Hochschule Augsburg University of Applied Sciences

Bachelor Thesis

Faculty of Computer Science

Degree course
Computer Science

Oliver Seitz

Development of an attitude stabilization algorithm for quadcopters based on an inertial measurement unit

First examiner: Prof. Dr. Hubert Högl
Second examiner: Prof. Dr. Peter Rösch
Submission of work: February 20, 2020

Hochschule für angewandte Wissenschaften Augsburg
University of Applied Sciences

An der Hochschule 1
D-86161 Augsburg

Phone +49 821 55 86-0
Fax +49 821 55 86-3222
www.hs-augsburg.de
info@hs-augsburg.de

Faculty of Computer Science
Phone +49 821 5586-3450
Fax +49 821 5586-3499

Author of the Bachelor Thesis:
Oliver Seitz
Schrobenhausener Straße 51
86551 Aichach
Phone +49 176 83311461
oliver.seitz@hs-augsburg.de
seitz-oliver@gmx.de

Abstract

Multicopters are fascinating aircraft. They can ascend, descend, tilt, and rotate just by their motors without any other mechanical component. Their mechanical simplicity makes them more inexpensive and easier to build than helicopters but, at the same time, more agile. Many businesses emerge around the multicopter like parcel delivery or videography. All of them require the multicopter to be able to stabilize its attitude to ensure a stable flight. The pilot should not have to secure the multicopter manually. Therefore advanced control systems are required that observe the current attitude of the multicopter and calculate all necessary steering commands to hold it steady.

During this work, a quadcopter is built and programmed to evaluate standard methods and algorithms for multicopter stabilization. This work explains which parts are required to create a multicopter from scratch. Further, it explains why each specific component is needed and how they interact with each other. This work focusses on the implementation of an IMU (Inertial Measurement Unit) to estimate the current attitude of the multicopter. An accelerometer and a gyroscope obtain attitude relevant information for the IMU. This work will present the basic principles of the MEMS accelerometer and gyroscope.

The multicopter experiences a lot of vibrations due to the rotating propellers. The vibrations cause the output of the sensor to be noisy and can therefore not be used directly. This work analyzes the performance of two filter algorithms for processing noisy signals, namely the Kalman filter and the complementary filter.

The Kalman filter turns out to be a robust algorithm for filtering sensor output data. However, the Kalman filter was not chosen to be the best approach for a multicopter attitude stabilization system due to its complex tuning process. This work will explain how the complementary filter can be deduced from the Klaman filter. The complementary filter has only one parameter and is, therefore, straight forward to optimize.

A PID controller computes the required steering commands and controls the motor speeds accordingly. Two different setpoints of the PID controller are evaluated. A setpoint based on the angular velocity of the quadcopter provides the best results and the most stabilized flight.

Contents

Chapter 1

Introduction

Multicopters are on their way to innovate many business segments, which are nowadays served by cars, planes, and humans. A multicopter is a flying object which uses only propellers to ascend, descend, accelerate, tilt and rotate. A multicopter is a very agile aircraft. The difference between a helicopter and a multicopter is that a helicopter steers by tilting its propeller blades into the direction it wants to fly, whereas a multicopter steers by applying different speeds to its multiple propellers which makes it mechanically very simple but preserves the agility of a helicopter over an airplane.

Most people nowadays imagine either a toy or videography device if they here the term multicopter, but because of its agility and ability to carry heavy loads, multicopters are also getting more and more attention in other business cases. The german package deliverer DHL, for example, is using multicopters to deliver parcels into regions where the infrastructure is insufficient for cars [1].

Next to transportation, the company DJI known for their videography drones is also manufacturing a multicopter capable of spraying pesticides and fertilizers [2].

Multicopters do have one big downside compared to conventional planes. Due to their physics, they are tough to fly without automatic stabilization. Therefore intelligent control modules are obliged capable of capturing the current attitude of the multicopter and calculating precise steering commands in order to stabilize the flight or hold the position.

The main focus of this thesis is the construction and programming of a multicopter with an automatic flight stabilization software capable of making the copter stabilize its attitude. In order to build up a working multicopter, this thesis introduces at first the central concepts and physics of multicopters. The

second part discusses all electronic components needed to capture the state of the copter, followed by all parts needed to issue, calculate, and execute steering commands. The following chapters then introduce several fundamental algorithms used to process sensor outputs and calculate steering commands followed by the actual implementation of a multicopter control and flight stabilization software. The last section of this thesis closes with an analysis of the achieved performance and a conclusion.

Chapter 2

Application Areas

As already explained in the introduction, multicopters enjoy great popularity within a wide variety of business sections. This chapter provides a deeper dive into the application areas of multicopters.

2.1 Hobby

Multicopters gained an impressive boost on the consumer market [3] [4]. Multicopers can deliver video material, which would be impossible for a human to capture. Many tourists and backpackers nowadays carry multicopter with them to get impressive shots of their destinations. A multicopter offers the possibility to explore the world from a birds-eye perspective. Famous brands are, for example, DJI, YUNEEC, and Parrot, while DJI dominates the market [4]. However, not only are those multicopters used by travelers, many people use them without any specific intention just as a toy to enjoy the world from a birds-eye perspective. This led to stricter laws in Germany to prevent accidents and violations of privacy.

There is also the big section of so-called racing drones. A racing drone is a multicopter which in most cases, has four rotors, and its purpose is to reach high speed and great agility. The fastest drone holding the world record can reach nearly 290 km/h [5]. Racing drones are small and light to support maneuverability. Pilots of racing drones either try to complete a course as first one or perform aerobatics using their multicopter. For that, there are competitions like the drone racing league or short DRL [6].

2.2 Professional

The application areas for multicopters in the commercial field are continually growing. The german parcel deliverer DHL works on a solution to deliver parcels not only faster but also to deliver parcels into areas that provide poor infrastructure. Such regions are, for example, remote villages. The multicopter of DHL is further used to deliver medication and other medical supplies in eastern African countries. Where cars often face impossible conditions like spilled roads or mountains, none of those are severe difficulties for a flying multicopter. Therefore the company Wingcopter [7] build the so-called 'parcel copter' in cooperation with DHL and the German Federal Ministry for Economic Cooperation and Development to collect blood samples of patients in order to transport them to laboratories for analyzing the samples and diagnosing the disease like malaria, typhoid, and schistosomiasis. This makes it possible for people living in inaccessible regions to receive medical treatment, which would not be possible without the Wingcopter drones since the equipment needed to analyze the samples is highly expensive, what makes building up laboratories in remote areas almost (or at least financially) impossible.

Another application area is getting explored by the manufacturer DJI. Spraying pesticides and fertilizers by hand is a time-consuming task. However, often it has to be done by hand since the environment is inaccessible for tractors, or there are just no tractors available since they cost high amounts of money. The AGRAS T16 is a multicopter for spraying pesticides and fertilizers developed by DJI capable of covering 10 hectares per hour. It can fly fully autonomous along a planned path and avoid obstacles. The AGRAS T16 can carry up to 16 liters liquid. As an additional feature, it helps to monitor plants by using its camera system. This way, it allows for quickly recognize threats like pests and diseases and monitor the irrigation of the fields.

There is a paper that proposes multicopters for taking portrait photos [8]. The proposed multicopter calculates trajectories to get from its starting position to an optimal position for taking the portrait picture. The copter is equipped with a camera that first tracks the pilot while approaching the desired position to ensure the camera is facing towards the pilot. After the multicopter reaches its final position, it takes the photo. The company 'Zero Zero Robotics' [9] develops such a copter, but instead of taking pictures, their multicopter can track a specific target and simultaneously record video material [10]. At this point, it must be mentioned that their product 'V-Coptr' [9] is not a multicopter per definition since it steers by tilting its propellers. Nevertheless, it reflects the versatility of multicopter like aircraft again.

Chapter 3

Overview

This chapter introduces the building blocks of multicopters. It further gives a quick overview of the following chapters by explaining which building block will be covered in which chapter.

The first big part is the body frame which holds all the electronical components. This work will not focus on the body frame and its specific properties. The chapter Building a Multicopter will make some assumptions about the body frame which are needed to fruther build up a mathematical model of the multicopter.

All the electronical components will be explained in the chapter Electronics. With respect to their importance for the main target of developing a stabilization software for multicopters the operating principal will be explained in more or less depth. Especially the components of the IMU (Inertial Measurement Unit) will be covered in great detail. This chapter will also provide insights into other projects like the Crazyflie 2.0 [11].

The stabilization software will read data from some of the electronical components described in the chapter Electronics and compute the steerings commands needed to stabilize the multicopters attitude. The chapter Control Theory will explain all core concepts required to process sensor readings, fuse the processed sensor information together and finally compute steering commands. This work makes use of several well known mathematical models. Since the focus of this work is on the application of those models to the field of attitude stabilization for multicopters this work will explain all approaches but omit extensive derivation.

After all concepts are clear this work will combine Electronics and Control Theory to explain how a microcontroller can be used to implement the stabilization software. This chapter will also provide insights into the development

process, analysis and debug capabilities of the STM platform.

The chapter Control Theory together with its successor Multicopter Control Software are the focus points of this work.

This work closes with a conclusion of the work done, achieved performance of the control software and possible, future, improvements.

Chapter 4

Building a Multicopter

This chapter will explain the multicopter that was built during this work.

A multicopter usually has four, six, or eight propellers, which are driven by the same number of engines. The number of propellers is not limited, but multicopters with more than eight propellers are not common. The built multicopter has four motors. Almost all multicopter use electrical engines. Chapter Actuators will explain two different types of electrical engines. This work uses brushless motors of the manufacturer T-Motor, named MN1806-13. Every motor comes with a propeller recommendation because the propeller needs to match the motor. A slow spinning motor can drive bigger propellers than a fast spinning motor. The propeller recommendation is the "8x2.7 Carbon Prop" of T-Motor. The propeller is made out of carbon, which makes it stiffer than plastic propellers. The two numbers denote the diameter, which is 8 Zoll, and the distance the propeller travels through the air when it spins 360 degrees, which is 2.7 Zoll. An important specification of the motor-propeller combination is the amount of thrust it provides. The motor-propeller combination works quadcopter can provide up to 384g * 4 = 1,536g of thrust. The final weight of the quadcopter is about 650g.

So-called ESCs regulate the motors, the chapter ESC Controller will provide more in-depth details. It applies the needed amount of power to the motors to control its speed. The quadcopter uses the "AIR 15A" of T-Motor.

The ESCs get their control commands from a microcontroller. The microcontroller is the head of the multicopter and runs an algorithm that can stabilize the attitude of the multicopter. This work goes in-depth with the stabilization in the chapters 7 and 8. This work chose the STM32G431KB of the manufacturer STMicroelectronics. The microcontroller is attached to the NUCLEO STM32G431, which provides all low-level components like the

debugger, power supply, USB connectivity.

The microcontroller needs to observe the attitude in order to issue the correct control commands. It has, therefore, various sensors connected, which provide realtime data about the multicopter attitude. The sensor principles are described in chapter Sensors. The quadcopter utilizes the ICM 20948 from the manufacturer InvenSense to observe attitude related data like acceleration and angular velocity. The ICM 20948 does also contain a magnetometer. However, the magnetometer is not used for this work. The last chapter explains the possible applications and problems of the magnetometer. The quadcopter also has an atmospheric pressure (GY-63) sensor and a GPS module (BN 880) attached, but both are not used during this work and are just mentioned for the sake of completeness.

Figure 4.1: This work will use the Spektrum DX6 remot control to issue control commands. The joystick on the left is used to control throttle, and yaw. The right joystick controls pitch and roll movements.

The receiver communicates with an external control unit and is connected to the microcontroller. The external control can then issue steering commands to alter the speed, height, or attitude of the multicopter, which the receiver will forward to the microcontroller.

A printed circuit board (PCB) interconnects all sensor-components, the GPS module, the receiver, and the ESCs with the microcontroller.

The body frame, which is the F330, holds all the components mentioned before. To reduce the vibrations that the ICM 20948 is exposed to, a foam material is used to hold the control unit. The motors are mounted onto rubber plates to reduce the vibrations. The later chapters explain why the damping is required and how the software can eliminate the vibrations almost entirely.

A battery supplies power to all components and is mounted to the multicopter as well. The battery is from the manufacturer Hacker and has a voltage of 11.1 volts and a capacity of 2400 mAh. Since the microcontroller units required a maximum of 5 volts, a step-down converter provides the 5V to the microcontroller.

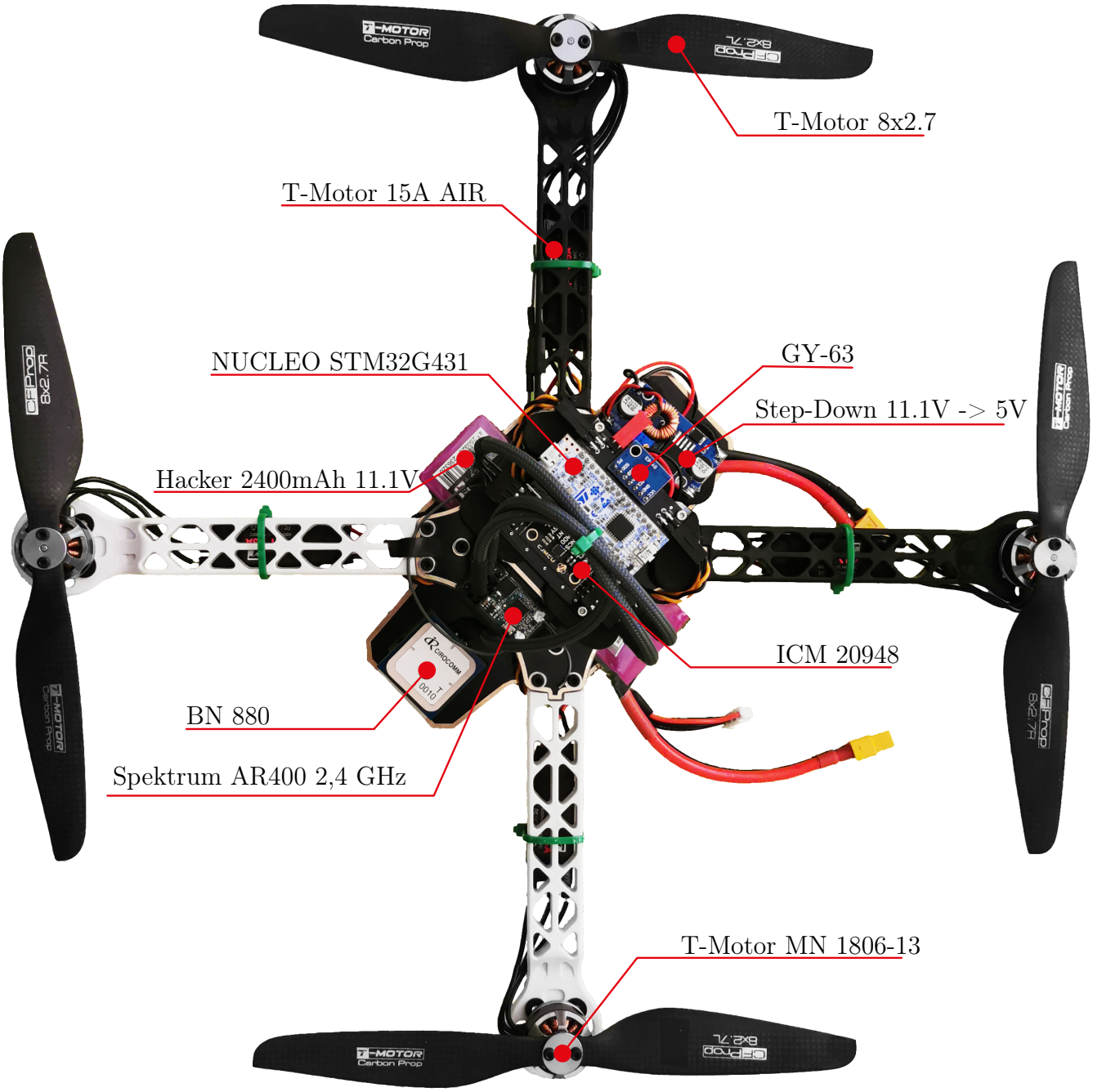


Figure 4.2: This image shows all components of the quadcopter that was built during this work.

Chapter 5

Quadrocopter Mathematic

5.1 Reference Frames

A quadrocopter operates inside a three-dimensional space and has six degrees of freedom. For this work, two coordinate frames are used, the world frame and the body frame.

The body frame is fixed to the aircraft. The z-axis is perpendicular to the plane spanned by the rotors. It is pointing towards the ground if the copter is standing on a flat surface. The airflow of the propellers follows the direction of the z-axis. The x-axis is pointing to the front. Since the coordinate frame is right-handed, the y-axis is pointing to the right side of the multicopter. The visualization in figure 5.1 shows the three axes.

The orientations of the axes of the world frame are equal to the aircraft frame when the aircraft is standing on a flat surface and facing north. The x-axis of the world frame is pointing north, and the y-axis is pointing eastward. The z-axis is parallel to the vector of gravity. The world frame origin is fixed to the earth reference frame and, therefore, stationary if the multicopter is moving.

5.2 Rotation

The following chapter provides an overview of 3d rotations. Representing rotations in a three-dimensional space is a complex task, and many methods have been developed. It will further explain the difference between intrinsic and extrinsic rotations.

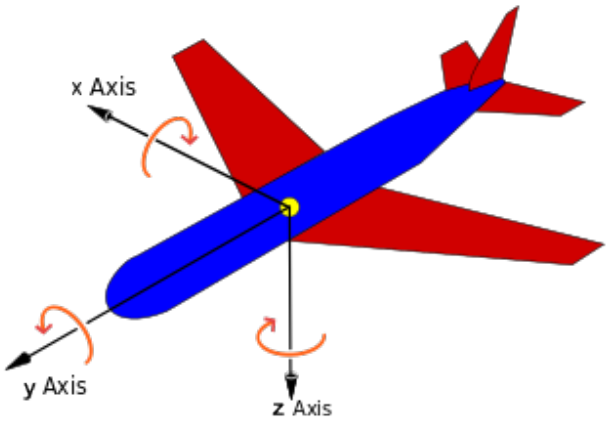


Figure 5.1: Visualization of the axes of aircraft. The reference frame is fixed with the aircraft. Source: `https://commons.wikimedia.org/wiki/File:Yaw_Axis_Corrected.svg` (edited), 2019-01-12

5.2.1 Conventions

The chapter 5.1 introduced different reference frames in the context of aircraft. The multicopter can move along its x-,y- and z-axis but also rotate around those axes. Rotations are defined with respect to the body frame. A rotation around the x-axis is named roll. Rotations around the y-axis are called pitch and cause the multicopter to accelerate forward. A z-axis rotation is called yaw. These rotation conventions apply to sea vehicles and spacecraft as well. Figure 5.2 provides a graphical representation of the three rotation axes.

5.2.2 Representation

There are multiple ways to represent rotations mathematically. The following chapter will present the different approaches, explains how each of them work and outlines some benefits and problems.

5.2.2.1 Axis-Angle

The axis angle method rotates a body by an angle of α around a known axis $\vec{u}$ [12]. The axis angle representation is an intuitive way of rotating an object in space. It can be proven that the axis angle approach allows finding

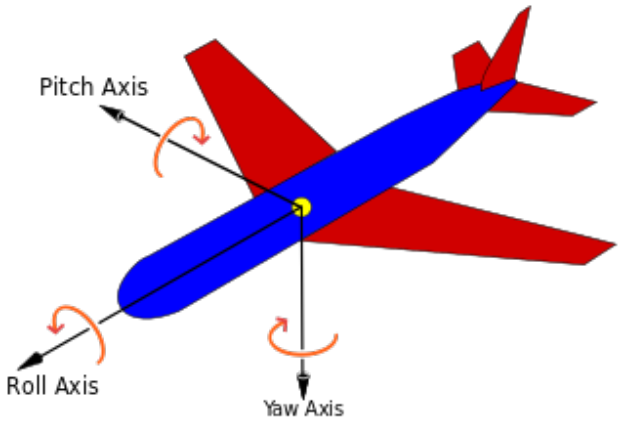


Figure 5.2: Visualization of the rotation axes of aircraft. Rotations are defined with respect to the body frame of the aircraft. Source: `https://commons.wikimedia.org/wiki/File:Yaw_Axis_Corrected.svg`, 2019-01-12

a rotation between all possible attitudes of an object [12]. That means an axis and an angle can represent every attitude of an object.

Axis-angle offers a parameterization of rotations, which is minimal in the number of parameters. However, the downside is that a rotation around some axis does not directly describe the resulting individual rotations around the reference frame axes. This information is needed to stabilize the multicopter.

5.2.2.2 Euler-Angles

Another approach is to preserve the individual rotations by expressing a compound rotation as three rotations around the reference frame axes. Euler-Angles describe a rotation as three distinct angles around the axes of the reference frame [12].

The critical point with Euler-Angles is that the rotations depend on the order they are applied. Euler-Angles apply each of the three rotations sequentially. The order in which the rotations are applied is called 'rotation order'. The rotation order is vital because a rotation with the same x, y, z angles, but different rotation orders will result in different attitudes. The rotation order is usually described by the names of the axes in the order they are rotated. The rotation order 'XYZ' makes the rotation around the x-axis to be applied first, second the y-axis rotation, and the z-axis rotation at last. A different rotation order (e.g., 'ZYX') results in a different pose of the object after the rotation was applied.

Further necessary is the type of Euler-Angles. A rotation with respect to the static reference frame is called extrinsic rotation. In the case of an aircraft, the static frame is the world frame. When rotating an object with respect to its axes, the rotation is called intrinsic. After an intrinsic rotation was applied, the attitude of the moving object changed, and any subsequent intrinsic rotations will be applied with respect to the mutated axes.

In the case of a multicopter, the sensors will only be able to capture intrinsic rotations. An extrinsic rotation can be observed by accumulating all intrinsic rotations. Both extrinsic and intrinsic rotations have to be rotation order aware.

5.2.2.3 Rotations Matrix

While Euler-Angles are a convenient way to store and talk about rotations intuitively, there is no direct way to apply them to a vector to rotate it. Since matrices can represent geometrical transformations, the rotation matrix is a convenient way to apply rotations.

Each of the three elementary rotations around the x, y, and z-axis can be represented as a rotation matrix, which results in a total of three matrices. The three single matrices can be combined by matrix multiplication - that way, a single matrix for any Euler angle combination can be found [12].

$$R_{combined} = R_x * R_y * R_z$$

As the chapter Euler-Angles already explained, the order of the elementary rotations is essential. Rotating an object in the X-Y-Z order results in a different attitude than the Y-X-Z rotation order. The non-commutativity of the Euler rotation is given as well by the rules of matrix multiplication.

$$AB \neq BA$$

$$R_x * R_y * R_z \neq R_y * R_x * R_z$$

The rotation matrix is a simple method to rotate a vector. Another critical point is that two rotations can be combined by simply multiplying their rotation matrices together.

$$R_{1-2} = R_1 * R_2$$

5.3 Steering

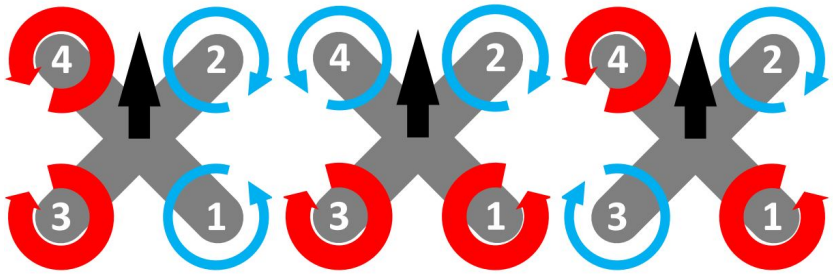


Figure 5.3: The black arrow marks the front of the quadcopter. A red spinning arrow visualizes high motor speed, a blue spinning arrow low motor speed. The image visualizes the impact of different motor speeds on the quadcopter attitude. The quadcopter in the left image will roll to the right side. The next will pitch forward, and the last will yaw leftwards around the z-axis. Source: [14]

Steering the multicopter to stabilize its attitude and account for turbulence is the central scope of this work. A multicopter is a simple vehicle when it comes to the engineering part. It has a limited amount of motors arranged around its body frame. Most multicopters have four, six, or eight motors and rotors to control their movements. Usually, multicopters with higher amounts of rotors have a more stable flight and can carry more weight. Each engine adds a specific amount of thrust and allows, therefore, higher amounts of load. Multicopters with six or more rotors are common in the area of agriculture and professional videography since they have to carry much weight with them. The further sections will discuss multicopters with four rotors, as already introduced in the chapter Building a Multicopter. This work will further use the word quadcopter since the following formulations are only valid for multicopters with four rotors. Platforms with a different number of motors require slightly different formulas.

Different motor speeds introduce attitude changes of the quadcopter since every motor with more speed than another motor will pull the quadcopter upwards. The following section will discuss how the quadcopter reacts to different motor speeds and how they are utilized to control the attitude.

5.3.1 Ascending and Descending

If the speed of all motors gets increased by the same value, the quadcopter will ascend since the propellers will generate more thrust. To sink the motor speeds are decreased.

5.3.2 Yaw Rotation

If the pilot likes to rotate the quadcopter around its z-axis, the speed of the motors on one diagonal axis is increased and decreased on the other axis. That increases and decreases the angular momentum on propellers rotating in the same direction. As a result, the quadcopter starts to spin in the same direction as the higher torque propellers are rotating. The quadcopter will not tilt on the x- and y-axis.

5.3.3 Roll Rotation

To tilt the quadcopter around the x-axis, a speed difference between the left and right side motors has to be induced. That way the quadcopter will lean on the side which has to lower torque. A direct result of the tilt is that the quadcopter will start moving in the direction it tilts, since the thrust does no longer act on the z-axis exclusively.

5.3.4 Pitch Rotation

A forward movement can be achieved by applying the different torques on the front and backside of the quadcopter. Depending on the tilt direction, the quadcopter will start to move either forward or backwards.

Chapter 6

Electronics

A quadrocopter requires a lot of electronic components hat need to work together in order to get from a static frame to a flying and controllable multicopter. A multicopter has to observe its state and issue controller commands to its actuators.

6.1 Sensors

The multicopter must know its current state at all times to ensure a stable flight. A sensor is a component which offers measurements of a specific physical property. Sensors do not have to be electronical components. Even before there were electronical components people knew the concept of sensors. By spanning a wire near above the ground they were able to be alerted when enemies or animals went by. A bell, functioning as an actuator, mounted to the wire rang when someone stepped onto the wire.

6.1.1 Accelerometer

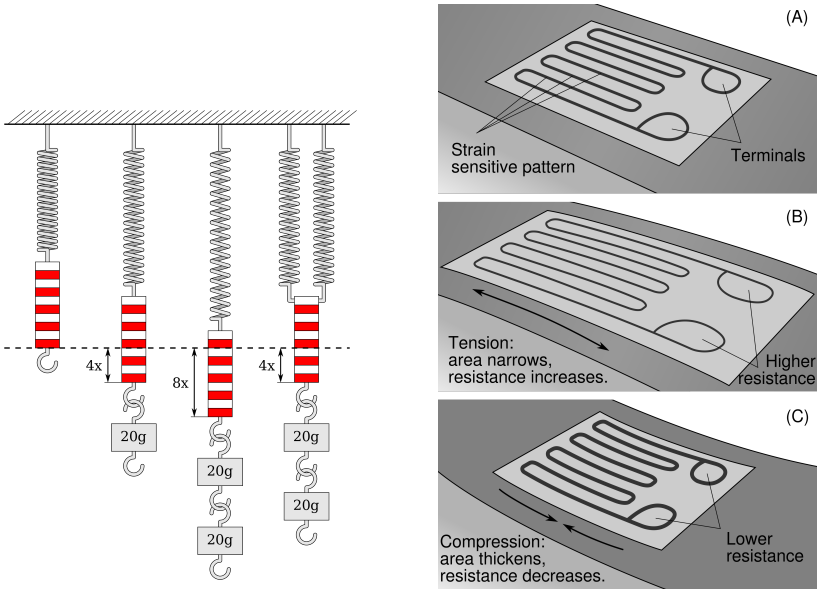
An accelerometer is a device which is used to measure acceleration which is a direct effect of an applied force. The multicopter is required to obtain its current attitude. By observing the force vector of gravity and calculating the angle between the orientation of the accelerometer and the gravity vector the attitude can be obtained. There are several approaches to measure force which should be introduced in this section.

There are classical tools using hooke's law which says that the force applied to a spring is proportional to the length it is stretching. Such tools are often

called dynamometer. They consist of a piston and a case where the piston is mounted to the case by a spring. When a linear force is applied to either the piston or the case and the counterpart part is mounted to a static object like a wall the spring stretches and the amount of force can be calculated by measuring the length at which the spring stretched and multiplying it by a constant factor (which is individual for each spring). Usually a dynamometer also has a scale, which already includes the multiplication and allows for directly reading the force. The principle is sufficient enough for many mechanical problems like analog kitchen scales. But when it comes to the application with electronics another solution is needed. A microcontroller is not able to read the analog dynamometer.

Another approach to measure force which also allows to read the measurements into a microcontroller is the so called load-cell [15]. A load-cell consists of a metal body made of aluminum and is therefore usually solid and frim. A load-cell contains strain gauges which are made of many very thin wires or a thin foil. Those cables are layed out in a grid pattern to increase the length of the wire or foil. When a force is applied to the gauge the strain gauge is stretched and the length of the wires increases (since the body of the gauge is minimally bent). Due to the now longer wires the resistance of the wires increases slightly [16]. The change of the resistance results in a higher voltage drop which is proportional to the force applied to the gauge. The voltage drop has to be amplified and can be read using an analog pin of a microcontroller. Strain gauges are commonly used by digital scales and have many other application areas in laboratories and the industy. Nevertheless load cells are not applicable to the problem of measuring the vector of gravity for multicopters. One reason for that is that they usually take up a lot of space. A better device to measure the vector of gravity is the MEMS accelerometer.

An accelerometer can measure static forces like gravity as well as dynamic forces (accelerations) like vibrations or motion. An important fact is that an accelerometer does not measure speed of a motion directly but the acceleration over time can be used to obtain the speed of the object where the accelerometer is attached to. A common type of accelerometer is the so called MEMS type [17]. The term MEMS means micro electro mechanical system and describes an electronical component manufactured using a microelectronic fabrication. A silicon structure with mass potential is suspended between two lanes of different potential. The air gap between the spring and each of the two fixed silicon lanes forms two capacitors which capacity can be measured. When an acceleration is applied the spring structure is displaced and the air gap between the fixed and spring part decreases on one side and



(a) Dynamometer (b) Load Cell

Figure 6.1: Comparison of the mechanical dynamometer and load cell.
Dynamometer (a): The spring stretches proportional to the amount of weight pulling it apart.
Load Cell (b): Strain gauges are placed in a zigzag pattern. When force is applied to one side of the load cell perpendicular to the strain gauges they stretch and the resistance is either increased or decreased.

increases on the other. The changed capacity can be measured and provides information about the magnitude of acceleration.

A MEMS accelerometer is a small and light device which can not only measure the acceleration on a single axis but also on multiple axes. Most MEMS accelerometers offer either two or three axes measurements. To determine the attitude of the multicopter we obtain a measurement of the accelerometer on all three axes. When the multicopter is hovering then the only acceleration which the accelerometer experiences is caused by gravity. When the multicopter is hovering parallel to the earth surface the acceleration on the z-axis equals $1g$ and the x and y axis each $0g$ since they are both perpendicular to the z-axis. When the multicopter is turned upside down then the z-axis gets the same magnitude but with a negative sign. When the copter gets rotated back 180 degrees the magnitude of acceleration changes depending on the rotation angle. But there is no difference whether the multicopter is rotated clockwise or counter clockwise. That is why two perpendicular

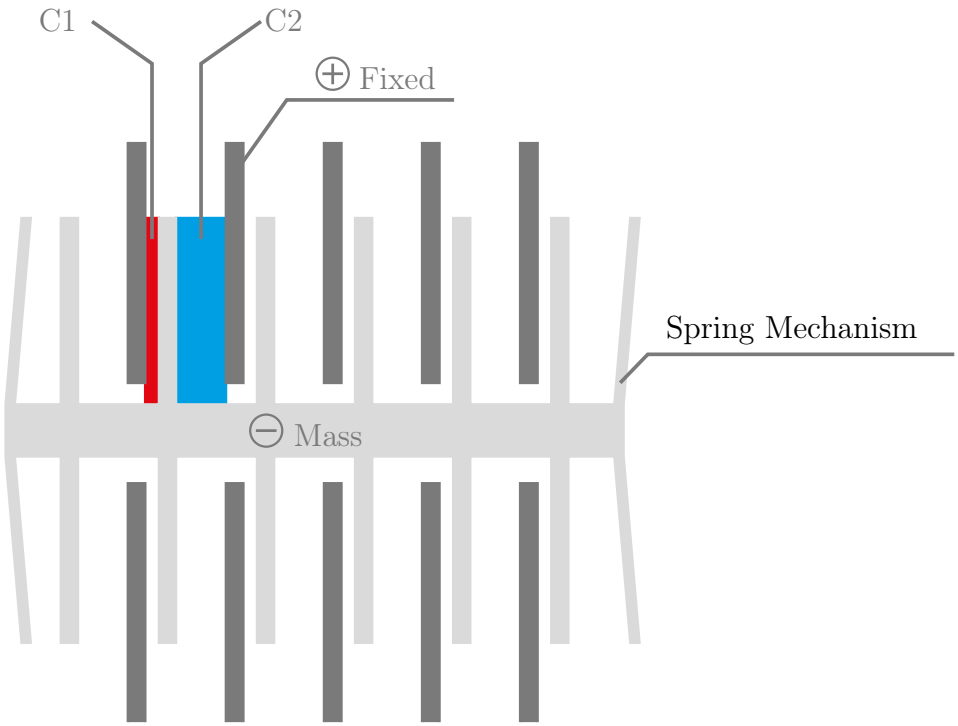


Figure 6.2: A MEMS accelerometer contains a proof mass suspended by a spring mechanism. When the proof mass experiences a force it moves between the fixed fingers that are positively charged. The gap between rotor and stator functions as a capacitor (red and blue areas) whose capacity is changed when the mass moves. The changed capacity is used to observe the amount of force applied to the accelerometer.

axes are needed to obtain the unique rotation angle of the multicopter. The acceleration measurements of two perpendicular axes can be used to distinguish the rotation direction. Therefore both measurements are interpreted as polar coordinates. Using the $atan2$ function the rotation angle of the polar coordinates can be calculated [18].

$$atan2(x,y) = \begin{cases} arctan(\frac{y}{x}) & x > 0 \\ arctan(\frac{y}{x}) + \pi & x < 0 \,\&\, y \geq 0 \\ arctan(\frac{y}{x}) - \pi & x < 0 \,\&\, y < 0 \\ +\frac{\pi}{2} & x = 0 \,\&\, y > 0 \\ -\frac{\pi}{2} & x = 0 \,\&\, y < 0 \\ undefined & x = 0 \,\&\, y = 0 \end{cases}$$

The $atan2$ function is a piecewise function that calculates the angle of the vector $(x\,y)^T$. The first condition accounts for all possible vectors in the first and fourth quadrant of the coordinate frame where the slope of the vector equals the tangens of the enclosing angle. Therefore the $arctan$ function returns the angle. When the vector falls into the upperleft quadrant $180\,deg = \pi\,rad$ have to be added and when the vector points into the lower left quadrant, $\pi\,rad$ have to be subtracted. The last 3 branches of the $atan2$ function account for edge cases where the vector aligns with the y axis or falls into the zero point in which case no angle can be calculated. The last case would apply when the multicopter is in free fall.

The open source multicopter Crazyflie 2 uses e.g. the accelerometer MPU-9250 of the manufacturer InvenSense [11].

It was shown, that the a MEMS accelerometer can be used to obtain the magnitude of gravity applied to a single axis of the accelerometer. If additionally the magnitude is known on all three axes, the rotation orientation of the multicopter can be calculated using the $atan2$ function.

6.1.2 Gyroscope

Another approach to observe the attitude of an object is by observing its angular rotation speed and integrating it over time. In order to observe the angular velocity of a rotating object, a gyroscope is required.

MEMS gyroscopes utilize the Coriolis effect [19]. The Coriolis effect is a force that acts on a moving object in a rotating reference frame. A famous example is an artillery that is mounted on a rotating plane and fires a projectile straight out of its barrel. Seen from the perspective of a person standing outside the rotating plane the ball is moving straight without any other force applied to it. However, seen from the perspective of the artilleryman, the projectile moves rightwards when the platform is rotating counterclockwise and moves leftwards if the platform rotates clockwise. From the viewpoint of the artilleryman, the projectile experiences an additional force perpendicular to the artillery. That force is called Coriolis force and is proportional to the angular velocity of the rotating platform.

A capacity change of a capacitor can measure the Coriolis force. One part of the capacitor is mounted to the case of the gyroscope, and the other is suspended on a spring structure. The structure is set to constant vibration, and when the gyroscope is rotating, the Coriolis force displaces that part of the capacitor and causes the capacity to change [19].

The benefit of using a gyroscope over an accelerometer to obtain the attitude of the multicopter is that the gyroscope experiences less noise. The following comparison shows both attitudes observed by the accelerometer and gyroscope.

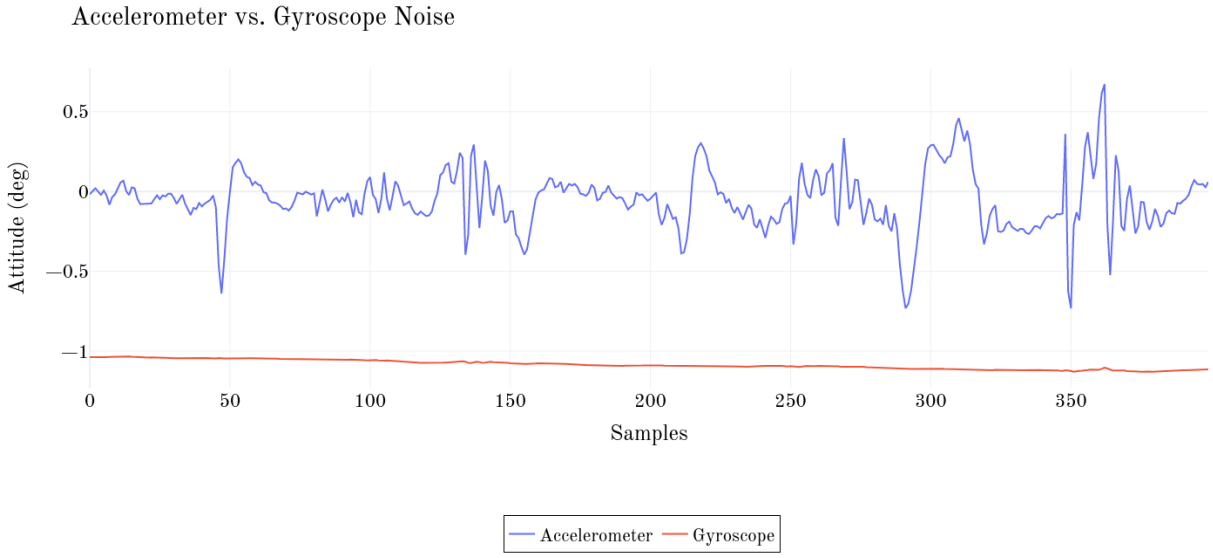


Figure 6.3: The graph shows data captured by the gyroscope and accelerometer. The gyroscope data was integrated over time to observe the rotation. The accelerometer has a much higher noise than the gyroscope. The multicopter was standing on a flat surface, but due to its drift, the gyroscope was detecting a small rotation.

A multicopter can tilt around the x and y-axis, called roll and pitch. Those movements are results of control commands by the pilot but will also occur when the multicopter has no control software applied and is hovering midair. In order to minimize those movements, software needs to know the current rotation of the multicopter and calculate steering commands. A gyroscope is a sensor that observes the angular velocity at which an object is rotating. The unit of this measurement comes as degrees per second. Gyroscopes can be either mechanical or electrical components where the mechanical ones work by a fast rotating wheel.

In order to observe the current orientation of the gyroscope, the angular velocity measurements have to be integrated over time. Gyroscopes tend to drift over time since the integration of the values over time is not accurate. Therefore another source is needed to align the integrated value with another ground truth. Take the apollo guidance system as an example. The inertial

measurement unit drifted about one milliradian per hour. In order to correct the drift, the pilots had to align the inertial unit with the stars.

6.2 Actuators

The quadcopter built during this work has four motors. This section provides a short overview of the basic concepts of electric motors. The main reason for this chapter is to explain the two types of electric motors and why the brushless (6.2.2) motor was chosen. For further information about each type, appropriate sources are provided.

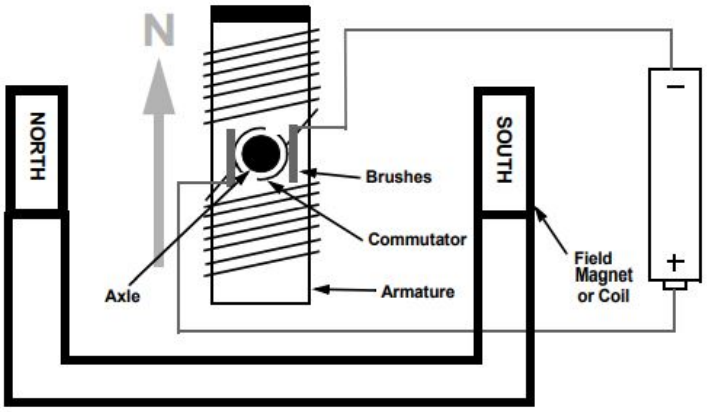


Figure 6.4: Illustration of a brushed motor. The armature rotates due to magnetic forces between the coil and the permanent magnets. The commutator will switch the polarity of the voltage every half rotation. Source: http://ww1.microchip.com/downloads/en/appnotes/00857a.pdf

6.2.1 Brushed Motors

A brushed motor contains a rotor, also called the armature. Wires wound around the armature form a coil. The rotor can rotate around the axis of the shaft. The coils create a magnetic field that interacts with the permanent magnets mounted to the housing around the shaft and pull the motor's rotor towards the permanent magnets. The coils get their power from a commutator mounted on the shaft as well. Since the whole armature rotates, it is not possible to connect wires directly to the coils. So-called brushes grind over the commutator to transfer the electricity to the coils. When the rotor gets aligned with the magnets, the commutator has rotated just far enough to

switch the polarity of the voltage. Since the current flows through the coils in the opposite direction, the magnetic field gets inverted as well. The rotor will continue to rotate.

A brush motor is simple and does not require sophisticated control electronics. It is, therefore, cheap and easy to regulate. Nevertheless, the brushes and the commutator wear out and have to be replaced after some time.

For more detailed information about brushed motors, please contact [20].

6.2.2 Brushless Motors

A brushless motor has one significant difference to brushed motors. The rotating part of a brushed motor is the coil, whereas the brushless motor has fixed coils but rotating magnets. The coils of a brushless motor are mounted either to the housing or the shaft. If the coils are mounted to the shaft, the housing is the rotating part. The propellers will then be mounted to the rotating housing. This principle makes the brushes and the commutator redundant since there is no need to get the energy to a moving part. An external component will alter the polarity of the supply voltage just as the brush-commutator combination does for the brushed motor [21].

The benefit of a brushless motor over a brushed motor is that it has a longer life since the brushes can not wear out. A brushless motor does also has a higher efficiency than a brushed motor [22]. The downside of brushless motors is that they need more advanced speed controllers since they require the polarity of the supply voltage to be precisely timed [22].

The built multicopter uses brushless motors due to the presented benefits.

6.2.3 ESC Controller

An Electronic Speed Control regulates the speed of motors. Since the multicopter uses brushless motors to drive the propellers, only the functioning principle of brushless ESCs will be explained.

A brushless ESC regulates the motor speed by controlling the polarity of the supply voltage to the motor. A three-phase brushless motor requires the polarity to be changed six times per full rotation. Precise timing is vital for the brushless motor to function correctly. There are two ways to switch the polarity.

Hall effect sensors are placed across from the rotating magnets. Each time a magnet passes the hall effect sensor, the sensor will notify a microcontroller. The microcontroller can then switch the polarity.

Motors without hall effect sensors are cheaper but require a different mechanism to recognize when to switch the polarity. Since a three-phase brushless motor has only voltage on two out of three lines at certain time, the third one is floating. When a coil which is currently on a floating line passes a magnet, an inductive voltage can be measured and taken as a signal to switch the polarity. The floating line changes every 60 degrees.

An ESC is usually controlled by PWM (6.3.2) signals sent from an external microcontroller. An ESC allows a microcontroller to precisely control the speed of the brushless motors, without knowing the details of directly altering the polarity of the motors supply voltage.

6.3 Communication Technologies

A multicopter should be controllable by a pilot using a normal remote control. It further utilizes various sensors (6.1) and has to regulate the motor speed (6.2). All components need to work together and exchange data. This chapter will explain the various communication techniques used by the electronic components. The chapter Multicopter Control Software will outline how the different communication channels are implemented by a common microcontroller.

Communication can be divided into internal and external. The external communication is issued from outside the multicopter. That is, for example, the remote control of the pilot. Another example of external communication is live video data sent from the multicopter to a video receiver.

6.3.1 2.4 GHZ

The remote control (Figure 4.1) communicates with a receiver using the 2.4 GHZ wireless range. The frequency bands 27 MHZ, 35 MHZ, 40 MHZ, and 72 MHZ are nowadays very rarely used [23].

The details of the 2.4 GHZ communication depend on the manufacturer and are not standardized. This is the reason why this work will not go into greater detail.

The 2.4 GHZ signals are received by a receiver and put out as PWM signals. This work will continue with the specifics of PWM transmissions.

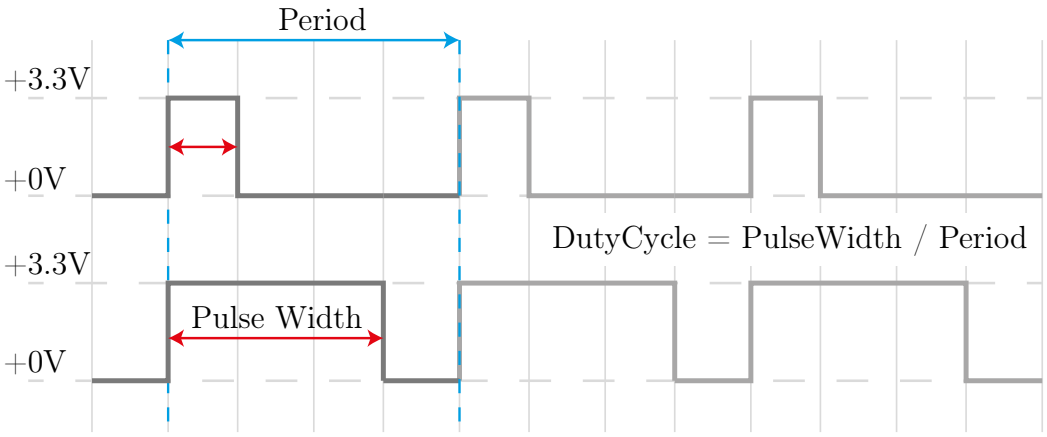


Figure 6.5: The period defines how fast the transmission of the data will be. It is the amount of time from rising edge to rising edge. It is constant during the transmission. The pulse width encodes the data to transmit. The proportion of the high level in relation to the length of the period is called duty cycle.

6.3.2 PWM

Reading inputs from the receiver and controlling the speed of the motors is done by PWM signal. PWM means 'Pulse Width Modulation'. It is a digital signal transmission technique which is widely used in the industry to control various components [24].

A PWM signal consists of three main components. Its operating voltage is the voltage level at which the PWM transmitter decodes a digital 1. The pulse width encodes the data to transfer. It is an amount of time the signal stays high. After the pulse width is over the signal turns back to low. The period defines how long a pulse width can be at its maximum. It is therefore an amount of time which specifies how fast the transmission of the data is. The term duty cycle refers to the proportion of the pulse width in relation to the length of the period [24] [25].

$$dutyCycle = \frac{t_{pulsewidth}}{t_{period}}$$

The frequency f of the signal defines how many transmissions per second are done.

$$f = \frac{1}{t_{period}}$$

The receiver sends out a PWM signal with a period of $t_{period} = 20ms$ and a pulse width between $1100us$ and $1900us$. The pulse width is proportional to the stick positions on the remote control. The PWM signals can be read by a microcontroller as explained in the chapter Read Single PWM Signals.

The motor speed is regulated by the ESCs which in turn are controlled by PWM signals. The details of the PWM controller are described in the chapter PWM Generation.

6.3.3 I2C

The I2C bus is a bus system widely used in microcontroller setups to establish a communication channel between different components. The I2C bus uses two lines SCL (Clock) and SDA (Data). Due to its design, the I2C bus is a cheap bus system that is a popular choice on lower speed peripherals. The I2C bus system usually operates at the speeds 100 kHz and 400 kHz [26].

The I2C bus is used to communicate with the ICM-20948 (accelerometer, gyroscope, and magnetometer). The chapter I2C Communication will explain how the microcontroller communicates with the sensor device.

6.3.3.1 Electric Specifics

The I2C bus lines carry the clock and data signal. Both lines require a pull-up resistor to either +3.3V or +5V, depending on the specification of the attached peripherals [26]. An I2C bus can have multiple peripherals attached. Each peripheral can pull-down the bus to signal its intention to start a transmission. Usually, only the master will start a transaction. The I2C buses used during this work will only use one bus master. Multi-master setups are possible but require specialized hardware since the masters have to coordinate their transactions [27].

6.3.3.2 Peripheral Address

Any peripheral attached to the bus needs an address to be contactable by the bus master. The device address is 7-bits long what makes it possible to drive up to 2^7 devices per bus [28].

Some electrical specifics might reduce the number of possible devices. The primary limiting factor next to the limited address range is the pull-up resistors included on breakout boards. Most breakout boards do already include

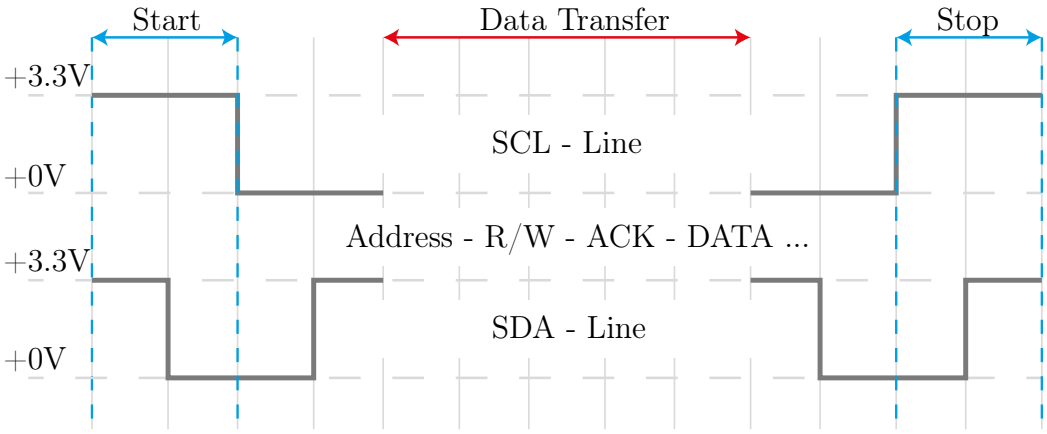


Figure 6.6: A low bit on SDA while the SCL line is high signals a transaction start. Similarly, a low bit on the SDA line while SCL is low ends the transmission. Start and stop bits are sent by the bus master [26].

pull-up resistors on both I2C lines. Multiple breakout boards connected to the same I2C bus will cause the total resistance to grow. A high total resistance will cause too much brownout, and the other devices wont recognize the bus signals.

6.3.3.3 Communication Protocol

The I2C communication protocol controls how the attached devices are allowed to communicate with each other. As already explained, the master starts a transaction with another peripheral.

To start a transaction, the master pulls the SDA line low while the SCL line is high. This bit signals the starting condition [26]. The next bits will contain the 7-bit device address of the target device. Data is transferred by sending the most significant bit first. When the master finishes the transmission of the 7-bit device address, it sends another bit indicating either a read/write operation. The slave with the specified address will answer with an acknowledge (ACK) bit (low). If no device exists with the specified address or the device fails to respond, the lines will go high due to the pull-up resistors. After the slave acknowledged the address, the master will start to send the register-address of the target register. Once the register-address is sent, the device will again respond with an ACK-bit. Depending on the operation (read/write), either the peripheral will start transmitting the data of the selected register, or the master will send the data to write into the selected register. At the end of the transaction, the master will send a stop-bit

that is identified by pulling SDA low while SCL is low and releasing SDA when SCL went high again [26] [27].

6.3.3.4 Auto Increment

In order to optimize the throughput, some I2C devices support an auto-increment mode that continues to send data in reading mode after the transmission of the requested byte completed. This mode saves a new transaction start, the transmission of the device address, and the target register [26].

Chapter 7

Control Theory

7.1 Filter

7.1.1 Moving Average

The moving average is a simple filter method to smooth out noise or small variations from a continuous time series. It can be used whenever the signal has to be smooth for further processing. That is the case for the data from the accelerometer. The accelerometer outputs a noisy signal, as can be seen in figure 8.5, 7.1. We can smooth that signal using a moving average filter.

The moving average filter calculates for each new measurement the average of the last N measurements, including the latest measurement. The moving average has one parameter for optimizing the results as needed by the use case.

$$X_i = \frac{1}{N} \sum_{j=i-(N-1)}^{i} (x_j)$$

The number of measurements N to include in the average calculation determines how fast the filter will adopt new trends in the data series. This setting is crucial, as the following section will discuss.

A low N offers what is called a quick filter response. That means that the output of a filter reflects changes in the input signal fast. For low N that is the case because the filter relies only on the latest data.

A high value for N results in a delayed filter response. That is the case because new data points weigh much less than previous data.

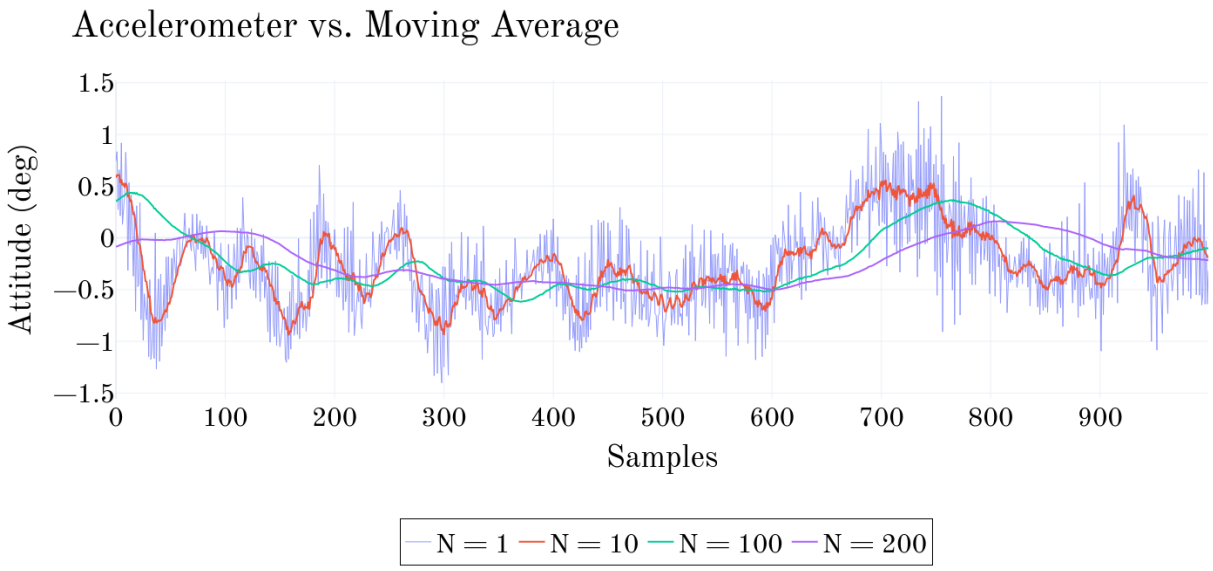


Figure 7.1: This graph shows the tradeoff between fast filter response (red line) and smooth output (purple line). The legend shows the number of samples N that were used to calculate the corresponding moving average. The N=10 moving average graph does still contain much noise. The attitude angle starts to grow at sample 600. The underlying signal reaches the peak at sample 700. While the red lined graph responds to the rise fast, the green and purple lines have a delay of 50, respectively 100 samples.

Both have benefits and downsides, and the parameter has to be tuned to achieve the optimal performance per usecase. The benefits of a delayed filter response (high value for N) is that it performs better at smoothing out noise. The downside is that it makes it impossible to react fast to changes since it takes the filter some time to adapt to real-world changes, see figure 7.1 for a visualization. That is impractical for a multicopter stabilization system, which requires a realtime observation of its attitude.

7.1.2 Kalman Filter

Because a moving average filter is a simple and computationally inexpensive method for filtering an input signal with random noise it is also performing poorly in the case of accelerometer data. The figure 7.1 shows that smooth output comes at the cost of delayed filter response.

This work will now present a filter mechanism which improves the filter response time by reducing the noise at the same time. This work will therefore

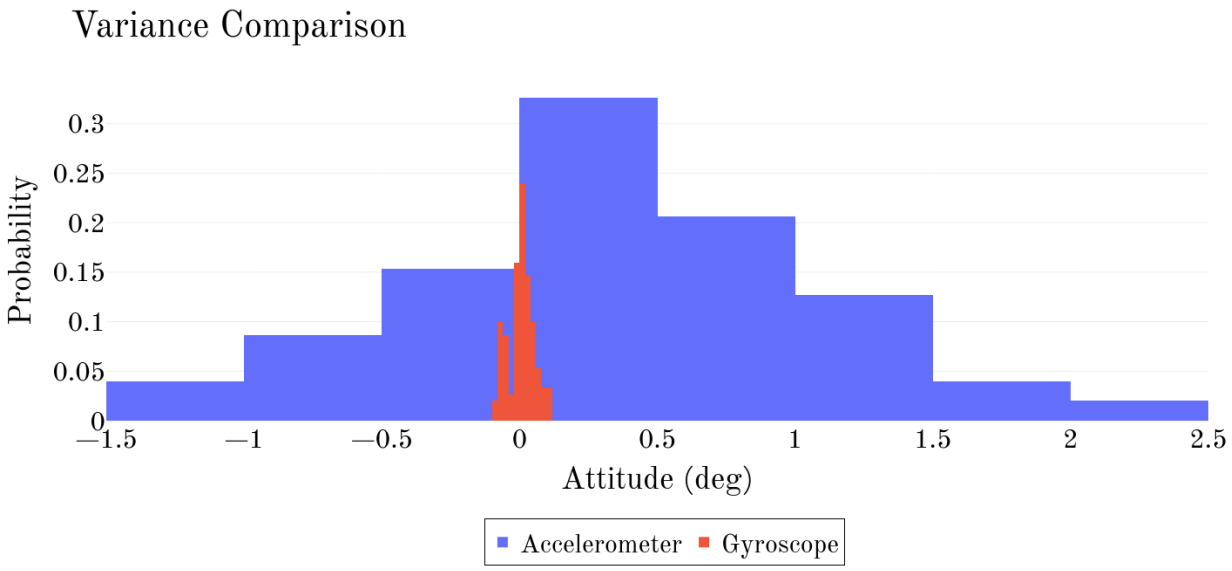


Figure 7.2: This diagram shows the variances of the accelerometer and gyroscope as a histogram. There are two critical things to notice. First, the accelerometer has a much higher variance than the gyroscope. Second, both measurements seem to be of Gaussian nature. The gaussian character of the signals is a requirement for the application of the Kalman filter.

introduce the Kalman filter, a widely used algorithm to process multidimensional signals. This chapter will focus on the theoretical and mathematical details as well as the applicability to multicopters. The later chapter IMU Filter Algorithm will further discuss which filter algorithm was used for the multicopter. For a detailed derivation of the Kalman filter please contact the following sources [29] [30] [31].

7.1.2.1 Gaussian Random Variables

The Kalman filter relies heavily on gaussian random variables which is the reason for this short introduction. A gaussian random variable $X \sim \mathcal{N}(\mu, \sigma^2)$ describes a value by specifying its expected value, also called mean μ and variance σ^2 [32].

An infinite number of observations of X have a mean value of μ. It can be described as well as that value which the variable X is most likely to have.

The variance σ^2 as well as the standard deviation $\sigma = \sqrt{\sigma^2}$ are a measurement for the uncertainty of the mean value μ. The bigger σ^2 the bigger the uncertainty.

A gaussian random variable can be visualized by its probability density function (PDF) $f(x)$ [32].

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{7.1}$$

The area under a specific range of the PDF describes the possibility that an observation falls within that range. The PDF has a max value at μ since that is the value an observation of X is most likely to have. The PDF of a random variable X with a high variance σ is wider than for a random variable with a small variance.

We assume the noise of an sensor to be gaussian. That means we can describe the accelerometer sensor output as normal distributed random variable X_{accel} with a mean value μ_{accel} and a variance of σ_{accel}.

$$X_{accel} \sim \mathcal{N}(\mu_{accel},\, \sigma^2_{accel}) \tag{7.2}$$

Another step towards more advanced filtering of the accelerometer data can be achieved by incorporating values of the gyroscope. The gyroscope provides information about the angular velocity it is rotating. The gyroscope output is another gaussian random variable.

$$X_{gyro} \sim \mathcal{N}(\mu_{gyro},\, \sigma^2_{gyro}) \tag{7.3}$$

7.1.2.2 Process Model

The gyroscopes angular velocity is the first derivative of the rotation angle. Integrated over time we can track the rotation of the multicopter.

$$\int angular\_velocity(t)dt = rotation\_angle \tag{7.4}$$

Since the continous world can only be captured by discrete obervations, we are effectively adding up gaussian random variables (multiplied by time).

$$\sum_{t=0} x_{gyro,t} * \Delta t \simeq rotation\_angle \tag{7.5}$$

This results in a constantly growing uncertainty since the variance of the sum of a gaussian random variables equals the sum of the variances.

$$X_{rot,i+1} = X_{rot,i} + X_{gyro,i} * \Delta t_i \tag{7.6}$$

$$\mu_{X_{rot,i+1}} = \mu_{X_{rot,i}} + \mu_{X_{gyro,i}} * \Delta t_i \tag{7.7}$$

$$\sigma^2_{X_{rot,i+1}} = \sigma^2_{X_{rot,i}} + \sigma^2_{X_{gyro,i}} * \Delta t_i \tag{7.8}$$

This neglects the fact that there is also some noise in the process since we are integrating discrete values. Time is probably not measured correctly as well. We will name this uncertainty process noise Q.

$$Q \sim \mathcal{N}(\mu, \sigma^2) \tag{7.9}$$

The process noise is white noise. The mean μ of white noise is 0 since it only affects the variance. The correct formulation of the state update is therefore.

$$X_{rot,i+1} = X_{rot,i} + X_{gyro,i} * \Delta t_i + Q \tag{7.10}$$

$$\sigma^2_{X_{rot,i+1}} = \sigma^2_{X_{rot,i}} + \sigma^2_{X_{gyro,i}} * \Delta t_i + \sigma^2_Q \tag{7.11}$$

The mean stays unchanged since the process noise Q has a mean value of 0. Its only purpose is to increase the variance.

If the equation 7.6 gets evaluated multiple times, the variance of the state variable X_{rot} will increase over time due to equation 7.11. Figure 7.4 shows a direct comparison of the attitude computed by accelerometer and gyroscope. It can be seen that the attitude computed by the gyroscope diverges from the accelerometers measurement.

Before the growing error is addressed, we define the Kalman state and the first part of the two-step Klaman filter process.

7.1.2.3 Kalman State

The equation 7.6 is called process model. Its purpose is to model a dynamic process based on a linear relationship of different state variables.

A state variable in terms of the Kalman filter is represented by multiple gaussian random variables that describe the state of a dynamic system. A vector $\vec{x_{mult}}$ holds the mean values of the different random variables and a matrix holds the covariances P_{mult}. The matrix is called covariance matrix. The diagonal entries of the covariance matrix describe the variances of the different gaussian random variables. The remaining entries describe the covariance between the different random variables [33] [32]. The interpretation of the covariance is not relevant to this work.

$$\vec{x_{mult}} = \begin{bmatrix} \mu_{rot} \\ \mu_{av} \end{bmatrix} \tag{7.12}$$

$$P_{mult} = \begin{bmatrix} \sigma_{rot}^2 & 0 \\ 0 & \sigma_{av}^2 \end{bmatrix} \tag{7.13}$$

μ_{rot} describes the expected attitude of the multicopter and μ_{av} refers to its current angular velocity. σ_{av}^2 describes the variance of the current angular velocity of the multicopter as well as σ_{rot}^2 describes the same for the current rotation.

7.1.2.4 Kalman Predict

The process model which was defined in the section Process Model describes how the angular velocity can be integrated over time to obtain the attitude of the multicopter. Then the section Kalman State introduced the Kalman state as gaussian random variables with an exptected value of $\vec{x_{mult}}$ and the covariance matrix P_{mult}.

The first step of the two-step Kalman filter process is to calculate the prior. The prior is based on the process model and the Kalman state. Since the Kalman filter is a linear estimator a matrix F_{mult} can be used to describe the process model. The multicopter process model as described in equation 7.6 can be describes as matrix as follows.

$$F_{mult} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \tag{7.14}$$

The mean of the prior can be calculated as the matrix product of $\vec{x_{mult}}$ and F_{mult}.

$$\vec{x'_{mult}} = F_{mult} * \vec{x_{mult}} \tag{7.15}$$

This is equivalent to equation 7.7.

Similarly, the variance of the predict step of one-dimensional equation 7.11 transforms into the following matrix operation. For a detailed derivation see [29] [30] [31]. For this work, it is only relevant, that during the predict step, the mean of the prior is calculated using the current state and a process model and the covariance of the prior increases since no ground truth is added to the system.

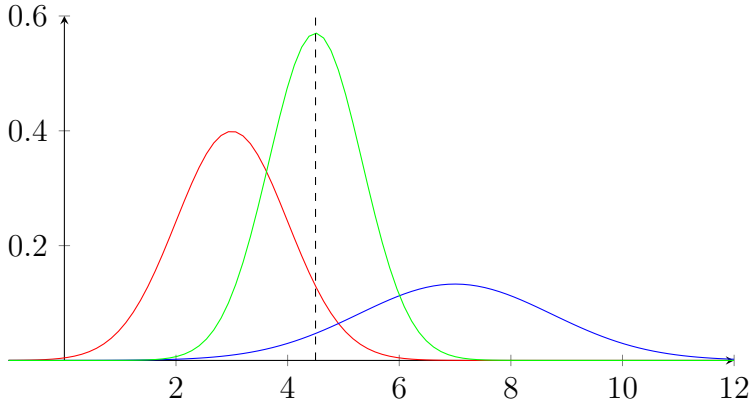
$$P'_{mult} = F_{mult} * P_{mult} * F^T_{mult} + Q \tag{7.16}$$

The result of the predict step is the prior mean $\vec{x'_{mult}}$ and the prior covariance P'_{mult}.

7.1.2.5 Gaussian Multipication

To counteract the growing error of the predict step, we are required to integrate another source of data. Even though the variance of the accelerometer is worse than that of the gyroscope, it can be used to improve the state prior X'. This step is called update step and incorporates measurements to improve the prior.

The intuition behind the Kalman filter update step is that two independent random variables which both are affected by a specific (Gaussian) noise can be fused to receive a variable, more precise than each individual variable. The following figure shows how two Gaussian random variables can be combined in order to obtain a Gaussian, which is more accurate. The red and blue curves represent the probability density function of the original measurements, and the green one represents the result when blue and red are fused together.



Even though the blue measurement has a much higher variance than the red one, it adds more information into the system and contributes to a reduced variance of the green curve. Mathematically what happens is a multiplication of the two probability density functions and normalizing them afterward. The mean of the green curve is the weighted average of the blue and red curve. The weights used to calculate the average are the variances of the red and blue random variables. The intuition is that data sources with a low variance

are more accurate and should, therefore, get more weight when calculating the new mean value.

To provide a mathematical formulation of the intuition described above, we start by defining two Gaussians.

$$X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$$
$$X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$$

The result of their multiplication is another gaussian random variable.

$$X = X_1 * X_2; X \sim \mathcal{N}(\mu_x, \sigma_x^2)$$

It can be shown that the mean and the variance of the product can be calculated as follows [29] [30] [31].

$$\mu_x = \frac{\sigma_1^2 * \mu_2 + \sigma_2^2 * \mu_1}{\sigma_1^2 + \sigma_2^2}$$

$$\sigma_x^2 = \frac{\sigma_1^2 * \sigma_2^2}{\sigma_1^2 + \sigma_2^2} \tag{7.17}$$

The calculation of the mean value μ_y can be transformed into the following expression.

$$\begin{aligned} \mu_x &= (\frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}) * \mu_2 + (\frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2}) * \mu_1 \\ &= (\frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}) * \mu_2 + (1 - \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}) * \mu_1 \\ &= K * \mu_2 + (1 - K) * \mu_1 \end{aligned} \tag{7.18}$$

$$K = \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} \tag{7.19}$$

Equation 7.18 shows that the mean of the product of two Gaussians is a weighted average. The weight K (equation 7.19) is based on the variance of the factors X_1, X_2. An important fact is, that the weight of μ_1 is based on σ_2^2 This reflects the intuition that the variable with the lower variance has a greater impact on the final result. That is because the variable with the

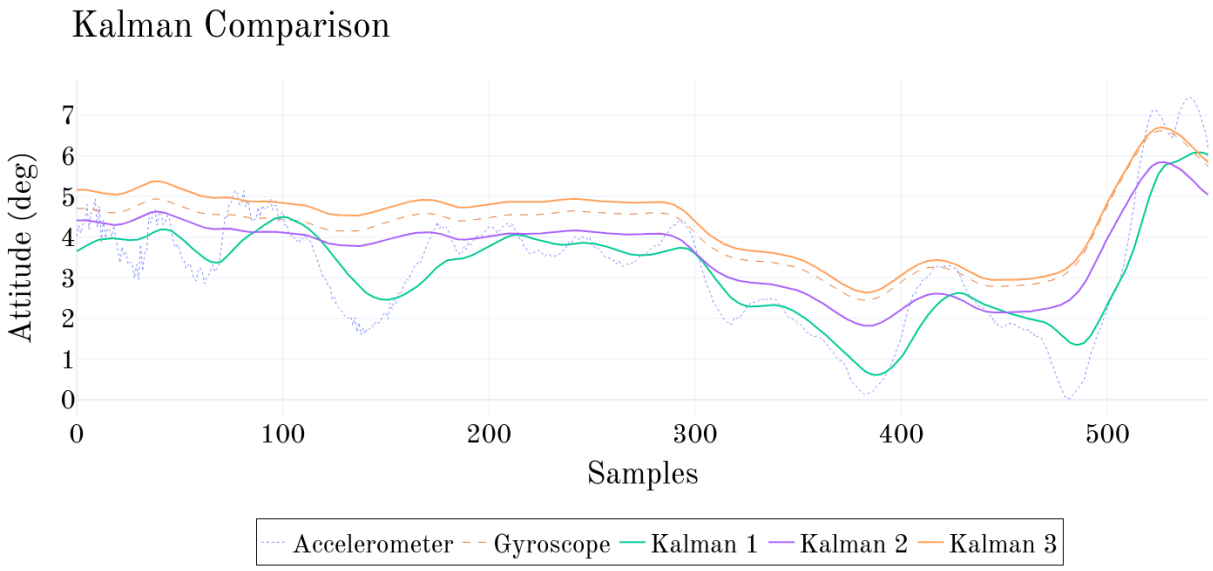


Figure 7.3: This diagram shows the effects of different Kalman gain values. The accelerometer and gyroscope lines show the sensor measurements. The gyroscope line is the raw integration of the gyroscope readings without any filter applied to it. It can be seen clearly that the gyroscope integration yields a specific error as it is offset from the accelerometer line. The Kalman filter output 'Kalman 1' still contains a lot of noise since the gain is high. The gain of 'Kalman 3' is too low, and the filter output drifts. The gain of 'Kalman 2' is high enough to prevent drift but also small enough to avoid sharp noise.

lower variance is more confident about its mean value than the variable with the higher variance.

If both variances are the same $\sigma_1^2 = \sigma_2^2$ then equation 7.18 simplifies to $\mu_x = 0.5 * \mu_2 + 0.5 * \mu_1$. The Gaussian multiplication can be used to combine two Gaussian random variables that describe the same property of a system in order to get a more precise Gaussian of the property.

7.1.2.6 Kalman Gain

The Gaussian multiplication is a weighted average computation where the weights depend on the variances of the variables (equation 7.18). The weight can be defined using the single variable K (equation 7.19). The Kalman gain is a single variable (in the one dimensional Kalman filter) and a matrix (in the multi dimensional Kalman filter) which defines how much the measurement drags the prior towards its value. The common formulation of the

Kalman gain update process can be derived by two simple transformations on equation 7.18.

The first step is to define the residual y as the difference between μ_2 and μ_1. In terms of the Kalman filter the residual is the difference between prior and measurement.

$$y = \mu_2 - \mu_1 \tag{7.20}$$

The second step is the substitution of μ_2 with $\mu_2 = y + \mu_1$ which yields the following equation.

$$\begin{aligned} \mu_x &= K * (y + \mu_1) + (1 - K) * \mu_1 \\ &= K * y + K * \mu_1 + \mu_1 - K * \mu_1 \\ &= \mu_1 + K * y \end{aligned} \tag{7.21}$$

The variance of a Gaussian product can be defined in terms of the Klaman gain K.

$$\begin{aligned} \sigma_x^2 &= \frac{\sigma_1^2 * \sigma_2^2}{\sigma_1^2 + \sigma_2^2} \\ &= \frac{\sigma_1^2 * \sigma_1^2 + \sigma_1^2 * \sigma_2^2 - \sigma_1^2 * \sigma_1^2}{\sigma_1^2 + \sigma_2^2} \\ &= \frac{\sigma_1^2 * \sigma_1^2 + \sigma_1^2 * \sigma_2^2}{\sigma_1^2 + \sigma_2^2} - \frac{\sigma_1^2 * \sigma_1^2}{\sigma_1^2 + \sigma_2^2} \\ &= \frac{\sigma_1^2 * (\sigma_1^2 + \sigma_2^2)}{\sigma_1^2 + \sigma_2^2} - \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} * \sigma_1^2 \\ &= \sigma_1^2 - \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} * \sigma_1^2 \\ &= \sigma_1^2 - K * \sigma_1^2 \\ &= (1 - K) * \sigma_1^2 \end{aligned} \tag{7.22}$$

7.1.2.7 Kalman Update

When the prior was calculated as described by the chapter Kalman Predict, the Kalman filter needs to integrate a measurement to counteract the growing

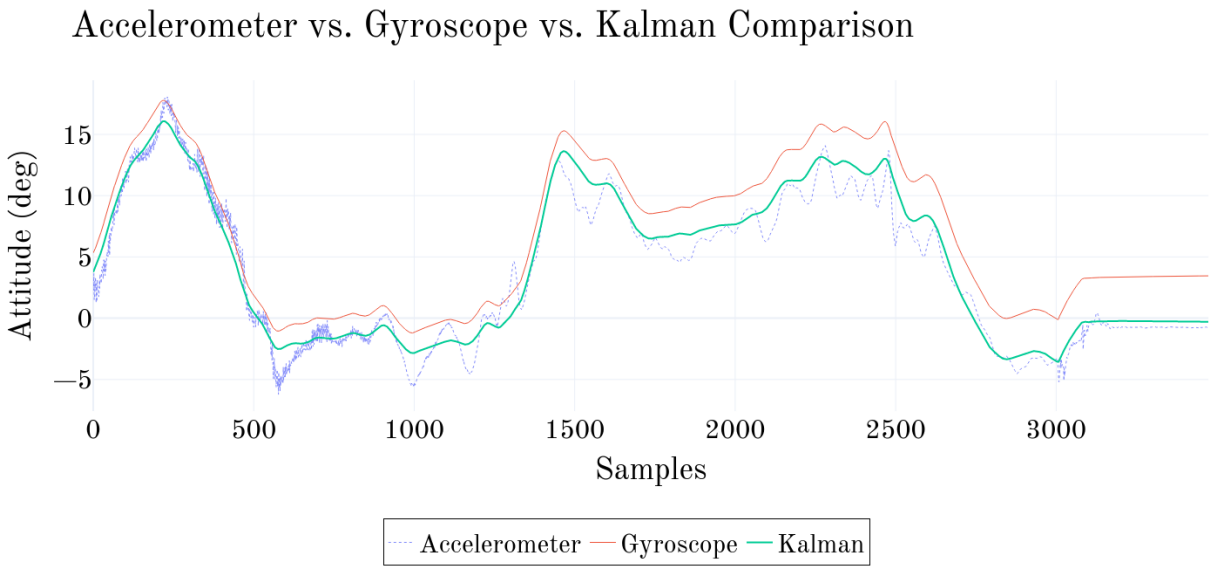


Figure 7.4: This graph compares the raw accelerometer and (integrated) gyroscope data with the Kalman filter output. It can be seen that the gyroscope output started to drift since the red line has 4 degrees offset from the ground truth accelerometer data. The accelerometer output contains much noise compared to the gyroscope data. The Kalman filter offers a smoothed output without the drift of the gyroscope. Source: Screenshot

variance which would occur as a result of equation 7.11. The measurement Z is another vector of gaussian random variables which usually has the same shape as the Kalman state 7.1.2.3. For this work the shapes of the measurement and state are equal but this is not demanded. If the shapes are not equal the measurement function H converts between measurement and state space [31]. This work will neglect the measurement function. The shape of the measurement is therefore.

$$\vec{z_{mult}} = \begin{bmatrix} \mu_{rot} \\ \mu_{av} \end{bmatrix} \tag{7.23}$$

The value μ_{rot} is the measurement from the accelerometer and the measurement μ_{av} is obtained from the gyroscope. To describe the uncertainties of the measurement we use a covariance matrix just as for our state. We name the covariance matrix of the measurement R.

$$R_{mult} = \begin{bmatrix} \sigma_{rot}^2 & 0 \\ 0 & \sigma_{av}^2 \end{bmatrix} \tag{7.24}$$

To apply the Gaussian multiplication to prior x'_{mult}, P'_{mult} and measurement z_{mult}, R_{mult} the equations 7.20, 7.19, 7.21, and 7.22 have to be transformed into Kalman notation. Further the equations need to be converted to matrix mathematics.

The one dimensional residual (7.20) transforms into the following.

$$\vec{y_{multi}} = \vec{z_{multi}} - \vec{x'_{multi}} \tag{7.25}$$

The multi dimensional Kalman gain can be computed using the covariance matrices [29] [30] [31]. The term $()^{-1}$ denotes the inverse matrix.

$$K = P'_{multi} * (P'_{multi} + R_{multi})^{-1} \tag{7.26}$$

The update of the prior with the residual (7.21) transforms to the following formula. The result is the new posterior.

$$x_{multi} = x'_{multi} + K * y_{multi} \tag{7.27}$$

The variance update of the one dimensional formula (7.22) becomes the following covariance update of the multi dimensional Kalman filter. The term I denotes the identity matrix.

$$P = (I - K) * P'_{multi} \tag{7.28}$$

7.1.3 Complementary Filter

The Kalman filter is a complex and powerful filter mechanism. Due to its complexity, the Kalman filter is difficult to tune for optimal performance. The later chapter 8.7.1 will discuss the details.

The complementary filter is more simple than the Kalman filter and takes a more straightforward approach. While the Kalman filter relies heavily on Gaussian random variables for its predict and update steps, the complementary filter uses a slightly adapted weighted average for filtering.

The complementary filter does not require the calculation of an inverse matrix nor multiple matrix multiplications, making it computationally easier.

The Kalman gain controls how much the measurement affects the prior to filter out the measurement noise. If all input variances of the Kalman filter (measurement noise, process noise) are constant, then the one dimensional

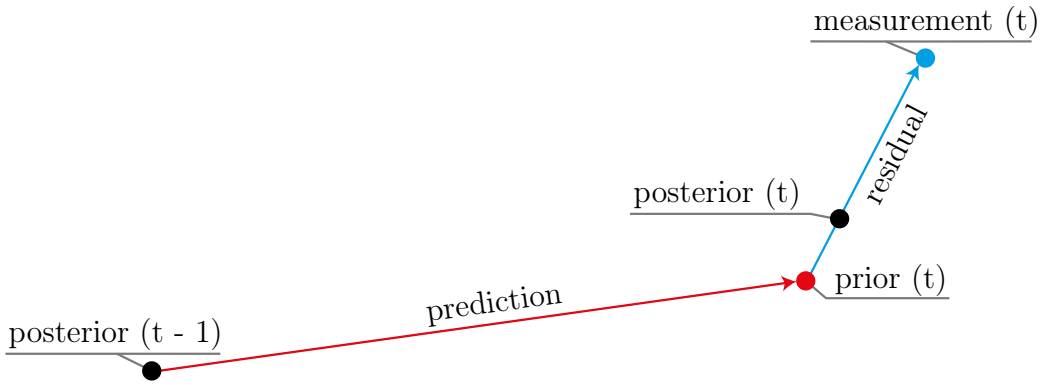


Figure 7.5: Visualization of the Kalman process. The prediction is based on the current system state (posterior) and a dynamic process model. The dynamic process of the multicopter is the integration of the angular velocity. The result (prior) is then updated using a (most likely) noisy measurement of the actual system state. The quality of the measurement (Kalman gain) regulates how much the new estimate adapts to the measurement.

Kalman filter will even converge towards the complementary filter once all variances reached a steady state.

The difference between the complementary filter and the moving average is that the complementary filter fuses two different signals while a moving average operates on one signal. The complementary filter does so by weighing the prior and measurement by a weight chosen by the user.

$$posterior = W * prior + (1 - W) * measurement \tag{7.29}$$

The factor W defines how much the prior gains weight over the measurement. A high value for W causes the measurement to have less impact on the prior.

Instead of the process model, the prior has to be calculated directly by integrating the gyroscope measurements.

7.2 PID

The flight controller has to regulate the motor speed to stabilize the attitude of the multicopter. The flight control knows the current attitude of the multicopter because of the filtering of accelerometer and gyroscope data. Based on the current attitude and the steering command issued by the remote control, the flight controller has to calculate the required motor speeds to reach

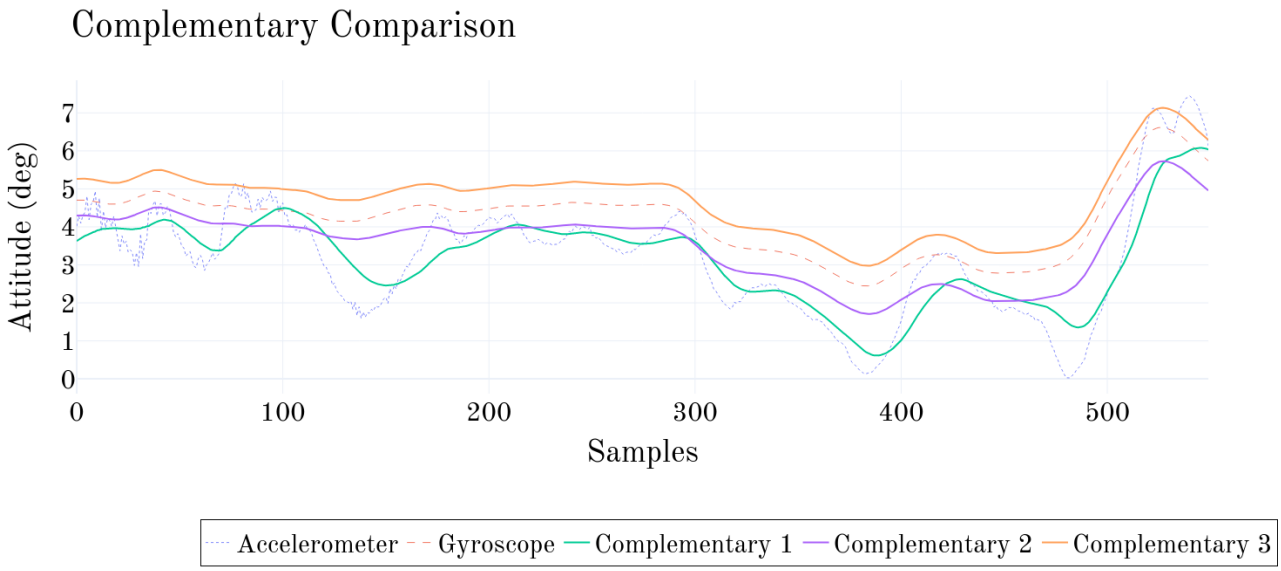


Figure 7.6: This diagram shows the same gyroscope and accelerometer measurements as in figure 7.3. The accelerometer signal is disturbed by a sharp noise, and the gyroscope signal drifts. The three complementary filter outputs show the results of three different weights. The green curve still has a lot of noise. That is because of the complementary filters putting too much weight on the accelerometer data. The orange curve shows a smoothed output but is drifting too much since it does not overlay the accelerometer measurements. The purple curve shows a good tradeoff between green and orange since it is smooth as the direct gyroscope data but also overlays the accelerometer data.

the target attitude. This work utilizes the so-called PID controller to regulate the motor speeds. PID is short for proportional-integral-derivative and names the three components of a PID controller.

PID controllers are widely used trough out the industry whenever a property of a dynamic system has to be regulated.

7.2.1 Fundamental

This section will first introduce the basic idea of a PID controller and explains how a PID controller is tuned to achieve the required performance. The next two sections will discuss two different approaches to implement a PID controller for a quadcopter.

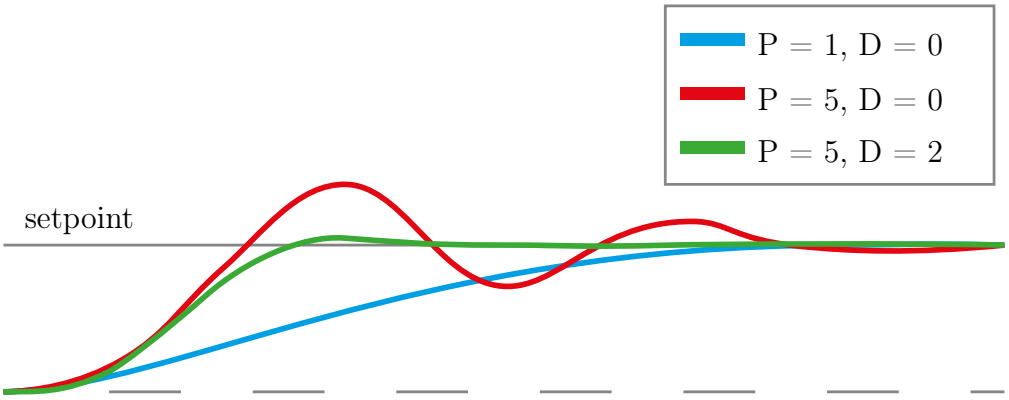


Figure 7.7: This visualization shows the different impacts of the P and D gain of a PID controller. A low P gain (blue) causes a delayed response but prevents overshooting like the red curve with a high P gain. Adding a D gain to the red curve helps to reduce the overshooting since once the green curve starts to increase too fast the D gain reduces its acceleration.

A PID controller tries to minimize the error between a process variable and a setpoint. The setpoint can either be constant or changing over time. The process variable can be measured. The PID controller sets up a feedback loop where it continually calculates the error between the process variable and setpoint and applies a correction based on that error to the control variable.

The purest form of a PID controller is the P-controller. The P-controller scales the error by a specific factor K_p and applies the result as the correction to the control variable. Once the process variable diverges from the setpoint, the non zero error will cause the PID controller to change the control output and counteract the change [35]. If the quadcopter starts tilting to the front, the P-controller will increase the speed of the front motors proportionally to the tilt angle. As figure 7.7 shows, a P-controller is likely to overshoot (red curve). This happens because the quadcopter is accelerating until the process variable reaches the setpoint. Due to its inertia and gained speed, it will not stop immediately after the setpoint was reached and therefore will start overshooting. Since a quadcopter does not experience much drag when hovering, it will start oscillating.

$$PID_p(t) = e(t) * K_p \tag{7.30}$$

The D-gain (derivative gain) helps to reduce the overshooting. It will add simulated drag to the PID controller. The simulated drag causes the quadcopter

to decelerate before the overshooting can start. The D-gain is the derivative of the error. The D-gain is negative if the error decreases and positive if the error increases. To control the amount of D-gain, the derivative of the error gets scaled by the factor K_d [35].

$$PID_d(t) = (error(t) - error(t-1)) * K_d \tag{7.31}$$

7.2.2 Control Input

The PWM output from the receiver has to be normalized and center-aligned. During the initialization of the quadcopter, the control software calculates the mean values for each PWM signal. To center-align the output, the mean of the PWM range has to be subtracted. After the centering, the signal range is approximately -400 to 400. The centered signal is then divided by 400 to normalize it to a range of -1 to 1. The centered and normalized signal is further denoted as the control input.

$$controlInput = \frac{t_{pulsewidth} - pwm_{mean}}{400} \tag{7.32}$$

The following two sections explain two different approaches to utilize the PID controller. It is important to note that the roll, pitch, and yaw axes get their own dedicated PID controller.

7.2.3 Attitude Error

The first PID implementation defined the setpoint to be a particular number of degrees that the quadcopter should be tilted based on the pilots' control input. When the sticks of the remote control are released, the setpoint is zero degrees, which should make the quadcopter hover steadily. The setpoint $setpoint_{attitude}$ is calculated as follows.

$$setpoint_{attitude} = controlInput * maxRange \tag{7.33}$$

The $controlInput$ is scaled by a constant factor $maxRange$ to control how sensitive the quadcopter reacts to the commands of the pilot.

The process variable is the current attitude of the quadcopter, which is obtained from the filter logic.

7.2.4 Angular Velocity Error

The second approach that was evaluated chooses a different approach for the setpoint and process variable.

The process variable of the attitude error (7.2.3) approach is equal to the attitude observed by the filter logic. The angular velocity error approach chooses the gyroscope output as the process variable. The current approach does, therefore, try to minimize the angular velocity of the quadcopter and not its attitude.

The velocity error setpoint $setpoint_{velocity}$ is calculated as follows.

$$setpoint_{velocity} = controlInput * maxRange - attitude * scale \tag{7.34}$$

The factor $maxRange$ has the same function as in the attitude error approach, which is to adjust how sensitive the quadcopter reacts to the pilots' control commands. Without the term, $attitude * scale$ the PID would be missing the information about the attitude of the quadcopter. The consequence is that the quadcopter rotates as long as the pilot pushes a stick in a specific direction. The quadcopter would be unable to stabilize its attitude after external turbulences altered its attitude. The control software subtracts the attitude of the quadcopter from the pilots' control input to include it in the setpoint. If the pilot is now pushing a stick in a specific direction, the quadcopter will stop rotating as soon as $controlInput * maxRange$ is equal to $attitude * scale$.

7.2.5 PD Controller

The PID controller of this work misses the I-part. The integral-part of a PID controller is usually required to generate an output in situations when the error is zero. An example of such a system is the altitude hold system of a quadcopter. The error of an altitude hold system is zero when the quadcopter is hovering in the desired height. The motors do still have to receive power to hold the altitude. The integral-gain of a PID controller will account for that [34].

$$PID_i(t) = PID_i(t-1) + error(t) * K_i \tag{7.35}$$

Since the attitude stabilization system does not require a zero error output, the control software does not implement the integral-part to reduce the

number of tuning parameters. Omitting an integral-part does also protect the controller from effects like integral windup that occurs when the integral-gain grows too big [35].

Chapter 8

Multicopter Control Software

8.1 STM32 Development

The multicopter control software was developed using an STM32G431 microcontroller of the company STMicroelectronics. This chapter explains the software development process for chips of the STM32 platform.

The company provides various tools to support engineers during the development process.

8.1.1 STM32CubeMX

The STM32CubeMX is an application used to generate initialization code for microcontrollers of the STM32 series. This tool simplifies the initialization of the required hardware units like timers, I2C connections, clocking, hardware accelerators like the CORDIC unit 8.4.

The first step is to choose the microcontroller that is used. By choosing the microcontroller, the tool knows what hardware units and output pins are available on the target platform. Then the developer can use a graphical user interface to configure all required functionalities. After the developer finishes the configuration, he can start the code generation. The code generator will generate a code to enable and initialize all units the developer requires.

The tool allows changing the device configuration at any time during the development process. For that to work correctly, the user has to take care of the particular code sections that the tool creates. Since the code generator will have to regenerate the code every time the configuration changes, a strategy to update the existing code is required. The Stm32CubeMX will

insert code comments to mark out segments where the user is free to add his code. These segments remain unchanged during the code generation process. If the developer places code in the wrong places, the code generator will overwrite the user's code.

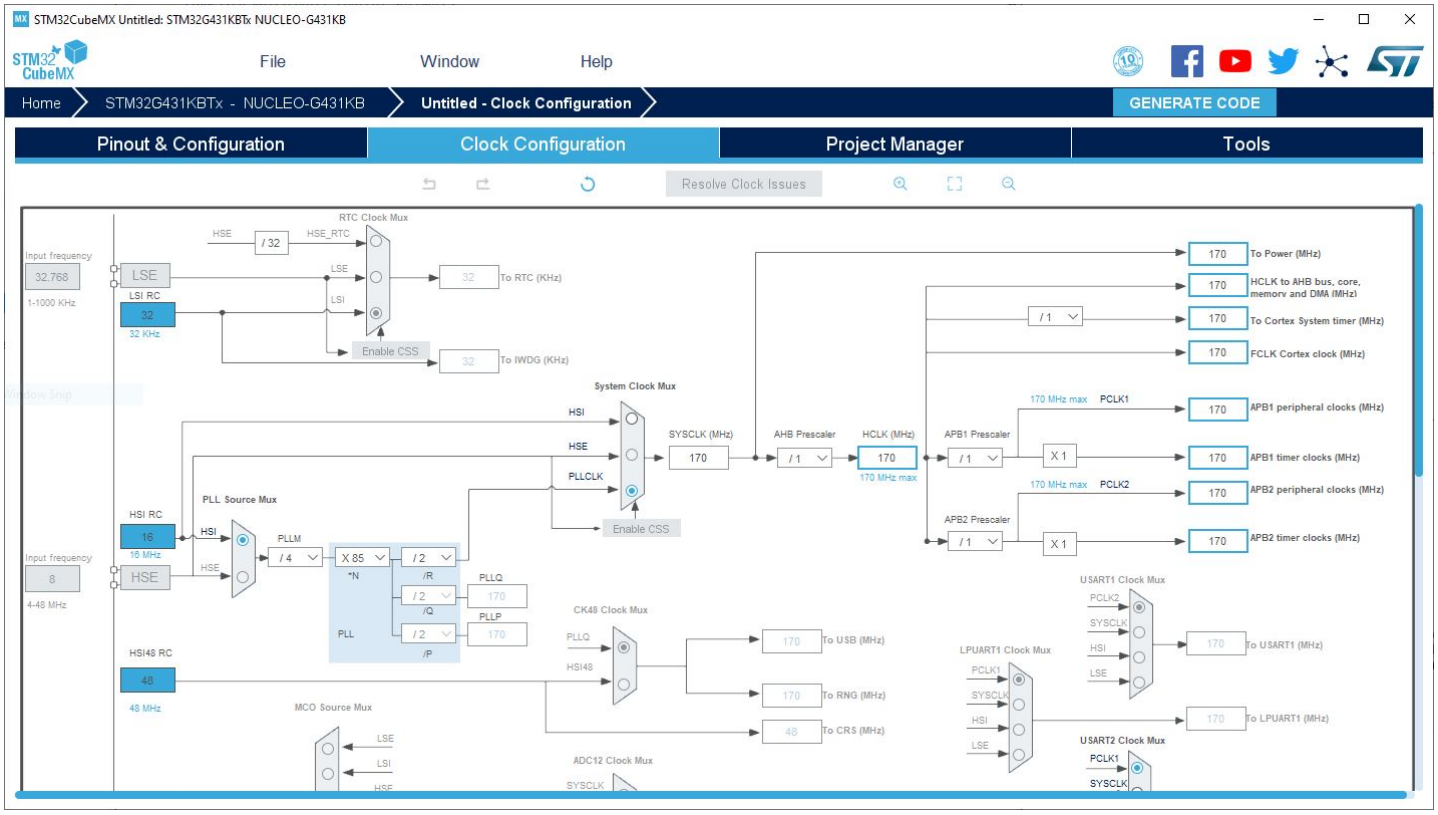


Figure 8.1: The STM32CubeMX software helps to initialize all the required hardware units of a STM32 microcontroller. The clock configuration allows to set up the clock frequency of the microcontroller as well as for the various timers and communication interfaces. Source: Screenshot

The figure 8.1 shows an example of the clock configuration user interface. The details of the clock configuration are not relevant for this work. The only altered property was the clock frequency. For more detailed information about the STM32CubeMX, please contact [36]

```c
int main(void)
{
  /* USER CODE BEGIN 1 */

  // user code can be placed here

  /* USER CODE END 1 */

  // this code here would be overwritten

  while (1)
  {
    /* USER CODE BEGIN WHILE */

    /* USER CODE END WHILE */

    // this code here would be overwritten
  }
}
```

8.1.2 STM32CubeIDE

The STM32CubeIDE is an Integrated Development Environment for the STM32 platform. The IDE is built upon the eclipse toolchain and integrates many other useful tools to support the developer during programming, debugging, and monitoring. The STM32CubeIDE does also include the code generator of the STM32CubeMX. Because of that, it is easy to apply changes to the device configuration during the development process.

The STM32CubeIDE does include a GCC-based compiler for the STM32 platform. The IDE also includes a debugger that can be used to debug the program code step by step. Further, it is possible to set breakpoints to interrupt the program execution and start debugging. While debugging the content of variables can be inspected dynamically.

The Serial Wire Viewer SWV technology allows it to monitor the read and write access on any variable of the program. The next chapter will introduce this technology in greater detail. The STM32CubeIDE offers various ways to utilize the monitoring capabilities of the SWV technology. It further allows us to profile the execution and inspect possible performance bottlenecks.

The following list explains the available tracing functions of the STM32CubeIDE.

- The SWV Data Trace view allows us to analyze the read and write

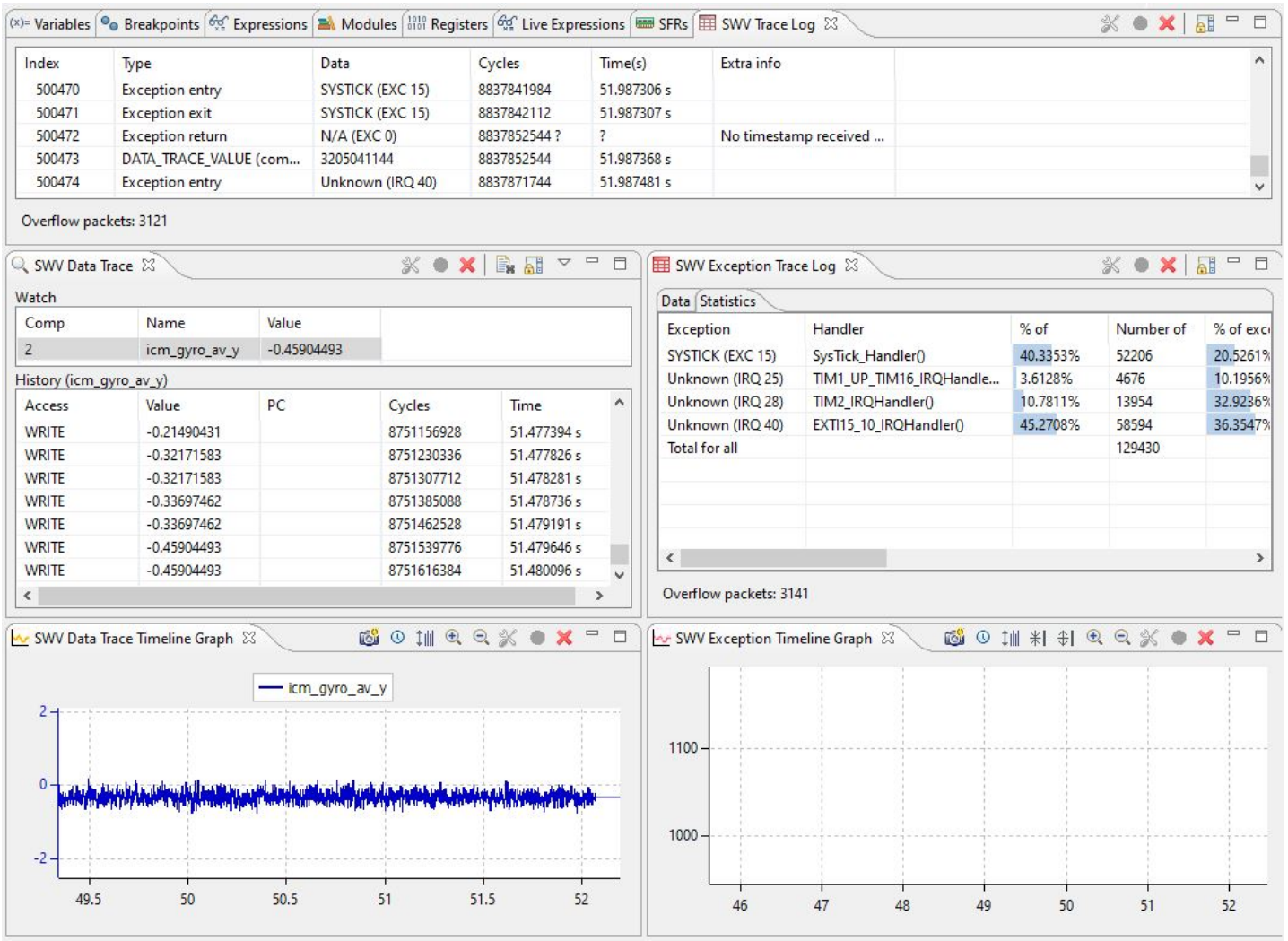


Figure 8.2: The STM32CubeIDE software offers a great variety of debugging and monitoring tools. A line graph can visualize time-series data. The Exception Trace log shows how often a particular exception occurred. Source: Screenshot

accesses to certain variables that were specified before. The log output does show the exact cycle number of the variable access, the corresponding time in seconds, the value read or written, and the type of access (read or write). It is possible to show only read or write accesses.

- The SWV Data Trace Timeline Graph view offers a graphical representation of the SWV Data Trace log. The time-series data of the SWV Data Trace is shown as a continuous line graph. This graph allows analyzing specific values in realtime quickly.
- The SWV Exception Trace Log shows all occurring exceptions during program execution. The statistics view allows for analyzing the frequency of all executions by showing the total occurrences, the percentual share, the type of the exception, and the exception handler.
- The SWV Exception Timeline shows a line graph of the occurred exceptions.
- The SWV Trace Log is a combined list view of exceptions, read and write accesses.

8.1.3 STM32 Serial Wire Debug

The Serial Wire Viewer (SWV) technology of STMicroelectronics offers excellent debugging capabilities. While the previous chapter already presented some analysis tools, this chapter will discuss the SWV technology. Since the in-depth details of the SWV technology are not relevant for this work, only the basics are explained.

The SWV capability is built into the microcontroller. It is not possible to use SWV with microcontrollers that do not explicitly offer this feature. SWV allows tracing multiple variables in real-time without any performance impact on the microcontroller [37].

In order to use SWV, a connection of the SWCKL and SWDIO pins to an ST-Link debugger is required. The ST-Link debugger is then connected to the capturing PC. The STM32-Nucleo series does already include an onboard ST-Link debugger. If no onboard ST-Link debugger is available, STMicroelectronics offers external ST-Link debuggers [37]. The SWV feature has to be enabled in the STM32CubeIDE run configurations and the clock frequency has to be set accordingly as shown in figure 8.3.

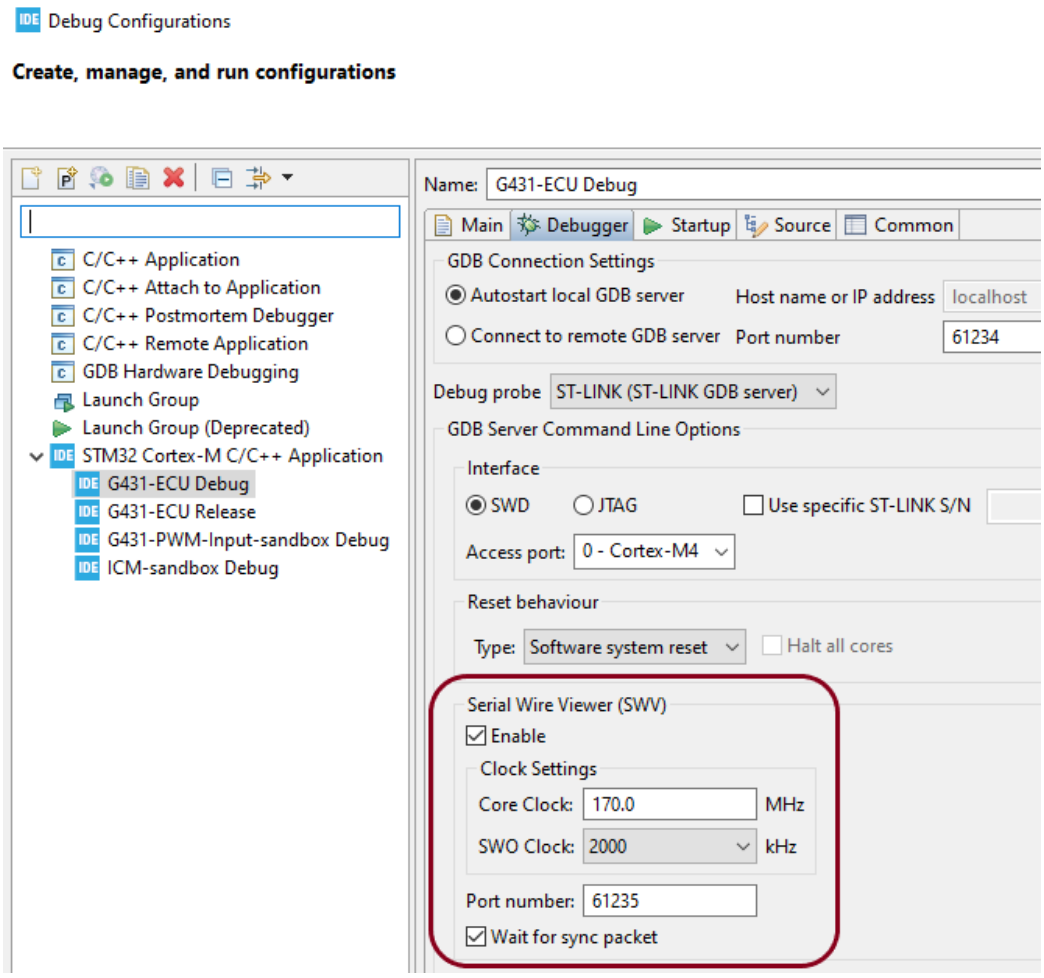


Figure 8.3: To use the SWV technology it has to be enabled in the run configurations. The clock frequency has to be set accordingly. Source: Screenshot

8.1.4 STMStudio

The STMStudio relies heavily on the Serial Wire Viewer technology. It enables the user to trace variables of a microcontroller program just as the STM32CubeIDE does.

The STMStudio provides a better user interface, can write trace files, and has more advanced visualizations. The number of variables to trace is limited to four in the STM32CubeIDE. The number of variables in the STMStudio, however, is theoretically unlimited, but the sample rate will reduce if more variables are captured. Another benefit of using STMStudio is that it allows dumping all captured data to a log file for later analysis. That way, the

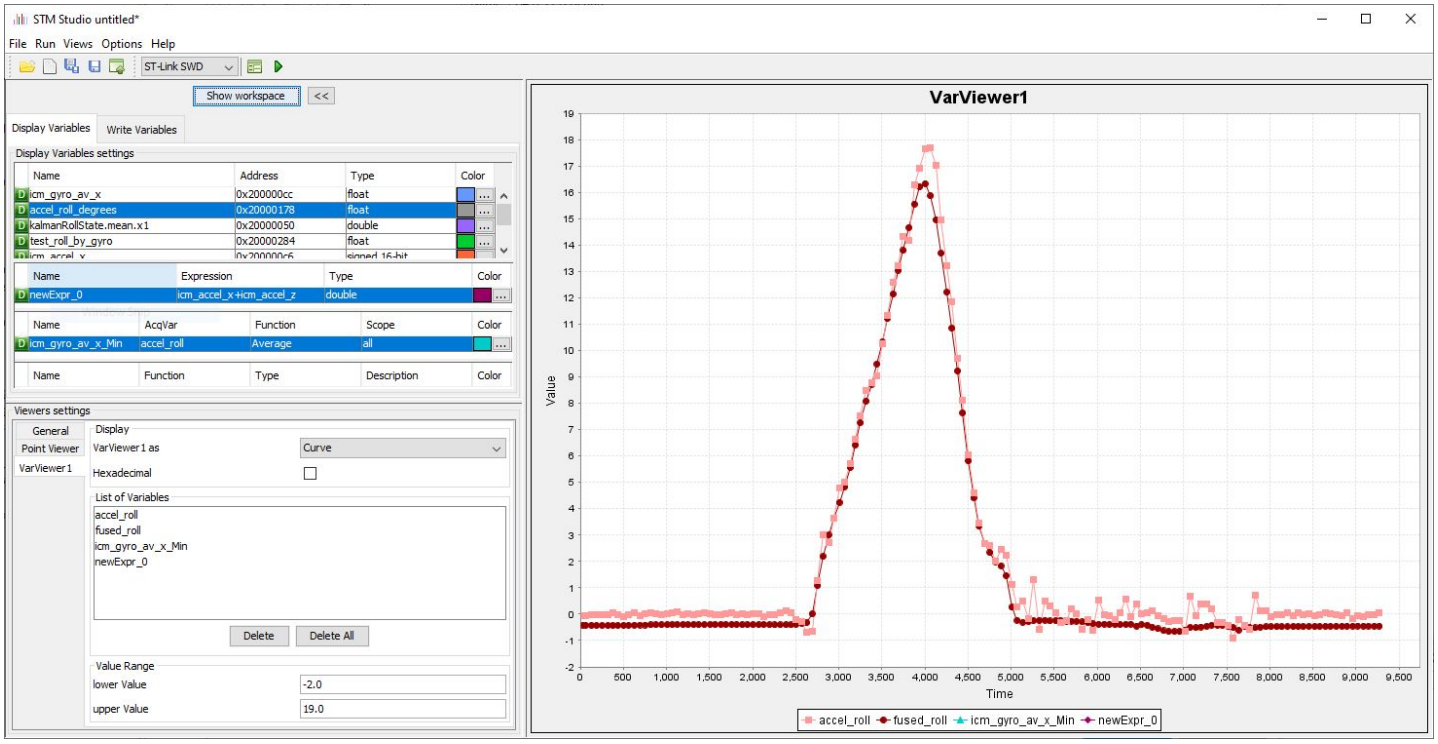


Figure 8.4: STMStudio allows to capture realtime traces. Source: Screenshot

developer can focus on data gathering and does not have to analyze the data in realtime. STMStudio does also allow bar charts and scatters plots over the default graph visualization. Further, it allows some basic mathematical operations to be directly applied to the captured signals. This includes the min, max, and average operations, as well as the calculation of the standard deviation. It is also possible to write simple expressions like the sum of two observed variables and visualize it as another graph.

STMStudio offers an easy way to import the required variables for tracing. The developer has to specify the path to the compiled binary file. STMStudio will then search all variables and present them to the user as a list. The developer can select the variables he wants to observe and import their binary addresses. If the addresses change due to a recompilation, STMStudio will automatically update the addresses before another trace is started.

The STMStudio was used to trace different sensor readings and filter outputs during the development of the quadcopter.

8.2 I2C Communication

8.2.1 Blocking IO

Using blocking mode to read the accelerometer data was expected to have the worse performance compared to the approach using DMA. Using blocking behaviour a lot of time is wasted by waiting for data to arrive from the ICM peripheral. Since there is a strong realtime need for the accelerometer and gyroscope data a continuous stream of the data is vital. This results in a lot of wasted time due to the blocking behaviour. If the data transfer was very rarely required then it would be enough to use blocking transfer since the amount of waiting time would be neglectable low.

The following code was used to observe the performance statistics. It issues 100000 reads requests from the ICM peripheral and measures the time needed. Only successful transfers are counted, if a transfer failed, it is repeated. After the benchmark run, we output the number of successful transfers (should be 100000), the total time needed to run the benchmark and the transfers/ms value representing the successful transfers per millisecond.

number of transfers	100,000			
transfer speed (khz)	100		400	
data transfered (byte)	12	24	12	24
total time (s)	142.9	253.0	38.5	68.0
us per transfer	1429	2530	385	680
us per transfer (theoretical)	1380	2460	345	615

Table 8.1: Blocking transfer mode benchmark results.

```
uint32_t no_reads = 0;
uint32_t target_no_reads = 100000;

uint32_t begin_tick = HAL_GetTick(); // current time in ms

while (no_reads < target_no_reads) {
  i2c_status = HAL_I2C_Master_Transmit(&hi2c3, ICM_ADDRESS,
  ↪  &icm_start_address, 1, HAL_MAX_DELAY);

  if (i2c_status != HAL_OK) {
    continue;
  }

  i2c_status = HAL_I2C_Master_Receive(&hi2c3, ICM_ADDRESS,
  ↪  (uint8_t*)icm_read_buffer, ICM_READ_BUFFER_LEN, HAL_MAX_DELAY);

  if (i2c_status != HAL_OK) {
    continue;
  }

  no_reads++;
}

uint32_t duration = HAL_GetTick() - begin_tick;
LOG("Took %lu ms for %lu successful reads = %lu r/ms", duration,
↪  target_no_reads, target_no_reads / duration);
```

The results of the upper benchmark are summarized in table 8.1.

8.2.2 DMA

To improve the performance of the flight controller, we utilize the DMA capability of the microcontroller. The time which was wasted for waiting for the data transfer to complete in the chapter 8.2.1 is now used for the computation of steering commands.

The benchmark 8.1 tells us that we can gain about 345 us more time per transfer for the calculations when transferring 12 bytes using Fast Mode (400 kHz).

Since the DMA transfer should only be triggered when data is available, the ICM peripheral has to notify the STM32 controller in some way. Otherwise, reading the same data twice would be again wasteful in terms of performance. Another problem that was prevented by design in the blocking IO mode is

the fact that we could trigger two DMA requests running in parallel (which is not supported by the I2C interface). This is now possible because if we trigger a DMA request for each iteration of the main computation loop, the old transfer might not yet be finished. This will happen every time the computations in the main loop last less than 345 us. When the blocking IO mode was used, this was not possible by design.

The ICM unit is capable of firing interrupts for various events. It can notify the STM32 controller when new data is available in its data registers (this applies to the accelerometer, gyroscope, and temperature data registers). To configure the corresponding interrupt, the INT_ENABLE_1 register needs to have the value $RAW\_DATA\_0\_RDY\_INT$ (bit 0) set to 1. The ICM then pulls the INT pin up, every time new data is available.

By analyzing the intervals of the raw data ready interrupt, we can calculate that the ICM supplies new data values every 885 us. Since the transfer of the data only takes about 345 us (using fast mode and 12 bytes payload size), we have to wait for about 540 us till new values are available. This tells us as well that the blocking IO mode pulled the same values two to three times since we have not waited till new data was available.

This also helps us to solve the problem that we could issue two DMA requests in parallel. If we only issue a DMA request once the new raw data available interrupt was fired, we can never send two DMA requests in parallel.

8.3 Timer

Each of the STM32 microcontrollers allows the user to measure time [38]. The control software of the microcontroller has many functionalities that require an exact time measurement unit. For example, the PWM controller each input and output need time measurements to determine the duty cycle of the underlying PWM signal. As already explained before, the PWM duty cycle is that time of the PWM signal where the signal has a high level. The input controller needs to measure the length of the duty cycle by measuring the delta time between the rising and falling edge of the signal. The output controller needs to send a high level as long as the user programmed it. Therefore another time measurement is needed.

8.3.1 Fundamental

Some fundamentals apply to various microcontroller timers like STM32 or the Arduino series. This work will focus on the STM32 timers and may not apply to other microcontroller architectures.

8.3.2 Clock Sources

A timer needs a clock source that can be used as ground truth to measure time differences. The STM32 architectures allow different clock sources that can be configured [38]. The user has to choose one depending on the use case of the timer.

Each STM32 microcontroller has an internal clock source which drives the processor. The internal clock can be used as well as a clock source for the timers. This clock source is useful to measure time since the internal clock frequency is a known variable. Assuming that f is the internal clock frequency in HZ, the duration of a single time tick δt can be calculated as follows.

$$\delta t = \frac{1}{f}$$

δt is also the smallest time unit that can be measured.

Another timer clock source is an external signal. This mode is useful to synchronize some internal microcontroller operations with an external component but not to measure time directly. When using this mode, an external component can send a high level, and the STM32 will use this signal to drive its counter register and interrupts. This mode is, for example, useful to synchronize two devices using a clock signal. The clock signal can be hooked up as a timer clock source, and on each timer interrupt, another operation can be issued.

8.3.3 Counter Mode

Each timer can be configured to either count the value of a dedicated timer register up, down, or change the direction each time it reaches the upper or lower limit [38]. The latter mode is called center aligned. The upper or lower limit of a timer is called 'counter-period' or 'auto-reload' and is configured using the ARR register. The counter-period controls when the timer overflow interrupt gets fired depending on the counter mode:

- For up-counting mode each time the CNT register value reaches the value in the ARR register.
- For down-counting mode each time the CNT register value gets zero.
- For center-aligned mode each time the CNT register value either reaches ARR or zero.

8.3.4 Prescaler

Most of the timers have 16 or 32 bit counter registers CNT [38]. That means that the maximum value of the counter register is restricted at least by the size of the register (when not a lower auto-reload value was configured). A microcontroller with an internal clock frequency of 100 MHZ increases the counter register every 10ns. A 16 bit counter register has a maximum size of 2^16 counter cycles. After 2^{16} cycles the register overflows, and an update event interrupt is fired. Such a timer allows measuring a period of exactly $2^{16} * 0.01us = 655.35us$. The prescaler-register PSC allows the developer to measure more extended periods. The register holds the timer ticks that are waited before the counter register gets increments (or decremented). When the prescaler of the 100MHZ microcontroller is set to 10000, it is possible to measure a total of $2^{16} * 0.01us * 10000 = 6553500us = 6.5535s$. Also, in case of a clock frequency of 48MHZ setting, the timer prescale register set to 48 allows to easily calculate time differences since one tick is equal to $1us$ and not $1/48us$ as it would be without the prescaler.

8.3.5 Timer Channels

Some timers have several channels attached. A channel is an input or output (depending on the configuration), which is connected directly into the timer unit. The following sections describe some of the possible applications of the timer channels.

8.3.6 Input Capture

An input capture channel can be used to trigger an interrupt when the input of the specific channel is in a particular state. Furthermore, it can be used to save the value of the timer when the state change of the channel occurred to a register called $CCRx$ [38]. The STM32 controllers are able to trigger on either a rising or falling signal edge or on both edges. This behavior is

controlled by the $CCER$ register of the timer and channel. To account for signals which take some clock cycles to stabilize an input filter can be applied which specifies the number of clock cycles the input capture unit should wait until a new input capture cycle is started.

8.3.7 Read Single PWM Signals

The input capture feature is useful to operate with incoming signals and can be used, for example, to read a PWM signal. As already explained the signals from the receiver are PWM signals. In order to react to those signals, the microcontroller needs to read the PWM signals. As already mentioned, the crucial part of a PWM signal is the duration of the duty cycle. The duty cycle is the period of a PWM signal where the signal level is high.

To read a single-channel PWM input the input capture unit is configured to trigger on a rising edge. The reason we are not using both edges trigger mode is that we can observe the configuration of the input capture configuration register $CCER$ to determine if we hit the rising or falling edge of the PWM signal. To change the input trigger requires to flip a bit in the $CCER$ register. If a rising edge is detected, the input capture interrupt gets fired, and the value of the input capture register $CCRx$ is written to a global variable called $ch1\_start$. We then switch the trigger mode to the falling edge and wait until the PWM signal changes to a low level. Again an interrupt is fired, and we subtract the $ch1\_start$ value from the current value of the input capture register $CCRx$ and call that result $ch1\_duty$. It is essential to configure the prescaler of the timer in such a way that the longest possible PWM duty cycle does not cause the counter register CNT to overflow since we would have to add additional logic to account for that. Otherwise, the calculated $ch1\_duty$ would be negative. Additionally, we have to reset the counter register CNT back to zero when the duty cycle was measured and the trigger event back to a rising edge.

8.3.8 Read Multiple PWM Signals

Some timers offer the possibility to use more than one capture compare channel. This can be used to read all four PWM signals of the receiver (roll, pitch, yaw, and throttle). The measurement of a single PWM channel was described in the previous section. The four PWM signals are sent over four different wires but arrive in a well-defined order. It is possible to switch the active channel each time a PWM signal was observed. The active channel can be

chosen by writing its number to the *DIER* register. After a PWM signal was read, the trigger event is switched to a rising edge. The following code demonstrates the multi PWM signal reader.

```
1  extern TIM_HandleTypeDef htim2;
2
3  volatile uint32_t ch1_start = 0;
4  volatile uint32_t ch2_start = 0;
5  volatile uint32_t ch3_start = 0;
6  volatile uint32_t ch4_start = 0;
7
8  volatile uint32_t ch1_duty = 0;
9  volatile uint32_t ch2_duty = 0;
10 volatile uint32_t ch3_duty = 0;
11 volatile uint32_t ch4_duty = 0;
12
13 /**
14 * @brief This function handles TIM2 global interrupt.
15 */
16 void TIM2_IRQHandler(void)
17 {
18   HAL_TIM_IRQHandler(&htim2);
19
20   if(__HAL_TIM_GET_IT_SOURCE(&htim2, TIM_IT_CC1) == SET) {
21     if (htim2.Instance->CCER == 1u << TIM_CHANNEL_1) {
22       ch1_start = htim2.Instance->CCR1;
23
24       htim2.Instance->CCER = 3u << TIM_CHANNEL_1;
25     } else {
26       ch1_duty = htim2.Instance->CCR1 - ch1_start;
27
28       htim2.Instance->DIER = TIM_IT_CC2;
29       htim2.Instance->CCER = 1u << TIM_CHANNEL_2;
30     }
31
32     return;
33   }
34
35   // handle the other PWM channels just like the previous one
36   // reset the counter on the falling edge of the last channel
37   // to prevent an counter overflow
38   // htim2.Instance->CNT = 0; // reset counter
39 }
```

The last if block will switch the active capture compare channel back to

channel one, set the trigger event to rising edge, and reset the counter to zero in order to prevent a timer overflow.

The global variables *chx_start* and *chx_duty* have to be declared volatile since they are accessed inside an interrupt service routine. When an interrupt occurs, the execution of the main code stops, and the interrupt service routine gets executed. After the service routine is finished, the execution continues where it was first interrupted. An optimizing compiler might expect the value *chx_duty* to be constant since the value is indeed never changed during the endless main control loop. The volatile keyword functions as a hint for the compiler to read the annotated variable directly from memory.

```
1  uint32_t ch1_duty = 0;
2
3  /**
4  * @brief This function handles TIM2 global interrupt.
5  */
6  void TIM2_IRQHandler(void)
7  {
8    ch1_duty++;
9
10   // Indeed the value of ch1_duty is changed here
11   // but the compiler thinks this function is not called
12   // during the main loop and concludes that
13   // ch1_duty stays unchanged.
14 }
15
16 int main() {
17   while(1) {
18     printf(ch1_duty);
19     // without declaring the ch1_duty variable
20     // volatile the compiler might never read
21     // the value of ch1_duty again since in it
22     // expects its value to never change.
23   }
24
25   return 0;
26 }
```

8.3.9 PWM Generation

The ESCs control the speed of the four motors. The microcontroller can control the motor speeds by sending a PWM signal to the ESCs.

The STM32 timers offer the possibility to generate PWM signals directly. The hardware-driven PWM generators ensure a stable signal and are simple to control.

The timer has to be set up in PWM generation mode and the period of the PWM signal as the auto-reload value. The duty cycle of the PWM signal can be controlled by the $CCRx$ register of the corresponding output channel [38].

8.4 CORDIC Accelerator

The STM32G431 microcontroller features a CORDIC accelerator unit that improves the speed of trigonometric calculations. The multicopter control software requires the calculation of the $atan2(x)$ and $sin(x)$ functions. The CORDIC unit was utilized to speed up the trigonometric calculations and improve the overall performance.

Jack E. Volder invented the CORDIC algorithm in 1959 to improve the performance of trigonometric calculations [39]. The CORDIC algorithm is an iterative algorithm that works by rotating a vector by decreasing rotations every iteration. The x and y coordinates of the rotated vector are then equal to the cosine and sine of the sum of all rotations. Further, the CORDIC algorithm allows computing the hyperbolic functions as well as the square root, natural logarithm, and exponent. This work will not explain the details of the CORDIC algorithm but will describe how the CORDIC unit of the STM32G431 was utilized. However, the original work of Jack E. Volder provides in-depth information about the CORDIC algorithm and its implementation [39].

The CORDIC accelerator unit requires a minimal configuration before it is ready to use. The STM32CubeMX and STM32CubeIDE allow enabling the CORDIC accelerator by using the graphical user interface. Once the CORDIC unit was enabled, it is ready to use.

The CORDIC processor supports a great variety of functions that have to be selected before executing computations. This work only uses the $atan2$ and sin functions. The input data for the processor has to be in q1.15 or q1.31 format. The Q-number format is used to describe a fixed-point number in 16 and 32 bit two's complement representation. The first bit represents the sign of the number, and the remaining bits the fractional part. Two q1.15 values can be passed as one q1.31 number to the CORDIC unit by shifting the first q1.15 number by 16 bits to the left.

In order to compute the $atan2(x, y)$ function, the CORDIC unit is configured to use the function named $CORDIC\_FUNCTION\_PHASE$. The input is one q1.31 value, containing two q1.15 values that represent the gravity components of the x- and z-, respectively y- and z axes. The result should be one q1.31 value containing the rotation angle in a range from -1 to $+1$, representing an angle from $-180deg$ to $+180deg$. The precision of the accelerator has to be specified by code. The CORDIC accelerator's accuracy improves approximately bitwise per iteration [40]. The precision is set to $CORDIC\_PRECISION\_6CYCLES$, which results in 6 $cycles * 4 = 24$ $bits$ precision. The code necessary to execute the computation is shown below.

```
// use the atan2 function
sConfig.Function = CORDIC_FUNCTION_PHASE;

// input is 16 bit for x and y
sConfig.InSize = CORDIC_INSIZE_16BITS;

// CORDIC should return one q1.31 value
sConfig.OutSize = CORDIC_OUTSIZE_32BITS;

// the precision (cycles * 4 = iterations)
sConfig.Precision = CORDIC_PRECISION_6CYCLES;

// read one q1.31 value
sConfig.NbRead = CORDIC_NBREAD_1;

// return one q1.31 value
sConfig.NbWrite = CORDIC_NBWRITE_1;

HAL_CORDIC_Configure(&hcordic, &sConfig);

// call the CORDIC accelerator

int32_t cordic_in;
int32_t cordic_out;

if (HAL_CORDIC_Calculate(&hcordic, &cordic_in, &cordic_out, 1,
↪  HAL_MAX_DELAY)) {
  // an error occured
}
```

The following table shows the performance of the CORDIC accelerator compared to other trigonometric functions like the $sinf(x)$ function of the math.h.

The comparison shows a 10 times faster calculation of the $atan2(x, y)$ function compared to the math.h library.

Function used	CPU cycles sin(x), cos(x)	CPU cycles atan2(x, y)
CORDIC in zero overhead mode	29	33
CORDIC in zero overhead mode including float ->q1.31 ->float conversion	79	107
math.h single precision floating point	416	332
math.h double precision floating point	4036	4194

Table 8.2: Comparison of the CORDIC accelerator and the math.h library. Source: [40]

8.5 Accelerometer Skew

During tests with the ICM-20948 an axis skew between the accelerometer and gyroscope axes was observed. The multicopter was mounted on a board together with a digital protractor. The board was rotated around a single axis. The multicopter pitch axis was aligned with the rotation axis. The attitude measurements of the multicopter should be equal to the protractor measurements. The gyroscope showed the same measurements as the protractor. Since the duration of the test was very short, the gyroscope drift was not able to distort the results.

The accelerometer data revealed a divergence from the gyroscope 8.5. Further analysis of the divergence revealed a linear correlation between gyroscope and accelerometer measurement. Figure 8.6 shows the quotient of both measurements which was further used to correct the skew.

8.6 Roll/Pitch Correction

The complementary and Kalman filter require the integration of the gyroscope data to obtain the prior. The first idea was to integrate the gyroscope axis per axis (for roll and pitch) (equation 8.1).

$$\begin{aligned} prior_x &= posterior_x + gyro_x * dt \\ prior_y &= posterior_y + gyro_y * dt \end{aligned} \tag{8.1}$$

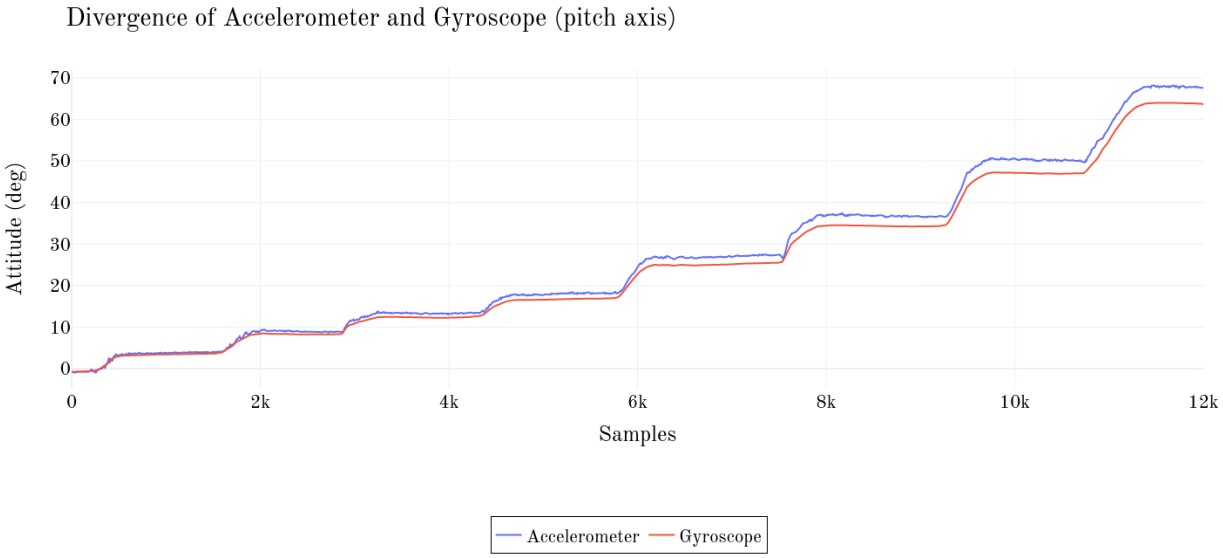


Figure 8.5: The multicopter was rotated, and the measurements of the accelerometer and gyroscope were compared with a digital protractor. A clear divergence of the gyroscope and accelerometer data is observable. The error grows when the board is rotated further.

If the quadcopter does not rotate around its z-axis, this approach works fine. But if the quadcopter starts turning around its z-axis, the z-axis rotation has to be taken into account. Otherwise, the integration can yield wrong results, as figure 8.7 shows. The body was initially rotated around its x-axis by ϕ degrees (left pose). The right figure shows the body after another rotation around its z-axis. The integration from equation 8.1 misses this rotation since it occurs around the z-axis of the body itself. The z-axis rotation results in a y-rotation in the reference frame of the body. The following integration was used to account for the z-axis rotation.

$$\begin{aligned} prior_y &= posterior_y + gyro_y * dt \\ prior_x &= posterior_x + gyro_x * dt \\ \\ prior_y &= prior_y - prior_x * \sin(gyro_z * dt) \\ prior_x &= prior_x + prior_y * \sin(gyro_z * dt) \end{aligned} \tag{8.2}$$

The difference to equation 8.1 is that each others prior is weighted by $sin(gyro_z * dt)$. To provide quick prove, we assume the body to be rotated 45 degrees

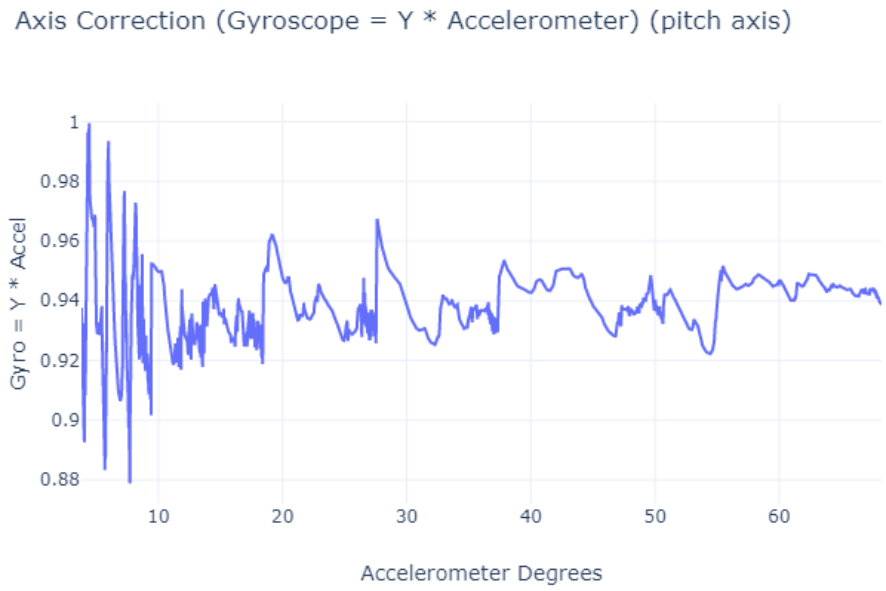


Figure 8.6: The graph shows the skew between gyroscope and accelerometer measurements.

around the x-axis. The body gets then rotated by 90 degrees around the z-axis. This is the same as shown in figure 8.7. Equation 8.3 shows that effectively $prior_x$ was switched with $prior_y$. The same effect can be seen in figure 8.7.

$$\begin{aligned} prior_y &= 0 + 0 * dt \\ prior_x &= 45 + 0 * dt \\ \\ prior_y &= 0 - 45 * 1 = -45 \\ prior_x &= 45 - 45 * 1 = 0 \end{aligned} \tag{8.3}$$

8.7 IMU Filter Algorithm

This work has already presented two approaches for sensor data fusion Complementary Filter (7.1.3), Kalman Filter (7.1.2). Both approaches can be used to collect attitude estimations for a multicopter. But during evaluations

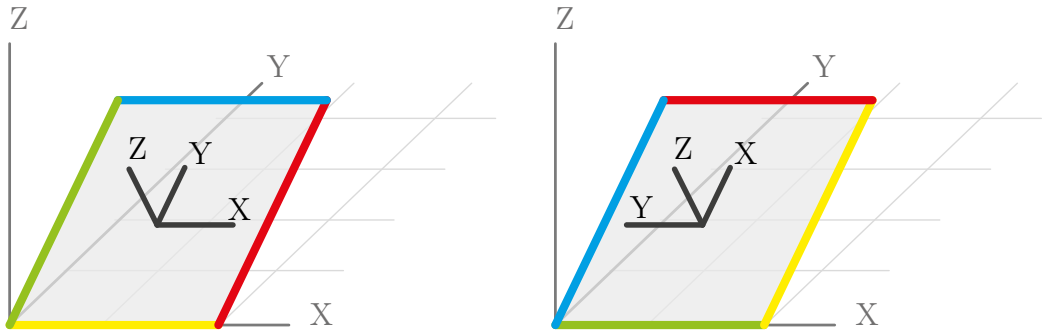


Figure 8.7: The left body is rotated 45 degrees around its x-axis. Additional rotation of 90 degrees around its z-axis causes the body to be effectively rotated 45 degrees around its y-axis.

the simpler complementary filter showed important benefits which made it the algorithm that is currently used by the built quadrocopter.

8.7.1 Kalman Filter

The Kalman filter is a professional approach to sensor data fusion. As already explained it utilizes gaussian distributions to cope with noise distorted measurements.

The first step is to define the Kalman state. That is a vector which describes the state of a certain system. In case of a multicopter that is the angular velocity and the attitude. The Kalman filter has an update step which implements a process model that represents the dynamics of the observed system. The Kalman filter calculates a new state estimated (called prior) based on the process model and its current state. The process model of a multicopter integrates the angular rotation velocities to calculate the absolute rotation. After the update step the measurement step integrates new measurements by weighting them with the variance of their source.

A big difficulty when modeling a systems state by a Kalman filter is the estimation of the variances. An easy approach would be to obtain N measurements and calculate the according variance σ.

$$\sigma = \frac{1}{N} \sum_{i=0}^{N} (\mu - x_i)^2$$

The problem is that those estimations are optimal in theory but are not usable for an IMU. The propellers of a multicopter cause vibrations which

affect the sensor measurements. Since the vibrations depend on the motor speed it is not possible to estimate a single variance for the most sensors. That does not only apply to multicopters but to any other dynamic system. It is possible to calculate speed dependent variances. The current motor speed can then be used to select the sensor variance.

A Kalman filter with variances estimated based on real world experiments does deliver good results but they are most likely not practicable. That is because different control systems have different requirements to the state estimate. The multicopter requires the state to be free of big noise or vibrations since otherwise it would try to counteract the vibrations. That would result in an oscillating multicopter.

There are various techniques to tune Kalman filter estimators [41] [42] [43] [44].

Multivariate Kalman filter are a powerful tool to model and estimate dynamic systems. But due to their power they are also difficult to optimize since they have a large number of parameters. The initial idea behind the implementation of a Kalman filter for the IMU was that it models the reality just by inserting variances obtained trough measurements. That concept was proven wrong and will therefore not longer be followed. Another method to build an IMU has to be developed with the goal to minimize complexity and maximize performance.

8.7.2 Complementary Filter

The complementary filter is a filter algorithm that is easy to tune but powerful. The details of this algorithm were already discussed in chapter Complementary Filter.

This chapter will discuss why the complementary filter was chosen as the preferred algorithm for the quadcopter.

As the previous chapter already explained, the biggest downside of the Kalman filter is its complexity. The estimation of the various parameters is difficult and error-prone. The complementary filter has only one parameter to tune, which makes it easy and straight forward.

Another benefit of the complementary filter over the Kalman filter is that the Roll/Pitch correction from the chapter Roll/Pitch Correction can not be modeled with a basic Kalman filter.

The complementary filter offers excellent performance when compared to the Kalman filter, which made it the preferred algorithm for the quadcopter.

8.8 Degradation

The control software contains a degradation mechanism for safety purposes. During initialization, the quadcopter calculates the mean values of the sensor inputs and the control inputs. If some values diverge more than a specific threshold value, the startup is aborted, and the motors will not start spinning. Some common causes for an aborted startup process are sticks of the remote control that are not centered or vibrations of the surface the quadcopter stands on.

During the flight, the quadcopter will degrade if it notices sensor inputs that exceed specific threshold values. This safety mechanism will shut down the motors as soon as the quadcopter flips over or the sensor output invalid data.

If the control software notices invalid PWM signals, it will wait if the next signal is valid and replay the old signal.

Chapter 9

Conclusion

This is the last chapter of this work and will go over all the topics that were discussed. It will as well provide an overview of the achieved performance and close with suggestions for future improvements.

9.1 What was done

The beginning of this work outlined the application areas of multicopters. Many prominent firms are working on multicopter technology to provide various services for their customers. One of them is DHL, which is working on delivering parcels by multicopters [1]. Another approach is to do professional videography and let the multicopter carry the camera and its equipment. Other people use multicopters for sports events or just for fun.

During this work, a quadcopter was built. The quadcopter was used to test different algorithms for sensor data fusion, filtering, and flight stabilization. The quadcopter is equipped with a gyroscope and an accelerometer to observe the attitude of the multicopter. An I2C bus connects the sensors to the onboard microcontroller, as well as a PWM line does for the receiver and ESCs.

The following chapter introduced the multicopter reference frames. The body frame describes the orientation of the multicopter relative to itself while the world frame is static and does not change while the multicopter is rotating and moving around. All sensors capture their data corresponding to the body frame. The sensor frame was never mentioned during this work since the sensor and body frame are aligned. Another work might choose to position the sensor in a different way what makes it essential to convert between

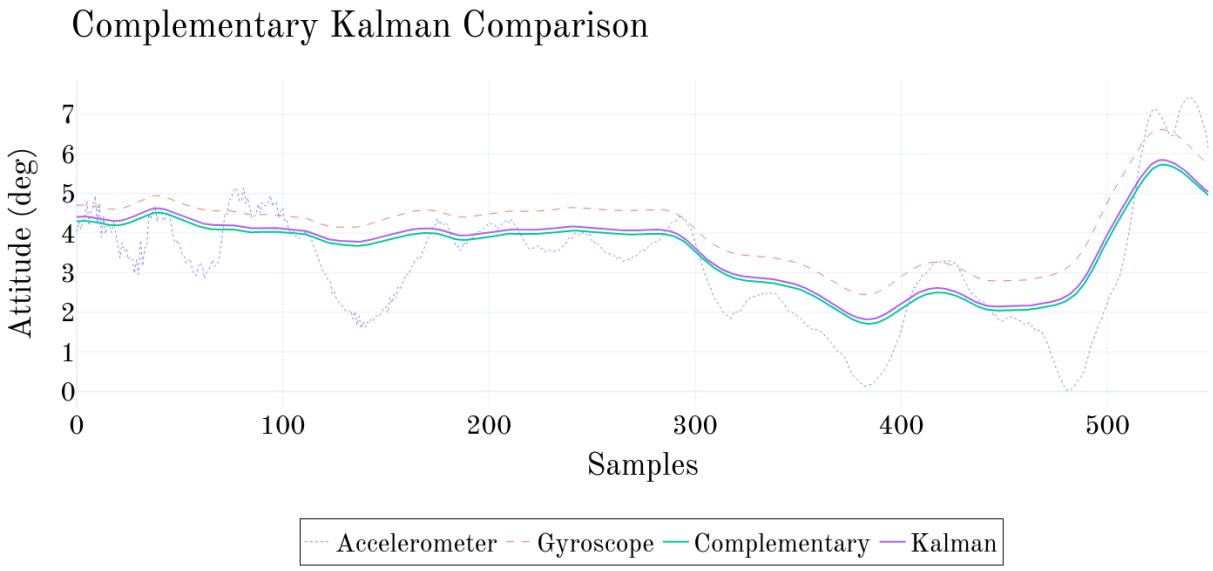


Figure 9.1: This diagram compares the Kalman filter algorithm with a Complementary filter. As can be seen, both approaches yield almost the same performance. The Kalman filter is a bit above the Complementary filter line because it was a bit more affected by the gyroscope drift than the Complementary filter, but this is adjustable by the Kalman gain parameter.

body and sensor frame. The next chapter describes the physics of quadcopter motion. A quadcopter steers by altering the motor speeds. The varying motor speeds allow the quadcopter to tilt around its x- and y-axis and rotate around its z-axis.

This work continued by explaining all the electronic principles needed to develop the quadcopter. The chapter has three sections, sensors, actuators, and communication technologies. The first section outlines what sensors the quadcopter contains and how the MEMS manufacturing method builds accelerometer and gyroscopes. Both sensors capture their data by measuring a capacity change. The accelerometer relies on the same principle as a dynamometer, and the gyroscope utilizes the Coriolis force. The section actuators explains why most of the quadcopters use brushless motors over brushed motors, its because the efficiency and lifetime of brushless motors are usually higher. The last section of this chapter, which is communication technologies, introduces the three technologies that the quadcopter uses to communicate with the sensors, the remote control of the pilot and the receiver, and ESCs. The 2.4 GHz communication uses a proprietary protocol that is not important for this work. The receiver and the ESCs communicate by pulse width

modulation (PWM), and the sensors publish their data over an I2C connection.

The first central part of this work is the filter algorithm that minimizes the noise of the sensor data to obtain a steady attitude estimation of the quadcopter. The first algorithm is the moving average filter that uses only the accelerometers' absolute measurements. The moving average filter returns a smooth value but at the cost of a slow filter response. The latter fact makes the moving average impractical for the quadcopter since a fast filter response is vital. The next algorithm is the Kalman filter. The Kalman filter is a professional and powerful approach to fuse multiple, by Gaussian noise affected, sensors readings together. The Kalman filter has a predict and an update step. The predict step integrates the gyroscope reading into the current state of the Kalman filter. Since the gyroscope reading is affected by a particular drift, the certainty of the Klaman filter decreases during this step. The update step incorporates a new measurement from the accelerometer. Even though the accelerometer data is affected by a sharp noise, the data adds a little more certainty to the Kalman state. How much impact the accelerometer data has is defined by the variance of the accelerometer data. While the Kalman filter is an advanced algorithm for data filtering, it is also complex to optimize. The complementary filter is a simple filter algorithm that has only one parameter to adjust. Where the Kalman filter uses variances to integrate new measurements, the complementary filter uses a single weight just like the moving average filter. The weight specifies how strong the measurement affects the prior. The prior is the result of the integration of the latest gyroscope data into the last attitude state.

The second part of the Control Theory chapter is the explanation of the PID controller, which controls the output signal to the motors. This work compares two PID controller setpoints. The first one defines the actual attitude of the multicopter as setpoint, and the second defines the angular velocity as the setpoint. This work finds that the second approach to control the angular velocity works better. The integral part of the PID controller is not needed since there is no offset that the PID controller has to compensate as it would be the case for an altitude control system, which has to generate an output to keep the quadcopter flying constantly.

The next chapter focusses on the actual implementation of the quadcopter control software. It explains many fundamental concepts of microcontroller programming like timers and peripheral access methods. The skew between the accelerometer and the gyroscope is another interesting aspect that this chapter explains. The accelerometer axes are offset from the gyroscope axes, which causes both sensors to output slightly different measurements. This

work proposes a linear scaling to mitigate the effects of the accelerometer skew. Another critical aspect is that when integrating the gyroscope over time, the z-axis rotation has to be taken into account. If the gyroscope is integrated without knowledge about the z-axis rotation, it will yield wrong values, as figure 8.7 shows. The last part of this chapter discusses the best filter algorithm for the quadcopter. As already explained, this work used the complementary filter because it is easier to tune and computationally less complex.

9.2 What can be improved

This chapter will explain which parts of the control software can be improved and extended.

The filter algorithm is a possible target for improvements. The filter does already output an almost optimal signal but is sometimes disturbed by stronger noises. There are multiple ways to optimize filter performance. The first option is to invest more time into filter tuning in order to achieve general lower noise levels in the filter output. This option might not reduce the noise bursts entirely but will help to obtain an overall more stable signal. The second option is to include another ground truth into the attitude-estimation. A magnetometer offers measurements of the magnetic field of the earth that can be used to estimate the orientation of the quadcopter. This estimation can extend the complementary filter by including another weight as equation 9.1 proposes. Both parameters W_a and W_m regulate how strong each measurement will be taken into account.

$$\begin{aligned} posterior =& (1 - W_a - W_m) * prior \\ &+ W_a * measurement_{accelerometer} \\ &+ W_m * measurement_{magnetometer} \end{aligned} \tag{9.1}$$

The third approach is to evaluate the plausibility of the accelerometer measurement and prevent distorted measurements from messing up the attitude estimation. The problem with the accelerometer is not only that its measurements are affected by intense noise, it is also that the accelerometer does not purely experience the gravitational acceleration. The accelerometer measurements are affected by any accelerations that the quadcopter experiences, including ascending, descending, and maneuvering around. These accelerations are not separated from the gravitational forces, which makes them disturb

the attitude estimations. A possible way to solve this issue is by validating the magnitude of all three axes and lower the weight of the complementary filter every time the magnitude exceeds or falls below a certain threshold.

The use of a magnetometer was already proposed to optimize filter performance. The magnetometer data would also allow us to set up a filter that can perform filtering for the z-axis rotation. Since the accelerometer can not measure the z-axis rotation, it is not possible now to build a filter.

Bibliography

[1] (Oct. 30, 2019). Dhl parcelcopter, [Online]. Available: https://www.dpdhl.com/en/media-relations/specials/dhl-parcelcopter.html.

[2] (Oct. 30, 2019). Agrasmg-1, [Online]. Available: https://www.dji.com/de/mg-1.

[3] (Jan. 1, 2020). Consumer drones market outlook, size and forecast 2017 to 2025, [Online]. Available: https://www.marketwatch.com/press-release/consumer-drones-market-outlook-size-forecast-2017-to-2025-2019-10-30.

[4] (Jan. 1, 2020). How the commercial drone market became big business, [Online]. Available: https://www.ft.com/content/cbd0d81a-0d40-11ea-bb52-34c8d9dc6d84.

[5] (Jan. 1, 2019). The drl racerx, [Online]. Available: https://thedroneracingleague.com/racerx/.

[6] (Jan. 1, 2019). The drone racing league, [Online]. Available: https://thedroneracingleague.com/.

[7] (Oct. 30, 2019). Wingcopter, dhl and giz pilot drone delivery in tanzania, [Online]. Available: https://wingcopter.com/deliverfuture/.

[8] S. Alabachi, G. Sukthankar, and R. Sukthankar, *Selfie drone stick: A natural interface for quadcopter photography*, 2019. arXiv: 1909.06491 [cs.RO].

[9] (Jan. 1, 2020). The worlds first v-shaped bi-copter, [Online]. Available: https://zerozerorobotics.com/.

[10] (Jan. 1, 2020). Hover 2 - the 4k drone that flies itself, [Online]. Available: https://www.kickstarter.com/projects/hover2/hover-2-the-4k-drone-that-flies-itself/comments.

[11] (Dec. 31, 2019). Crazyflie 2.0, [Online]. Available: https://store.bitcraze.io/collections/kits/products/crazyflie-2-0.

[12] H. A. Hashim, *Special orthogonal group so(3), euler angles, angle-axis, rodriguez vector and unit-quaternion: Overview, mapping and challenges*, 2019. arXiv: 1909.06669 [math.OC].

[13] H. Parwana and M. Kothari, *Quaternions and attitude representation*, 2017. arXiv: 1708.08680 [cs.SY].

[14] W. Koch, R. Mancuso, R. West, and A. Bestavros, *Reinforcement learning for uav attitude control*, 2018. arXiv: 1804.04154 [cs.RO].

[15] D. Trent. (Dec. 31, 2019). Load cell and strain gauge basics, [Online]. Available: https://www.800loadcel.com/load-cell-and-strain-gauge-basics.html.

[16] (Jan. 2, 2020). Strain gauge measurement - a tutorial, [Online]. Available: http://elektron.pol.lublin.pl/elekp/ap_notes/NI_AN078_Strain_Gauge_Meas.pdf.

[17] L. L. Silva, J. R. Rodrigues, A. Passaro, and V. R. Almeida, *Dynamic sensitivity study of mems capacitive acceleration transducer based on analytical squeeze film damping and mechanical thermoelasticity approaches*, 2017. arXiv: 1708.01812 [physics.app-ph].

[18] D. Nemec, A. Janota, M. Hrubos, and V. Simak, "Intelligent real-time mems sensor fusion and calibration", *IEEE Sensors Journal*, vol. 16, no. 19, pp. 7150–7160, Oct. 2016, ISSN: 2379-9153. DOI: 10.1109/jsen.2016.2597292. [Online]. Available: http://dx.doi.org/10.1109/JSEN.2016.2597292.

[19] H. Xie and G. Fedder, "Integrated microelectromechanical gyroscopes", *Journal of Aerospace Engineering*, vol. 16, Apr. 2003. DOI: 10.1061/(ASCE)0893-1321(2003)16:2(65).

[20] R. Condit. (Jan. 28, 2020). Brushed dc motor fundamentals, [Online]. Available: http://ww1.microchip.com/downloads/en/appnotes/00905a.pdf.

[21] W. Brown. (Jan. 28, 2020). Brushless dc motor control made easy, [Online]. Available: http://ww1.microchip.com/downloads/en/appnotes/00857a.pdf.

[22] C. McGrady. (Jan. 28, 2020). Brushless vs brushed motor: Best dc motor for your project, [Online]. Available: https://www.arrow.com/en/research-and-events/articles/which-dc-motor-is-best-for-your-application.

[23] R.
bibinitperiod Schwarz. (Jan. 19, 2020). Protecting the sky signal monitoring of radio controlled civilian unmanned aerial vehicles and possible countermeasures, [Online]. Available: http://www.rohde-schwarz-usa.com/rs/324-UVH-477/images/Drone_Monitoring_Whitepaper.pdf.

[24] J. Heath. (Jan. 18, 2020). Pwm: Pulse width modulation: What is it and how does it work?, [Online]. Available: https://www.analogictips.com/pulse-width-modulation-pwm/.

[25] N. Instruments. (Jan. 18, 2020). What is a pulse width modulation (pwm) signal and what is it used for?, [Online]. Available: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019OkFSAU.

[26] W. Gay, "I2c bus", in *Advanced Raspberry Pi: Raspbian Linux and GPIO Integration*. Berkeley, CA: Apress, 2018, pp. 259–281, ISBN: 978-1-4842-3948-3. DOI: 10.1007/978-1-4842-3948-3_15. [Online]. Available: https://doi.org/10.1007/978-1-4842-3948-3_15.

[27] ——, "I2c", in *Beginning STM32: Developing with FreeRTOS, libopencm3 and GCC*. Berkeley, CA: Apress, 2018, pp. 195–221, ISBN: 978-1-4842-3624-6. DOI: 10.1007/978-1-4842-3624-6_11. [Online]. Available: https://doi.org/10.1007/978-1-4842-3624-6_11.

[28] A. Subero, "Usart, spi, and i2c: Serial communication protocols", in *Programming PIC Microcontrollers with XC8*. Berkeley, CA: Apress, 2018, pp. 209–276, ISBN: 978-1-4842-3273-6. DOI: 10.1007/978-1-4842-3273-6_9. [Online]. Available: https://doi.org/10.1007/978-1-4842-3273-6_9.

[29] M. Ribeiro and I. Ribeiro, *Kalman and extended kalman filters: Concept, derivation and properties*, Apr. 2004.

[30] H. Masnadi-Shirazi, A. Masnadi-Shirazi, and M.-A. Dastgheib, *A step by step mathematical derivation and tutorial on kalman filters*, 2019. arXiv: 1910.03558 [stat.OT].

[31] R. Labbe. (Jan. 26, 2020). Multivariate kalman filters, [Online]. Available: https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/06-Multivariate-Kalman-Filters.ipynb.

[32] A. Mathai, *Probability and Statistics : a Course for Physicists and Engineers*. Berlin Boston: De Gruyter, 2017, ISBN: 978-3-11-056254-5.

[33] C. Peter, "Analysis of covariance", in *The International Encyclopedia of Communication Research Methods*. American Cancer Society, 2017, pp. 1–12, ISBN: 9781118901731. DOI: 10.1002/9781118901731.iecrm0005. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118901731.iecrm0005. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118901731.iecrm0005.

[34] "Basics of pid control", in *Practical PID Control*. London: Springer London, 2006, pp. 1–18, ISBN: 978-1-84628-586-8. DOI: 10.1007/1-84628-586-0_1. [Online]. Available: https://doi.org/10.1007/1-84628-586-0_1.

[35] "Anti-windup strategies", in *Practical PID Control*. London: Springer London, 2006, pp. 35–60, ISBN: 978-1-84628-586-8. DOI: 10.1007/1-84628-586-0_3. [Online]. Available: https://doi.org/10.1007/1-84628-586-0_3.

[36] STMicroelectronics. (Jan. 30, 2020). Um1718 user manual, stm32cubemx for stm32 configuration and initialization c code generation, [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/user_manual/10/c5/1a/43/3a/70/43/7d/DM00104712.pdf/files/DM00104712.pdf/jcr:content/translations/en.DM00104712.pdf.

[37] ——, (Jan. 30, 2020). An4989 application note, stm32 microcontroller debug toolbox, [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/application_note/group0/3d/a5/0e/30/76/51/45/58/DM00354244/files/DM00354244.pdf/jcr:content/translations/en.DM00354244.pdf.

[38] ——, (Feb. 1, 2020). An4776 application note, general-purpose timer cookbook for stm32 microcontrollers, [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/application_note/group0/91/01/84/3f/7c/67/41/3f/DM00236305/files/DM00236305.pdf/jcr:content/translations/en.DM00236305.pdf.

[39] J. E. Volder, "The cordic trigonometric computing technique", *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959, ISSN: 0367-9950. DOI: 10.1109/TEC.1959.5222693.

[40] STMicroelectronics. (Feb. 1, 2020). An5325 application note, getting started with the cordic accelerator using stm32cubeg4 mcu package, [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/application_note/group1/50/31/98/a8/

b5/da/4e/a4/DM00614795/files/DM00614795.pdf/jcr:content/translations/en.DM00614795.pdf.

[41] Z. Chen, N. Ahmed, S. Julier, and C. Heckman, *Kalman filter tuning with bayesian optimization*, 2019. arXiv: 1912.08601 [eess.SY].

[42] S. M. M, N. Naik, R. M. O. Gemson, and M. R. Ananthasayanam, *Introduction to the kalman filter and tuning its statistics for near optimal estimates and cramer rao bound*, 2015. arXiv: 1503.04313 [stat.ME].

[43] M. R. Ananthasayanam. (Jan. 18, 2020). Tuning of the kalman filter using constant gains, [Online]. Available: https://www.intechopen.com/books/introduction-and-implementations-of-the-kalman-filter/tuning-of-the-kalman-filter-using-constant-gains.

[44] T. Ting, K. Man, E. Lim, and M. Leach, "Tuning of kalman filter parameters via genetic algorithm for state-of-charge estimation in battery management system", *TheScientificWorldJournal*, vol. 2014, p. 176 052, Aug. 2014. DOI: 10.1155/2014/176052.

List of Figures

List of Tables

Acronyms

ESC Electronic Speed Control. Controls the speed of brushless and brushed motors. 7, 8, 24, 25, 27, 62, 71, 72

I2C I2C. Communication technology for peripherals. 27–29, 48, 57, 71, 73

IDE Integrated Development Environment. A program for writing source code with an integrated compiler, debugger, and other tools. 50

IMU Inertial Measurement Unit. Controls the speed of brushless and brushed motors. 5, 68, 69

PDF Probability Density Function. Describes the likelihood of a certain random variable. 33

PID Proportional Integral Derivative. A mechanism to regulate a dynamic process. 43–46

PWM Pulse Width Modulation. A communication technology which transfers data by varying the length of high level signals. 25–27, 45, 57, 60–63, 70, 71

SWV Serial Wire View. A technology for tracing STM32 microcontroller. 50, 52, 53

Erklärung zur Abschlussarbeit

Hiermit versichere ich, die eingereichte Abschlussarbeit selbständig verfasst und keine andere als die von mir angegebenen Quellen und Hilfsmittel benutzt zu haben. Wörtlich oder inhaltlich verwendete Quellen wurden entsprechend den anerkannten Regeln wissenschaftlichen Arbeitens zitiert. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht anderweitig als Abschlussarbeit eingereicht wurde.

Das Merkblatt zum Täuschungsverbot im Prüfungsverfahren der Hochschule Augsburg habe ich gelesen und zur Kenntnis genommen. Ich versichere, dass die von mir abgegebene Arbeit keinerlei Plagiate, Texte oder Bilder umfasst, die durch von mir beauftragte Dritte erstellt wurden.

_____ _____

Ort, Datum Unterschrift des/der Studierenden