University of West Bohemia in Pilsen

Faculty of applied sciences

Department of Cybernetics

MASTER'S THESIS

PILSEN, 2022 MILAN MALINA

DECLARATION

I hereby submit for assessment and defense a master's thesis prepared at the end of my studies at the Faculty of Applied Sciences of the University of West Bohemia in Pilsen.

I declare that I have composed the master's thesis independently and exclusively using the professional literature and sources, the complete list of which is a part of it.

In Pilsen (date):

……..……..

signature

ACKNOWLEDGEMENT

I would like to thank my supervisor Ing. Martin Goubej, Ph.D. for his help and recommendations during the running of this project. His input was helpful. Furthermore, I would like to thank my parents Milan Malina, Hana Malinová for their help with practical testing and financial support. Lastly, I would also like to thank my girlfriend Milagros E. Chávez for her motivation and support, especially when it was needed.

Annotation

The master's thesis Design, assembly and automatic control of unmanned aerial vehicle prototype is divided into eight sections. First, construction of a mode aircraft is described. Second, the development of avionics components of this aircraft is described. That is followed by electronic design of the flight computer itself. After that, attitude estimation system is developed using Extended Kalman Filter. Next, Telemetry and Control app and realistic HIL simulation environment is designed in Unity. Lastly, 3 linear uncertain model are identified using the simulation data and are then used to perform a robust performance analysis with 3 PID controllers which are designed using MATLAB PID Tuner.

KEYWORDS

UAV, aircraft, control system design, controller, PID, model, linear model, control software, aerodynamics, system identification, robust control

Anotace

Magisterská práce na téma Návrh, realizace a automatické řízení prototypu bezpilotního letounu je rozdělena do osmi sekcí. Nejprve je popsána konstrukce bezpilotního letounu. Za druhé je popsán vývoj avioniky tohoto letounu. Následuje popis návrhu samotného palubního počítače. Poté je popsán způsob odhadu polohy za použití rozšířeného Kalmanova filtru. Dále je navrhnuta jak aplikace pro zobrazování telemetrických dat a ovládání tak realistická HIL simulace v Unity. Nakonec jsou identifikovány 3 neurčitostní lineární modely za použití dat z HIL simulace. Ty jsou poté použity k provedení analýzy robustní kvality řízení se třemi PID regulátory, které jsou navržené pomocí MATLAB PID Tuner.

KLÍČOVÁ SLOVA

UAV, letadlo, návrh řídicího systému, regulátor, PID, model, lineární model, řídicí software, aerodynamika, identifikace systému, robustní řízení

1 Table of contents

2 Acronym list

Acronym	Definition
A320	Airbus A320
B737	Boeing 737
BLDC	Brushless Direct Current
CA	Cyanoacrylate
CAD	Computer Aided Design
CG	Center of Gravity
COM	Communication
EDF	Electric Ducted Fan
EKF	Extended Kalman Filter
ELEC	Electrical
F/CTL	Flaps and Control Surface
FC	Flight Computer
FM	Flight Manager
FSU	Flight Stabilization Unit
HIL	Hardware In the Loop
I2C	Inter-Integrated Circuit
IAS	Indicated Airspeed
IMU	Inertial Measurement Unit
LOC	Location
MCP	Mode Control Panel
MCS	Master Caution System
MISCCU	Miscellaneous Control Unit
PFD	Primary Flight Display
PLA	Polylactic acid
PWM	Pulse-width modulation
RBI	Remote Button Indicators
RMCU	Retract Motors Control Unit
RVF	Remote Value Fields
SPI	Serial Peripheral Interface
TAS	True Airspeed
TPU	Thermoplastic polyurethane
UAV	Unmanned Aerial Vehicle
UML	Unified Modeling Language

3 Introduction

The purpose of this thesis is to develop a control system for an aircraft, which is capable of autonomous flight with minimum interference of the operator. Multiple different sections describe this process: UAV assembly, avionics design, telemetry and control app development, HIL simulation development and lastly controller synthesis and analysis.

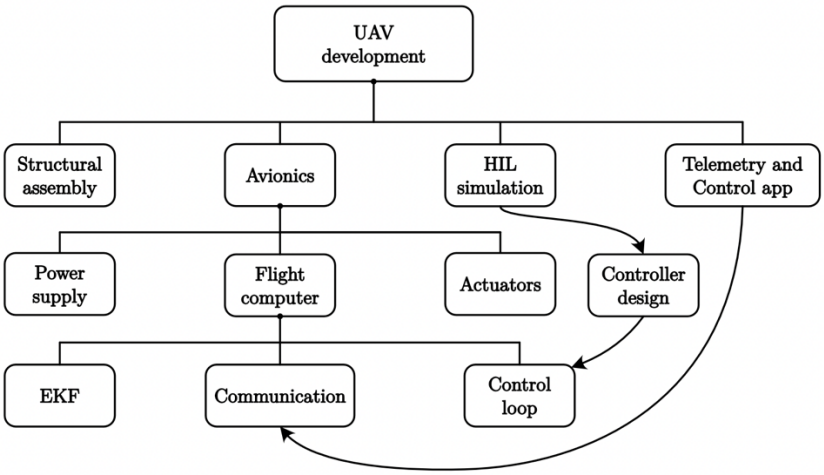

Figure 1: UAV development diagram

The motivation and goal behind this project and therefore this thesis is to develop a custom control system and evolve a deep and firsthand understanding of the technical and practical difficulties in order to earn as much experience in different (mainly but possibly not exclusive to) technical disciplines and areas. This is one of - but not the only - reason why I decided to develop a control system, essentially, from zero and was not inclined to using already developed flight controllers, either commercial or more hobby-grade, such as Pixhawk, Betaflight or Ardupilot.

The use for larger scale drones and UAVs is vast and ranges from civilian use, such as delivery of goods or terrain mapping, through use in emergency services, such as blood transport for patients in critical condition, surveillance or help in firefighting, to use in military applications, such as reconnaissance missions, remote target engagement, etc.

Another reason for developing the whole system myself is to have full control over the functionality and features and not be dependent on sometimes limited or not fully documented option to make changes in

implementation or different levels of functionality of already developed flight controllers. This is more of a personal preference, and it is not an objective statement of which solution is better.

4 UAV assembly

4.1 Introduction

As already stated, the design of the UAV is a scaled model of L-39 fighter aircraft. 3D files were acquired online and the loaded into a 3D slicing software which generated g-code files which were then loaded into a 3D printer. Some files had to be first slightly modified before the slicing process. This was done to ensure compatibility with other components that had to be ordered online (such as servomotors, shock absorbers, pushrods, etc.). As a modeling software for modifying and modeling new parts, Fusion 360 by Autodesk was used. For slicing, Ultimaker Cura was used. Material used is PLA for the most part. The exception are tires and some EDF components. Those are printed using TPU rubber material.

The figure below shows the overall design of the aircraft:

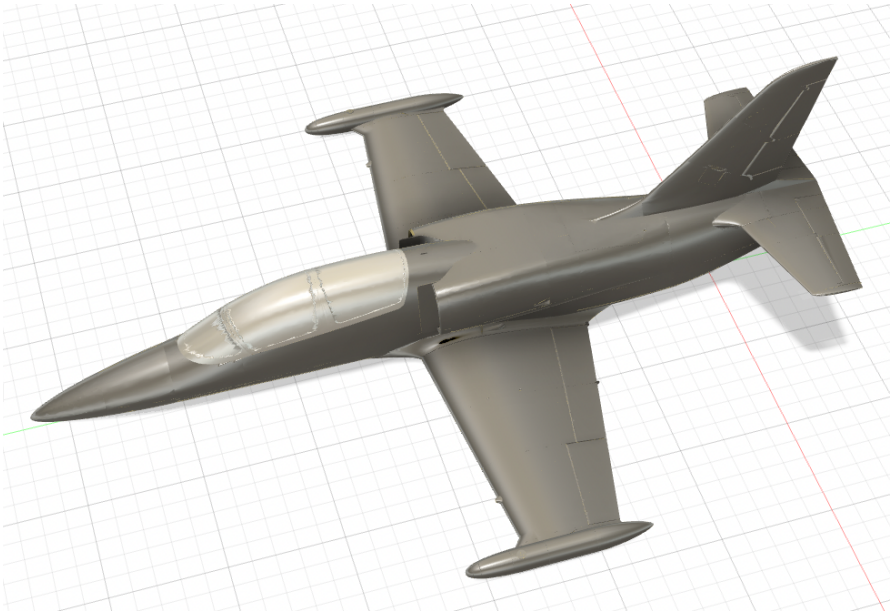

Figure 2: UAV overall design

4.2 Nose and avionics bay assembly

The nose and the avionics bay consist of 6 individual segments. These are connected using carbon rod segments and glued together using CA glue. The front landing gear is stored inside of the second and third segment. A nose landing gear door is used as a cover when the landing gear is in transit from UP→DOWN or DOWN→UP position.

Moving further, the avionics bay follows. It houses power source in a form of lithium-polymer batteries, flight computer which is described in higher detail in the avionics design chapter. The following figure shows computer visualization of the avionics bay fitted with the flight computer components:

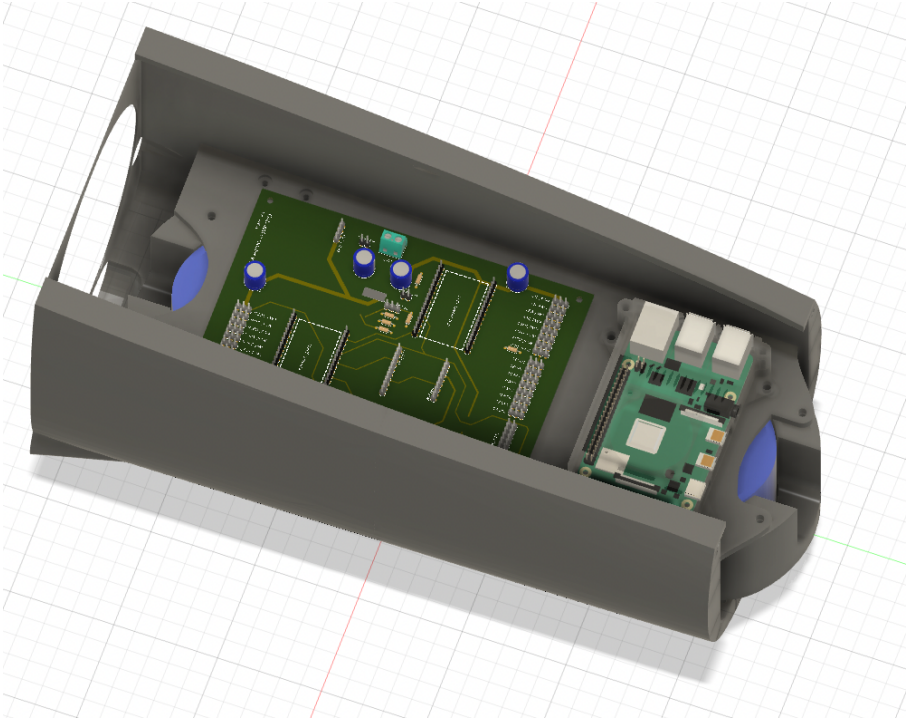

Figure 3: Avionics bay with FC components in Fusion 360

The batteries are placed on the bottom of the avionics bay. Other electronic components are placed above that and screwed to a 3D printed plate which separates the PCBs from the batteries.

4.3 Center fuselage section

This section of the fuselage contains two air-intakes (one on each side) which are used to direct the air to the EDF. This section, as well as the rest of the sections of the fuselage and wings provide cavities for the cables to be led through. It consists of 3 individual segments. These are again connected using carbon rod segments and glued together using CA glue. There are two carbon tubes connecting main wings and providing enhanced structural stability. The most inner section of the main wing is permanently glued to the fuselage and the rest of the main wings is detachable. This ensures that the UAV can operate on the landing gear even when the main wings is detached. The main landing gear is stored in a main landing gear bay which is placed inside the fuselage. Analogically to the nose landing gear, main landing gear doors are used as a cover when the landing gear is in transit from UP→DOWN or DOWN→UP position.

The following figure shows the UAV without wings attached:

Figure 4: UAV without wings attached

The landing gear is connected to the fuselage with electric retract system. This enables the landing gear assembly to rotate 90 degrees so that it can be deployed and retracted when needed. The graphics below shows one retract for an illustration:

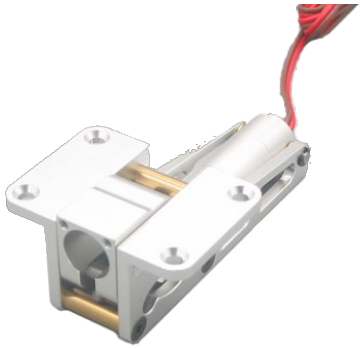

Figure 5: Retract unit

4.4 Rear fuselage section

This section of the fuselage contains the EDF itself which functions as a propulsion unit for the whole aircraft. This section consists of 3 individual segments. These are again connected using carbon rod segments and glued together using CA glue. On top there is an EDF hatch. This hatch is removable and is used to service the whole EDF assembly located in the center of the fuselage. The EDF itself is described in the following chapter in more detail. The rest of this fuselage section contains an air-outtake through which the air is propelled by the EDF.

The following figure shows the EDF assembly installed inside of the fuselage:

Figure 6: EDF assembly installed inside of the fuselage

The three black cables at the top of the image supply 3-phase signal to a BLDC motor inside of the EDF. The rest of the cables are used to connect elevator and rudder servomotors and transfer information from a temperature sensor (Inside the EDF).

4.5 EDF assembly

The EDF assembly consists of BLDC motor, impeller, EDF housing, fan-intake, rails to which the housing is crewed and rubber spacers (printed from TPU) which are used to reduce vibrations from the EDF. The BLDC motor is equipped with a heatsink in order not to overheat. There is a temperature sensor mounted to the motor-heatsink assembly to ensure a safe operation. The EDF assembly also includes two seal tubes to increase the "air-tightness" and easy the installation of the assembly. The following figure illustrates how the EDF assembly is assembled:

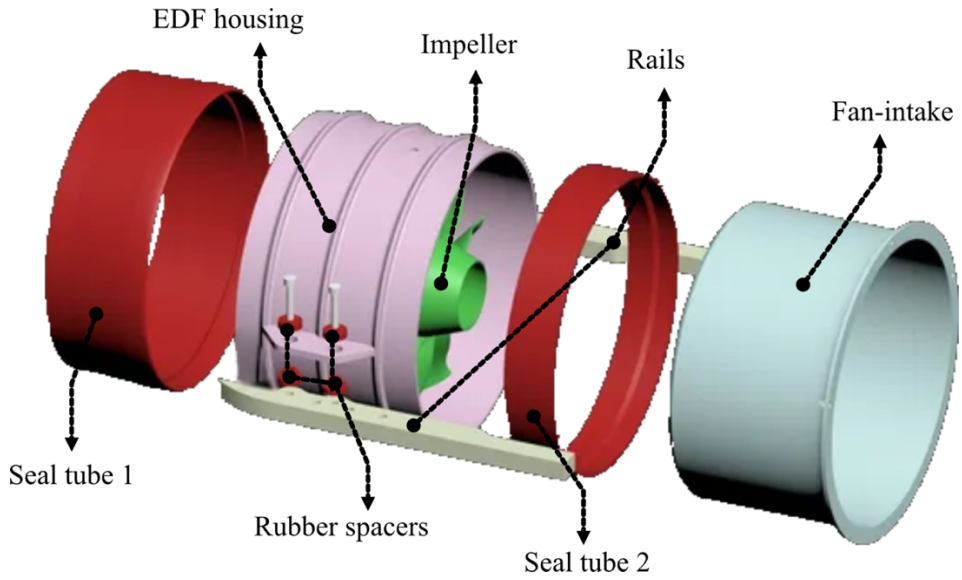

Figure 7: EDF assembly

The motor that powers the EDF is a 4.8kW BLDC motor. It runs at around 45-50V. Because of the high power, a proper cooling must be ensured. That is in a form of added heatsink and air flow cooling the motor to reasonable temperatures. For safety reasons, there is a temperature sensor mounted right on the motor (inside a heatsink gab). The temperature information is monitored by the flight computer and a warning issued when the temperature is not in safe limits. The following figure shows the motor itself:

Figure 8: Typhoon HET 800-73 Motor

The following figure shows the motor with the heatsink installed:

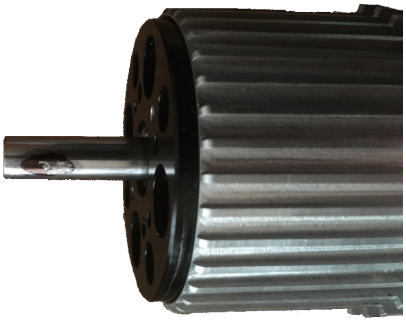

Figure 9: BLDC motor with heatsink installed

The following figure shows the EDF housing fitted with the motor and the impeller:

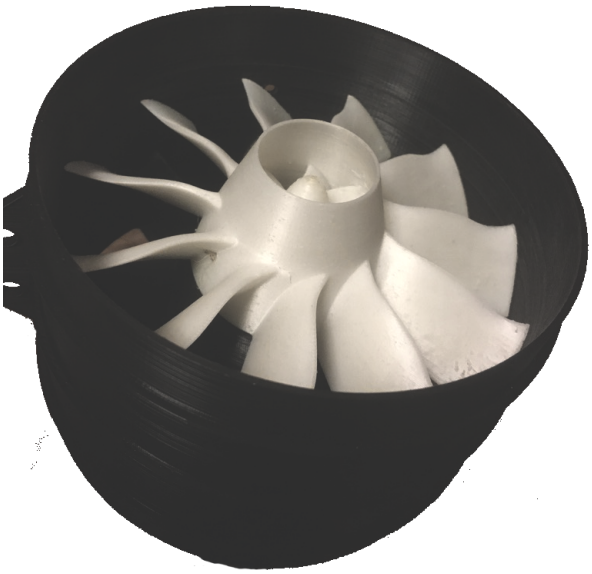

Figure 10: EDF housing fitted with the motor and the impeller

The EDF assembly is one of the most advanced parts of the aircraft. The impeller reaches very high RPM and therefore must be properly tested before increasing the power of the motor. It can be dangerous if tested or operated without necessary safety equipment and procedures.

4.6 Main wing, vertical and horizontal stabilizer

Main wings include ailerons in order to control the aircraft along the longitudinal axis (roll) and flaps to increase lift generated by the airfoil especially when flying at low speeds (landing, taking off). Both, ailerons

and flaps are manipulated by individual servomotor. All control surfaces of the aircraft are fitted with bearings to move with as low resistance as possible. Left and right main wing are detachable from the fuselage in order to transport aircraft easier and for possible maintenance needs. The left main wing is pictured in the figure below:

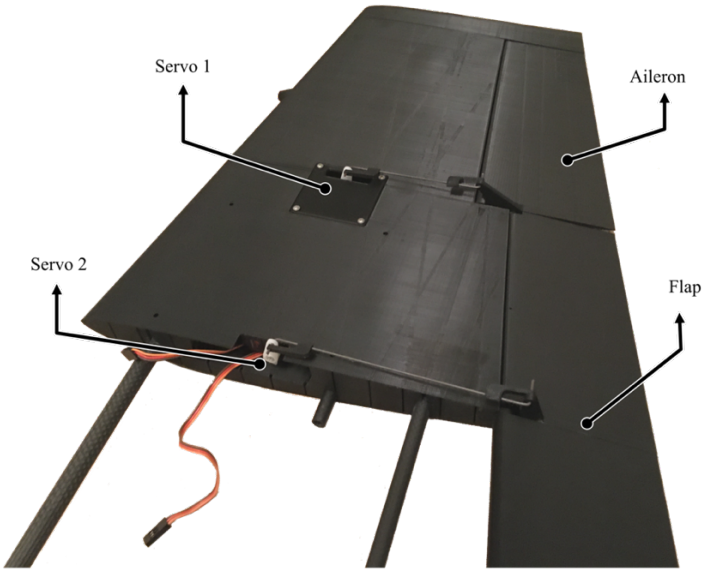

Figure 11: Left main wing

Servo 1 controls the aileron to which is coupled through a thin steel piano wire and secured with a custom 3D printed clevis. This form of physical connection is utilized for all control surfaces of the aircraft. Servo 2 controls the flap.

The vertical stabilizer contains the rudder control surface for control in the vertical axis (yaw). The horizontal stabilizer contains left and right elevators to control the aircraft in the lateral axis. Vertical and horizontal stabilizers including their control surfaces are assembled using the same technique as the main wing.

4.7 Conclusion

The weight of the assembled UAV with the electronics fitted in is approximately 8.6 kg. Which is still in the acceptable range. Based on the data provided by the model designer, the maximum take-off weight is 10 kg.

5 Avionics design

5.1 Introduction

In this chapter the goal is to use UML to describe an UAV autopilot system including the avionics. The avionics divided into multiple sections and each of them is also described in more detail. In order to model the hardware characteristics (from mechanical and electrical perspective), an UML structural diagram is used. The following figure illustrates the system as a whole:

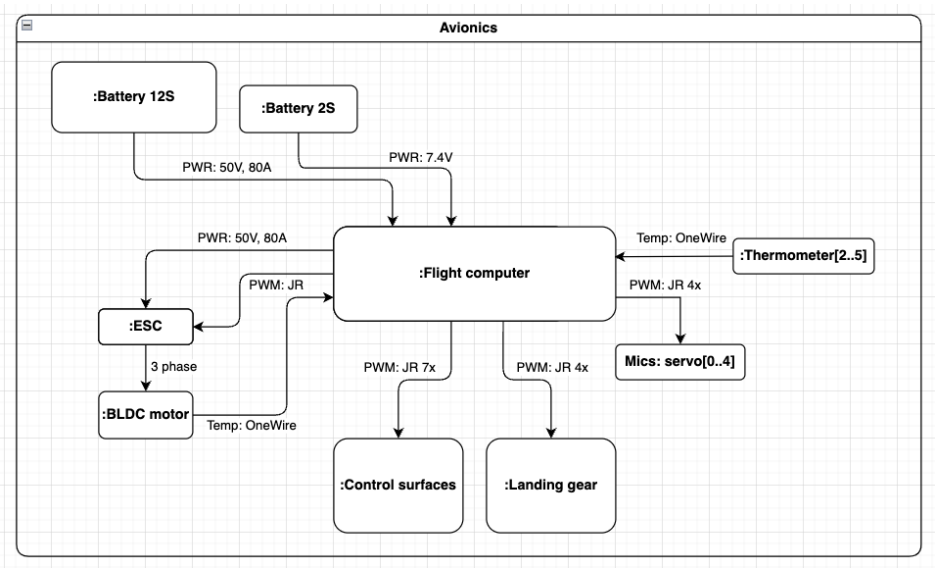

Figure 12: Avionics general diagram

As show in the figure above, the Aircraft Control System diagram consists of multiple subsystems. The more complicated ones are more closely described below. Those include battery (2S and 12S), control surfaces, landing gear and the flight computer.

5.2 Battery

This diagram describes both 2S[1] and 12S[2] battery. Those are assembled by connecting 1-cell (for 2S) and 6-cell (for 12S) batteries in series.

[1] Referring to a 2-cell lithium polymer battery

[2] Referring to a 12-cell lithium polymer battery

Balance ports are wired individually so that both batteries can be charged using a dual charger in parallel. 2S battery outputs around 7.4V and is used to power the avionics and low-power actuators. 12S battery outputs around 45-50V and is used to power a propulsion unit which is made of an EDF[3]. The battery assembly is illustrated in the following diagram:

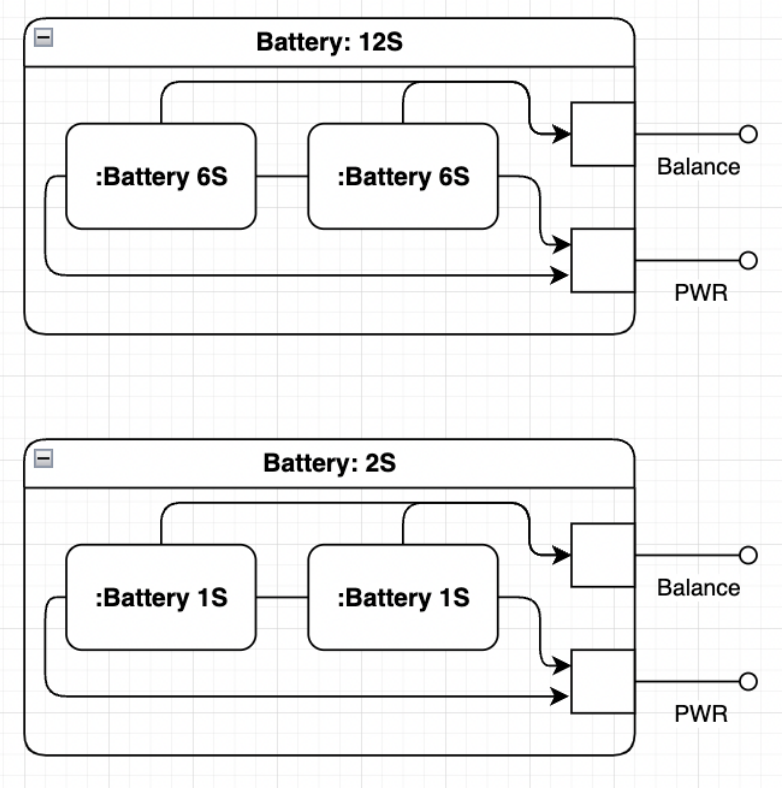

Figure 13: Battery structural diagram

5.3 Control surfaces

In order to achieve stable flight or change the attitude of the aircraft, control surfaces must be used. As described in the chapter "UAV assembly" those include ailerons to control the roll angle, elevators to control the pitch angle and rudder to control the yaw or the aircraft. Flaps are used to increase the lift generated by the main wing without increasing the AoA[4] so that stall does not occur even at lower speeds (for example

[3] Referring to an electric ducted fan which is located inside of the aircraft

[4] Referring to angle of attack - the angle between the chord line of an airfoil and the aircraft's velocity vector

during take-off and landing). Each control surface is manipulated by a servo motor which is controlled using PWM signal as shown in the following figure:

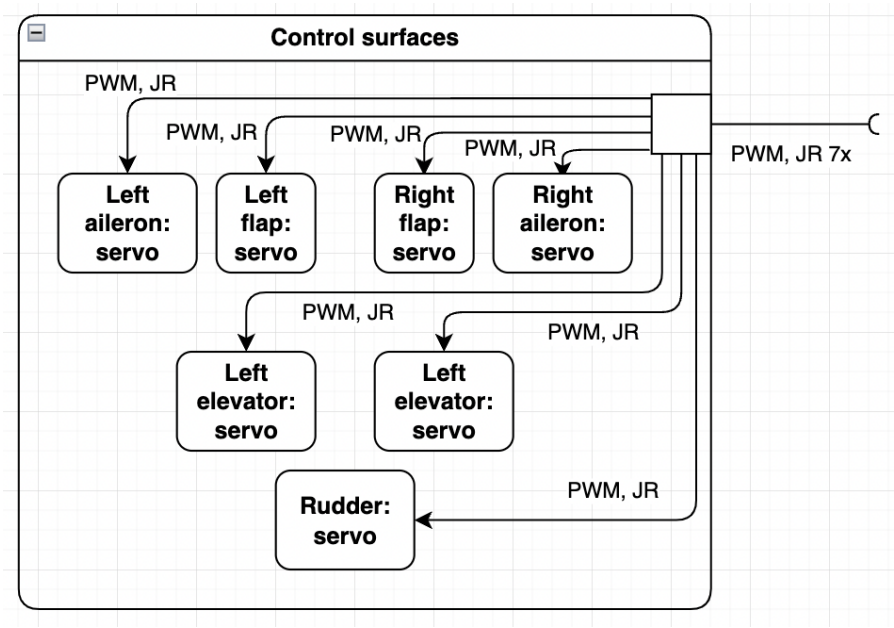

Figure 14: Control surfaces structural diagram

5.4 Landing gear

In its retracted state (gear up), the landing gear is hidden inside a landing gear bay and covered by dedicated landing gear cover plates (landing gear bay door). Those and the landing gear retracts are both controlled by the retracts control unit. There is an option for adding left and right brakes which could be are used to bring the aircraft to stop after landing and during taxi more efficiently. Those would be part of the main landing gear assembly and controlled by individual servomotors. In order to steer the aircraft when on the ground, during taxi, the nose landing gear is equipped with a steering assembly which is controlled by another servomotor. All the functionality mentioned is summarized in the following figure:

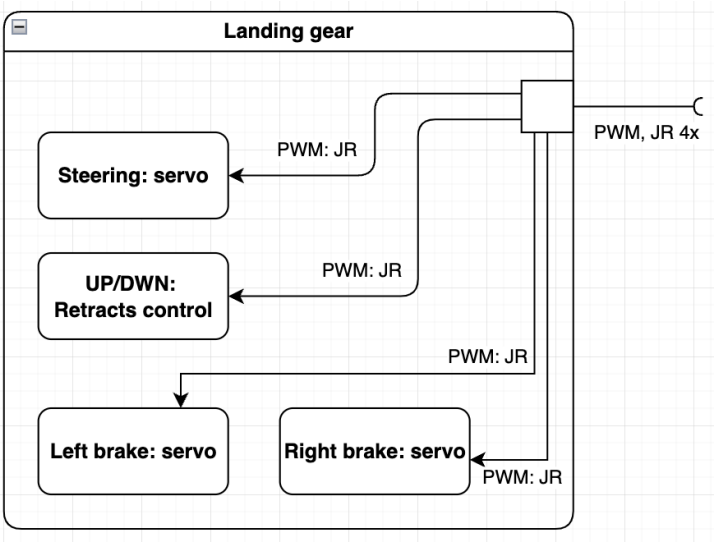

Figure 15: Landing gear structural diagram

The two main landing gear are assembled in a similar way. The nose landing gear includes a steering assembly driven by a servomotor. The steering assembly uses a ball bearing to reduce friction when turning and is coupled with the servomotor through a thin steel piano wire. To reduce structural load when landing, landing gear is equipped with a shocked absorber which is filled with 10000 CST differential oil. As mentioned above, the tires are 3D printed with TPU filament. This makes sure the tires are slightly compressible and therefore also help with reducing the structural load. The following graphics illustrates the layout of the front landing gear:

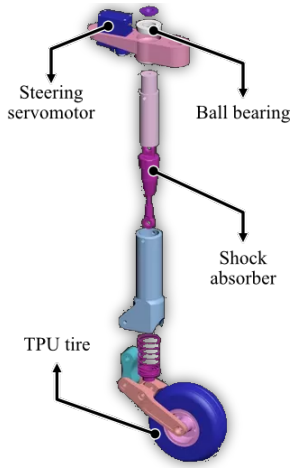

Figure 16: Landing gear assembly

5.5 Flight computer

The most important and complex part of the avionics is the flight computer itself. It mainly consists of flight manager, flight stabilization unit, Miscellaneous control unit. Flight stabilization unit (FSU) is described in more detail, from the software perspective using a class diagram later in the work. Flight manager is used to communicate over network with the user as well as communicate over I2C with FSU and Miscellaneous control unit. Miscellaneous control unit mainly controls the landing gear functionality and collects information from the temperature sensors. Inertial measurement unit, barometric sensor, self-diagnostics components and human interface input/output components are also included as shown in the figure below:

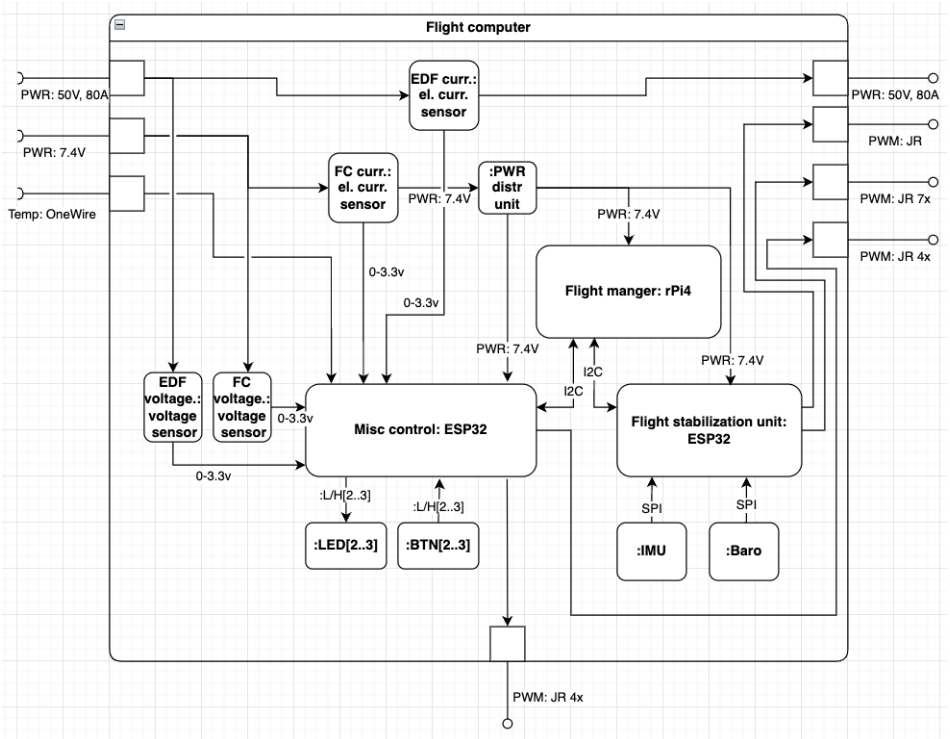

Figure 17: Flight computer structural diagram

5.6 FSU - software implementation

As stated above, the flight stabilization unit is part of the flight controller. Its task is to control the control surfaces and the propulsion unit based on the commands received from the flight manager, calculate the aircraft's attitude, gather the telemetry data, etc. In order to achieve this task, FSU is itself divided into in total 8 bigger classes. The programming language of the current implementation is C++, mainly because of its speed.

Class *QuatIMUHandler* handles the communication with the IMU, calibration of the IMU, providing data to and gathering from the EKF. *QuatKalman* class contains the implementation itself. It is responsible for all the matrix calculations necessary. Classes *ComHandler* and *CMDHandler*, like the names suggest, handle the communication and command execution. *MCPHandler* is responsible for controlling the autonomous aspects of the system - the *PIDHandler* containing the PID controller's implementation, as well as enabling/disabling manual or automatic control completely. *CtrlSurfAndThrottleHandler* controls the control surfaces, flaps and throttle of the aircraft. The main class is the *Flight_Stabilization_Unit* class. It contains references to other classes and runs the main control loop and other critical loop using the RTOS implementation.

The implementation can be described by the UML class diagram shown on the next page. This diagram was created during the development of the project, so the final implementation (names, etc.) may slightly vary.:

Figure 18: FSU class diagram

5.7 Communication and command execution

The FSU must be able to exchange information both ways with the flight manager. For example: desired attitude of the aircraft as a setpoint for the controller (flight manager → FSU), user set calibration setting for the control surfaces (flight manager → FSU) or calculated attitude of the aircraft for visualization, etc... (FSU → flight manager). This is done over I2C using the class called *ComHandler*.

Each data variable being provided to the Flight Manager (through the I2C, but no exclusive to) is described in the table below, which each respected source and description.

Table 1: Communication variables

IDX	ABBRV	DESC	FSU	MISCCU	EXT
0	DTIM	datetime	X		
1	PTCH	pitch	X		
2	ROLL	roll	X		
3	HDG	heading	X		
4	ATMP	atm press	X		
5	MSL	mean sea level	X		
6	PIIN	pitch_input	X		
7	ROIN	roll_input	X		
8	YWIN	yaw_input	X		
9	THIN	throttle_input	X		
10	LADF	LA_def	X		
11	RADF	RA_def	X		
12	LFDF	LF_def	X		
13	RFDF	RF_def	X		
14	LEDF	LE_def	X		
15	REDF	RE_def	X		
16	RDDF	RD_def	X		
17	EDFP	edf_pow	X		
18	GRUD	gear_ud		X	
19	EDFT	edf_temp		X	
20	RPIT	rpi temp			X
21	BMPT	bmp temp	X		
22	C1LTC	client 1 ping latency			X

23	C2LTC	client 2 ping latency			X
24	FCPU1	FSU CPU 1 load	X		
25	FCPU2	FSU CPU 2 load	X		
26	MCPU1	MISCCU CPU 1 load		X	
27	MCPU2	MISCCU CPU 2 load		X	
28	CLTP	control loop time period	X		
29	KECV	Kalman estimation covariance matrix trace	X		
30	GLAT	GPS latitude	X		
31	GLNG	GPS longitude	X		
32	GSPD	GPS speed	X		
33	GPSP	GPS data precision	X		
34	LVBL	Low voltage battery voltage		X	
35	HVBL	High voltage battery voltage		X	
36	ESCT	ESC temperature		X	
37	BTCT	Battery compartment temperature		X	
38	MCPE	MCP enable status 0/1	X		
39	MCPM	MCP manual input	X		
40	MCPS	MCP setpoint input	X		
41	FLTCY	FSU I2C latency	X		
42	MLTCY	MISCCU I2C latency		X	
43	QBCLS	Stream queue data backlash	X	X	

The following two tables show how data is packed and its corresponding identifiers. If DTYPE is BYTE, it means that the data is scaled (using the min, max values) and pre-packed into an array of 4 bytes. This results in lower precision of this data but also in less data required to be transferred. For this reason, this is only applicable for variables that don't need to hold high precession and are ranged, either inherently, or for this purpose.

Table 2: FSU packs

ID	POS1	POS2	POS3	POS4	DTYPE	MIN	MAX
1	PTCH	-	-	-	FLOAT	-	-
2	ROLL	-	-	-	FLOAT	-	-
3	HDG	-	-	-	FLOAT	-	-
4	PIIN	ROIN	YWIN	THIN	BYTE	-1	1

5	LADF	RADF	LFDF	RFDF	BYTE	-1	1
6	LEDF	REDF	RDDF	EDFP	BYTE	-1	1
7	ATMP	-	-	-	FLOAT	-	-
8	MSL	-	-	-	FLOAT	-	-
10	FCPU1	FCPU2	BMPT	FLTCY	BYTE	0	100
12	CLTP	QBCLS	-	-	BYTE	0	100
13	KECV	GPSP	-	-	BYTE	0	3
14	GLAT	-	-	-	FLOAT	-	-
15	GLNG	-	-	-	FLOAT	-	-
16	GSPD	-	-	-	FLOAT	-	-
17	MCPE	MCPE	-	-	BYTE	0	255
18	MCPM	MCPM	MCPM	MCPM	BYTE	-100	100
19	MCPS	MCPS	MCPS	MCPS	BYTE	-100	100

Table 3: MISCCU packs

ID	POS1	POS2	POS3	POS4	DTYPE	MIN	MAX
100	LVBL	HVBL	MLTCY	-	BYTE	0	100
101	EDFT	ESCT	BTCT	-	BYTE	-20	120
102	MCPU1	MCPU2	QBCLS	-	BYTE	0	100

To clarify, the ID column in the tables above does refer to the identifier of the pack but not identifier of the command. The identifier of the pack is used to determine which variable and in which format is being transmitted and the command identifier is used to determine what type of command is being transmitted in general (To transmit data variables a specific constant command identifier is used - *CMD_UPDATE_STATE_DATA*) - more about that in the following paragraphs.

In order to execute received commands, the class *CMDHandler* is used. It contains the pointers to the other classes so that it can directly call corresponding methods. Depending on the identifier of the command, (those are defined in a special header file) it executes the corresponding command.

Commands are transmitted in packet. Each command can contain one or more data variable. It has a fixed size of 8 Bytes and composes of an identification identifier (1 Byte), parameters (3 Bytes) and data payload (4 Bytes). Data payload can be either one 32bit integer, array of 4 (either signed or unsigned) 8bit integers or one floating point value.

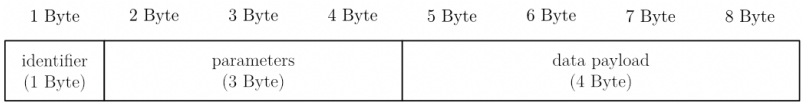

Figure 19: I2C package layout

5.8 Attitude computation

In order to calculate attitude of the aircraft, IMU9250 module is connected and communication established over SPI. The class *QuatIMUHandler* is responsible for receiving data containing information about linear acceleration (accelerometer), angular velocity (gyroscopic sensor) and magnetic vector (magnetometer) from this module. This "raw" information is then filtered using Kalman filter provided by *QautKalman* class. This class is responsible for running an implementation of extended Kalman filter. Quaternion arithmetic is used for all attitude calculations because it has many advantages over using Euler angles (avoiding gimbal lock, etc..) and that is the reason why classic Kalman filter algorithm cannot be used and extended Kalman filter must be used instead for attitude estimation.

5.9 Actuator control

Before the PWM signal is sent to the actuators, the appropriate manipulated variable must be computed. This is done using the *MCPHandler* class. This class oversees the individual controllers. Depending on the commands received, this class selects which controllers are used and what data is passed onto these controllers. In the current implementation, there are standard PID controllers (3 PID controllers for 3 process values: pitch, roll, speed) implemented inside the *PIDHandler* class.

The data from the *MCPHandler* class is then sent over to the actuators using PWM signal by the *CtrlSuftAndThrottleHandler* class. This class can also be used perform calibration of the control surfaces, set limits for the control surfaces and to manually control the co control surfaces and the propulsion unit.

6 Flight computer PCB design

6.1 Introduction

As described above, the FC (flight computer) consists of mainly 3 components. Those are Raspberry Pi 4 acting as a master – flight manager (FM) and two ESP32s acting as flight stabilization unit (FSU) and miscellaneous control unit (MISCCU). As described above FM communicates with FSU and MISCCU over I2C. The main role of the FM is to communicate with the ground station and command the FSU and MISCCU so that the whole system works as required by the operator. The FM is connected to the internet over LTE so the theoretical range is unlimited (when mobile data coverage is ensured). FSU and MISCCU are placed inside a custom-made PCB. This PCB also houses other important modules and pins to interface with servomotors.

The graphics below shows the layout of the avionics with the main components placed on top of the separation plate. The Raspberry Pi 4 is enclosed inside of a 3D printed enclosure which is screwed to the plate. The graphics is rendered inside Fusion 360:

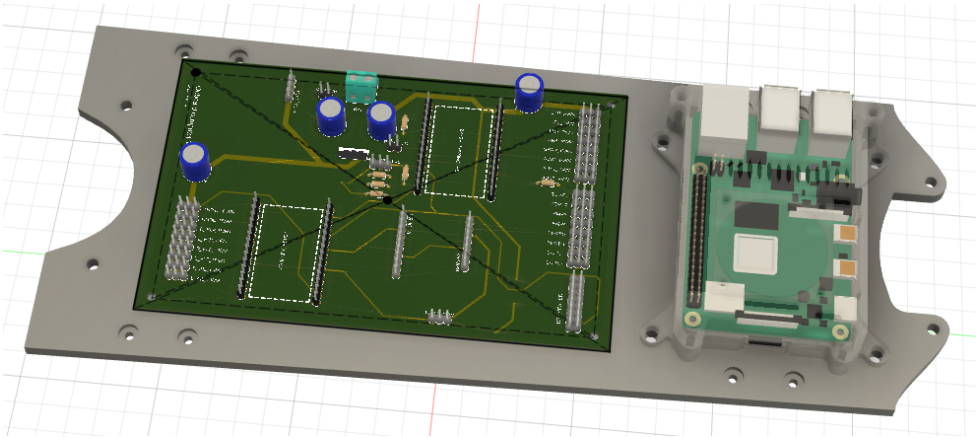

Figure 20: Avionics layout

6.2 Custom PCB schematics

6.2.1 Power supply and distribution

The power output from the 2S lithium-polymer battery is connected through a master switch to a step-down buck converter. Linear voltage regulator was initially indented to be used but because of its low efficiency (and therefore reaching high temperatures) and low voltage regulation quality, more expensive and higher quality buck converter is used instead. There are electrolytic capacitors used near the servo pins to help with the power stability.

There is also power a LED added for indication and voltage divider to be able to measure a battery voltage in order to determine the percentage. It has an optional 100 nF ceramic capacitor for filtration. The following figure shows the schematic diagram of this section:

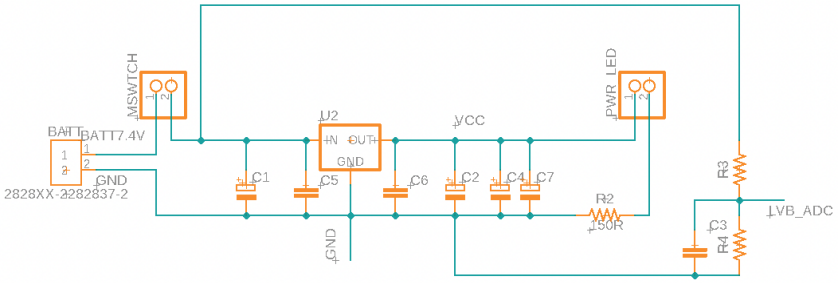

Figure 21: Power supply and distribution

There is a second voltage divider whose purpose is to measure the voltage of the 12S battery. It also has an optional 100 nF ceramic capacitor for the purpose of filtration. Both voltage dividers are connected to an ADC enabled pin of the MISCCU. It is visible in the schematic diagram below:

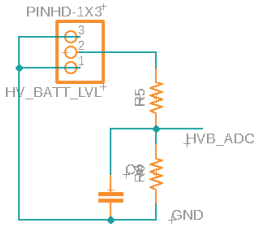

Figure 22: HV voltage divider

6.3 ESP32s and Raspberry Pi 4 connection

As stated above, the FC contains two ESP32s (FSU and MISCCU) and one Raspberry Pi 4 (FM). FSU and MISCCU are connected to the FM over I2C. The following diagram shows how the FM (RPI_HEADER), FSU and MISCCU are connected:

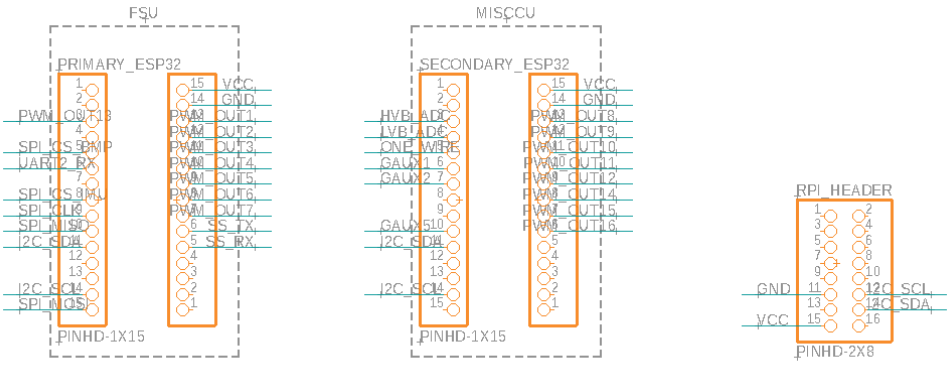

Figure 23: FM, FSU, MISCCU connection

The FSU is connected to servomotors of left aileron (PWM_OUT1), right aileron (PWM_OUT2), left flap (PWM_OUT3), right flap (PWM_OUT4), left elevator (PWM_OUT5), right elevator (PWM_OUT6), rudder (PWM_OUT7). It is also connected to BLDC motor ESC[5] (PWM_OUT13) which generates the correct 3-phase waveform for the motor. These pins are shown in the diagram below:

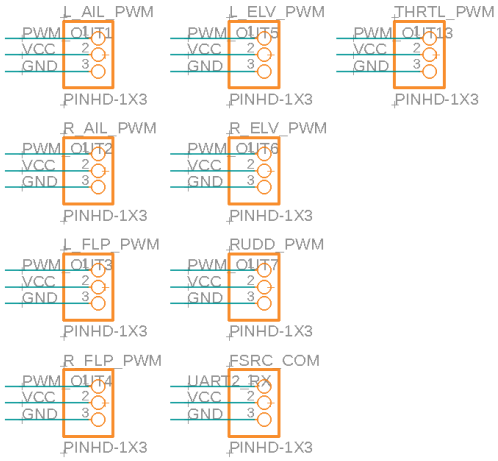

Figure 24: FSU PWM ports

[5] ESC = electronic speed controller

Furthermore, the FSU is connected over SPI to a 9DOF IMU, barometric pressure sensor and over UART to GPS module as shown in the diagram below:

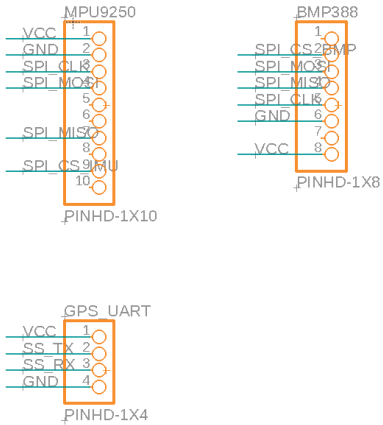

Figure 25: MPU, BMP, GPS connections

The FSU is also connected to a FSRC radio receiver with which it communicates over proprietary communication protocol iBUS. This protocol is used by the most RC radio receivers. It is connected over UART2_RX to the FSU (FSRC_COM pin shown on the bottom of the diagram above). The following transmitter is used for manual control of the aircraft.

Figure 26: Fly Sky-16X 2.4 GHz transmitter

The transmitter above comes with this iBUS enabled RC receiver:

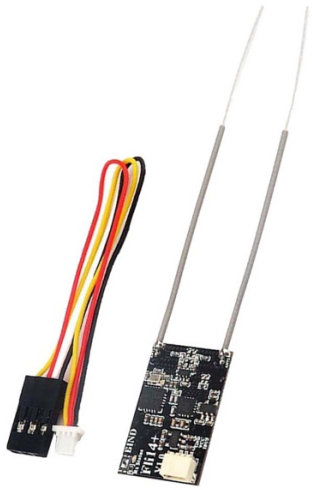

Figure 27: FS RC receiver

The MISCCU is connected to the landing gear retract motors control unit (RMCU) (over pin PWM_OUT12). These motors can rotate over a worm gear individual landing gear assembly so that the landing gear can move up and down. The RMCU has capabilities to already control the landing gear bay door servo motors but with the specific type of servo-motors used, it does not work reliably. For this reason, the RMCU only controls the retract motors and the bay door servo motors are controlled directly with the MISCCU (over pins PWM_OUT9, PWM_OUT10 and PWM_OUT11). Inside the nose landing gear assembly there is a steering servo motor (connected to MISCCU over PWM_OUT8), providing the steering capabilities of the nose landing gear. There are 3 extra auxiliary PWM enabled port available which could be used to connect additional servomotor or other actuators if needed. (Those are connected over pins PWM_OUT14, PWM_OUT15 and PWM_OUT16.) The following diagram shows the MISCCU PWM ports:

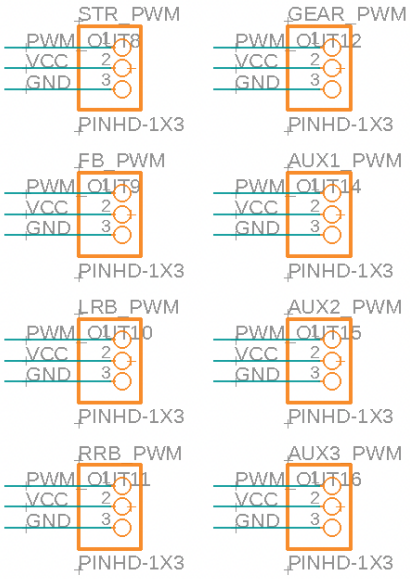

Figure 28: MISCCU PWM ports

The MISCCU is connected to both voltage divider to ensure power level measuring capabilities, it is also connected to temperature sensors (DS18B20) over 1-Wire protocol developed by Dallas Semiconductor Corp. Two extra auxiliary pins which can be used as input or output are added for futureproofing. Temperature sensor and extra pins are shown in the diagram below:

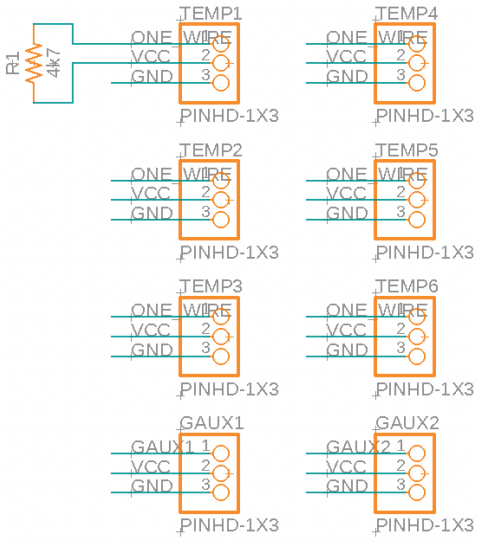

Figure 29: DS18B20 sensors connected over 1-Wire and extra pins

6.4 PCB layout

PCB layout design is also done in Fusion 360. The power supply and distribution section is placed in the top-center. The FSU (PRIMARY_ESP32) is located in the bottom left. Its PWM pins can be found to the left of the FSU and IMU, barometric sensor and FSRC receiver connection pins can be found to the right from the FSU in the bottom-center of the PCB. The GPS UART connection pins are above the FSU. The MISCCU and its corresponding PWM, 1-Wire temperature sensor and other pins are located in the top-left of the PCB. Below that, in the bottom-left of the PCB there are pins for connecting the FM over I2C and power delivery to the FM. The design of the whole PCB is shown in the diagram below:

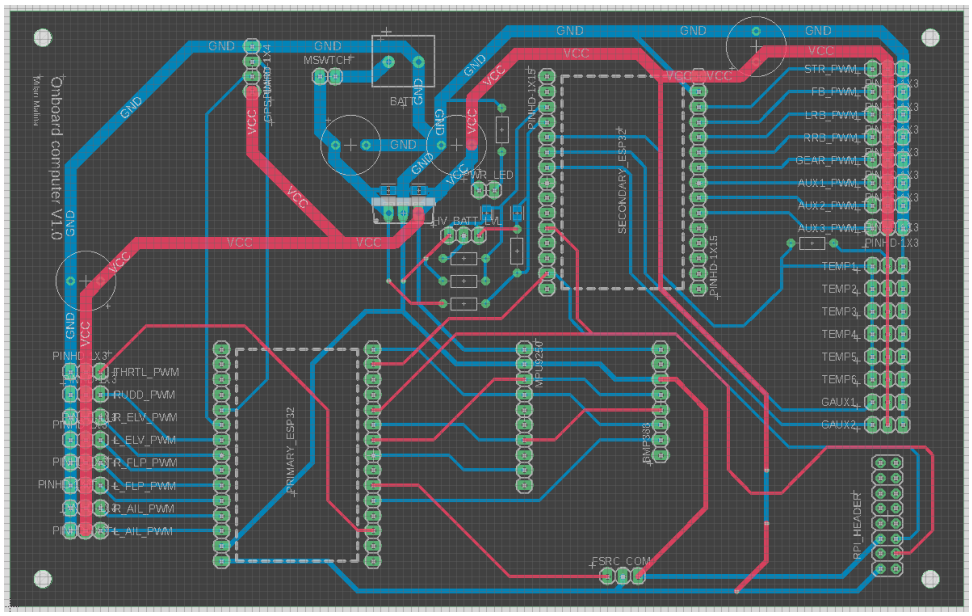

Figure 30: PCB layout design

Fusion 360 allows users to preview the designed PCB in a 3D view:

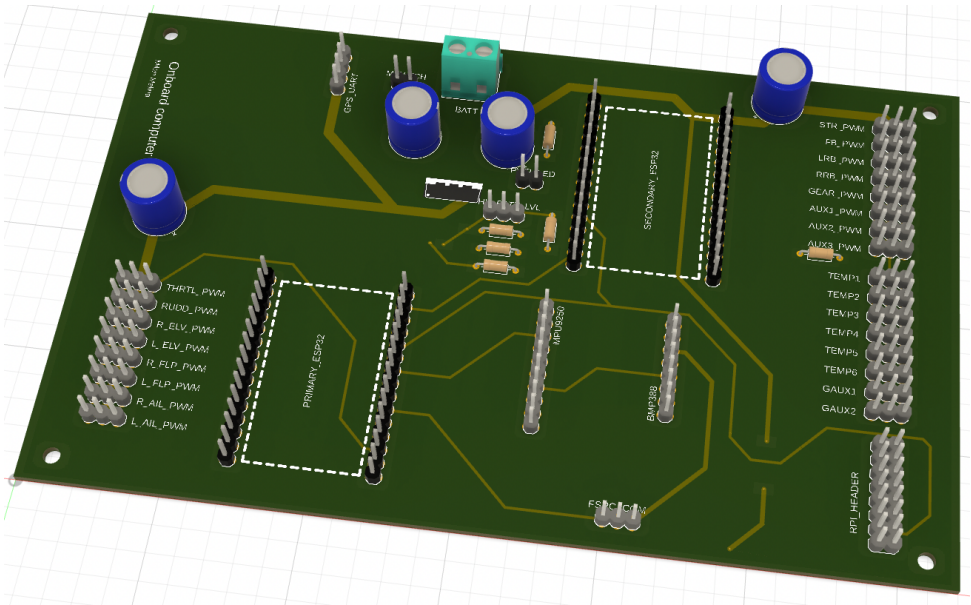

Figure 31: Fusion 360 generated 3D preview

Gerber files which are then supplied to the PCB manufacturer can be generated directly inside Fusion 360. After the manufacturing process and component assembly the real PCB is mounted into the modeled and 3D printed platform:

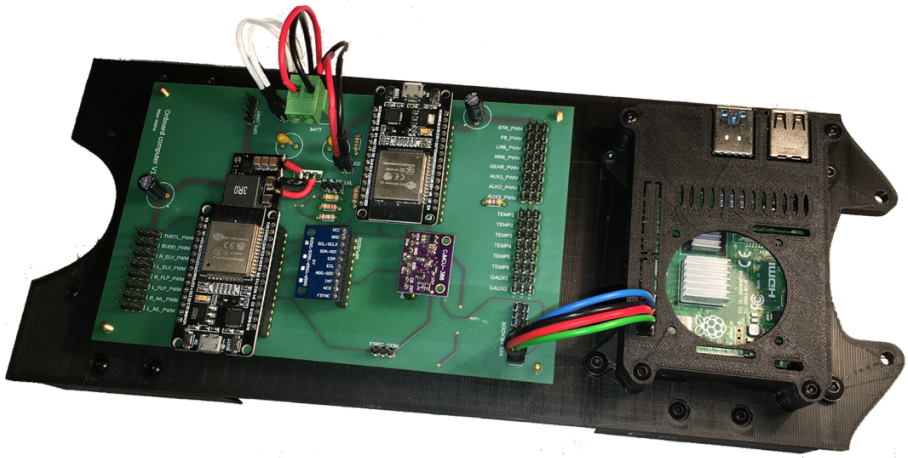

Figure 32: PCB mounted on a 3D printed platform

7 Attitude estimation using Extended Kalman Filter

7.1 Introduction

The purpose of this section is to develop a working implementation of an extended Kalman filter which uses quaternions for attitude estimation. Since this is a very complex problem with a possibility of other various improvements to be implemented (in order to suit a specific application), a more general solution will be discussed. This allows a broader applicability into various other system. Because the general idea of the algorithm does not change. Another goal of this section is to use affordable hardware and to still be able to achieve useful final results in a form of estimated attitude.

7.2 General structure

As stated above, this section's goal is both mathematical derivation and software implementation in a low-level programming language. The mathematical derivation is inspired by an online post [2]. This post was very helpful during the initial stages of the development. The mathematical derivation is discussed in more detail in upcoming sections.

The final implementation of the extended Kalman filter (EKF) is written in C++ and runs on ESP32 development kit. FreeRTOS implementation is used for better control over the structure and timing of the whole program. The algorithm run with a period of over 50 Hz when connected to a custom diagnostics tool (developed for the purpose of diagnostics and visualization in Unity 3D. This tool and its features are described in the chapters below.)

The inertial measurement unit used for the purpose of this project is MPU9250. It is a low cost 9-DOF IMU. This means it includes an accelerometer, a gyroscope and a magnetometer. The combination of those three sensors allows for pitch, roll and heading estimation.

Heading estimation is done (mainly - EKF combines all information) using the data from magnetometer. For the estimation to function

properly, the algorithm assumes that the acceleration vector and the vector of magnetic field points into two different directions (preferably 90 degrees apart). Since this is the case only at the equator, this assumption can cause problems with increasing (absolute) value of latitude. It is commonly known, that heading estimation around north and south pole is technically problematic.

7.3 Magnetometer calibration

In order to ensure optimal results, the magnetometer data supplied to the filter algorithm must be calibrated first. This process can be subdivided into two parts: hard iron and soft iron calibration. First, 2-dimensional analogy is shown, then the problem is generalized into 3 dimensions.

7.3.1 Hard-Iron error compensation

Hard-Iron error is caused by permanent magnets or other objects that produce permanent magnetic field and are physically attached to the reference frame of the magnetic field sensor itself. Hard-Iron errors manifest in a form of an offset in the readings provided by the sensor. If sensor with no error gives readings illustrated in the figure below (Figure 33), then the sensor with only hard-iron error would output data illustrated in the Figure 34.

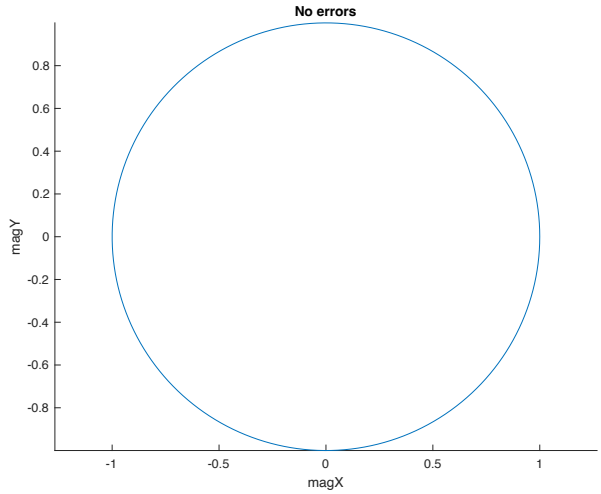

Figure 33: Illustration of a sensor data with no error

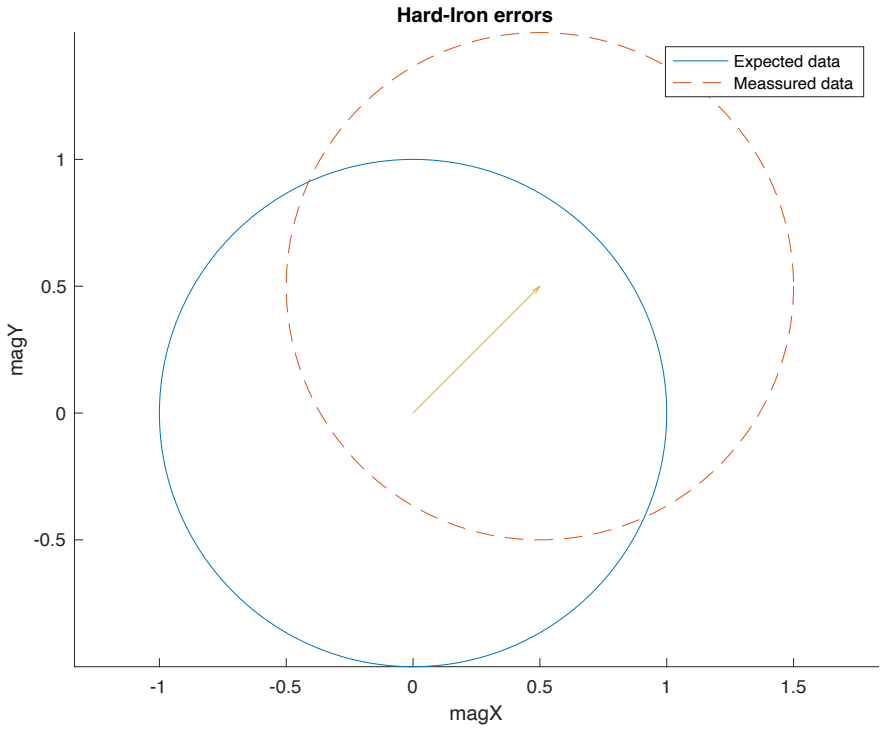

Figure 34: Illustration of a sensor data with hard-iron error only

7.3.2 Soft-Iron error compensation

This error is frequently caused by “soft” magnetic metals. Those are for example nickel-iron or iron-silicon alloys. This error manifests as twisting/stretching of the magnetic field vector. In two dimensions, it can be illustrated in Figure 35.

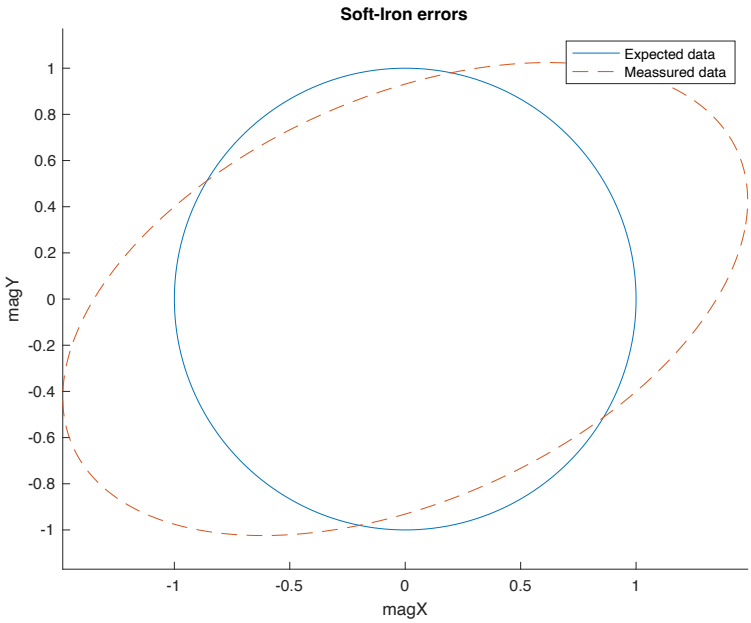

Figure 35: Illustration of a sensor data with a soft-iron error only

In general case, both, hard-iron and soft-iron errors are present which causes the measurement data to be offset and stretched/twisted. This, in 2 dimensions could be illustrated as follows:

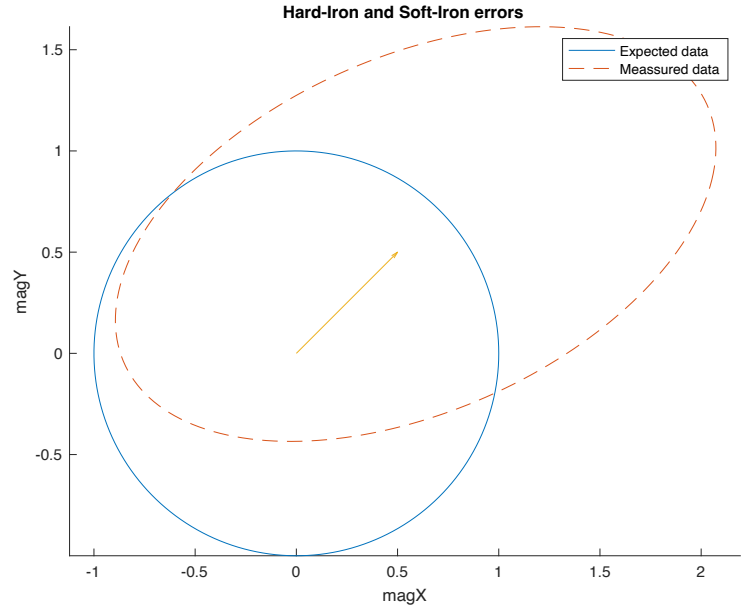

Figure 36: Illustration of a sensor data with hard-iron and soft-iron

In the illustrations above, the measurement data was represented in two dimensions using ellipses/circles.

7.3.3 3D generalization

In 3 dimensions this can be generalized using ellipsoids/spheres. The measured data makes up an ellipsoid which then needs to be transformed into a sphere.

This can be modelled as a mathematical transformation as follows:

$$\overrightarrow{m}_m = A\overrightarrow{m}_c + b \tag{1}$$

where $\overrightarrow{m}_m$ is the measured magnetic field vector, $\overrightarrow{m}_c$ is the correct (real) vector, pointing to the north (in ideal case, usually it points slightly to the ground, depending on where on Earth the observer is located). A is a calibration matrix (defining the stretching/twisting) and b is calibrations vector (specifying the offset). These can be determined by using a python script ("*magnetometer_calibration.py*"). This script has been obtained from the website mentioned in the introduction [2]. In short, the script gathers a specific amount of magnetometer measurements. Those measurements in a form of a point array form an ellipsoid:

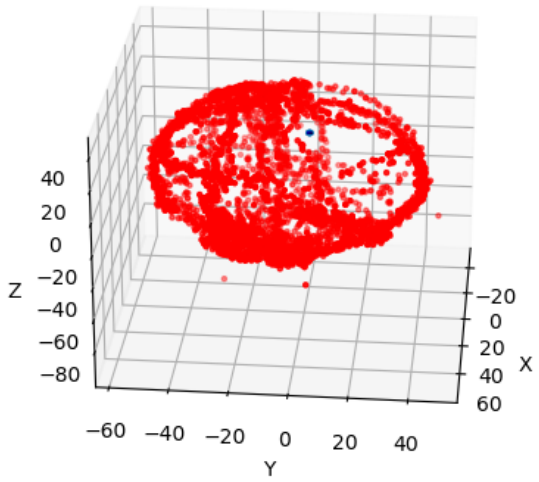

Figure 37: Measured magnetometer data before fitting

During the measurement, the IMU was rotated around each of its axis to fill the surface of the ellipsoid as densely as possible. It is also good to

notice the blue dot around the center of the points. This, in reality, is a unit sphere (in 2D analogy circle).

The python script attempts fitting this ellipsoid onto a sphere. This process outputs calibration parameters A^{-1} and b. Those can then be finally used to transform the measured data $\overrightarrow{m}_m$ to the correct magnetometer output data $\overrightarrow{m}_c$ using the following formula:

$$\overrightarrow{m}_c = A^{-1}(\overrightarrow{m}_m - b) \tag{2}$$

The transformed points can be viewed in the following graph (the small "dot" from the Figure 37 is displayed in an adequate size looks like a sphere, when different axis scale is taken into account):

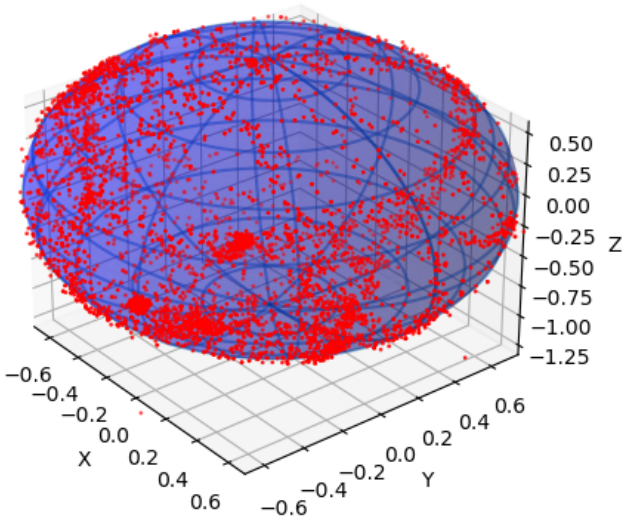

Figure 38: Transformed magnetometer data

Data transformed in such a way can finally be used as an input to the EKF.

7.4 Reference vectors acquisition

In order for the Kalman filter algorithm to work properly, acceleration reference vector and magnetic field reference vector must be calculated.

This is simply done by measuring a set of data from accelerometer and magnetometer and averaging them.

During this phase, the IMU must be stationary, heading to north, with 0 degrees in both pitch and roll. Next, the averaged vectors are normalized. This is automated in the Python script (named "*sensor_stat_analysis.py*").

Before running the script, it is advised to doublecheck the variables mag_Ainv, mag_b which are obtained during magnetometer calibration and variable $DATA\_COUNT$, which specifies how big the data set is (value of 200 is sufficient).

Magnetic field measurements are first calibrated using mag_Ainv, mag_b, before they are used for the statistical analysis. Apart from that, this script calculates the expected values and variances for each sensor. Also, this script outputs the average acceleration magnitude, which is then also used in the Kalman filter algorithm initialization.

7.5 Extended Kalman filter algorithm

7.5.1 Introduction

Extended Kalman Filter (EKF) algorithm works in two distinctive steps. Those are prediction phase, also known as "Time update" and update phase, also known as "Measurement update". Those two steps are run in each time step. Before the algorithm can be first started initialization must be completed. In the following chapter, the EKF algorithm for quaternion attitude estimation is derived.

7.5.2 EKF algorithm derivation

Let's first define the state vector. The state vector Equation (3) has 7 elements. 4 of which are used to describe quaternion defining the orientation and 3 elements are used for estimating drift of the gyroscope. The gyro drift is automatically adjusted by the EKF to provide the optimal estimation. The state vector has the following form:

$$x = [q_0 \quad q_1 \quad q_2 \quad q_3 \quad g_{b1} \quad g_{b2} \quad g_{b3}] \tag{3}$$

Next, the system dynamics equation is needed in order to calculate the prediction estimate from a previously estimated state vector. This is done using the following equation:

$$\dot{q} = \frac{1}{2} S(\omega) q = \frac{1}{2} S(q) \omega \tag{4}$$

Where $S(q)$ is the current estimated attitude in form of matrix:

$$S(q) = \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix} \tag{5}$$

and ω is the angular rate provided by the gyroscopic sensor. The math behind the Equation (4) is beyond the scope of this section but is very comprehensively explained in this post [3]. The only variable left to update is the gyro drift. This is solved simply by using the previous value. In order to take the drift of the gyro into account, Equation (4) is adjusted as follows:

$$\dot{q} = \frac{1}{2} S(\omega - g_b) q = \frac{1}{2} S(q) \omega - \frac{1}{2} S(q) g_b \tag{6}$$

The overall equation to calculate the predicted state vector is following:

$$x_{k+1}^{'} = \begin{bmatrix} I_{4x4} & -\frac{T}{2} S(q) \\ O_{3x3} & I_{3x3} \end{bmatrix} x_k + \begin{bmatrix} \frac{T}{2} S(q) \\ O_{3x3} \end{bmatrix} \omega_k \tag{7}$$

This equation is used in the EKF algorithm during prediction phase to calculate the next predicted state vector from the previous one. It is in the form of:

$$\boxed{x_k = A_{k-1} x_{k-1} + B_{k-1} \omega} \tag{8}$$

Where $A_k = \begin{bmatrix} I_{4x4} & -\frac{T}{2} S(q) \\ O_{3x3} & I_{3x3} \end{bmatrix}$ and $B_k = \begin{bmatrix} \frac{T}{2} S(q) \\ O_{3x3} \end{bmatrix}$,
$q = [x_0 \quad x_1 \quad x_2 \quad x_3]|_k$ (latest estimated attitude is used to evaluate $S(q)$.) To clarify, both are A_k and B_k are, of course, time variant. A_k and B_k are also used when calculating predicted covariance matric. This can be done in a usual form:

$$\boxed{\acute{P}_k = A_{k-1} P_{k-1} A_{k-1}^T + Q} \tag{9}$$

Where Q is the process covariance matrix. Values for Q, P_0 and x_0 (initial) are discussed in the next chapter.

Next, lets derive the equation used in the update phase. It is needed to find a connection between the measured data by accelerometer and

magnetometer and the state vector. This can be achieved using the following equation:

$$d_m = R(q)v^s_{ref} + E \tag{10}$$

where d_m is the measured data given by the sensor and

$$R(q) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_1q_3) & 2(q_2q_3 - q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

is rotational matrix constructed using the estimated quaternion, v_{ref} is a reference vector for the corresponding sensor. Its acquisition process is closely described in Chapter 5, E is the error of the sensor and the prediction composed of a bias (if sensor is uncalibrated) and a noise. $y^{\acute{}s}_k = R(q)v^s_{ref}$ is the predicted sensor output. In ideal case this should correspond to the value given by the sensor. When the IMU is placed with attitude of 0 pitch, 0 roll and heading to north, the data outputted by the sensor must be essentially the same as the v_{ref}, which also explains the v_{ref} acquisition process and can be used during debugging. Equation (10) is in a form of: $y = C^s x + D$, in order to apply Kalman filter algorithm, this equation must be linear. Given the nature of quaternions, this equation is strongly non-linear. Taylor expansion is used to approximate this non-linear equation with a linear one, around the operational point, which in this case is the current attitude estimate (from the previous timestep). For the purpose of EKF algorithm, only C^s term is requiered, therefore D does not need to be calculated. The matrix C^s is time dependent and can be calculated as follows:

$$C^s_k = \frac{\partial}{\partial q}(R(q_{k-1})v_{ref}) \tag{11}$$

After multiplication and finding the derivative (Jacobian), The matrix C^s can be expressed as[6]:

[6] v^i_{ref} signifies i-th element of the v_{ref} vector as opposed to v^s_{ref} which signifies the whole reference vector of the sensor s.

$$
\begin{aligned}
C_k^{s[0,0]} &= q_{k-1}^0 v_{ref}^0 + q_{k-1}^3 v_{ref}^1 - q_{k-1}^2 v_{ref}^2 \\
C_k^{s[0,1]} &= q_{k-1}^1 v_{ref}^0 + q_{k-1}^2 v_{ref}^1 + q_{k-1}^3 v_{ref}^2 \\
C_k^{s[0,2]} &= -q_{k-1}^2 v_{ref}^0 + q_{k-1}^1 v_{ref}^1 - q_{k-1}^0 v_{ref}^2 \\
C_k^{[s0,3]} &= -q_{k-1}^3 v_{ref}^0 + q_{k-1}^0 v_{ref}^1 + q_{k-1}^1 v_{ref}^2 \\
C_k^{s[1,0]} &= -q_{k-1}^3 v_{ref}^0 + q_{k-1}^0 v_{ref}^1 + q_{k-1}^1 v_{ref}^2 \\
C_k^{s[1,1]} &= q_{k-1}^2 v_{ref}^0 - q_{k-1}^1 v_{ref}^1 + q_{k-1}^0 v_{ref}^2 \\
C_k^{s[1,2]} &= q_{k-1}^1 v_{ref}^0 + q_{k-1}^2 v_{ref}^1 + q_{k-1}^3 v_{ref}^2 \\
C_k^{s[1,3]} &= -q_{k-1}^0 v_{ref}^0 + q_{k-1}^3 v_{ref}^1 + q_{k-1}^2 v_{ref}^2 \\
C_k^{s[2,0]} &= q_{k-1}^2 v_{ref}^0 - q_{k-1}^1 v_{ref}^1 + q_{k-1}^0 v_{ref}^2 \\
C_k^{s[2,1]} &= q_{k-1}^3 v_{ref}^0 - q_{k-1}^0 v_{ref}^1 - q_{k-1}^1 v_{ref}^2 \\
C_k^{s[2,2]} &= q_{k-1}^0 v_{ref}^0 + q_{k-1}^3 v_{ref}^1 - q_{k-1}^2 v_{ref}^2 \\
C_k^{s[2,3]} &= q_{k-1}^1 v_{ref}^0 + q_{k-1}^2 v_{ref}^1 + q_{k-1}^3 v_{ref}^2
\end{aligned}
\tag{12}
$$

$$
C_k^s = 2 \begin{bmatrix} C_k^{[0,0]} & C_k^{[0,1]} & C_k^{[0,2]} & C_k^{[0,3]} \\ C_k^{[1,0]} & C_k^{[1,1]} & C_k^{[1,2]} & C_k^{[1,3]} \\ C_k^{[2,0]} & C_k^{[2,1]} & C_k^{[2,2]} & C_k^{[2,3]} \end{bmatrix}
$$

The final matrix C for the Kalman filter can be constructed from previously calculates matrices C^s as follows:

$$
C_k = \begin{bmatrix} C_k^{acc} & O_{3x3} \\ C_k^{mag} & O_{3x3} \end{bmatrix} \tag{13}
$$

O_{3x3} matrices signify independency of the measured data on the bias of the gyro.

7.5.3 Initialization

During the initial phase algorithm variables are initialized so that EKF can then run. The initial values for variables used by EKF are in the following table:

Table 4: EKF implementation variable names

Variable name	Initial value
x_hat	[1 0 0 0 0 0 0]
accel_reference	FROM PY
mag_reference	FROM PY
gravity_constant	FROM PY

accelerometer_prec_attenuation	APRX 10, AS REQ
P	APRX 0.01, AS REQ[7]
Q	APRX 0.001, AS REQ[1]
R	APRX 1, AS REQ[1]
mag_calib_A_inv	FROM PY
mag_calib_b	FROM PY

The initial value for the state vector is $x_{k=0} = [1 \quad 0 \quad \mathbf{0} \quad \mathbf{0} \quad 0 \quad 0]$ corresponding to no deviation in attitude and no gyro bias. Acceleration and magnetic field reference vectors and gravitational constant are calculated by the python script as described in Chapter 5. Accelerometer precision attenuation constant is used to artificially (and drastically) increase the measurement covariance matrix. This is done in order to account for acceleration caused by change in linear velocity (not gravitational force). Given the Equivalence principle, these two causes for perceived acceleration (gravitation and change in linear velocity) are fundamentally indistinguishable. It can be assumed, that if the magnitude of the acceleration vector differs to a higher degree from the acceleration vector, then change in velocity is present and is introducing error into the update phase by twisting the acceleration vector. This effect is negated by drastically increasing the accelerometer covariance matrix and in such way letting the filter know not to momentarily trust the accelerometer measurement. P is the covariance matrix.

Trace of this matrix can be used to determine the precision of the estimate at a given time. The lower the initial value (values in this case refers to values of the diagonal elements, non-diagonal elements are set to 0) is, the longer it will take for the filter to start trusting the measurement resulting in higher initial delay before an accurate attitude estimate can be obtained. Q is the process noise matrix. This matrix specifies the uncertainty caused during the prediction phase when using the dynamic model to calculate the prediction of the state vector estimate. Increasing the value of the diagonal elements of this matrix will result in a greater overall covariance and the algorithm giving higher importance to the measured data in the update phase (acceleration and magnetic field vector). On the other hand, decreasing values of the diagonal elements will

[7] For matrices, the actual value is the scalar value V multiplied by $eye(V)$ - the diagonal matrix with value V at the main diagonal

result in EKF trusting the dynamic model in the prediction phase more as opposed to the measurements in the update phase. Matrix R specifies the measurement noise and has essentially the opposite effect as matrix Q. Diagonal values of this matrix can be calculated using a python script, but that requires adequate values of the matrix R. Instead, it is recommended to use the date from the table above as a base guess and then fine-tune the values for specific characteristics of the IMU module used. Values of calibration constants $A^{-1}_{mag\_calib}$ and b_{mag_calib} are calculated using a python script as described above in Chapter 4.

7.5.4 Prediction phase

During the prediction phase, EKF receives data from the gyroscope in a form of angular velocity in rads/sec and combining this data with the information about the system dynamics calculates prediction of the state vector for the following timestep. Prediction of the state vector and covariance matrix are obtained by utilizing the following equation:

$$x_{k+1}^{'} = \begin{bmatrix} I_{4x4} & -\frac{T}{2}S(q) \\ O_{3x3} & I_{3x3} \end{bmatrix} x_k + \begin{bmatrix} \frac{T}{2}S(q) \\ O_{3x3} \end{bmatrix} \omega_k \tag{14}$$

and:

$$\acute{P}_k = \begin{bmatrix} I_{4x4} & -\frac{T}{2}S(q) \\ O_{3x3} & I_{3x3} \end{bmatrix} P_{k-1} \begin{bmatrix} I_{4x4} & -\frac{T}{2}S(q) \\ O_{3x3} & I_{3x3} \end{bmatrix}^T_{k-1} + Q \tag{15}$$

As described above, q is the currently most precise estimate of the attitude calculated during the previous update phase. During this phase, predicted sensor output and matrix C is calculated as described in the previous section.

7.5.5 Update phase

During the update phase, measurement from accelerometer and magnetometer are normalized (and transformed using the calibration values), the magnitude of the acceleration vector is determined (to compensate for change in linear velocity) and lastly, the sensor data is used to update-correct the prediction obtained in the prediction phase by weighting the

information provided by the prediction and the sensor measurement using Kalman gain K. This gain is calculated as follows:

$$K_k = \frac{\acute{P}_k C_k^T}{C_k \acute{P}_k C_k^T + r_{mult} R} \tag{16}$$

where r_{mult} is accelerometer covariance multiplier accelerometer and is calculated using the following formula:

$$r_{mult} = 10^{abs(g_{const} - \|a\|) a_{prec\_att}} \tag{17}$$

where a_{prec_att} is precision attenuation constant and g_{const} is gravity constant calculated by a python script. To ensure numerical stability, r_{mult} is clamped between 1 and 1000.

Kalman gain is then used to calculate the new estimate of the state vector and covariance matrix:

$$x_{k+1} = \acute{x_{k+1}} + K_k(m - \acute{y_k}) \tag{18}$$

where m is the concatenated measurement vector $m = [m_a \quad m_m]^T$ and $\acute{y_k} = [\acute{y_k^a} \quad \acute{y_k^m}]$. $\acute{y_k^a}$ and $\acute{y_k^m}$ are accelerometer and magnetometer output predictions.

Covariance matrix is calculated using following formula:

$$P_k = (I - K_k C_k)\acute{P}_k \tag{19}$$

At the end quaternion segment of the state vector is normalized.

7.6 EKF testing data acquisition

In order to debug and further experiment with different parameters and features, testing data acquisition application has been developed. It is meant as a developer and testing tool only, so it runs directly in the editor of Unity 3D engine (scene named "DataGenerator"). Scripts are written in C#. To properly use this tool, some prior experience with Unity 3D is required.

In order to generate testing data, two different options are available. First option (1) is to programmatically specify sequence of attitudes through which the application automatically switches and at the end of the sequence, the play mode is exited. In the file "ManualMover.cs" one can specify this sequence including timing. (Do-Tween plugin is used for implementation.) This script can also be used to specify a linear acceleration for testing purposes. The second option (2) is to use Unity Editor to move the game-object manually using a mouse and Unity gizmos.

Virtual data synthesis is performed by the "Virtual 9DOFMPU.cs" script. For this component to function properly, its attributes must be setup correctly. The most important are reference vectors, gravity constant and sensor variances. It is crucial to consider different coordinate system which Unity uses. Specifying these attributes mistakenly due to human error when manually performing coordinate transform can be source of a possible errors and can be difficult to debug. For this reason, a simple tooltip has been added to be shown (when mouse is moved over the attribute name.)

Scripts named "TransformDataSaver.cs" and "SensorDataSaver.cs" are used to store the real transform and the emulated sensor values into a CSV file. This data can be then used for EKF test in python.

7.7 EKF test with emulated data

Testing data is acquired utilizing Unity tool described in the previous chapter. Testing data acquisition 2 is used resulting in less smooth and more natural movement. Simulated sensor data generated during programmatically specified movement can be viewed in the figure below:

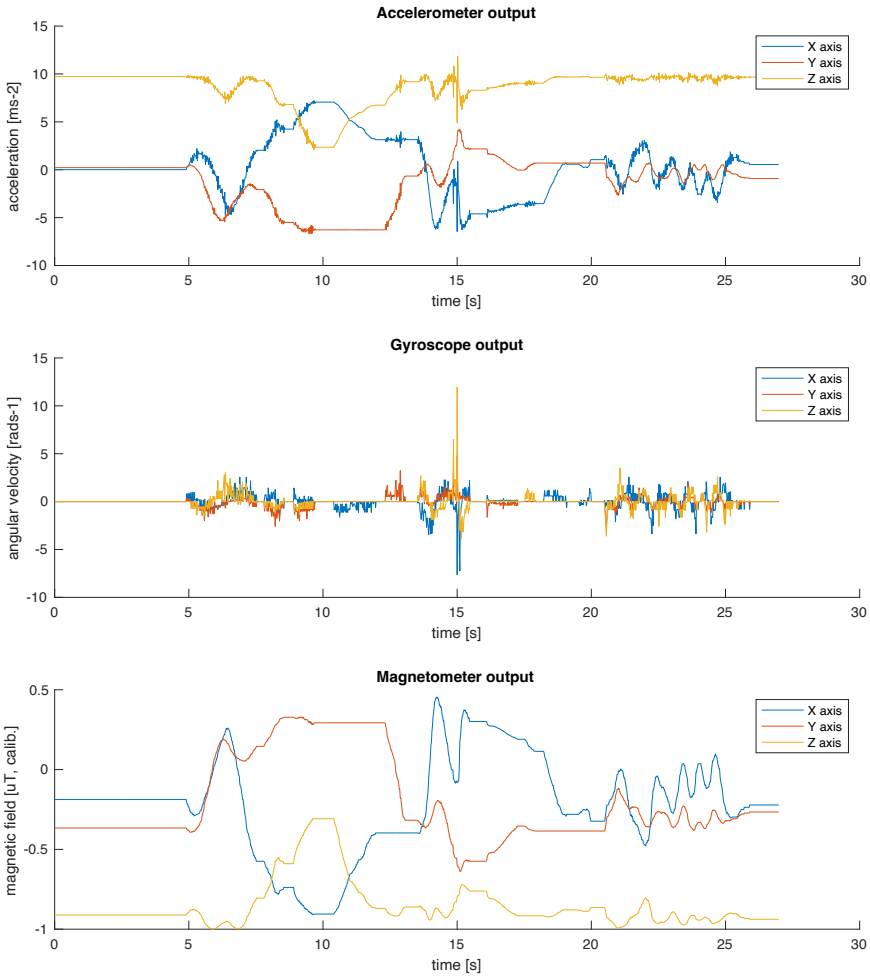

Figure 39: Sensor data during programmatically specified movement

While generating this data, there were no explicit attempts to cause linear acceleration. This can be seen in the first subplot, where the acceleration (per axis) is barely reaches gravitational acceleration constant.

In the following figure, the virtual IMU was moved very sporadically in a linear motion, causing a drastic linear acceleration:

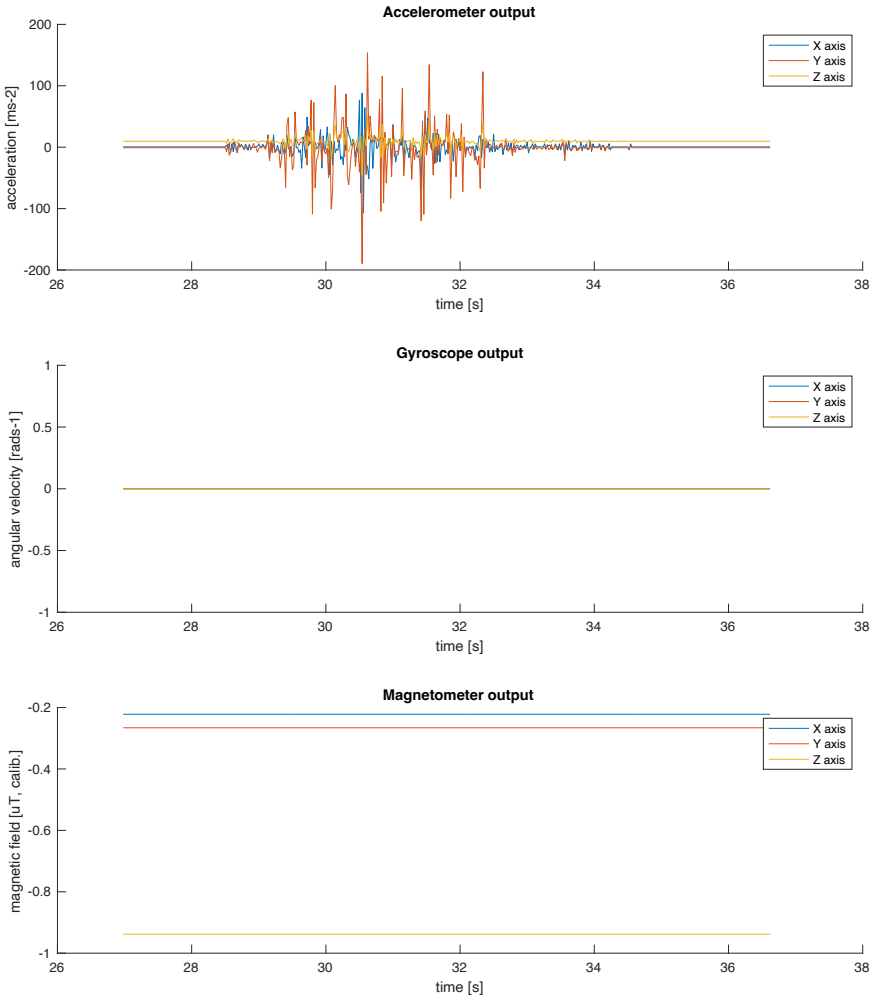

Figure 40: Sensor data while IMU under drastic linear acceleration

Since the virtual IMU was not experiencing any angular velocity, the data gathered from the gyroscope and the magnetometer is constant. The accelerometer data shows very high values of acceleration (and deceleration) caused by the sporadic movement.

Finally, it the EKF algorithm can be executed provided the generated data. The first figure shows EKF algorithm output during the first phase of the experiment (when IMU wasn't experiencing any extreme acceleration):

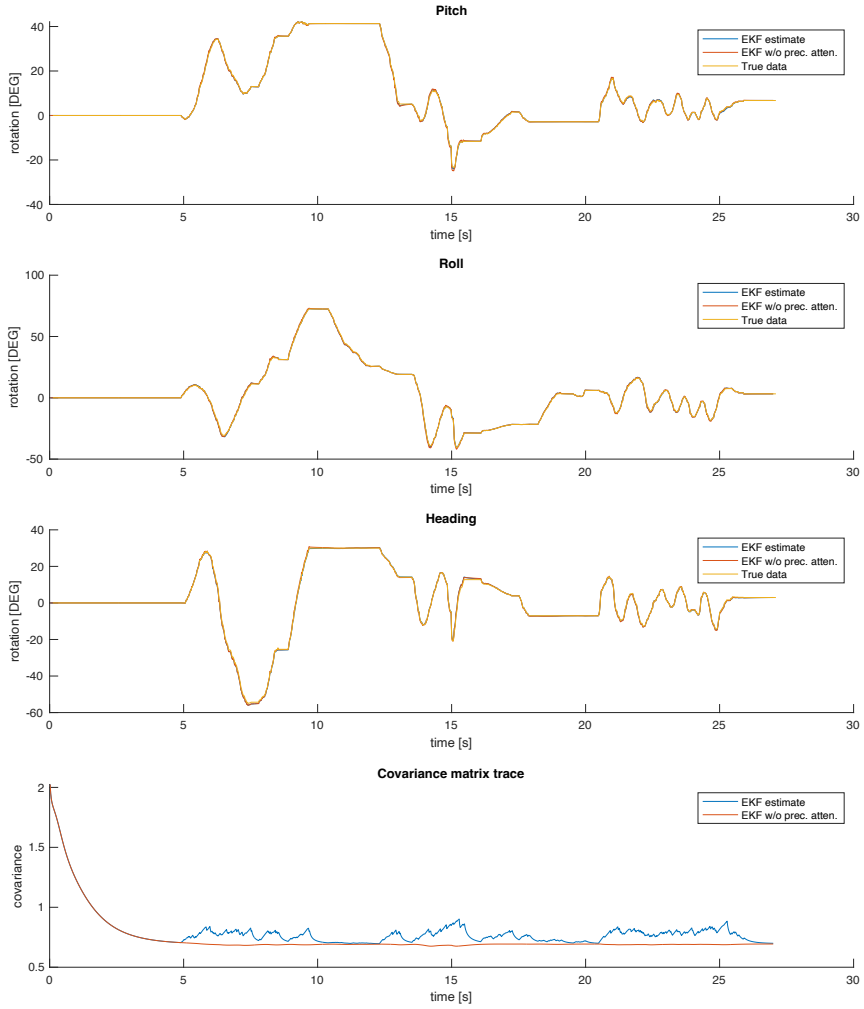

Figure 41: EKF output with no drastic linear acceleration

It is visible that both EKF variants (with and without accelerometer measurement covariance matrix adjustment) perform in a similar way. This only visible different is in the estimation covariance matrix trace plot. While the variant without precision attenuation follows the normal trend of the trace reaching a stable steady state value and (for the most part - caused by non-linearities) keeping it, the variant with the attenuation adjustment is keeping the trace value slightly higher. While in this case it is mainly caused by numeric properties of the selected multiplier function, it still shows the "hesitance" of the algorithm to process and "trust" any acceleration value which could be slightly higher. But as

mentioned above, this effect is practically indistinguishable in the output data itself (pitch, roll, heading).

Though, this does not apply when the IMU moves more sporadically. This is illustrated in the following figure:

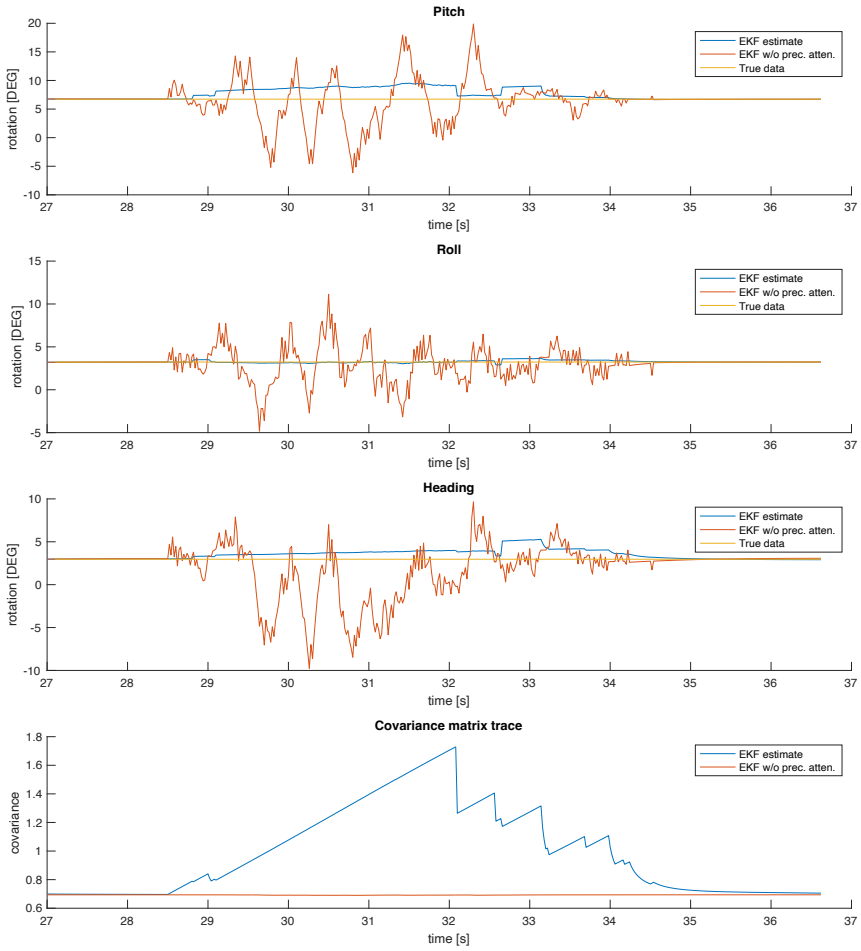

Figure 42: EKF output with drastic linear acceleration

From this figure it is visible the difference between the two variants on the algorithm. The estimate provided by the variant without the attenuation feature is significantly worse. The theoretical covariance trace is lower, this is caused by incorrect a priory expectation of the algorithm. If a real value of covariance trace were to be calculated, it would be higher.

The covariance trace of the variant with attenuation significantly increases, which results in the algorithm's lower "belief" in the

accelerometer measurement and so this measurement is taken into consideration significantly lower (by automatic adjustment of the Kalman gain). This behavior is very useful and even a necessity for application in UAVs where drastic acceleration and deceleration is a frequently present.

7.8 EKF C++ real-time implementation

The final implementation of the EKF algorithm is meant to run on ESP32 board together with the autonomous controller and other critical software components. It is important to ensure execution of the most critical tasks in time. For this reason, RTOS is utilized to optimally schedule tasks based on their priority, specifically a free-ware implementation FreeRTOS. This is one of the reasons why the programming language of the final EKF implementation is C++.

In order to perform matrix operations required for EKF algorithm, Linear Algebra library [4] is used. This library is fully compatible with ESP32 MCU. For previous testing with python code, UART is used to send the data (gathered by the IMU over SPI). With the C++ implementation the data is processed directly on the MCU.

7.9 EKF diagnostics tool

In order to display the data computed by the C++ implementation of the EKF, a proper diagnostic tool is needed. This is implemented using the Unity 3D engine again but now Unity is used to visualize the data received by the ESP32 over the UART.

Due to the dynamic nature of this test, few different scenarios were physically executed: Z-axis rotation on a stable turntable, sporadic movement and linear acceleration followed by a deceleration while attempting to keep still in attitude.

The following figure shows data from the first scenario:

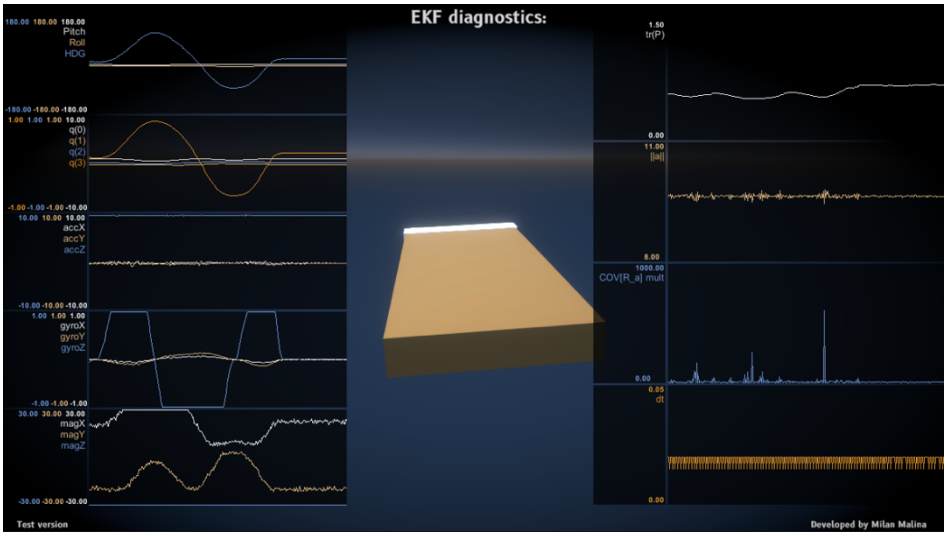

Figure 43: First scenario data

The first plot on the left shows the estimated attitude in degrees. As expected in this scenario, only heading is being changed from about 0 degrees to +90 degrees then to -90 degrees and then back to 0 degrees. This corresponds with the physical movement the IMU experienced. The second plot on the left shows the 4 elements of the quaternion. Similar curve as shown in the plot above can also be observed here, although any more exact interpretation is complex due to the nature of quaternions. The third plot shows the acceleration measured by the IMU. In this scenario this plot seemingly does not change, since the IMU is being rotated only around the axis which has practically the same direction as the gravitational acceleration. The fourth plot shows the angular velocity measured by the gyroscope. This value also changes as expected (mostly rotation around Z-axis). The saturation is only visual in the plot. There is a y limit set up so that smaller values are also distinguishable. The fifth plot from the left shows the magnetometer data. Due to specific direction (NOT always 90 degrees perpendicular to gravitational acceleration) of magnetic reference vector this data has more complex behavior.

The first plot on the right shows the trace of the covariance matrix. This value can indicate how the EKF is "secure" about its estimate. The second plot on the right shows the magnitude of the acceleration vector. This is used to account for linear acceleration. In this case there was no significant linear acceleration, therefore the magnitude is around $g_{constant}$. The third plot on the right shows the current r_{mult} value. Since the acceleration magnitude is around $g_{constant}$, the r_{mult} is for the most part

close to 1. The last plot on the right shows the time period with which the algorithm runs. This does not exceed 0.02s. This equivalates to execution frequency of 50Hz or more. Which for the final application is sufficient. It is also worth noting, that portion of this time is taken by the UART communication. The approximate estimate is around 30% but no testing has been performed regarding that.

The following figure shows data from the first scenario:

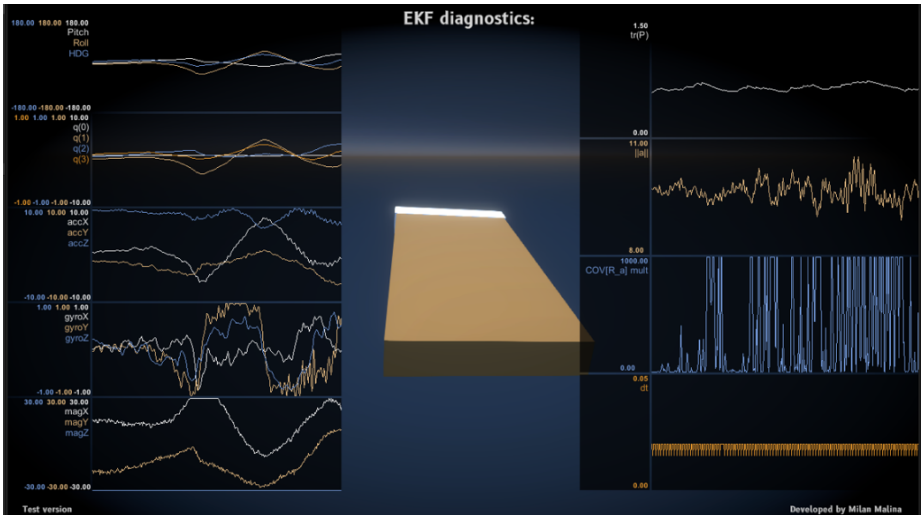

Figure 44: Second scenario data

This scenario was executed more for illustrational purposes. It can be observed that the acceleration magnitude was significantly higher and more sporadic, which can anyways be said about of the data from this scenario. The data from the third scenario can be seen in the figure below:

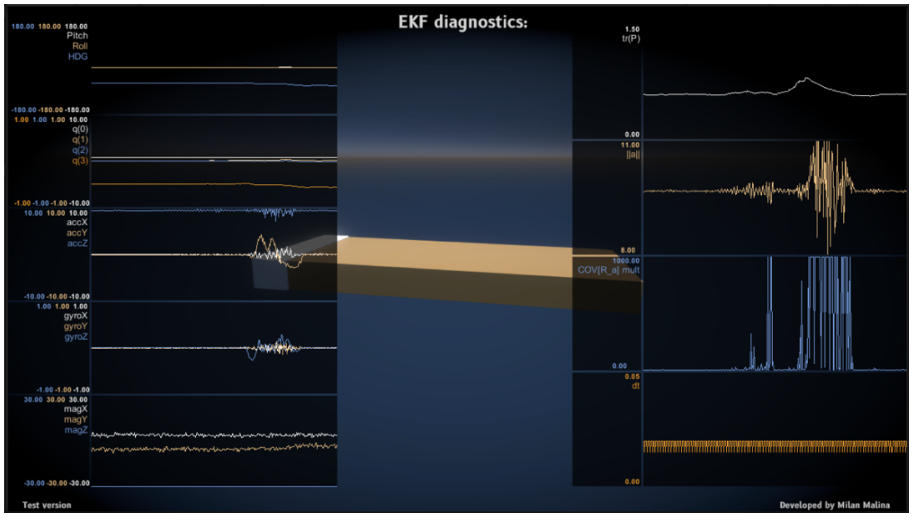

Figure 45: Third scenario data

In this scenario, the IMU was first accelerated and then decelerated with as low change in attitude as possible. This can be observed by change in acceleration alongside the Y-axis. Above all, the magnitude of the acceleration vector dramatically rises. This causes the r_{mult} to dramatically rise as well. This results in the EKF algorithm essentially ignoring the corrupted accelerometer reading. Since the EKF is unable to use the accelerometer reading, the covariance matrix trace automatically increases, signifying higher uncertainty in the estimated attitude. During the experiment, the estimated attitude is practically unchanged, which corresponds with the actual physical movement.

A YouTube video showing the behavior in more dynamic way can be accessed at: youtu.be/qy9yPrqckdY on my channel. It shows a practical demonstration of the EKF. In the demonstration, IMU is being moved by hand. First, simple movement is tested, followed by more sporadic and complex movement. Response to a linear acceleration and deceleration is successfully tested as well.

7.10 Conclusion

The purpose of this section was to implement and test the functionality and applicability (to UAV attitude estimation) of Extended Kalman filter algorithm. Algorithm was first tested with simulated data and then was implemented in C++ on ESP32 microcontroller unit. Based on the simulation and real-world results and availability and cost of the hardware used, the implementation appears to be practical for smaller, experimental unmanned aerial vehicles. Possible issues could arise from high noise presence in the magnetometer readings. This could be amplified in case of high electrical currents flow near the IMU. This could be solved in the design phase of a UAV by physically distancing the high current components of the avionics as far from the IMU as possible or by adjusting the EKF algorithm - modelling the electrical currents in order to for the EKF to be able to predict the change in magnetic field and account for it.

8 Aircraft control interface and telemetry visualization

8.1 Introduction

The purpose of the application is to display flight telemetry data and control the aircraft. Communication with Flight Manager (RPi4) is established via LTE. The user is allowed to control various flight parameters (pitch, roll, speed, etc.) and read telemetry data in real time. The application is developed in the Unity3D Engine and the code is written in C#. It composes of different panels, each providing information about different systems of the aircraft. Every panel, if applicable, contains a status overview which provides an over information about the corresponding system.

The visual aspect of the user interface is inspired by flight displays implemented in Airbus aircrafts. Similar to real aircrafts this application features a Master Caution System (MCS) including audio-visual alarm.

The figure below shows the graphical user interface which will be discussed in more detail in the following chapters:

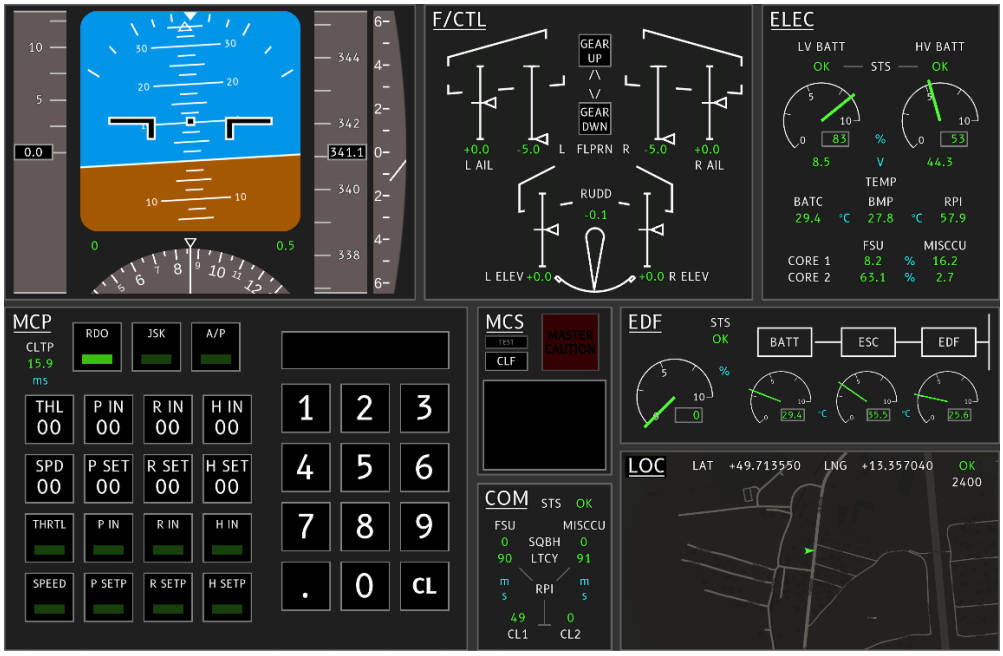

Figure 46: GUI overview

8.2 Primary Flight Display (PFD)

The PFD shows the basic flight data, such as speed (measured by the GPS sensor, data like TAS, IAS are not possible to obtain due to the absence of more accurate and industry specific components like pitot-tubes, etc.). This panel composes of an artificial horizon (1), sped indicator (in knots) (2), altitude indicator (in meters) (3), vertical speed indicator (in feet-per-minute) (4) and heading indicator (5). This panel also includes numerical information about the EKF estimate accuracy (6) and speed converted to kilometers per hour (7).

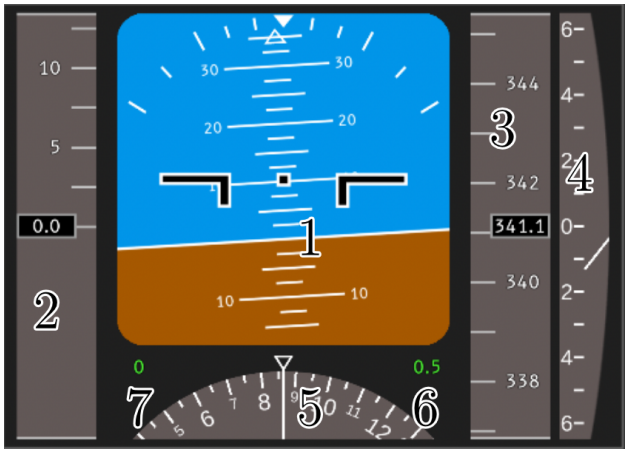

Figure 47: PFD overview

8.3 Flaps and Control Surface (F/CTL)

The F/CTL panel provides user the information about the status of the control surfaces and flaps (deflection in degrees). It also provides control of the landing gear.

The following figure describes the individual elements of this panel: left aileron (1), left flap (2), right flap (3), right aileron (4), left elevator (5), right elevator (6), rudder (7) and gear control (8).

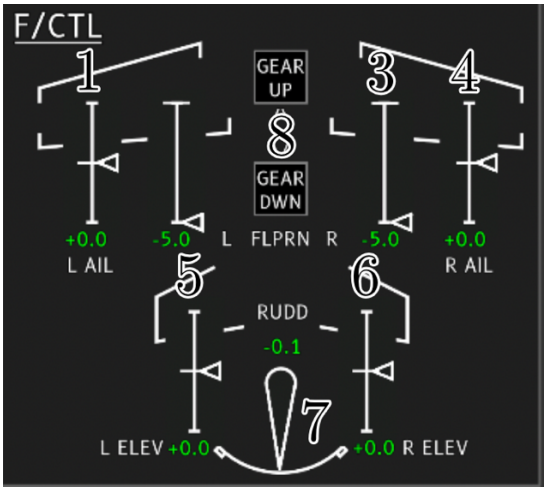

Figure 48: F/CTL overview

8.4 Electrical (ELEC)

The ELEC panel displays the most important information the electrical status of the aircraft. It also provides the user with temperature readouts in different critical areas of the aircraft. It is integrated with the MCS to alarm the operator in case of critical situation.

The following figure describes the individual elements of this panel: LV and HV battery lever and voltage indicators (1), battery compartment, BMP sensor and Flight Manager (RPI4) temperature readings (2) and usages of FSU and MISCCU both cores (3).

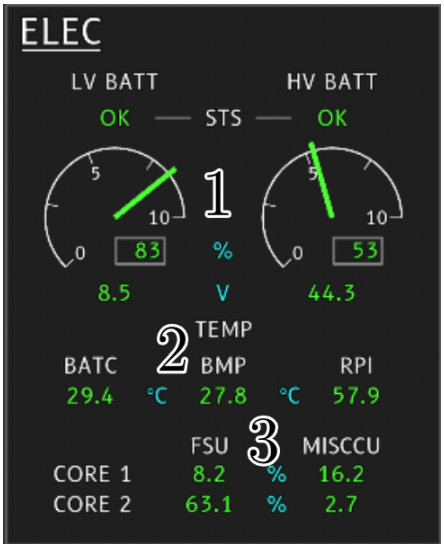

Figure 49: ELEC overview

8.5 Mode Control Panel (MCP)

The MCP is used to control the modes of the aircraft, mainly the autopilot. It contains two different types of custom interactable elements: Remote Button Indicators (RBIs) and Remote Value Fields (RVFs). RBIs are used to enable or disable the radio/autopilot (1) and set different mode of the autopilot (4) (by enabling/disabling corresponding controllers). RVFs are used to set either specific input to corresponding control axis (2) (pitch, roll, etc.) or setpoint of a corresponding controller (3). In order to set a value to a RVF this value must be first specified using keypad and then can be set to a specific RFV by pressing it (RVF).

The following figure displays the layout of the MCP:

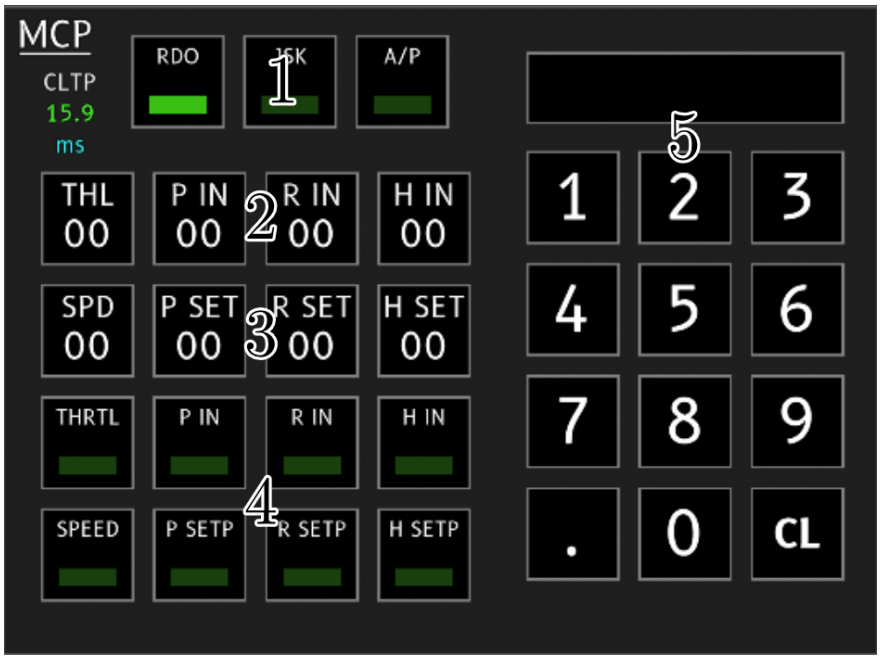

Figure 50: MCP overview

8.6 Electric Ducted Fan panel (EDF)

The EDF panel displays vital information about the EDF assembly and the electronic speed controller. Both components are fairly complex and can overheat, therefore additional temperature information is useful especially in preventing critical failure of the aircraft.

This panel displays battery compartment, electronic speed controller and EDF temperatures (2) and current EDF power setting (1) (in percent):

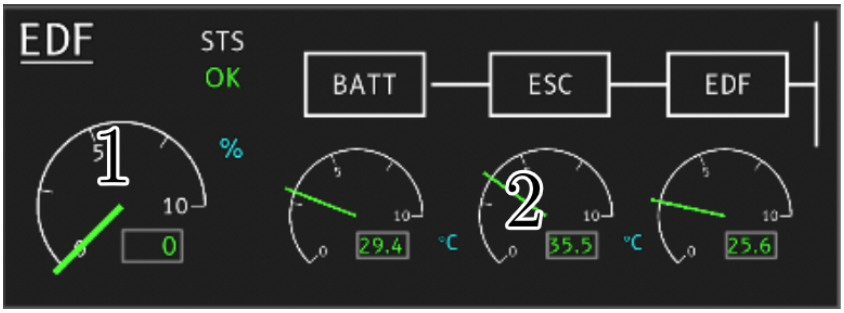

Figure 51: EDF panel overview

8.7 Communication panel (COM)

This panel displays a numerical information regarding the latencies between different components of the whole control system (LTCY) and it also contains information where there is a “backlash” in I2C communication between the ESP32s and the RPI4 (SQBH)

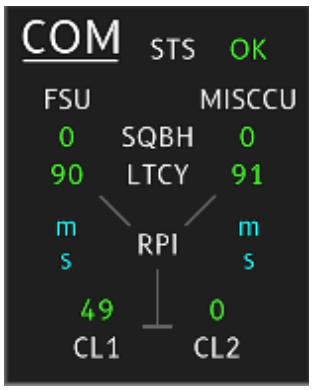

Figure 52: COM panel overview

8.8 Location panel (LOC)

This panel displays current GPS position of the aircraft on the map and also the raw LAT/LNG coordinates (1). It also provides information about the "age" of this positional data (2).

Figure 53: LOC panel overview

8.9 Master Caution System panel (MCS)

This system handles caution/error message display. Individual elements of different other systems can trigger it and cause it to display a corresponding caution message consisting of the "parent" panel label and the specific cause of the caution. This panel contains main caution light (MCL) [8](1), button to test the MCS by "injecting" a testing caution message (2), Clear First button (CLF)[9] (3) and the panel to display the caution messages, if available (4).

Figure 54: MCS panel overview

[8] This button completely extinguishes the caution light and removes all current caution messages

[9] This button removes only one (first available caution message, if it is the last one, it also extinguishes the MCL)

9 HIL simulation environment

9.1 Introduction

The idea behind this application is to simulate real-life-like flight dynamics in order to be able to perform system identification and then verity the functionality of the onboard controller. The simulation runs in real time and provides graphical output allowing the user to utilize it in functional and applicable way. Engine Unity3D is used, and the code is written in C#. The communication between the simulation and the FSU is established over UART. [4]

9.2 IMU and GPS emulation

The IMU is also emulated itself withing the simulation. This means that the simulation does not directly send information about the plane's attitude and other information but instead it emulates the accelerometer, gyroscopic sensor and magnetometer, including each respective noise parameters other behavior. The following figure shows the configuration options for the virtual IMU sensor:

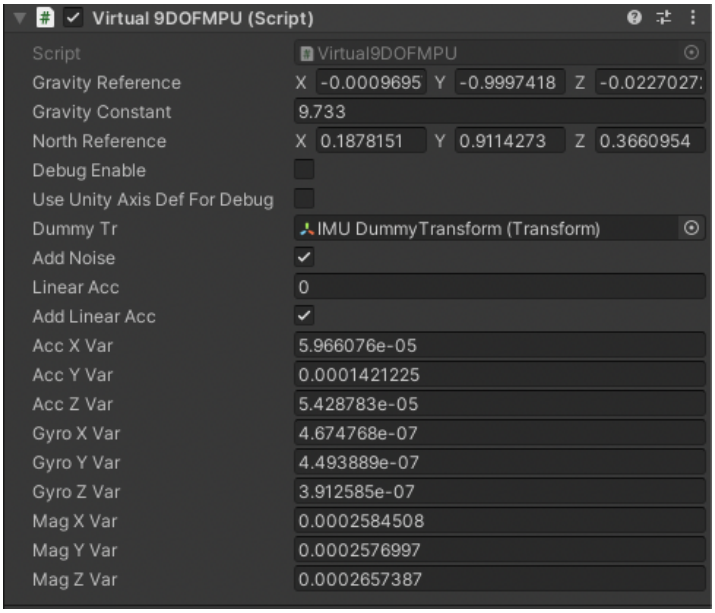

Figure 55: Virtual IMU sensor

Another sensor that needs to be emulated is GPS. For the purposes of the system analysis and controller testing, only speed needs to be gathered. This is done fairly simply by utilizing Unity's Rigidbody components and using measured refresh frequency (approximately 1.5 to 2 s).

9.3 Flight model

For the purpose of simulating aerodynamics effects on the aircraft, a premade Unity package " [5]" is used. This toolkit allows user to set up, model and simulate any Fixed-wing aircraft. It essentially uses a finite element method to determine the correct force and angular momentum values for each section of aerodynamic surface.

The following figure shows the actual simulated aircraft with its wings, vertical and horizontal stabilizer, and corresponding control surfaces set up. The blue sphere in the middle presents the CG[10] location:

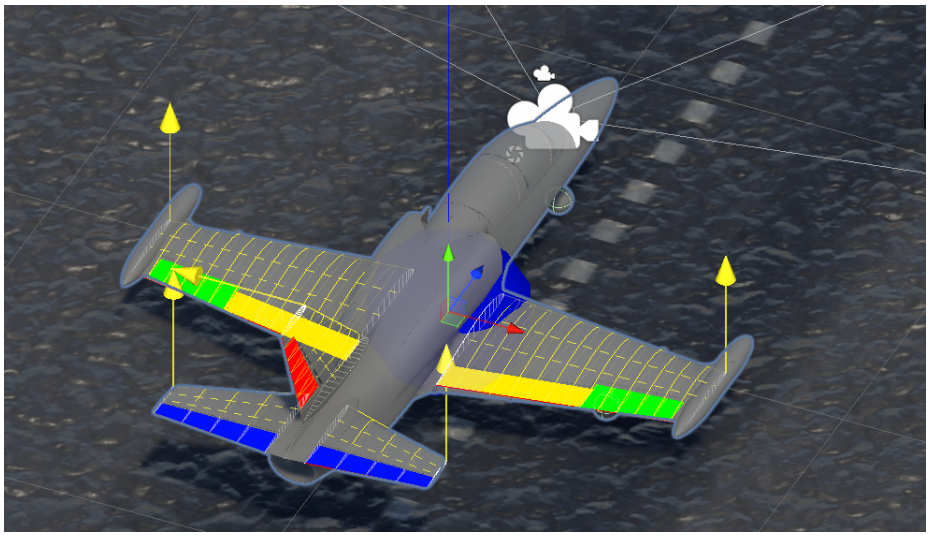

Figure 56: Simulated aircraft with visualized control surfaces

[10] Center of Gravity

10 Model identification

10.1 Identification experiment and fitting

In order to identify the system model, identification experiment must be performed. Experiments for identification of behavior in pitch axis and for speed control are performed in a different way from experiments for identification of behavior in roll axis. That is because behavior in roll axis is inherently unstable. Pitch axis behavior and speed response are stable. For this reason, pitch axis and speed behavior experiments can be performed using only open loop, only by changing the input.

The following two figures show pitch and speed behavior identification experiments

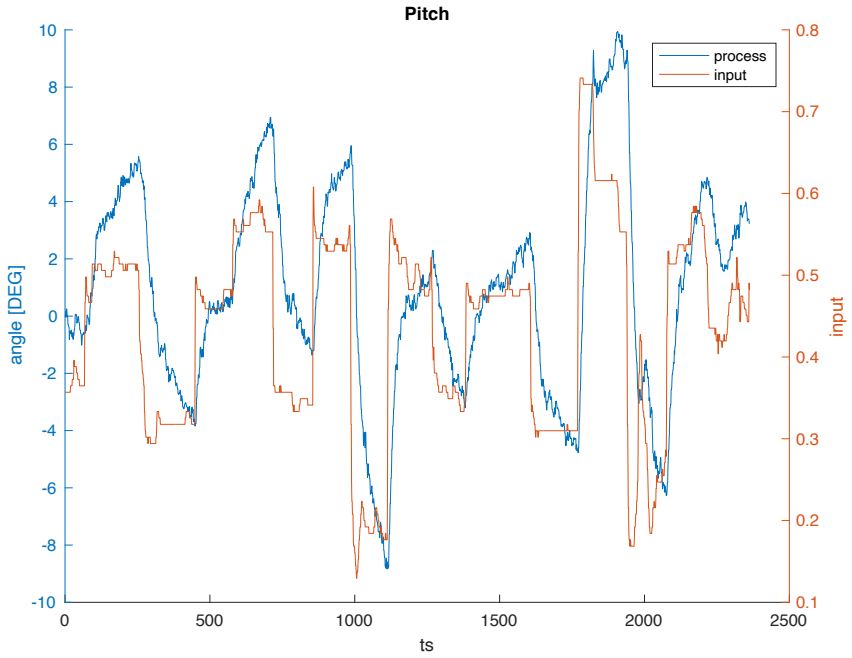

Figure 57: Pitch behavior identification experiment

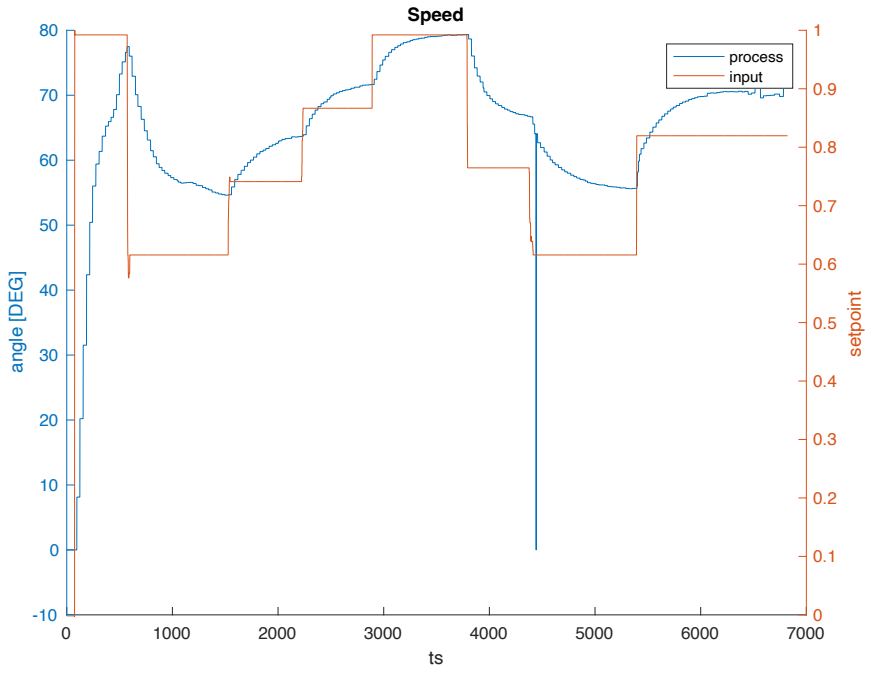

Figure 58: Speed behavior identification experiment

Identification of the roll behavior is slightly more complicated because, as mentioned above, it is unstable. In other words, if a constant input is applied, the output diverges. For this reason, this behavior must be tested in closed loop. A controller to be used for this purpose is a manually tuned PID. It does not provide closed loop behavior of a sufficient quality, but it can be used for the identification experiment, since it stabilizes the aircraft in roll axis.

The following figure shows the closed loop behavior with a manually tuned PID controller:

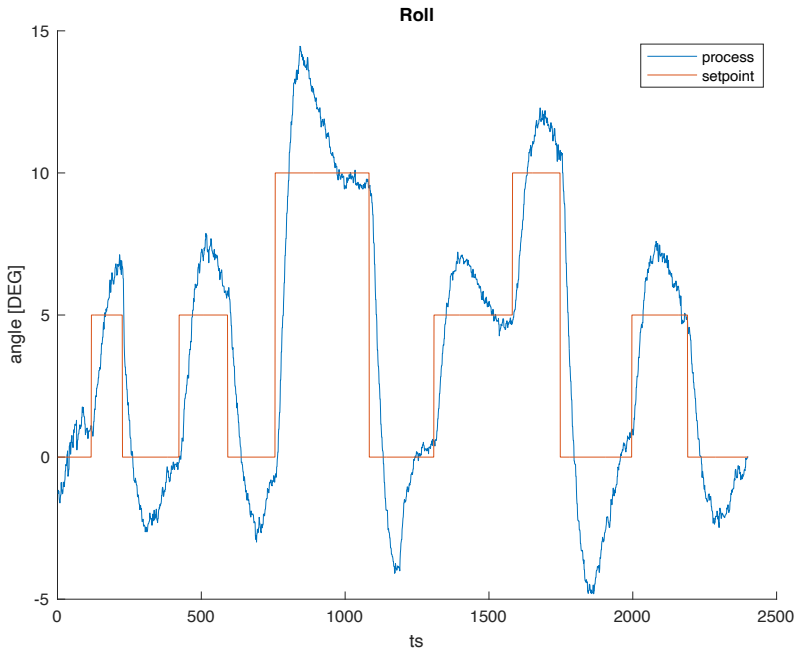

Figure 59: Closed-loop roll behavior identification experiment

Based on the pitch behavior identification experiment fitting using MATLAB *ident* toolbox is performed. First, experimental data at all its length is used to identify a nominal model ("pitch_model") and then the experimental data is split into multiple partially overlapping sections and fitting is performed individually. This results in multiple additional models ("pitch_model{1-5}"). This is done because during the experiment the orientation of the aircraft may change, resulting in slightly different dynamics. Since fitting of (in this case specifically) model #3 has not been very successful, it is discarded.

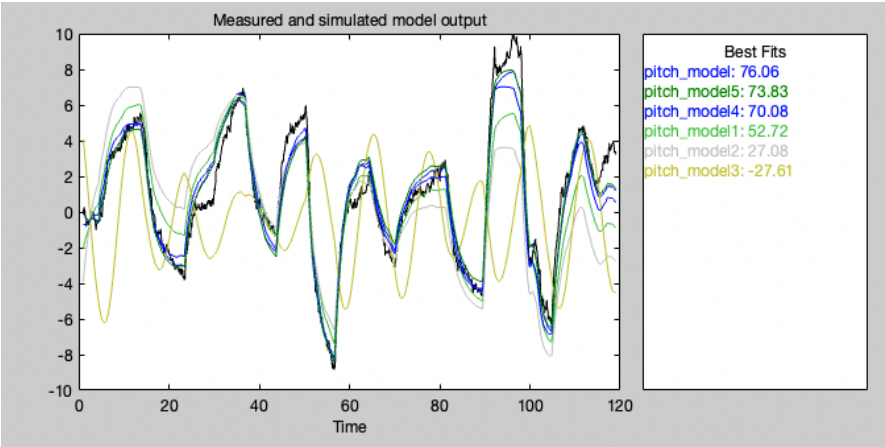

Figure 60: Pitch model fitting

The same process is performed with speed and closed loop roll data. The following figure shows the speed model fitting process. In this case, model number 4 and 5 are discarded:

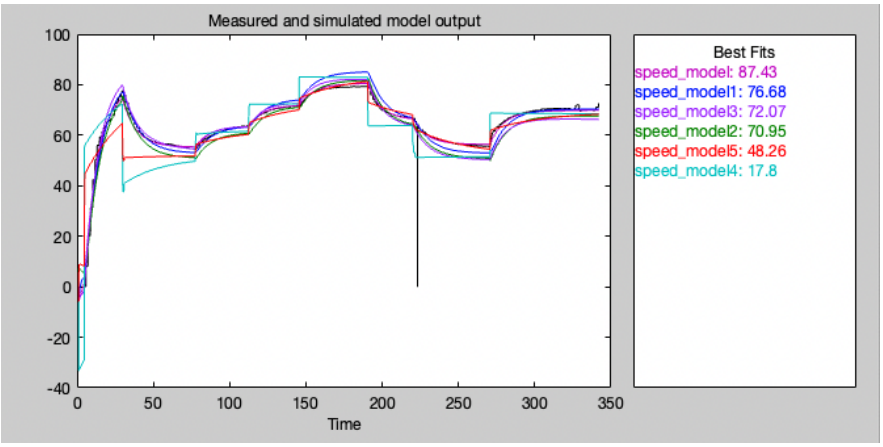

Figure 61: Speed model fitting

The following figure shows the closed loop roll model fitting process. In this case, none of the model is discarded:

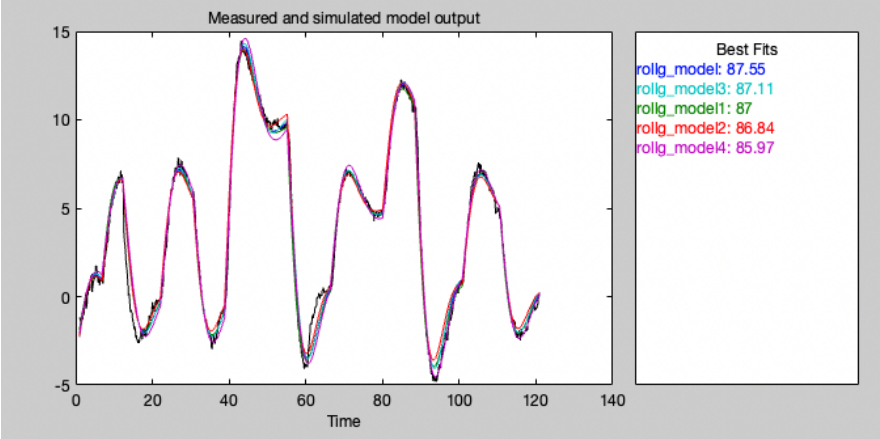

Figure 62: Closed loop roll model fitting

All fitted transfer function are second order and stable non-oscillating (poles are only real and negative).

10.2 Uncertain system model

Utilizing MATLAB build-in function *ucover* [6], array of LTI responses is used to output an uncertain model in input multiplicative form as well as shaping filter W_2:

$$P_u(s) = P_{nom}(s)(I + W_2(s)\Delta(s))$$

where $\|\Delta(s)\|_\infty \leq 1$. For each array of LTI responses (inc. nominal model) this method is used to obtain a corresponding uncertain system. Also, as a confirmation, bode diagrams of relative errors and shaping filter $W_2(s)$ are displayed to ensure the relative errors are enveloped by the shaping filter.

The following figures show corresponding bode diagrams for confirmation:

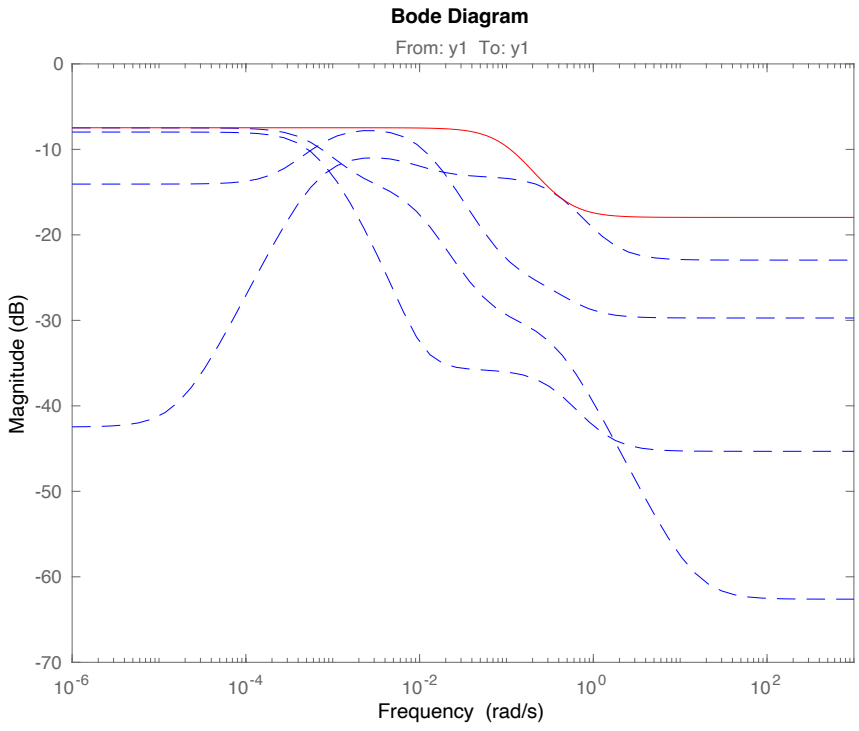

Figure 63: Pitch relative errors confirmation

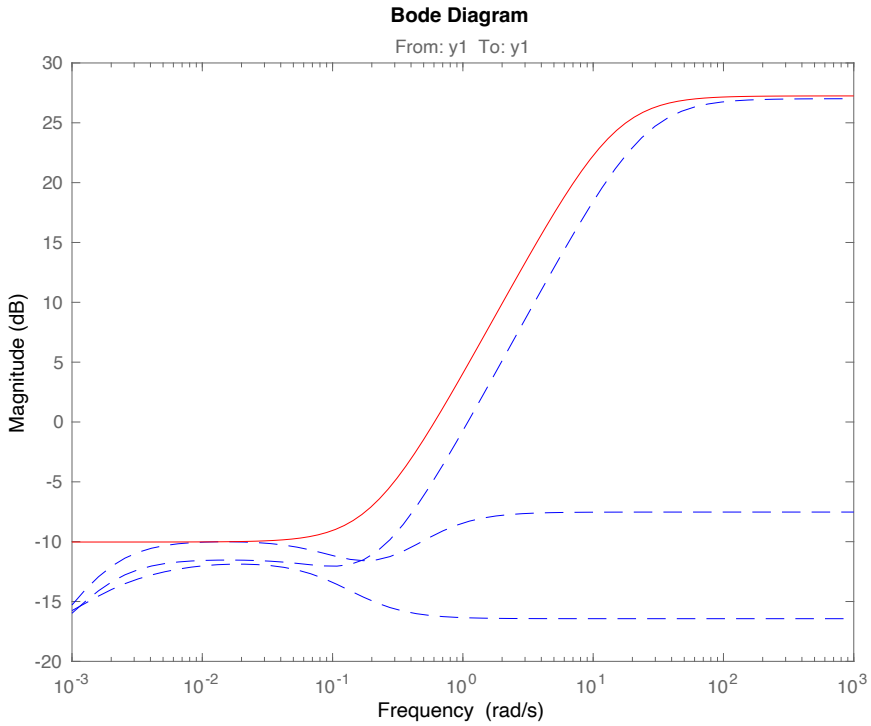

Figure 64: Speed relative errors confirmation

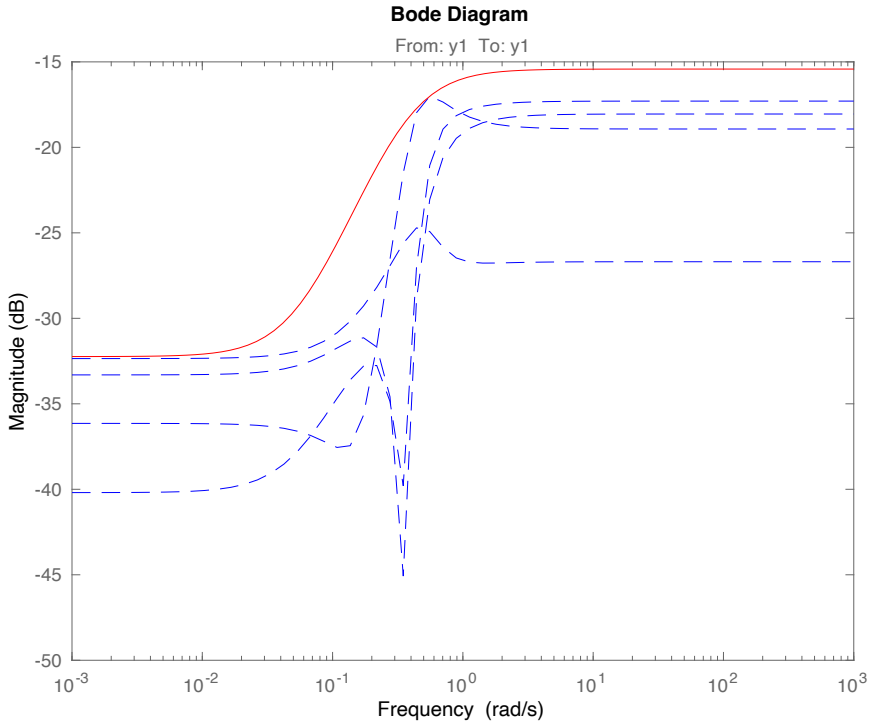

Figure 65: Roll relative errors confirmation, 1st order

From the last figure, it can be observed, that the first order shaping filter might be redundant and higher order filter might prove to be more adequate. As an example, the figure below shows the use of second order shaping filter:

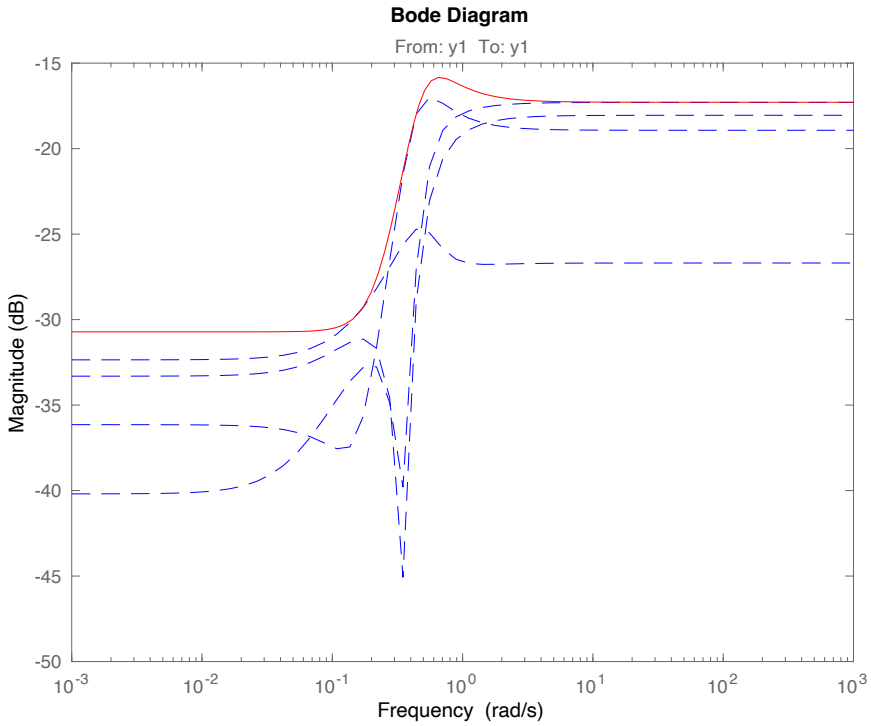

Figure 66: Roll relative errors confirmation, 2nd order

In case of roll, it is necessary to calculate the process transfer function $P(s)$ from the uncertain closed loop system $G(s)$. Since we know the parameters of the controller $C(s)$, it can be done in a following way:

$$G(s) = \frac{C(s)P(s)}{1 + C(s)P(s)} \tag{20}$$

$$P(s) = \frac{G(s)}{C(s)(1 - G(s))} \tag{21}$$

After calculating the actual roll process model (open loop), it comes out as 6th order unstable and oscillating (1 pole positive, 4 negative and one located in zero, two imaginary). This is in correlation with real-life expectation, based on the aircraft geometry.

11 Controller design and verification

11.1 MATLAB controller synthesis and step test

All individual controllers are designed as PI/D controllers using the MATLAB build-in "PID Tuner" app by adjusting the Response Time and Transient Behavior sliders to get a desired behavior - ideally very little overshoot and fast reference tracking. Each controller is tested using its corresponding uncertain system and then also compared with the manually tuned controller.

The following figure shows the comparison of the pitch controllers. The test with the manually tuned controller has slightly noticeable overshoot and slightly higher settling time as opposed to when actually tuned PID controller is used:

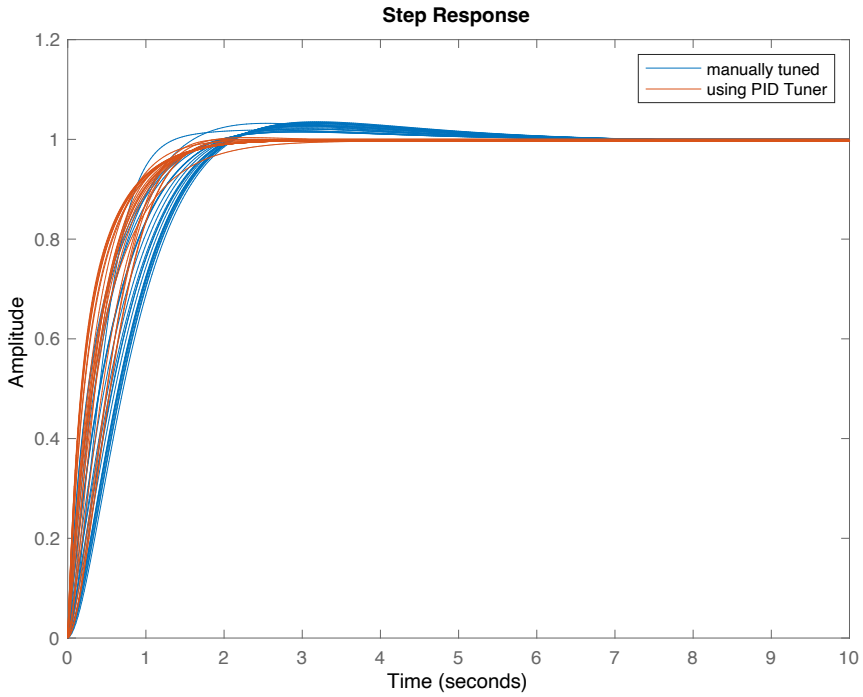

Figure 67: Pitch step response

The following figure shows the comparison of the pitch controllers. The test with the manually tuned controller has a fairly noticeable overshoot

and high settling time as opposed to when actually tuned PID controller is used. The improvement is more noticeable, in this case:

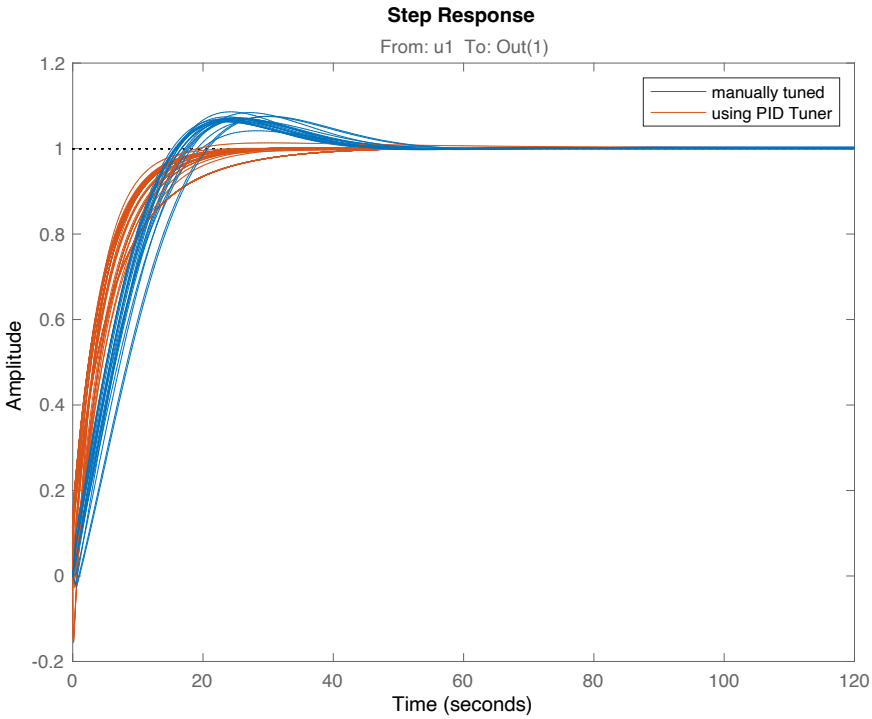

Figure 68: Speed step response

In case of roll controller, there is a visible significant improvement when compared to the manually tuned controller. Unlike the previous two controllers being practically only PI controllers, due to dynamics of the roll behavior (unstable) this controller is an actual PID.

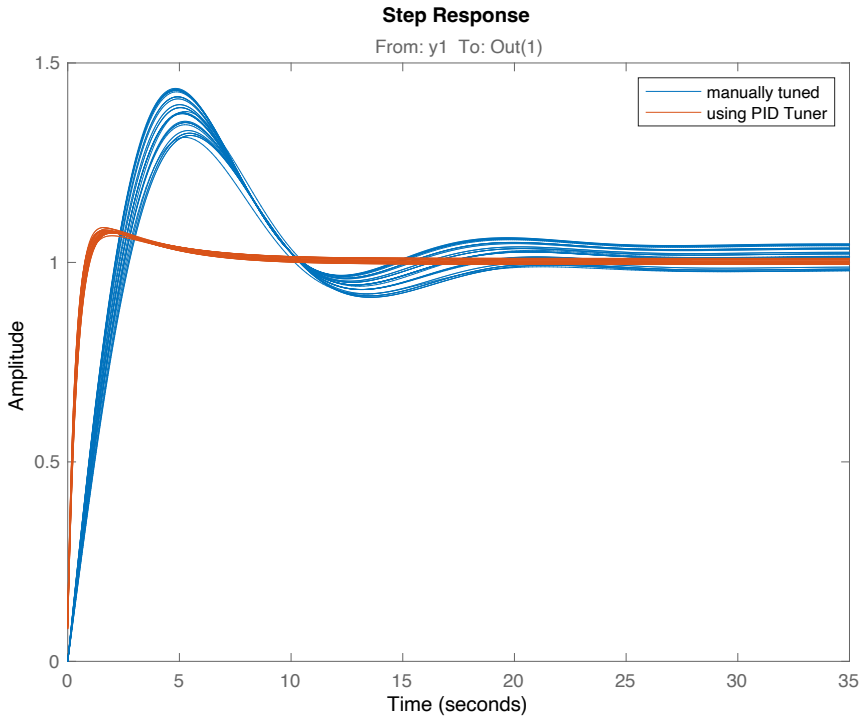

Figure 69: Roll step response

For clarity, parameters of all (manually and tuned using MATLAB PID Tuner) are shown in the table below:

Table 5: PID controllers' comparison

	P term	I term	D term
Pitch - manual	0.1	0.05	0
Speed - manual	0.01	0.0025	0
Roll - manual	0.0025	0.00125	0
Pitch - MATLAB	0.145	0.145	0
Speed - MATLAB	0.0281	0.00312	0
Roll - MATLAB	0.0169	0.00561	0.000527

11.3 Robust performance analysis

In this section, each model (pitch, roll and speed) is analyzed with its controller (respectively) in closed loop to verify whether conditions of nominal performance (NP), robust stability (RS) and at the end, robust performance (RP) are met for all of those controllers.

For each model, weighting function W_1 is chosen in order to verify NP condition in the form: $\|W_1 S\|_\infty < 1$.

> As stated in the book Feedback control theory by John Doyle [6]: "In several applications, for example aircraft flight-control design, designers have acquired through experience desired shapes for the Bode magnitude plot of S. In particular, suppose, that good performance is known to be achieved if the plot of $|S(j\omega)|$ lies under some curve."

Let ${W_1}^{-1}$ be the curve the sensitive function $S(jw)$ of the closed loop lies under.

The following figures show the choice of W_1, or rather its inverse for each model:

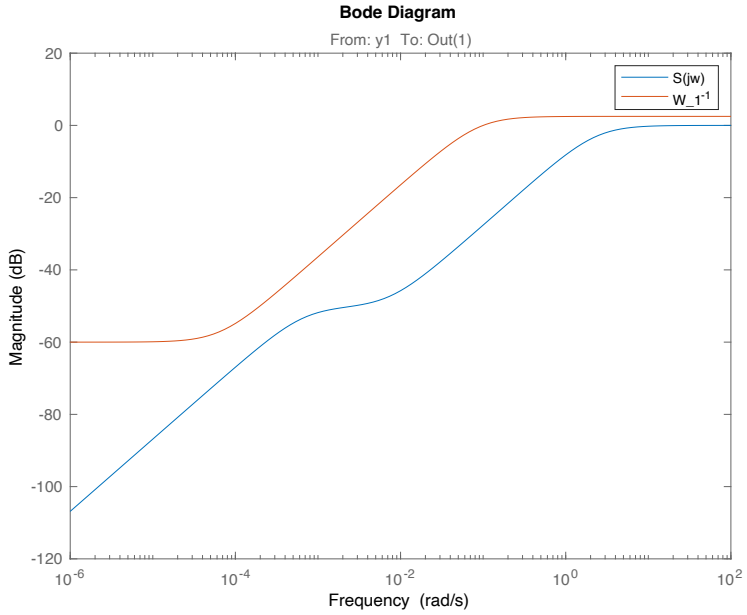

Figure 70: W_1 design for pitch model

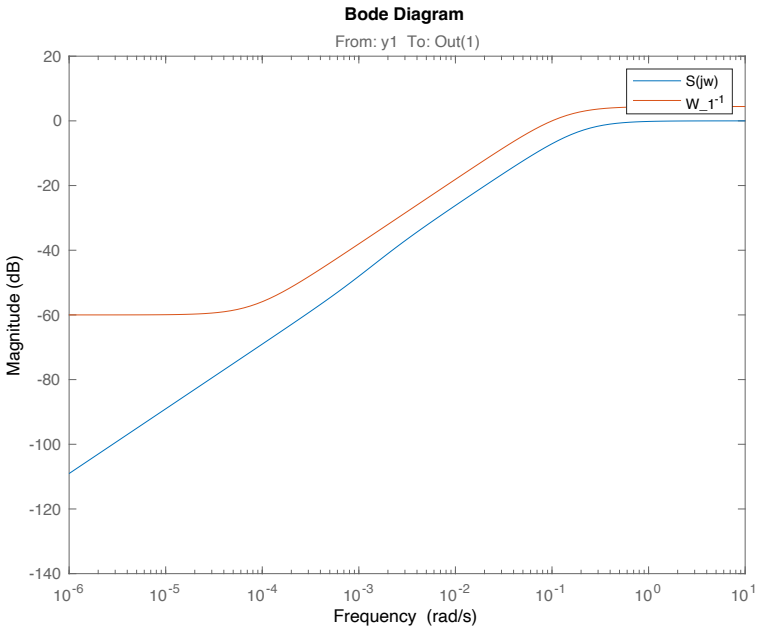

Figure 71: W_1 design for speed model

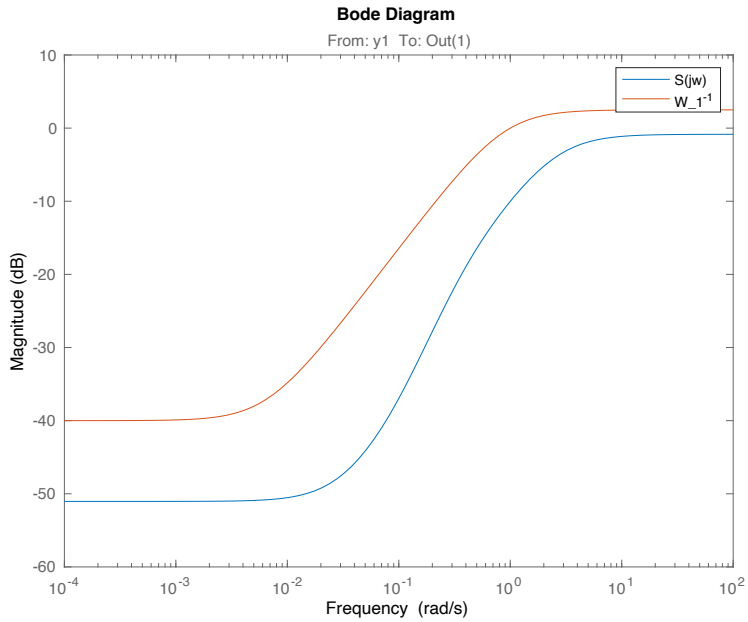

Figure 72: W_1 design for roll model

This allows to perform the NP tests:

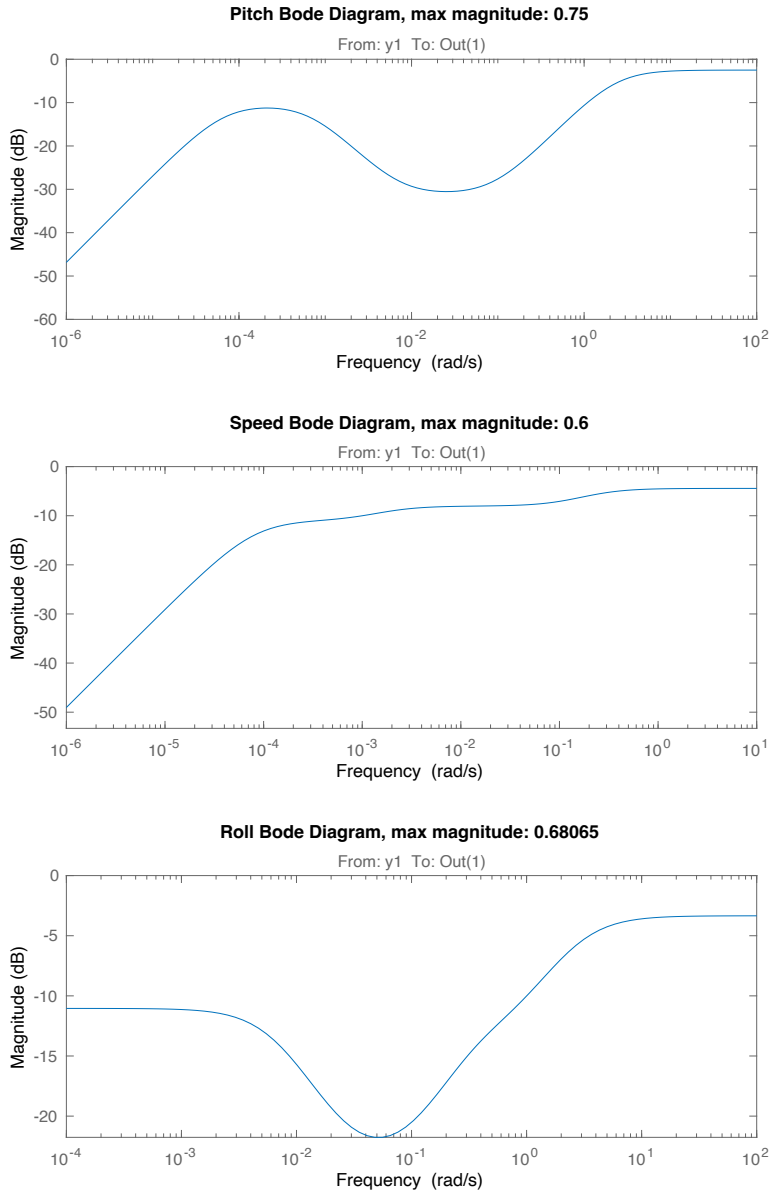

Figure 73: NP test

Given those results, it is safe to conclude, that all 3 models pass the nominal performance test. This ensures stability radius of 0.75 in pitch control, radius of 0.6 in speed control and radius of 0.75 in roll control.

Next, RS test is performed using the respected W_2 functions for each model. The condition $\|W_2 T\|_\infty < 1$ must be met for each model, where T

is the complementary sensitivity function. That will ensure that all the set of each uncertain model is stable:

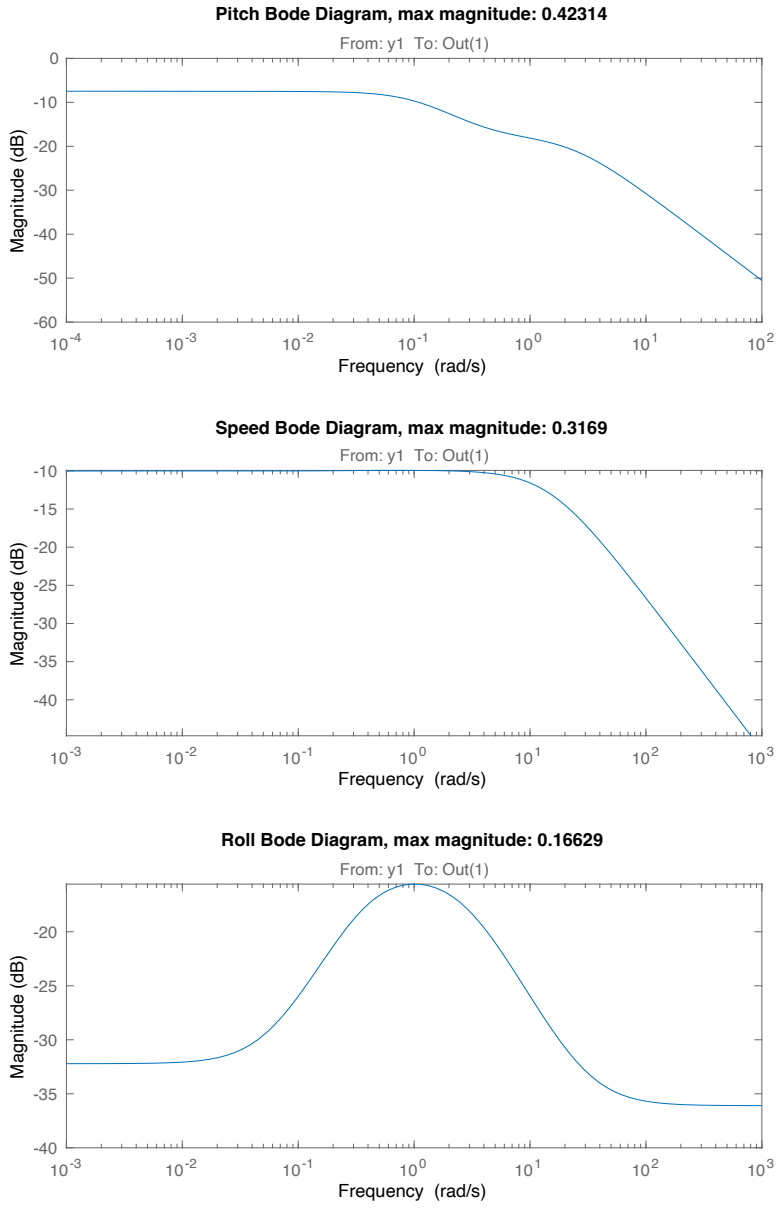

Figure 74: RS test

Given those results, it is safe to conclude, that all 3 models pass the robust stability test.

Lastly, since all model pass both, NP and RS, robust performance test can be performed. For that, the condition $\|W_1 S + W_2 T\|_\infty < 1$ must be met for each model:

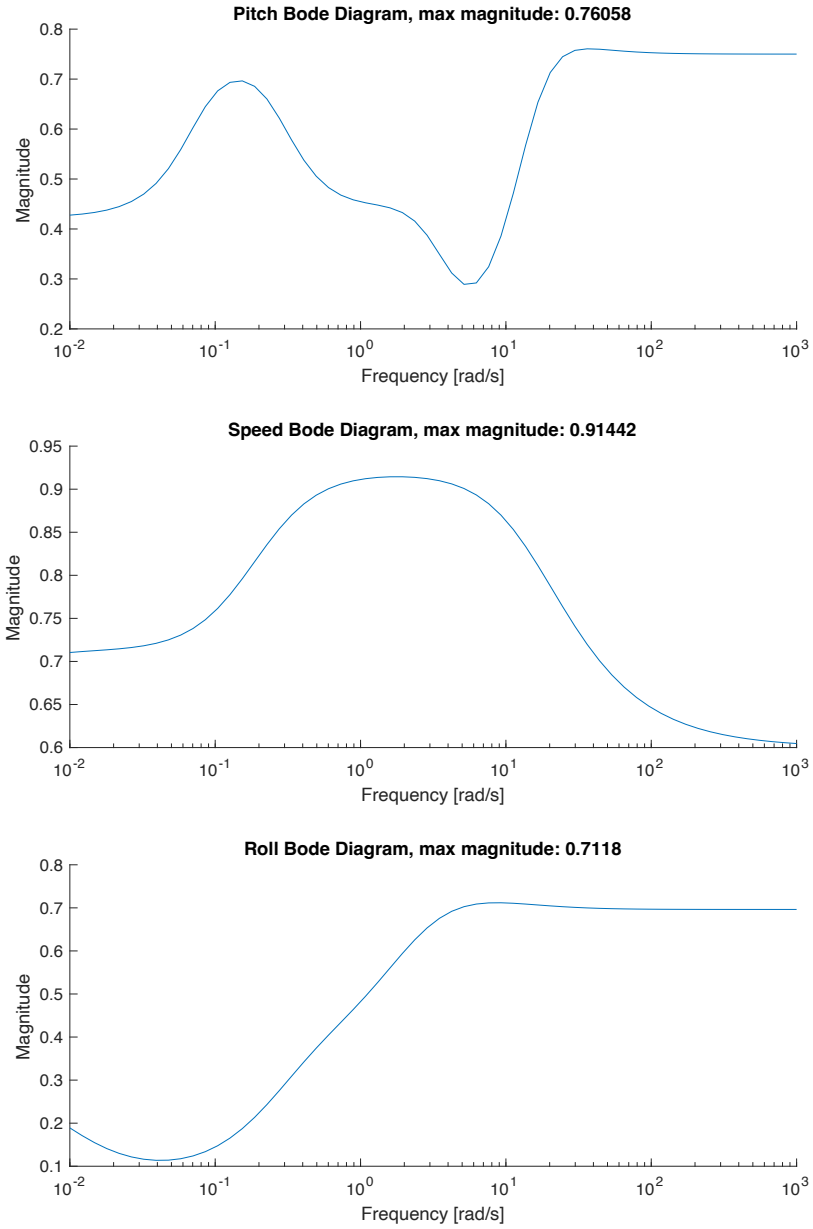

Figure 75: RP test

It can be observed that all 3 systems are robust performant given the chosen weighting functions.

11.4 Testing on the real-life Flight Computer

This testing was performed using the developed avionics including the flight computer which stores data into an SQLite database. The following three figures show the comparison on the manually tuned PID controllers and designed using MATLAB:

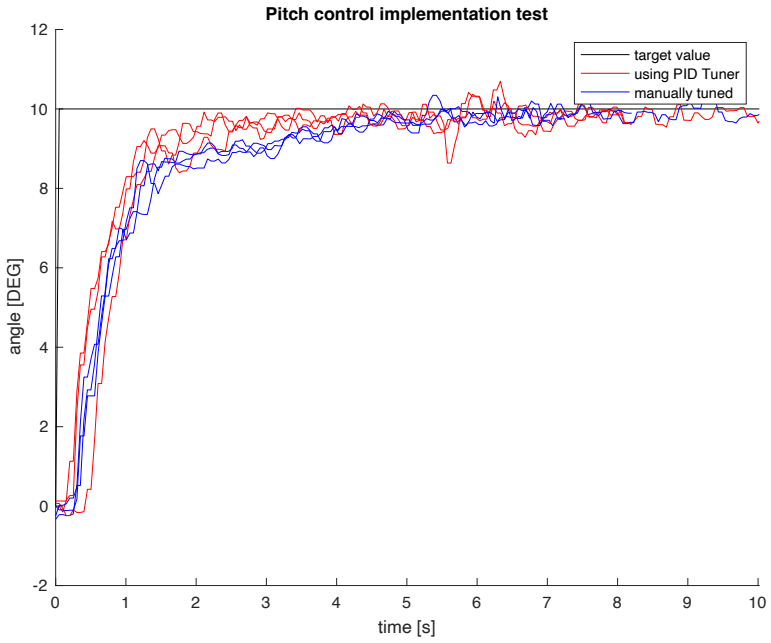

Figure 76: Pitch control implementation test

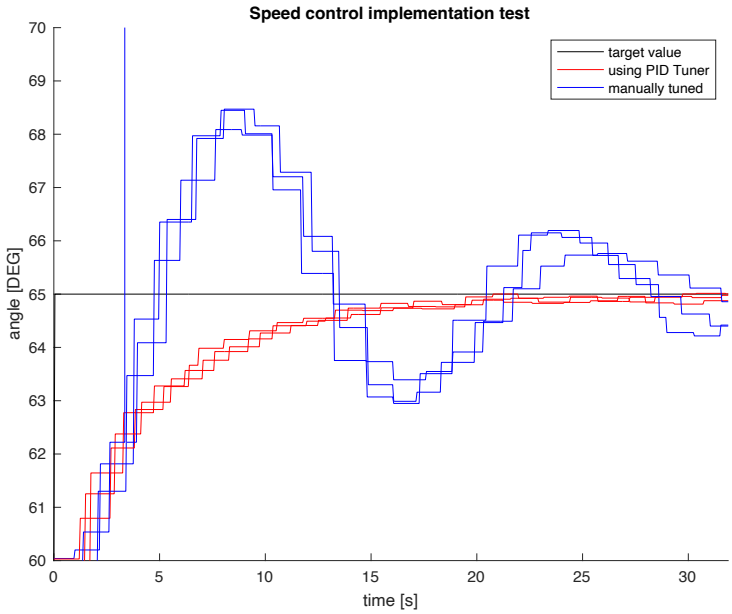

Figure 77: Speed control implementation test

The "stair" like behavior in this figure is not caused by a different sampling period of the system, but it is caused by the behavior of the GPS module which provides the speed estimates. It provides these estimates approximately every 1.5 - 2.0 seconds. Since the virtual sensor in the simulation tries to realistically emulate real-life behavior, this "stair-like" pattern can be observed.

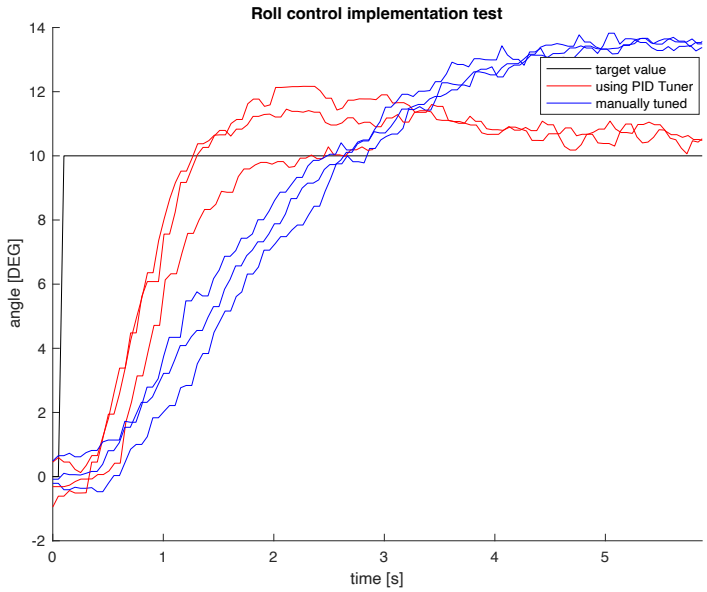

Figure 78: Roll control implementation test

12 Maiden flight

Due to the complexity and scale of this project and the limited time frame of the development, there has been only limited tests of the aircraft conducted. Those included mainly the landing gear testing and ground testing, near-take-off speed ground tests and some very short flight tests[11].

Like with real-to-scale aircrafts, each test brings information about how to adjust different procedures and processes. Those, include the behavior of the aircraft systems itself but also, very importantly safety procedures and preliminary flight checklists, as well as post-flight/post-test procedures. Although, this may sound obvious, it is important to point that out specifically.

The following few pictures illustrate the telemetry and control station as well as me performing preflight procedures:

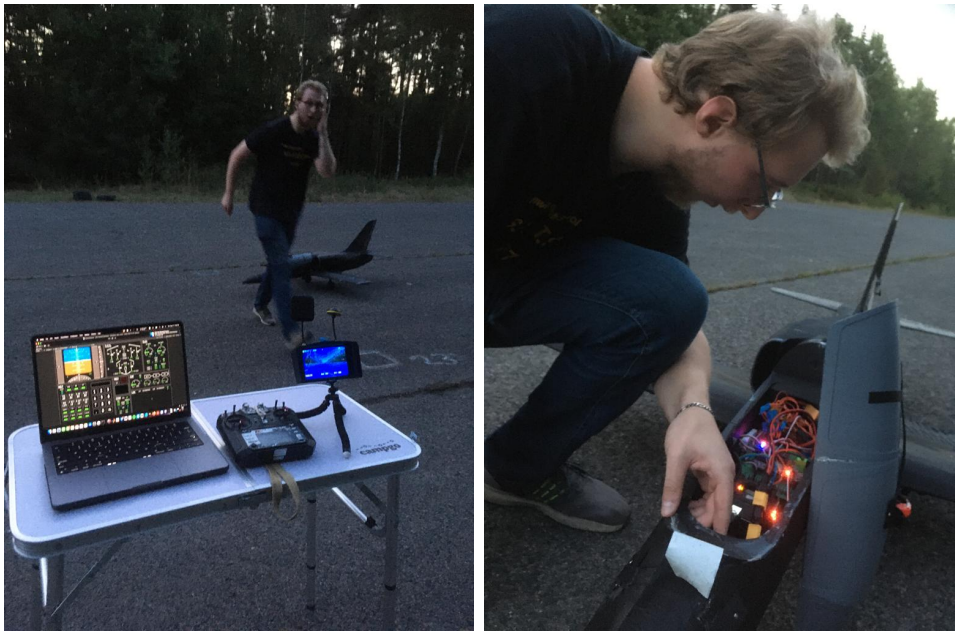

Figure 79: Preflight illustration

[11] With the last one resulting in a critical damage to the fuselage due to the loss of control caused by aerodynamical instability which is a result of material deformation due to high ambient heat of the storage location

Figure 80: To scale comparison. (For reference, my height is 1.89m)

13 Conclusion

The purpose of this thesis was to develop a control system for an aircraft, which is capable of autonomous flight with minimum interference of the operator. This was done in multiple phases, designing the avionics, including the flight computer and installing electronics components by hand, developing software in C++ and Python that runs on the flight computer and is responsible for control, data acquisition, internal communication (among the flight computer components) and external (with the ground control app), building the actual aircraft model, developing a realistic HIL simulation including emulating behavior of different sensors in order for the simulation to be able to use to verify the behavior of the avionics in real-time. Lastly, the Telemetry and Control App was developed to ensure user friendly and most importantly reliable interaction of the operator with the aircraft flight computer.

As illustrated in this thesis, all those phases went to plan and produced a reliable and robust control system. As mentioned above, due to the scale of this project and the limited time frame, actual test flights had to be very limited.

This project also required the strong enhancement of my not only different technical disciplines (such as CAD design, electronics design, mechanical engineering, knowledge of systems in commercial aircrafts such as B737, A320), but also larger scale project management and planning as well as safety and precautionary measures.

14 Works Cited

[1] Riki, "ThePoorEngineer," [Online]. Available: thepoorengineer.com.

[2] A. Narayan, "ashwinnarayan.com," [Online]. Available: https://www.ashwinnarayan.com/post/how-to-integrate-quaternions/.

[3] "Linear Algebra," [Online]. Available: https://tttapa.github.io/Linear-Algebra/arduino/Doxygen/index.html.

[4] Oyedoyin, "Unity Asset Store," [Online]. Available: https://assetstore.unity.com/packages/tools/physics/silantro-flight-simulator-toolkit-128025.

[5] I. The MathWorks, "MathWorks Help," [Online]. Available: https://www.mathworks.com/help/robust/ref/lti.ucover.html#mw_61bb8a18-7c60-42b6-9997-2d1fda3296f6.

[6] B. F. A. T. John Doyle, Feedback Control Theory, Macmillan Publishing Co., 1990.

[7] Tahustvedt. [Online]. Available: https://cults3d.com/en/3d-model/various/el-39-semi-scale-rc-jet-for-120-mm-edf.

15 List of figures