formed by cReComp. cReComp generates an HDL file and interface software in the form of a C++ file.

The paper of *Y. Nitta et al.* [69] describes the design and implementation of a small robot with a camera. The motors are controlled via a special embedded board which listens to the ROS network. The data from the camera is acquired using the FPGA and later pushed to ROS via the ARM cores. As concluded in this paper the FPGA is not fully utilized since only the image is acquired via FPGA. The image processing is still done on CPU.

Both these papers still require an external software interface that publishes the data retrieved from the FPGA onto the ROS network. It looks like it is not possible to directly publish from the FPGA as a software interface is always needed.

## 2.5 Trade-offs of techniques for sensors

In order to realize a fully autonomous drone it is important that it can locate and avoid obstacles. The flight speed of the drone will be limited to the range of a certain sensor. The faster the drone flies the earlier the obstacle has to be detected in order to prevent it from crashing. There are several techniques available to detect obstacles. Additionally, a distance sensor can be mounted looking down, this enables a more precise flight height.

### 2.5.1 Camera

Cameras with object recognition capabilities in cars have been used to identify other vehicles, lines, and objects. However, within cars these cameras always seem to work in collaboration with other sensors.

*D. Valencia et al.* [70] shows that it is possible to implement an obstacle detection and avoidance system using only a monocular camera. From the results there is a 90 percent rate that the obstacle will be avoided. However, an external computer is used to detect the obstacles and it can only detect obstacles which are in front of the drone, preferably in the center.

### 2.5.2 Stereo Camera

The stereo camera uses the same principle as the human eye. By placing two cameras at a small distance from each other the depth can be calculated using triangulation and for instance 3D Gaussian distributions [71]. This is a computationally intensive method to determine the depth which could be utilized by an FPGA. Stereo cameras available typically have a maximum depth range of 10 till 20 meters.

*Y. Xiao et al.* [72] proposes a multi-obstacle detection algorithm based on stereo vision which has a range between 1 and 15 meters. The tests prove that this is a possible solution. The only requirement for it in order to work is that there is good weather conditions since the camera relies on external light.

### 2.5.3 3D Camera

A 3D camera works different than the other cameras. It uses an extra infrared emitter and infrared image sensor to determine the depth. This is either done via the structured light or Time of Flight principle. These sensors can be made very compact since they only need an infrared emitter and an infrared sensor to determine the depth. Hence, why these sensors are used in small devices like smart phones. The sensors typically have a range of up to 5 meters.

*Jia Hu et al.* [73] shows that using a RealSense R2000 which uses structured light scanning [74] to perform obstacle avoidance on a UAV. Their solution shows a working example as long as the obstacles are not placed to close. Furthermore, they also used an external processing board, the Jetson TK1 [75] to process the depth image. Additionally, this solution is for indoor flight and not for outdoor flight which will be the case for this project. Flying outdoor causes extra interference from the sun.

### 2.5.4 Radar

Radar uses a high frequency signal of multiple Gigahertz to measure the distance. It sends out short pulses and measures the time it takes to reflect back to the sensor and then calculates the distance. This is presented as a 3D depth image . A big benefit of radar is that it works independent of the weather conditions, thus even when there is fog.

*S. Clark et al.* [76] discusses a 77 GHz millimetre wave radar as a guidance sensor for autonomous land vehicle navigation. Using an extended Kalman filter it maintains an estimate of map features. The map is made with radar reflectors designed for high visibility. The entire mapping process is done without any prior knowledge of the environment. The usage of radar reflectors is done because only radar is used for position estimation, if an extra sensor as for instance GPS is used the radar reflectors are not needed.

### 2.5.5 Ultrasonic

Ultrasonic uses sound waves instead of radar waves. These sound waves are fired in rapid succession, reflected by an object and when received again by the sensor, the distance is calculated. Since it uses sound it is better for short range applications and not suitable for long range. It could be used to assist a long range sensor for nearby application. A typical ultrasonic sensor has a maximum range of 5 meters. An interesting remark is the possible interference of measurements from wind current, either generated by the rotors or regular high-speed wind [77].

*J. Lim et al.* [78] proposes a rotating ultrasonic range sensor to measure distance to the walls and detect obstacles. A smart phone is used to process the data of the sensors for the heavy-duty computations. The usage of rotating ultrasonic range sensors ensures that less sensors are required and reduces the scanning time. The experiments within the project shows that it is feasible to use a rotating ultrasonic range sensor for indoor navigation. No tests were conducted where objects and walls were further away than 4 meters which is the maximum range of the ultrasonic sensor being used.

### 2.5.6 Infrared Time of Flight

Infrared Time of Flight sensors are sometimes included within 3D camera's to determine the depth. They use an infrared LED and Infrared receiver and by measuring the time of flight the distance can be retrieved. These sensors are compact since only a single infrared LED and infrared receiver is required. The big downside to these sensors is that they heavily rely on reflection of the object. A black object does not reflect well and with the already limited range of approximately 2 meters, it is therefore not feasible for this project. No example implementation of an infrared navigation system was found in literature.

### 2.5.7 Lidar

Lidar works by illuminating a target with laser light and measuring the reflected light with a sensor. Because it uses a laser instead of, for instance ultrasonic or Infrared the range is longer, up to several 100 meters. The range is, however, heavily influenced by the reflected object. However, since the laser is really concentrated it can still detect these objects from 10s of meters compared to 5 meter maximum range of the other sensors. Lidar is used for instance in automotive. Many of the Lidars on the market are capable of mapping the environment in 360 degrees by using a rotating Lidar. Because of the narrow beam used by the Lidar this is still on a 2D plane. High end sensors can even have a vertical field of view of up to 40 degrees to create a very complete 3D map of the environment with only one single sensor. These sensors however are much more expensive compared to the other sensors and the Lidar technology

is affected by fog. Additionally, they contain rotating sensor which has the disadvantage that it will wear and will therefore be less reliable.

*Y. Peng et al.* [79] proposes an obstacle avoidance algorithm based on 2D Lidar. For this test a 180 degree Lidar is used that looks into the forward direction. In the end the Lidar was tested in a static way and the obstacle avoidance algorithm was simulated using MATLAB.

### 2.5.8 Conclusion

Table 2.2 compares the typical characteristics of various sensor techniques.

Table 2.2: Typical sensor characteristics

| Sensor technique | Depth Range | Beam width | Accuracy | Cost |
|---|---|---|---|---|
| Camera | - | 70 degrees | - | ∼ €10 |
| Stereo Camera | 20m | 110 degrees | ∼1cm | ∼ €450 |
| 3D camera | 5m | 70 degrees | ∼1cm | ∼ €100 |
| Radar | 160m | 42 degrees | ∼10cm | unknown |
| Ultrasonic | 4m | ∼15 degrees | ∼1cm | ∼ €5 |
| Infrared TOF | 2m | ∼10 degrees | ∼1cm | ∼ €10 |
| Lidar | 40m | ∼1 degree | ∼2.5cm | ∼ €130 |
| 360 Lidar | 40m | 360 degrees | ∼2cm | ∼ €600 |

From Table 2.2 and what was stated before a couple of remarks can be made.

**Camera** has the possibility to detect obstacles with only a single camera with a 90 percent success rate. This solution however, has difficulties with recognizing objects which appear suddenly.

**Stereo Camera** is a viable solution, the FPGA could be used to process the data from the stereo camera to create a depth image. Stereo camera does not have the possibility to work at night since it needs external sunlight.

**3D Camera** is not a good solution because of the interference of sunlight.

**Radar** is a good solution, it has a long range in combination with a good width, however these sensors are very limited available or made for specific applications.

**Ultrasonic** is good for short range application, it is also really cheap where more expensive ones up to 6 meters range. Ultrasonic sensors are not affected by the sun but can be possibly affected by wind.

**Infrared Time of Flight** is not a good solution for the same reason as 3D camera, there will be too much interference from sunlight.

**Lidar** sensors have a very good range but a very small beam width, these sensors are more expensive than Ultrasonic sensors and are affected by the weather conditions.

**360 Lidar** offer similar range as the normal Lidar sensor but the beam is in 360 degrees. One sensor can map the environment around the drone making them very suitable for this project, they have the same negatives as the Lidar, being expensive and interference from sunlight.

In the end the choice for which sensor setup to use will be influenced by what the flight speed will be. Ultrasonic sensors with only a range of 5 meters can be used when the flight speed will be 1 m/s but when flying at 10 m/s this can cause some difficulties. The best solutions from these comparisons are the Ultrasonic, Stereo Camera, Radar, Lidar or 360 Lidar. Different sensors with these techniques will be further investigated in the Section 2.6.

## 2.6 Trade-offs of sensors for navigation

There are a lot of different sensors available which can be used for the navigation that use the techniques discussed in the previous section. This is a short overview of the sensors available. A more extensive list can be found in Appendix A.

### 2.6.1 Stereolabs zed

The stereolabs zed is a stereo camera with an integrated processing unit. It has a big field of view of 90 by 60 degrees and works outside. It has a reported depth range of 20 meters and with the correct calibration it can reach up to 40 meters of range. It is connected via a USB3.0 port and its data can be retrieved via ROS or the internal API. The package of the sensor is relative compact and it only weighs 159 grams. The sensor is shown in Figure 2.1.


Figure 2.1: Stereolabs zed [1]

### 2.6.2 µSharp

The µSharp is one of the few radar sensors available for consumers which does not use the Doppler effect [80]. It is developed by Aerotenna and has a field of view of approximately 50 by 30 degrees. It uses a frequency of 24 Ghz to determine the distance with a maximum range of 120 meters. The output from the sensor can be retrieved via Universal asynchronous receiver-transmitter (UART). The sensor is in a small housing and weighs only 43 grams. The sensor is shown in Figure 2.2.


Figure 2.2: µSharp [2]

### 2.6.3 HC-SR04

HC-SR04 is one of the most used ultrasonic sensors in robotics made by various companies. It works by first writing a high signal to the trig pin which then sends out an 8 cycle sonic burst and via the echo pin the duration will be returned. This duration should then be multiplied by 0.034 and divided by 2 to get the correct distance. It has really small field of view of approximately 15 by 15 degrees and a maximum range of 4 meters. The sensor is much smaller than the other sensors because of its simplicity and weighs only 8.5 grams. The sensor is shown in Figure 2.3.

### 2.6.4 LIDAR-Lite v3

The LIDAR-Lite v3 is a laser sensor which uses a 905 nm wavelength laser to measure the distance. It is developed by Garmin and comes in a normal variant which works at 500 Hz. A High Performance version which works at 1 kHz and has a water-resistant casing is available. The field of view for these sensors is much smaller of only approximately 0.5 by 0.5 degrees but the range is much larger with 40 meters. The sensor can be read via either PWM or I2C. It weighs 22 grams for the normal version and 38 grams for the High Performance version. The sensor is shown in Figure 2.4.


Figure 2.3: HC-SR04 [3]

Figure 2.4: Lidar-Lite [4]

### 2.6.5 RPLidar A3/S1

RPLidar is a 360 degree Lidar meaning it can map in a full circle of 360 degrees. It is developed by Slamtec and comes in multiple variants. The S1 sensor is the successor of the A3 sensor and comes with outdoor functionality. The field of view of these sensors is 360 degrees by 0.4 degree with a maximum range of 20 meters for the A3 and 40 meters for the S1. The sensors can be read via UART which then sends the current degree and distance. The RPLidar A3 weighs 190 grams and the RPLidar S1 weighs 105 grams. The sensor is shown in Figure 2.5.

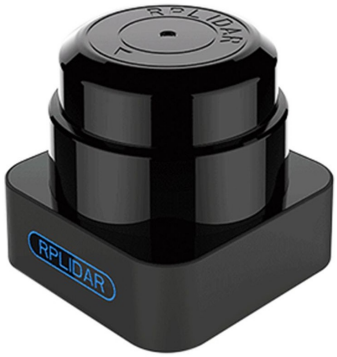


Figure 2.5: RPLidar S1 [5]

### 2.6.6 Puck Lite

The Puck Lite is a 360 degree Lidar which is developed by Velodyne and is the lighter version of the compact Puck sensor. The field of view of the Puck Lite is 360 degrees by 30 degrees and a maximum range of 100 meters. The sensor can be read via a 100 Mbps Ethernet connection. The Puck Lite weighs 590 grams. The sensor is shown in Figure 2.6.


Figure 2.6: Velodyne Puck Lite [6]

### 2.6.7 Conclusion

An important aspect for the sensors is the power impact which can be derived using a small calculation.

The battery that is going to be used on the drone is 48.8 Wh, the motors are the TMotor 900 kV brushless DC motors. This means that for every volt it spins at 900 rotations per minute. At 100% throttle the motor consumes 147.4 watts and generates a thrust of 1000 grams resulting at an efficiency of 6.78 grams per Watt [81]. The efficiency of the motor decreases the more throttle is used and thus the worst-case scenario will be used. This means that the power impact can be approximated as

$$Powerimpact = Powerusage + \frac{weight}{6.78} \tag{2.1}$$

A description of potential sensors can be seen in Table 2.3. A table with more sensors can be found in the Appendix A. It can overall be observed that the longer the range the higher the power usage. In the text there was only a small reflection on the weight of the sensor but this is of real importance. The drone must be able to lift all the weight of the drone and the sensor. Additionally, Equation 2.1 shows, the higher the weight, the higher the power impact.

Table 2.3: Overview of potential sensors

| **Sensor** | **Max range** | **Beam width** | **Power** | **Weight** | **Power Impact per sensor** | **Cost** |
|---|---|---|---|---|---|---|
| Stereolabs ZED | 20 meters | ∼90 degrees | 2 W | 159 g | 25.5 | ∼€450 |
| µSharp | 120 meters | ∼50 degrees | 1.25 W | 43 g | 7.6 | €625 |
| HC-SR04 | 4 meters | ∼15 degrees | 75 mW | 8.5 g | 1.3 | €5 |
| LIDAR-Lite v3 | 40 meters | 0.5 degree | 650 mW | 22 g | 3.9 | €130 |
| LIDAR-Lite v3 HP | 40 meters | 0.5 degree | 325 mW | 38 g | 5.9 | €150 |
| RPLidar A3 | 25 meters | 360 degrees | 6 W | 190 g | 34.0 | €600 |
| RPLidar S1 | 40 meters | 360 degrees | 1.75 W | 105 g | 17.2 | €650 |
| Puck LITE | 100 meters | 360 degrees | 8 W | 590 g | 95.0 | ∼€4000 |

It can be observed that the power impact factor is heavily influenced by the weight of the sensor. The Lidar-Lite v3 HP uses half the power but due to it weighting 16 grams more the power impact is much higher. It can also be observed that the RPLidar S1 has a really good power impact compared to the other 360 lidars and even compared to the Stereolabs ZED. The puck LITE is the best sensor overall if we do not take the Power Impact in consideration. However, it would cost a lot of extra energy to fly with the puck LITE sensor and the interface via Ethernet is not optimal for the platform.

From what is found the RPLidar S1 is the best choice. It has a 40 meter range inside with a range of approximately 20 meters outside which should be sufficient for obstacle avoidance.

## 2.7 Implementing drone-software on an FPGA

Multiple studies have been performed in implementing drone software on an FPGA.

*B. L. Sharma et al.* [82] discusses what type of flight controller, PID, KALMAN or Fuzzy Logic controller can be implemented on an FPGA. In the end they concluded that the fuzzy logic controller is best suited because it can better react to unexpected conditions as for instance weather.

*J. Kok et al.* [83] describes the development of a path planner for UAVs running on an FPGA using Evolutionary Algorithms. It is concluded that with the implemented path planning algorithm the FPGA is a suitable platform for flight-constrained autonomous UAV applications.

In *F.A. Abouelghit et al.* [84] two things are being discussed. A fault-tolerant FPGA-base architecture and fuzzy logic to design an obstacle avoidance system. For electronic components radiation and electromagnetic interference can seriously affect system reliability. In order to create a Fault-Tolerant Architecture they used the 1-out-of-2 Fault Tolerant techniques with two identical FPGA's and a Micro controller. When the micro controller no longer receives the watchdog signal it tries to re-program the FPGA. If this does not work it will disconnect the FPGA and the output of the other FPGA is used until it is repaired. The fuzzy logic obstacle avoidance was tested using MATLAB simulations and proved its superior performance.

*G. Premkumar et al.* [85] describes the design and implementation of an FPGA based flight controller. The complete system is implented in the programmable logic of a ZED board. The end result is that the drone controller system is fully running on the ZED board. It is however only capable of very limited flying where it requires an input from the user to do something.

*N. Monterrosa et al.* [86] describes the development and implementation of a UAV flight controller based on a state machine on an FPGA is discussed. For this project a fixed-wing UAV was being developed. They used an FPGA in collaboration with an ATmega processor which reads the sensors. In the end parts of the prototype were developed and these parts were being verified with simulations, but it was never tested in a real life application.

These papers show that it is possible to implement parts of autopilot software on an FPGA and show the advantages of using an FPGA versus a micro controller.

## 2.8 Design flow

The design flow is of importance to research what would be the way to implement an autonomous drone.

*P. smyczynski et al.* [87] discusses the system design flow of an autonomous drone control system with object tracking. It started with the usage of an already working flight controller. The pixhawk and as a computing platform it uses the Raspberry Pi. The software is build in a modular way and the communication between these modules had to be fast and stable. ROS is used for that. Since the pixhawk is taking care of the flying the only focus had to be in the vision system, mission planning and sending commands to the flight controller.

The paper of *A. Janarthanan et al.* [88] discusses the design of an UAV with autonomous flight path planning. However, this research uses a flight controller and a Raspberry Pi separately for the path planning. They started their design by using the working flight controller as a basis.

*E. Chirtel et al.* [89] discusses the development of an autonomous quadcopter which is spatially aware using a SLAM algorithm. A phone is used to read the data from a Lidar sensor. The map is created and high-level flight planning occurs on the smartphone. A Pixhawk v4 running Ardupilot receives high-level mission commands as speed and direction setpoints from the phone and will execute these.

The paper of *W.Y. Lai et al.* [90] discusses an enhancement of an off-the-shelf product that is capable of achieving semi-autonomous flight. Features like auto-throttle which ensures that the drone will fly at a certain height. This is done by using a flight controller based on the Arduino Mega and an ultrasonic sensor pointing downwards that measures the distance.

These papers show that for most of the projects involving an autonomous drone a flight controller is used in collaboration with a computing platform. These papers do not go in depth about what design flow is exactly used. It seems that most of these first realize a flying platform. After that develop the specific hardware for the project, combine it and test it.

## 2.9 conclusion

The novelty of this research will be describing the complete design flow of developing an autonomous drone on a MPSoC. This includes describing the design flow, sensor selection for obstacle avoidance, and proving that it works. The state of the art implementations of obstacle avoidance in drones start with a working prototype and extending this with an extra board and sensor. This research proposes the design flow on a new platform with taking obstacle avoidance into consideration. This causes different problems to occur compared to starting with a ready to fly platform.

# Chapter 3

# Background

This chapter presents background information regarding the essential topics for this thesis. First, it is discussed what a quadcopter is. Secondly, autopilots are being discussed and what they exactly do. Then, the hardware for current UAVs is discussed. After that, the XDP-platform is discussed. Next, real-time is discussed. Lastly, the build environment is discussed.

## 3.1 Quadcopter/Drone

A quadcopter stands for quadrotor helicopter, this means that it is a helicopter propelled by four rotors. It is called a drone when there is no human pilot on board. It either navigates autonomously or is controlled by a human pilot which is not present in the vehicle (for instance via Radio Control). A quadcopter will thus always consist of four rotors and can be controlled by an on board human pilot. A drone can have any amount of rotors or can even be a non-aerial vehicle.

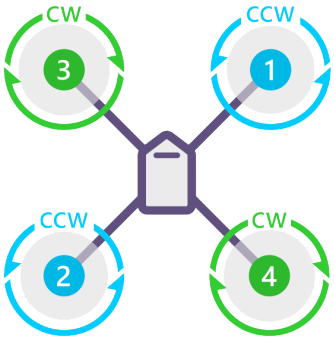


Figure 3.1: Quadcopter [7]

As can be observed from Figure 3.1 two of the rotors spin clockwise while two rotors spin counterclockwise. This behaviour prevents the quadcopter from spinning around its axis when it flies. This behaviour can also be observed in a helicopter where the tail rotor prevents this behaviour.

A quadcopter has four main parameters; roll, pitch, yaw, and throttle. A roll is performed by increasing the thrust on either the left or right side of the drone. This makes the drone move to the left or to the right. Pitch is performed by increasing the thrust either in the front or in the back rotors. This makes the drone move forwards or backwards. Yaw is performed by increasing the thrust on the rotors that move in the same direction. This makes the drone spin around its axis. Throttle is important for the altitude of the drone and at what speed the drone flies. The more throttle the more thrust all rotors will generate and thus fly higher or move faster.

## 3.2 Autopilot

An autopilot is a system used to control the trajectory of an aircraft without constant 'hands-on' control by a human operator being required [91]. Most autopilots available are very similar. They all have the same goal, make the vehicle move autonomous. An autopilot requires input from various sensors in order to register the movement of the vehicle e.g. gyroscope, accelerometer, magnetometer, and barometer. Extra sensors can be added to increase the accuracy or add extra features. The autopilot will use the sensor data to calculate which path has to be taken in order to reach the specified target. The motors will then be controlled in order to reach such target.

## 3.3 Hardware

Most of the consumer drones use the same kind of processors for the flight controller namely the cortex ARM M series. These are 32 bit processors which use Reduced Instruction Set Computer (RISC) which is known for its high energy-efficiency and cheap price. Most flight controllers use only a single core M7 processor for the flight controller. Flight controllers only have to read a small amount of sensors and control the motors according to the input received from the remote controller. Optionally they have to transmit the camera feed. These computations do not require a lot of computational power and therefore a small single core M7 is sufficient. When additional features are required as for instance mapping of the environment, this is handled by the specific peripheral. It is done via post processing or it is done by an external processor.

## 3.4 XDP

XDP stands for Xilinx Drone Platform and is the platform developed by Topic specifically for the usage in drones and can be observed in Figure 3.2.

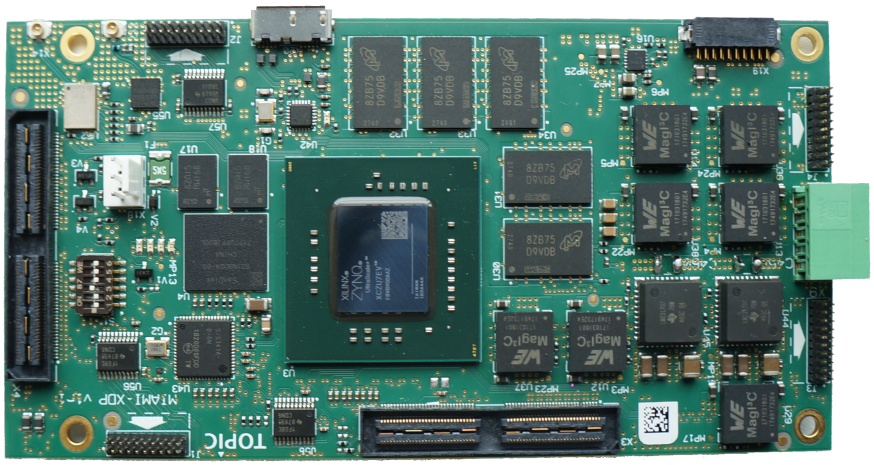


Figure 3.2: The XDP

For hardware it has all the required sensors available in order to fly. The IMU is the Bosch BMI088 [92] which is a 6-axis motion tracking sensor. It combines its accelerometer and gyroscope data to form an IMU. Due to its high vibration robustness and small footprint it is ideal for the usage within drones. The magnetometer is a Bosch BMM150 [93] which is a low power and low noise 3-axis digital geomagnetic sensor. Due to the stable performance over a large temperature range, the BMM150 is ideal for the usage

within drones. The data of the magnetometer can be fused with the data of the IMU to increase the reliability. Additionally, there is an environmental sensor, the Bosch BME680 [94]. It has the capability to measure barometric pressure and altitude, humidity, temperature, and gas. Barometric pressure is of importance for drones since this can be used to calculate the altitude. For the Global Positioning System (GPS) the ZOE-M8B module [95] is used which is a low power GPS sensor. It uses Global Navigation Satellite System (GNSS) to ensure that it can use multiple navigation systems. For processing the MPSoC contains four A53 cores, two R5 cores, ARM-Mali400 GPU and Programmable logic. A small overview of the MPSoC is shown in Figure 3.3.

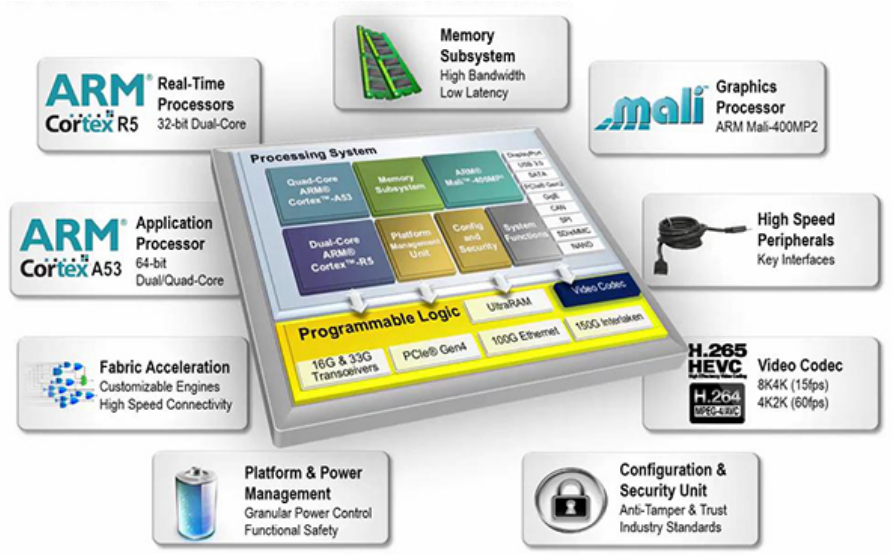


Figure 3.3: MPSoC overview [8]

## 3.5 Real-time

In Section 3.4 it is mentioned that the MPSoC contains Real-Time Processing units. These have the capability to run a Real-Time Operating System (RTOS). A RTOS guarantees that tasks will be run with very consistent timing. Additionally, tasks can be preempted, have deadlines and different schedules can be chosen. This makes the use of a RTOS beneficial for time critical application e.g. an airbag or the flight controller of a drone.

## 3.6 Build environment

The build environment used within Topic is the Yocto project. The Yocto project is an open-source project that is used to create operating system images for embedded Linux devices. The tools from the Yocto Project are based on the OpenEmbedded project. It uses the BitBake build tool to create Linux images. A Yocto project typically consist of several meta layers. These layers include multiple hardware and/or software components scattered over multiple recipes located in the meta. For instance there is a ROS layer which contains multiple recipes like the navigation core, opencv camera and many more.

Using BitBake an image can be generated which can be written to an SD card to be used by the MPSoC.

# Chapter 4

# Design

## 4.1 Required computation of PX4

As mentioned in Section 2.3.1, PX4 is the flight controller part of dronecode, it is responsible to ensure a safe flight. This consists of multiple parts, a small overview is given in Figure 4.1. A more extensive overview can be observed in Appendix B. Appendix B shows most of the essential modules used for flight and the messages being communicated between these modules.

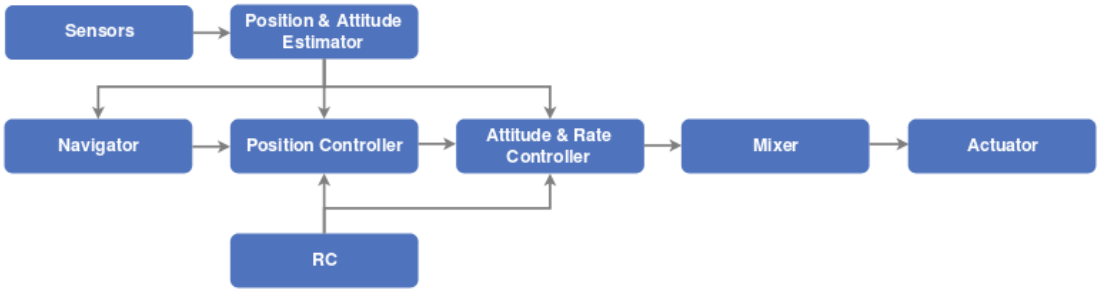


Figure 4.1: Flight stack overview [9]

### 4.1.1 Messaging protocol

The communication between these modules is done via the micro Object Request Brokers (UORB) protocol. The UORB protocol uses the shared memory to communicate between the modules. It is asynchronous and lock-free, a subscriber does not wait for a publisher and vice versa. Modules can publish and subscribe to a topic to which the messages are being communicated. This has been achieved by using a separate buffer between a publisher and subscriber. The UORB protocol has been optimized to minimize the memory footprint and latency.

### 4.1.2 Sensors

The sensors module gathers the low-level output from the sensor drivers and turns it into a more usable form for the rest of the system. It will gather the input from all the different sensor drivers. When there are multiple sensors of the same type it will do voting and failover handling. Additionally, it will apply the board rotation and temperature calibration. Finally, it publishes the data such that it can be used by the rest of PX4.

### 4.1.3 Local position estimator

The local position estimator works in collaboration with the attitude estimator discussed in Section 4.1.5. The local position estimator uses an extended Kalman filter to generate the 3D position and velocity states. It uses the data from the sensor that was combined by the sensors module. It publishes the vehicle local position, where the origin is the start point and vehicle global position, which is the actual position on the world map.

The local position estimator publishes:

- Velocity at the IMU - North,East,Down (m/s)
- Position at the IMU - North,East,Down (m)
- IMU delta angle bias estimates - X,Y,Z (rad)
- IMU delta velocity bias estimates - X,Y,Z (m/s)
- Earth Magnetic field components - North,East,Down (gauss)
- Vehicle body frame magnetic field bias - X,Y,Z (gauss)
- Wind velocity - North,East (m/s)

The computation of a Kalman filter can be found in *R.E. Kalman* [39]. The extended Kalman filter which is used by PX4 can be found in *S.J. Julier et al.* [96]. The extended Kalman filter can also be implemented on an FPGA as proven by *L. Idkhajine et al.* [97].

### 4.1.4 Navigator

The navigator is the module that is responsible for the autonomous flight modes. Additionally, it is also responsible to check for Geo-fence violations. Using the Inertial Navigation System [98], the trajectory, corrections, and the global position are calculated and send to the rest of the system. The navigator receives commands from the commander, for instance takeoff or fly to a certain destination. After that it sends 'vehicle command messages' to the commander what the drone should do. At the end of the message the navigator will report whether the mission is completed or not.

### 4.1.5 Attitude estimator q

The attitude q estimator is a simple quaternion based complementary filter. It calculates the IMU data, which it receives from the sensors module. It publishes the vehicle attitude. This consist of the Quaternion rotation from XYZ body frame to NED earth frame and the amount by which the quaternion has changed during last reset. A complementary filter is useful in cases where there are two different measurement source for the estimation of one variable. One source gives information in low frequency region while the other source gives it in high frequency. An example would be data from the IMU and GPS data.

### 4.1.6 Mc att control

The multicopter attitude and rate controller takes the vehicle attitude setpoint as input and outputs the actuator control messages. The attitude and rate controller contain 2 loops. A P loop for the angular error and a PID loop for the angular rate error as researched by *D. Brescianini et al.* [99]. Depending on the mode, the outer P loop is bypassed, this loop is only used when holding position or when the requested velocity on an axis is null. An overview for this controller can be observed in Figure 4.2. In order to reduce the control latency of the mc att control it directly polls the gyro from the bmi088 module.

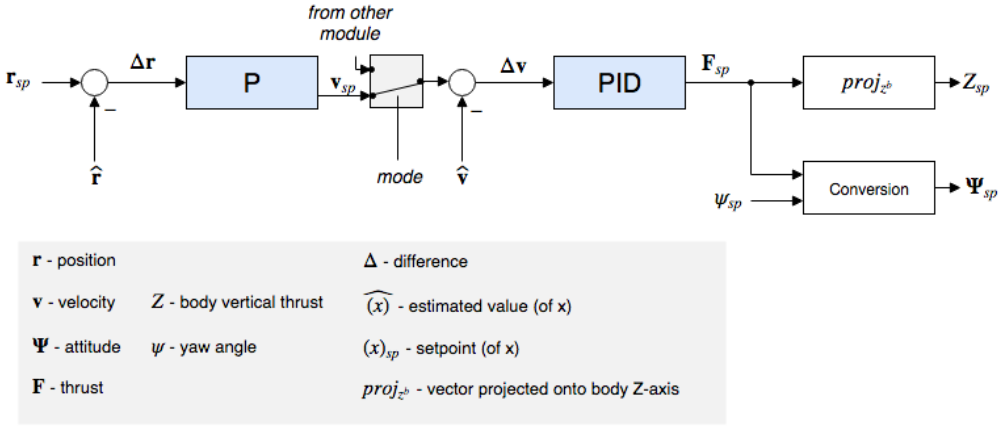


Figure 4.2: Multicopter position controller [10]

### 4.1.7 Mc pos control

The multicopter position controller outputs the vehicle local position setpoint / trajectory setpoint. It contains two loops just like the mc att control, a P loop for position error and a PID loop for the velocity error. The output of the controller is a thrust vector. It is split into thrust direction (rotation matrix for multicopter orientation) and thrust scaler (multicopter thrust itself).

### 4.1.8 Mixer

The mixer translates the commands received from the controller to values that can be used by the actuator. The output is calculated for every rotor by multiplying the roll, pitch, yaw, and thrust by their scale and then sum these results. These are all very basic operations which require limited processing power. It is included within the Linux pwm out.

### 4.1.9 Linux pwm out

The Linux pwm out module is responsible to control the motors. It reads from the actuator_controls_0 topic and sends this via a file to the motors. The connection between the processing system to control the motors will be explained in Section 5.1. Additionally, it also receives data from the actuator_armed topic which ensures that when the motors are armed, they always spin.

### 4.1.10 Land detector

The land detector is the module that detects free fall or the landed state of the vehicle and publishes it on the vehicle_land_detected topic. To land it reports 3 possible states, ground contact, maybe landed and landed. Ground contact is the first step after which the thrust setpoint is lowered. Maybe landed requires the ground contact with lower thrust and no velocity in the horizontal plain. After the maybe landed state has been active for a minimal amount of time the quadcopter concludes to be landed.

### 4.1.11 Commander

Commander is one of the most important modules within PX4. It is responsible to calibrate the accelerometer, gyroscope, magnetometer, and determine the level horizon. Furthermore, it sends the different commands to the navigator as for instance takeoff, land or go to a certain position. It will do all the pre-flight checks in order to ensure that all the sensors are calibrated. During flight the commander can be thought of as a big state machine that switches between different states according to the needs of the user.

### 4.1.12 Conclusion

The calculations done within the application are very simple but very powerful when combined. In the end the computation power available within the platform is expected to be sufficient in order to correctly implement PX4.

A small test on the board with running PX4 confirms this. The average load of PX4 is approximately two percent, the A53 core has a DMIPS of 2.3 DMIPS/MHz. Since the four A53 cores run at 1.2 GHz for a total of $4 * 1200 * 2.3 = 11040$ DMIPS. Approximately 2 percent is used thus $11040 * 2\% = 220.8$ DMIPS is required for PX4 to run.

## 4.2 Required computation for ROS

ROS is required for the obstacle avoidance in collaboration within PX4. In a normal setup ROS runs on a companion computer. For this implementation the ROS environment will also run on the platform itself. The sensor that will be used for the project is the RPLidar S1 sensor. Data from the sensor can be retrieved via UART at a baud rate of 256000. This means that the port is capable of transferring a maximum of 256000 bits per second or 256 kilobits per second.

The sensor receives 360 degrees of sensor data with an angular resolution of 0.391° for a total of $\frac{360}{0.391} \approx 920$ points per rotation. Since it works at 10 Hertz there will be 9200 samples per second. Each of these samples has to do 2 multiplications and 1 division to get the correct compensated angle. In total this evaluates to approximately 27600 calculations per second to retrieve the sensor data and convert it to the correct angle. If it would be assumed that 1 calculation takes 2 instructions this would still only be 55200 instructions per second. Since the processor has approximately 11040 DMIPS available it is expected that the computation for ROS fits within the system.

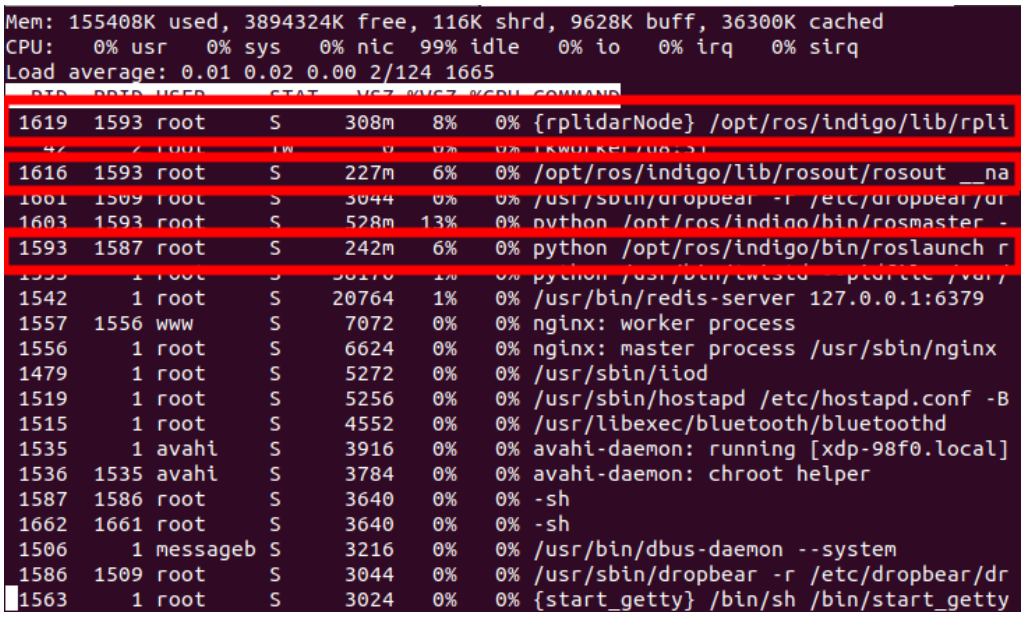


Figure 4.3: CPU usage when running ROS with Lidar, processes are highlighted

From Figure 4.3 it can be observed that when running on the A53 cores the CPU load is indeed very low as it is below 1 percent.

## 4.3 Mapping of the processes

The MPSoC used on the XDP-platform is the Zynq Ultrascale+ ZU7EV device. It can be observed in Figure 4.4 and contains two main blocks, the Processing System (PS) and Programmable Logic (PL). The PS consists of an APU which consists of four ARM Cortex-A53 cores, a GPU which is an ARM Mali-400, a Real-Time Processing Unit which consists of two ARM Cortex-R5 cores and some additional peripherals. The PL is the FPGA part of the MPSoC, it consists of 504 thousand System Logic Cells, 38 Mb of memory, 1728 DSP Slices and offers a maximum of 464 I/O Pins including 12 Gigabit transceivers.

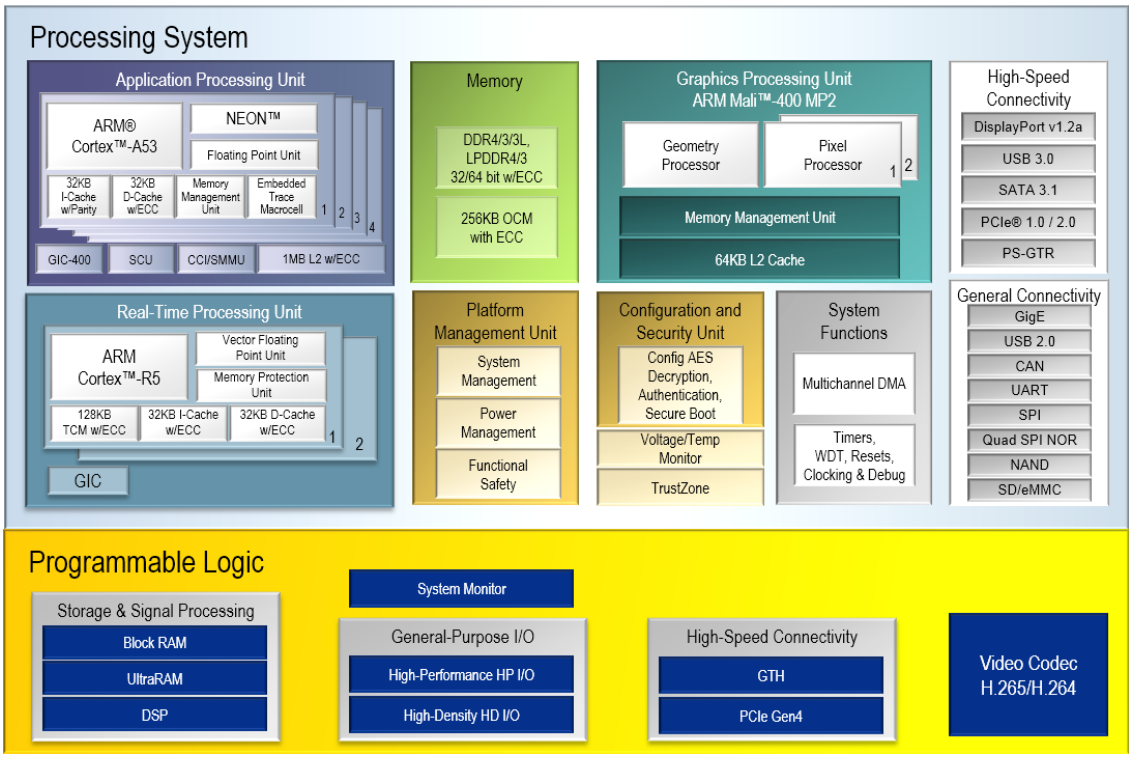


Figure 4.4: The Zynq Ultrascale+ [11]

The system architecture for the Dronecode platform can be observed in Figure 4.5.

It can be observed that the flight controller which consists of many different modules is drawn as one big block. It is drawn this way because all the communication happens within the flight controller. Therefore it is preferred to have this run on the same processor.

The drivers for the Rotors, IMU, Barometer, Distance, and GPS are connected via their respective connections. They can be read either in the Programmable Logic or in the Processing System.

Within the platform there are 4 theoretical possibilities where the autopilot can be placed; Programmable Logic, A53 cores, R5 cores or the mali-400 GPU. It is however, not a feasible solution to implement the entire autopilot on either the Programmable Logic or the mali-400 GPU. To implement the autopilot or a flight controller would cost a lot of resources of the FPGA and leads to a long implementation time. To implement the autopilot on the mali-400 GPU would be extremely inefficient. A GPU excels in simple parallel computation's but not in running a complex state machine.

Either the Real-Time cores or the APU seem therefore like the best solution. The advantages for each of these solutions will be discussed in the following sections. Additionally, there will be a small discussion about the FPGA and GPU.

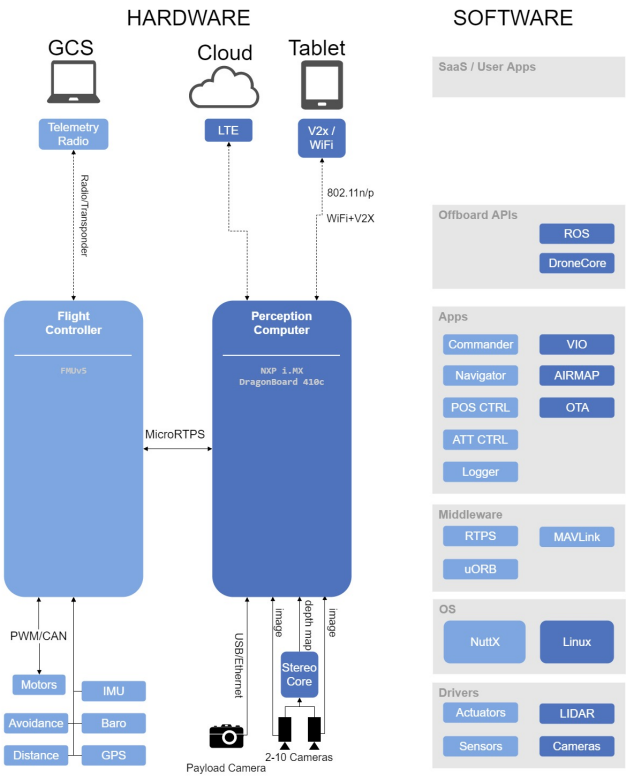


Figure 4.5: Dronecode platform [12]

### 4.3.1 Advantages Real-Time cores

The biggest advantage of running the entire system on the Real-Time Processing unit is that the overall behaviour of the system will be more deterministic. The Real-Time Processing unit has preemptive possibilities which is better for security. When a single task hangs this will not influence any other tasks running on the Real-Time Processing unit. If an application hangs on the ARM cores it has the possibility to influence the other tasks. This could cause the miss of deadlines which could have big impact on the motor control.

### 4.3.2 Advantages A53 cores

The A53 cores have more compute power available compared to the Real-Time cores and it was proven that these can easily run PX4 and ROS. The current setup for the platform uses the A53 cores as basis for its Linux distribution and the Real-Time cores are not used at all. All the sensors already work on the Linux distribution which runs on the A53 cores. Additionally, the board support package (BSP) is already developed for the current Linux distribution and used within the build environment. Another big advantage of using the A53 cores is that ROS is runs on Linux which is an essential part for this project.

### 4.3.3 Advantages FPGA

An FPGA or also called the Programmable Logic has several advantages. The first one is that the FPGA has much more Input/Output (IO) available compared to APU or real-time cores. This IO can be read by the Programmable Logic in a very deterministic way at specific intervals. This is not possible for the A53 cores. The second advantage is that the Programmable Logic is able to pre-compute data. For instance, signal processing on data that comes in from the IO. This prevents the microprocessor from having to do these calculations. This reduces the load on the microprocessor. The third advantage is that it is possible to expand the platform with more complex algorithms. As long as these algorithms can be implemented in

the FPGA logic. Examples for these algorithm could be signal processing, image processing or even things like image recognition. An FPGA is able to do these applications in parallel where a microprocessor has to do these operations sequentially. The final advantage is that an FPGA is a much more suitable platform for redundancy of components. When double or triple redundancy is applied, the Programmable Logic can be multiplied by the same factor. An arbiter can be used to determine the correct outcome [100]. For a micro controller this would increase the computation required from a certain sensor by the amount of redundancy required.

### 4.3.4 Advantages of GPU

The GPU available within the platform is the mali-400 GPU. The mali-400 is a small GPU that is a pure 3D engine which renders graphics into memory and passes the rendered image over to another core to handle display. A big advantage of having a dedicated GPU is the capability for floating-point operations. Additionally, many signal-processing algorithms are designed for GPUs.

### 4.3.5 ROS on real-time processor

A big component of this research is based around ROS for the obstacle avoidance. According to the website, ROS can only run on a Linux or windows based system. However there are some examples in literature that claim to run ROS in real-time.

*H. Wei et al.* [101] ([102]) discusses a real-time ROS architecture on multi-core processors called RT-ROS. RT-ROS is a hybrid combination that supports both Linux and a RTOS. It can use ROS nodes running non-real-time and real-time. This is tested running Linux on one core and running Nuttx on the other core which communicate using shared memory. It is shown that the real-time core meets it constraints independent of the load on the Linux OS. The Master node however, is still running on the Linux core and can not run just on the real-time core.

### 4.3.6 Conclusion

Unfortunately, it is not possible to run ROS on the real-time cores alone, it needs cores running Linux. Since ROS is essential for obstacle avoidance the master has to run on the A53 cores. This means there are only two possibilities. Either everything runs on the A53 cores or there is a hybrid where the flight controller runs on the R5 cores and ROS on the A53 cores.

The use of a hybrid solution would have the benefit of the load being spread over the processors. The real-time cores actually being utilized and of course have the advantage of being able to run PX4 on an RTOS. Running PX4 on an RTOS has the advantages of it being more predictive, predictable, and staying reactive.

Running everything on the A53 cores has the advantage that the build environment is already setup and all the sensors are already working on the platform. It is unknown how much the effort would be to implement the hybrid solution in the existing system and to get all the sensors working on the RTOS.

The hybrid solution is likely the best solution in order to get everything to work correctly, however, it will likely cost a lot of time, First to get the real-time cores running a RTOS. Then to get all the sensors to work. Finally, to establish the connection between the RTOS and Linux distribution to get ROS to work. Due to the implementation time constraints the decision is made to implement both PX4 and ROS on the A53 cores. With the knowledge available it likely costs too much time to implement everything correctly. It will be interesting to see how the load of the A53 cores will influence the performance of the system.

The Programmable Logic is very suitable to implement the peripherals of the flight controller like the motors. This would ensure that there is a deterministic input and output which ensures that the motors are being controlled at a specified frequency. This should result in shorter delays and a more controlled behaviour. This also means that the motor control which is a critical component will no longer run on the A53 cores and cause the system to be more secure. When the A53 core hangs it will not influence the motor control loop.

The Programmable Logic could also be used to do signal processing over the data gathered from sensors or process data from additional sensors (sensor fusion). Additionally, redundancy could be im-

plemented in the Programmable Logic to ensure a safer platform but that is not within the scope of this project.

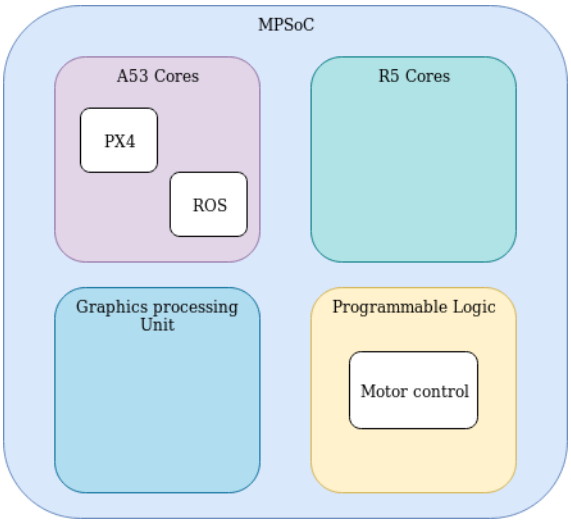


Figure 4.6: The mapping

The goal is to run both the perception computer and flight controller on the MPSoC. This causes the drone to become fully autonomous in the sense that the platform can fly on its own. Most other solution use a special companion computer that plans the route and avoids obstacles. This will prevent the need for a lot of communication over the air. This makes it possible to fly much further because there is no need for a connection to a ground station.

## 4.4 Obstacle avoidance

### 4.4.1 Connection between PX4 and ROS

ROS and PX4 have to communicate with each in order to utilize obstacle avoidance, for this Micro Air Vehicle Link (MAVLink) is used. MAVLink is a protocol to communicate with small unmanned vehicles. To use the MAVLink protocol within ROS the mavros node has to be used. Mavros makes it possible to transfer data from ROS to PX4 over the MAVLink in various ways, Serial, UDP or TCP.

### 4.4.2 Obstacle avoidance in PX4

Obstacle avoidance within PX4 can be achieved in two different ways, Offboard Mode Avoidance or Mission Mode Avoidance.

Offboard mode avoidance ensures that the desired route comes from a ROS node. This is passed into an obstacle avoidance module, which is another ROS node. The avoidance software sends the planned path to the flight stack as a stream of SET_POSITION_TARGET_LOCAL_NED messages. The SET_POSITION_TARGET_LOCAL_NED format can be seen in Table 4.1. Since the navigation is done within ROS for offboard mode avoidance, it is possible to use all sensors that can be used to detect an obstacle as within ROS

Mission mode avoidance is the other solution. PX4 communicates with the obstacle avoidance software using an implementation of the MAVLink path planning protocol. It is different in the sense that a waypoint is reached when the vehicle is within the goal radius of its goal not taking the heading into consideration. This is because the obstacle avoidance algorithm has full control of the vehicle heading. PX4 then emits a new waypoint when the goal is reached via the TRAJECTORY_REPRESENTATION _WAYPOINTS. The data format can be observed in Table 4.2. The avoidance software will respond with

Table 4.1: SET_POSITION_TARGET_LOCAL_NED [26]

| Field Name | Type | Units | Values | Description |
|---|---|---|---|---|
| time_boot_ms | uint32_t | ms | | Timestamp (time since system boot). |
| target_system | uint8_t | | | System ID |
| target_component | uint8_t | | | Component ID |
| coordinate_frame | uint8_t | | MAV_FRAME | Valid options are: MAV_FRAME_LOCAL_NED = 1, MAV_FRAME_LOCAL_OFFSET_NED = 7, MAV_FRAME_BODY_NED = 8, MAV_FRAME_BODY_OFFSET_NED = 9 |
| type_mask | uint16_t | | POSITION_TARGET_TYPEMASK | Bitmap to indicate which dimensions should be ignored by the vehicle. |
| x | float | m | | X Position in NED frame |
| y | float | m | | Y Position in NED frame |
| z | float | m | | Z Position in NED frame (note, altitude is negative in NED) |
| vx | float | m/s | | X velocity in NED frame |
| vy | float | m/s | | Y velocity in NED frame |
| vz | float | m/s | | Z velocity in NED frame |
| afx | float | m/s/s | | X acceleration or force (if bit 10 of type_mask is set) in NED frame in meter / $s^2$ or N |
| afy | float | m/s/s | | Y acceleration or force (if bit 10 of type_mask is set) in NED frame in meter / $s^2$ or N |
| afz | float | m/s/s | | Z acceleration or force (if bit 10 of type_mask is set) in NED frame in meter / $s^2$ or N |
| yaw | float | rad | | yaw setpoint |
| yaw_rate | float | rad/s | | yaw rate setpoint |

the TRAJECTORY_REPRESENTATION_WAYPOINTS message with only the current setpoint of the message. If the waypoint is inside an obstacle which makes it unreachable. The obstacle avoidance software tries to enlarge the acceptance radius, if this is not possible it might be stuck. Because mission mode avoidance uses the MAVLink path planning protocol, it therefore requires a point cloud and can thus potentially not work with all available sensors.

Figure 4.7 gives a simple visual representation of the main difference between both modes. It shows what messages are communicated between ROS and PX4. Additionally, it shows whether ROS or PX4 is in control to fetch the new waypoint and complete the mission. The mission mode is preferable since this does not change anything for the user. The user can still do the advanced flight modes from PX4 which would not be possible when using offboard mode control.

Different planners can be used for the path planning. The local planner and the global planner. The local planner is the default planner, it creates a local map of the environment with the data from the Lidar. It will not always find the most optimal path to its goal. The local planner will not save data about the map, only a local map. This causes the local planner to require less computation than the global planner.

The global planner uses a global map, the drone flies within the map and extend it with data from the Lidar. In a known environment the global planner will more likely find the optimal path its goal. Since it has to save the global map it requires more computation than the local planner.

The local planner is the default for PX4 and is heavily flight tested using 3D stereo cameras. Therefore, there has been chosen to use the local planner.

Table 4.2: TRAJECTORY_REPRESENTATION_WAYPOINTS [26]

| Field Name | Type | Units | Values | Description |
|---|---|---|---|---|
| time_usec | uint64_t | us | | Timestamp (UNIX Epoch time or time since system boot). The receiving end can infer timestamp format (since 1.1.1970 or since system boot) by checking for the magnitude the number. |
| valid_points | uint8_t | | | Number of valid points (up-to 5 waypoints are possible) |
| pos_x | float[5] | m | | X-coordinate of waypoint, set to NaN if not being used |
| pos_y | float[5] | m | | Y-coordinate of waypoint, set to NaN if not being used |
| pos_z | float[5] | m | | Z-coordinate of waypoint, set to NaN if not being used |
| vel_x | float[5] | m/s | | X-velocity of waypoint, set to NaN if not being used |
| vel_y | float[5] | m/s | | Y-velocity of waypoint, set to NaN if not being used |
| vel_z | float[5] | m/s | | Z-velocity of waypoint, set to NaN if not being used |
| acc_x | float[5] | m/s/s | | X-acceleration of waypoint, set to NaN if not being used |
| acc_y | float[5] | m/s/s | | Y-acceleration of waypoint, set to NaN if not being used |
| acc_z | float[5] | m/s/s | | Z-acceleration of waypoint, set to NaN if not being used |
| pos_yaw | float[5] | rad | | Yaw angle, set to NaN if not being used |
| vel_yaw | float[5] | rad/s | | Yaw rate, set to NaN if not being used |
| command | float | | MAV_CMD | Scheduled action for each waypoint, UINT16_MAX if not being used. |

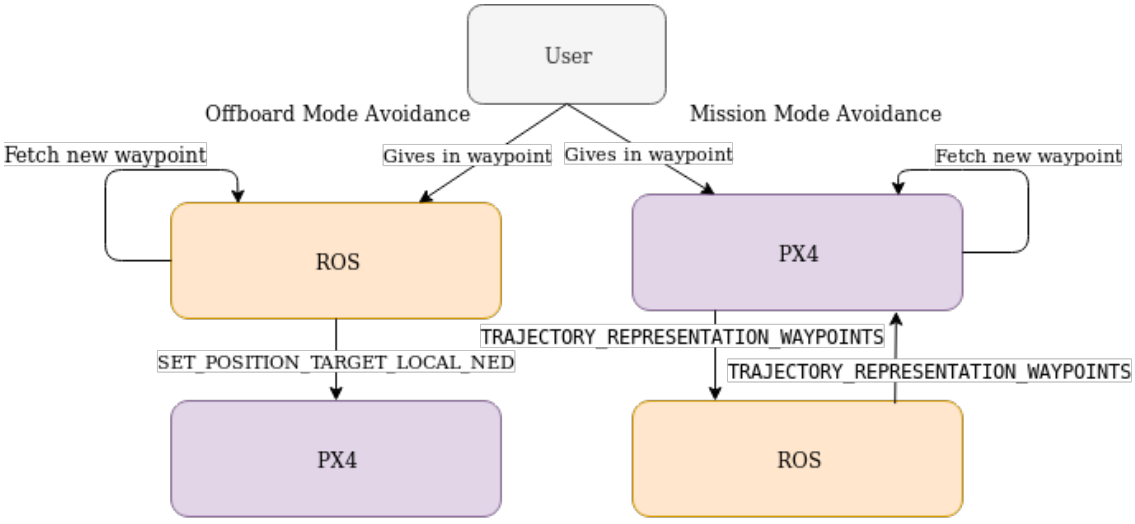


Figure 4.7: A visual representation of offboard and mission mode

## 4.5 Design flow

The implementation for this project is done in an agile/scrum way, each sprint it is determined what the goal will be for the sprint. At the start of every day day there will be a stand up during which the problems encountered can be discussed. At the end of the sprint it is the goal to always have a partially working product. The goal for the sprints are defined as milestones within the project. Milestones that take multiple sprints can be split in sub-milestones. These milestones can be for instance: Building the software on the platform, running the motors, reading the GPS, etc. Additionally, it is important that for each of these milestones there is a backup available. What happens if the milestone costs too much time or does not seem to be feasible? There should be a backup in such a way that the project can still continue with minor alterations.

It is important that after a milestone has been realized it is extensively tested. This way it prevents possible problems which are encountered at a much later stage and require more work to repair again. This also prevents incorrect assumptions being made which caused changes within the system.

Using this design flow results in if the project would come to a stop earlier, there would still be a prototype that could be shown. If some parts of the project take longer than expected in the end there will still be a partial working prototype, compared to having a prototype that is not working yet.

# Chapter 5

# Implementation

## 5.1 Connect Processing System and Programmable Logic

To control the motors from the Processing System via the Programmable Logic a couple of alterations have to be done and additional applications have to be developed.

### 5.1.1 Device Driver

A device driver is a software application that is designed to enable interaction between the software and hardware devices [103]. Device drivers are the bridge between the application software and the hardware. Using various methods data can be send to the device driver which will then use this data to do the corresponding hardware action.

A device driver generates a kernel module. A kernel module is a piece of code that is loaded or unloaded into the kernel. Because a device driver is loaded into the kernel it has access to various locations which are not accessible for user space programs. A kernel module that controls hardware is called a kernel driver. In this project the kernel module will write to a specific memory location and not directly control the hardware, in that sense it is not a real device driver.

The device driver that has been developed has a very simple task. When it receives a value, this value is written to a certain memory location. This value can be read later. When the Kernel driver is initialized it will generate the correct registers and set them to 0 by default.

### 5.1.2 Device Tree

The Device Tree describes the hardware components of a particular system in such a way that the kernel can use and manage those components. Typical instances described in a device tree are memory, CPU, and buses [104]. Since the kernel driver is going to write the data to a special memory address. Changes have to be made to the device tree. An additional node has to be added to the device tree which holds information like, the bus, address, and which drivers has to be loaded for this device. Additionally via the device tree parameters can be given to the Kernel drivers which in this case is the number of motors.

### 5.1.3 Programmable logic

The previous sections were about sending data from the processing system to a certain memory address. An important aspect however is that this data is going to be used by the programmable logic in order to control the peripheral, in this case the motors. In order to do so there is a special interface used on the FPGA called Advance eXtensible Interface (AXI) [105]. This address from the kernel driver should then be within the specified AXI memory region to form a memory mapped AXI [106].

To use the data received via the AXI protocol an Intellectual Property (IP) has to be developed which can be connected to the AXI bus. A project should be made which contains an AXI-slave. This AXI-slave translates the data in the correct way such that it can be used by parts of the program which directly

control the hardware. This project should then be packaged as an IP and added to the FPGA image and connected to the AXI bus.

An overall impression can be observed in Figure 5.1.

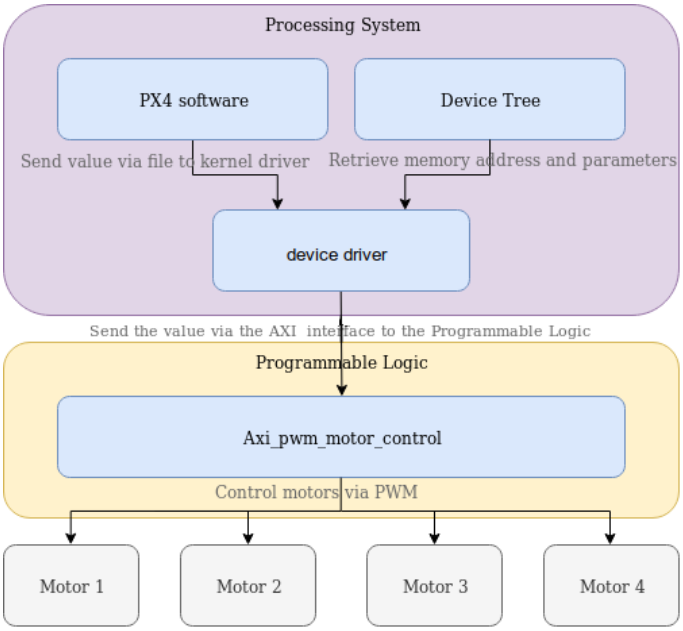


Figure 5.1: The communication between the Processing system and Programmable Logic visualized

## 5.2 How to utilize the sensors in PX4?

This task can be split in two separate tasks, reading the sensor data correctly and publishing the correct data on the PX4 network. These will be discussed in the following sections.

### 5.2.1 Reading the sensor data

The sensors are already implemented on the platform and can be read in two ways. They can be read via either the default file system, where files are located at various locations in sysfs or via the Industrial Input/Output (IIO) [107] standard. The advantage of using file operations to read the data from the sensors is that it is much easier to program. The basic file IO is already incorporated within PX4. The disadvantage however will be the speed when using file IO. It is necessary that each time the file is opened and thus also closed for each measurement. Because of this the amount of time it takes to read via the file IO is much higher. A small experiment was conducted where the file IO would be read a 100 times, a single time and also the data would be retrieved via libiio [108]. The results are shown in Table 5.1. This showed that on average, when it is invoked using libiio it is twice as fast as file IO. When invoked 100 times in a row libiio is still approximately 25 percent faster. The smaller difference when invoking it directly in quick succession could be explained by the data being loaded in cache. However, it is safe to conclude from these tests that for this system it is faster to use IIO with libiio than using file IO.

Table 5.1: Comparison Industrial IO and File IO

| | 1 time (ns) | 100 times (ns) |
|---|---|---|
| Industrial IO | 72821 | 6664427 |
| File IO | 185872 | 8670267 |

So what exactly is IIO? IIO has as main purpose to provide support for devices that perform either analog-to-digital conversion or digital-to-analog conversion at high sample rates. As is the case for most sensors. Each IIO device can be found in the Linux file system at `/sys/bus/iio/iio:deviceX/`. This folder contains the name of the sensor along with all it channels. Each sensor that is connected has a number of channels available. For instance, when a magnetometer sensor is connected there will be a channel for the X-axis, Y-axis, and Z-axis. Of course there can also be additional channels for the temperature and voltage. All of these channels have a number of attributes available which can specify for instance the scale, raw values or sampling frequency. For the magnetometer sensor used in this project an example is shown in Figure 5.2.

```
Device name bmm150_magn, Device channels count 4, Device attr count 2, Device id iio:device3

The channel id: magn_x with i 0 corresponds to name (null) with attr oversampling_ratio with attr index 0
The channel id: magn_x with i 0 corresponds to name (null) with attr raw with attr index 1
The channel id: magn_x with i 0 corresponds to name (null) with attr sampling_frequency with attr index 2
The channel id: magn_x with i 0 corresponds to name (null) with attr scale with attr index 3

The channel id: magn_y with i 1 corresponds to name (null) with attr oversampling_ratio with attr index 0
The channel id: magn_y with i 1 corresponds to name (null) with attr raw with attr index 1
The channel id: magn_y with i 1 corresponds to name (null) with attr sampling_frequency with attr index 2
The channel id: magn_y with i 1 corresponds to name (null) with attr scale with attr index 3

The channel id: magn_z with i 2 corresponds to name (null) with attr raw with attr index 0
The channel id: magn_z with i 2 corresponds to name (null) with attr oversampling_ratio with attr index 1
The channel id: magn_z with i 2 corresponds to name (null) with attr sampling_frequency with attr index 2
The channel id: magn_z with i 2 corresponds to name (null) with attr scale with attr index 3
```

Figure 5.2: The channels and attributes of the Magnetometer

If the sensor would be used on another board it can be found immediately using the name. If one would use another sensor within PX4 only the name has to be swapped and the correct channels and attributes should be chosen correctly.

### 5.2.2 Publishing the data in PX4

PX4 communicates via modules via the so called UORB protocol [48]. It is therefore important that the data that is being read via the IIO standard is also being broadcast on the network. For this PX4 has a special structure. It supports various sensors that do exactly the same it is designed in such a way that one can use special library functions to publish the data. These library functions will then calculate the derived parameters necessary for the drone platform. For instance, the integrated values and derivation. These functions then handle publishing the data on the network. This can be seen as a form of abstraction and prevents the developer from having to figure out all of the complex parts. It also ensures that not all sensors contain the same code.

## 5.3 PID tuning

A PID controller [109] is the standard controller used within drones, it controls the roll, pitch, and yaw of the quadcopter. Each of these has their own PID controller which have to be tuned separately.

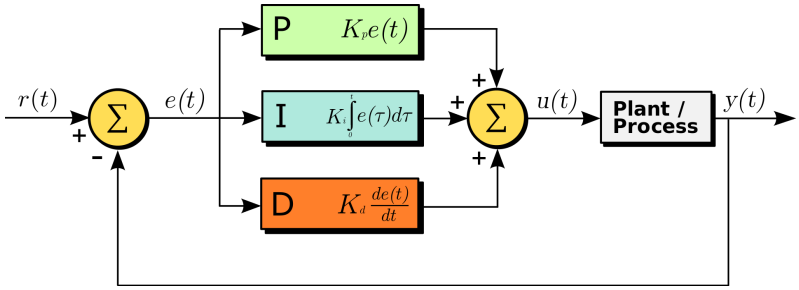


Figure 5.3: Default PID controller [13]

Each of these controllers have their own P, I, and D component as can be observed from Figure 5.3 hence the name PID which can be tuned individually.

- The P term is used for the present error, the further the distance between the current value and the set-point. The bigger the difference will be and the more the P term will contribute to push towards the set-point. In a drone changing the P can be thought of as the sensitivity and responsiveness of the drone. A higher P means sharper control, however, if the P is too high it will over-correct and cause overshoots.
- The I term is used for the past error, this is done by using the integral over the incoming data. It ensures that past errors are counteracted. In a drone this can be thought of as the stiffness of the drone, it prevents the drone from drifting away. If the I is too high the quadcopter will become stiff and unresponsive.
- The D term is used for future error, this is done by calculating the derivative over the incoming data. This way the rate of change of error can be calculated and brought to zero. It aims to flatten the error trajectory into a horizontal line and so reduces the overshoot. If the D is too high it will potentially amplify the noise and in case of a drone causes vibration.

From Figure 5.4 it can be observed that for the first takeoff there is a big problem. There is a big oscillation in the roll, which eventually causes the quadcopter to crash. This means that the P term is too high. It is clear that the PID needs to be tuned in order to work correctly.

### 5.3.1 Methods found in literature

One of the most known methods/heuristic is the Ziegler and Nichols method [110]. It is based on experimentally determining the point of marginal stability. The proportional gain of the controller is increased until the process becomes marginally stable. A disadvantage of the continuous cycling method is that the system is driven towards instability. This can lead to dangerous situations.

An improvement of the Ziegler and Nichols method is given by *Åström and Hägglund* [111]. They propose to use a relay feedback, because of this the system should no longer be driven towards instability.

### 5.3.2 Methods in drones

PX4 has its own PX4 PID tuning guide [112]. It starts with the precondition that the vehicle already flies and by default the gains are set to low values. The gains should slowly be increased by 20-30% per iteration. Too large gains may cause dangerous oscillations. In our case where the vehicle does not fly at all

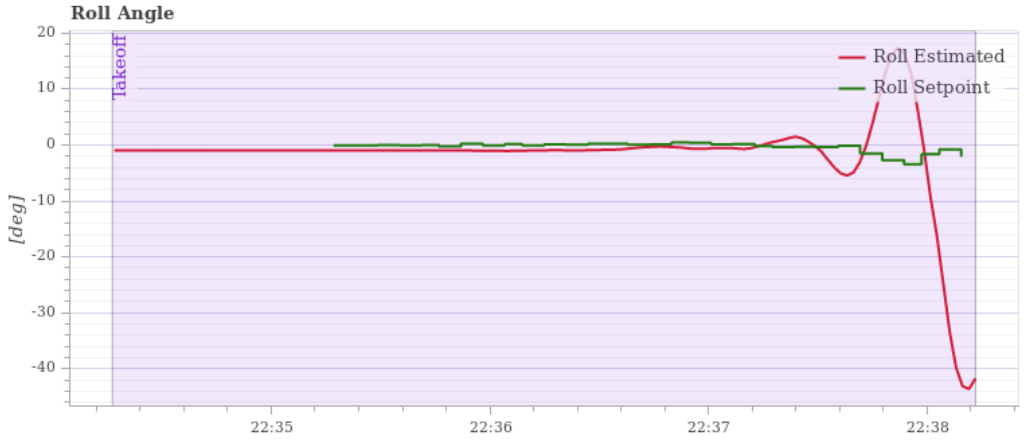


Figure 5.4: The roll angle during first takeoff


Figure 5.5: Roll rate after PID tuning

the P and D gains should be decreased until takeoff.

For drones enthusiasts there are multiple sites available to read about drones and discuss with fellow enthusiasts. `OscarLiang.com` is one of these sites which has an explanation of the quadcopter PID [113] which explains how to tune the PID. It advises to divide the default values by two and then increase the values until undesired behaviour is observed. First the roll then the pitch and finally the yaw should be tuned. For each of these first the P should be tuned, then the D should be tuned and finally the I should be tuned. There is a possibility that there needs to be switched back and forth to fine tune each value because they influence each other.

The decision was made to use the method of the PX4 tuning guide in collaboration with the information provided by `oscarliang.com` because the PID consists of multiple stages. The results after the tuning of the PID can be observed in Figure 5.5. There is no longer a big oscillation in the roll and crash during the takeoff, there will still be a small oscillation but this oscillation will slowly converge to 0.

## 5.4 Obstacle avoidance

### 5.4.1 Reading the sensor in ROS

The sensor used for the project is the RPLidar S1. A big benefit of this sensor besides it being lightweight and having low energy consumption is the serial interface which it uses to communicate the data. Within the FPGA it is easy to add an extra serial interface and utilize this. This can then directly be used within the special RPLidar ROS node. The sensor is connected via the pins on the same header as the motors and receives its power from the Battery Elimination Circuit (BEC) of the ESCs. The BEC can provide up to 3 Ampère and is capable of handling the RPLidar S1 during startup which requires a maximum of 1.5 Ampère. The results from the ROS node can be observed in Figure 5.6. Some of the structures like the walls of the building can be observed. The wall on the far right is approximately 25 meters and is still detected correctly as expected.

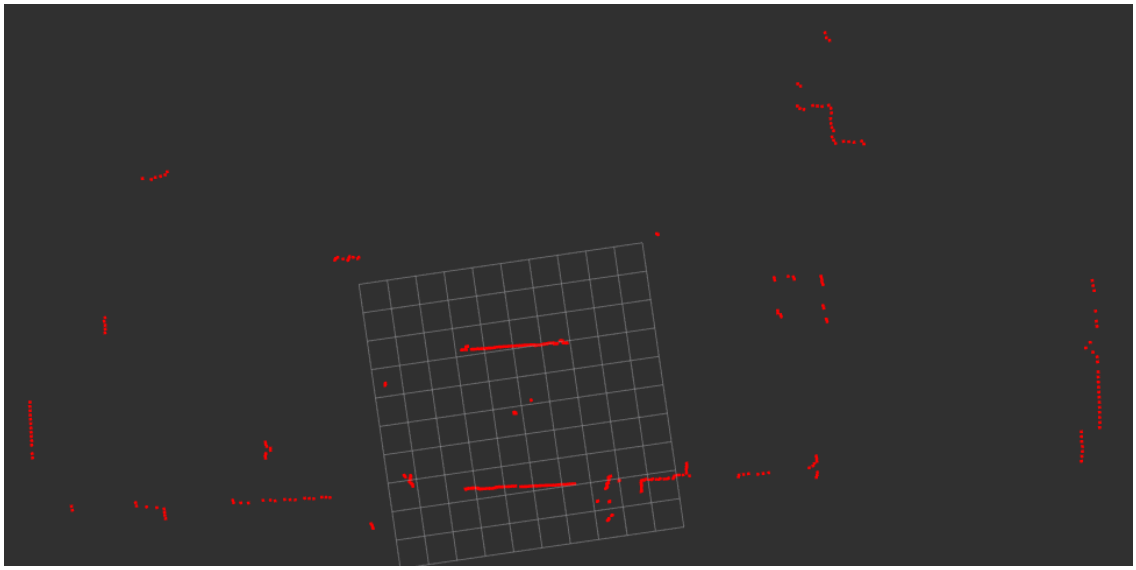

Figure 5.6: Output of RPLidar in ROS

### 5.4.2 Converting the data

As mentioned in Section 4.4.2, PX4 requires a point cloud and not a laser scan, which is the output of the RPLidar ROS node. This thus has to be converted, for this a special ROS package is used, vigir_lidar_proc. This contains a laser scan to point cloud node which converts reads the message on a specific ROS topic and publishes it as a point cloud. Using this the data can be used by PX4 for avoidance.

### 5.4.3 Connection

The connection between PX4 and ROS is done via mavros as mentioned in Section 4.4.1. For test purposes all the avoidance software is installed on a companion computer. This way tests can be done and visualized easily on the system. The connection between PX4 and ROS via mavros is established and the avoidance system is started. Unfortunately, the working of the obstacle avoidance is not tested.

# Chapter 6

# Experiments & Results

This chapter contains the experiments and results. Most of the experiments have been conducted indoors unless stated otherwise. The indoor tests involving flight were conducted within the Topic Gym which can be observed in Figure 6.1. Unfortunately the space within the gym is limited and there is no GPS reception within the building. The GPS has been mocked. This results in it always reporting the same location. This results in it being less reliable and requiring to do advanced flying tests outdoors.

Figure 6.1: The Topic gym

## 6.1 Testing of the sensors

There are several sensors on the XDP platform which should be interpreted correctly by PX4. These consist of the IMU, magnetosensor, and environmental sensor. As specified before drivers have to be written in order to use this data in PX4. It is also important that this data is correctly interpreted with the correct units.

### 6.1.1 Testing the barometer

The barometer is of importance to determine the height, especially in the indoor testing environment where GPS cannot be used. The barometer uses the barometric pressure to determine its height, the lower the pressure the higher the drone is located. To test the barometric pressure sensor an outdoor emergency staircase will be used. It is expected that there will be an increase of approximately 10 meters (the height of the staircase), and when returned back it should be at approximately the original height.

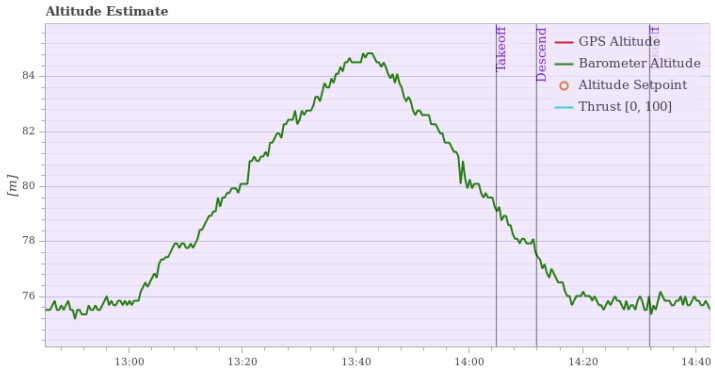


Figure 6.2: The barometric sensor data

From Figure 6.2 it can be observed that the barometer behaved as expected. The barometer altitude increases, at a certain point reaches the top and then once again goes down. One can observe that the starting height is already at 76 meter. This is because the barometric pressure is not constant for an area because of weather influences. This can be adjusted, however, is not necessary since during startup it will be determined what the offset is.

Additionally, the barometric pressure could be influenced by the air displacement of the drone. It is important that this is tested in order to assure the correct working during flying. It is expected that due to a distance of circa 5 centimeters between the barometric pressure sensor and the propellers this will not be influenced. In *E. Kuantama et al.* [114] it can be observed that the airflow is only influenced close by the propeller. In order to test this the drone will be strapped to the ground and will try to takeoff to simulate the real world scenario.


Figure 6.3: Barometric sensor data during takeoff strapped to the ground

Figure 6.3 shows the result of the test. It can be observed there is a small time where during takeoff on average it will be 15 centimeters higher than without running rotors. It seems there is a difference. However, this deviation is only really small and could be caused by the fact that the drone is strapped to the ground. This behaviour can be observed in Figure 6.4 and is called the ground effect. The airflow is directly influencing the airflow above the platform and can thus have an impact on the air pressure.

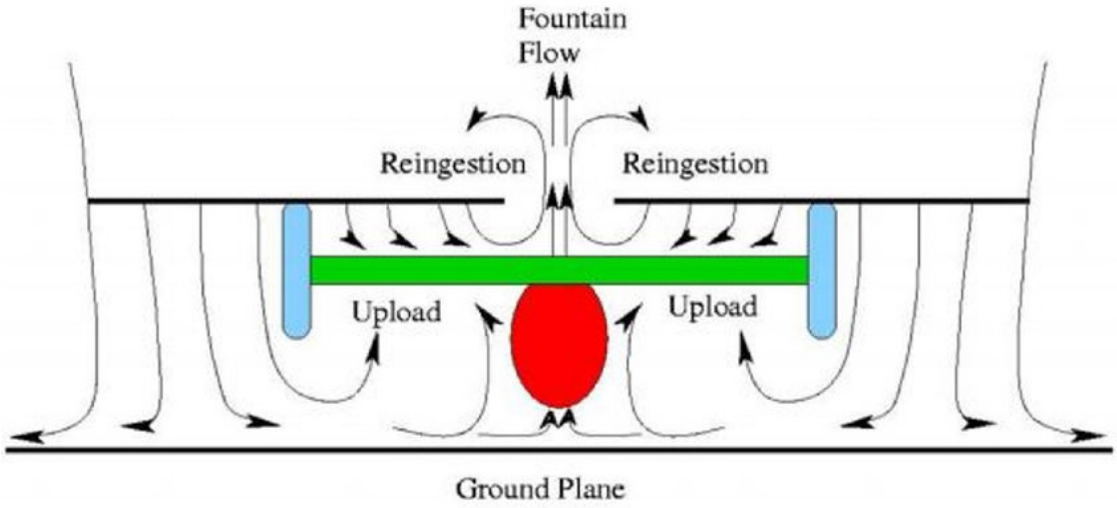


Figure 6.4: Airflow of a quadcopter with the ground effect [14]

In order to confirm if it is really caused by the ground effect the quadcopter is tested at an elevation. This causes the ground effect to no longer occur and it is expected that there will be no deviation in air pressure.

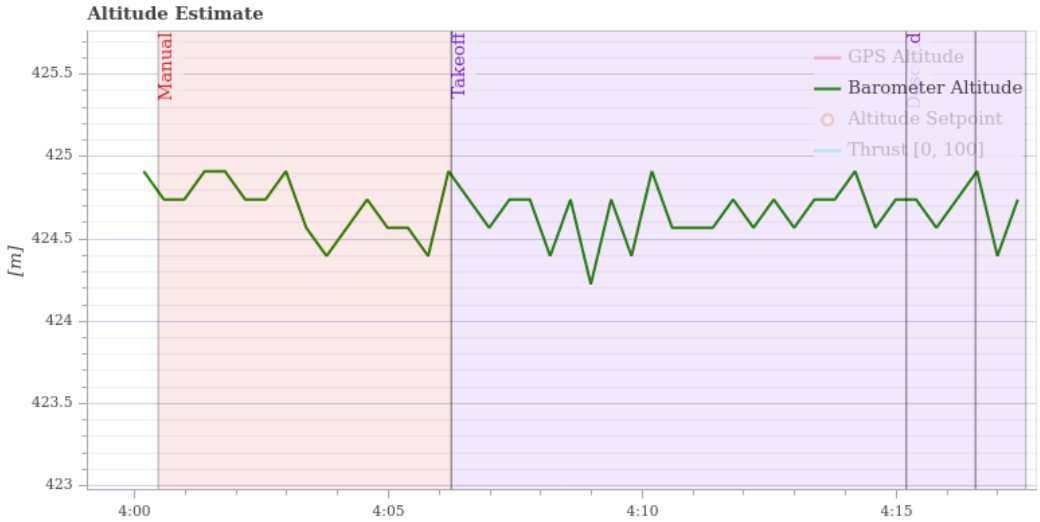


Figure 6.5: Barometric sensor data during takeoff strapped elevated

As can be observed in Figure 6.5 there is no longer a deviation during takeoff. This confirms the expectations and the ground effect is indeed the cause for the deviation. This means that during takeoff there will be a small deviation in estimated altitude but during flight this will no longer occur.

### 6.1.2 Testing the Magnetometer

The Magnetometer is of importance to determine the direction in which the aircraft is flying. It uses the earth magnetic field to determine what its relative position to north is. The earth magnetic field is really weak (25 to 65 microteslas [115]). The common case within the Netherlands is around 50 microtesla [116]. It is therefore possible to experience influence from electricity flowing on the pcb or electricity flowing through cables. Whether it can be influenced can be calculated using Ampère's law:

$$B = \frac{\mu_0 * I}{2 * \pi * r} \tag{6.1}$$

$\mu_0$ is the permeability of free space which is equivalent to $4.7 * 10^{(-7)} T * m/a$ , I is the current in Ampère and r is the distance from the wire in meters. To see the amperage needed per distance the Equation 6.1 can be rewritten to:

$$I = B * \frac{2 * \pi * r}{\mu_0} \tag{6.2}$$

If then the different cases for earth magnetic field would be used for the calculation a plot can be made.

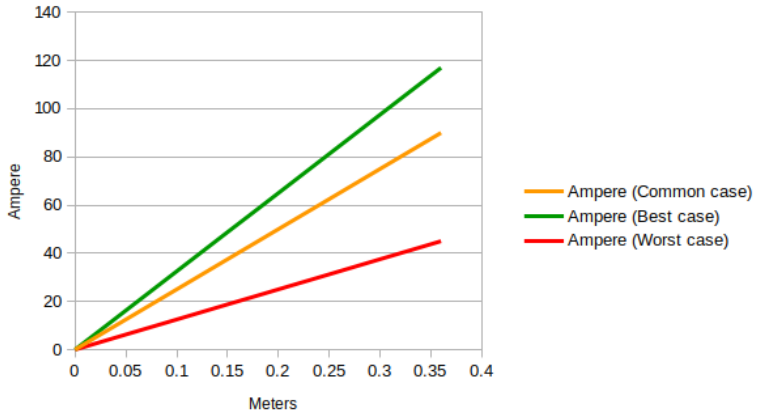


Figure 6.6: Amperage needed to influence magnetic field at certain distance

As can be observed from Figure 6.6 in best case when 40 Ampère is flowing at approximately 12 centimeters the magnetometer will be influenced. At 100 percent thrust one motor draws 13.4 Ampère and thus in theory the 4 motors combined can draw a total of up to $4 * 13.4 = 53.6$ Ampère (only for the motors). In best case this evaluates to approximately 16 centimeters and in common case 21 centimeters, in worst case conditions it would be 43 centimeters.

In the current setup the distance between the sensor is 16 cm in length and 8 centimeters in height. Using the Pythagorean theorem [117] this evaluates to $16^2 + 8^2 = 320$ and thus the distance is approximately 17.89 centimeters. In best case this would be sufficient, however, in the common case, this would not be sufficient. If the cables would be lowered by 6 centimeters in theory the common case would no longer be influenced, unfortunately, this is not possible. In order to test whether this is possible a small test was conducted in which the drone was strapped to the ground and the motors were turned at full strength. Then the raw values for the magnetic field were plotted. With the results found from the calculations it is expected that there will be an influence on the raw magnetic field when the motors are turned on.

From Figure 6.7 it can be observed that what was expected is indeed correct. When the motors are turned on for takeoff at approximately 9:51 all the raw magnetic field strength values are increased and thus influence the reading.

The next question that should be asked is how is this normally solved within the drone industry? Industry solves this by mounting the magnetometer as far as possible from the flight controller. For DJI this can be observed in the fact that the flight controller has an external GPS-Compass Pro module which

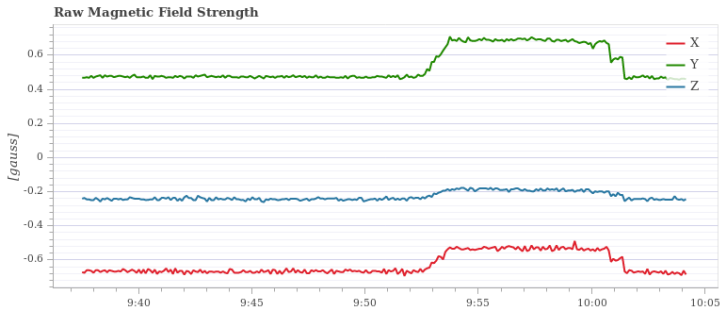


Figure 6.7: Raw magnetic field

contains the compass [118]. For PX4 the recommendation is given to mount the combined GPS + compass as for away from the motor/ESC power supply lines as possible [119]. Within the literature not much information can be found on an internal magnetometer, only on the use of an external magnetometer as in *Endrowednes K. et al.* [120].

When the motors are turned off however, it is expected that the magnetometer will accurately report the degrees that the board is heading. In order to test this the board is turned approximately 90 degrees and it is checked whether this value is reflected within PX4. To prevent any external interference the test has been conducted in a remote area. An analog compass is used to verify north and that there is no external interference. Figure 6.8 shows the setup.

These tests show that the value shown in PX4 is not accurate as can be observed in Table 6.1. It can be observed that for south and west the values are still approximately accurate. However, North, and East are way off showing that there is likely a problem with the board.

Table 6.1: Results from magnetometer experiment

| **Direction of board** | **Expected value** | **Reported value** |
|---|---|---|
| Up (South) | 178 | 177 |
| Right (West) | 264 | 252 |
| Down (North) | 356 | 303 |
| Left (East) | 87 | 18 |

This has two possibilities, either there is a problem in PX4 or a problem with the magnetometer. To verify what is the problem the magnetometer will be read directly. If this data is not accurate it is not going to be accurate in PX4 either.

When reading the data directly from the sensor this data is the same as in Table 6.1.

It should be investigated why the sensor does not give the expected reading, whether it is related to the sensor or to the board. However, this will likely cost a lot of valuable time, additionally, there is an alternative available. The use of the IMU is sufficient to determine the direction of the drone. With the use of GPS this can be made more redundant.

The tests with the IMU have been conducted in the same was as the tests for the magnetometer have been conducted and are discussed in Section 6.1.3. The result however when using only the IMU is better than with the IMU and magnetometer combined. Therefore there has been chosen to not use the magnetometer. If there was more time available it could be researched why exactly the magnetometer is not working properly.

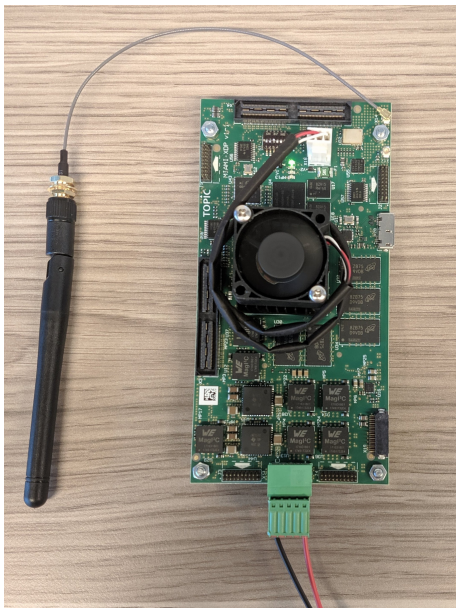

Figure 6.8: Start orientation up

### 6.1.3 Testing the IMU

The IMU is the combination of an Accelerometer and a Gyroscope within the same package, combining this gives the Inertial Measurement Unit. The data of the IMU is of importance to determine the roll, pitch, and yaw of the drone and the position of the drone. The sensors will be tested by moving the drone over its roll pitch and yaw manually and then verifying these results via the logs. It is expected that this will work smoothly. The magnetometer has already been disabled for this test and will thus not have any influence on the data received via the logs.

The first test is the pitch of the drone, the pitch is the rotation over the y axis and can be thought of as leaning forward or backward. For the test there will first be pitched forward and than pitched backwards at approximately 60 degrees. It is expected that this can be observed from the graphs and that in the end the pitch will be returned to approximately 0 when the test is done.


Figure 6.9: Pitch Angle

Figure 6.9 shows the results from the test. It can be observed that first there is a big spike in negative pitch to approximately 60 degrees and then a spike in positive pitch to approximately 55 degrees. After

these spikes it returns to 0 and thus confirms that the expectations are correct.

The second test is the roll of the drone, the roll is the rotation over the x axis and can be thought of as leaning left or right at approximately 60 degrees. For the test there will first be roll to the left and then roll to the right. It is expected that this can be observed from the graphs and that in the end the roll will be returned to approximately 0 when the test is done.

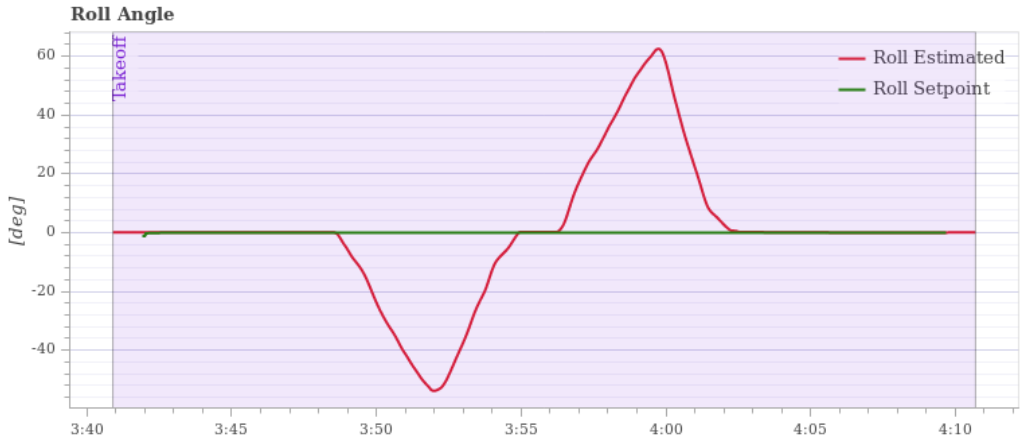


Figure 6.10: Roll Angle

Figure 6.10 shows the results from the test. It can be observed that first there is a big spike in negative roll to approximately 55 degrees and then a spike in positive roll to approximately 60 degrees. After which it returns to 0 and thus confirms that the expectations are correct.

The final test is the yaw of the drone, the yaw is the rotation over the z axis and can be thought as simply turning it left and right. This test is a bit more extensive, the drone will be turned 180 degrees, then in the original position, then 270 degrees and finally it will be returned in the original position. It is expected that this behaviour can be observed from the graph and that the begin position is almost equivalent to the final position.

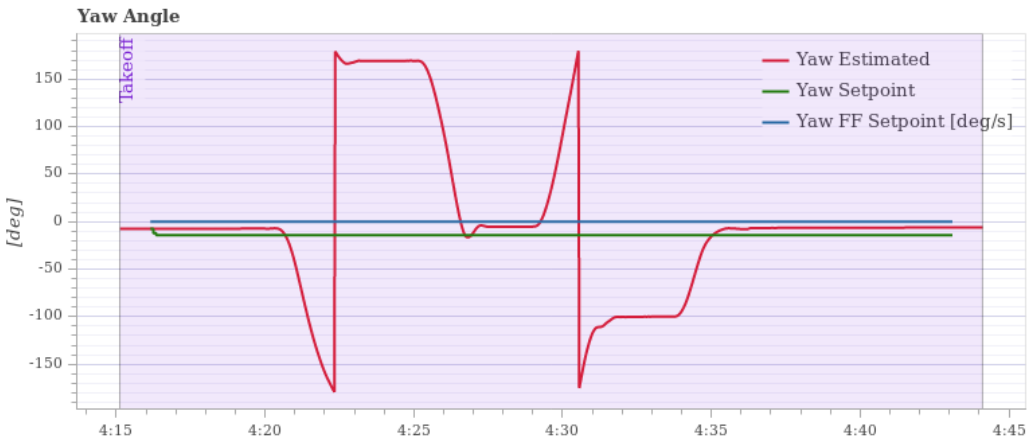


Figure 6.11: Yaw Angle

Figure 6.11 shows the results from the test. It can be observed that after turning the drone for approximately 180 degrees there is a big spike into the other direction. This is caused by the angle of the drone will always be within -180 degrees or 180 degrees and this is thus reflected in the graph. Then the graph goes back to approximately the start position and then it turns 270 degrees to finally go back to the starting position again. This confirms that the expectations are correct.

### 6.1.4 Testing the GPS

The GPS is of importance to determine the location of the drone. The sensors on the drone will have an estimation error which is corrected by the GPS. Additionally the GPS can also be used to determine the current height and can be fused with the barometer for a more accurate height estimation. The GPS will be tested by taking a walk around the perimeter outside. The whole test will be done outside close to the building. It is expected that the GPS will follow the trajectory of the path taken.

Figure 6.12: Purple: Gps trajectory, Red: Real trajectory approximated

From Figure 6.12 it can be observed that the GPS trajectory does not always nicely overlap with the real trajectory. When taking the data from Figure 6.13 into consideration it can be observed that for certain area's there are big spikes in the position accuracy and the number of satellites used drops. This is caused by to the fact of being close to the building at a low height and thus losing a big line of sight. This can be prevented by elevating to higher height or staying further away from buildings. When the GPS is tested in a more open environment as in Figure 6.14 and Figure 6.15 it can be observed that the GPS will be very accurate over the entire path.

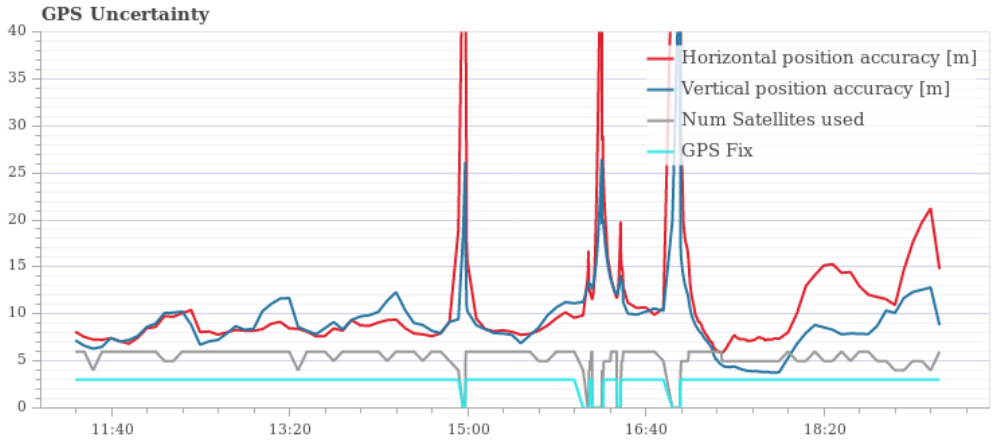


Figure 6.13: Statistics for GPS uncertainty near building

Figure 6.14: GPS trajectory open environment, purple: GPS trajectory, Read: Real trajectory approximated

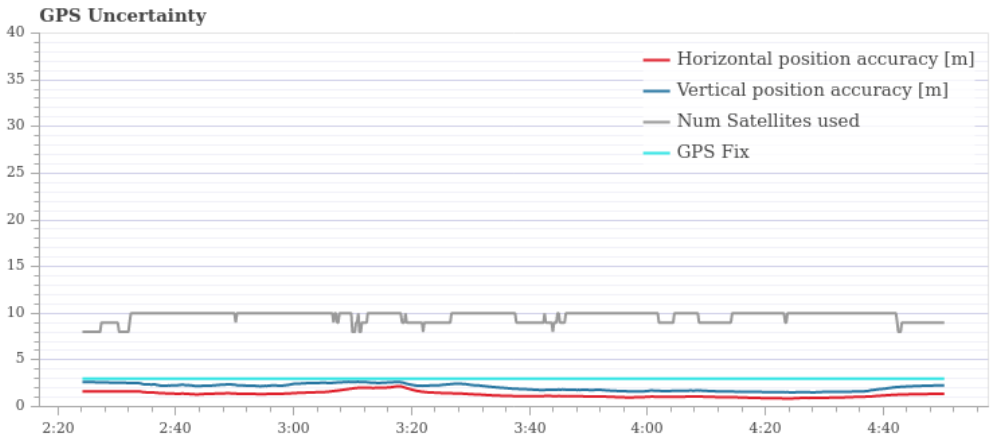


Figure 6.15: Statistics for GPS uncertainty in open environment

## 6.2 Testing of the motors

The layout of the motors can be observed in Figure 6.16. The numbers given by Figure 6.16 will be used within this section and in the logger. The log data will be used to verify whether the experiments are as expected, it has been verified that the data being logged is the correct data being send to the motors.

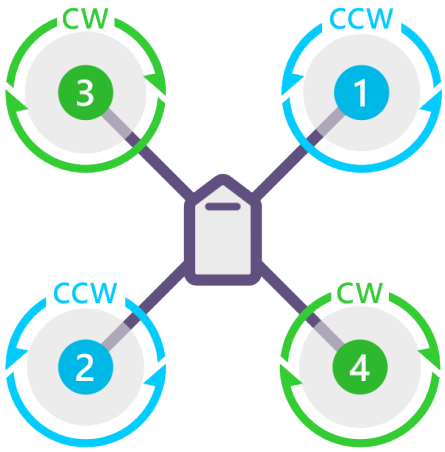


Figure 6.16: Quadcopter layout [15]

A small remark that has to be taken into consideration is that motor 1 corresponds to output 0, motor 2 corresponds to output 1, etc. Additionally, the value of the motors is displayed between -1 and 1 where -1 is the minimum throttle and 1 is full throttle.

### 6.2.1 Arming and Calibration

When the motors are disarmed and go to arming it is expected that the output will rise by a small margin. When the motors calibrate the output should be the maximum output possible, this will be the case for all the motors equally.

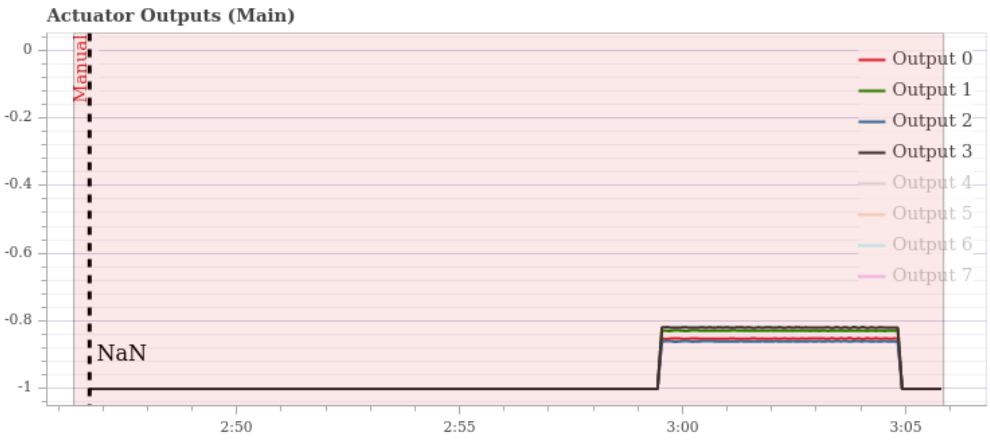


Figure 6.17: Arming of the motors

From Figure 6.17 and Figure 6.18 it can be concluded that what was expected is indeed correct. All the motors rise simultaneously. It can be observed that when arming the output rises a bit to a maximum of -0.8 and for the calibration the output rises to 1 (the maximum).

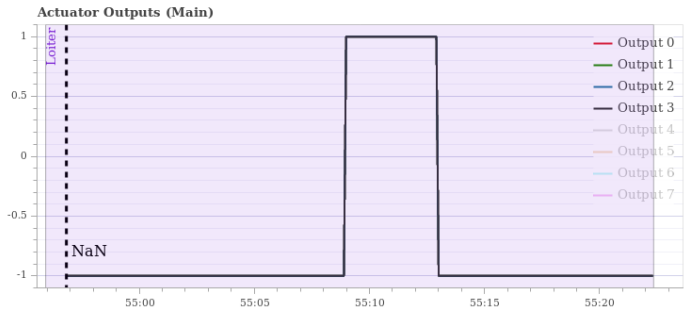


Figure 6.18: The calibration of the motors

### 6.2.2 Tilt forwards and backwards

When tilting forwards it is expected that motor 1 and 3 will increase their thrust while 2 and 4 will decrease their thrust in order to stabilize again. When tilting backwards it is expected that 2 and 4 will increase their thrust while 1 and 3 will decrease their thrust.

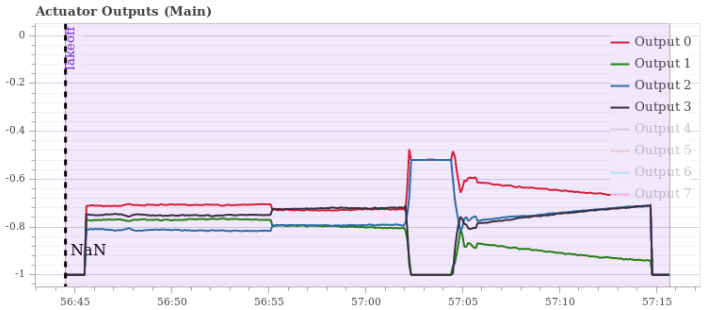


Figure 6.19: Tilting forwards (lifting the back)

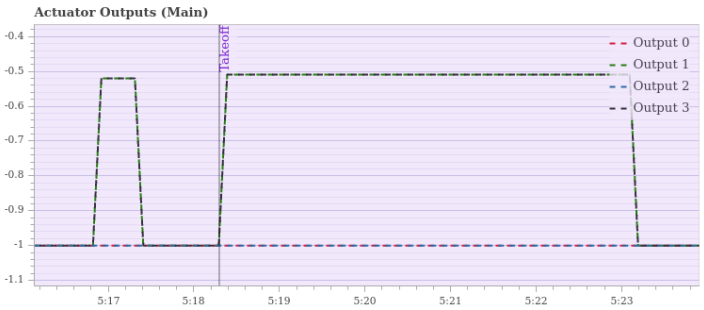


Figure 6.20: Tilting backwards (lifting the front)

These Figures are a bit more difficult to interpret, because of the takeoff being engaged the output is no longer -1 but tries to correct the board in most situations. This causes some other difficulties where fail safes are being engaged since the system does not react as expected. The behaviour of tilting forwards can be observed in Figure 6.19. Output 0 and output 2 will higher their values and output 1 and 3 will lower their value which correspond to the expected behaviour. The behaviour of tilting backwards can be observed in Figure 6.20. Output 1 and output 3 are high while output 0 and output 2 are low which corresponds to the correct motors.

### 6.2.3 Tilt left and right

When the board is tilted to the left it is expected that motor 2 and 3 will increase their thrust while motor 1 and 4 will decrease their thrust. When tilting to the right it is expected that motor 1 and 4 will increase their thrust while motor 2 and 3 will decrease their thrust.

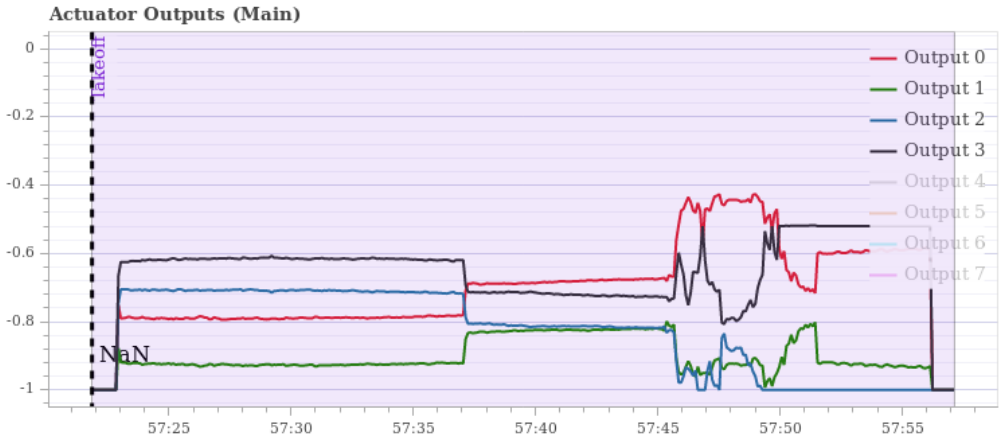


Figure 6.21: Tilting right (lifting the left)

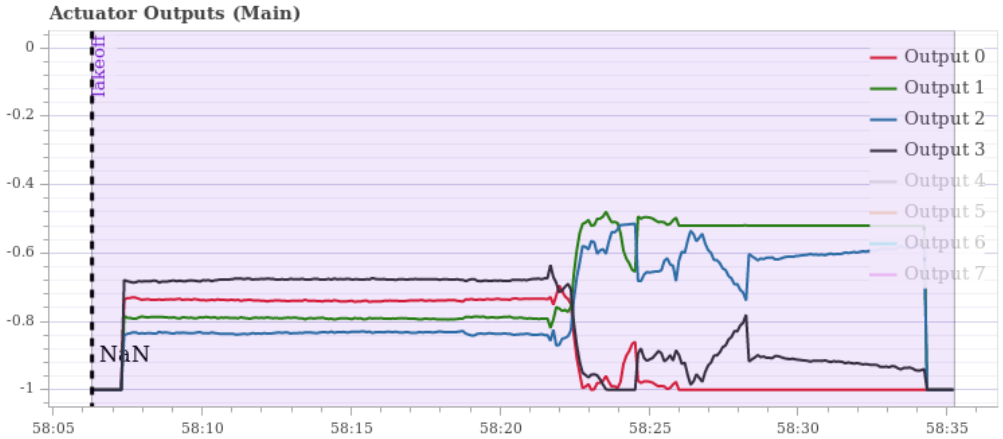


Figure 6.22: Tilting left (lifting the right)

From Figure 6.21 and Figure 6.22 we can observe that what we expected is correct. When tilting right the thrust is increased on motor 1 and motor 4 while the thrust is decreased for motor 2 and 3. When tilting left the thrust is increased for motor 2 and 3 while the thrust is decreased for motor 1 and 4.

## 6.3 Takeoff

It is expected that when the takeoff command is being send to the drone platform the drone will takeoff to the desired height and hover at this position. The height to which it takes off can be given by a parameter and it is expected that the drone will go to a close approximation of this height. The takeoff should be stable and it should not move to much in another direction during takeoff.

The first takeoff did not go as expected as observed in Figure 6.23. There is a large oscillation in the roll resulting in a crash, in order to prevent this behaviour the internal PID control loops have to be tuned.

After several iterations tuning the PID parameters the end result of Figure 6.24 was achieved.

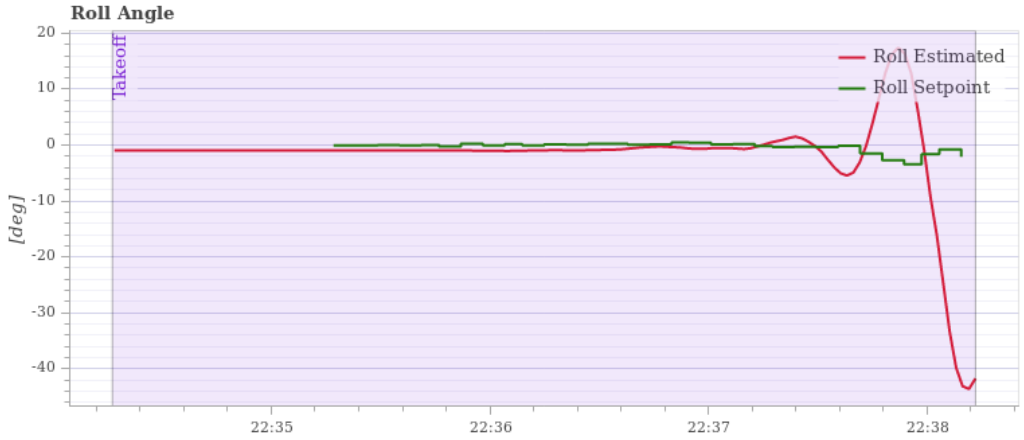


Figure 6.23: The roll angle during first takeoff

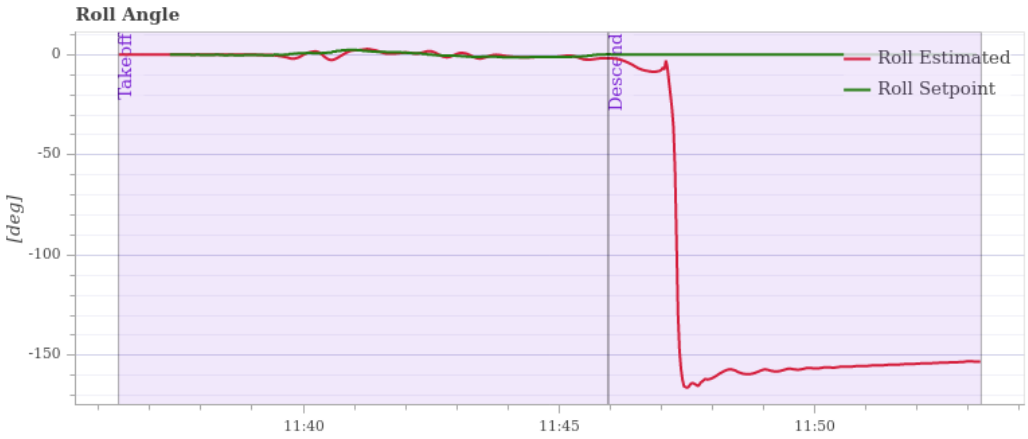


Figure 6.24: The roll angle after PID tuning

As can be observed in Figure 6.24 the roll angle is an almost flat line. It can be observed that in the end there is a huge spike. This is because the drone crashed and therefore could no longer compensate its roll. The behaviour of the system is now as expected. The drone will takeoff to approximately the given height and hover.

## 6.4 The system under load

The decision has been made to implement everything on the A53 cores instead of the real-time cores with a RTOS. The disadvantage to this is that there will be more load on the A53 cores since both ROS and PX4 are running on these. Additionally the behaviour is less deterministic compared to when running on an RTOS. And finally the biggest factor is that there is a possibility that if the A53 cores are under high load that PX4 will hang. In order to test this a small test has been conducted.

To simulate load on the CPU a small script is run which executes the following command eight times: CAT /DEV/URANDOM > /DEV/NULL, this causes the CPU to get random data from /DEV/URANDOM and write it to /DEV/NULL and create a big load on the system.

This is executed eight times in order to run the program on all cores.

It is expected that because of the load the system will react less smooth which results in the flying being less accurate Another possible behaviour could be that safety triggers are triggered because sensors

not reaching their expected update rates.

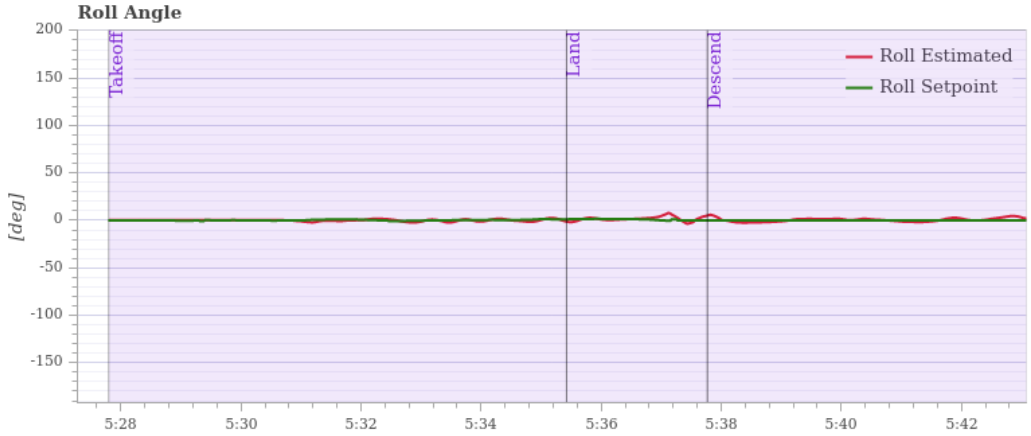


Figure 6.25: The roll rate under load

After some tests and looking at the log data as can be observed in Figure 6.25 the behaviour did not change, the roll rate is still stable during flight.

A possible explanation for this behaviour is that PX4 is written according to the reactive manifesto [121]. Reactive systems are responsive, resilient, elastic, and message driven. Elastic means that the system stays responsive under varying workload. "Reactive Systems can react to changes in the input rate by increasing or decreasing the resources allocated to service these inputs", which is happening in this situation. Even though the load of the system is very high PX4 still stays responsive.

## 6.5 First test flight with GPS

All the individual aspects of the system have been tested. This means that in theory everything should work. It is therefore expected that when the takeoff command is send to the drone that the drone will start to takeoff and hover at roughly the specified height. After this a certain location will be send to the drone and it is expected that it will travel to this location.

The test, however, did not go as expected, the drone kept elevating after which it was manually killed. It fell from approximately 4 meter high destroying the chassis of the drone. During indoor tests it has been observed that the drone is able to takeoff to the correct height and then hover at this height, however, in the outdoor test it failed at this step, so something went wrong.

Figure 6.26 shows a part of the possible explanation. When looking at the fused altitude estimation it can be observed that it barely goes up although the barometers altitude goes up by 5 meters. Additionally it can be observed that the GPS altitude is a perfect straight line. It is expected to contain some noise as can be observed in Figure 6.27.

It seems like the altitude estimate from the GPS is not accurate. Looking into the data of the GPS as observed in Figure 6.28 it becomes much more clear what went wrong. It can be observed that at timestamp 11:37 there is the last update of the GPS after 10 seconds at timestamp 11:47. There are no satellites found and thus the data is no longer valid. The data between these points is interpolated.

What happened is that the barometer registered the elevation and the GPS did not. However, the influence of the GPS is higher since GPS is not affected by possible external factors like weather and temperature.

In order to prevent this from happening it is important to know what exactly caused this. Either something went wrong in the software or the GPS was not physically working anymore, however, in Section 6.1.4 the GPS did work. A possible explanation could be that the antenna broke of during the test flight. In order to test whether this could be correct a small simple test will be conducted. The drone will

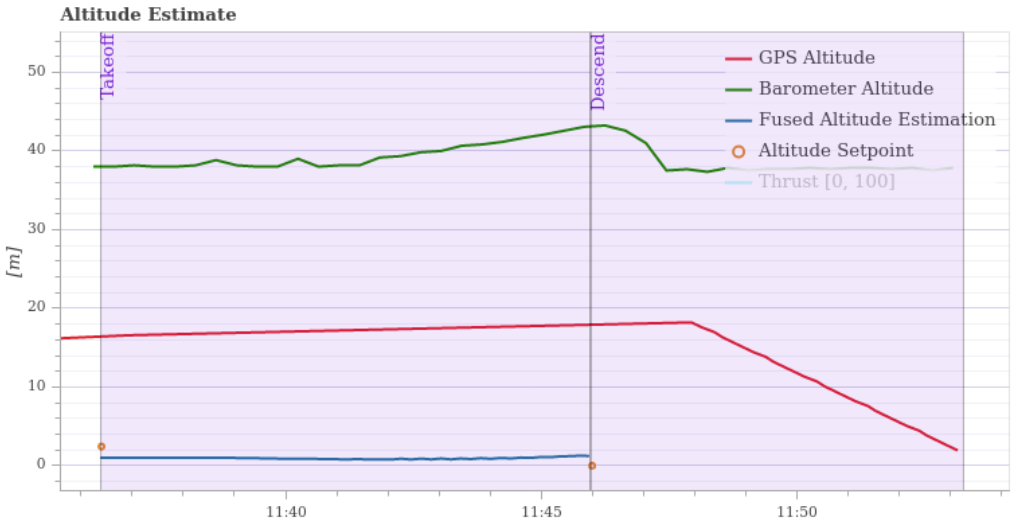


Figure 6.26: Altitude estimate during first test flight

try to takeoff without its rotors connected and the GPS will be disconnected. It will be manually elevated and checked whether the same behaviour can be observed.

The test has been conducted again near Topic. The drone was told to takeoff without any motors spinning and then the GPS antenna was manually disconnected. However, the behaviour is different as can be observed in Figure 6.29 and Figure 6.30. At approximately 3:00 the GPS has been disconnected and immediately the horizontal and vertical position accuracy rises. Compared to Figure 6.28 where the horizontal and vertical position accuracy only increases after a certain amount of time the system reacted differently. Another interesting remark that can be observed and not mentioned before is that the accuracy during the test flight in Figure 6.28 is really high. The accuracy difference can be observed compared to the GPS trajectory test in Figure 6.15.

Prior to the takeoff a warning was given by the Local position estimator and GPS module. These messages can be observed in Figure 6.31. As observed by the timestamps these messages occurred much earlier than the actual flying took place but these could still have an influence. The GPS timeout warning happens after the data from the GPS has not been updated for 1 second. This could already mean that there was a problem with the GPS. The other message states there is an invalid GPS message received with a payload of 46556 bytes. Normally it expects a message with a payload of 92 bytes. This would evaluate to approximately 506 messages from the GPS being interpreted as 1 message and could be an explanation why the GPS timeout warning was displayed.

A possible scenario that might have happened during the test flight is that the GPS was already providing incorrect information during takeoff. After it crashed the GPS antenna broke and the horizontal and vertical position accuracy rise as can be observed in Figure 6.28. Which would explain why it is the same behaviour as in Figure 6.29. The simple way to prevent this is looking more careful at the warning since it does not happen all the time. During the GPS test on the parking lot the warning did not occur and the GPS worked all the time. Unfortunately, it is not found possible to reproduce the behaviour that happened during the first test flight.

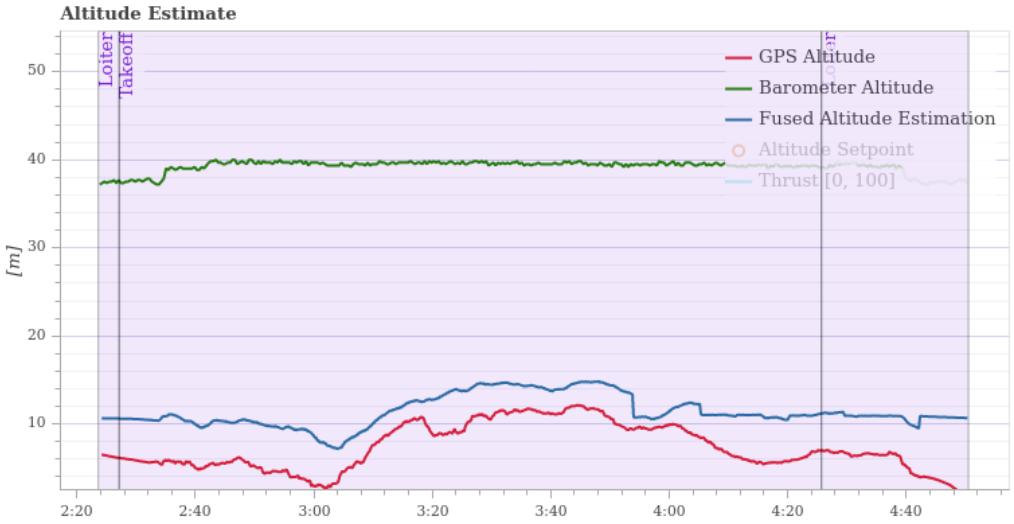


Figure 6.27: Altitude estimate from the GPS test of section 6.1.4

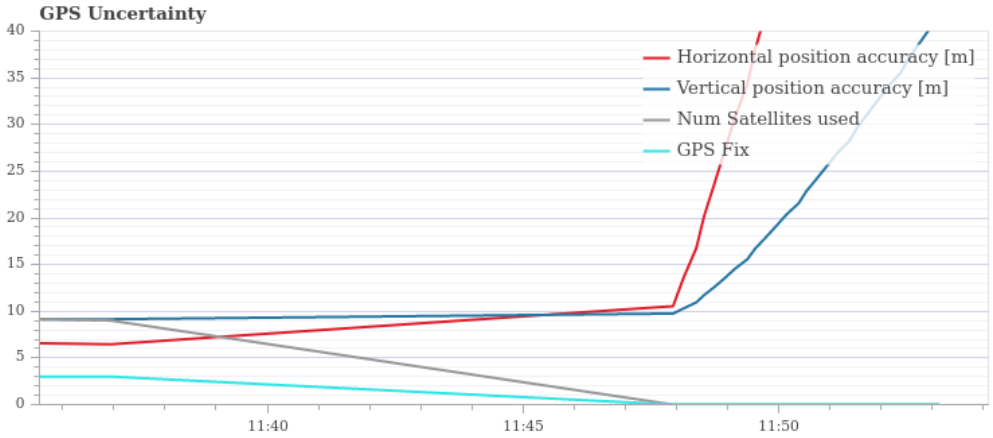


Figure 6.28: GPS uncertainty during test flight

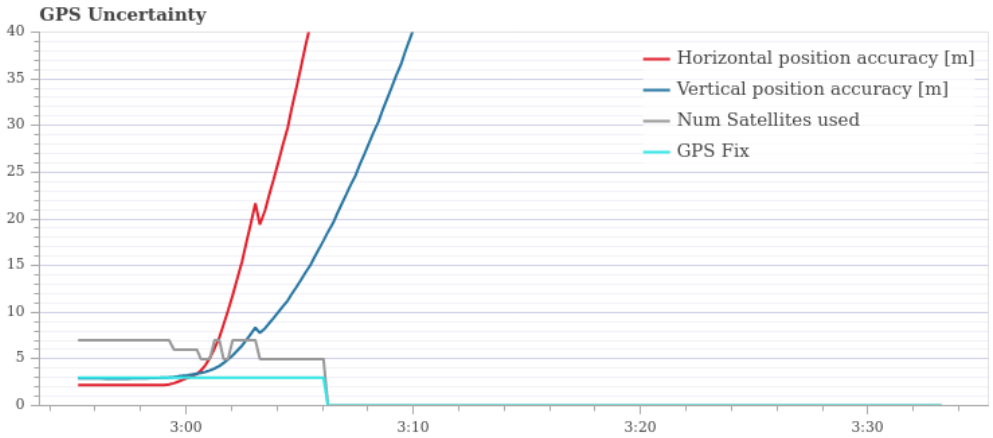


Figure 6.29: GPS uncertainty during test near topic

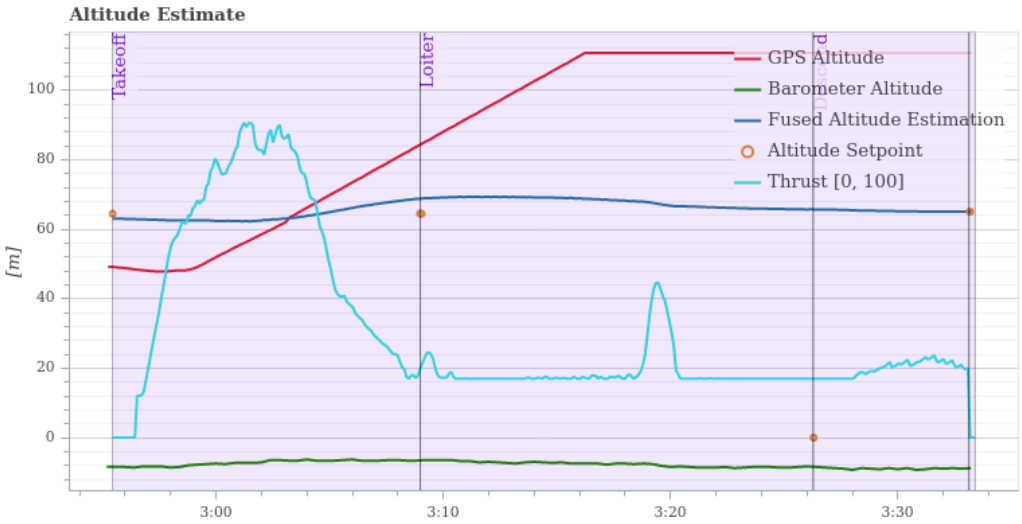


Figure 6.30: altitude estimate during test near topic

| # | Time | Level | Message |
|---|---|---|---|
| 0 | 0:05:04 | WARNING | [local_position_estimator] [lpe] GPS timeout |
| 1 | 0:08:22 | WARNING | [gps] ubx msg 0x0107 invalid len 46556 |

Figure 6.31: Log messages

# Chapter 7

# Conclusion

## 7.1 Conclusions

In the end, the goal of the project has been partially realized. The autopilot has been implemented and tested successfully. Unfortunately, not a lot of outdoor tests have been conducted to verify the correct working in different environments such as wind. All these functions have been tested in simulation and the autopilot has been extensively tested in an outdoor environment by the PX4 community. It has been proven that the sensor can be connected to the platform and the data can be read directly. The connection between PX4 and ROS has been established, however, the obstacle avoidance has not been tested.

On the MPSoC it is proved that the design flow used within the project is correct. First the drivers for the motors were implemented and tested. After that the sensors were implemented and the correct functioning of these sensors within PX4 was tested. Then tests with the autonomous takeoff were performed and lastly the obstacle avoidance sensor was incorporated. The sensor proved to be suitable for the drone platform. The obstacle avoidance was implemented but not tested.

The trade-offs for different mappings on the MPSoC have been investigated and considered. Mapping it all on the A53 cores was the correct choice for this project. Even at full load the system was still responsive and it saved implementation time.

Different sensors have been researched and the trade-offs for these sensors have been presented. Depending on what is exactly expected from the sensor, a different sensor can be chosen. For this project the RPLidar S1 is the most suitable choice for obstacle avoidance sensor.

PX4 was found as the software that offers all the necessary functionality and proved to be the correct decision. It is a very modular autopilot and offers the possibility for obstacle avoidance in conjunction with ROS.

The project had its fair share of challenges. As could be expected when open-source software is implemented without any prior knowledge and experience. The correct functioning of the open-source software proved to be difficult. The software runs in multiple threads and communicates over various topics making it difficult to debug. The tuning of the PID control loops within the autopilot showed to be difficult. This was extra difficult because it was immediately tested with the use of the automatic takeoff function of PX4 and not tested with a remote control. If a remote control would be used, tuning the different PID loops would have been easier. The incorrect working of the magnetometer on the platform caused some strange behaviours. These were not immediately identified as problems caused by the magnetometer. This caused extra time trying to debug and in the end the decision was made to not use the magnetometer for this project. Additionally, it was difficult to test the drone in an outdoor area. It is not legal to fly with a drone within 20 kilometers of Topic due to Eindhoven Airport.

It probably would have been better to start with a platform that already flies. This would require less time being spend on tuning the PID. This would also be a good example to compare the performance between the Topic platform and the default autopilot hardware.

## 7.2 Future works

The drone industry is still rapidly evolving and not much research has be done on fully autonomous drones. Hence, the following topics represents possible future works.

- Verify that the obstacle avoidance works correctly.
- Better utilize the FPGA, run parts of the autopilot on the FPGA e.g. PID control loops.
- Use multiple obstacle avoidance sensors and combine the data with sensor fusion.
- Add redundancy to the platform. Since drones will elevate to higher heights the chance for cosmic radiance effect (single-event upsets) will be higher. It would be interesting to research the benefits of the added redundancy within the platform.
- Utilize the FPGA by running for instance, video processing on the platform.
- Implement obstacle avoidance with radar and compare its performance to Lidar

# Bibliography


[1] Stereolabs, "Stereolabs zed." https://www.stereolabs.com/zed/, 2019. xixi, xiixii, 10, 65

[2] Aerotenna, "Aerotenna μsharp product overview." https://aerotenna.com/sensors/#usharp, 2019. xixi, xiixii, 10, 66

[3] Elecfreaks, *Ultrasonic Ranging Module HC-SR04*. https://cdn.sparkfun.com/assets/b/3/0/b/a/DGCH-RED_datasheet.pdf. xixi, xiixii, 11, 66

[4] Garmin Ltd., "Lidar-lite v3hp product page." https://buy.garmin.com/en-US/US/p/578152/pn/010-01722-10, 2019. xixi, xiixii, 11, 67

[5] Shanghai Slamtec , "Rplidar s1 product page." https://www.slamtec.com/en/Lidar/S1, 2019. xixi, xiixii, 11, 68

[6] Velodyne Lidar, "Velodyne lidar puck lite product page." https://velodynelidar.com/vlp-16-lite.html, 2019. xixi, xiixii, 12, 68

[7] R. Mackay, H. Willee, C. Elder, and A. Trigdell, "Connect escs and motors." http://ardupilot.org/copter/docs/connect-escs-and-motors.html, January 2019. xixi, 15

[8] Digikey, "Xilinx authorized distributor." https://www.digikey.co.uk/en/supplier-centers/x/xilinx, 2019. xixi, 17

[9] H. Willee and B. Kung, "Px4 architectural overview." https://dev.px4.io/en/concept/architecture.html, February 2018. xixi, 19

[10] H. Willee, M. Bresciani, and R. Bapst, "Controller diagrams." https://dev.px4.io/en/flight_stack/controller_diagrams.html, May 2019. xixi, 21

[11] Xilinx Inc., "Zynq ultrascale+ mpsoc." https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html, 2019. xixi, 23

[12] H. Willee, "Px4 platform hardware/software architecture." https://dev.px4.io/master/en/concept/dronecode_architecture.html, November 2019. xixi, 24

[13] A. Urquizo, "Pid controller — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=PID_controller&oldid=923999209, December 2011. xixi, 34

[14] M. Rabah, A. Rohan, M. Talha, K.-H. Nam, and S. Kim, "Autonomous vision-based target detection and safe landing for uav," *International Journal of Control, Automation and Systems*, vol. 16, pp. 3013–3025, 12 2018. xixi, 39

[15] H. Willee, C. Dongcai, and B. Kung, "Airframes reference." https://dev.px4.io/v1.9.0/en/airframes/airframe_reference.html, January 2020. xiixii, 47

[16] Intel, "Intel realsense depth cameera d400-series." https://software.intel.com/en-us/realsense/d400, 2019. xiixii, 65

[17] Carnegie Robotics, "Carnegie robotics multisense s7." https://carnegierobotics.com/multisense-s7, 2019. xiixii, 65

[18] MaxBotic Inc., "Maxsonar ultrasonic sensor product selector." https://www.maxbotix.com/Ultrasonic_Sensors.htm, 2019. xiixii, 66

[19] LeddarTech Inc., "Leddartech leddarone product page." https://leddartech.com/lidar/leddarone/, 2019. xiixii, 67

[20] Benewake , "Benewake tf-2 lidar product page." http://en.benewake.com/product/detail/5c345c9de5b3a844c4723299.html, 2019. xiixii, 67

[21] Hokuyo Automatic , "Hokuyo utm-30lx product page." https://www.hokuyo-aut.jp/search/single.php?serial=169, 2019. xiixii, 68

[22] Shenzhen Yuedeng Technology, "Ydlidar g4 product page." http://www.ydlidar.com/products/view/3.html, 2019. xiixii, 68

[23] Ocular Robotics Limited, "Ocular robotics re05 3d lidar scanner product page." https://www.ocularrobotics.com/products/lidar/re05/, 2019. xiixii, 69

[24] SICK AG, "Sick lms1000 2d lidar sensor product page." https://www.sick.com/nl/en/detection-and-ranging-solutions/2d-lidar-sensors/lms1000/c/g387151, 2019. xiixii, 69

[25] Terabee, "Teraranger tower evo product page." https://www.terabee.com/shop/lidar-tof-multi-directional-arrays/teraranger-tower-evo/, 2019. xiixii, 69

[26] Hamish Willee, "Mavlink common message set." https://mavlink.io/en/messages/common.html, February 2019. xiiixiii, xiiixiii, 27, 28

[27] B. Canis, "Unmanned aircraft systems (uas): Commercial outlook for a new industry," 2015. 1

[28] G. M. Crutsinger, J. Short, and R. Sollenberger, "The future of uavs in ecology: an insider perspective from the silicon valley drone industry," *Journal of Unmanned Vehicle Systems*, vol. 4, no. 3, pp. 161–168, 2016. 1

[29] F. Giones and A. Brem, "From toys to tools: The co-evolution of technological and entrepreneurial developments in the drone industry," *Business Horizons*, vol. 60, no. 6, pp. 875 – 884, 2017. THE GENERATIVE POTENTIAL OF EMERGING TECHNOLOGY. 1

[30] T. H. Cox, C. J. Nagy, M. A. Skoog, I. A. Somers, and R. Warner, "Civil uav capability assessment," *NASA, Tech. Rep., draft Version*, 2004. 1

[31] D. Murugan, A. Garg, T. Ahmed, and D. Singh, "Fusion of drone and satellite data for precision agriculture monitoring," in *2016 11th International Conference on Industrial and Information Systems (ICIIS)*, pp. 910–914, Dec 2016. 1

[32] A. J. A. Dhivya and J. Premkumar, "Quadcopter based technology for an emergency healthcare," in *2017 Third International Conference on Biosignals, Images and Instrumentation (ICBSII)*, pp. 1–3, March 2017. 1

[33] S. K. Datta, J. Dugelay, and C. Bonnet, "Iot based uav platform for emergency services," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 144–147, Oct 2018. 1

[34] R. Obradović, I. Vasiljević, D. Kovačević, Z. Marinković, and R. Farkas, "Drone aided inspection during bridge construction," in *2019 Zooming Innovation in Consumer Technologies Conference (ZINC)*, pp. 1–4, May 2019. 1

[35] Gartner Research, Dale Kutnick, "10 Critical Components Driving the Robot and Drone Revolution ." Gartner Research report, October 2017. ID: G00328769. 1

[36] S. Jin, J. Cho, X. D. Pham, K. M. Lee, S. Park, M. Kim, and J. W. Jeon, "Fpga design and implementation of a real-time stereo vision system," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, pp. 15–26, Jan 2010. 1

[37] Markets and Markets, "Unmanned Aerial Vehicle (UAV) Market by Application (ISR, Precision Agriculture, Product Delivery), Class (Tactical, MALE, HALE, UCAV), System (Avionics, Sensors, Payload), MTOW (<25Kg, 25-150Kg, >150kg), Range, Type, and Region - Global Forecast to 2025." Market Research report - AS2802, February 2018. 3

[38] L. Schäffer, Z. Kincses, and S. Pletl, "A real-time pose estimation algorithm based on fpga and sensor fusion," in *2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY)*, pp. 000149–000154, Sep. 2018. 3

[39] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Fluids Engineering*, vol. 82, pp. 35–45, 03 1960. 3, 20

[40] M. E. Conde, S. Cruz, D. M. Muñoz, C. H. Llanos, and E. L. F. Fortaleza, "An efficient data fusion architecture for infrared and ultrasonic sensors, using fpga," in *2013 IEEE 4th Latin American Symposium on Circuits and Systems (LASCAS)*, pp. 1–4, Feb 2013. 3

[41] M. Itani, A. Haroun, and W. Fahs, "Obstacle avoidance for ultrasonic unmanned aerial vehicle monitoring using android application," in *2018 International Arab Conference on Information Technology (ACIT)*, pp. 1–4, Nov 2018. 3

[42] K. Li, C. Wang, S. Huang, G. Liang, X. Wu, and Y. Liao, "Self-positioning for uav indoor navigation based on 3d laser scanner, uwb and ins," in *2016 IEEE International Conference on Information and Automation (ICIA)*, pp. 498–503, Aug 2016. 3

[43] C. Cigla, R. Brockers, and L. Matthies, "Image-based visual perception and representation for collision avoidance," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 421–429, July 2017. 4

[44] N. Gageik, P. Benz, and S. Montenegro, "Obstacle detection and collision avoidance for a uav with complementary low-cost sensors," *IEEE Access*, vol. 3, pp. 599–609, 2015. 4

[45] R. Li, J. Liu, L. Zhang, and Y. Hang, "Lidar/mems imu integrated navigation (slam) method for a small uav in indoor environments," in *2014 DGON Inertial Sensors and Systems (ISS)*, pp. 1–15, Sep. 2014. 4

[46] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics Automation Magazine*, vol. 13, pp. 99–110, June 2006. 4

[47] The Linux Foundation, "Dronecode - the open source uav platform." `https://www.dronecode.org/`, 2019. 4

[48] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6235–6240, May 2015. 4, 33

[49] The Linux Foundation, "Px4 autopilot." `https://px4.io/`, 2019. 4

[50] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A system for autonomous flight using onboard computer vision," in *2011 IEEE International Conference on Robotics and Automation*, pp. 2992–2997, May 2011. 4

[51] ArduPilot, "Ardupilot open source autopilot." https://http://ardupilot.org, 2019. 4

[52] B. Gati, "Open source autopilot for academic research - the paparazzi system," in *2013 American Control Conference*, pp. 1478–1481, June 2013. 5

[53] F. Ruess, "Paparazzi uav." http://wiki.paparazziuav.org/wiki/Main_Page, december 2018. 5

[54] M. Lizarraga, G. H. Elkaim, and R. Curry, "Slugs uav: A flexible and versatile hardware/software platform for guidance navigation and control research," in *2013 American Control Conference*, pp. 674–679, June 2013. 5

[55] UCSC Autonomous Systems Lab, "Santa cruz low-cost uav gnc system." https://slugsuav.soe.ucsc.edu/, 2009. 5

[56] LibrePilot, "Librepilot open-source software suite to control multicopter." https://www.librepilot.org/site/index.html, 2019. 5

[57] Open Source Robotics Foundation, "Robot operating system." https://www.ros.org/, 2019. 5

[58] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009. 5

[59] D. Koch and J. Jackson, "Rosflight." https://rosflight.org, January 2018. 5

[60] J. Perron, "Robot operating system 2." https://index.ros.org/doc/ros2/, May 2019. 5

[61] UAVenture AG, "Uaventure a complete hybrid vtol autopilot solution." http://uaventure.com/, 2018. 5

[62] A. Astudillo, P. Muñoz, F. Álvarez, and E. Rosero, "Altitude and attitude cascade controller for a smartphone-based quadcopter," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1447–1454, June 2017. 6

[63] G. Kravit, "Fpga implementation of a digital controller for a small vtol uav," Master's thesis, Massachusetts Institute of Technology, 12 2014. 6

[64] M. A. Lukmana and H. Nurhadi, "Preliminary study on unmanned aerial vehicle (uav) quadcopter using pid controller," in *2015 International Conference on Advanced Mechatronics, Intelligent Manufacture, and Industrial Automation (ICAMIMIA)*, pp. 34–37, Oct 2015. 6

[65] Free Software Foundation, "Frequently asked questions about the gnu licenses." https://www.gnu.org/licenses/gpl-faq.en.html, 2019. 6

[66] Free Software Foundation, "Various licenses and comments about them - modified bsd." https://www.gnu.org/licenses/license-list.html#ModifiedBSD, 2019. 6

[67] K. Yamashina, H. Kimura, T. Ohkawa, K. Ootsu, and T. Yokota, "crecomp: Automated design tool for ros-compliant fpga component," in *2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*, pp. 138–145, Sep. 2016. 7

[68] Xillybus, "Xillinux: A linux distribution for z-turn lite, zedboard, zybo and microzed." http://xillybus.com/xillinux, 2019. 7

[69] Y. Nitta, S. Tamura, and H. Takase, "A study on introducing fpga to ros based autonomous driving system," in *2018 International Conference on Field-Programmable Technology (FPT)*, pp. 421–424, Dec 2018. 7

[70] D. Valencia and D. Kim, "Quadrotor obstacle detection and avoidance system using a monocular camera," in *2018 3rd Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, pp. 78–81, July 2018. 7

[71] L. Matthies and S. Shafer, "Error modeling in stereo navigation," *IEEE Journal on Robotics and Automation*, vol. 3, pp. 239–248, June 1987. 7

[72] Y. Xiao, X. Lei, and S. Liao, "Research on uav multi-obstacle detection algorithm based on stereo vision," in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 1241–1245, March 2019. 7

[73] J. Hu, Y. Niu, and Z. Wang, "Obstacle avoidance methods for rotor uavs using realsense camera," in *2017 Chinese Automation Congress (CAC)*, pp. 7151–7155, Oct 2017. 8

[74] O. Hall-Holt and S. Rusinkiewicz, "Stripe boundary codes for real-time structured-light range scanning of moving objects," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 2, pp. 359–366, IEEE, 2001. 8

[75] E-linux community, "Jetson tk1." https://elinux.org/Jetson_TK1, July 2019. 8

[76] S. Clark and G. Dissanayake, "Simultaneous localisation and map building using millimetre wave radar to extract natural features," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 2, pp. 1316–1321 vol.2, May 1999. 8

[77] A. Corporation, "Technical guide for ultrasonic sensors." http://us.azbil.com/CP-GC1003E_Vol.1_a_c/B/TEC_ULTRASONIC.pdf, November 2014. 8

[78] J. Lim, S. Lee, G. Tewolde, and J. Kwon, "Indoor localization and navigation for a mobile robot equipped with rotating ultrasonic sensors using a smartphone as the robot's brain," in *2015 IEEE International Conference on Electro/Information Technology (EIT)*, pp. 621–625, May 2015. 8

[79] Y. Peng, D. Qu, Y. Zhong, S. Xie, J. Luo, and J. Gu, "The obstacle detection and obstacle avoidance algorithm based on 2-d lidar," in *2015 IEEE International Conference on Information and Automation*, pp. 1648–1653, Aug 2015. 9

[80] N. Giordano, "The doppler effect," in *College Physics* (Cengage Learning, ed.), pp. 421–424, Cengage Learning. 10

[81] Aerolab, "T-motor ms2216 kv900." https://www.aerolab.de/brushless-motoren/t-motor-ms/t-motor-ms2216-kv900_107002_1096, 2019. 12

[82] B. L. Sharma, N. Khatri, and A. Sharma, "An analytical review on fpga based autonomous flight control system for small uavs," in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pp. 1369–1372, March 2016. 13

[83] J. Kok, L. F. Gonzalez, and N. Kelson, "Fpga implementation of an evolutionary algorithm for autonomous unmanned aerial vehicle on-board path planning," *IEEE Transactions on Evolutionary Computation*, vol. 17, pp. 272–281, April 2013. 13

[84] F. A. Abouelghit, H. ElSayed, G. I. Alkady, H. H. Amer, and I. Adly, "Fpga-based fault-tolerant quadcopter with fuzzy obstacle avoidance," in *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1–4, June 2019. 13

[85] G. Premkumar, R. Jayalakshmi, and M. Akramuddin, "Design and implementation of fpga based quadcopter," *International Journal of Engineering Technology Science and Research*, vol. 5, pp. 558–562, 2018. 13

[86] N. Monterrosa, J. Montoya, F. Jarquín, and C. Bran, "Design, development and implementation of a uav flight controller based on a state machine approach using a fpga embedded system," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pp. 1–8, Sep. 2016. 13

[87] P. Smyczyński, Starzec, and G. Granosik, "Autonomous drone control system for object tracking: Flexible system design with implementation example," in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, pp. 734–738, Aug 2017. 13

[88] A. Janarthanan, H. W. Ho, L. Gopal, V. Shanmugam, and W. K. Wong, "An unmanned aerial vehicle framework design for autonomous flight path," in *2019 7th International Conference on Smart Computing Communications (ICSCC)*, pp. 1–5, June 2019. 13

[89] E. Chirtel, R. Knoll, C. Le, B. Mason, N. Peck, J. Robarge, and G. C. Lewin, "Designing a spatially aware, autonomous quadcopter using the android control sensor system," in *2015 Systems and Information Engineering Design Symposium*, pp. 35–40, April 2015. 13

[90] W. Y. Lai, M. J. Er, Z. C. Ng, and Q. W. Goh, "Semi-autonomous control of an unmanned aerial vehicle," in *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 1–4, Nov 2016. 13

[91] Federal Aviation Administration, "Automated flight controls." http://www.faa.gov/regulations_policies/handbooks_manuals/aviation/advanced_avionics_handbook/media/aah_ch04.pdf, 2014. 16

[92] Bosch SensorTec, *BMI088: Data sheet*. Bosch SensorTec, https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMI088-DS001.pdf, 1.4 ed., September 2018. BMI088 6-axis Motion Tracking for High-performance Application. 16

[93] Bosch SensorTec, *BMI150: Data sheet*. Bosch SensorTec, https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMM150-DS001.pdf, 1.2 ed., April 2019. BMI150 Geomagnetic Sensor. 16

[94] Bosch SensorTec, *BME680: Data sheet*. Bosch SensorTec, https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMM150-DS001.pdf, 1.3 ed., July 2019. BME680 Low power gas, pressure, temperature and humidity sensor. 17

[95] U-blox, *ZOE-M8B: Data sheet*. U-blox, https://www.u-blox.com/sites/default/files/ZOE-M8B_DataSheet_%28UBX-17035164%29.pdf, r03 ed., March 2018. ZOE-M8B Ultra-small, super low power u-blox M8 GNSS SiP module. 17

[96] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, pp. 401–422, March 2004. 20

[97] L. Idkhajine, E. Monmasson, and A. Maalouf, "Fully fpga-based sensorless control for synchronous ac drive using an extended kalman filter," *IEEE Transactions on Industrial Electronics*, vol. 59, pp. 3908–3918, Oct 2012. 20

[98] B. Barshan and H. F. Durrant-Whyte, "Inertial navigation systems for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 11, pp. 328–342, June 1995. 20

[99] D. Brescianini, M. Hehn, and R. D'Andrea, "Nonlinear quadrocopter attitude control: Technical report," tech. rep., ETH Zurich, 2013. 20

[100] P. Mallavarapu, H. N. Upadhyay, G. Rajkumar, and V. Elamaran, "Fault-tolerant digital filters on fpga using hardware redundancy techniques," in *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, vol. 2, pp. 256–259, April 2017. 25

[101] H. Wei, Z. Shao, Z. Huang, R. Chen, Y. Guan, J. Tan, and Z. Shao, "Rt-ros: A real-time ros architecture on multi-core processors," *Future Generation Computer Systems*, vol. 56, pp. 171 – 178, 2016. 25

[102] H. Wei, Z. Huang, Q. Yu, M. Liu, Y. Guan, and J. Tan, "Rgmp-ros: A real-time ros architecture of hybrid rtos and gpos on multi-core processor," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2482–2487, May 2014. 25

[103] J. Corbet, A. Rubini, and G. Kroah-Hartman, *Linux Device Drivers, Third Edition.* O'Reilly Media, Inc., 3 ed., 2005. 31

[104] Embedded Linux Contributors, "Device tree usage," April 2019. 31

[105] Fu-ming Xiao, Dong-sheng Li, Gao-ming Du, Yu-kun Song, Duo-li Zhang, and Ming-lun Gao, "Design of axi bus based mpsoc on fpga," in *2009 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication*, pp. 560–564, Aug 2009. 31

[106] Xilinx, *Vivado Design Suite - AXI Reference Guide*, July 2017. 31

[107] "iio, a new kernel subsystem." 32

[108] Analog devices, "What is libiio?," January 2018. 32

[109] Kiam Heong Ang, G. Chong, and Yun Li, "Pid control system analysis, design, and technology," *IEEE Transactions on Control Systems Technology*, vol. 13, pp. 559–576, July 2005. 34

[110] J. G. Ziegler and N. B. Nichols, "Optimum Settings for Automatic Controllers," *Journal of Dynamic Systems, Measurement, and Control*, vol. 115, pp. 220–222, 06 1993. 34

[111] K. J. Åström and T. Hägglund, *PID controllers : theory, design, and tuning.* Research Triangle Park, N.C.: International Society for Measurement and Control, 1995. 34

[112] H. Willee and M. Grob, "Multicopter pid tuning guide." https://docs.px4.io/v1.9.0/en/config_mc/pid_tuning_guide_multicopter.html, September 2019. 34

[113] O. Liang, "Quadcopter pid explained." https://oscarliang.com/quadcopter-pid-explained-tuning/, July 2018. 35

[114] E. Kuantama, D. Craciun, and R. Tarca, "Quadcopter body frame model and analysis," *ANNALS OF THE ORADEA UNIVERSITY. Fascicle of Management and Technological Engineering.*, vol. Volume XXV (XV), 2016/1, 05 2016. 38

[115] C. C. Finlay, S. Maus, C. D. Beggan, T. N. Bondar, A. Chambodut, T. A. Chernova, A. Chulliat, V. P. Golovkov, B. Hamilton, M. Hamoudi, R. Holme, G. Hulot, W. Kuang, B. Langlais, V. Lesur, F. J. Lowes, H. Lühr, S. Macmillan, M. Mandea, S. McLean, C. Manoj, M. Menvielle, I. Michaelis, N. Olsen, J. Rauberg, M. Rother, T. J. Sabaka, A. Tangborn, L. Tøffner-Clausen, E. Thébault, A. W. P. Thomson, I. Wardinski, Z. Wei, and T. I. Zvereva, "International Geomagnetic Reference Field: the eleventh generation," *Geophysical Journal International*, vol. 183, pp. 1216–1230, 12 2010. 40

[116] National Aeronautics and Space Administration, "Measuring earth's magnetism." https://earthobservatory.nasa.gov/images/84266/measuring-earths-magnetism, September 2014. 40

[117] J. D. Sally and P. Sally, *Roots to research: a vertical development of mathematical problems.* American Mathematical Society Bookstore, 2007. 40

[118] DJI, "Introducing the new a2 gps pro plus module." https://www.dji.com/nl/newsroom/news/introducing-the-new-a2-gps-pro-plus-module, January 2015. 41

[119] H. Willee, P. Riseborough, and B. Kung, "Gps compass." https://docs.px4.io/v1.9.0/en/gps_compass/, August 2019. 41

[120] E. Kuantama, I. Tarca, R. Tarca, and D. Craciun, "Aspects regarding fly control of quadcopter," *Recent Innovations in Mechatronics*, vol. 3, 09 2016. 41

[121] Lightbend, "The reactive manifesto." https://www.reactivemanifesto.org/, September 2014. 51

[122] Windows2universe.org, "The multispectral sun, from the national earth science teachers association." http://www.windows2universe.org/sun/spectrum/multispectral_sun_overview.html, April 2007. 68

# Appendix A

# Appendix sensors

## A.1 What currently available sensor can be used to detect obstacles?

### A.1.1 Stereo camera

**Intel Realsense**

Intel is mostly known for its CPU's, but they also offer various other products as for instance the realsense camera. These use two camera's to determine the depth and have a wide field of view of 85 degrees. The exact maximum range is stated as 10 meters but with correct calibration this can be extended. The output is send over a USB-C port and the API of realsense can be used to retrieve the data from the sensor. The sensor itself is also guaranteed to work outdoor without any performance loss.

**Carnegie Robotics**

Carnegie robotics specializes in robust sensors for robotics, they have a couple of stereo sensors available for long and short range applications. Their camera has a wide field of view of 80 degrees and they have a C++ library and even ROS nodes available. The output is send over Ethernet. These sensors have several demo's online available and work outdoor without any performance loss.

Figure A.1: Intel realsense D435 [16]

Figure A.2: Carnegie Multisense S7 [17]

**Stereolabs**

Stereolabs offers different stereo camera's, the zed and the zed mini. Like all the Stereo camera's it has a big Field of view of 90 by 60 degrees and works good outside. It has a reported depth range of 20 meters up to 40 meters. It is connected via a USB3.0 port and its data can be retrieved via ROS or the API. The package of the sensor is relative small and only weighs 159 grams.


Figure A.3: Stereolabs zed [1]

### A.1.2 Radar

**µSharp**

The µSharp is developed by Aerotenna. It uses a frequency of 24 Ghz and has a maximum range of 120 meters. It costs approximately 625 euros. The output can be retrieved via UART. It has a measuring angle of approximately 50 by 30 degrees and 1 sensor has a power consumption of 1.25W. The sensor is in a compact housing and due to the 24 Ghz frequency it works good in for instance rain.


Figure A.4: µSharp [2]

### A.1.3 Ultrasonic

**HC-SR04**

This is one of the most used ultrasonic sensors in robotics, it is cheap with a price of only 5 euro and works really simple. First a high signal is written to the trig pin which sends out the message, then an 8 cycle sonic burst will be send out by the sensor and via the echo pin the duration will be returned. This duration must be multiplied by 0.034 and divided by 2 because the signal travels 2 times the distance to the object. It has an operation range between 2 and 400 centimeters. It has a measuring angle of approximately 15 degrees. The power usage of 1 sensor is approximately 5V and 15mA thus 75mW.

**MaxSonar**

These sensors are a bit more expensive, ranging from 25 to 50 euro. It is a series from MaxBotix with various variants ranging in things like for indoor or outdoor usage, range, output, beam angle, input voltage etc.

It is a series from MaxBotix with various variants ranging in things like for indoor or outdoor usage, range, output, beam angle, input voltage etc. These sensors can have a range up to 10 meters with a very shallow beam and reflected on a perfect flat surface, the expected range of this sensor is around 6 meters. For controlling this sensor only an input voltage is needed, the output is then presented in Analog Voltage, RS232 Serial or Pulse width. It only uses an average of 3.4mA with 5V for a total of 17mW. Optionally these sensors are available for outdoor usage with protective casing but this will greatly reduce either the range or the beam angle of the sensor.

Figure A.5: HC-SR04 [3]

Figure A.6: MaxSonar [18]

### A.1.4 Lidar

**LIDAR-Lite v3**

This sensor is available for 130 to 150 euro. It is developed by Garmin and has two variants, the normal variant which can sample up to 500Hz and the HP variant which can sample at rates around 1kHz and has a water-resistant casing. Both variants have a range of up to 40 meters which is considerable bigger than the ultrasonic sensors. However, the beam that it covers is also much shallower of only approximately 0.5 degree. This sensor can be read via 2 different methods, either PWM or I2C. It also requires a bit more power, it still uses 5V but on average it uses 130mA which results in a total of 650mW.

**LeddarTech**

This company has several Lidar sensors available ranging with a beam from only 3 degrees up to 100 degrees. Going to a wider beam will greatly reduce the range of the sensor varying from 60 meters to only 12 meters. These sensors are also a bit more expensive where it starts with 150 euro up to 1100 euro for the wide range variants. In order to read this sensor either UART can be used or RS-485. It uses more power than the previous sensor as the power consumption is rated at 1.3W which is thus twice the usage of the previous sensor. For the wide beam sensors this can increase to 2W.

Figure A.7: Lidar-Lite [4]

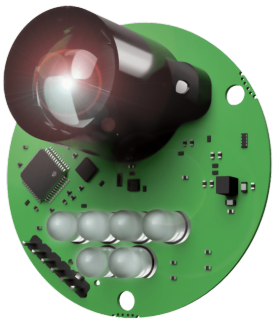

Figure A.8: LeddarOne by LeddarTech [19]

**Benewake**

Benewake offers the possibility to choose between a Short-range, Mid-range or Long-range Lidar sensor. Ranging from a maximum operating range of 12 meters to a maximum range of 180 meters. All these sensors have like most Lidar devices a really small beam width and cost between 100 and 250 euros. The sensors can be read using either UART or CAN. All these sensors are less than 100 grams and have an average power usage of 0.6W.

Figure A.9: Benewake TF02 [20]

### A.1.5 360 Lidar

**RPLIDAR**

RPLidar is developed by Slamtec, the price of the RPLidars range from only 115 euro up to 650 euro. There are different Lidars available with different ranges and different number of samples. Where the maximum range is between 12 and 40 meters. The sensor gives as output the distance in millimeter and the current degree of the measurement via UART. The power consumption by the sensor is approximately 5V at 350mA for a total of 1.75W. These sensors also have the possibility to communicate with ROS which can be used to plan a new path.

**Velodyne**

Velodyne has several Lidars available, many of the sensors used in the automotive industry are being produced by Velodyne. Velodyne is one of the biggest high-end Lidar companies available. They have several variants of Lidar sensors available which have a small form factor and low weight as for instance the Puck LITE, although it still weights 590 grams. It has a range of 100 meters with a vertical angle of 30 degrees. In order to read this sensor UDP packets are send over Ethernet. This sensor requires a total of 8W to run. This sensor is obviously much better than what we have seen so far but this also comes at a price, the exact prices are not known but rumored around 4000 euro.

Figure A.10: RPLidar S1 [5]


Figure A.11: Velodyne Puck [6]

**Hokuyo**

Hokuyo is a manufacturer of a lot of different industrial products with a lot of different Lidar sensors available ranging from 1000 euro up to 5000 euro. Most of the Lidar sensors have a range of 270 degrees. This is not really an issue compared to the 360 degrees of the other sensors since it can be avoided to fly backwards. The range of the Lidar sensors can be up to 30 meters and they have special sensors which can work outside. For receiving the output of the Hokuyo sensors RS232 can be used or the data can be directly used in ROS like the RPLidar to plan the new path. The sensors require 12 V at 0.7A for a power consumption of 8.4W.

**YDLidar**

YDLidar offers 3 different Lidars. A cheap Lidar for indoor small robotics usage and a more advanced Lidar capable of scanning up to 16 meters for a price of around 450 euro. The output of this sensor can be retrieved via UART. It uses 5V with 450mA for a power consumption of 2.25W. This sensor also has the possibility to be integrated with ROS to avoid obstacles. An interesting fact is that this Lidar uses a different Wavelength than most other Lidar devices at 785nm instead of 905nm. This likely results in more interference from the sun [122].

Figure A.12: Hokuyo UTM-30LX [21]

Figure A.13: YDLidar G4 [22]

**Ocular robotics**

Ocular Robotics offers a lot of different sensors including 2 Lidars which cost around 8000 euro. According to Ocular robotics they are the smartest Lidars on the market. This is due to the many different modes that can be chosen to scan, either everything that is possible or only a very narrow band or a certain region. They have high range up to 160 meters. The output of the Lidar is given via Ethernet. It has a power consumption of approximately 50W and it weights around 3kg.

**SICK**

SICK is a company that specializes in sensors, they have a couple of 2D and even 3D Lidars available. These sensors typically have a beam width of 275 degrees and sensors which are especially for outdoor usage. The typical range of these is 30 meters. Additionally, these sensors include different filters to achieve better results and neglect the effects of fog as much as possible. The output can be retrieved using Ethernet. The typical power usage of the sensor is 18W.


Figure A.14: Ocular Robotics RE05 [23]


Figure A.15: SICK LMS1000 [24]

### A.1.6 Time of Flight LED

**Terabee**

Terabee is a company that offers multiple sensor modules all based on Time of Flight sensors. They have different variants of Time of Flight sensors available. A special long range sensor which has a range of 60 meters in a really compact housing. Outdoor these sensors would get a range of 20-30 meters which would make them really suitable. This is however doubtful since all other Time of Flight LED sensors report a maximum range of several meters. The data can be retrieved via I2C or UART. The typical power usage of the sensor is 13W.

Figure A.16: TeraRanger Tower Evo [25]

### A.1.7 Criteria

Batteries are 48.8 Wh.

The motors that are going to be used in this project are the TMotor 900kV brushless DC motors, this means for every volt it spins at 900 rotations per minute. At 100% throttle the motor consumes 147.4 watts and generates a thrust of 1000 gramms resulting in an efficiency of 6.78 Gramms per Watt. Since the same motors are used this value can be used to determine the power impact. The efficiency decreases the more throttle that is used and thus the worst-case scenario will be used. This means that the flight impact can be approximated as $Powerimpact = Powerusage + \frac{Weight}{6.78}$.

Below in Table A.1 some of the available sensors are given. Not all different sensors are in the table since that would make the table to cluttered. It can overall be observed that the longer the range the higher the power usage. An important parameter not reflected that much in the text is the weight. A drone must be able to lift the entire drone and sensor. The higher the weight the more throttle must be used and thus power must be used.

Additionally, it is not required to get full 360 degree visibility, since the drone will likely only move forwards or sideways. If we look at the table there are several options available. Multiple small beam width sensors could be used like the HC-SR04 and Lidar-Lite, as done in the TeraRanger Tower Evo (see Figure A.16). Multiple stereo camera's can be used in order to get vision all around the drone. The ZED has the best specifications for this project from the stereo cameras. The µsharp would also be a possibility although multiple of these sensors would be needed to ensure good vision.

Another approach that can be taken is using a 360-degree Lidar. This would only require one single sensor to map the entire surroundings. There are several 360-degree Lidars available from which the RPLidar S1, YDLidar G4 or puck LITE seems like the best solution. The rest of the sensors either lack range, are to heavy or have better alternatives. The puck LITE is a really high-end sensor with its high range with relative low power usage and low weight. Something that could give a problem is the interface which is Ethernet and the price of approximately 4000 euro. The YDLidar G4 has good specifications for its low price, the only questionable thing about this sensor is that it uses a different wave length compared to other Lidars. This is much more radiated by the sun and likely result in more interference when flying outdoors especially when taking into consideration that the maximum range is only 16 meters. This will likely be reduced heavily outside. In the end the RPLidar S1 seems like the best solution in terms of 360 Lidar. Another possible solution could be the TeraRanger Tower Evo, which on paper offers good range (which is doubtful) and also the 360 degrees vision. Tests would have to be conducted to see which performs best outdoors.

Table A.1: Table of available sensors

| **Sensor** | **Max range** | **Beam width** | **Power** | **Weight** | **Power Impact per sensor** | **Cost** |
|---|---|---|---|---|---|---|
| Intel Realsense D435 | 10 meters | ∼85 degrees | 1.25 W | 72 g | 11.9 | €160 |
| Carnegie Multisense S7 | 10 meters | ∼80 degrees | 20 W | 1.2 kg | 197.0 | ∼€4000 |
| Stereo labs ZED | 20 meters | ∼90 degrees | 2 W | 159 g | 25.5 | ∼€450 |
| µSharp | 120 meters | ∼50 degrees | 1.25 W | 43 g | 7.6 | €625 |
| HC-SR04 | 4 meters | ∼15 degrees | 75 mW | 8.5 g | 1.3 | €5 |
| MaxSonar | 6 meters | ∼10 degrees | 17 mW | 32 g | 4.7 | €25-€50 |
| LIDAR-Lite v3 | 40 meters | 0.5 degree | 650 mW | 22 g | 3.9 | €130 |
| LIDAR-Lite v3 HP | 40 meters | 0.5 degree | 325 mW | 38 g | 5.9 | €150 |
| LeddarOne | 40 meters | 3 degree | 1.3 W | 14 g | 3.4 | €150 |
| LeddarVU8 | 185 meters | ∼20 degrees | 2 W | 110 g | 18.2 | €475 |
| Benewake TF02 | 22 meters | 3 degree | 0.6 W | 52 g | 8.3 | €100 |
| RPLidar A3 | 25 meters | 360 degrees | 6 W | 190 g | 34.0 | €600 |
| RPLidar S1 | 40 meters | 360 degrees | 1.75 W | 105 g | 17.2 | €650 |
| puck LITE | 100 meters | 360 degrees | 8 W | 590 g | 95.0 | ∼€4000 |
| Hokuyo UTM-30LX | 30 meters | 270 degrees | 8.4 W | 210 g | 39.4 | ∼€4900 |
| YDLidar G4 | 16 meters | 360 degrees | 2.25 W | 214 g | 33.8 | €350 |
| Ocular Robotics RE05 | 160 meters | 360 degrees | 50 W | 2.8 kg | 463.0 | ∼€8000 |
| SICK LMS1000 | 64 meters | 275 degrees | 18 W | 1.2 kg | 195.0 | ∼€3000 |
| TeraRanger Tower Evo | 60 meters | ∼360 degrees | 13 W | 130 g | 32.2 | ∼€600 |

# Appendix B

# Overview image

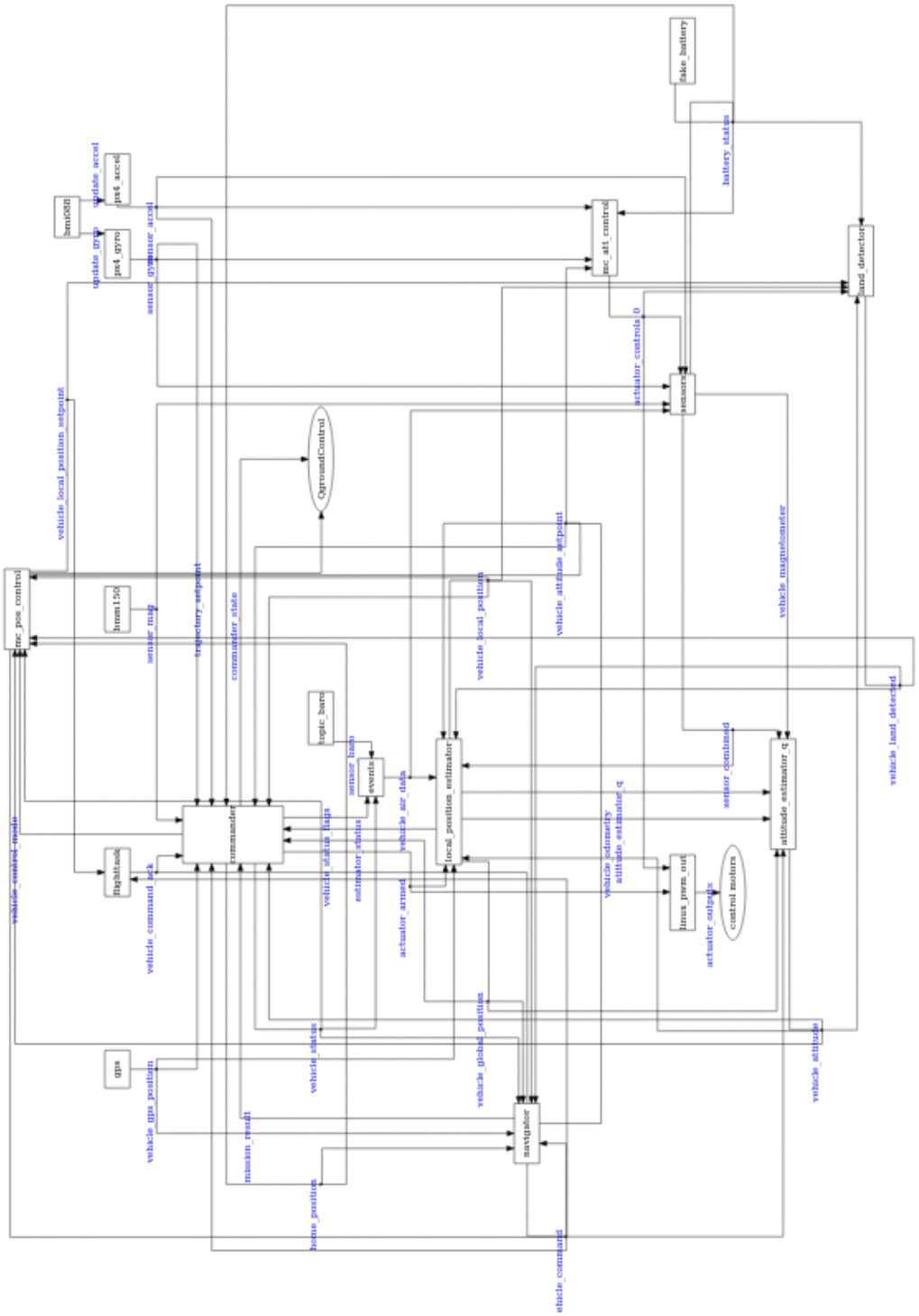


Figure B.1: Black rectangles represent modules, lines indicate messages being communicated with the blue fonts naming the message