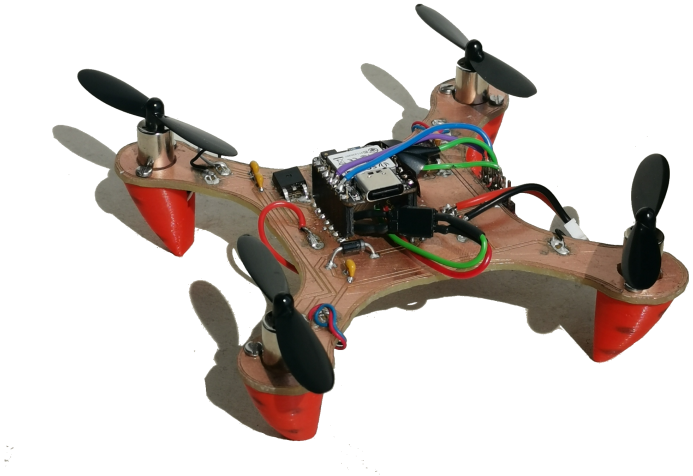
SDU

BEng Mechatronics

University of Southern Denmark Sønderborg

Semester Project 4 - Ladybug



Ruta Miglava, Johan Paul, Choyon Mainul Hasan, Thor Uerkvitz, Artis Fils, Tobias Blaabjerg Karlsen

Supervised By

Lars Duggen

Davi Goncalves Accioli

June 2023

Contents

1 Background

Possessing the ability of flight and minimising effort and casualties has always been desirable for the utility flight can provide. The first unmanned aircrafts can be dated back to 1849, where Austria seemingly had utilised unmanned air balloons with stuffed explosives to attack Venice. [?] Ever since an unmanned aircraft vehicle (UAV), is one that is flown by technological means or as a pre-programmed flight without pilot control, as defined by the ECAA Transport Agency [?], nowadays called drones, have risen in popularity.

Because of this, UAVs come in a wide range of sizes and weights. UAVs often include multirotor, radio-controlled miniature helicopters, and aeroplanes [?]. As a result, there are several methods to categorize drones. The performance parameters of UAVs, such as weight, wingspan, wing load, flight range, maximum flying altitude, speed, and production cost, are typically used to categorize them [?]. According to how the lift is produced, drones may also be divided into fixed-wing and rotating-wing types. According to the drone code category, the European Aviation Safety Agency (EASA) categorizes unmanned aircraft by weight. The EASA regulations for open categories, or drones without an EASA class designation, are summarized succinctly and simply in Figure ?? [?].

Self-built drones weighing up to 250 g, as described in Figure ??, may be used without registration if the drone is a toy or the drone is not equipped with a camera, the remaining drones must be registered, and the pilot must pass examinations [?]. In this paper, self-built rotary drones with four wings or propellers are the objective, making weight-based classification suitable.

UAS	Operation		Drone operator/pilot		
Max weight	**Subcategory**	**Operational restrictions**	**Drone operator registration**	**Remote pilot competence**	**Remote pilot minimum age**
< 250 g	A1 (can also fly in subcategory A3)	— No flight expected over uninvolved people (if it happens, overflight should be minimised) — No flight over assemblies of people	No, unless camera / sensor on board **and** the drone is not a toy	— No training required	No minimum age
< 500 g			Yes	— Read carefully the user manual — Complete the training and pass the exam defined by your national competent authority or have a 'Proof of completion for online training' for A1/A3 'open' subcategory	16*

Figure 1: Classification and restrictions for non-EASA class drones [?]

When it comes to the state-of-the-art project, PULP-DroNet is a deep learning-powered visual navigation engine that enables autonomous navigation of a pocket-size quadrotor in a previously unseen environment. Thanks to PULP-DroNet the nano-drone can explore the environment, avoiding collisions also with dynamic obstacles, in complete autonomy – no human operator, no ad-hoc external signals, and no remote laptop! This means that

all the complex computations are done directly aboard the vehicle and very fast. The visual navigation engine is composed of both a software and a hardware part. [?]

When it comes to the future, the simulated pollination of agricultural plants by means of nano copter can provide collecting and delivering pollen in the mode of automatic control. A design of nano copter for pollination can be made on the basis of innovative modification of existing model by its reprogramming with regard to its flight controller that is to be fully adapted to computer interface. The robotic system is offered specially for artificial pollination in conditions of greenhouses and minor agricultural enterprises. [?]

2 Problem statement

The utility of smaller drones are immense, where it can be used in surveillance, toys and potentially to also be part of a swarm of drones. Although, there are smaller drones existing in the current market, we would like to challenge ourselves to build one ourselves, where certain goals ranging from functionality to budget are listed below.

2.1 Primary goals

- Net maximum weight of the drone is 250 grams. Weight under 250 grams ensures it falls under A1 category in EU regulations. 1
- Flight time of 20 seconds.
- Stress of the structural system should not exceed rupture point. System does not experience fracture.

2.2 Secondary goals

- Flight time of minimum one minute.
- Can land with acceleration less than 9.8 m/s2.
- Stress of the drone system should not exceed the yield point. System does not experience plastic deformation.
- Drone is remote controllable.
- Drone can fly in formation with another identical drone.
- Total production cost of the drone is under 500 DKK (Not including remote controller).
- Drone can play audio.

2.3 Constraints

- Budget for entirety of project is 2000 DKK.
- Time available to finish the project is 4 months.
- Drone should have a minimum hover time of 5 seconds.
- Drone should be fully functional and able to take off again after landing.
- No use of flight controller software or unmanned vehicle Autopilot software Suite, capable of controlling autonomous vehicles.

3 Test Specifications

3.1 Primary goals:

- To test this, the drone will be weighed with a scale of a precision on 0,1 grams.
- In order to test the flight time, a stopwatch will be started from the moment the drone leaves the ground and is stopped as soon as it lands.
- This goal will be the tested trough FEM, ensuring that the chosen material for the drones body, will not rupture.

3.2 Secondary goals

- This will be tested with the same method as primary goals tests point 2.
- This will be tested with a mobile phone, recording the drones landing, using the drones position compared to the timestamp of the video.
- This will be tested with the same goals as primary goals test point 3.
- This will be tested by the possibility of sending wireless signals to the drone, with the drone reacting to those send signals.
- This will be tested purely by ear, listening to the drones output.
- This will be tested by mobile phone video, looking at the drones positions at given timestamps.
- This will be tested trough summing the price for each single part, ensuring that it doesn't exceed 500 DKK.

3.3 Constraints

- This will be done with the same method as the secondary goals test, though ensuring the project cost is over 2000 DKK.
- To evaluate the time constraint point of the project, the goal fullfilments will be evaluated in the end of the project period. In the case that all primary goals are fulfilled, the constraint is succeeded.
- This will be tested with a stopwatch, ensuring that the hover time is atleast 5 seconds.
- This will be tested with making the drone take off right after a landing, making sure that the drone is fully operational at the second take-off.
- This will fulfilled by not employing any of the aforementioned in the drone.

3.4 Morphology table

In this section the morphology table is provided where the specific functions of the product and their potential solutions are provided. Three different possibilities for a combination of solutions were chosen in total and they are marked with the following respective colours.

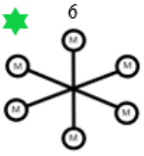
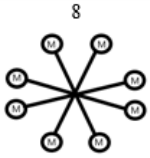
Functions	Solutions			
Number of Propellers	4 	6 	8 	
Motor type	DC 	Brushless DC 	Stepper 	
Drone MCU	Seeed Studio XIAO nRF52840 Sense	Arduino Nano 33	ESP32	
Remote controller MCU	Raspberry Pi Pico	ESP32	Seeed Studio XIAO nRF52840	
Remote controller communication	BLE	NRF24L01	MQTT	CoAP
Remote controller physical	Joystick	Buttons	Switch	Touchscreen
Battery	Li-Po	Li-ion	Ni-Cd	LiFePO
IMU	6-axis	9-axis		
Logging	SD card	Wireless data transfer	Flash memory	

Figure 2: Morphology table for quadrotor design

3.5 Evaluation matrix

Below is the evaluation matrix, where all the combinations of solutions are graded according to each criterion with a certain importance. The criterions stem from the project goals and constraints. For every combination each grade given to a criterion is multiplied with the importance and then added with the other grades in their respective criterions multiplied with their importance.

Criteria	Importance	Solution 1	Solution 2	Solution 3
Manufacturing	2	3	2	1
		Absence of ESCs and a smaller number of propellers and motors make the drone easier to configure and manufacture.	Getting the ESCs for the BLDC motors and 9-axis IMU to function contains a longer time commitment.	Configuring ESCs and 6 motors to work together requires higher time commitment.
Weight & compactness	3	3	2	2
		Li-Po batteries contribute more compactness and less weight.	Li-ion batteries weigh more than Li-Po batteries.	Li-ion batteries weigh more than Li-Po batteries.
Cost	1	3	2	1
		Brushed DC motors and 6-axis IMU contribute to cheap costs.	BLDC motors are generally more expensive than brushed DC motors.	Higher number of motors equal higher costs.
Performance (thrust and reaction time)	3	2	3	2
		6-axis IMU cannot accurately measure yaw of the drone.	9-axis IMU gives more accurate readings, and the BLDC motors provide more thrust and torque.	6-axis IMU cannot accurately measure yaw of the drone.
Total		**24**	21	15

Figure 3: Quadrotor evaluation matrix

3.6 Risk Assesment

To ensure the safety under operation and development of the drone, a risk assessment is created for the drone project, based on the Fine & Kinney method [**?**].

#	Risk	Description	Actions to minimize risk
1	**Safety risks in a operational perspective**		
1.a	**Collision** [R=1.5]	Collisions by the drone flying into either items or persons	Ensuring that the drone is only operated in enclosed spaces or under restraints, so that the drone is unable to fly into unwanted areas.
1.b	**Wear/breakage of components** [R = 0.3]	E.g. by a propeller being operated in a broken condition	Inspecting the propellers and other vital components before every flight or test session.
1.c	**Battery charging malfunctions, fire** [R = 1.75]	Charging the battery with incorrect charger or under undesirable conditions	Ensure that the batteries are only charged with the correct charger.
2	**Safety risks concerning the development phase**		
2.a	**Collision** [R=1.5]	Collisions caused to malfunctions in sensor readings or adjustments of control software giving unwanted results	Ensuring that the drone is only operated in enclosed spaces or under restraints, so that the drone is unable to fly into unwanted areas.
2.b	**Wear/breakage of components** [R = 1.5]	Incorrect mounting of components causing breakage or failure	Checking components after mounting, ensuring correct mounting
2.c	**Battery charging malfunctions** [R = 1.75]	Charging the battery with incorrect charger or under undesirable conditions	Ensure that the batteries are only charged with the correct charger.
2.d	**Shorting** [R = 6]	Having unwanted connections in the produced PCB or a wire connection soldered onto an incorrect pad.	Checking all connections compared to the schematic and checking for unwanted shorts with a multimeter in continuity mode
2.e	**Cuts** [R = 0.1]	Getting cut by propellers	Never operating the drone unrestrained and close to people.

Figure 4: Risk assesment table

Marked in all the risks are the risk scores calculated by the Fine & Kinney method of Risk, Risk score = Probability * Exposure * Consequence. These risk scores are then compared in the risk clusters of the method, with five defined classes. After calculating the proposed risk ratings of the found risks, it's apparent that all risks fall under the lowest factor of the Fine & Kinney method of scores the: 'Risk; Perhabs acceptable category'.

3.7 GitHub Interaction

Each semester project has a main focus-point for educational purposes. The main focus-point for SPRO4 is control engineering which covers implementations such as IMU combined with programming and PID for the drone. This means a lot of the project development will be software related.

Working together in a large group for a single deterministic purpose can be cumbersome, so in order to better track progress, a GitHub repository has been made to share files and collaborate more effectively. GitHub requires little to no extra effort to implement and use, and will only require a small change in work culture to use correctly, potentially saving a lot of time.

The GitHub repository is an easy way to share and keep track of the software development progress, and gives many advantages when working with embedded systems.

3.7.1 Advantages of Using GitHub

- GitHub works by having one main code or branch which is shared between everyone. Users can then create their own branches and work on something individually, before it is sent back to the main. This means past revisions of work can be saved and reviewed by everyone, and changes can easily be reverted if a fundamental mistake occurs- or is spotted during development. Mistakes in the software are also easier to identify and can be solved without having to change the main program at all. This also means that changes can be made without interfering with other peoples workflow, giving a clear advantage when working/collaborating on similar problems in the software. [**?**] With GitHub, open source software becomes easily available and secure, while also being compatible with other GitHub repositories. This means GitHub can also be used to keep dependencies up to date for already existing projects, and provide live updates to multiple libraries and dependencies automatically and safely. [**?**]

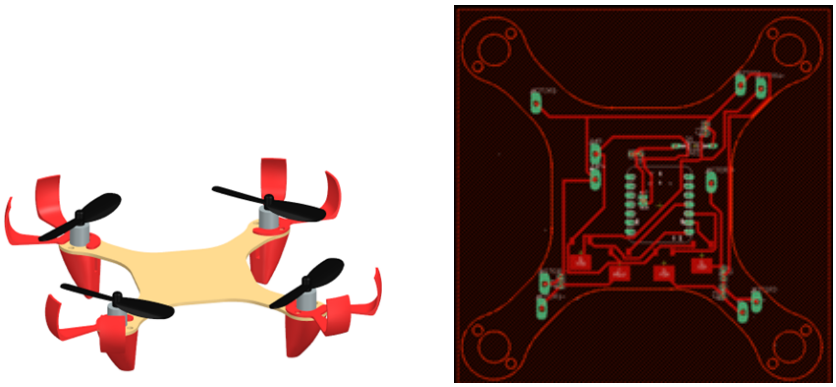
3.7.2 Usefulness Beyond the Project

GitHub is very standard among many engineering companies, and is one of the most widespread UML service. Its used by many companies in the industry, so its very useful to familiarize with GitHub before entering jobs, especially in software development. And with GitHub's wide usage and accessibility it has quickly become the worlds largest software collaboration tool in the world with over 100 million developers. [**?**] GitHub is also owned by Microsoft, and works with Microsoft programs, which is very standard for many work environments, providing high compatibility.

All in all, GitHub seems to be a necessary skill to have in the future as an engineer.

4 Design and manufacturing of quadrotor

The main body of the drone is created as a PCB, in order for the body to host both the structural element and the electrical connections of the drone. Since the project formulation's main goals dictates the drones size and mechanical properties, the drones size was chosen to be $100mm^2$, so that the PCB could accommodate all the needed components of the drone. The PCB shape itself was designed in Siemens NX as a simple outline with holes for the motor mounts. This outline was exported in a DXF format into Autodesk Eagle were the actual PCB was designed from the needed electrical circuit. The electrical schematic was created in Eagle's schematic designer and then automatically converted to a PCB board with the auto-router feature. The trace width of the PCB is specifically chosen to be 30 mills so it can supply the needed 1.72 amps [**?**] for the motors at maximum rpm. The microcontroller of the drone is mounted by pin headers, the motors by the motor mounts and the remaining components by soldering onto the top layer. The position of the microcontroller is offset from the center, in order to ensure that the In-built 6-axis IMU is centered to the exact middle of the PCB.



(a) CAD View (b) PCB View

Figure 5: Drone model

In order to control the speed of the motors, one of the potential options would be to use an H-bridge to control the DC motors. However that would be excessive, as the motors does not need to run in reverse as contain many components As a less power consuning simple and lighter option, a single N-channel mosfet was used for each motor.

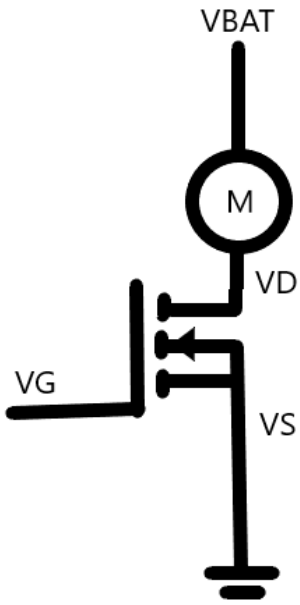
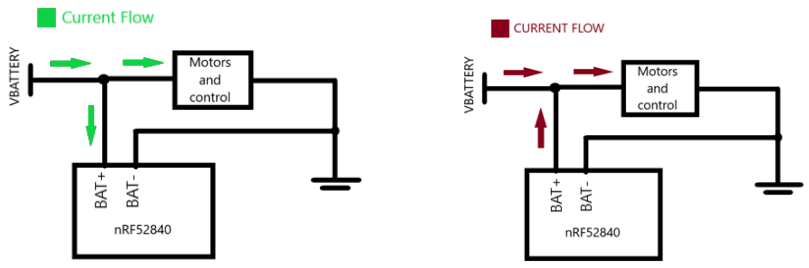


Figure 6: Motor control with MOSFET.

As illustrated in figure **??**, the drain of the MOSFET is connected to the motor, which is supplied by the battery, and the source of the MOSFET is grounded. Meanwhile, the VG pins are connected to one of the PWM pins of the nRF52840 MCU, where the opening of the gate is proportional to the PWM. Thus, when VG (simulated by PWM) is smaller than the V threshold of the MOSFET, the motors are static, and if VG surpasses V threshold, then the motors start spinning with higher RPM as PWM is increased. The MOSFETs used for this situation are FDD8896 [**?**], as it has a low threshold voltage of 2.5V (MCU pins can supply up to 3.3V), and can handle up 94A in continuous drain current.

As there is high currents being drawn from the motors, one of the things needed to be configured was protection against overcurrent being drawn from the microcontroller. When the motors are switched on or off, a surge current arises which can result in additional overcurrent being pulled from the microcontroller if the battery is not alone capable of supplying the current.



(a) Continuous Current (b) Potential Surge current

Figure 7: Current flows

In order to prevent reverse current from the MCU, during a surge, one of the potential options to utilise would be a diode placed in the following configuration.

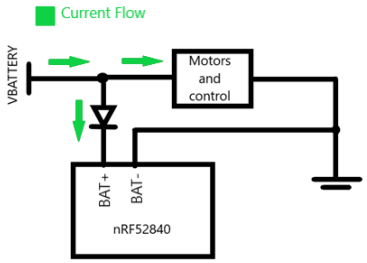


Figure 8: Diode added to prevent reverse current flow from MCU.

Additionally decoupling capacitors were added in parallel to the positive and negative motor pins and the MCU BAT+ and BAT- pins.

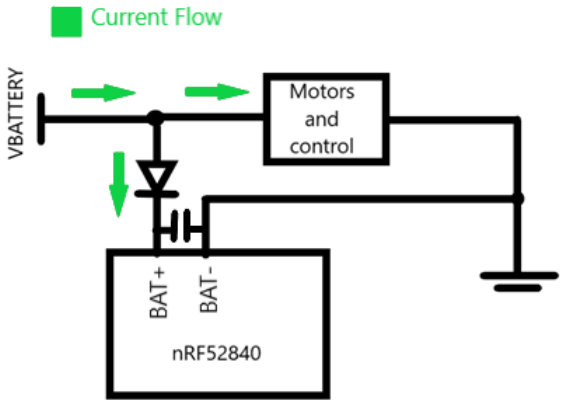


Figure 9: Example of decoupling capacitor for stable voltage to MCU.

The diode utilised is 1N5818 [**?**], as it has a low forward voltage of under 0.5V, resulting in low power losses. Moreover, the decoupling capacitors are cermaic capacitors as they generally smaller in size, and they are rated at 100 nF, which follows the general guidelines of decoupling capacitor values. [**?**]

With the combination of the CAD-outline and the electrical schematic, PCB Gerber files could be output for production. The only excess parts needed for the drone would be motor mounts and prop guards (prop guards solely for testing, not for final use). The Motor mounts act as both landing legs and motor mounts. They would be designed in order for the motors to press fit into the central hole, mounted onto the PCB by screws into the motor mount going through the PCB. The designed mounting parts was printed in PLA on a Prusa MK3S+ 3D printer. The Parts were printed with a single perimeter and 3% infill in order for each leg to weigh 1.1 grams while still having the nescessary structure to have a stable press-fit.

(a) Motor Mount (b) Mounted Motor (c) Prop Guard

Figure 10: 3D-Printed drone parts

The project formulation states that the "Stress of the drone's structural system should not exceed rupture point. System does not experience fracture". This was tested through Ansys Mechanical Static Structural Analysis. Using the CAD model of the drone, the following boundary conditions were applied: Fixed constraint on the bottom area (excluding the drones arms), Z direction forces (upwards) was applied on each of the motor mount screw holes according to the found motor thrust, a single force vector in the Z direction (downwards) was applied to the top face (excluding the drones arms), to resemble the gravity force caused by the weight of the drone. Besides the boundary condition, the setup contained a mesh refinement of 3 steps, a material assignment of FR4 Fiber Glass Epoxy Board and a solution setting of equivalent stress. Below is a picture of the stress simulation of the initial design, which showed clear stress concentrations in the corners where the motor arms runs into the body section. The stress concentration had a magnitude of 1.6839 MPa which is satisfactory regarding the requirement of fracture, since the tensile strength of the given material is 320 MPa [**?**].

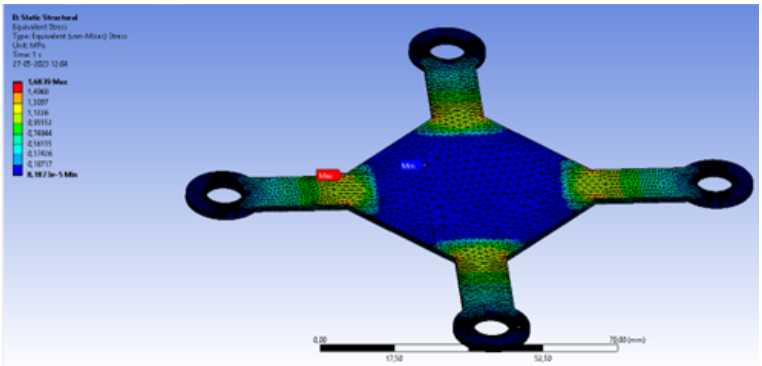


Figure 11: Initial Ansys Simulation

Although it was clear that the drone body would be strong enough to operate under max power, the stress concentrations were undesirable. Therefore 20 mm radii was added to the stress all corner points of the drone body. Repeating the simulation with the same boundary conditions, it was clear that the added radii removed the stress concentrations and instead distributed the stress over a broader area of the motor arms. Furthermore, it

was found that the maximum stress was found to be 0.87154 MPa, which is approximately half the maximum strength of the initial model that had the stress concentrations.

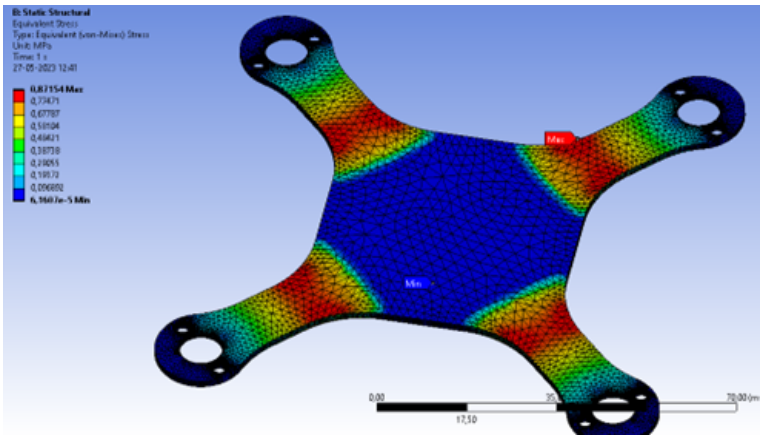


Figure 12: Final Ansys Simulation

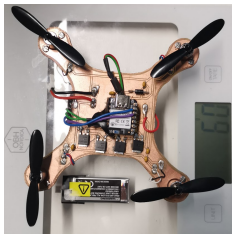
To power the drone a single cell Turnigy Li-Po battery is used, at 3.7 - 4.2 V and 300 mAh. The battery is chosen specifically to be able to supply the needed current of the drone. With a C-rating of 35C, the battery is able to discharge continously at 10.5 amps which is sufficent for the electrical circuit. The motors used are 8520 brushed Coreless DC-motors, which properties are characterized in the control chapter under thrust testing using the stock 2-bladed propellers that the motors were shipped with.

(a) Turnigy Battery (b) 8520 DC-motor

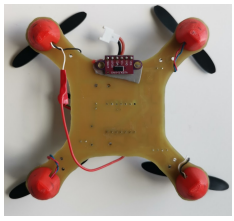
Figure 13: Motor and Battery used on the drone

After assembly of the drone including all the parts, it became apparent that the initial weight estimation of the drone was too low. After weighing the final physical setup it weighed 70 grams, XX grams over the intital estimation. This of course meant that the drones motors had less excess thrust than initially expected, meaning less control authority. In an attempt to mitigate this, the option of 3-bladed propellers were explored since they in theory would be able to provide extra thrust. The 3-bladed propellers were printed with SLA in the Tough 2000 resin on a Form 3 machine. Testing the new propellers in the same thrust setup used to characterize the motors using the 2-bladed propellers, it

showed that the 3-bladed propellers produced 12 grams total extra (combining 4 motors), but also pulled a total of 4 amps extra. Since the battery used does not have the sufficient continous amp supply this option was not further explored, instead lightweighting options was used to reduce the total weight to 60 grams. In order to have an alitude feedback a GY-VL53L0XV2 ToF sensor was emplyed on the underside of the drone, to measure the drone-to-ground distance. In development a budget was created to ensure that the possibilty of achieving the project goals was always possible, the budget for the project can be found in appendix **??**.



(a) Motor Mount



(b) Mounted Motor

(c) Prop Guard

Figure 14: 3D-Printed drone parts

5 Control prototyping

5.1 Plant Model Equations

This section of the report will explain how the plant model equations are obtained. Followed by how these equations are implemented in MATLAB/Simulink.

The plant model is a series of equations with multiple variables, which uses inputs, states, and parameters to get a certain output. For this project, the plant model is designed to take a height input the drone should hover at, paired with the inputs from the sensors, to achieve the desired hover height and stability.

5.1.1 State Equations

The drone possesses 6 degrees of freedom, enabling it to navigate in various directions and perform rotations. It can move along the X, Y, and Z axes, while also being capable of pitch (rotation around the Y axis), roll (rotation around the X axis), and yaw (rotation around the Z axis). We can identify the precise angle and position of the drone at any time by defining the axis of rotation and its accompanying state variables. [**?**]

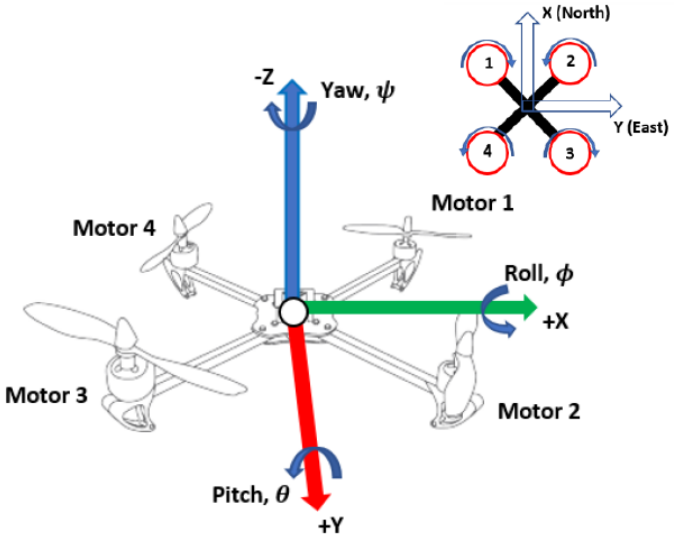


Figure 15: Quadcopter 'x' configuration

The drone's states include critical physical properties like as acceleration, velocity, and position, which are all detected by sensors. A gyroscope is used to record the angular changes of the drone. An accelerometer is a device that measures acceleration along the x, y, and z axes. Since the gyroscope drifts and the accelerator has measurement spikes when subjected to high acceleration, a complementary filter will be needed, which will be discussed in the "IMU" subsection. A dedicated infrared sensor is also used to measure distances in the z-direction, allowing for exact location information. A state table can be found in Appendix E.

Before plant model equations can be generated, parameters for the drone need to be calculated. The parameters are what explains the drone mathematically and is an integral part of modelling the drone. The more accurate these values are the more realistic the plant model will be. The parameters needed can be seen in the table below: [?]

Symbol	Description	Unit
I_{xx}, I_{yy}, I_{zz}	Inertia moments	$Kg\ m^2$
J_r	Rotor inertia	$Kg\ m^2$
l	Rotor axis to copter center distance	m
b	Thrust coefficient	N/s^2
d	Drag coefficient	Nm/s^2

Figure 16: State table

5.1.2 Inertia Moments

A moment of inertia is a constant value that explains the force/torque required to rotate an object in the direction of the moment. A desired angular velocity around a rotation

axis can be found. The moments of inertia are found using estimation. If it is assumed that the drone is symmetrical in all axis, the moments of inertia can be found using parallel axis theorem to approximate the moments for each individual part of the drone in the x, y, z directions. For the calculations it's assumed that the centre of mass for the drone is perfectly in the centre of the drone's body. The equations used for each individual part of the drone can be found in. [?]

For the approximation smaller electrical components were not considered, such as wires and mosfets, since their collective total weight is less than a gram, they are deemed negligible for the accuracy estimation.

Motor MoI	$I_{motor,x} = 2\left(\frac{1}{12}m(3r^2 + h^2)\right) + 2\left(\frac{1}{12}m(3r^2 + h^2) + md_{a2a}^2\right)$ $I_{motor,y} = 2\left(\frac{1}{12}m(3r^2 + h^2)\right) + 2\left(\frac{1}{12}m(3r^2 + h^2) + md_{a2a}^2\right)$ $I_{motor,z} = 4(\frac{1}{2}mr^2 + md_{a2a}^2)$
Arms MoI	$I_{Arm,x} = 2\left(\frac{1}{12}m(w^2 + h^2)\right) + 2\left(\frac{1}{12}m(h^2 + d^2) + md_{a2a}^2\right)$ $I_{Arm,y} = 2\left(\frac{1}{12}m(w^2 + h^2)\right) + 2\left(\frac{1}{12}m(h^2 + d^2) + md_{a2a}^2\right)$ $I_{Arm,z} = 4(\frac{1}{12}m(w^2 + d^2) + md_{a2a}^2)$
Battery MoI	$I_{Battery,x} = \frac{1}{12}m(w^2 + h^2)$ $I_{Battery,y} = \frac{1}{12}m(h^2 + d^2)$ $I_{Battery,z} = \frac{1}{12}m(w^2 + d^2)$
Flight Controller MoI	$I_{FC,x} = \frac{1}{12}m(w^2 + h^2)$ $I_{FC,y} = \frac{1}{12}m(h^2 + d^2)$ $I_{FC,z} = \frac{1}{12}m(w^2 + d^2)$
Rotor moment of inertia	$J_r = \frac{1}{2}mR^2$

Figure 17: Inertia moments equations

A mathematical document can be found in the appendix as "SPRO4math". Which shows the detailed process of calculation.

$$I_{rotor} := 729$$

$$I_x := 48932.24391$$

$$I_y := 49929.46266$$

$$I_z := 99299.32284$$

Figure 18: Moments of inertia in the x, y, z axis, in $g * mm^2$ as calculated from equations, used in the final version of the quadcopter

5.1.3 Characterization of Motor

Some drone parameters can not be determined due to a lack of data concerning the motors used in the project. The motors are of a commercial product, but with no available data sheet. This means that motor constants and thrust coefficients needs to be determined through testing. Other motors within the same weight class were available, but those other alternatives did not provide the thrust desired compared to their weight, since the weight requirement set for the project is around 40 grams. For the same reason the motors currently in use are good, since they have a good thrust to weight ratio, with the caveat of not having a data sheet
The next part of the report is a small summary of the tests done to determine the data needed for the motors. The full reports concerning testing and acquiring these values can be found in the appendix.

- Thrust testing can be found in appendix "Motor Thrust Testing"
- Motor constants testing can be found in appendix "Motor Testing"

5.1.4 Thrust testing

For thrust testing the motor is placed in a frame on a scale to measure the change in weight when the motor is operating. A sketch of the setup can be found in appendix A2,2. The motor used had an operating voltage window of 0-4,2 volts, therefore the motor was tested between 1-4,2 volts incremented with 0,1 volts, and the weight change noted for each increment. Characterizing the motor yielded the results seen in the figure below. As a general rule of thumb, provided by project supervisors, the motor's need to produce double the thrust of the total weight of the drone to operate ideally. At maximum voltage level of the battery, a single motor produces 22.2 grams of thrust. In conclusion,

at maximum voltage level the drone can ideally weigh 44.4 grams in total. The battery will slowly discharge meaning that it's impossible to always have max output, so it should be a little less than 44.4 grams in practice.

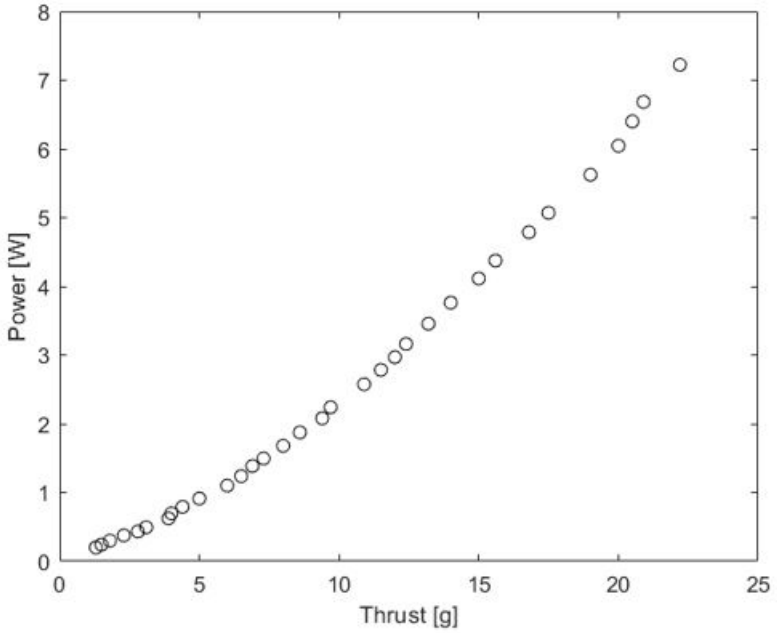


Figure 19: Motor Thrust Graphed

5.1.5 Motor Constants, and Thrust Coefficient

Motor constants relevant for this project include, "Motor velocity constant" (K_v) and "Motor torque constant" (K_T) With a simple setup the motor's RPM can be tested proportionally to PWM. The motor was tested in 20 increments, using a photo sensor to measure RPM. Using the data from the test, we get the following result. [**?**]

$$K_{\mathrm{v}} = \frac{\omega_{\text{no-load}}}{V_{\text{peak}}} \qquad K_{\mathrm{T}} = \frac{\tau}{I_{\mathrm{a}}} = \frac{60}{2\pi K_{\mathrm{v(RPM)}}}$$

Figure 20: Equations used for calculating thrust- and velocity constants

$$K_v := \frac{\omega}{V} = 8943.333333 \frac{RPM}{V}$$

$$K_\tau := \frac{60}{2 \cdot \pi \cdot K_v} = 0.001067755861 \frac{N \cdot m}{A}$$

Figure 21: Motor Constants

Combining the two test data sets it is possible to determine the thrust coefficient of the motor. Using the graph from the thrust test the thrust can be calculated by taking the voltage drop over the motor. This thrust is then divided with the angular velocity to get the thrust coefficient ($TC = 3.59E - 05$).

The constants/coefficients are calculated around 3.0V inputs, which is the voltage needed to achieve the theoretical hover thrust. This is sensible since most of the time the drone will be operating around these values.

5.1.6 Environmental Forces

The drone is very small, weighing only approximately 55grams as per requirement and with little surface area. Because of this, aerodynamic effects have been neglected for the modelling. Environmental forces mainly include but are not limited to wind resistance and liftoff turbulence.

5.1.7 Generating Plant Model Equations

Now with the all the states and parameter defined, plant model equations can be generated. First, we will look at how the motors should operate together to move the quadcopter in the desired direction. A quadcopter can move in x and y directions using pitch and roll. When the quadcopter rotates around one of its axis, all the rotor blades will generate thrust at an angle, moving the drone in a direction. It is important to control the individual motors so the drone won't tilt too much, causing it to flip in the air. On the picture below the different cases of directional movement for the quadcopter can be seen.

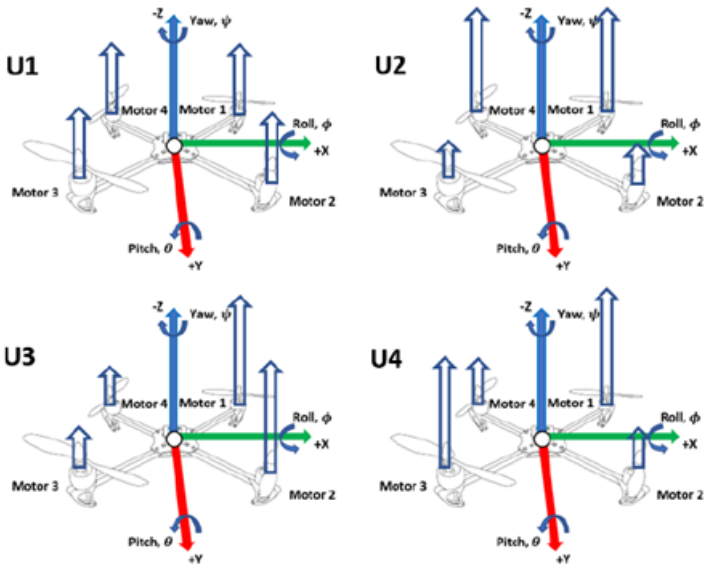


Figure 22: Visual representation of thrust equations [?]

Here U1 is the general thrust behaviour, providing thrust equally and symmetrically to move only the Z axis. U2 creates a roll, thrusting the drone in the Y direction. U3 creates a pitch, thrusting the drone in the X direction. U4 increases the thrust for motors turning the same direction and decreases the thrust of the motors rotating the opposite direction, allowing the drone to rotate in the Z axis e.g., yaw due to non-symmetrical rotational forces created by the moments of the motors. [?]

Each motors thrust is then defined as angular rotational speed to acquire the following equations.

Input	Thrust Equation	Description	
$U1_x$	$b(\Omega_1^2+\Omega_2^2+\Omega_3^2+\Omega_4^2)$	General Thrust	(2.8)
$U2_x$	$b\sin\left(\frac{pi}{4}\right)(\Omega_1^2-\Omega_2^2-\Omega_3^2+\Omega_4^2)$	Roll Thrust	(2.9)
$U3_x$	$b\sin\left(\frac{pi}{4}\right)(\Omega_1^2+\Omega_2^2-\Omega_3^2-\Omega_4^2)$	Pitch Thrust	(2.10)
$U4_x$	$d(\Omega_1^2-\Omega_2^2+\Omega_3^2-\Omega_4^2)$	Yaw Thrust	(2.11)

Figure 23: Thrust equations [?]

The plant equations for this project were gotten from source [?]. The source has a more in depth description of how the plant equations were found. A summary of all the plant equations can be found in appendix E.

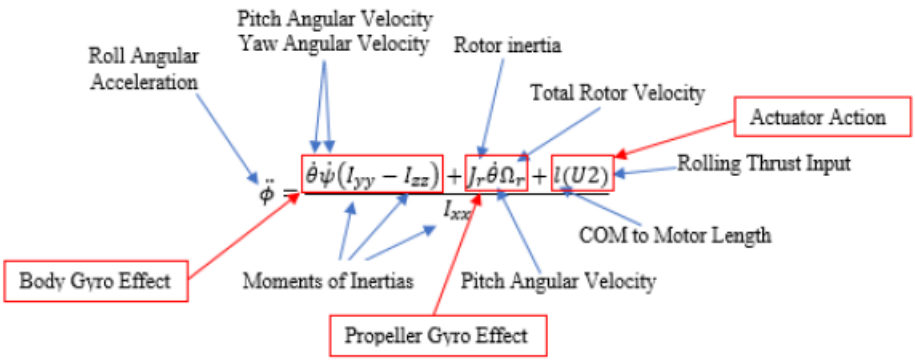


Figure 2.14-Rolling acceleration equation explained

Figure 24: Example of plant equation for phi. [?]

Body gyroscopic effects from changes in the quadcopter orientation. Propeller gyroscopic effects from propeller rotation and angle/orientation Actuation action forces produced by the rotors/propellers.

5.1.8 SIMULINK/Matlab Implementation

To make the Simulink model the thrust equations must be transferred into the program using the angular speed as inputs and the motor parameters. Below we see the thrust

equations in Simulink with the inputs on the left and outputs on the right. (The outputs are U1, U2, U3, and U4, as well as Omega) In SIMULINK the math is done using "blocks", the addition and subtraction is performed first, and then the next blocks take the product. Essentially the math is converted using blocks in SIMULINK.

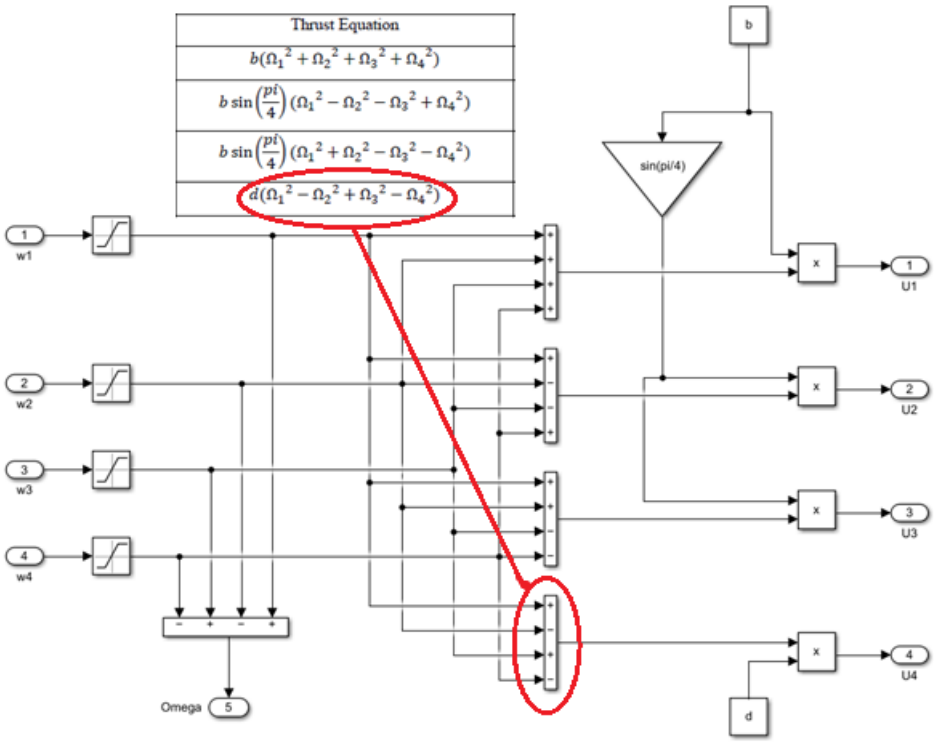


Figure 25: Thrust equations in SIMULINK

The rest of plant equations are then also transferred to SIMULINK using the same process, using blocks to represent the math. The next part of the plant takes the thrust equations as an input to calculate theta, phi, and psi, for position, velocity, and acceleration.

Below is an example for the equation of phi double dot. We use the thrust equations and feedback together with the drone parameters to get the result. Since all the equations need feedback from the other plant equations, an initial value must be set. It makes sense to make the initial value 0 since we haven't moved yet and there has been no change in any direction or angle. Throughout the flight the drone will then fill out the theta, phi, and psi values using the censors to determine the change in position. For example, if the drone is slanted a bit in the direction of motor 4 and 1, the drone will have a change in the roll aka phi value. The censor will detect the magnitude of this change, and U2 will increase to give more thrust to those motors to stabilize. Thrust is increased proportionally to smoothing the stabilization.

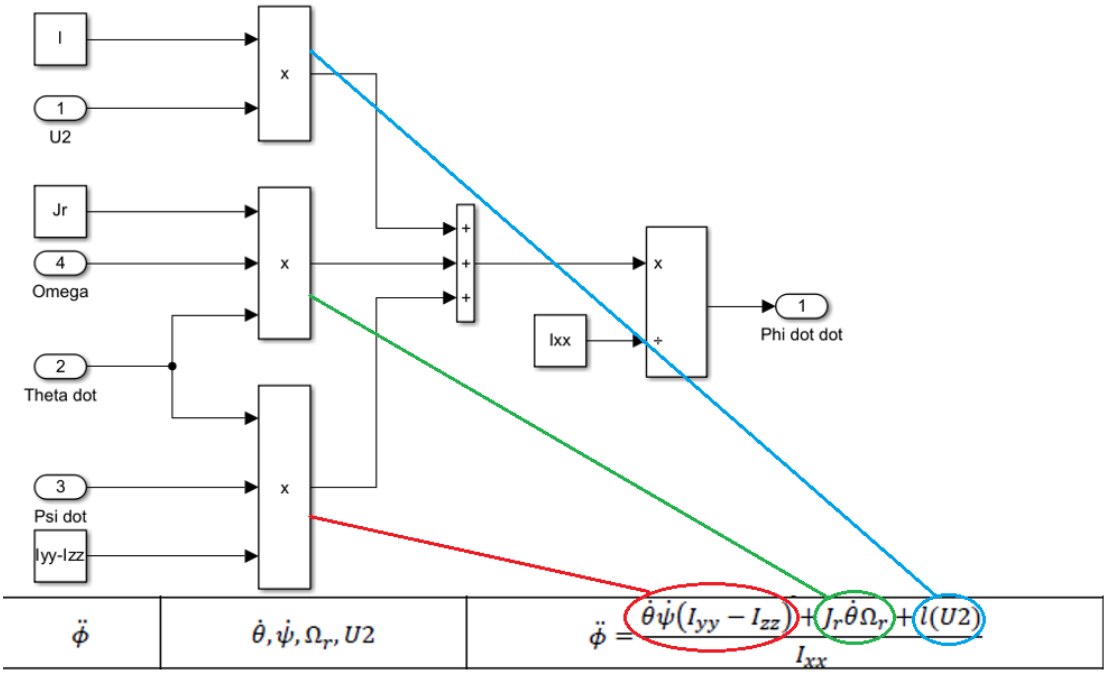


Figure 26: Plant equation for phi double dot

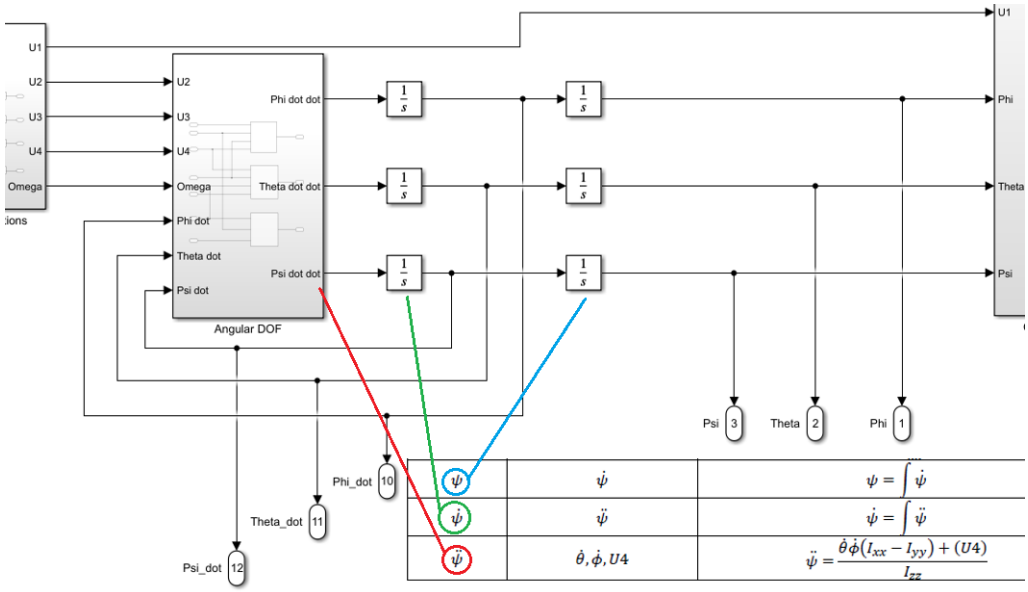


Figure 27: Angular equations overview example

To finish out the plant, the X, Y, and Z values for acceleration, velocity, and position are calculated using the values from theta, phi, and psi, these equations rely on feedback from each other in the same way that angles do, and use the same principles when it comes to determining the initial value.

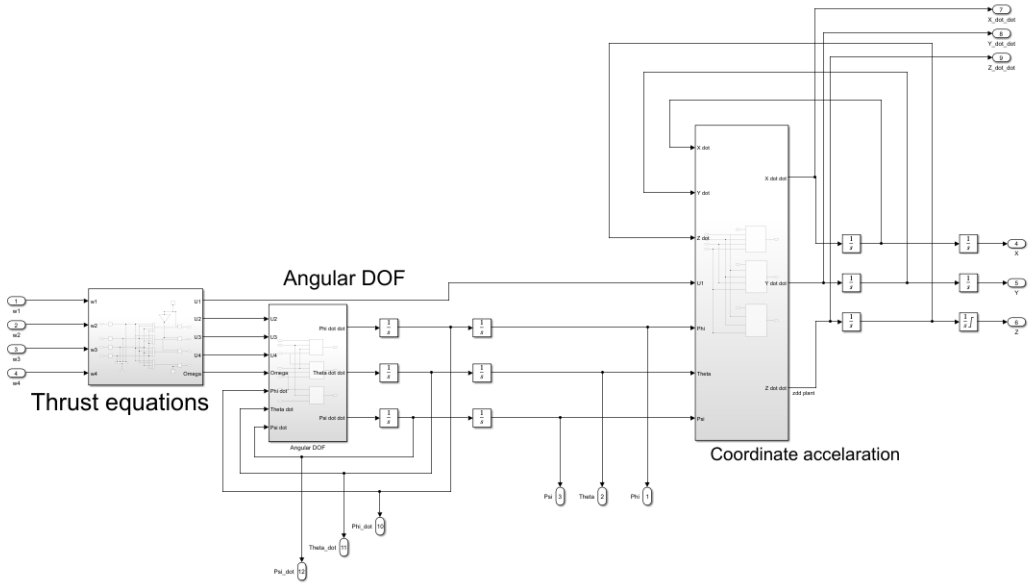


Figure 28: Plant overview in SIMULINK

5.2 Controller

In order to fly the drone in a controlled manner, a PID controller is implemented. The controller takes a desired state (for example hovering at 1.5 meters) and subtracts the output of the system, to create an "error signal". This error signal tells the drone how displaced it is based on the magnitude of the error. The job of the controller is to correct this error through PID to drive it as close to 0 as possible, achieving a stable state/flight.

A more in depth explanation of how the PID works will be presented later in the report.

In the figure above we see the controller model. The highlighted part is the change in Z. The PID outputs a new angular speed needed, which is divided by the change in theta and phi, since if the drone is tilted at an angle there will be a change in upwards thrust. This new value is then divided with the motor thrust coefficients to determine the magnitude of the change in Z compared to the thrust the motors provide. In short, the controller takes these changes Z depend on, and mathematically determines the change needed in angular speed for each motor, in order to go towards the desired height. [**?**]

This process is very much the same for the change in phi, theta, and yaw. The depending changes affect the outcome of the angular speed, which is put into the thrust equations to determine change for each specific motor, until the drone is stable at the set values.

It's important to note that a saturation block is put right before the output of the system to set an upper limit on the output to 3500, which is the maximum angular speed possible for the motors. This ensures that the model is realistic with the real life motors.

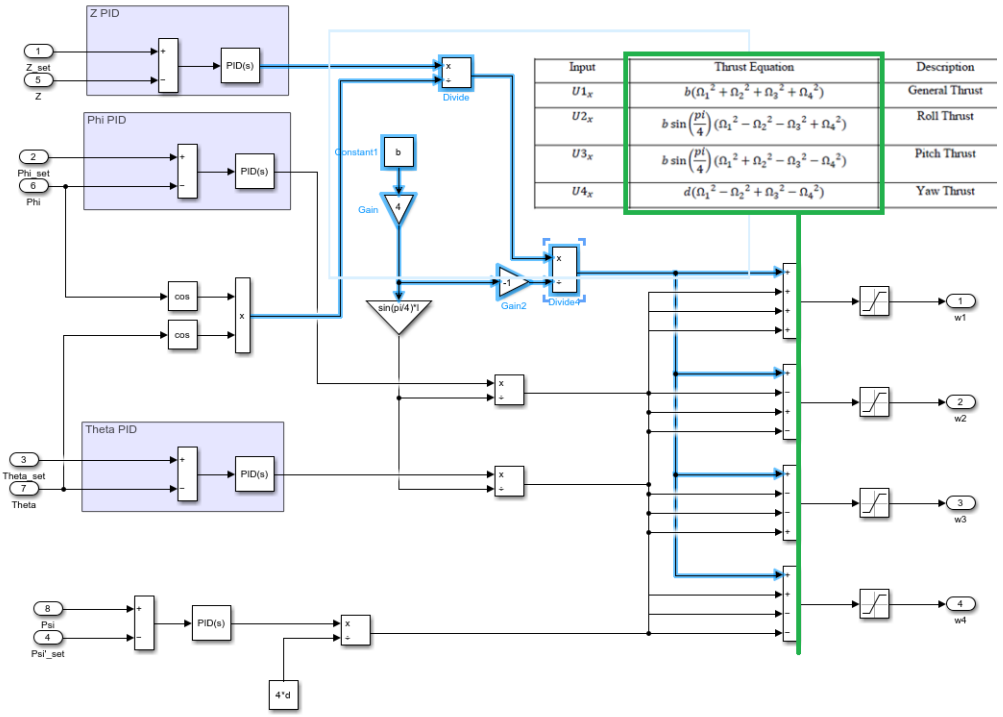


Figure 29: Overview of controller, with Z highlighted

5.3 UAV Toolbox

To achieve a visual way for checking if the quadrotor plant has been constructed correctly, the UAV Toolbox was used. (See Figure **??**) For studying PID control, the MATLAB UAV 3D simulation provides a realistic and interactive environment. The simulation was used to examine the behavior of the UAV system rather than relying exclusively on theoretical concepts and mathematical equations. Experiments with different PID controller parameters were conducted, to see how they affect the quadrotor's reaction and acquire hands-on experience tweaking the controller. This immersive learning experience aid in the integration of theory and practice. Control performance in complex circumstances can be analyzed using the MATLAB UAV 3D simulation. When numerous PIDs are involved, it can become difficult to intuitively understand how one effects another; thus, a visual representation aids in grasping and understanding the concepts.

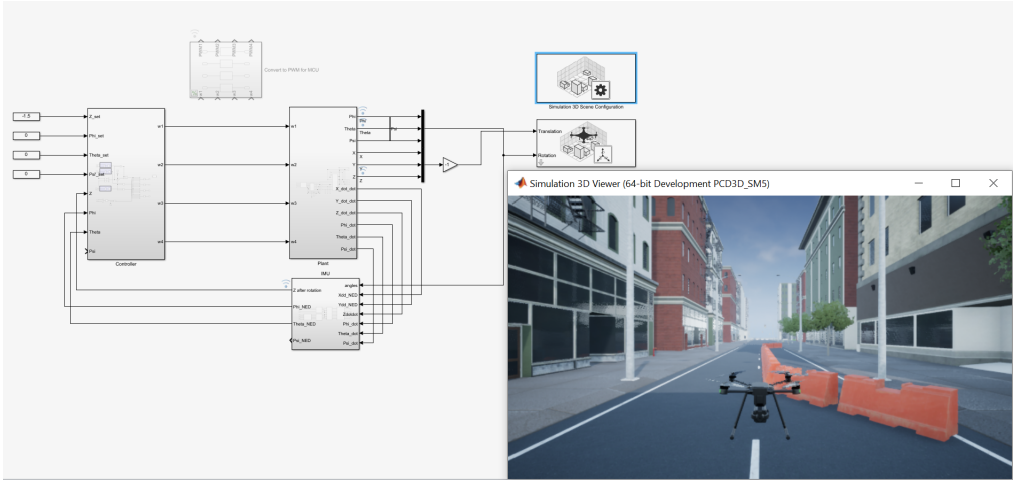


Figure 30: Ilustration of the workflow, with the utilization of the UAV Toolbox

5.4 The theory of PID

5.4.1 Principles of PID Control

PID (Proportional-Integral-Derivative) control is a popular feedback control algorithm that is utilized in a variety of control systems. PID control is made up of three parts: proportional, integral, and derivative. The proportional term controls the process based on the current difference between the desired setpoint and the measured process variable. To decrease steady-state mistakes and improve system stability, the integral term accumulates past errors over time. The derivative term computes the error's rate of change and aids in dampening quick changes, improving response time and lowering overshoot. [?]

5.4.2 PID Control Loop

The PID control loop is made up of four basic components: the process or plant under control, in this case roll and pitch, the process or plant under control, and the process or plant under control. A sensor or measurement device. The PID controller and the actuator. The IMU measures the process variable (angle) and compares it to the desired setpoint. The PID controller calculates the control signal based on the mistake, and the actuator alters the system accordingly. In this closed-loop feedback system, the control signal is continuously changed to keep the process variable close to the setpoint. [?]

5.4.3 Benefits of PID Control

PID is a versatile and frequently used control algorithm that provides excellent control in a variety of applications. Other solutions exist, such as model predictive control, but they are less widespread and thus have a smaller community. [?]

Stability PID control helps maintain system stability by continuously adjusting the control signal based on the error feedback. The proportional, integral, and derivative terms work together to provide a balance between stability and responsiveness. [**?**]

Robustness PID control is robust and effective in handling disturbances, noise, and external factors that may affect the system's performance. The integral term, in particular, helps eliminate steady-state errors caused by external disturbances. [**?**]

Adaptability PID control can be tuned and adjusted to optimize performance based on specific system requirements. The controller gains can be modified to enhance stability, reduce overshoot, or improve response time. [**?**]

Simplicity PID control is relatively easy to implement and understand. It offers a straightforward approach to control systems without requiring complex mathematical models or extensive computational resources. [**?**]

5.4.4 Ziegler-Nichols method

The performance of a PID controller depends on appropriate tuning to match the characteristics of the controlled system. Tuning involves adjusting the proportional, integral, and derivative gains to achieve the desired response. To tune the quadrotor's PID Ziegler-Nichols method was used, as it is compromise between complexity and time. [**?**]

The value of the proportional-only gain that allows the control loop to oscillate forever at steady state is used to calculate the ultimate gain, K_u . This means that the gains from the I and D controllers are set to zero in order to determine the influence of P. It evaluates the robustness of the K_c value in order to optimize it for the controller. The final period is another key number related with this proportional-only control tuning approach (P_u). The ultimate period is the amount of time it takes to complete one entire oscillation while the system is in steady state. These two parameters, K_u and P_u, are employed to determine the controller's loop-tuning constants (P, PI, or PID). [**?**]

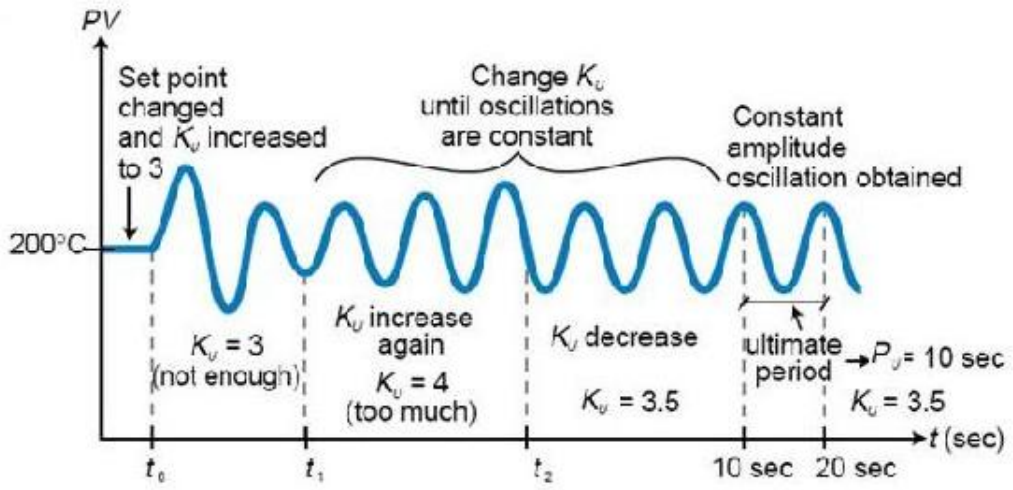


Figure 31: Example of a system tuned using the Ziegler-Nichols method [?]

Ziegler-Nichols method steps are as follows:

1. Take out the integral and derivative actions. Set the integral time (T_i) to 999 or the highest number possible, and the derivative controller (T_d) to zero.
2. Change the set point to cause a slight perturbation in the loop. Increase or decrease the gain of the proportional until the oscillations have a constant amplitude.
3. Mark down the gain value (K_u) and period of oscillation (P_u).
4. Input these values into the Ziegler-Nichols equations to find the controller settings. (See Table **??**)

Control Type	K_p	T_i	T_d	K_i	K_d
P	$0.5K_u$	–	–	–	–
PI	$0.45K_u$	$0.83T_u$	–	$0.54K_u/T_u$	–
PD	$0.8K_u$	–	$0.125T_u$	–	$0.1K_uT_u$
classic PID	$0.6K_u$	$0.5T_u$	$0.125T_u$	$1.2K_u/T_u$	$0.075K_uT_u$
some overshoot	$0.33K_u$	$0.5T_u$	$0.33T_u$	$0.66K_u/T_u$	$0.11K_uT_u$
no overshoot	$0.2K_u$	$0.5T_u$	$0.33T_u$	$0.4K_u/T_u$	$0.066K_uT_u$

Table 1: Formulas for determening the PID constants via Ziegler-Nichols

5.5 Linearization of the plant

Multiple sources indicate, that linearization needs to be considered. “PID is a linear-type controller and hence is only efficient for a limited operating range when used to control non-linear processes.” [?] “PID controllers perform well only on linear systems or systems that are linearized.” [?] “PID control can only be applied for linear plant dynamics 99 percent of applications” [?]

As the quadrotor, is a non-linear system, due to its non-linear terms in the plant equations, to use PID, linearization would be needed. It is planned to have three PID controllers: Z, ϕ/roll, and θ/pitch. The equations below indicate that the only non-linear equation is Z.

$$\ddot{\phi} = \frac{\dot{\theta}\dot{\psi}(I_{yy} - I_{zz}) - J_r\dot{\theta}\omega_r + l(U_2)}{I_{xx}}$$

$$\ddot{\theta} = \frac{\dot{\phi}\dot{\psi}(I_{zz} - I_{xx}) - J_r\dot{\phi}\omega_r + l(U_3)}{I_{yy}}$$

$$\ddot{Z} = \frac{mg - (cos\theta cos\phi)U_1 - A_z\dot{Z}}{m}$$

The non-linear terms are due to ϕ and θ. There is a simple way of linearizing the Z-axis plant equation, as the intended goal is only hovering. For hovering, ϕ, and θ would be zero, which would be ensured due to PID control to respective axis, they would become constants, thus the non-linear terms

$$cos(\phi) * cos(\theta) \longrightarrow cos(0) * cos(0) = 1$$

$$\ddot{Z} = \frac{mg - (cos\theta cos\phi)U_1 - A_z\dot{Z}}{m} \longrightarrow \ddot{Z} = \frac{mg - U_1 - A_z\dot{Z}}{m}$$

Thus the plant Z-axis equation is linearized. Only downside, is to accept this assumtion, it should be ensured, that ϕ and θ PID controllers should be calibrated first, to ensure, that the values remain zero, when calibrating Z-axis PID.

5.6 The simulation of PID

5.6.1 ϕ and θ hyper-tuning

As mentioned before, the calibration in ϕ and θ axis should be done first. As they are linear terms, the Simulink PID tune app can be used. Before using the tool, one must understand the usable sliders. By adjusting the Response Time slider, how aggressively the controller should respond to changes in the setpoint can be specified. If the Response Time is increased, the controller will be more aggressive in responding to any difference between the current output and the desired setpoint, it may also cause more overshoots or oscillations. Setting the Response Time to a lower value will cause the controller to respond more slowly to setpoint changes, resulting in smoother but potentially slower response times. Transient Behavior Slider: The Transient Behavior slider in the PID tune app regulates the contribution of the derivative term. The derivative term aids in dampening oscillations and lowering overshoot, as discussed in the "Theory of PID" chapter. The PID tuner app was used to hyper-tune and obtain results that could then be used in the physical setup and fine-tuned, thus the two sliders were set in the middle

for later adjustment. The results were optimal. (See figure **??** and **??**)

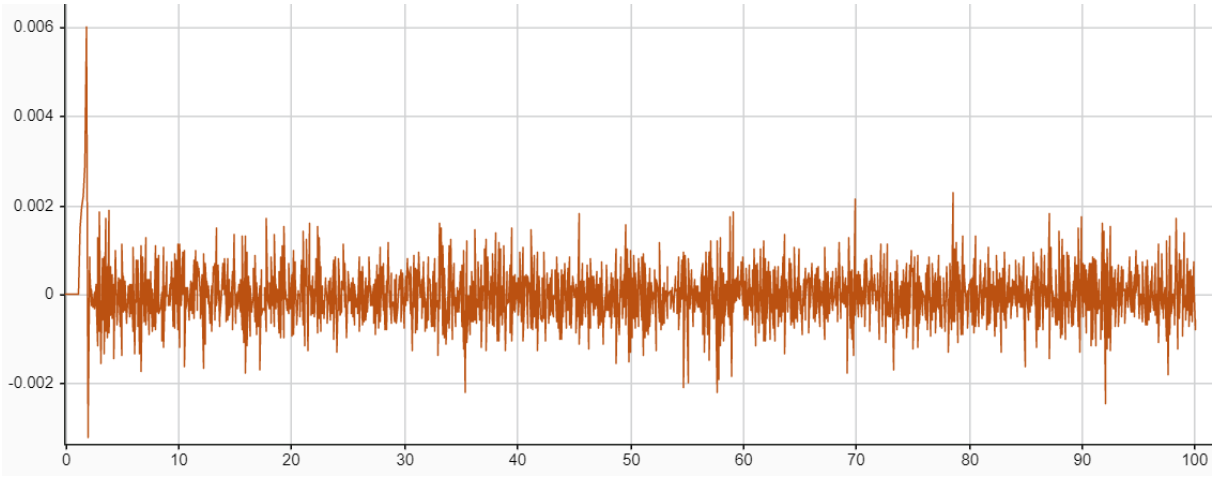


Figure 32: Response of ϕ, with hyper-tuned PID

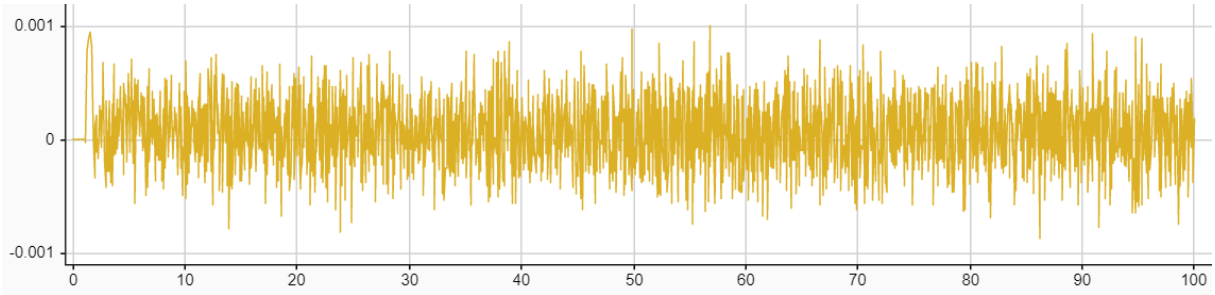


Figure 33: Response of θ, with hyper-tuned PID

5.6.2 Z hyper-tuning

After ϕ and θ have been hyper-tuned, it can be safely assumed, that they are 0, thus the previously discussed linearization can be utilized. The Ziegler-Nichols approach, as discussed in the previous chapter, was employed for calibration in the Z PID controller simulation. To begin, the Simulink PID blocks were adjusted to only have the gain value and then adjusted until an oscillating response was obtained, around a specified setpoint. (See figure **??**) Simulink data inspector was used to viewing the output because it also included useful features, such as measuring with two cursors, which came in handy while measuring the period of the oscillation period.

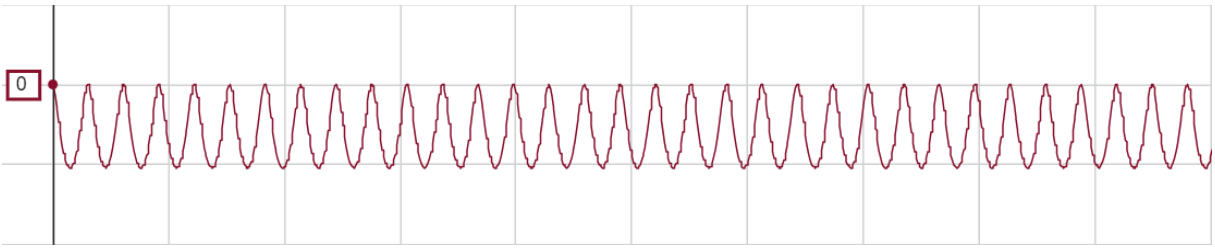


Figure 34: An oscillating response with only gain in Simulink

The measured K_u and T_u, were plugged into the Ziegler-Nichols formula table (See table **??**) and then fine-tuned to achieve less of an overshoot and faster response time. The final values for an adjusted PID system where as follows, where red indicates the calculated values, and green - the fine-tuning. (See figure **??**)

Control Type	K_p	T_i	T_d	K_i	K_d
P	$0.5K_u$	–	–	–	–
PI	$0.45K_u$	$0.83T_u$	–	$0.54K_u/T_u$	–
PD	$0.8K_u$	–	$0.125T_u$	–	$0.1K_uT_u$
classic PID	$0.6K_u$	$0.5T_u$	$0.125T_u$	$1.2K_u/T_u$	$0.075K_uT_u$
some overshoot	$0.33K_u$	$0.5T_u$	$0.33T_u$	$0.66K_u/T_u$	$0.11K_uT_u$
no overshoot	$0.2K_u$	$0.5T_u$	$0.33T_u$	$0.4K_u/T_u$	$0.066K_uT_u$

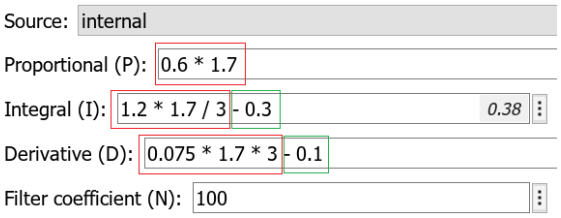


Figure 35: Tuned PID controller block values

After tuning the PID, the output response was ideal. (See figure **??**)

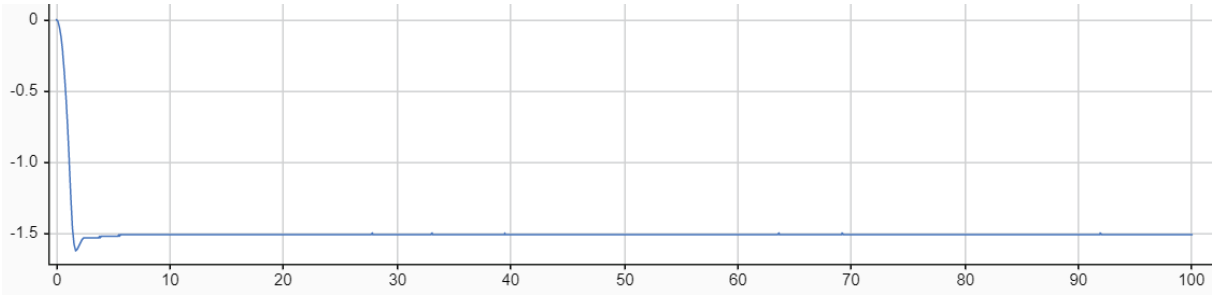


Figure 36

Unfortunately, this is the plant's response when there is no noise, thus when noise is introduced, the response drifts. (See figure **??**) This was caused by the integral term summarizing all noise-induced errors. This was a clear indication that more than one sensor was required for accurate Z-axis calibration. Thus, instead of using the IMU Z acceleration as input, an IR sensor was added to the real system and used as input.

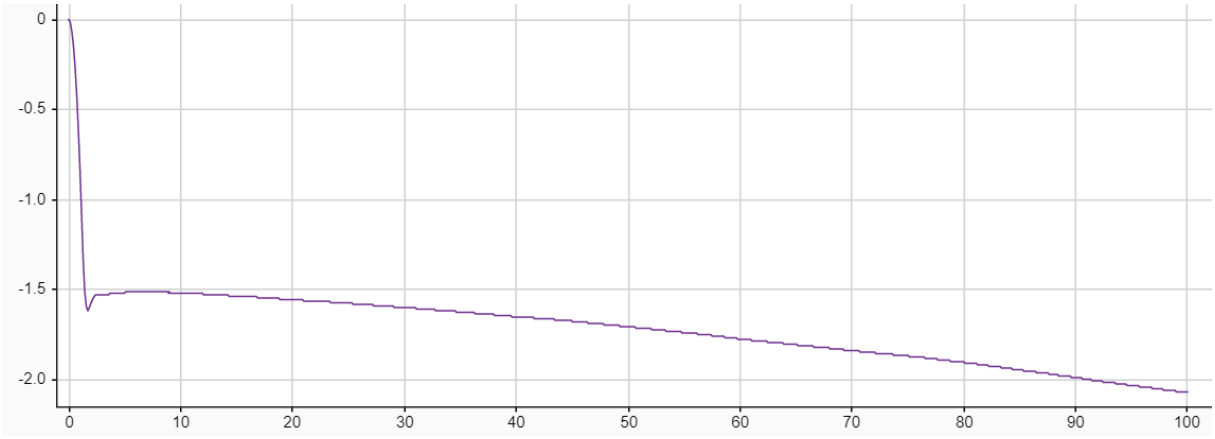


Figure 37: Z-axis response, tuned using the Ziegler-Nichols method

5.7 The Implementation of PID

5.7.1 One-axis setup

Making PID for all axis is possible, but not ideal as determining the source of the problem is more difficult. Therefore, it was decided to tune one axis at a time, excluding the Z-axis which would be tuned after fully calibrating the rest, as it is non-linear. The initial 'P' calculation was made from the simulation formula for maximum thrust. The value was calculated to be 0.0045, but as seen in figure **??**, the system is stable, but for the Z-N method, systems need to be marginally stable as shown in figure **??**. In figure **??**, the PWM effect of PID on motors is also shown to make sure that QuickPID works as predicted and without delay. To test the stability of system, it was displaced by a finger in order to mimic large turbulence.

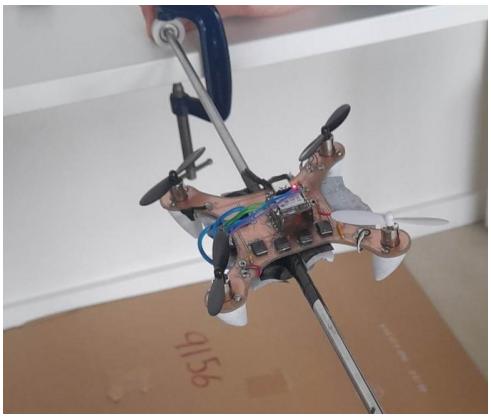


Figure 38: Picture of axis rig

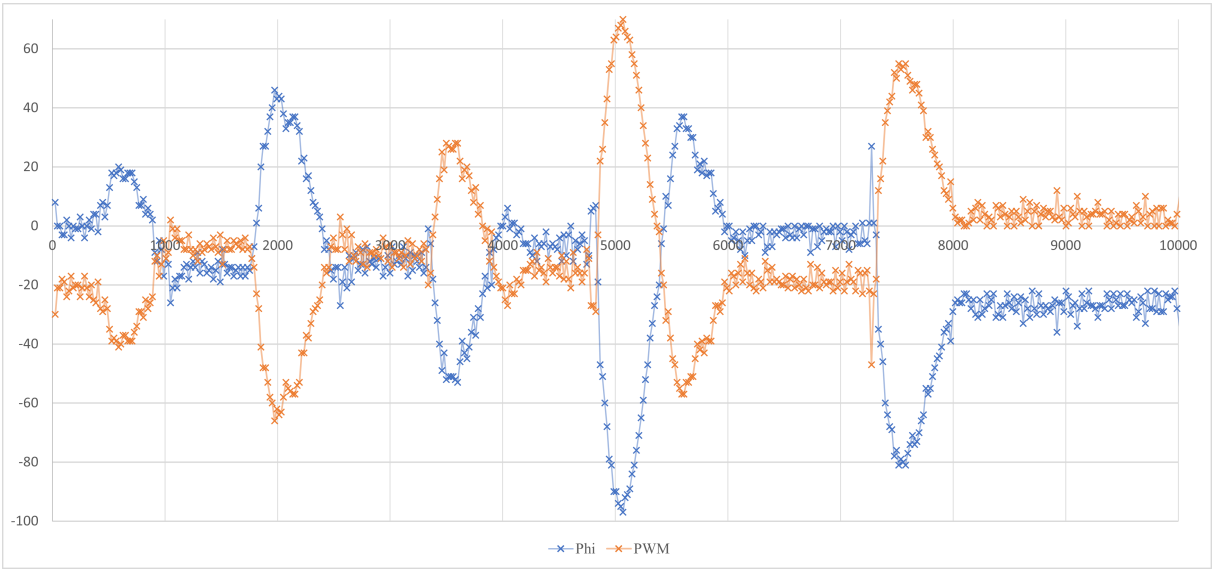


Figure 39: Plot of marginally stable system on Y-axis with purely proportional control

After determining the marginally stable value of P to be 0.0065, the Z-N method was used and applied to the system resulting in figure **??**. The system is stable, but has a large settling time and long period. This was improved by enlarging the 'P' constant by 0.002 as shown in figure **??**.

For tuning the θ, the same testing rig was utilised and the values of the calibrated ϕ were used as starting point. Those were calibrated and the result is shown in figure **??**.

From the testing, it was evident that overshoot was common, but it was chosen as a trade-

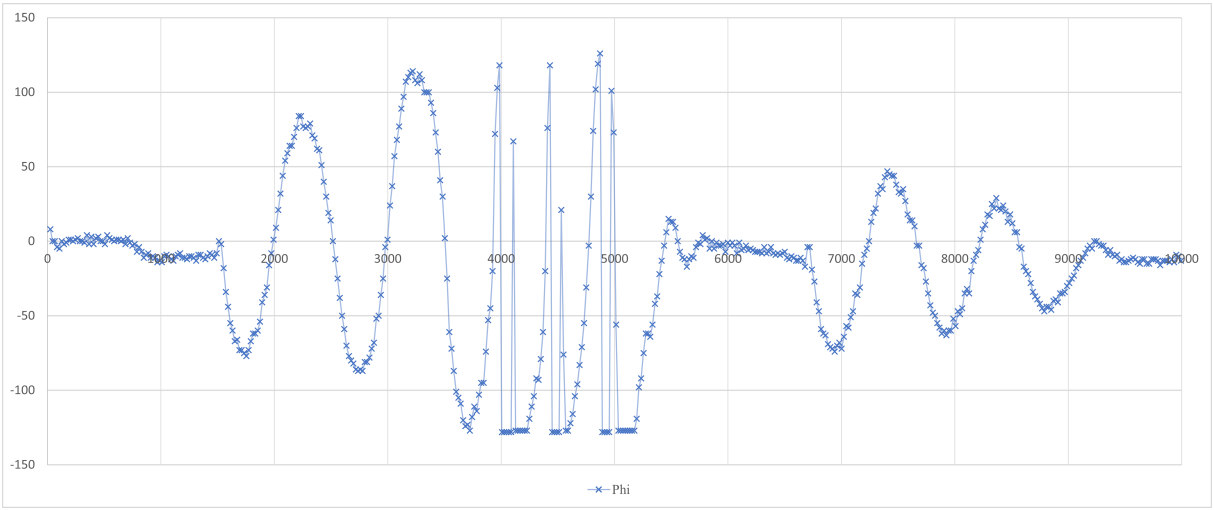


Figure 40: Plot of stable system on Y-axis with purely proportional control

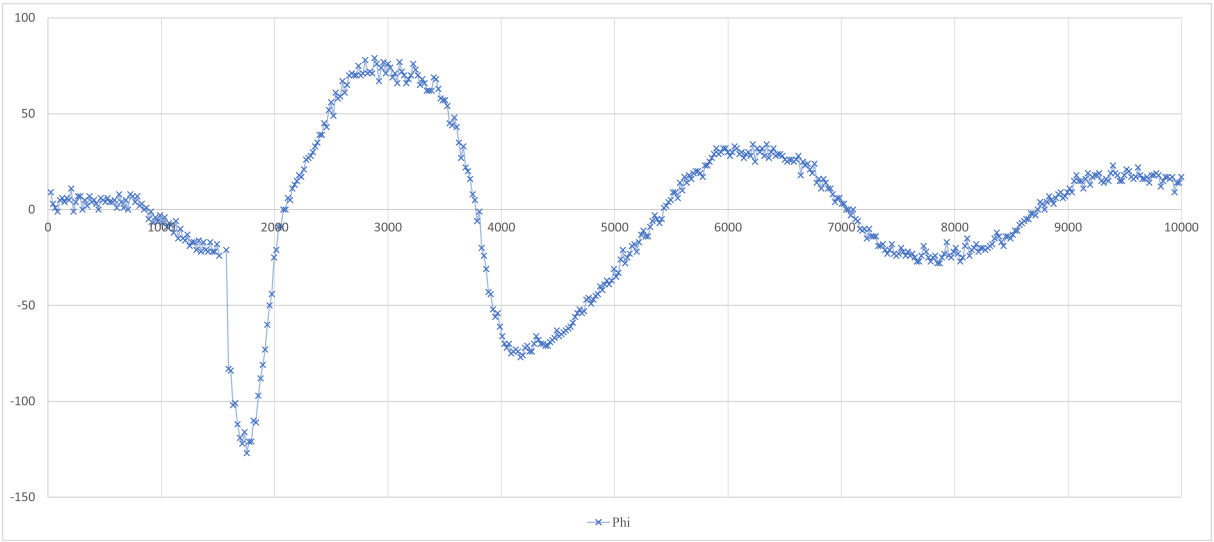


Figure 41: Plot of the drone on Y-axis with values from Z-N method

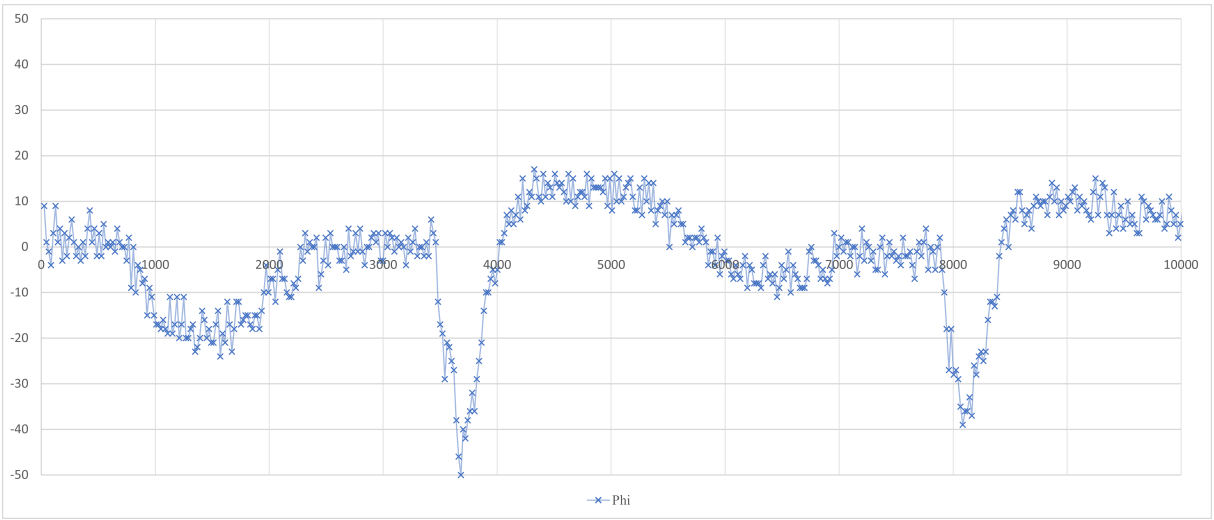


Figure 42: Plot of the drone on Y-axis with calibrated values from Z-N method

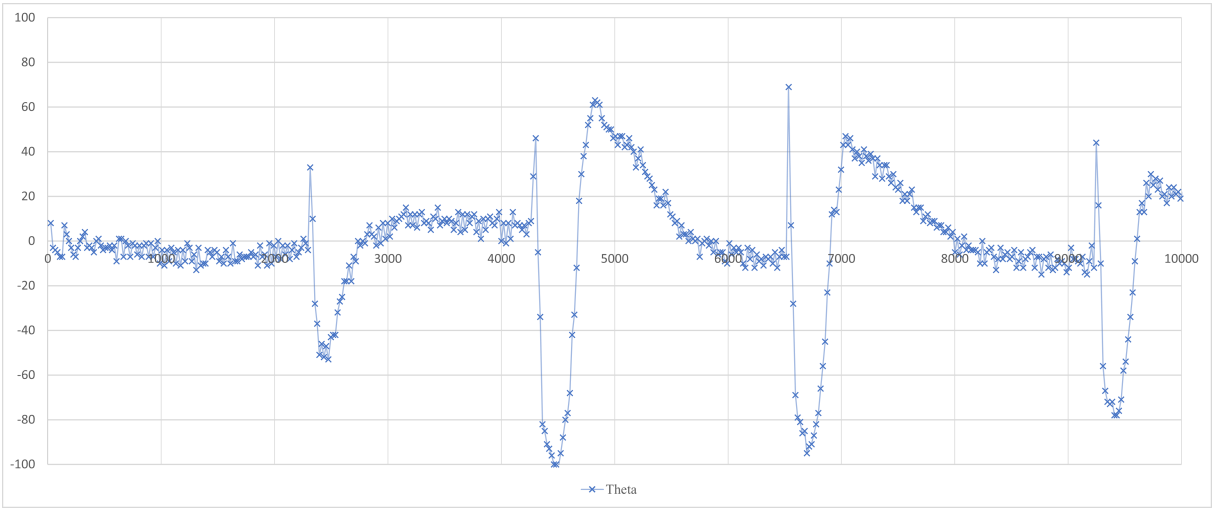


Figure 43: Plot of the drone on X-axis with calibrated values from Z-N method

off for the aggressive transient behaviour. Additionally, the interference from the finger results in a larger deviation in angle than it would in actual free flight circumstances.

5.7.2 Diagonal setup

Testing in the string rig after tuning values in one-axis resulted in uncontrolled oscillations of the drone with no regards to the PID values. Due to that, the diagonal rig was set up and the code was changed accordingly. The same procedure of axial testing was repeated for the diagonal rig and is described with figures **??** and **??**.

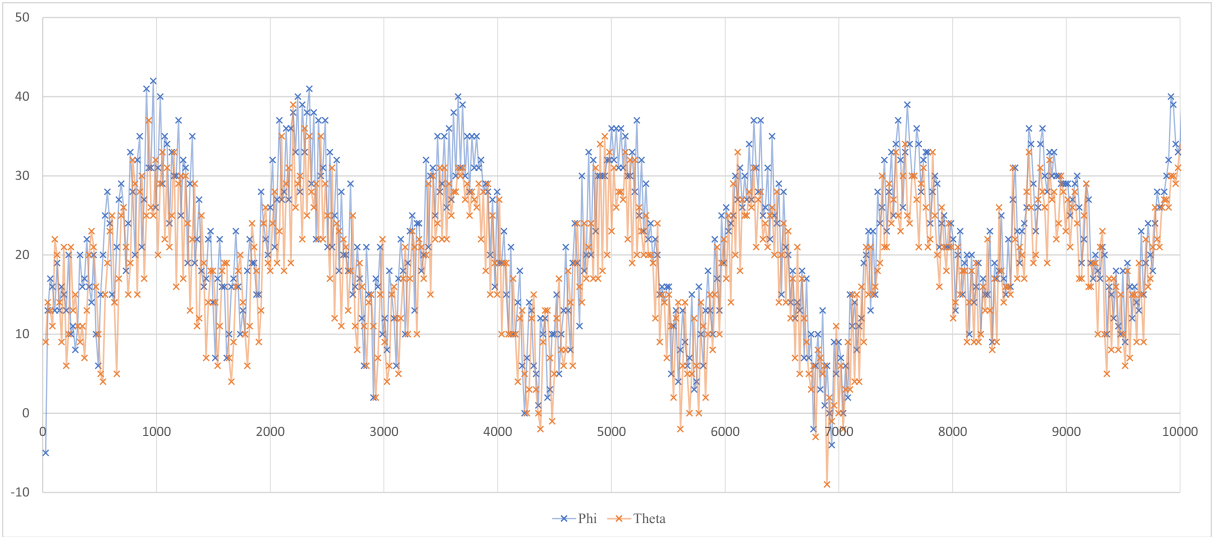


Figure 44: Plot of purely proportional control in diagonal setup for Z-N method

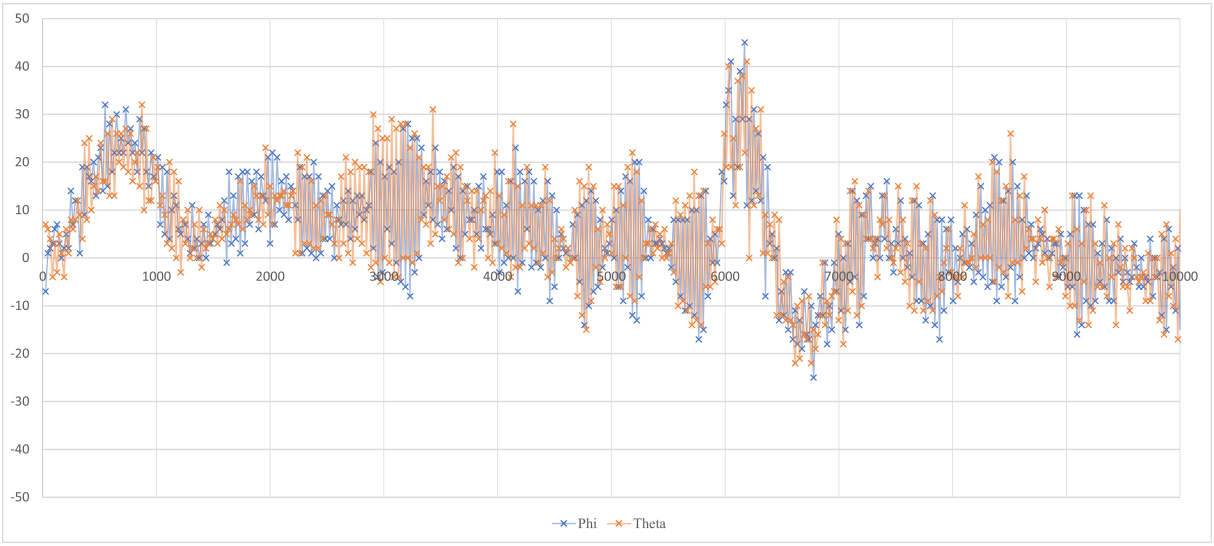


Figure 45: Plot of angles after using Z-N method from 'P' in figure **??**

5.7.3 String setup

The final testing rig was chosen to be the string rig (figure **??**). A halo was attached to the top of the drone, so that the string attached to the drone would be put through, thus resulting in the strings not colliding with the propellers in different angles.

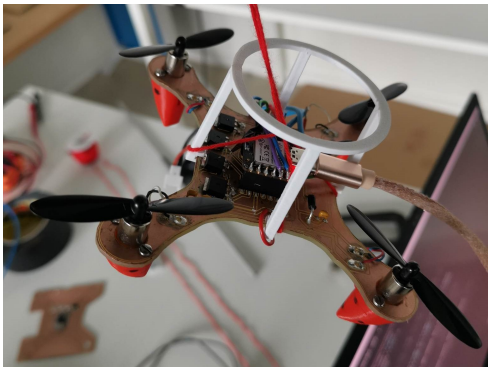


Figure 46: String rig

This provides the closest possible resemblance to flight that could achieved. It may adversely effect the flight characteristics, as the string pulls the drone to the center, which may result in tilt while flying away from the anchor point, but the other option was a gimbal, which introduced large friction.

The values from previous test runs were used at the start, but were found to be insufficient to compensate. Post testing, the ideal values were found to be 10 times the P and 5 times the D compared to the ones used in the diagonal rig. With these values ϕ was stable, however θ was oscillating between stability and instability, as it was initially stable, then became unstable, then became stable again. This was accounted for with higher PID constants for theta.

5.7.4 Conclusion of PID testing

While independent flight was not achieved due to insufficient thrust, the PID system worked in a limited time frame. The instability in θ during string testing was nearly unaffected even with the 5x multiplier. This points to the possibility of the insufficient thrust affecting the authority of the PID control. The other sources of instability could

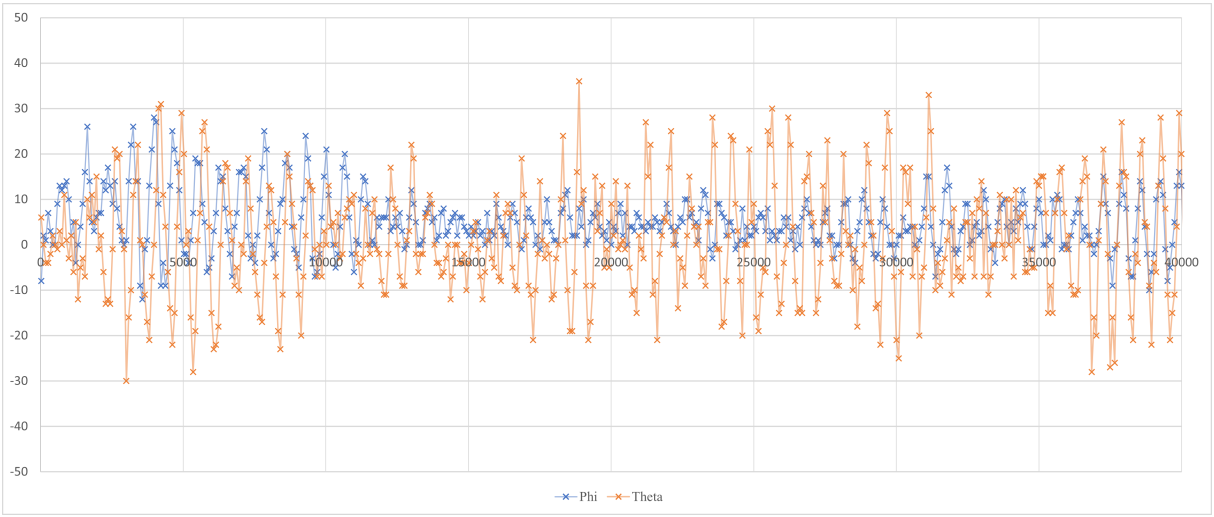


Figure 47: Plot of angles in string setup with calibrated PID values

potentially be the differences between motors in different temperatures and RPMs or IMU deviation and drift.

graphicx float

5.8 IMU theory

Inertial Measurement Unit is a key component of the drone control as it provides feedback about the position in time which is crucial for adjusting the control input signal.

As described in the morphology for our project we have chosen to use the inbuilt IMU sensor with 6DOF available on the Seeed Studio XIAO nRF52840 (Sense) micro-controller board. Since noise on incoming is a concern, it is beneficial to use an inbuilt component in an effort to reduce the interference on IMU readings.

The IMU consists of a gyroscope that provides angular velocity data and an accelerometer providing axial acceleration data.

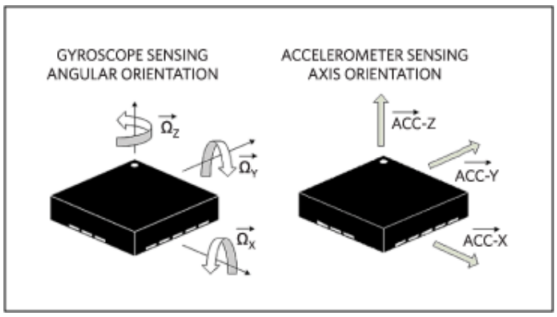


Figure 48: Gyroscope and Accelerometer internal axis

Each gyroscope channel measures the rate of rotation around one of the internal axes in degrees per second (as our gyroscope has an inbuilt ADC unit, the actual output will be

in bits that can be re-interpreted as degrees per second). With these readings, the angular position can be calculated by integrating the gyroscope measurements over time. If the angular velocity is denoted by $\dot{\Theta}$, the angle of rotation can be calculated as follows:

$$\dot{\theta} = \frac{d\theta}{dt}$$

$$\theta(t) = \int_0^t \dot{\theta}(t)dt$$

Figure 49: Integration of gyroscope readings

To better understand the accelerometer readings we can imagine a box with a ball inside of it. In zero gravity conditions the ball would be floating in the air and no force on any of the walls would be detected.

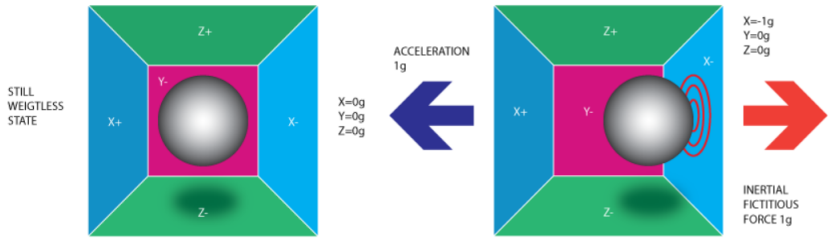


Figure 50: Accelerometer, zero gravity and motion on X axis

If the box was moved in positive X axis direction with acceleration of $9.81m/s^2$, the ball would hit the wall of the box in negative X axis direction and register force of $1g$. From this we can note that accelerometer is reading a force in the opposite direction of the motion.

Similarly we can now imagine that when the box is sitting still in gravitational field of the Earth, we would get a reading of $1g$ in the downwards Z axis direction. If the IMU was rotated in respect to Earth surface, the reading of 1g would be split into x, y and z components on the accelerometer internal axis. We can use these readings to calculate the positional angles with respect to earth surface with the help of the following formula.

$$\phi_A = \tan^{-1}\left(\frac{\ddot{Y}}{\ddot{Z}}\right)$$

$$\theta_A = \tan^{-1}\left(\frac{\ddot{X}}{\sqrt{\ddot{Y}^2 + \ddot{Z}^2}}\right)$$

Figure 51: Positional angle calculation

It is important to note that all measurements on accelerometer are read in the IMU body frame. In order to relate them to Earth's surface we need to choose a reference coordinate system and relate the measurements from the body reference frame to the chosen coordinate system.

For our model we have chosen to work with NED (north-east-down) coordinate system, where X axis is pointing to north, the y-axis to east, and the z-axis is pointing down.

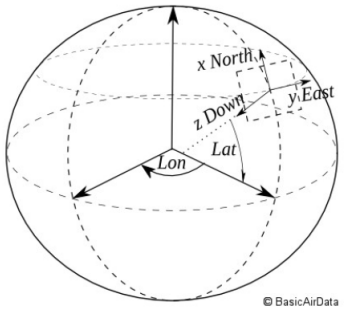


Figure 52: NED coordinate system

In order to relate sensor readings from body reference frame to NED, we are using Euler angles, which provide a way to represent the 3D orientation of an object using a combination of three rotations about different axis. Rotation about X axis is denoted by Φ, rotation about Y with Θ and rotation around Z axis with Ψ.

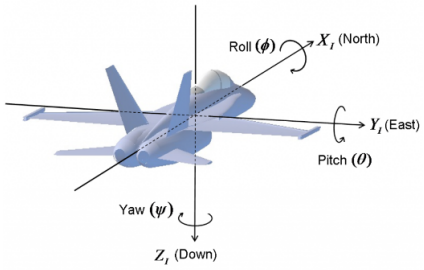


Figure 53: Euler angles

Any object rotation can be described by multiplying a rotational matrix with the original position vector. Any rotation matrix R can be decomposed as a product of three elemental rotation matrices - Roll, Yaw and Pitch.

$$Roll = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\Phi) & -\sin(\Phi) \\ 0 & \sin(\Phi) & \cos(\Phi) \end{pmatrix} Pitch = \begin{pmatrix} \cos(\Theta) & 0 & \sin(\Theta) \\ 0 & 1 & 0 \\ -\sin(\Theta) & 0 & \cos(\Theta) \end{pmatrix} Yaw = \begin{pmatrix} \cos(\Psi) & -\sin(\Psi) & 0 \\ \sin(\Psi) & \cos(\Psi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 54: Rotational matrixes

The complete rotation matrix for moving from the NED frame to the body frame is achieved by first performing Yaw then Pitch then Roll resulting in the following rotation matrix ($R((\Phi\ ,\ \Theta\ ,\ \Psi))$:

$$\begin{pmatrix} \cos(\Psi)\cos(\Theta) & -\cos(\Theta)\sin(\Psi) & \sin(\Theta) \\ \cos(\Phi)\sin(\Psi)+\cos(\Psi)\sin(\Phi)\sin(\Theta) & \cos(\Phi)\cos(\Psi)-\sin(\Phi)\sin(\Psi)\sin(\Theta) & -\cos(\Theta)\sin(\Phi) \\ \sin(\Phi)\sin(\Psi)-\cos(\Phi)\cos(\Psi)\sin(\Theta) & \cos(\Psi)\sin(\Phi)+\cos(\Phi)\sin(\Psi)\sin(\Theta) & \cos(\Phi)\cos(\Theta) \end{pmatrix}$$

To transition from body reference frame to NED, we need to perform the rotations in opposite order - Roll, then Pitch, then Yaw from negative angles ($R(-\Psi\ ,-\Theta\ ,-\Phi)$) :

$$\begin{pmatrix} \cos(\Psi)\cos(\Theta) & \cos(\Phi)\sin(\Psi)+\cos(\Psi)\sin(\Phi)\sin(\Theta) & \sin(\Phi)\sin(\Psi)-\cos(\Phi)\cos(\Psi)\sin(\Theta) \\ -\cos(\Theta)\sin(\Psi) & \cos(\Phi)\cos(\Psi)-\sin(\Phi)\sin(\Psi)\sin(\Theta) & \cos(\Psi)\sin(\Phi)+\cos(\Phi)\sin(\Psi)\sin(\Theta) \\ \sin(\Theta) & -\cos(\Theta)\sin(\Phi) & \cos(\Phi)\cos(\Theta) \end{pmatrix}$$

5.9 Sensor fusion

Now that we have established two methods of obtaining angular position, we must take into account the deficiencies related to each of the methods. The accelerometer gives good accuracy when the drone is not accelerating and the only acceleration we are reading is from gravity. The gyroscope can read quick movements but since the readings are being integrated, any error in the measurements is repeatedly added to the calculated position thereby leading to a drifting signal.

The solution is to introduce either a filter or an observer to fuse the measurements from accelerometer and gyroscope. For our project we have implemented a complimentary filter consisting of a low pass filter on the accelerometer and a high pass filter on gyroscope and summing the filtered signals.
The transfer functions of low pass and high pass filters in s domain are given below (where τ is the time constant and ω_o is the angular cut-off frequency in rad/s):

$$H(s) = \frac{V_{out}(s)}{V_{in}(s)} = \frac{\omega_0}{(s+\omega_0)} = \frac{1}{\tau s+1}$$
$$H(s) = \frac{V_{out}(s)}{V_{in}(s)} = \frac{s}{(s+\omega_0)} = \frac{\tau s}{\tau s+1}$$

Cut-off frequency in Hz:

$$f_c = \frac{\omega_o}{2\pi}$$

Since the filter will be digital, the real time transfer function needs to be discretized by moving to z domain. This can be done with the help of Bilinear transform (Tustin's

method), with the following substitution (where T_s is the sampling time):

$$s = \frac{T_s}{2}\frac{1-z^{-1}}{1+z^{-1}}$$

This results in discrete transfer function:

$$H(z) = \frac{\omega_o}{\frac{T_s}{2}\frac{1-z^{-1}}{1+z^{-1}}+\omega_o} = \frac{b_o+b_1 z^{-1}}{1-a_1 z^{-1}}$$

Giving us discrete time implementation of:

$$y(t) = a_o y_{n-1} + b_o x_n + b_1 x_{n-1}$$

To calculate the coefficients a_o, b_o, b_1 we can use Matlab Transfer function $tf()$ and Continuous to discrete time function $c2d()$

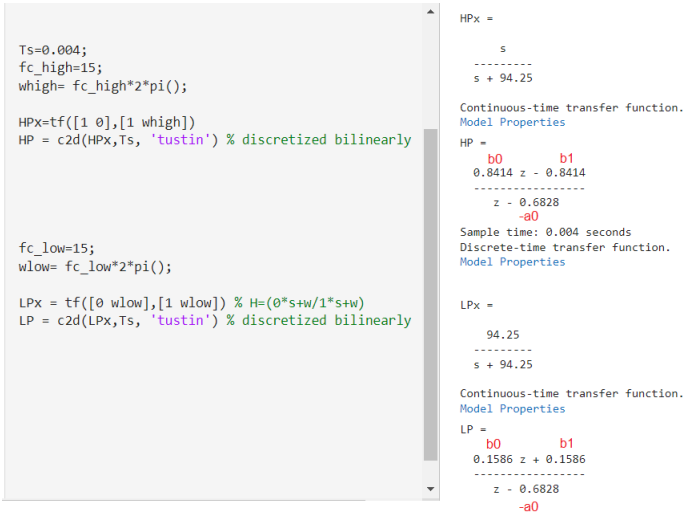


Figure 55: Bilinear transform in Matlab

The sampling time T_s used in calculations is the loop time of the code we run for the drone control.

After the calculated angles are passed through their respective filters, the values can be summed up resulting in a complimentary filter (as shown in the visual below):

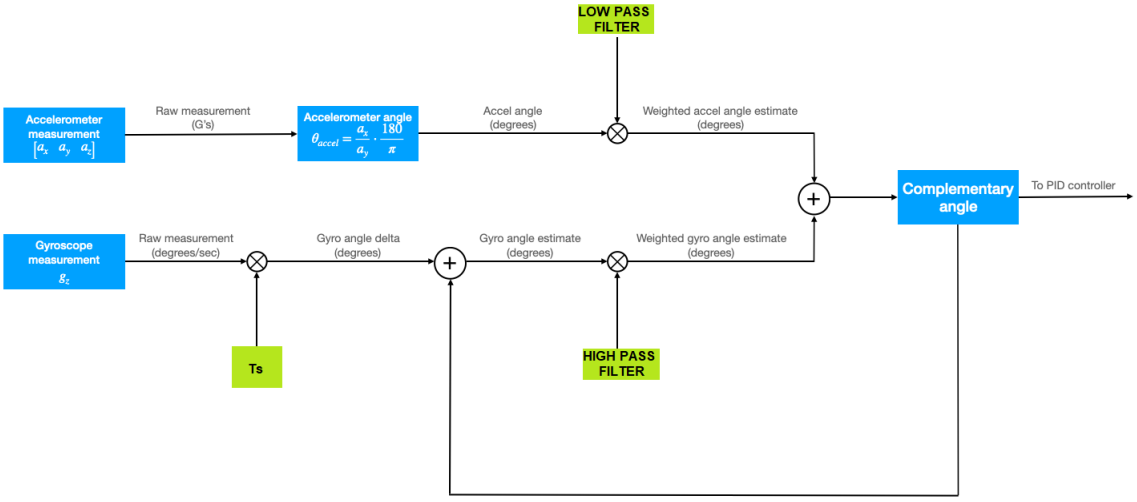


Figure 56: Complimentary filter implementation in discrete time

5.10 Simulink

To simulate the IMU readings as feedback in Simulink we first need to consider what input variables will be handled by the gyroscope and the accelerometer plants. The drone plant in the simulation is operating in NED reference frame, therefore any outputs coming from it are also in NED. However the physical IMU sensor will be handling measurements in body reference frame, therefore we need to create a function block to transform the drone output values into body reference frame values.

For the accelerometer this can be done by taking all linear motion acceleration outputs from drone plant in the form of a column vector $Acc = (\text{X"}, \text{Y"}, \text{Z"})$ and multiplying it with the rotational matrix for moving from NED to body. This equation requires knowing the Euler angles to be used in the rotational matrices. Luckily these values can also be taken from the drone plant.

```
Acc=[Xdd_NED;Ydd_NED;Zdd_NED]
Body_Acc=(Roll)*(Pitch)*(Yaw)*Acc
```

$$Acc = \begin{pmatrix} \mathrm{Xdd_{NED}} \\ \mathrm{Ydd_{NED}} \\ \mathrm{Zdd_{NED}} \end{pmatrix}$$

$$Body\_Acc = \begin{pmatrix} \mathrm{Zdd_{NED}}\sin(\Theta) + \mathrm{Xdd_{NED}}\cos(\Psi)\cos(\Theta) - \mathrm{Ydd_{NED}}\cos(\Theta)\sin(\Psi) \\ \mathrm{Xdd_{NED}}\,(\cos(\Phi)\sin(\Psi) + \cos(\Psi)\sin(\Phi)\sin(\Theta)) + \mathrm{Ydd_{NED}}\,(\cos(\Phi)\cos(\Psi) - \sin(\Phi)\sin(\Psi)\sin(\Theta)) - \mathrm{Zdd_{NED}}\cos(\Theta)\sin(\Phi) \\ \mathrm{Xdd_{NED}}\,(\sin(\Phi)\sin(\Psi) - \cos(\Phi)\cos(\Psi)\sin(\Theta)) + \mathrm{Ydd_{NED}}\,(\cos(\Psi)\sin(\Phi) + \cos(\Phi)\sin(\Psi)\sin(\Theta)) + \mathrm{Zdd_{NED}}\cos(\Phi)\cos(\Theta) \end{pmatrix}$$

Figure 57: Rotational transformation from NED to body ref. frame, Matlab

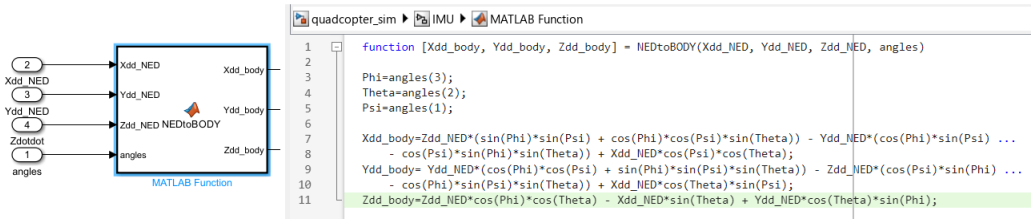


Figure 58: NED to body transformation function block, Simulink

For steady-level flight we can assume that angular velocities are the same in body and NED reference frames so we do not need to perform rotational transformation on these values. Therefore we can use the angular velocity output$(\dot{\Phi}, \dot{\Theta}, \dot{\Psi},)$ from the drone plant as the input for the gyroscope plant.

In the simulation both accelerometer and the gyroscope plants are reading the ideal values from the drone plant, however in the physical model there would be noise present therefore we need to generate noise by adding a random number generator on the incoming signals. In the physical IMU there is also a bias on the readings, which can be measured and simulated in the readings.

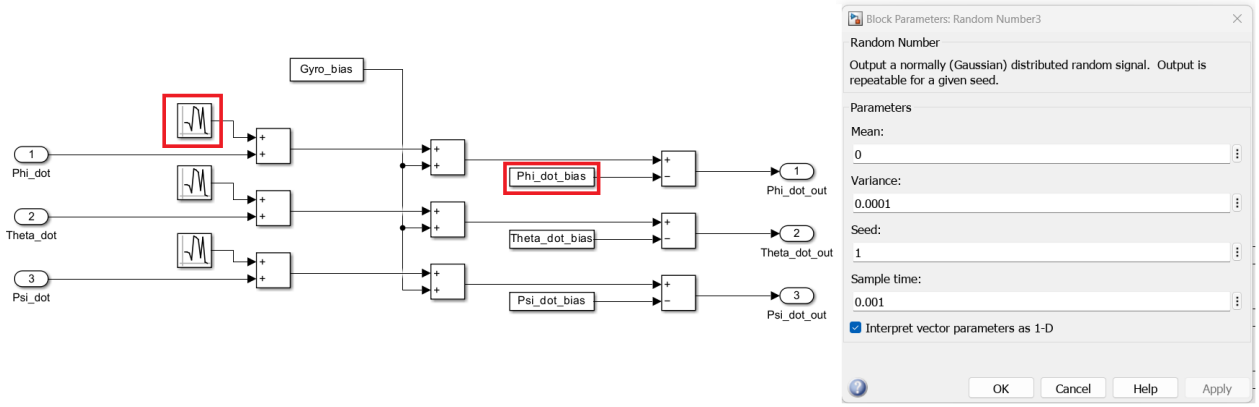


Figure 59: Gyroscope plant in Simulink

The IMU readings are then passed through the Complimentary filter plant, where gyroscope readings are used to calculate positional angles via integration and passed through a high pass filter and the angles calculated from accelerometer readings are passed through a low pass filter. For the high pass filter the transfer function used is $\frac{T_{high}*s}{T_{high}*s+1}$ where T_{high} is the period of the cutoff frequency of the filter $T_{high} = \frac{1}{f_c}$

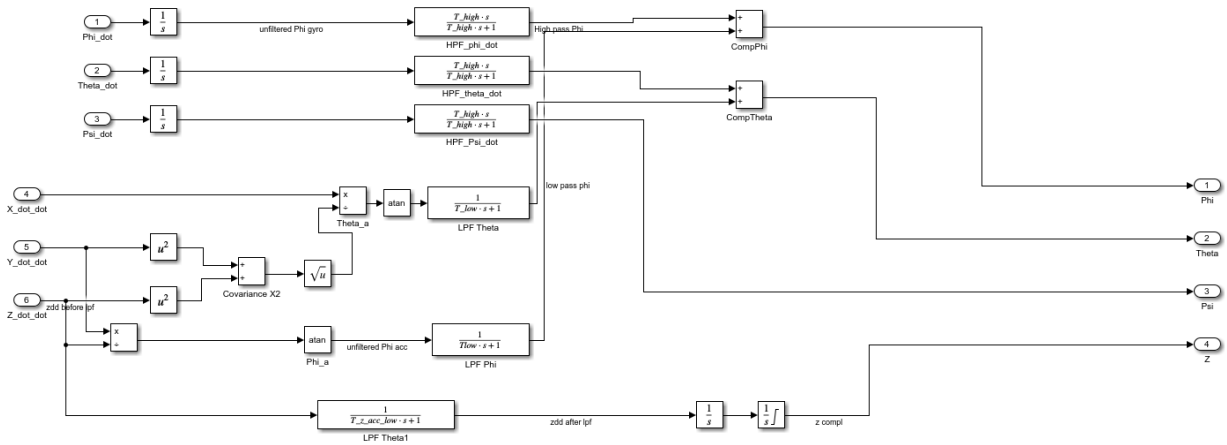


Figure 60: Complimentary filter plant in Simulink

The complimentary filter plant also includes calculation of vertical position Z by performing double integration of the vertical acceleration reading. In the physical model we have included a time of flight sensor for this measurement, however it is not modeled in the simulation as the sensor was a later addition. In ideal scenario we would also include a sensor that could be used as additional reference for the Yaw, for example, a magnometer, as only using the gyroscope reading to track this angle introduces drift. Unfortunately we only have an IMU with 6DOF. The filtered values are then converted back to NED reference frame by using calculations described previously and then passed back as feedback input for the control plant.

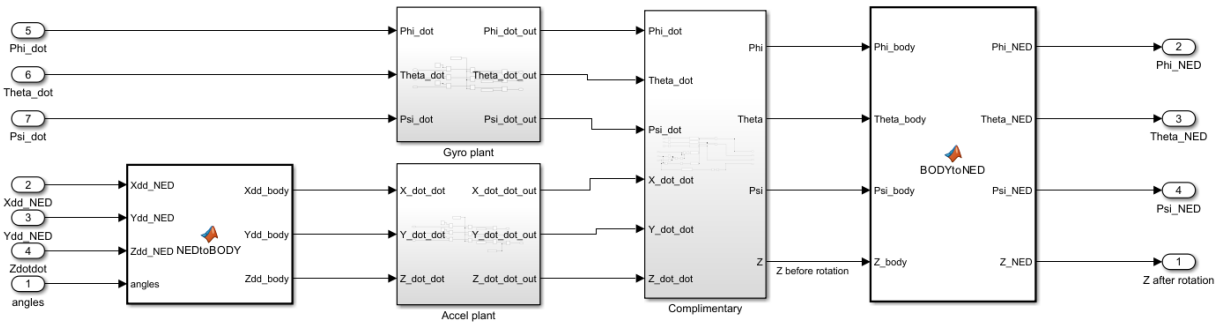


Figure 61: IMU plant in Simulink

6 IMU code

To obtain the IMU readings from our sensor we first need to establish connection with the micro-controller board and luckily using Arduino IDE makes this simple by offering Seeed Studio XIAO nRF52840 (Sense) board package. Additionally we need to include several libraries, such as math.h and LSM6DS3.h.

The LSM6DS3.h pre-defines the I2C address as 0x6B but can be overwritten in code - as in our case the address is in fact 0x6A. The library also pre-defines IMU settings -

enables both sensors and sets Gyro and Accelerometer sampling rate as 416 Hz. This means sampling time for IMU readings of 2.4 ms, which works with the code for drone control as the sampling frequency is higher than control input calculation frequency.

By default the maximum range is selected for both sensors - 16g for Accelerometer and 2000 dps for the Gyro. The library also conveniently performs the necessary calculations and returns the measurements in gravitational constants and degrees per second and sensor readings can be accessed by using functions readFloatGyroX() and readFloatAccelX().

```
settings.gyroEnabled = 1;  //Can be 0 or 1
settings.gyroRange = 2000;   //Max deg/s.  Can be: 125, 245, 500, 1000, 2000
settings.gyroSampleRate = 416;   //Hz.  Can be: 13, 26, 52, 104, 208, 416, 833, 1666
settings.gyroBandWidth = 400;  //Hz.  Can be: 50, 100, 200, 400;
settings.gyroFifoEnabled = 1;  //Set to include gyro in FIFO
settings.gyroFifoDecimation = 1;  //set 1 for on /1

settings.accelEnabled = 1;
settings.accelODROff = 1;
settings.accelRange = 16;      //Max G force readable.  Can be: 2, 4, 8, 16
settings.accelSampleRate = 416;  //Hz.  Can be: 13, 26, 52, 104, 208, 416, 833, 1666, 3332, 6664, 13330
settings.accelBandWidth = 100;  //Hz.  Can be: 50, 100, 200, 400;
settings.accelFifoEnabled = 1;  //Set to include accelerometer in the FIFO
settings.accelFifoDecimation = 1;  //set 1 for on /1
```

Figure 62: Default settings in LSM6DS3.h library

The IMU offers inbuilt analog Low Pass and High pass filters, however we have chosen not to use them and to only rely on the digital filters implemented by us.

There are two important additional sensor initiation steps that are not covered by the library - removing sensor bias and aligning the internal axis of the Gyro and Accelerometer units.

To calibrate the sensor, we use a simple *for* loop that runs 10 000 iterations reading the sensor output and averages the values. Since we know that when the sensor is sitting still, we should be reading 0 angular velocity, we can subtract these average readings from every future reading, thus creating the ReadGyro() function.

```
void calibrate_gyro()
{
  float sumX = 0, sumY = 0, sumZ = 0;

  for (int i = 0; i < 10000; i++)
  {
    readgyro();
    sumX = sumX + gyro[0];
    sumY = sumY + gyro[1];
    sumZ = sumZ + gyro[2];
  }

  gyro_bias[0] = sumX/10000.0;
  gyro_bias[1] = sumY/10000.0;
  gyro_bias[2] = sumZ/10000.0;
}
```

Figure 63: Gyro calibration

```
void readgyro_f()
{
  for (i=0; i<3; ++i)
  {
    gyro_prev[i]=gyro[i];
  }

  gyro[0] = myIMU.readFloatGyroX()-gyro_calib[0];
  gyro[1] = -(myIMU.readFloatGyroY()-gyro_calib[1]);
  gyro[2] = myIMU.readFloatGyroZ()-gyro_calib[2];
}
```

Figure 64: $\text{read}_{g}yro_{f}()function$

To calibrate the accelerometer, we need to change the orientation of the sensor between each for loop to have each of the positive and negative axis pointing downwards. As per instructions in the Data sheet the full range of readings on each axis should be $2g$.

The second initialization step is to align the axis for gyroscope and accelerometer before using the readings for any further calculations. The direction of the x,y,z axis is pre-set by factory, however it will not be the same for all sensors and since we are using a complimentary filter to fuse the measurements, it is crucial to make sure that the positive direction gyroscope and accelerometer axis are the same and that the positive angle of rotation matches the references used in motor control.
During testing one of the Seeed Xiao micro-controllers was damaged and had to be swapped for a new one. It turned out that on the new micro-controller board the direction of y axis was the opposite for both sensor units and as a result was affecting the filtered sensor readings and the control algorithm.

We introduce several functions needed for our positional calculations:
$readGyrof(), readAccf()$:
All sensor readings are stored in arrays of 3 float elements - one for each axis reading and since the readings from previous iteration are necessary for the digital filtering, before fetching the new sensor output values, old ones are shifted in a separate array.

$CalcAngleGyroF()$:
Calculates the Euler angles by "integrating" the gyroscope readings. It takes the last known angle of attitude and adds the current angular velocity times the time step of the full control loop. We can use this method since the drone will be launched from horizontal position therefore we can assume the inital conditions to be 0.

```
float calc_angle_gyro_f(float angle, float speed)
{
  float calculated = 0;  //calculated = Phi_unfiltered;
  calculated = angle + dt * speed*0.001;
  return(calculated);
}
```

Figure 65: CalcAngleGyroF() function

$UnfiltAngleGyroF(), UnfiltAngleGyroPrevF()$:
Needed for the digital filter - calculates unfiltered attitude angles based on current and previous gyroscope readings.

$CalcPhiAccF(), CalcThetaAccF()$:
Calculates Phi and Theta from Accelerometer readings interpreting full range of motion to be from $-\Pi$ to Π.

```
float calc_Theta_acc_f(float AccX, float AccY, float AccZ)
{
  float calculated;
  if  (AccZ>0)
  {
    calculated = atan(AccX/sqrt(AccY*AccY+AccZ*AccZ))*1/(3.142/180);//degrees roll / phi
  }

  if  (AccZ<=0)
  {
    if (AccX>=0)
    {
      calculated = 180-atan(AccX/sqrt(AccY*AccY+AccZ*AccZ))*1/(3.142/180);//degrees pitch / phi
    }
    if (AccX<0)
    {
      calculated = -180-atan(AccX/sqrt(AccY*AccY+AccZ*AccZ))*1/(3.142/180);//degrees pitch / phi
    }
  }
  return(calculated);
}
```

Figure 66: CalcThetaAccF() function

$HPFGyroAngleF(), LPFAccAngleF()$:

Calculates the values filtered via digital filters. Based on the following discrete time implementation of high pass/ low filters explained earlier in the document:

$$y(t) = a_o y_{n-1} + b_o x_n + b_1 x_{n-1}$$

We can create a function to carry out this calculation, where y_{n-1} is the previous filtered value, x_n is the unfiltered value based on current readings and x_{n-1} is the unfiltered value based on previous readings. The structure of both high pass and low pass filters is the same, however the coefficients calculated by the Bilinear transform differ. Most notably - the $b1$ coefficient for High pass filter is negative.

```
void HPF_gyro_angle_f()
{
  HPF_gyro_angle[0] = coefgyro_HPF[0] * HPF_gyro_angle[0] + coefgyro_HPF[1] * calc_angle_gyro_f(Unfilt_angle_gyro[0], gyro[0]) + coefgyro_HPF[2]* cal
  HPF_gyro_angle[1] = coefgyro_HPF[0] * HPF_gyro_angle[1] + coefgyro_HPF[1] * calc_angle_gyro_f(Unfilt_angle_gyro[1], gyro[1]) + coefgyro_HPF[2]* cal
  //HPF_gyro_angle[2] = coefgyro_HPF[0] * HPF_gyro_angle[2] + coefgyro_HPF[1] * calc_angle_gyro_f(Unfilt_angle_gyro[2], gyro[2]) + coefgyro_HPF[2]* c
}
```

Figure 67: HPFGyroAngleF() function

```
float coefgyro_HPF[3] = {0.6828, 0.8414, -0.8414}; // HPF coefficients
float coefacc_LPF[3] = {0.6828, 0.1586, 0.1586}; // LPF coefficients
```

Figure 68: Digital filter Bilinear transform coefficients

$Complimentary F()$:
Combines high pass and low pass filter output.

For the time of flight sensor we need to include the
$Adafruit_V L53L0X.h library and then the vertical distance can be measured with the help of measure.Range M$

```
Z_actual = measure.RangeMilliMeter*cos((3.142/180)*Complimentary[0])*cos((3.142/180)*Complimentary[1]); // distance to ground in mm
```

Figure 69: Calculating Z position from time of flight sensor

To choose optimal cut-off frequencies for the digital filters, we can print Complimentary filter output alongside the unfiltered Euler angles from gyroscope and accelerometer to see how much the filtered signal is attenuated and observe the phase shift.

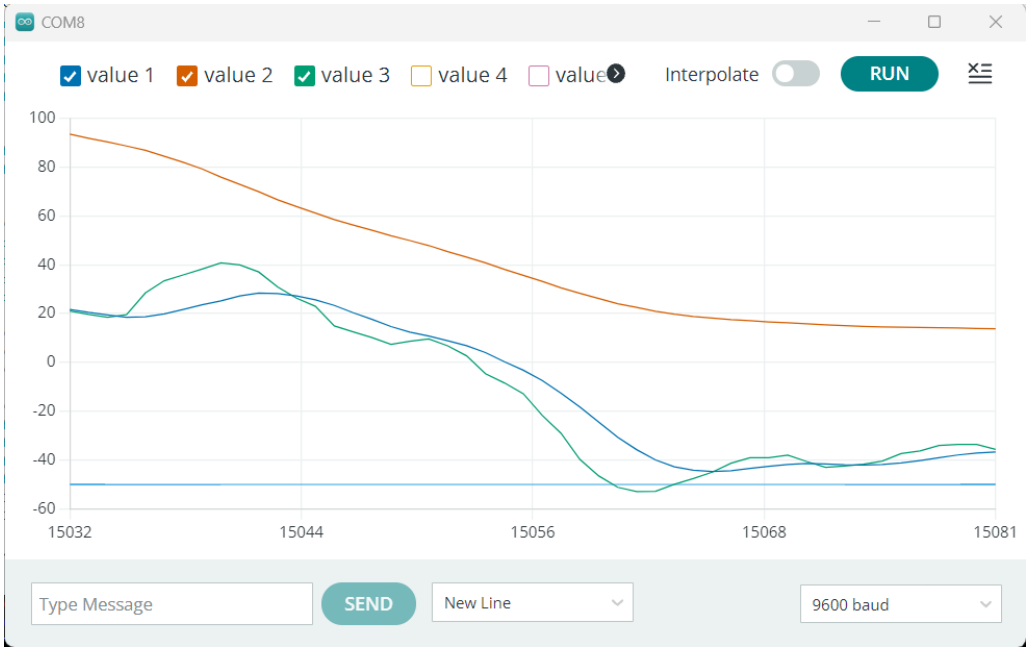


Figure 70: Complimentary filter output vs unfiltered values