University of West Bohemia in Pilsen

Faculty of applied sciences

Department of Cybernetics

BACHELOR'S THESIS

PILSEN, 2020 MILAN MALINA

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2019/2020

Studijní program: Aplikované vědy a informatika
Forma studia: Prezenční
Obor/kombinace: Kybernetika a řídicí technika (KŘT)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení:	**Milan MALINA**
Osobní číslo:	**A17B0560P**
Adresa:	Klatovská 359, Plzeň – Litice, 32100 Plzeň 21, Česká republika
Téma práce:	Návrh systému autonomního řízení pro bezpilotní letoun
Téma práce anglicky:	Control system design for unmanned aerial vehicle autopilot
Vedoucí práce:	Ing. Martin Goubej, Ph.D. Katedra kybernetiky

Zásady pro vypracování:

1. Vytvořte simulační letový model zvoleného typu bezpilotního prostředku
2. Seznamte se s technikami autonomního řízení pro systémy udržování výšky, kurzu, rychlosti a orientace pro daný typ letounu
3. Navrhněte vhodnou strategii řízení na simulačním modelu
4. Implementujte algoritmy autonomního řízení na zvolené SW a HW platformě.
5. Otestujte navržený systém v režimu Hardware-in-the-loop simulace, v případě možnosti také na reálném modelu

Seznam doporučené literatury:

Skogestad, Postletwaithe, Multivariable Feedback Control, Wiley 2012

Podpis studenta: Datum:

Podpis vedoucího práce: Datum:

DECLARATION

I hereby submit for assessment and defense a bachelor'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 bachelor'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 consistent help and guidance during the running of this project. The meetings were vital and helpful. Furthermore, I would like to thank my parents Milan Malina, Hana Malinová for their care and my girlfriend Milagros E. Chávez for her support and motivation.

ANNOTATION

The bachelor's thesis Control system design for unmanned aerial vehicle autopilot is divided into six sections. First, development of the onboard controller is described. Second, the functionality of the ground control app is introduced together with its capabilities. It is followed by the description of the flight simulation and the equations used in this simulation. Later, these equations are utilized in the creation of the mathematical model which makes a part four. A non-linear mathematical model of the aircraft is obtained in this part. In the fifth part of this thesis this non-linear model is linearized using MATLAB. As the final part of this thesis 3 PID controllers are designed using the linearized model. The functionality of these controllers is verified using hardware in the loop simulation using the onboard controller with PID controllers implemented and the flight simulation mentioned in part three of this thesis.

KEYWORDS

UAV, aircraft, control system design, controller, PID, model, linear model, non-linear model, control software, aerodynamics

ANOTACE

Bakalářská práce na téma Návrh systému autonomního řízení pro bezpilotní letoun je rozdělena do šesti sekcí. Nejprve je popsán vývoj palubního ovladače. Za druhé, aplikace ground control je představena spolu s jejími schopnostmi. Následuje popis simulace letu a rovnic použitých v této simulaci. Následně jsou tyto rovnice využity při tvorbě matematického modelu, který tvoří čtvrtou část. V této části je získán nelineární matematický model letadla. V páté části této práce je tento nelineární model linearizován pomocí programu MATLAB. V závěrečné části této práce jsou navrženy 3 PID regulátory pomocí linearizovaného modelu. Funkčnost těchto regulátorů je ověřována pomocí hardware in the loop simulace palubního ovladače s implementovanými regulátory PID propojeného s letové simulace popsané ve třetí části této práce.

KLÍČOVÁ SLOVA

UAV, letadlo, návrh řídicího systému, regulátor, PID, model, lineární model, nelineární model, řídicí software, aerodynamika

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

This project is divided into three different sections. The first goal of this project is to design a solution for an autopilot out of commonly available parts. After their integration together the working prototype must be able to sustain a stable flight in a simulation and has to have the option of being deployed into a real-life airplane model. Therefore, the ability to control servomotors and BLDC electric motor is also required.

Second goal of this project is to enhance previous project (Flight simulation and Telemetry visualization) and implement specific features allowing "hardware-in-the-loop" simulation in order to verify the functionality of the hardware solution in form of Onboard Computer (OC) and its ability to sustain a stable flight. That means to ensure efficient enough both ways communication of the OC with the flight simulation. Also, Telemetry visualization must be able communicate with the OC wirelessly in order to ensure its purpose of delivering relevant data to the user. Therefore it does not function only as a visualization but as a full ground control (GC).

The third goal is to design a mathematical model in MATLAB and Simulink that can be used to precisely simulate the physical behavior of the aircraft which can be later used to design an efficient form of control of the aircraft.

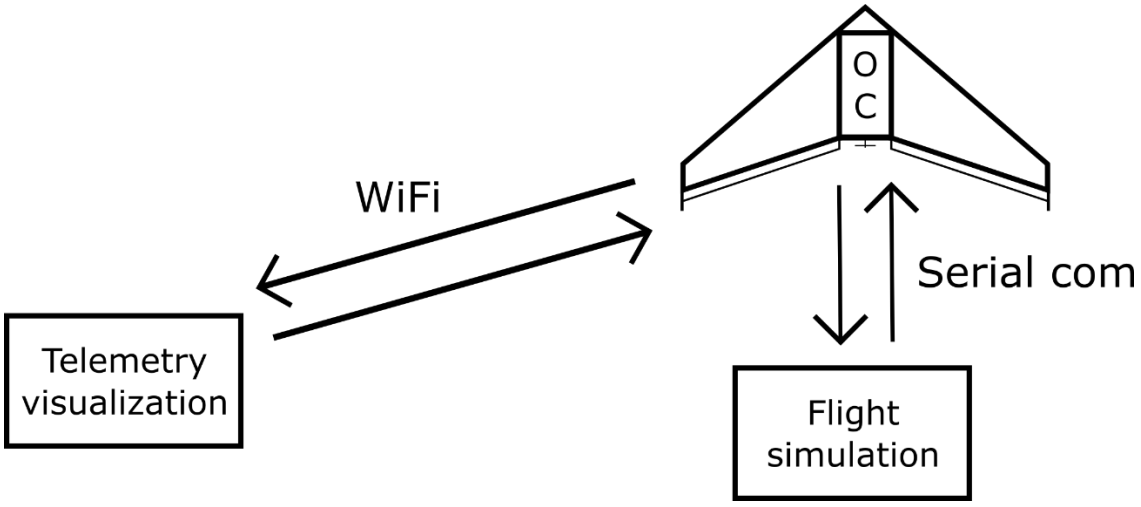

Figure 1: Overall structure

CHAPTER 2: ONBOARD COMPUTER

2. 1 INTRODUCTION

Onboard computer (OC) is essential part of any aircraft not only if autonomous operation mode is required. In order to ensure user-friendly experience, there are 3 operation modes: DISABLED – aircraft is unarmed and control outputs have default values (zero control surface deflection angle and zero thrust), MANUAL – aircraft is armed and control outputs can be changed manually and AUTOPILOT - aircraft is armed and control outputs are driven by the output of the flight controllers which reference values can be changed manually. OC consists of main MCU which purpose is to read data from the sensors and provide correct input for the driver of the actuators, communicate with the ground control and if in a simulation mode, also send and receive data to and from the flight simulation.

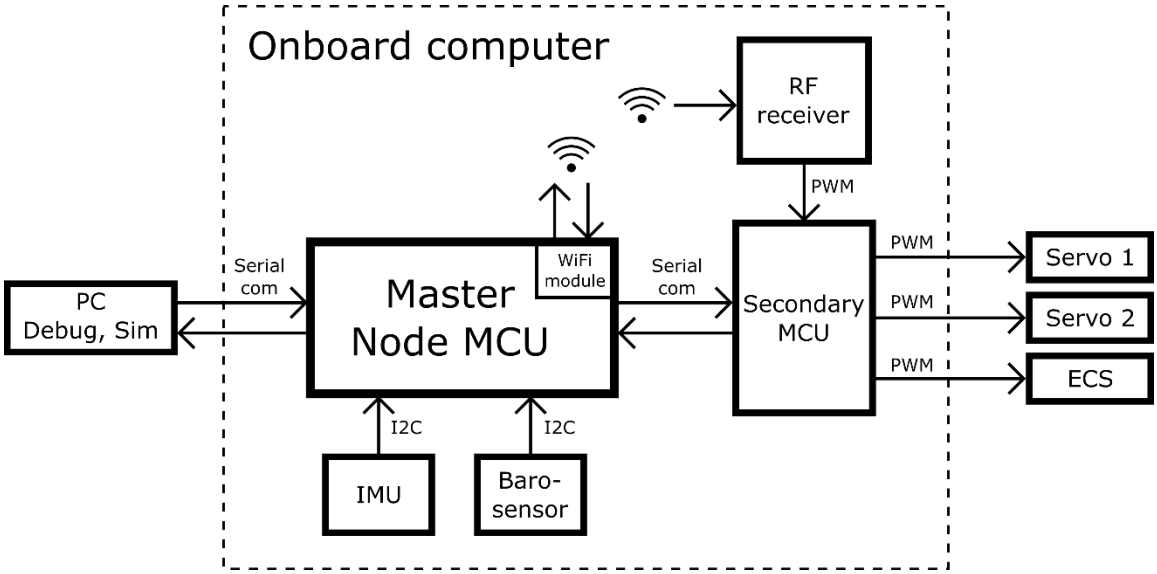

Figure 2: Onboard computer structure

2. 2 MASTER – NODE MCU

This board is the center and the main part of the OC. It reads and processes the data from the IMU and Barometric sensor, handles data exchange with both the GC and the simulation, if in simulation mode. It also sends data containing information about current control surfaces and thrust setting to the secondary MCU (this data can be read from the simulation to duplicate the current state of actuators in the simulation environed). If in manual mode, it reads data containing information about joystick channel from the secondary MCU (if joystick is enabled). This board can be either set up as WIFI AP or can connect to another AP. It runs a server to which ground control app can connect.

If autopilot is engaged, it computes the input for the actuators using 3 PID controllers (pitch, roll and velocity). The PID parameters and setpoints for all 3 PID controllers can be set up from the GC app over WIFI.

In order to process all the mentioned above, powerful enough MCU must be chosen. The current MCU is Node MCU. It can run by default on 80 MHz with the possibility of 160 MHz. For stability reasons, all testing so fast has been done using the clock speed of 80 Mhz. In most cases that proved to be sufficient enough but for future development, ESP32 with the 160 MHz up to 240 MHz clock speed would be more fitting. Both Node MCU and ESP32 are very budged friendly and they come with a build in WIFI capabilities, unlike for example Teensy 4.1 which has clock speed of 600 MHz [1] but costs way more and doesn't come with WIFI module.

Figure 3: NodeMCU illustration [5]

2. 3 SECONDARY MCU

This board has two purposes: actuator driver and RF handler. It receives information containing current desired actuator settings from the main MCU in from of serial data feed. This data is read and both servos and ESC [1] is setup accordingly. Both, servos and ESC accept 5V PWM signal generated by this board. The second purpose of this board is to read data from the RF receiver which outputs it also in the form of PWM signal for each channel. Currently 3 channels are used. By measuring the duty cycle if each of these signals, corresponding value is calculated and can be send to the main MCU over serial port. This task can be tricky and has to be done precisely and efficiently by reading directly from the registers in order to work reliably.

Arduino Leonardo Micro Pro running at 16 MHz is dedicated to being used only for these two tasks in order to eliminate possible timing issues.

Figure 4: Arduino Leonardo Micro Pro illustration [4]

2. 4 RF RECEIVER

In case manual control is selected and joystick is enabled, data from the transmitter is wirelessly transmitted to the RF receiver. It outputs this data in form of PWM signal for each channel. Currently, 3 channels are used (pitch, roll and throttle) and this specific receiver allows up to 4 channels. 3 PWM signals are then processed by the secondary MCU as described above. This allows the aircraft to be controlled in manual mode using a radio transmitter.

[1] Electronic speed controller controls the BLDC motor by outputting 3 phase AC power.

Figure 5: RF transmitter[2]

Figure 6: RF receiver[3]

2. 5 INERTIAL MEASUREMENT UNIT, BAROMETRIC SENSOR

In order to calculate the orientation in space and the altitude of the airplane, inertial measurement unit and barometric sensor must be present. It communicates with the main MCU over I2C. Using the data from IMU (MPU9250 module is used), pitch, roll and heading angles can be calculated. This is done using trigonometry so far. Kalman filter could be used to ensure better robustness.

To calculate the altitude above the sea level barometric sensor is necessary. It communicates with the main MCU over I2C the same as the IMU. The current altitude is calculated from the barometric pressure using the barometric formula.

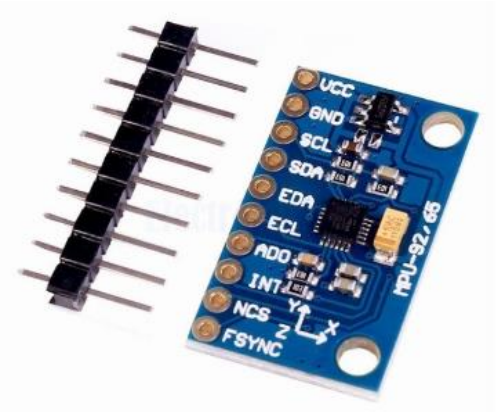

Figure 7: IMU (MPU9250)[6]

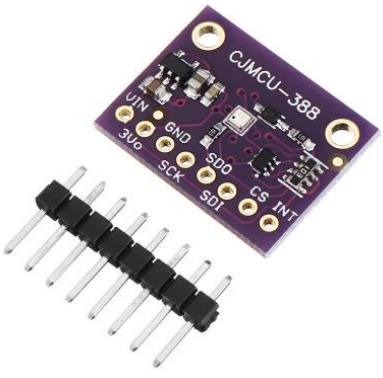

Figure 8: Barometric sensor (BMP388)[7]

CHAPTER 3: GROUND CONTROL

3. 1 INTRODUCTION

The purpose of the application is to display flight telemetry data and control the aircraft. Communication with the main MCU that controls the aircraft is established via WIFI. The user is allowed to set various flight parameters (direction, speed, etc.) and also to change MCU settings (PID controller parameters, etc.). The application is developed in the Unity3D Engine and the code is written in C#.

Figure 9: Ground control layout

3. 2 COMMUNICATION

Two-way communication with the MCU is established over WIFI and works as follows. A byte with the ID of the command to be executed (for example, telemetry data request or setting a parameter) is sent from the application to the MCU. After it is received, the MCU executes the corresponding command and sends the information on whether the execution was successful. (if telemetry data from the MCU is requested, it is sent instead of executing command.)

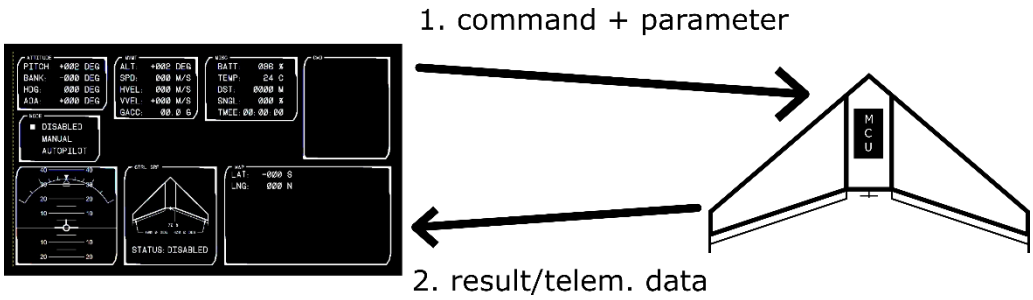

Figure 10: Communication overview

A command and its parameter separated with a space can be typed into the CMD panel in GC app for it to be executed. Available commands are described in the following table:

Table 1: Command list

Command	Reference	Parameter	ID	Description
---	READ_CMD	---	82	Send data
D	DISCONNECT_CMD	---	68	Terminate connection
M	SET_MODE	{0, 1, 2}	1	Change mode
AH	SET_ALT_HLD	{0, 1}	2	Altitude hold (YES/NO)
ILED	SET_IND_LED	{0, 1}	3	MCU LED (ON/OFF)
LS	SET_LEFT_SERVO	(0;180)	4	Left servo deflection
RS	SET_RIGHT_SERVO	(0;180)	5	Right servo deflection
T	SET_THROTTLE	(0;100)	6	Throttle level
J	SET_USE_JOYSTICK	{0, 1}	7	Use joystick (YES/NO)
P	SET_DES_PITCH	(-180;180)	8	Pitch angle setpoint
R	SET_DES_ROLL	(-180;180)	9	Roll angle setpoint
V	SET_DES_VEL	(0;50)	10	Velocity setpoint
PPP	SET_PPP	(0;inf)	11	Set pitch PID controller P term
PPD	SET_PPD	(0;inf)	12	Set pitch PID controller D term
PPI	SET_PPI	(0;inf)	13	Set pitch PID controller I term
RPP	SET_RPP	(0;inf)	14	Set roll PID controller P term
RPD	SET_RPD	(0;inf)	15	Set roll PID controller D term
RPI	SET_RPI	(0;inf)	16	Set roll PID controller I term
VPP	SET_VPP	(0;inf)	17	Set throttle PID controller - P term
VPD	SET_VPD	(0;inf)	18	Set throttle PID controller - D term
VPI	SET_VPI	(0;inf)	19	Set throttle PID controller - I term
SIM	SET_USE_SIM	{0, 1}	20	Hardware-in-the-loop mode - simulation

After a command is received by the main MCU the ground control app receives command execution result based on a following table:

Table 2: Command execution result list

Reference	ID	Description
RES_OK	100	Execution successful
RES_FAIL	101	Execution failed

After GC app startup, attempt to connect to the control MCU via WIFI is made immediately. Then, telemetry data is started to be pulled from the MCU using Unity Coroutine named `HandlePullingData` and the READ_CMD command. Telemetry data is received as a byte array from which structure named `FlightData` is created using `System.Buffer.BlockCopy` method. This structure contains the following variables: `pitch, bank, hdg, alt, velX, velY, velZ, accX, accY, accZ, gyroX, gyroY, gyroZ, magX, magY, magZ, batt, temp, posX, posY, posZ, sgnl, upTime, lsDef, rsDef, throtlle, mode`.

```
IEnumerator HandlePullingData()
{
    while (pullingEnabled)
    {
        while (networkBusy)
            yield return new WaitForEndOfFrame();
        networkBusy = true;

        float[] floatArr = { (byte)(READ_CMD), 0 };
        byte[] result = new byte[floatArr.Length * sizeof(float)];
        System.Buffer.BlockCopy(floatArr, 0, result, 0, result.Length);
        int byteSent = sender.Send(result);

        byte[] messageReceived = new byte[System.Runtime.InteropServices.Marshal.SizeOf<FlightData>()];

        while (sender.Available == 0)
            yield return new WaitForEndOfFrame();

        int byteRecv = sender.Receive(messageReceived);
        flightData = new FlightData(messageReceived);
        signalStrength = RSSIToPercent(Mathf.RoundToInt(flightData.sgnl));

        networkBusy = false;
        yield return new WaitForEndOfFrame();
    }
}
```

Figure 11: Data pulling coroutine

To send the command, wrapper method `SendCmd` is used, which runs the corresponding Coroutine, converts the command and parameter to a byte array, and sends it using `Socket.Send (byte [] buffer)`. Coroutine waits for a response containing information about the result of the execution of the given command and writes the result to the CMD panel.

```
IEnumerator SendCmdCor(int cmdIndex, float param, string cmdStr)
{
    while (networkBusy)
        yield return new WaitForEndOfFrame();
    networkBusy = true;

    float[] floatArr = new float[] { cmdIndex, param };
    byte[] result = new byte[floatArr.Length * sizeof(float)];
    System.Buffer.BlockCopy(floatArr, 0, result, 0, result.Length);

    sender.Send(result);

    while (sender.Available == 0)
        yield return new WaitForEndOfFrame();

    byte[] messageReceived = new byte[1];
    int byteRecv = sender.Receive(messageReceived);
    ConsolePanel.Instance.LogBackCmd(cmdStr, messageReceived[0] == RES_OK, messageReceived[0]);

    networkBusy = false;
}
```

Figure 12: Data sending coroutine

3. 3 DATA DISPLAYS

3. 3. 1 ATTITUDE PANEL

This panel is mostly used to display angular information of aircraft. These are: pitch angle, roll angle and yaw angle. The value of angle of attack can also be found here. All angular values are outputted in degrees and defined as represented by following figure:

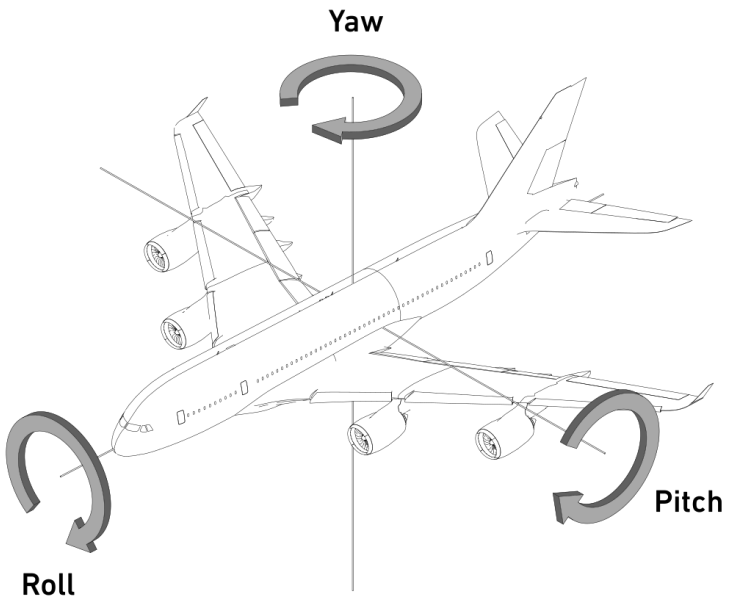

Figure 13: Attitude of the aircraft

Artificial horizon is also used in the GC app to represent the orientation:

Figure 14: Artificial horizon in GC app

3. 3. 2 MVMT PANEL

This panel is used to display more detailed information about the characteristics of movement of the aircraft in general. Those are: current flight altitude (in meters), velocity of the aircraft, its horizontal and vertical element (in ms^{-1}) and linear acceleration exerted currently exerted on the aircrafts body. This information is supplied as a g-force measurement.

3. 3. 3 MISC PANEL

This panel displays other important information. That is: remaining battery power, temperature inside of the drone (measured by IMU and barometric sensor and averaged afterwards), distance from takeoff location, WIFI signal strength and time since main MCU startup. (Some of those couldn't been implemented yet due to the current situation in the world resulting in logistics and supply delivery issues.)

3. 3. 4 CMD PANEL

The purpose of this panel is to allow the user to enter commands to be executed by the main MCU. This panel is active while the app is running and is waiting for keyboard input. Detailed description of the commands is available above the chapter Communication.

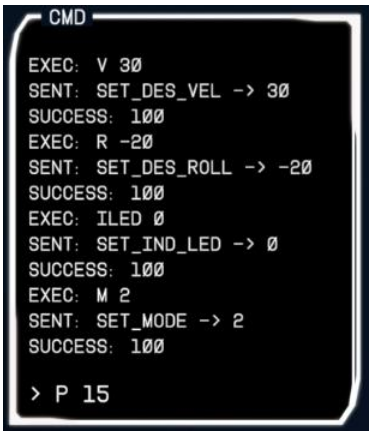

Figure 15: CMD Panel example use

3. 3. 5 CTRL SRF PANEL

This panel shows and overview of the current actuator status of the aircraft. This includes information about current control surfaces deflection and current thrust level. Aircraft status is also displayed.

Figure 16: CTRL SRF Panel description

Panels shows left control surface deflection (1), right control surface deflection (3) and current thrust level in percentage (2)

3. 4 ENHANCED CONTRAST MODE

Due to the possibility of the GC app being used outside in bright light environment, enhanced contrast mode can be toggled using F1 key on the keyboard. Following figure illustrates the difference between normal and enhanced contrast mode.

Figure 17: Comparison (normal contrast mode - left, high contrast mode - right)

CHAPTER 4: FLIGHT SIMULATION

4. 1 INTRODUCTION

The idea behind this application is to simulate real life flight dynamics in order to test the reliability and stability of the onboard controller. The simulation runs in real time and provides graphical output allowing user to understand what scenario of the simulation the OC is undergoing. Engine Unity3D is used and the code is written in C#. Thanks to that, real life environment graphics is included in order to provide better experience and better visual reference.

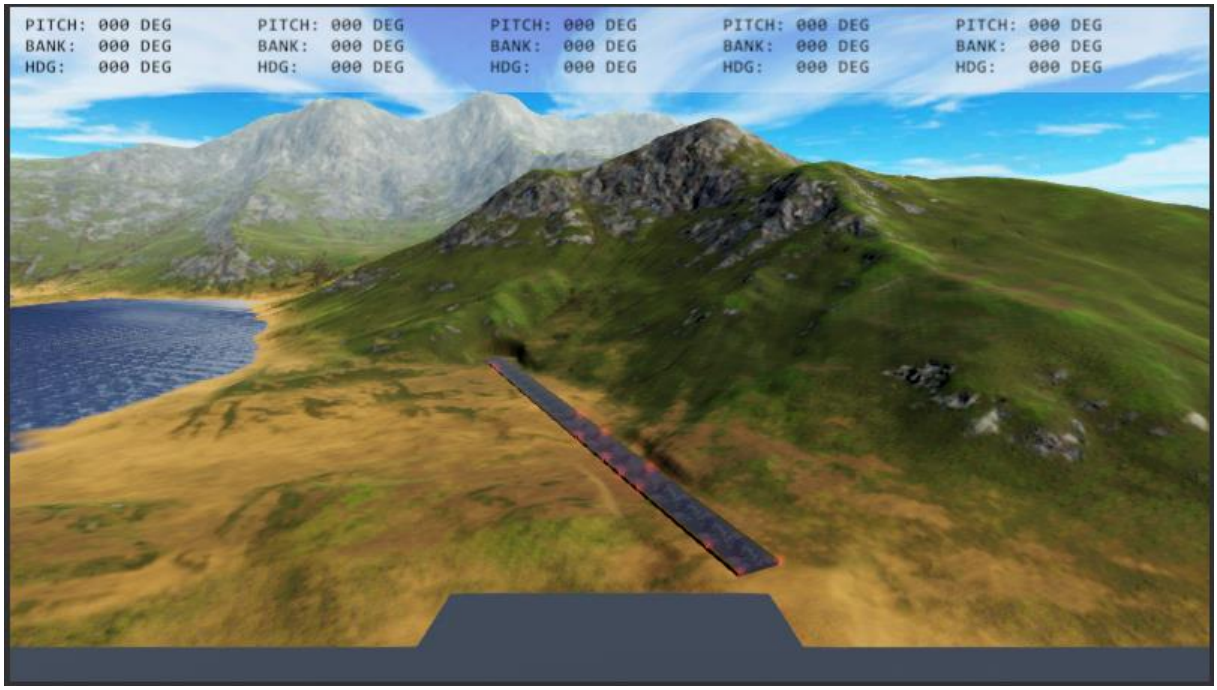

Figure 18: Flight simulation visual overview

4. 2 AIRFOIL MODEL

In order to create a flight simulation a certain amount of simplification must be considered. In this simulation, simple shaped wings only are considered which is precise enough for the purpose of this application. As stated above, the simulation is written in C# and built using Unity3D.

In the simulation, two main airfoil designs are used: NACA0015 which produces no lift at angle of attack of 0 and AG36 which produces a positive lift at the same angle of attack. (The lift and drag datapoints were obtained at webpage: airfoiltools.com where data of various wing designs is available for download in a form of a CSV table that can be loaded in code. Charts are also included on the page

so that data can be easily interpreted.) This data in a form of relations of lift coefficients and drag coefficients on angle of attack is then used in lift and drag force calculations for each airfoil.

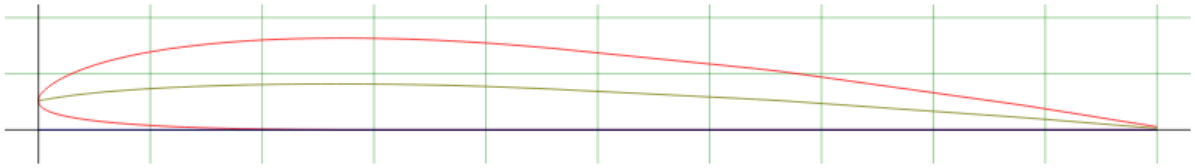

Figure 2: AG36 winglet cross section

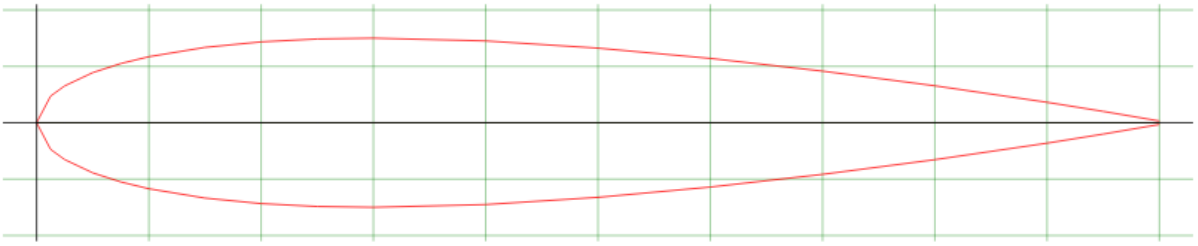

Figure 3: NACA0015 winglet cross section

The lift coefficients and drag coefficients data set values also depends on Reynolds number of the environment where the winglet characteristics were measured.

In order to choose the right lift coefficient and drag coefficient values, current angle of attack must be calculated as an angle between a vector of relative velocity of the airfoil and a vector that points the same direction as the airfoil.

In order to obtain correct lift forces, following formulas must be used:

$$F_L = \frac{1}{2} C_L \rho v^2 A$$

where C_L is the coefficient of lift obtained from the table mentioned above, ρ is air density, v is the current velocity of the airfoil and A is its wing area. To get the direction of the force it is necessarily to cross multiply the world velocity vector with a vector which points to the right of the airfoil.

Drag force can be calculated in a similar way:

$$F_D = \frac{1}{2} C_D \rho v^2 A$$

where C_D is the drag coefficient also obtained from the table mentioned above and the direction is the opposite as the direction of the world velocity vector.

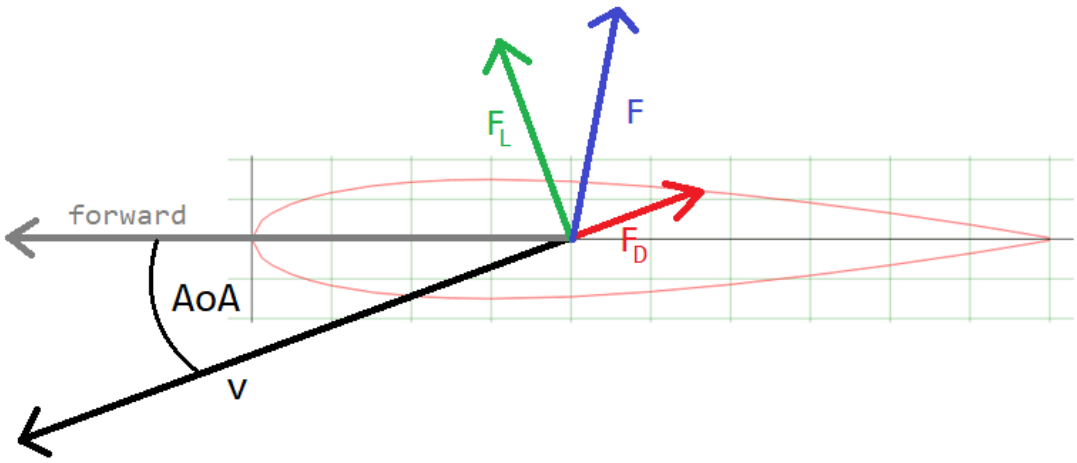

Figure 4: Forces acting on a wing

As mentioned above, Unity3D provides a built-in physics engine. Each object that physically interacts with its environment, it needs to have a Rigidbody component attached to it. This component provided by the Unity API brings various functionality and allows developer to directly define forces applied to this Rigidbody. For this, the most important method is: *AddForceAtPosition(Vector3 force, Vector3 position, ForceMode mode)*. This function takes two vectors and one enum describing the way the force will being applied. Vector called *force* describes the direction and the magnitude of the force and vector *position* defines at what point in the world coordinates the force is applied. Enum *mode* specifies in which mode the force should be applied. It depends on what the result should be. In this case it's set to *ForceMode.Force*. After method mentioned above is called for every wing surface, the physics engine calculates position, rotation, velocity and angular velocity for the next physics frame. The physics engine runs on a separate thread. The physics update is fixed and independent from visual update (graphics render). To ensure higher precision, the physics fixed update runs with a period of 2ms that means 500 times per second.

4. 3 DATA TRANSFER

In order to ensure hardware-in-the-loop simulation capabilities, the data transfer between the OC and the simulation is necessary. OC (main MCU exactly) can be connected using two serial ports. (The main MCU – Node MCU is capable of only 2 serial connections – debug, secondary board communication) This could be solved by either disabling debug option and use the debug serial port for the two-way communication with the simulation or by using for example ESP32 board, which allows more hardware serial connections. Unfortunately, debug capabilities proved to be too valuable and due

to logistic problems, ESP32 couldn't be acquired either. The communication had to be designed as follows. The debug serial port is used only one way (Sim -> main MCU) in order to send data from the simulation to the OC so it can be used instead of data from the real sensors, therefore not interrupting debug messaged from the Node MCU. And the data containing the current actuators settings can be acquired directly by reading the Main MCU -> Secondary MCU data stream. Following figure explains the data traffic.

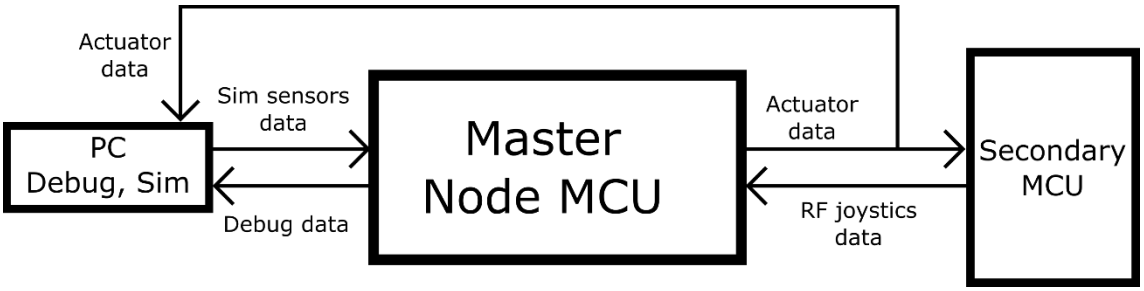

Figure 19: Hardware in the loop overview

CHAPTER 5: MATHEMATICAL MODEL

5. 1 INTRODUCTION

In order to implement functional autopilot solution including flight controllers, mathematical model must be developed. MATLAB and Simulink can be used to develop a nonlinear model which later can be linearized using Control systems toolbox. In order to verify the models accuracy, already developed Unity flight simulation can be used. In order to perform verification the data from MATLAB must be stored and loaded by the Unity engine. Following schematics shows the Simulink subsystem simulating the flight dynamics:

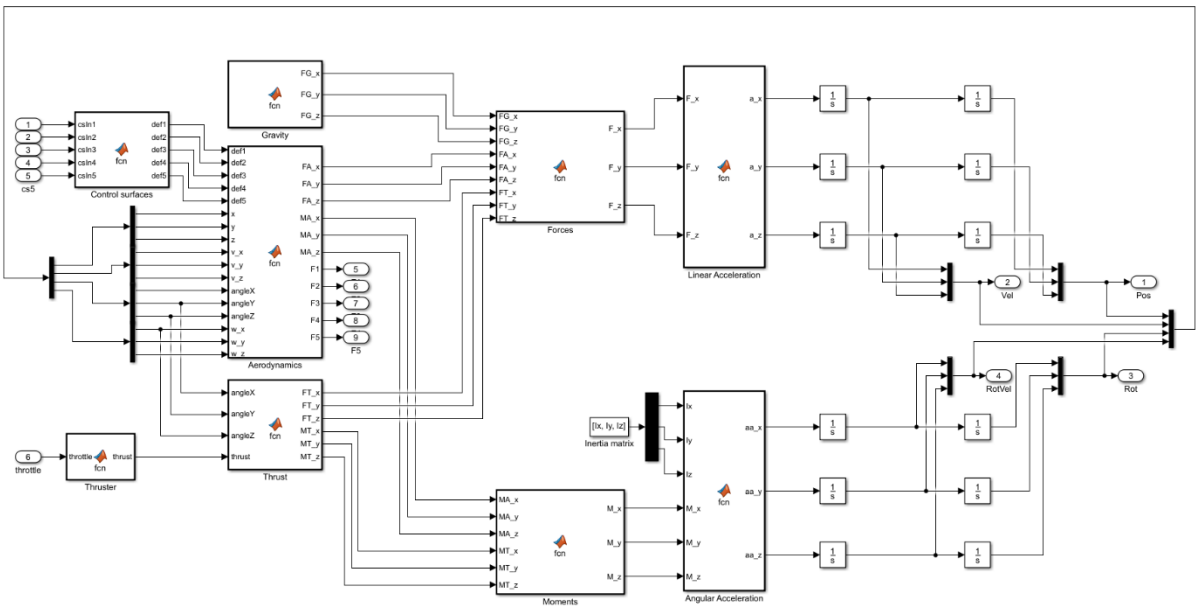

Figure 20: Flight dynamics subsystem

Following table describes which input block corresponds to which actuator:

Table 3: Subsystem inputs

Input #	Description
1	Right aileron input
2	Left aileron input
3	Right elevator input

4	Left elevator input
5	Rudder input
6	Throttle input

Following table describes which output block corresponds to which information:

Table 4: Subsystem outputs

Input #	Description
1	Linear position vector (x, y, z)
2	Linear velocity vector (x, y, z)
3	Angular position vector (Euler angles)
4	Angular velocity vector (Euler angles)
5	Force vector (right front wing)
6	Force vector (left front wing)
7	Force vector (right rear wing)
8	Force vector (left rear wing)
9	Force vector (rudder)

5. 2 COORDINATE SYSTEM

In order to describe the orientation of any vessel in general a coordinate system must be defined. In order to be able to used Newtons laws of motion, this coordinate system must be inertial. For that, coordinate system connected with ground can be used. The origin of this global coordinate system is the takeoff location. The individual axes are defined as follows:

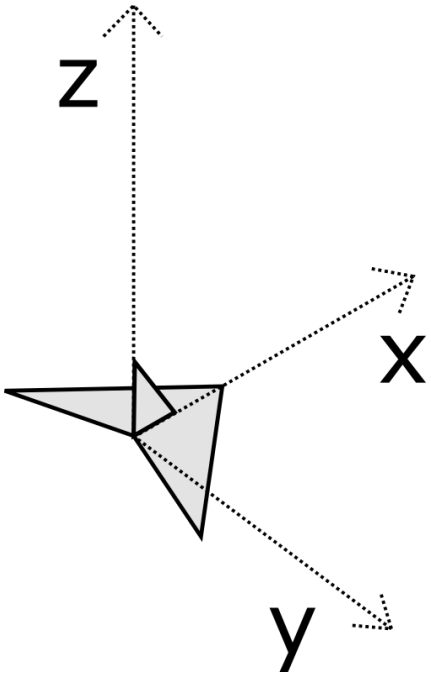

Figure 21: MATLAB model coordinates

This figure shows how the aircraft is positioned at the beginning of the simulation and how the coordinate system is defined. The positive x-axis points forward, positive y-axis points right and positive z-axis points up. The coordinate system in Unity3D is defined as follows, that's why in order to view and verify data obtained from MATLAB in Unity the must be converted into a correct format. Unity3D defined its coordinate system by default as follows:

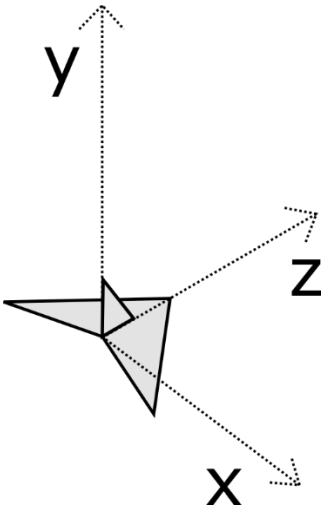

Figure 22: Unity sim coordinates

The positive z-axis points forward, positive x-axis points right and positive y-axis points up.

In order to perform transformation between the global coordinate system and a local coordinate system connected with the aircraft, rotational transformation matrix R can be used as illustrated in the following graphics:

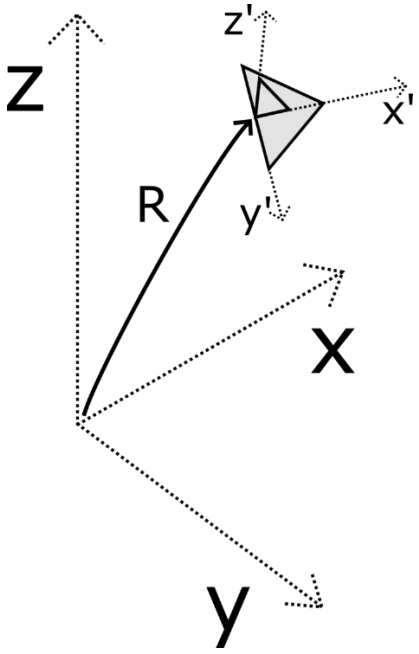

Figure 23: Coordinate transformation

Matrix R can be defined as follows: $R = R_1 R_2 R_3$, where $R_{1,2,3}$ are individual rotation matrices. These matrices can be defined as follows:

$$R_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & \sin(\varphi) \\ 0 & \sin(\varphi) & \cos(\varphi) \end{pmatrix}$$

$$R_2 = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{pmatrix}$$

$$R_3 = \begin{pmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Where φ, θ, ψ described angular orientation of the aircraft. In order to perform transformation from the aircrafts local coordinate system to global coordinate system $R_{INV} = R^{-1}$ matrix can be used.

5. 3 LINEAR/ANGULAR ACCELERATION FUNCTIONS

Based on Newtons laws of motion, linear acceleration is calculated as a following ratio: $a = \frac{F}{M}$ and angular acceleration as follows: $\alpha_i = \frac{\tau_i}{I_i}$, where F is a vector of sum of all forces acting on the rigidbody, τ is a vector of sum of all moments acting on the rigidbody. F a τ are supplied by blocks names as "Forces" and "Moments". M is mass of the rigidbody and I is its moment of inertia in a corresponding axis i. Matrix of inertia I is defined as $I = \begin{pmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{pmatrix}$ and its values are calculated in MATLAB for a specific aircraft geometry.

5. 4 FORCES/MOMENTS FUNCTIONS

These blocks only add up vectors of gravitation force F_G, aerodynamic force F_A and thrust force F_T. Respective moments are added up as well. The rotational center is defined as a center of mass, therefore gravitational moment is zero.

5. 5 GRAVITY FUNCTION

Based on Newtons second law, the gravitational force [2]vector is calculated as follows: $F_G = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix}$, where m is the mass of the rigidbody and g is gravitational acceleration defined as $g = 9.81\, ms^{-1}$.

5. 6 AERODYNAMICS FUNCTION

This function is responsible for calculating the aerodynamics effects of the airfoils of the aircraft. For each airfoil, it runs *CalcWingEffect* function. This function takes airfoil deflection, lift offset

[2] In a global coordinate system

taken from the airfoil characteristics graph, airfoil Euler rotation in local aircraft space, airfoil area, airfoil position in local aircraft space, aircraft linear and angular velocities and aircraft Euler rotation in global space. Based on these data it calculates and outputs the force and momentum exerted by the airfoil. It uses the same aerodynamics formulas for lift and drag magnitude as the Unity flight simulation. Direction of lift can be obtained as a cross product of velocity vector and wing direction vector $F_{A_{dir}} = v \times D_w$. Wing direction vector is defined as $D_w = [0 \quad 1 \quad 0]$ relative to the wing. The direction of drag is opposite the velocity: $F_{D_{dir}} = -v$. The total force acting on the wing is the sum of lift and drag: $F = F_A + F_D$. The total moment is also calculated with the cross product (using wing position and the total force).

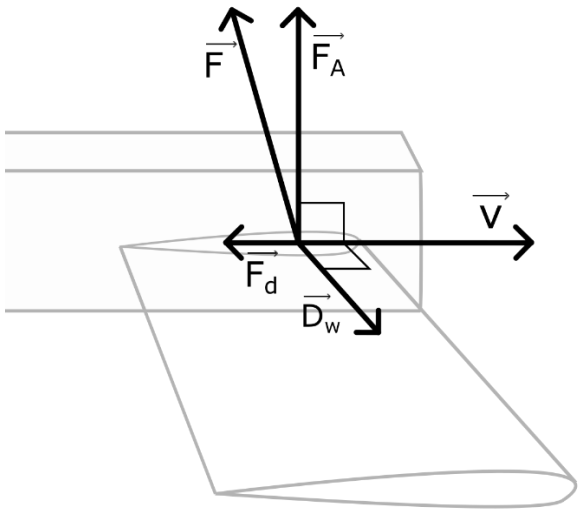

Figure 24: Forces acting on an airfoil

In order to obtain correct lift force magnitude, following formula must be used:

$$F_A = \frac{1}{2} C_L \rho v^2 A$$

where C_L is the coefficient of lift described in more detail below, ρ is air density, v is the current velocity of the airfoil and A is its wing area. Drag force magnitude can be calculated in a similar way:

$$F_D = \frac{1}{2} C_D \rho v^2 A$$

where C_D is the drag coefficient. In order to supply the right lift coefficient and drag coefficient values, current angle of attack must be calculated as an angle between a vector of relative velocity of the airfoil and a vector that points the same direction as the airfoil. Depending on AoA and the geometry of the airfoil, corresponding value of coefficient of lift and drag is obtained.

5. 7 CONTROL SURFACES FUNCTION

This function supplies the model with information about the current deflection of the control surfaces. It remaps and clamps the values inputted into the subsystem based on the physical characteristics of the modeled aircraft. More advanced dynamics or the control surfaces could also be implemented. For each control input i, the function performs following operation and outputs value o.

$$o = Clamp(Remap(i, -1,1, -d, d\ ), -d, d)$$

(for ailerons and rudder),

$$o = Clamp(Remap(i, 1, -1, -d, d\ ), -d, d)$$

(for elevons). Where value d is the max deflection, function Clamp is defined as follows:

$$Clamp(v, min, max) = \begin{cases} \min if & v < min \\ \max if & v > max, \\ \mathrm{v}\, if & else \end{cases}$$

And function Remap is defined as follows:

$$Remap(v, a_1, a_2, b_1, b_2) = b_1 + \frac{(v - a_1)(b_1 - b_2)}{a_2 - a_1}$$

5. 8 THRUSTER/THRUST FUNCTION

Similarly to the Control surfaces function, Thruster function utilizes the Clamp and Remap functions to calculate the correct thrust force depending on the throttle input. Thrust function takes into consideration the aircrafts rotation and using transformation matrix transforms the thrust force vector accordingly. It outputs the calculated global thrust force and moments into the model.

5. 9 MODEL VERIFICATION

Unity3D has a robust and versatile physics engine capable of simulating motion of rigid bodies. Its robustness proved to be useful when creating the flight simulator, when only after defining the lift and drag forces the whole flight dynamics could be utilized.

As mentioned above, Unity3D has a built in physics engine and therefore Unity flight simulation can be used to verify the functionality of the MATLAB mathematics model. Using a simple script, data containing information about the global position and global orientation in Euler angles of the rigidbody can be obtained and stored in a CSV file.

The comparison on the global position of the rigidbody in MATLAB model and Unity simulation is possible from the three figures below (one figure for each axis):

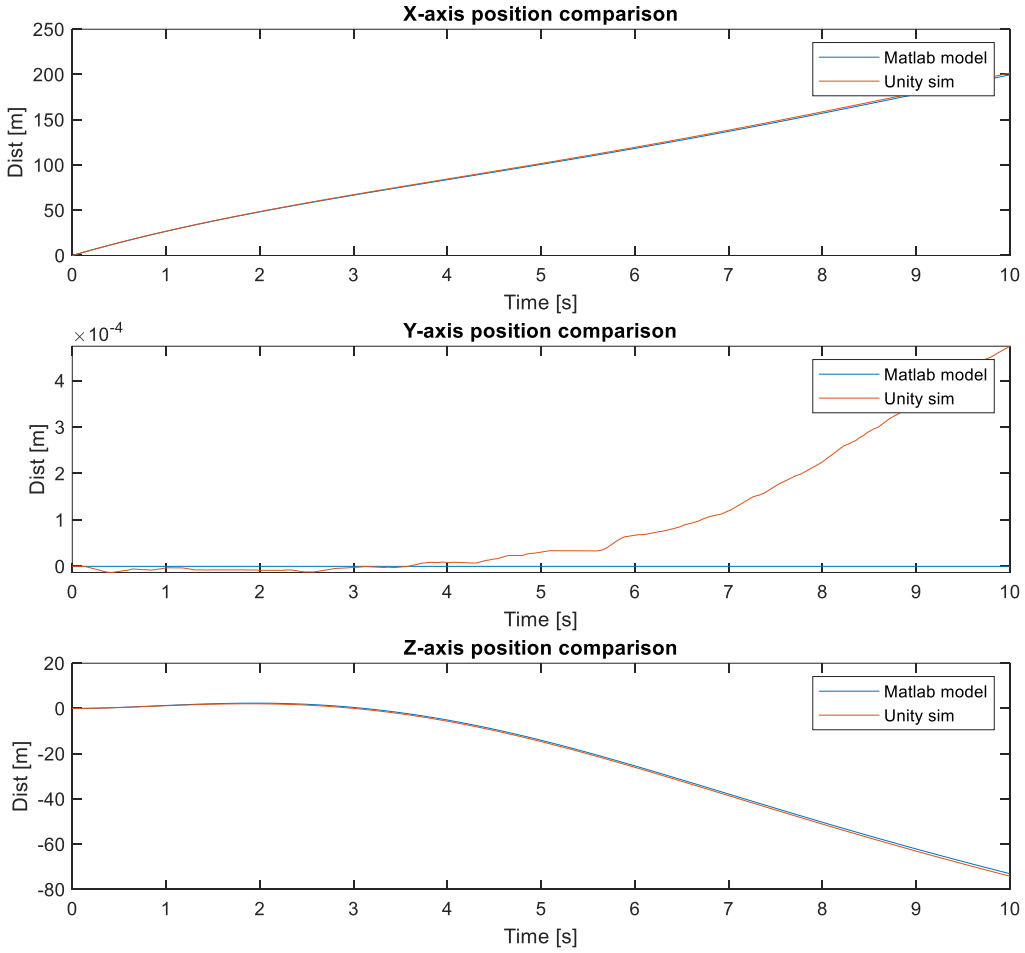

Figure 25: Data comparison - position

In the first figure (corresponding with the x-axis – forward direction) the value increases over time – aircraft is moving forward. In the third figure (corresponding with the z-axis – upward direction) the value first increased and then quickly starts dropping – the aircraft ascends slightly but as soon as the speed is dropped it starts losing altitude and gliding down. The only visible difference in the data is in the seconds figure. The difference is caused by a floating point inaccuracies in Unity engine, but the scale of this plot is 10^{-4}, therefore the effect is minimal and for this application negligible. Aircraft doesn't move left or right.

The comparison on the global orientation in Euler angles of the rigidbody in MATLAB model and Unity simulation is possible from another three figures below (one figure for each axis):

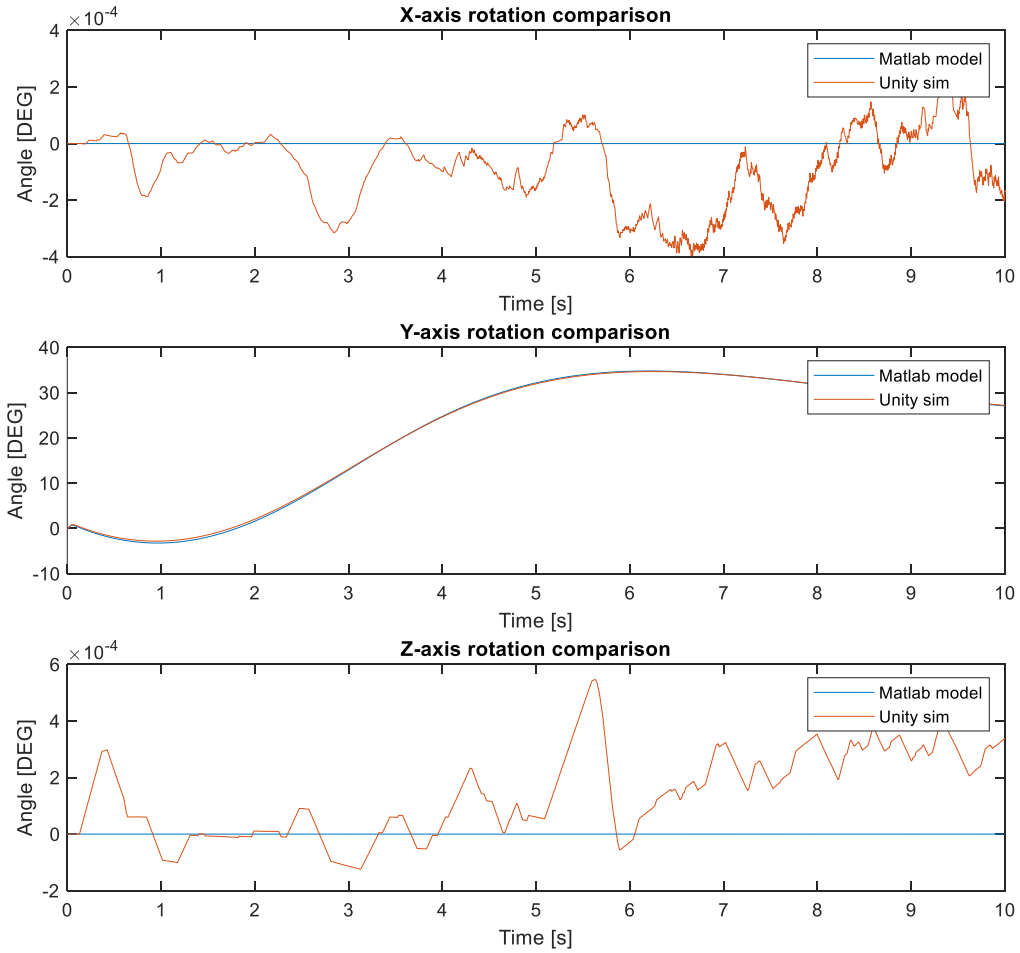

Figure 26: Data comparison - rotation

In the first figure (corresponding with the y-axis – pitch). Aircraft first pitches up (negative values) and then pitches down (and starts losing altitude). The inaccuracies in the first and the third figure are again cause by floating point math in Unity3D. But the scale of this plot is 10^{-4}, therefore the effect is minimal and for this application again negligible.

CHAPTER 6: LINEAR MODEL

6. 1 INTRODUCTION

In real life, most of the processes we observe have a non-linear characteristic. In order to be able to analyze them in a simpler way and use modern controller design techniques, non-linear model must be substituted with a linearized model around an operating point. The linearized model behaves the same (or very similar) around this point and offers controller design options that would not be possible with the non-linearized model.

In general, we have a following non-linear system in a following form:

$$\dot{x}(t) = f(x(t), u(t))$$

$$y(t) = h(x(t), u(t))$$

Where x is its state at time t, y is its output and u is its input.

First, we need to specify a an operating point of the system around which the process of linearization will be performed. We set $\dot{x}(t) = 0$, $x(t) = x_r$ and the input and output is constant. With that in mind, equations above can be rewritten into a following form [8]:

$$\dot{x}(t) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} x + \begin{pmatrix} \frac{\partial f_1}{\partial u_1} & \cdots & \frac{\partial f_1}{\partial u_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1} & \cdots & \frac{\partial f_n}{\partial u_m} \end{pmatrix} u$$

$$y(t) = \begin{pmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_p}{\partial x_1} & \cdots & \frac{\partial f_p}{\partial x_n} \end{pmatrix} x + \begin{pmatrix} \frac{\partial h_1}{\partial u_1} & \cdots & \frac{\partial h_1}{\partial u_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_p}{\partial u_1} & \cdots & \frac{\partial h_p}{\partial u_m} \end{pmatrix} u$$

Where the derivatives are taken at the equilibrium point. This can be rewritten using a state space representation:

$$\dot{x}(t) = Ax + Bu$$

$$y(t) = Cx + Du$$

6. 2 LINEARIZATION USING MATLAB

In case the mathematical equations are more complicated linearization tool which is part of MATLAB system control toolbox. It uses a similar approach as described above but is solves it numerically. For this to work, the target non-linear system must be present in Simulink and open loop input and output correctly chosen. The non-linear system has multiple inputs and multiple inputs. The goal of this thesis is to derive 3 SISO linear systems which represent behavior of the aircraft in pitch and roll axis and also the velocity of the aircraft. In the future, this could be improved by obtaining MIMO linear system in order to design more robust and intelligent controller considering more complex inner interactions inside of the model.

6. 2. 1 PITCH MODEL

The input of this model is a left and right elevator input (cs3 and cs4). The output is the value of pitch (angular position around y axis). Values for the other inputs are set to zero with the exception of throttle in order for the aircraft not to stall. Model layout for linearization is shown on the figure below:

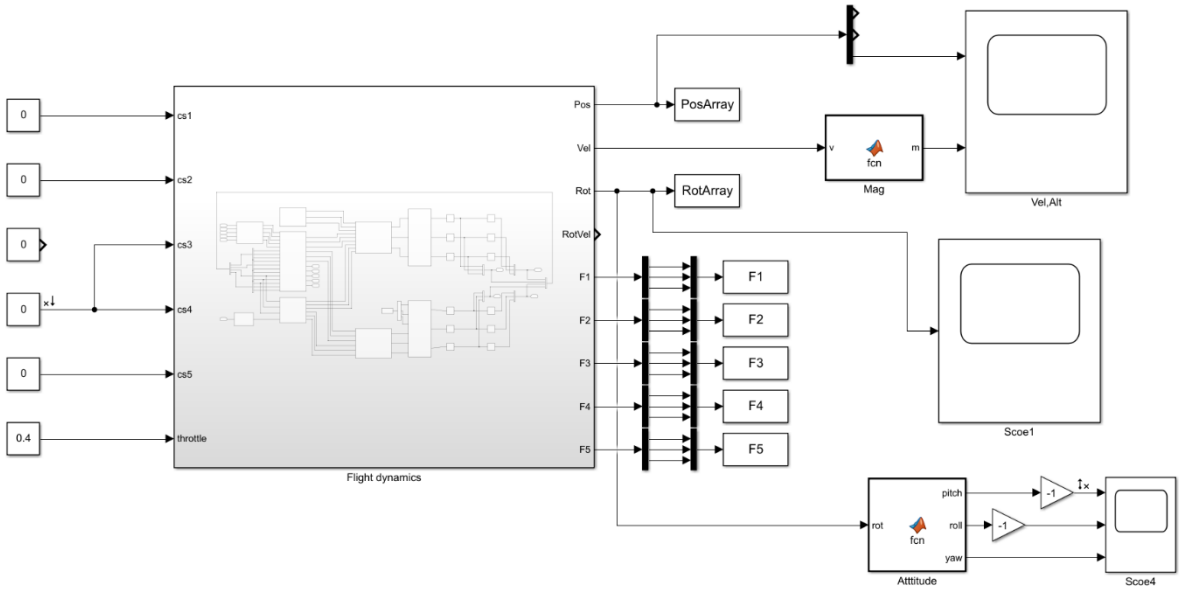

Figure 27: Model schematics in Simulink

Using MATLAB Linear Analysis tool, following step response corresponding to the pitch behavior can be obtained (aircraft is stable in pitch):

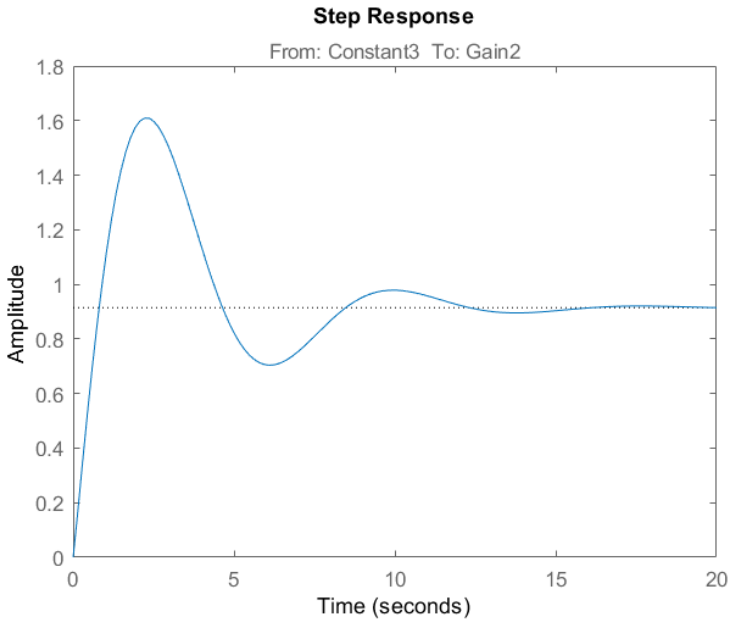

Figure 28: Pitch model step response

6. 2. 2 ROLL MODEL

The input of this model is also left and right elevator input (cs3 and cs4), but in order to control roll, following must true: $cs3 = -cs4$. At the end, the overall input to cs3 and cs4 will be combination from pitch and roll controller. The output of the roll model is the value of roll (angular position around x axis). Values for the other inputs are set to zero with the exception of throttle for the same reason as with pitch model. Model layout for linearization is shown on the figure below:

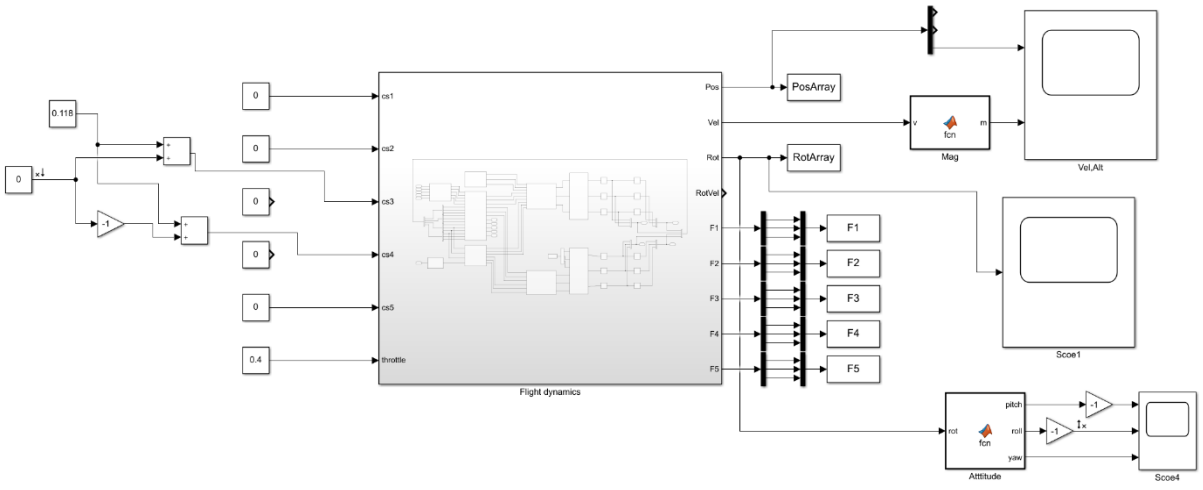

Figure 29: Model schematics in Simulink

Again, using MATLAB Linear Analysis tool, following step response corresponding to the roll behavior can be obtained (aircraft is unstable in pitch):

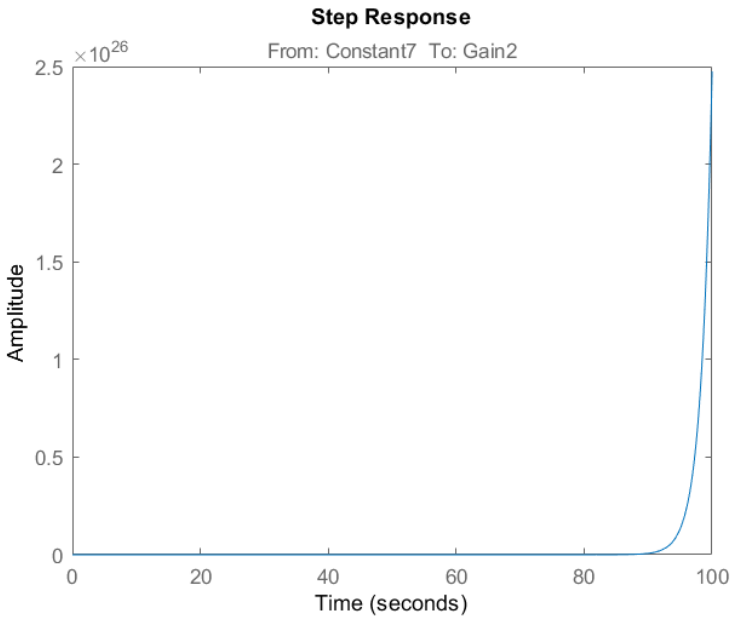

Figure 30: Roll model step response

6. 2. 3 VELOCITY/THROTTLE MODEL

The input of this model is the throttle of the aircraft. The output is the magnitude of the velocity vector. Values for the other inputs are set to zero with the exception of cs3 and cs4. These inputs are set to the stable value using which the aircraft pitch angle is approximately 0. Model layout for linearization is shown on the figure below:

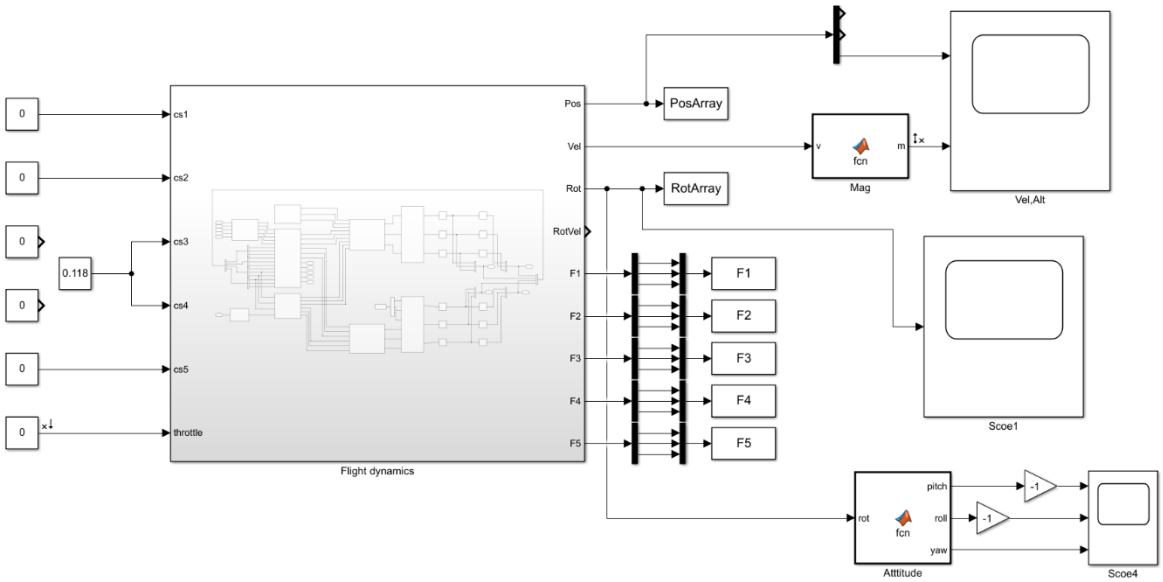

Figure 31: Model schematics in Simulink

Again, using MATLAB Linear Analysis tool, following step response corresponding to the behavior of velocity can be obtained (aircraft is unstable in pitch):

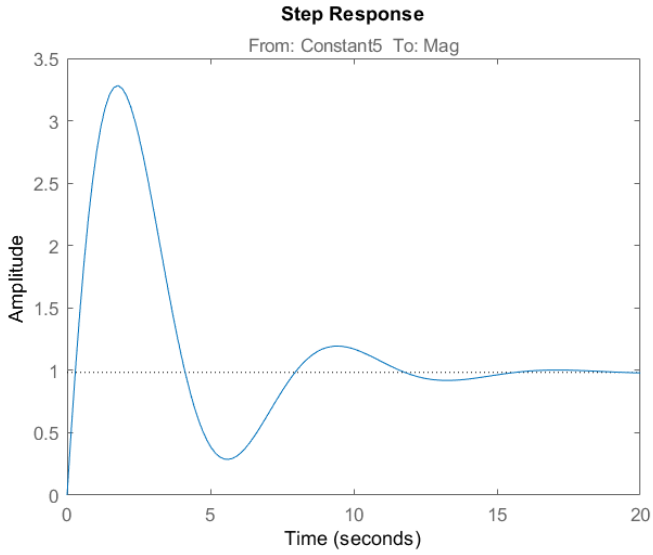

Figure 32: Velocity model step response

CHAPTER 7: CONTROLLER DESIGN

Using the linear models derived above, controllers for pitch, roll and throttle can be obtained. In order to design each controller, SISOTOOL, which is part of MATLAB can be used. This tool allows to use controller design methods such as loop-shaping, which is beneficial in this case. This method allows directly changing the frequency characteristics of the important transfer functions. The two most important once are the sensitivity function and complementary sensitivity function. In an ideal case, the value of the sensitivity function would be zero, in order to suppress output disturbances. Unfortunately, due to the Bode's sensitivity integral, this cannot be achieved and optimal solution based on the current problems characteristics must be used. The complementary sensitivity function is useful to ensure controller precision (lower frequencies) and based on its maximum amplitude, resonant frequencies must be taken into consideration.

The controller is obtained in a form of a transfer function. It can be transformed into the form if a PID controller with its coefficients. These coefficients can be entered into the Simulink model or later into the onboard controller.

7. 1 PITCH CONTROLLER

SISOTOOL allows to design following controller:

$$C_{pitch} = 2.4613 * \frac{(1 + 1.4s)(1 + 0.33s)}{s(1 + 0.01s)}$$

Root locus and step response with this controller are shown on the figure below:

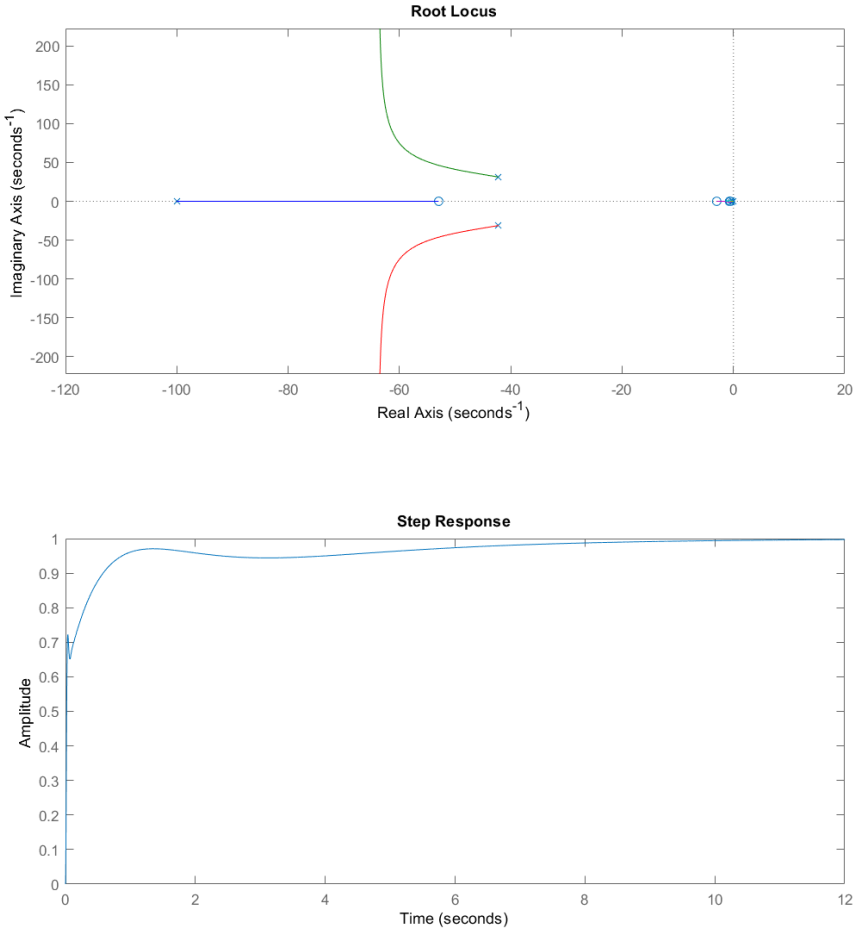

Figure 33: Pitch controller design

Using Simulink, linear and non-linear model behavior inside of the closed loop with a corresponding controller can be compared. For that, following Simulink schematics can be used:

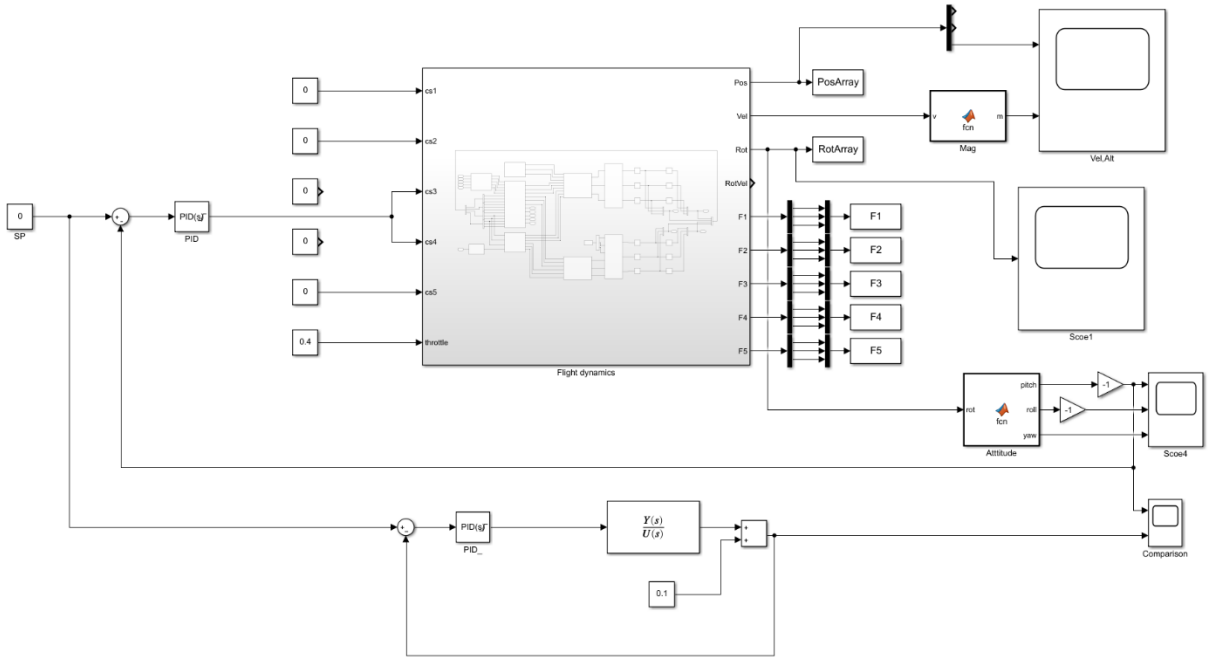

Figure 34: Model schematics in Simulink

The comparison itself is shown in a figure below:

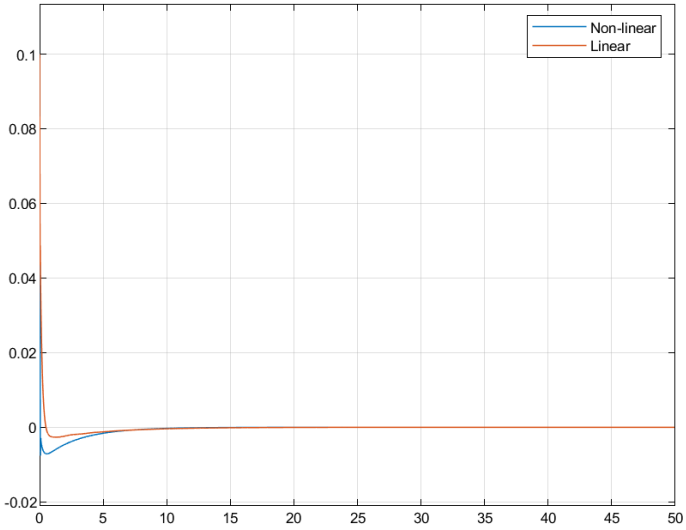

Figure 35: Pitch model comparison

7. 2 ROLL CONTROLLER

SISOTOOL allows to design following controller:

$$C_{pitch} = 0.3 * \frac{(1+2s)(1+0.16s)}{s(1+0.01s)}$$

Root locus and step response with this controller are shown on the figure below:

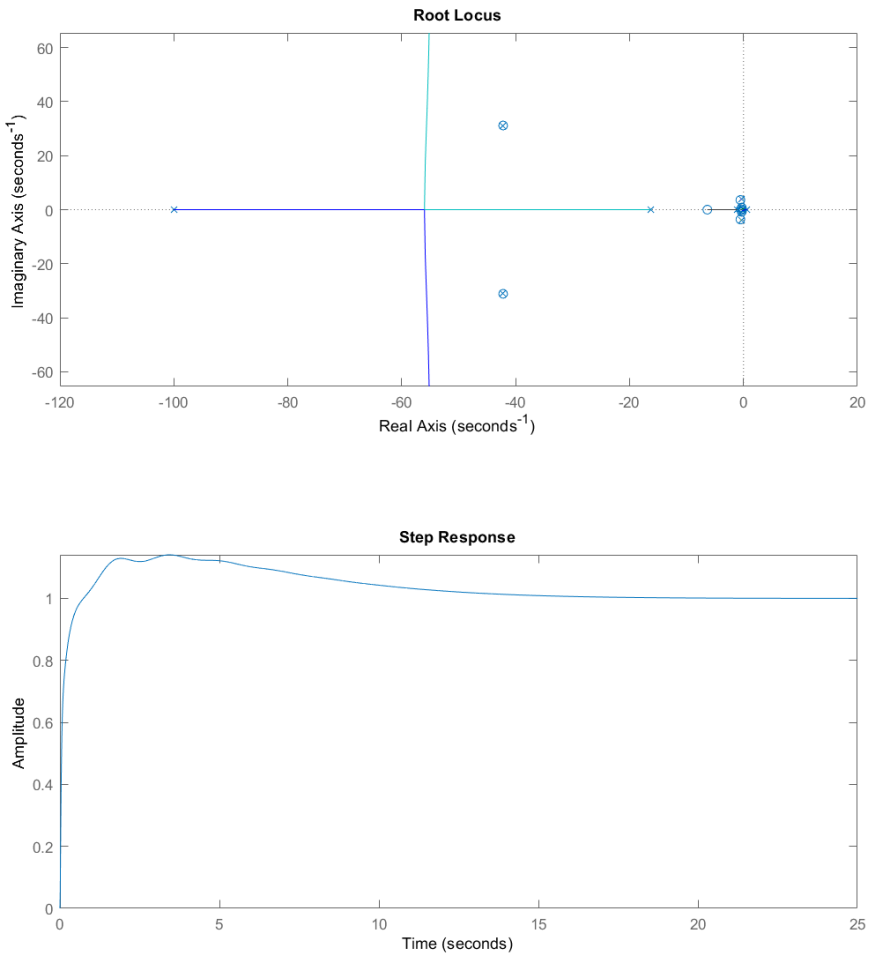

Figure 36: Roll controller design

Using Simulink, linear and non-linear model behavior inside of the closed loop with a corresponding controller can be compared. For that, following Simulink schematics can be used:

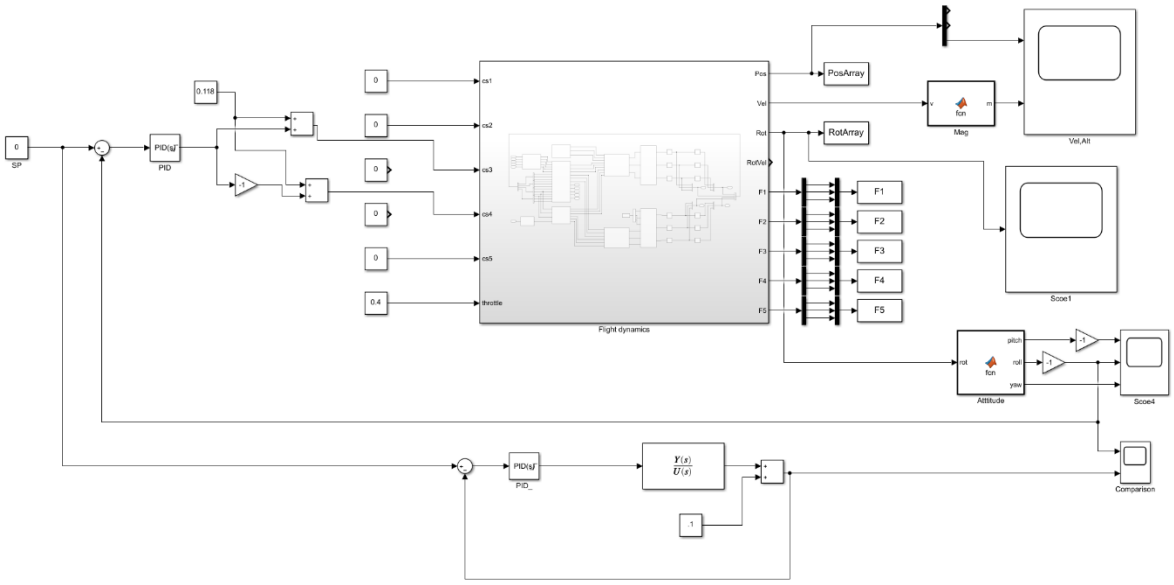

Figure 37: Model schematics in Simulink

The comparison itself is shown in a figure below:

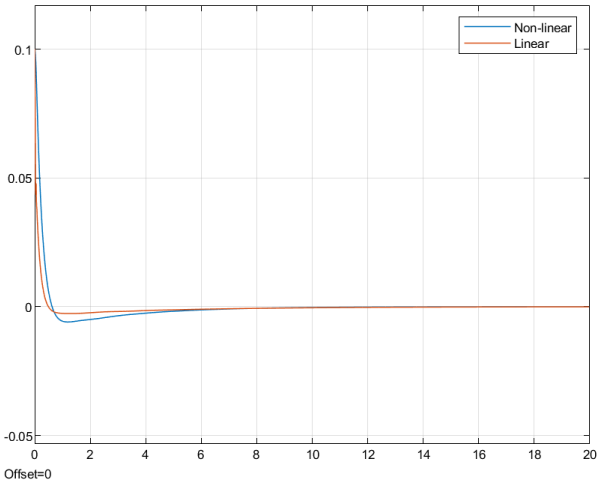

Figure 38: Roll model comparison

7. 3 VELOCITY CONTROLLER

SISOTOOL allows to design following controller:

$$C_{pitch} = 1.977 * \frac{(1 + 0.031s)(1 + 0.74s)}{s(1 + 0.01s)}$$

Root locus and step response with this controller are shown on the figure below:

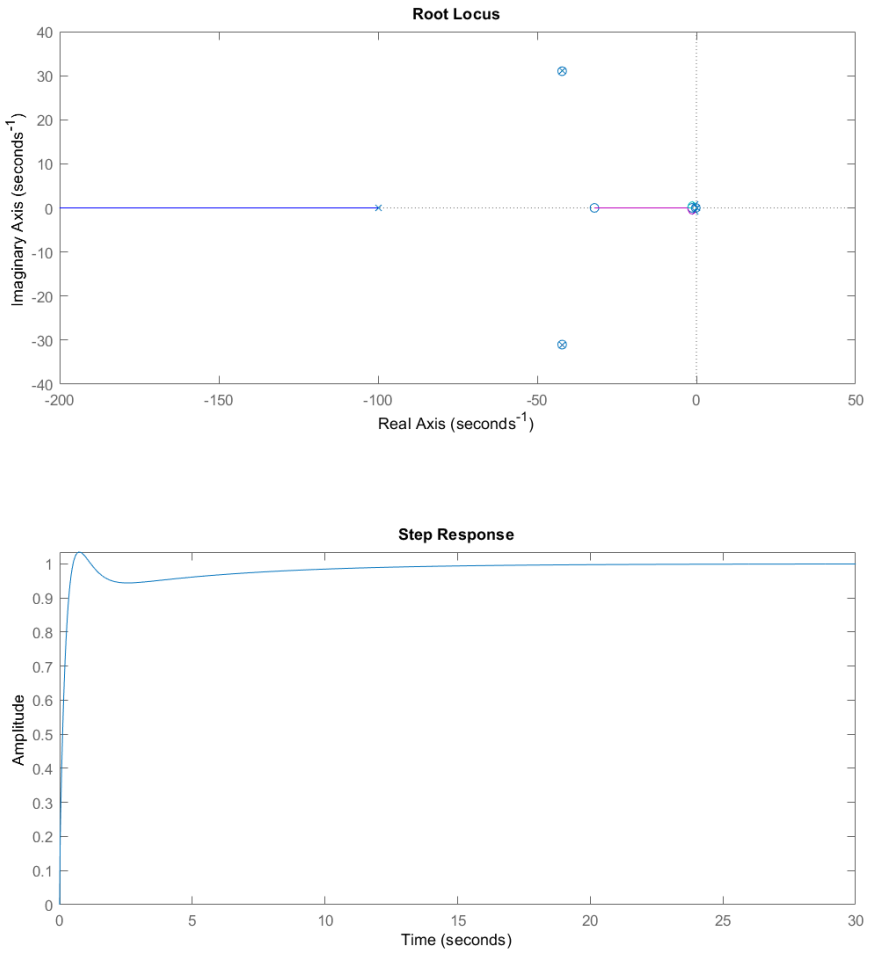

Figure 39: Velocity controller design

Using Simulink, linear and non-linear model behavior inside of the closed loop with a corresponding controller can be compared. For that, following Simulink schematics can be used:

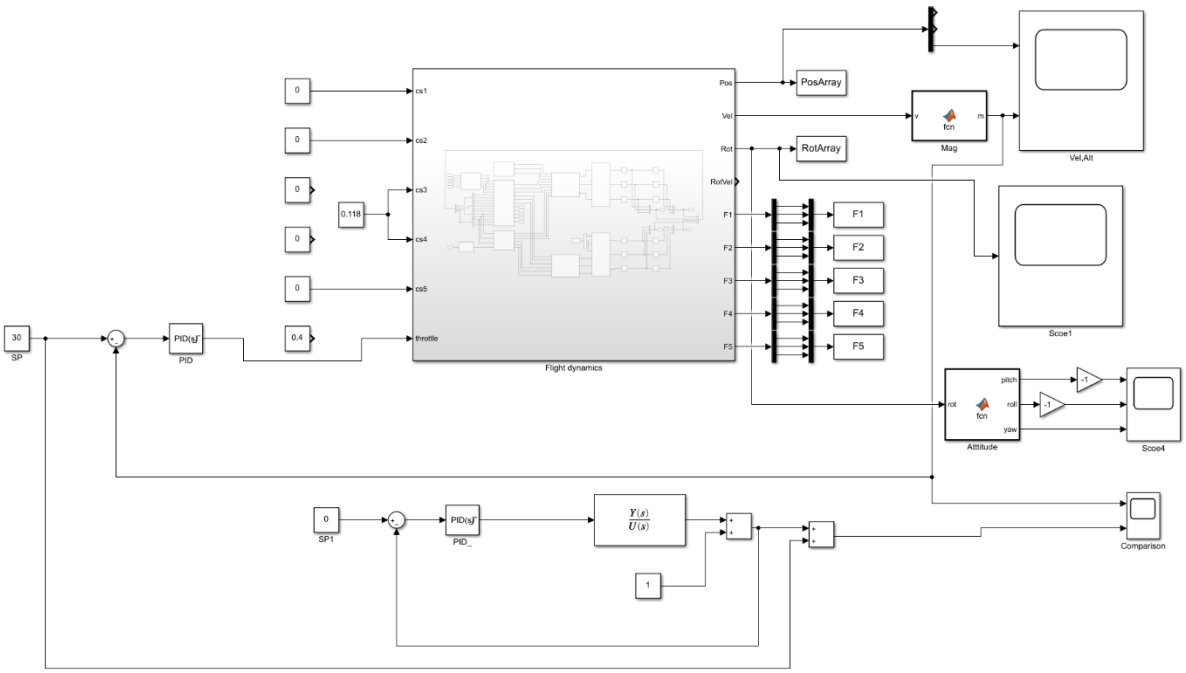

Figure 40: Model schematics in Simulink

The comparison itself is shown in a figure below:

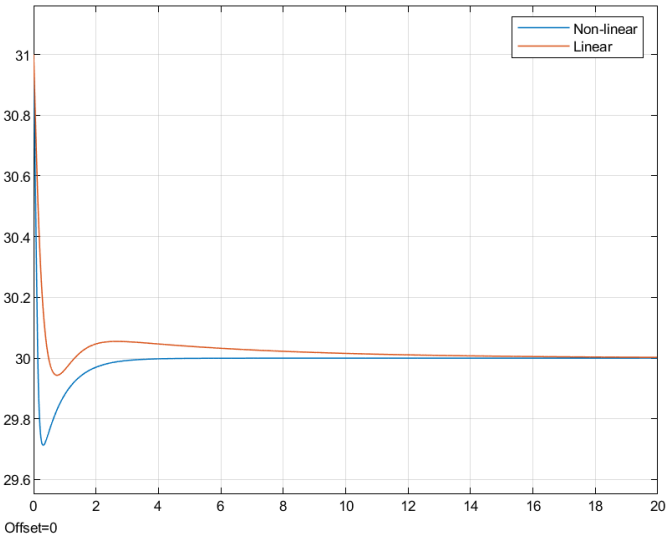

Figure 41: Velocity model comparison

7. 4 TEST WITH ALL 3 CONTROLLERS

To verify the functionality of all three controllers and the same time and to ensure that the inner interactions will not significantly influence the behavior of the system inside of the closed loop, following schematic is used. It includes all 3 controllers (for pitch, roll and velocity).

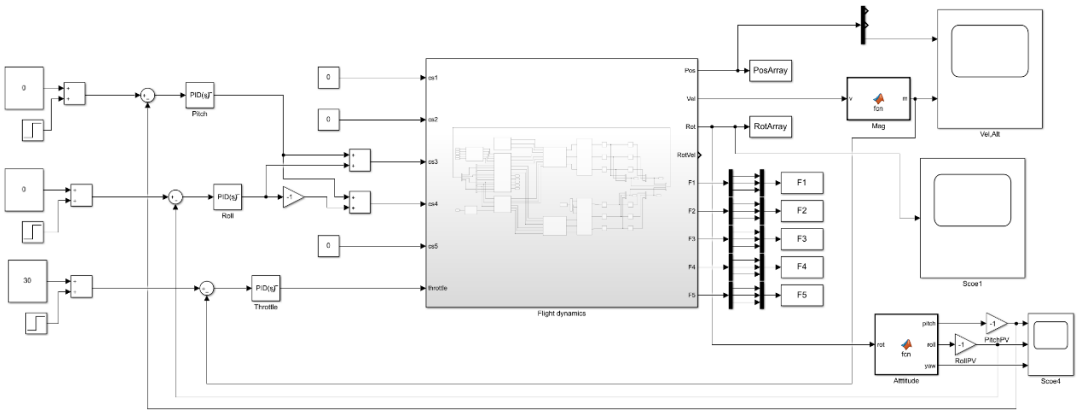

Figure 42: Model schematics in Simulink

Initially, model starts with a slight offset from the operating point and at time of 20s, the setpoints for the PID controllers are changed to verify the stability. Following figure show values of the pitch and roll angle in time:

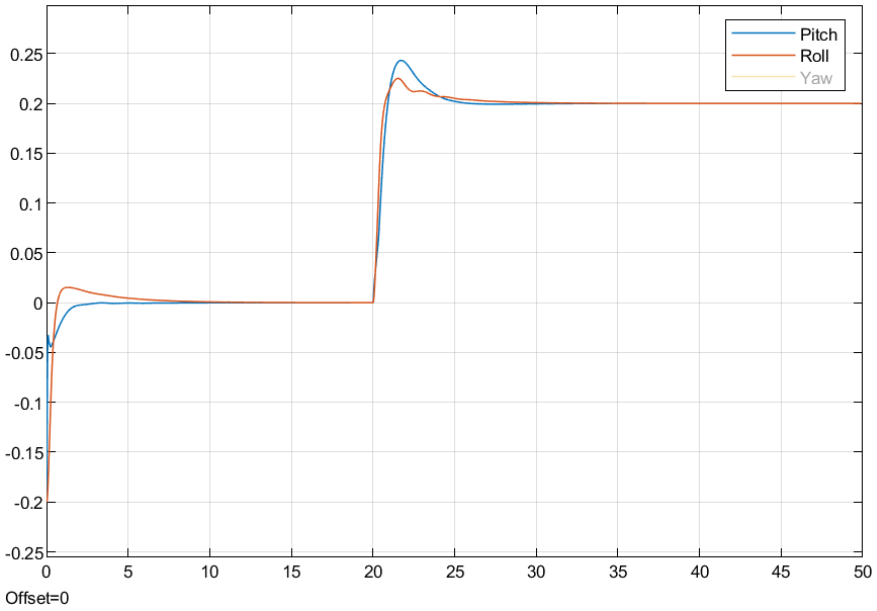

Figure 43: Pitch and roll process values

The next figure show the value of the velocity vector magnitude. (It also displays the value of z linear position which corresponds to altitude. With higher pitch angle and higher velocity, the aircraft ascends much faster.)

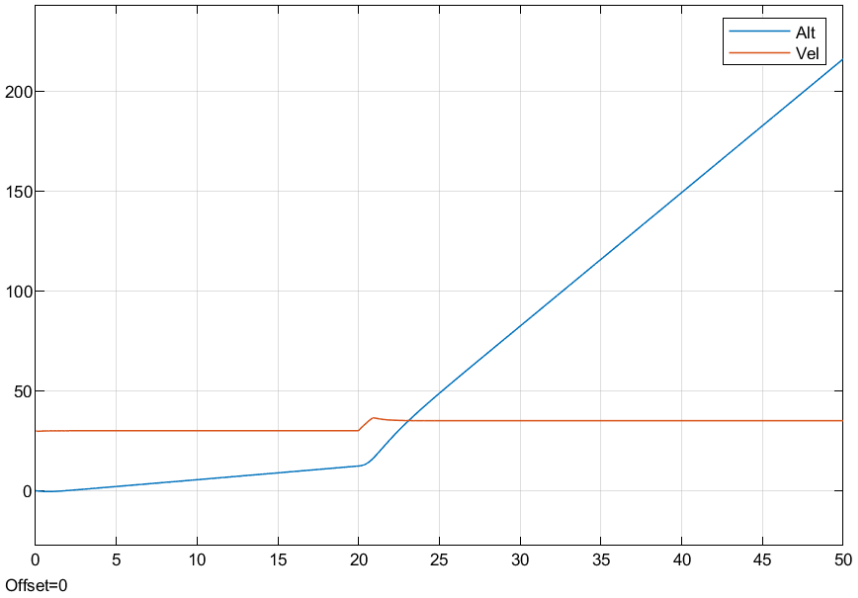

Figure 44: Velocity process value

7. 5 HARDWARE IN THE LOOP TEST

The last verification of the obtained controllers can be performed in the Unity simulation with onboard MCU connected in the simulation mode. Aircraft undergoes series of pitch, roll and velocity setpoint changes. The attitude and speed of the aircraft is recorded into a csv file which can we viewed in MATLAB. This test is affected by the real data transfer speed between the simulation and the OC and the clock speed of the onboard controller itself. Based on the IDE used to debug the OC, the period with which the data is obtained and calculated with the current MCU is more that 0.1s, that corresponds to a frequency less than 10 per second which is quite slow. This could be easily solved by using a faster MCU as suggested above. Following figure shows the behavior of the aircraft during this test:

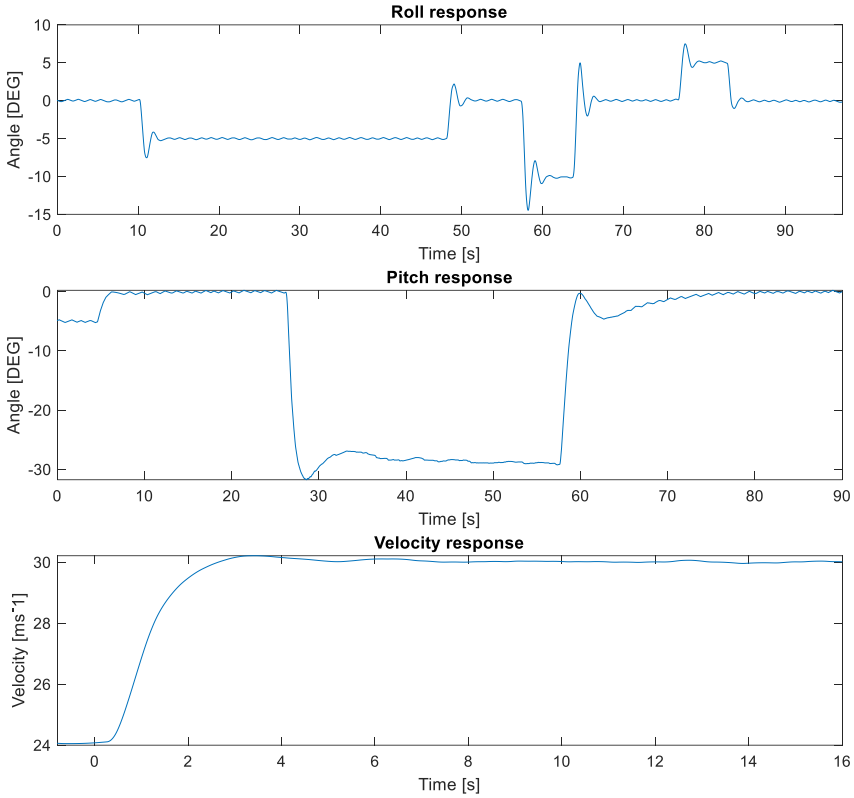

Figure 45: Unity flight simulation process value data

Responses differ from the mathematical model, that is caused by the limited data transfer speed from the simulation to the OC and from the OC to the simulation and by the limited processing power of the OC. Even though the real world limitations, aircraft is stable and is able to follow commands given by the user using the ground control app.

CHAPTER 8: CONCLUSION

The onboard computer designed for this purpose works as required. It is able to control the model in the flight simulation and process data obtained by the virtual sensors. The ground control application is capable of displaying crucial flight telemetry data and control the aircraft based on the user needs.

Developed mathematical model allows to simulate required geometry of the aircraft. It can be used to precisely simulate different flight conditions and to obtain linearized model in order to design a controller. Mathematical model was verified using Unity3D built in physics system as a reference and showed to be even more precise. (Unity3D as a game engine uses floating point math.)

In order to sustain a stable flight, 3 PID controllers were designed. This was done using the mathematical model of the aircraft and the tools provided by MATLAB. Hardware in the loop simulation was used to verify the functionality of the autopilot as a whole. Aircraft is able to sustain a stable flight and responds to the input given by user via the ground control app.

In the future, the on board controller could be improved by using a faster and more robust MCU as a master instead of the current NodeMCU allowing it to perform computation in the autopilot mode with a higher frequency. Also, custom PCB design would be necessarily before deploying in a real life UAV. In order to ensure higher robustness, instead of 3 SISO models it would be beneficial to develop 1 MIMO model increasing the model precision and allowing use of mode advanced control design techniques.

CHAPTER 9: REFERENCE

1. Teensy USB Development Board. [online] [2020-3-22, 22:04 UTC+01:00] Available at: pjrc.com/teensy/
2. RC transmitter [online] [2020-4-06, 01:23 UTC+01:00] https://www.banggood.com/cs/RadioMaster-TX16S-Hall-Sensor-Gimbals-2_4G-16CH-Multi-protocol-RF-System-OpenTX-Mode2-Transmitter-for-RC-Drone-p-1652191.html?gpla=1&gmcCountry=CZ&currency=CZK&createTmp=1&utm_source=googleshopping&utm_medium=cpc_bgs&utm_content=lijing&utm_campaign=ssc-cz-toys-0313&ad_id=425205428295&gclid=CjwKCAjwk6P2BRAIEiwAfVJ0rHTzfOahZe6y1sg2ngYOTglBDO--F3Nkd1lketEtRlq6JzDmHuZ43RoCORUQAvD_BwE&ID=424826287800&cur_warehouse=CN - radio transmitter photos
3. RC receiver [online] [2020-4-15, 23:41 UTC+01:00] https://www.banggood.com/cs/DUMBORC-X6FG-2_4G-6CH-Receiver-with-Gyro-for-RC-X6-Transmitter-Remote-Controller-p-1520829.html?rmmds=search&p=X01101427485842019l2&custlinkid=723808&cur_warehouse=CN
4. Arduino [online] [2020-4-18, 22:45 UTC+01:00] https://www.vokolo.cz/arduino-leonardo-pro-micro-atmega32u4/?variantId=31617
5. Node MCU [online] [2020-4-23, 03:57 UTC+01:00] https://www.banggood.com/cs/Geekcreit-NodeMcu-Lua-WIFI-Internet-Things-Development-Board-Based-ESP8266-CP2102-Wireless-Module-p-1097112.html?gpla=1&gmcCountry=CZ&currency=CZK&createTmp=1&utm_source=googleshopping&utm_medium=cpc_bgs&utm_content=lijing&utm_campaign=ssc-cz-usw-all-0313&ad_id=425206122393&gclid=CjwKCAjwk6P2BRAIEiwAfVJ0rDcry4jYvrJ1w9XYux15ByY76LydocSHyHPN7fUQLkGlFZ8opv7OWhoC_bcQAvD_BwE&cur_warehouse=CN
6. IMU [online] [2020-4-30, 16:19 UTC+01:00] https://www.laskarduino.cz/arduino-9dof-gyroskop-akcelerometr-magnetometr-mpu-9250-spi-iic/?gclid=CjwKCAjwk6P2BRAIEiwAfVJ0rGVJ5tru5LtpYa-341ammPg32MSpjJeDcYofIv6vwHhbtUfONcVH_hoC4u0QAvD_BwE
7. Pressure sensor [online] [2020-5-01, 02:52 UTC+01:00] https://www.banggood.com/cs/CJMCU-388-BMP388-Digital-Temperature-and-Atmospheric-Pressure-Sensor-with-Low-Power-Consumption-24-Bits-Low-Noise-p-1470290.html?gpla=1&gmcCountry=CZ&currency=CZK&cur_warehouse=CN&createTmp=1&utm_source=googleshopping&utm_medium=cpc_bgs&utm_content=lijing&utm_campaign=ssc-cz-usw-all-0313&ad_id=425206122393&gclid=CjwKCAjwk6P2BRAIEiwAfVJ0rGyVK4I3YQwkrHZ9zySG-vulH2-LJvqXiesuB_6Nap6MJ-80oMrQQBoCucAQAvD_BwE
8. GOUBEJ, Martin, MELICHAR, Jiří. LINEÁRNÍ SYSTÉMY 1. Plzeň, 2017. (Učební text) Západočeská univerzita v Plzni. Fakulta aplikovaných věd, KKY

CHAPTER 10: LIST OF FIGURES

CHAPTER 11: LIST OF TABLES