# A short overview about what I managed to do

Rus Ioan-Vasile

May 18, 2024

# About

Firstly, I thought I would manage to do a bit more, but unfortunately, this semester at university was a little bit time-consuming.

In a big picture this is what I have to show you:

- a stabilized platform using a gyroscope and 4 Stepper motors, of course with Arduino as you suggest.
- finding a controller for a VTOL(Vertical Take-off and Landing), this project was unexpected, but well welcomed.
- some basic circuits using different components(Relays, transistors,capacitors etc.)
- readings from an audio sensor

I want to apologize for any possible errors I made while writing in English.

# Bibliography

In special for the arduino, I read a book about it:

Arduino Programming in 24 Hours Richard Blum Softarchive Net

For the VTOL, I used the materials provided by Miss Cristina Muresean, who is my course teacher for Control Engineering II.

The basic circuits, which I was referring to were made by following the book Make:Electronics by Charles Platt (I've only gone through half of the book).

# Contents


**1 Stabilized Platform** **4**
- 1.1 Description . . . 5
- 1.2 Pictures . . . 6
- 1.3 Code . . . 7
- 1.4 Conclusion . . . 12

**2 VTOL-Vertical take-off and landing** **13**
- 2.1 Description . . . 14
  - 2.1.1 Identification . . . 14
- 2.2 Controller-Gain and Phase margin . . . 17
  - 2.2.1 About . . . 17
  - 2.2.2 Formulas . . . 18
  - 2.2.3 Code . . . 19
  - 2.2.4 Results . . . 20
- 2.3 Controller-Vogel-Edgar . . . 21
  - 2.3.1 About . . . 21
  - 2.3.2 Formulas . . . 23
  - 2.3.3 Code . . . 24
  - 2.3.4 Results . . . 25
- 2.4 Conclusions . . . 26

**3 Basic circuits** **27**
- 3.1 Description . . . 27
- 3.2 Pictures . . . 28

**4 What I learn this Semester** **29**

# Chapter 1

# Stabilized Platform

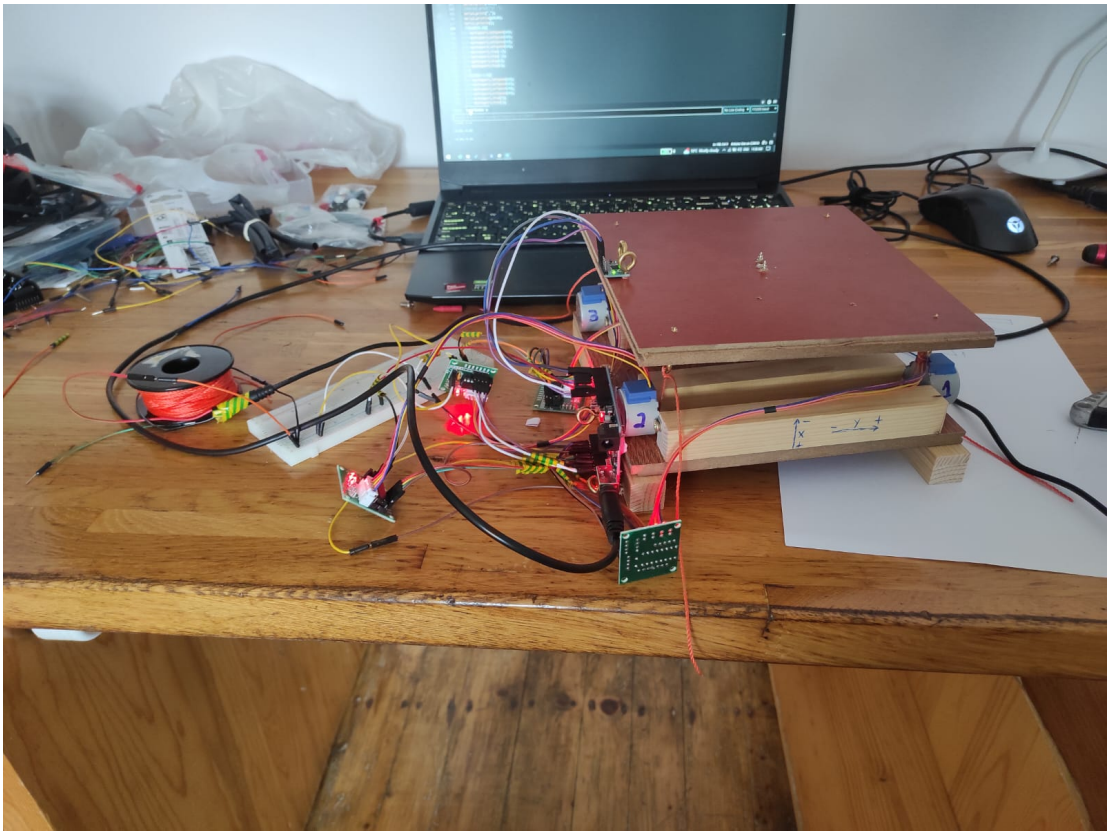


Figure 1.1: Stabilized Platform

## 1.1 Description

I choose this type of project because includes data acquisition from a sensor, usage of 4 motors and also involves designing a controller.

I have upload a video-clip on GitHub, for exemplification, there are all the images, videos, documents, books for the projects.

The link is here: https://github.com/ioanrus/Different-projects.

Components:

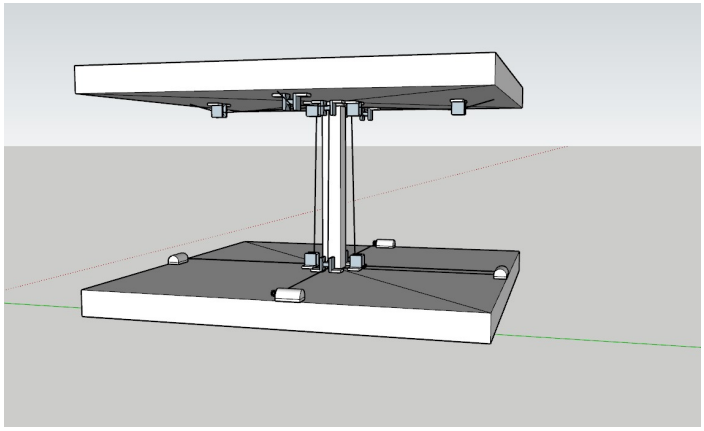
- Arduino Uno.
- MPU6050 gyroscope and accelerometer.
- 4 Step Motors 28BYJ-48 5V DC.

The platform is made of two square wood pieces. The top platform is the stabilized one and the one from below is the rigid part.

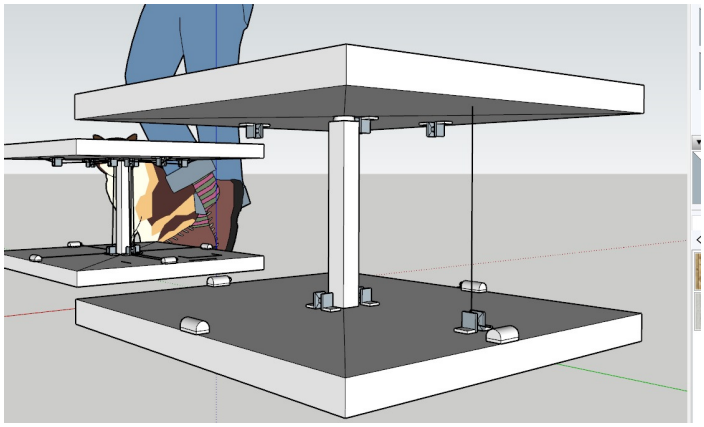
The reference angle for the platform is 0°. Before starting with implementation I made a sketch to show my Dad if is possible to implement it with our tools/materials.

## 1.2 Pictures

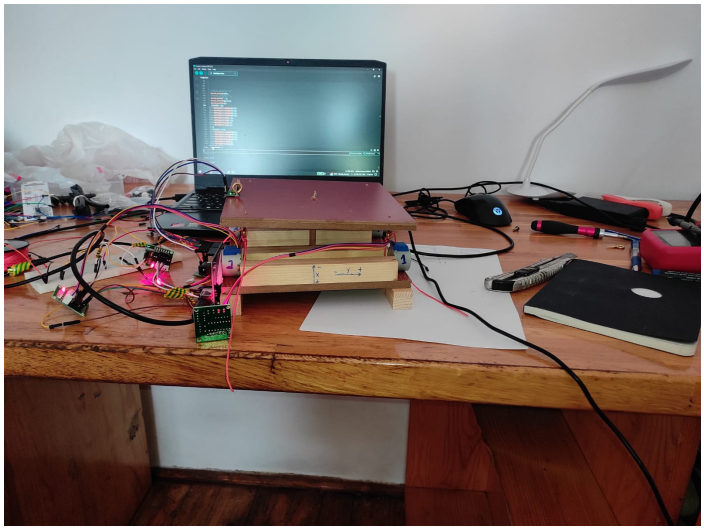
Demonstrative pictures:



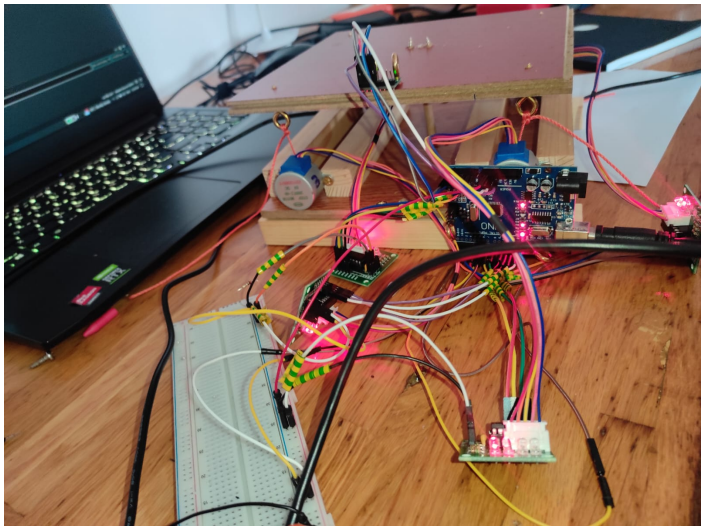
(a) First concept, to many parts.



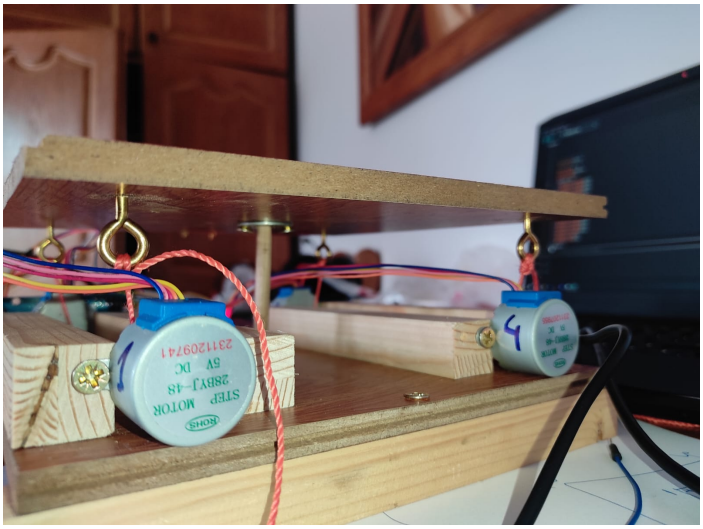
(b) Second concept, you can see in the middle the spherical joint.



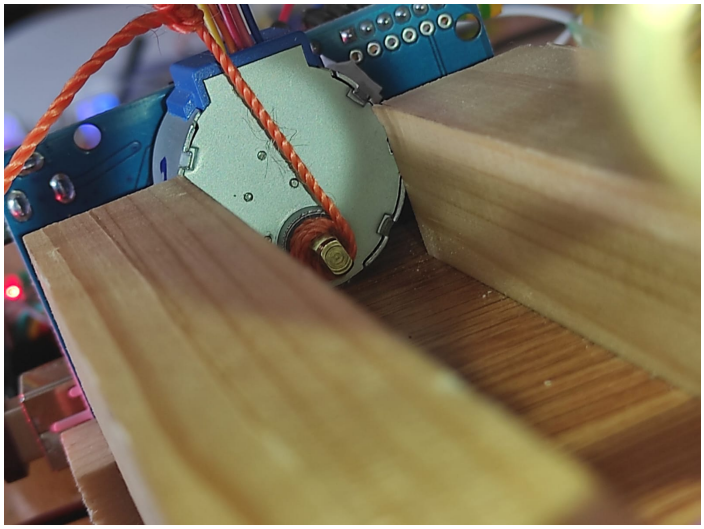
(c) A big overview.



(d) Cable connection.



(e) Final concept, the ideal and simplest.



(f) I drill a hole in the shaft and made a knot.

Figure 1.2: Check the video I uploaded on gitHub.

## 1.3 Code

```
#include <MPU6050.h>
#include <Wire.h>
#include <Stepper.h>

// Define step for our motor type
const int stepsPerRevolution = 2038;

// Set the pins for SpeerMotor
Stepper myStepper3(stepsPerRevolution, 10, 11, 12, 13);
Stepper myStepper2(stepsPerRevolution, 6, 7, 8, 9);
Stepper myStepper1(stepsPerRevolution, 2, 3, 4, 5);
Stepper myStepper4(stepsPerRevolution, A0, A1, A2, A3);

// MPU6050 I2C address
const int MPU = 0x68;
float AccX, AccY, AccZ;
float GyroX, GyroY, GyroZ;
float accAngleX, accAngleY, gyroAngleX, gyroAngleY, gyroAngleZ;
float roll, pitch;
float AccErrorX, AccErrorY, GyroErrorX, GyroErrorY, GyroErrorZ;
float elapsedTime, currentTime, previousTime;
int c = 0;

void setup() {
  Serial.begin(115200);
  Wire.begin();
  // Initialize comunication
  // Start communication with MPU6050 at 0x68
  Wire.beginTransmission(MPU);
  // acces the register
  Wire.write(0x6B);
  // make 0 in the register for reseting the mpu6050
  Wire.write(0x00);
  // close the transmision
  Wire.endTransmission(true);

  // we configure the sensivity of our sensor

  // Configure Accelerometer Sensitivity
  Wire.beginTransmission(MPU);
  // make connection with the register 1C witch take care of full
    scale range, bit 4-3=00(+-2g),4-3=01(+-4g),4-3=10(+-8g)
    ,4-3=11(+-16g)
  Wire.write(0x1C);
  // in our case we set +-8g, 00010000
  Wire.write(0x10);
  Wire.endTransmission(true);


  // Configure Gyro Sensitivity
  Wire.beginTransmission(MPU);
  // make connection with the register 1B witch take care of full
    scale range, bit 4-3=00(+-250deg/s),4-3=01(+-500deg/s)
    ,4-3=10(+-1000deg/s),4-3=11(+-2000deg/s)
```

```
  Wire.write(0x1B);
  // in our case we set +-1000deg/, 00010000
  Wire.write(0x10);
  Wire.endTransmission(true);

  // this function is used to calibrate the sensor
  // delay(20);
  // calculate_IMU_error();
  // delay(20);
}
void loop() {
  // === Read acceleromter data ===
  Wire.beginTransmission(MPU);
  Wire.write(0x3B); // Start with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 6 registers total, each
    axis value is stored in 2 registers
  //For a range of +-8g, we need to divide the raw values by 4096,
    according to the datasheet
  // X-axis value
  AccX = (Wire.read() << 8 | Wire.read()) / 4096.0;
  // Y-axis value
  AccY = (Wire.read() << 8 | Wire.read()) / 4096.0;
  // Z- axix value
  AccZ = (Wire.read() << 8 | Wire.read()) / 4096.0 ;

  // Calculating Roll and Pitch from the accelerometer data
  //error 0.61
  accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 /
    PI) - 0.61;
  //error 1.42
  accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) *
    180 / PI) + 1.42;

  // === Read gyroscope data ===
  previousTime = currentTime;
  currentTime = millis();
  elapsedTime = (currentTime - previousTime) / 1000; // we convert to
    seconds

  //read from gyro register
  Wire.beginTransmission(MPU);
  Wire.write(0x43);
  Wire.endTransmission(false);
  //each axis value is stored in 2 registers
  Wire.requestFrom(MPU, 4, true);
  GyroX = (Wire.read() << 8 | Wire.read()) / 32.8; // For a +-1000deg
    /s range we have to divide first the raw value by 32.8, according
    to the datasheet
  GyroY = (Wire.read() << 8 | Wire.read()) / 32.8;

  // we take in consideration de error
  GyroX = GyroX + 0.7;
  GyroY = GyroY - 1.52;

  // we multyp with the time to get degreese
```

```
100  gyroAngleX = gyroAngleX + GyroX * elapsedTime;
101  gyroAngleY = gyroAngleY + GyroY * elapsedTime;
102
103  // on long term the gyroscopes drifts so we need to adjust it with
       accelometer
104  gyroAngleX = 0.96 * gyroAngleX + 0.04 * accAngleX;
105  gyroAngleY = 0.96 * gyroAngleY + 0.04 * accAngleY;
106
107  roll = gyroAngleX;
108  pitch = gyroAngleY;
109
110
111  // the motors that need to pull the string have a greater speed,
       because if not the string will go out of the saft
112
113
114  Serial.print(roll);
115  Serial.print(",");
116  Serial.println(pitch);
117  Serial.println();
118  if(roll>1.0){
119     myStepper1.setSpeed(40);
120     myStepper2.setSpeed(40);
121     myStepper3.setSpeed(60);
122     myStepper4.setSpeed(60);
123     myStepper1.step(-1);
124     myStepper2.step(-1);
125     myStepper3.step(1);
126     myStepper4.step(1);
127   }
128  if(roll<-1.0){
129     myStepper1.setSpeed(60);
130     myStepper2.setSpeed(60);
131     myStepper3.setSpeed(40);
132     myStepper4.setSpeed(40);
133     myStepper1.step(1);
134     myStepper2.step(1);
135     myStepper3.step(-1);
136     myStepper4.step(-1);
137   }
138    if(pitch>1 && (roll>-1.0 && roll<1) ){
139     myStepper1.setSpeed(40);
140     myStepper2.setSpeed(60);
141     myStepper3.setSpeed(60);
142     myStepper4.setSpeed(40);
143     myStepper1.step(-1);
144     myStepper2.step(1);
145     myStepper3.step(1);
146     myStepper4.step(-1);
147   }
148    if(pitch<-1 && (roll>-1.0 && roll<1) ){
149     myStepper1.setSpeed(60);
150     myStepper2.setSpeed(40);
151     myStepper3.setSpeed(40);
152     myStepper4.setSpeed(60);
153     myStepper1.step(1);
```

```
    myStepper2.step(-1);
    myStepper3.step(-1);
    myStepper4.step(1);
   }

}
void calculate_IMU_error() {
  // first we position the sensor perfectly at 0 degrees, and if the sensor is good we should not get any error, in case we get an error we add it in the code
  // we reed a number of outputs from teh sensor and then we do the mean for each axis

  // Read accelerometer values 200 times
  while (c < 200) {
    Wire.beginTransmission(MPU);
    Wire.write(0x3B);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU, 6, true);
    AccX = (Wire.read() << 8 | Wire.read()) / 4096.0 ;
    AccY = (Wire.read() << 8 | Wire.read()) / 4096.0 ;
    AccZ = (Wire.read() << 8 | Wire.read()) / 4096.0 ;
    // Sum all readings
    AccErrorX = AccErrorX + ((atan((AccY) / sqrt(pow((AccX), 2) + pow((AccZ), 2))) * 180 / PI));
    AccErrorY = AccErrorY + ((atan(-1 * (AccX) / sqrt(pow((AccY), 2) + pow((AccZ), 2))) * 180 / PI));
    c++;
  }
  //Divide the sum by 200 to get the error value
  AccErrorX = AccErrorX / 200;
  AccErrorY = AccErrorY / 200;
  c = 0;
  // Read gyro values 200 times
  while (c < 200) {
    Wire.beginTransmission(MPU);
    Wire.write(0x43);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU, 4, true);
    GyroX = Wire.read() << 8 | Wire.read();
    GyroY = Wire.read() << 8 | Wire.read();
  //  GyroZ = Wire.read() << 8 | Wire.read();
    // Sum all readings
    GyroErrorX = GyroErrorX + (GyroX /32.8);
    GyroErrorY = GyroErrorY + (GyroY / 32.8);

    c++;
  }
  //Divide the sum by 200 to get the error value
  GyroErrorX = GyroErrorX / 200;
  GyroErrorY = GyroErrorY / 200;
  // Print the error values on the Serial Monitor
  Serial.print("AccErrorX: ");
  Serial.println(AccErrorX);
  Serial.print("AccErrorY: ");
  Serial.println(AccErrorY);
```

```
205   Serial.print("GyroErrorX: ");
206   Serial.println(GyroErrorX);
207   Serial.print("GyroErrorY: ");
208   Serial.println(GyroErrorY);
209 
210 }
```

## 1.4 Conclusion

Going throw this project I had learn more about how to use the $I^2C$ serial protocol, how important is to know the registers of the sensor. Firstly, was a bit confusing to find a starting point , I found on internet a low-level example, where every steps was explained ( Dejan from HowToMechatronics).

At the beginning I tried to make a 3D model, and then to speak with my Dad, if we can do it with wood only. The important things was to take as much as I can from the load of motors, because these steppers doesn't support a large amount of stress so I put a pillar in the middle. By making a spherical join, the platform benefits of 2 degree of freedom. After this idea my dad made the cuts on the materials and with my help clue them together. (like in the pictures).

After the physical part was finished, I had to make the connections and to make a little control, the positioning of the sensor is very important, I put it on the top platform, because only in this mode I can close the loop (get the error). The control is like this, the motors are moving until the top platform is between $\pm 1^\circ$ on x axis, and $\pm 1^\circ$ on y axis.

Doing all these I encountered some small problems like: over-heating of the motors(because of the external source for motors), arduino stops working after some seconds, but in finally I found for every problem a solution.

Overall, after going throw all the steps in developing it, I am delighted with the outcome.

# Chapter 2

# VTOL-Vertical take-off and landing

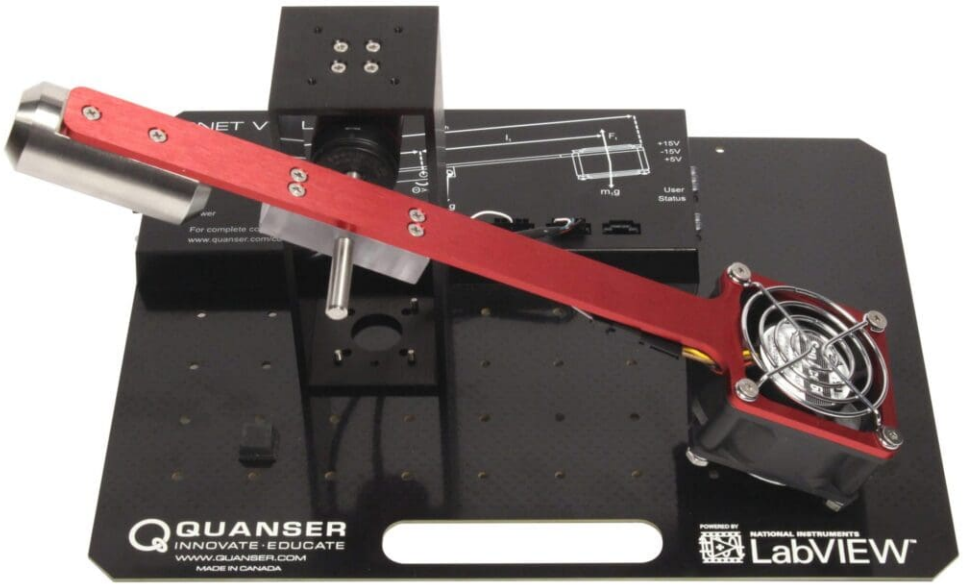



Figure 2.1: QNET 2.0 VTOL Board

## 2.1 Description

The project consist of working on a real system, a oscillating one. For this project I had to make an Identification, then base on that developing two discrete controllers.

This projects came as a rewarding for a good grade in the first semester. I couldn't refuse a practical project, I enjoyed to see how it's working.

I short description on this project, the system is made of 3 main components a balancing weight, an high resolution encoder and a variable speed fan. The input of the system is the voltage applied to the variable speed fan. This causes the fan to rotate and generates thrust in order to move the beam upward.

As usual I will upload on GitHub more images and a video clips.

### 2.1.1 Identification

In this part I used a Second order transfer function with a smaller $\zeta = 0.0885$ ($Initial : \ 0.24$). I did this because all the methods I know for developing a controller are working for maximum second order element. In plus, I increase the number of oscillation to make a controller for worst case scenario.

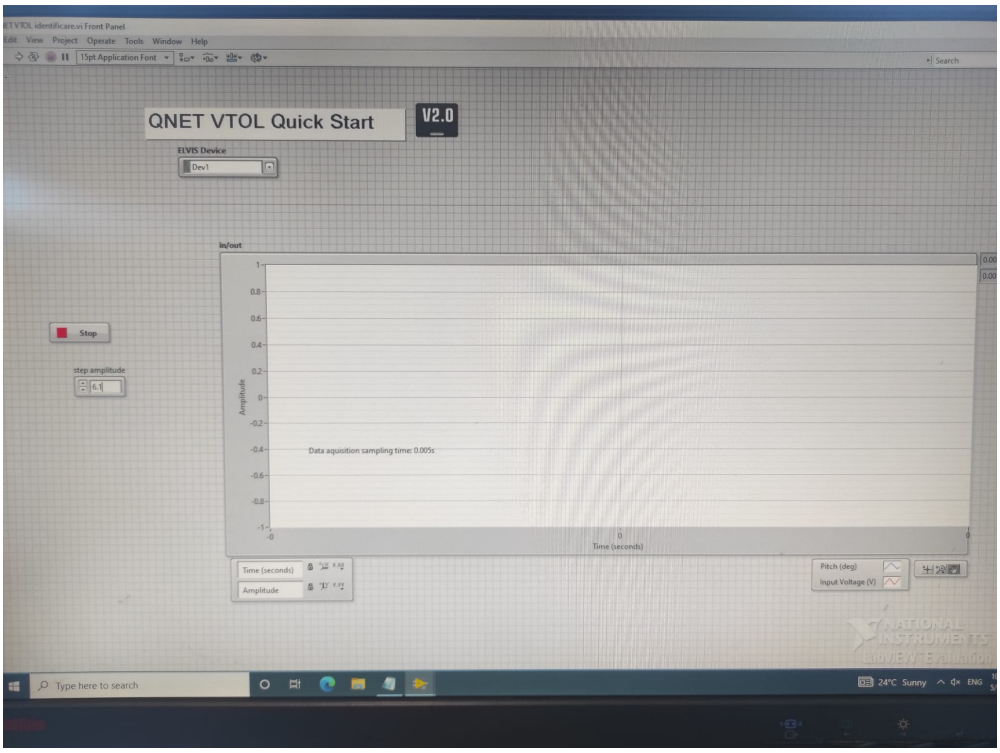



Figure 2.2: IdentificationPannel

For retrieving data from the VTOL I used a special made panel in LabView witch give me the possibility to apply a voltage. I applied a value of $6.1V$ to get the pitch angle of $0°$ .The panel is shown in Figure 2.2.

Transfer function:

$$H(s) = \frac{19.10}{s^2 + 0.37s + 4.56}$$

The results can be seen in Figure 2.4 and the video is available on Github.

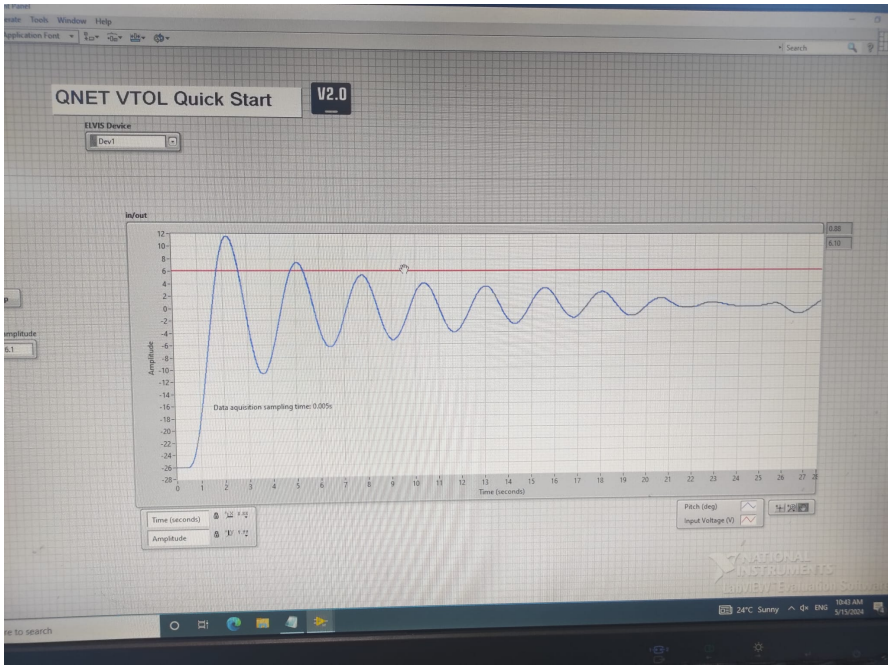



Figure 2.3: Data output

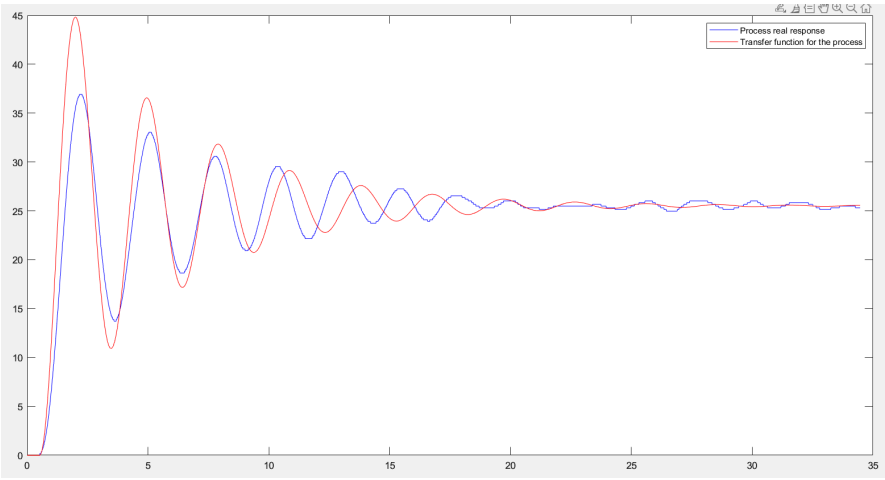



Figure 2.4: Transfer function approximation

# Code:

```
clear all;
load("IdentificareTF.mat")
plot(iden)
u=iden.u;
%Normalizing the output
iden.y=iden.y-iden.y(1);
t=(0:iden.Ts:(length(u)-1)*iden.Ts)';
y=iden.y;

% Steady State values for input/output
yss=mean(y(4395:end));
uss=u(23);
% get the time for first 3 oscilations
t1=2.24;
t2=3.625;
t3=5.14;
% calculate gain
k=yss/uss;
%calculate overshoot
M=(36.92-yss)/yss;
%computi zita
zita=log(1/M)/sqrt(pi^2+log(M)^2);
%Period
T0=t3-t1;
% natural frequency
wn=2*pi/T0/sqrt(1-zita^2);
% delay from plot
delay=0.52;
zita=zita-0.16
wn=wn-0.1
Hp=tf(k*wn^2,[1 2*zita*wn wn^2],'iodelay',delay);
t=(0:iden.Ts:(length(u)-1)*iden.Ts)';
% Simulation
plot(iden)
y_aprox=lsim(Hp,u,t);
plot(t,y,'b');hold on;
plot(t,y_aprox,'r')
legend("Process real response","Transfer function for the process")
```

Listing 2.1: Identification code

## 2.2 Controller-Gain and Phase margin

### 2.2.1 About

This method only base on reading a bode diagram and playing a bit with some numbers and off course knowing some stability criterion.

Looking at the bode diagram we can see that the gain and phase margin have considerable negative value (Figure 2.5) which lead the system to instability.

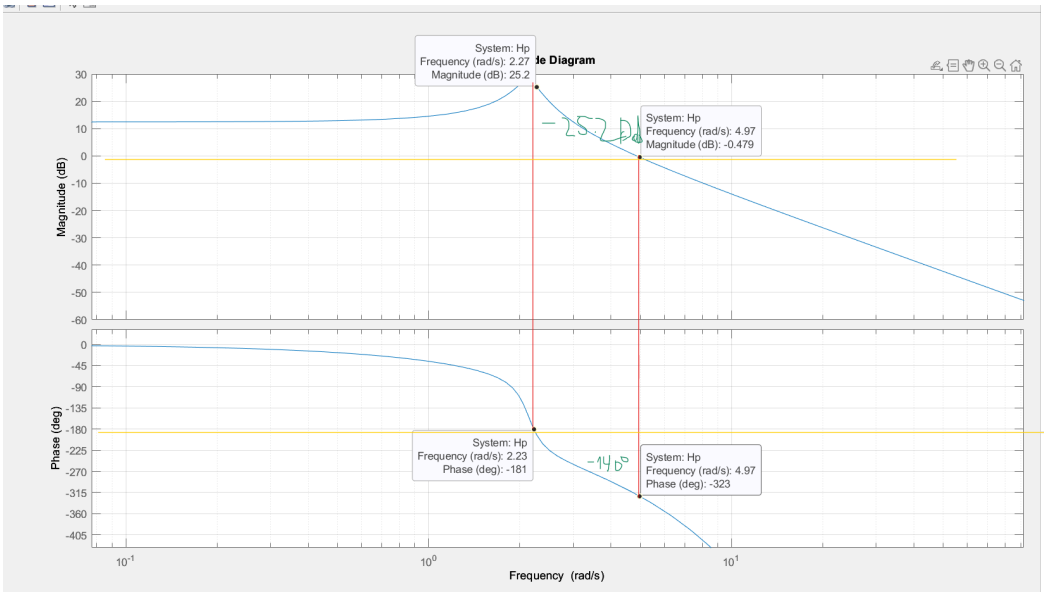



Figure 2.5: Unstable system

Only thing we can do is to move the gain crossover frequency to the left, before that resonance peek. After doing some test I got this crossover frequency $\omega = 0.145$. Here are the results after forcing the crossover frequency.

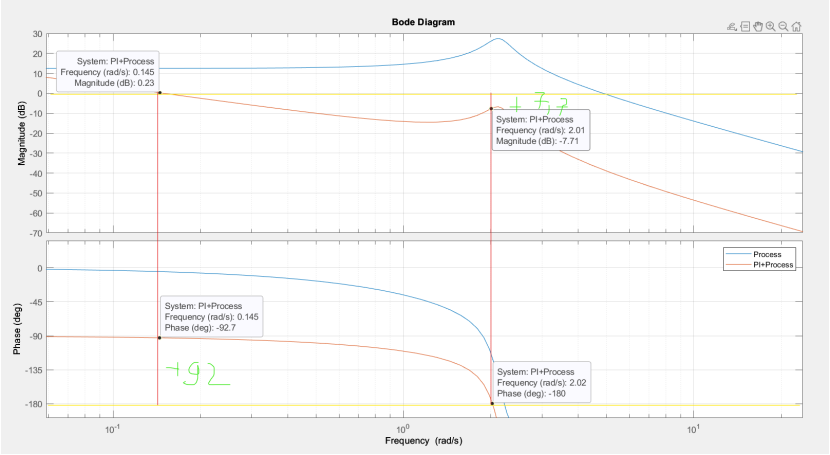
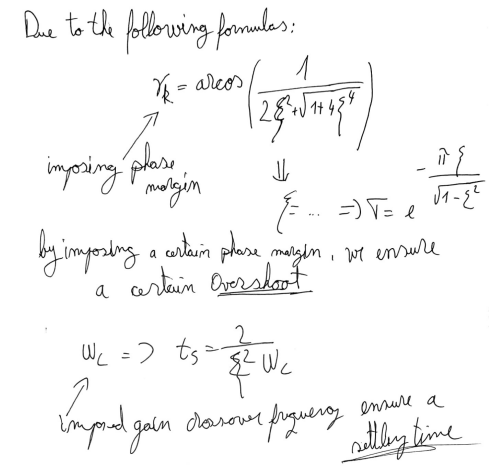
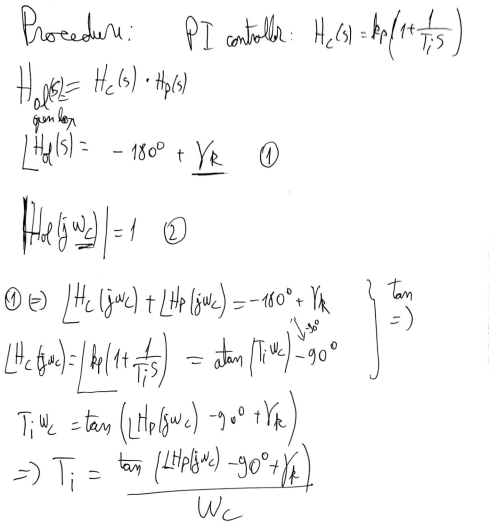


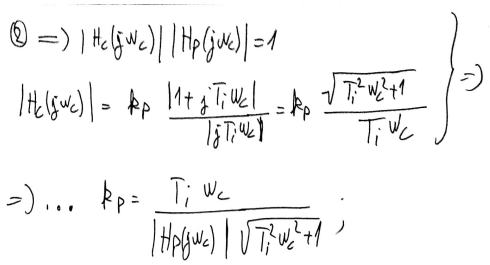

Figure 2.6: Stable system

### 2.2.2 Formulas

On this page are all the formulas which helps me develop a PI controller and the code for Mathlab as well.



### 2.2.3 Code

```
figure
%Hp process function
bode(Hp)
% choose the best phase
phase=45;
% here you can see that i want 1.68 the cutting frequency,
% but I got 0.145, It's happen because I improve the gain
% coeefficient to be better for my case and maybe that resonance peek
% influence
wc=1.68;
% after formula
ti=tand(-90+phase+70)/wc;
% playing with the value for a stabilized process
% -0.031 help me to stabilize the process
kp=ti*wc/db2mag(20.3)/sqrt(ti*ti*wc*wc+1)-0.031;
%PI controller
Hc=tf([kp*ti kp],[ti 0]);
% see the result on bode
% phase and gain have normal values
bode(Hp);hold on
bode(Hc*Hp);grid;
legend("Process","PI+Process")
%discretize controller at 5ms becasue that are the specification
Hc_d=c2d(Hc,0.005,'zoh');
```

Listing 2.2: First controller

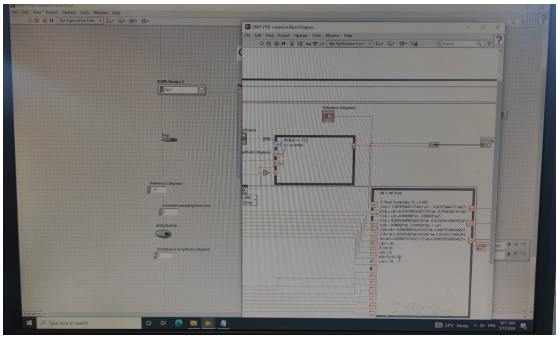
After I did the controller and discretize it, I have to make a recurrence relation which will be used by the controller(c-control value,e-error):

$$H_c(z^{-1}) = \frac{0.00982 - 0.00965z^{-1}}{1 - z^{-1}} = \frac{c(z^{-1})}{\epsilon(z^{-1})}$$

$$c(z^{-1})(1 - z^{-1}) = \epsilon(z^{-1})(0.00982 - 0.00965z^{-1})$$
$$c(k) - c(k-1) = 0.00982 \cdot \epsilon(k) - 0.00965 \cdot \epsilon(k-1)$$
$$c(k) = c(k-1) + 0.00982 \cdot \epsilon(k) - 0.00965 \cdot \epsilon(k-1)$$

Here is the equation in put in the system:

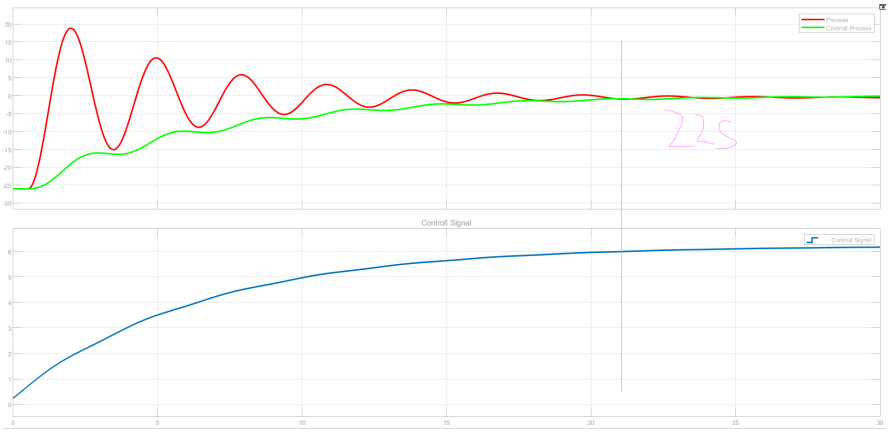
ck=ck1+0.009827074323175*ek-0.009650050800462*ek1;

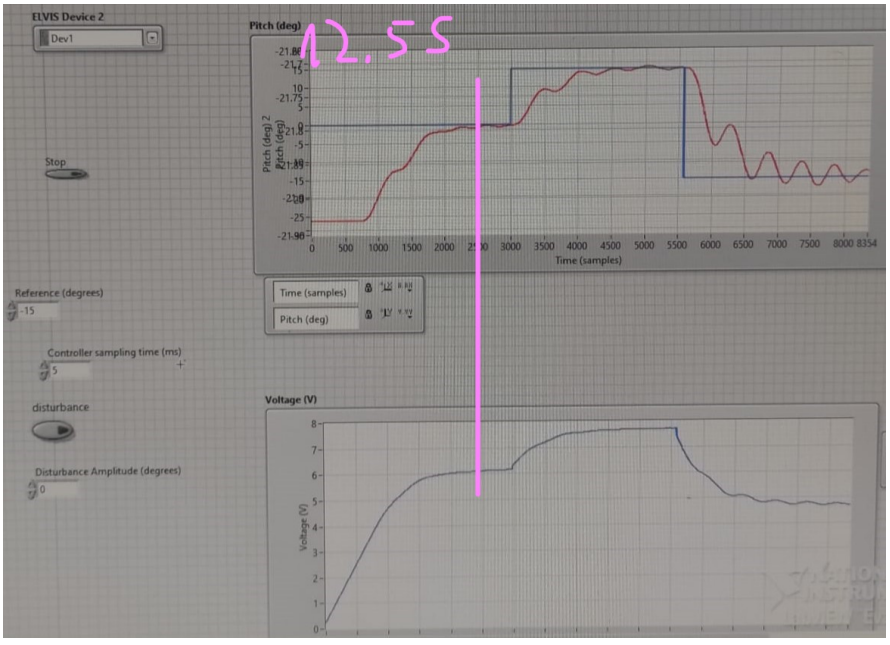


### 2.2.4 Results

Here are the result I got after I did my own simulation in Simulink and the real life one. The settling time differs a little bit. Firstly, I had some problems because of that. I implemented a fast controller with a settling time of 4.5 s and then I didn't know why is not working on the real system.

A upload a video on GitHub about the experiment.



## 2.3 Controller-Vogel-Edgar

### 2.3.1 About

This controller is a bit complicated to demonstrate or to go in deeply details about formulas, but I can tell you what I understood from this.

Firstly, the controller is an improvement of Dahlin Algorithm. Dahlin is a method for implementing a discrete controller by assuming the close loop transfer function to be of First Order with or not time delay, depending of the process.

In the Figure 2.7 you can see an example of how Dahlin works for a process with a transfer function of first order.

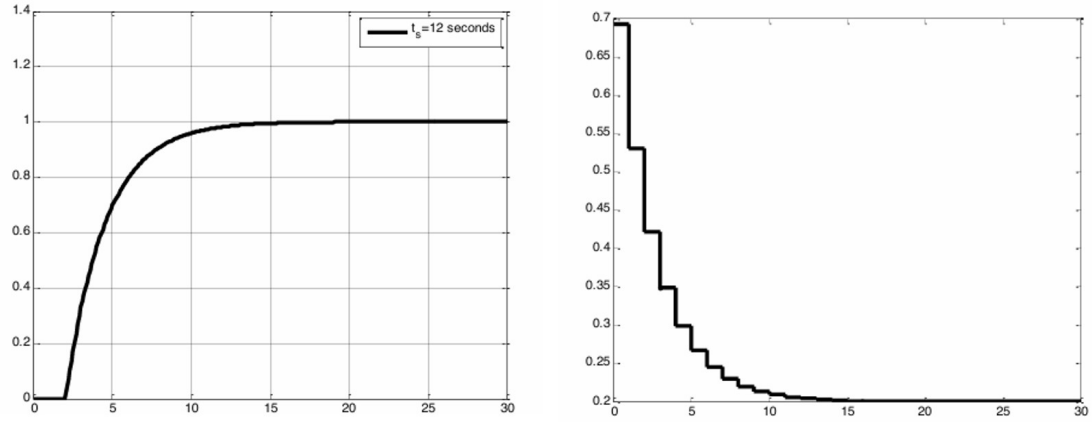



Figure 2.7: Process is of First order

Things got complicated after you have a second order for the process. Because, it's appear an effect of ringing witch cause the controller to face a lot of stress. In plus, the output is not smooth, it's contain oscillation.

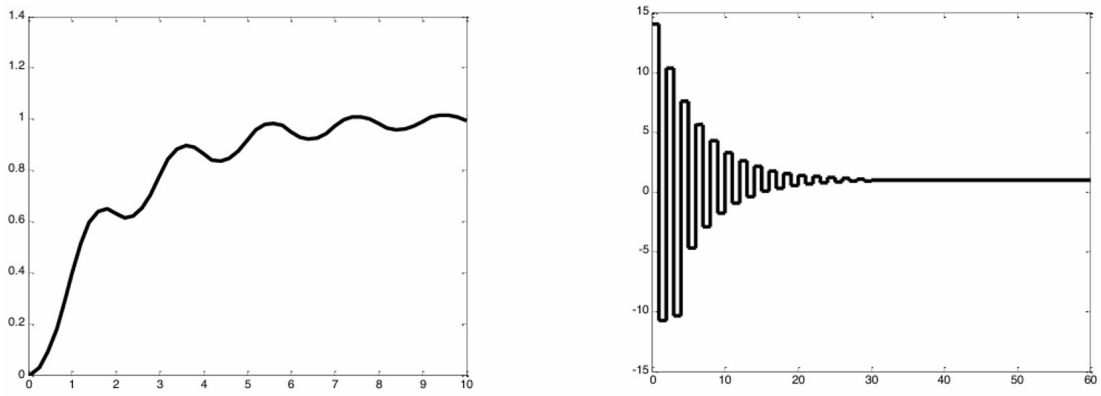


Figure 2.8: Process is of Second order

Dahlin managed to resolve this problem by replace the ringing pole with the sum of it's coefficients (Figure 2.9) .

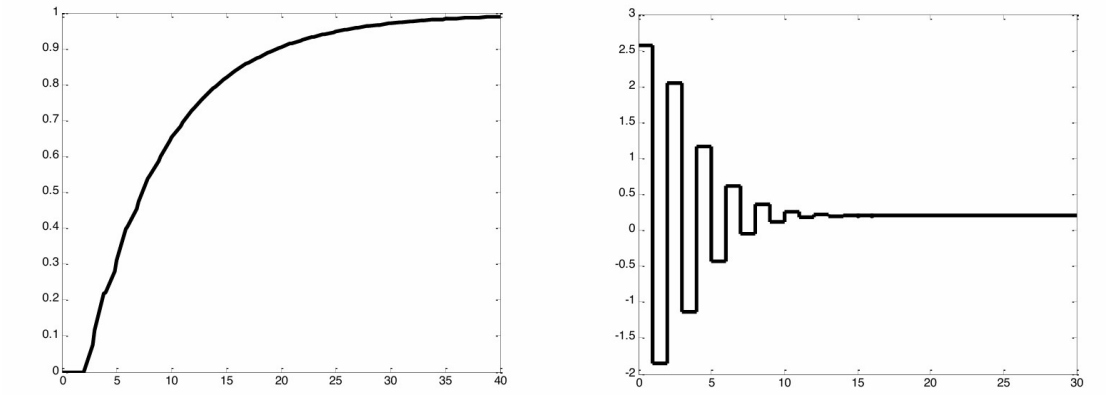



Figure 2.9: Improved controller

Now Vogel-Edgar algorithm comes with a better improvement, they added a zero to the closed loop transfer function. The controller stress was diminish drastically.

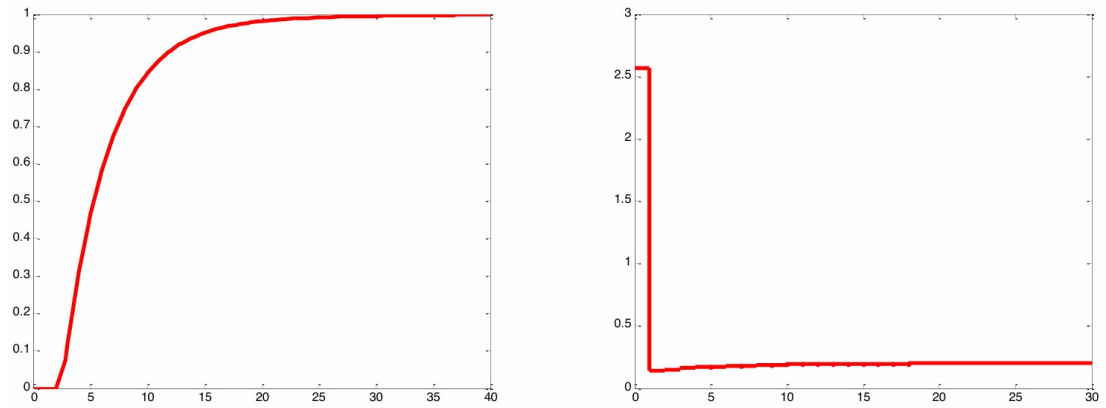



Figure 2.10: Vogel-Edgar controller for SecondOrder

### 2.3.2 Formulas

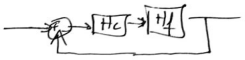
The next pages are only for a better view about what I wrote earlier.

$H_0(s) = \frac{e^{-\tau_m s}}{\lambda s + 1}$ — dead lag; $\tau_m$ - seconds; $\lambda$ - time constant, depends for how we want the settling time

Discretized ⇓ zero Order Hold (ZOH)

$$H_0(z^{-1}) = \frac{(1 - e^{-T_s/\lambda})\, z^{-N-1}}{1 - e^{-T_s/\lambda} z^{-1}}$$

$$H_0(z^{-1}) = \frac{H_f(z^{-1})\, H_c(z^{-1})}{1 + H_f(z^{-1}) H_c(z^{-1})}$$




⇓

$$H_c(z^{-1}) = \frac{1}{H_f(z^{-1})} \cdot \frac{H_0(z^{-1})}{1 - H_0(z^{-1})}$$

Works like in fig. 2.7 for a first Order Tf process.

Dahlen improved for 2nd Order process

$$H_f(z^{-1}) = \frac{b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} z^{-N} \qquad N = \frac{\tau_m}{T_s}$$

process is approximated with integer delay

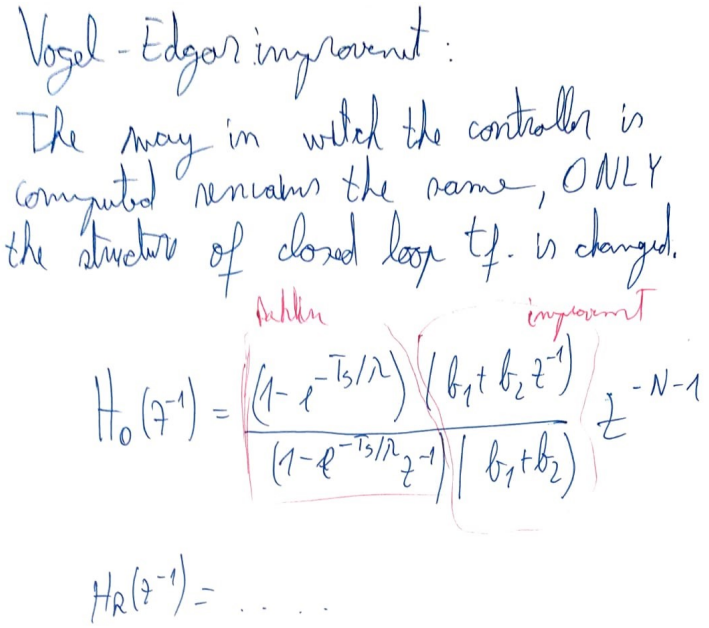
$$H_0(z^{-1}) = \frac{(1 - e^{-T_s/\lambda})\, z^{-1}}{1 - e^{-T_s/\lambda} z^{-1}} z^{-N}$$ is the same

- doing the same computation we get:

$$H_R(z^{-1}) = \frac{(1 + a_1 z^{-1} + a_2 z^{-2})(1 - e^{-T_s/\lambda})}{(b_1 + b_2 z^{-1})(1 - e^{-T_s/\lambda} z^{-1} - (1 - e^{-T_s/\lambda}) z^{-N-1})}$$

Setting $z = 1$, eliminates the ringing

$$H_R(z^{-1}) = \frac{(1 + a_1 z^{-1} + a_2 z^{-2})(1 - e^{-T_s/\lambda})}{(b_1 + b_2)(1 - e^{-T_s/\lambda} z^{-1} - (1 - e^{-T_s/\lambda}) z^{-N-1})}$$




### 2.3.3 Code

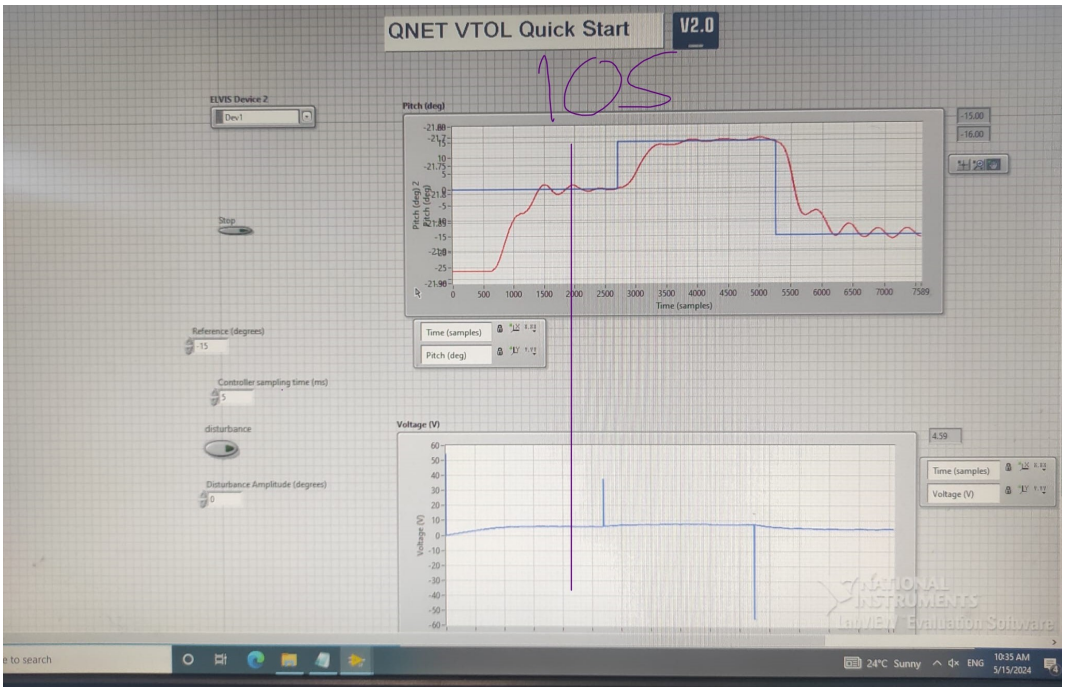
```
%taking numerator from discrete tf
num=cell2mat(Hp_d.Numerator);
den=cell2mat(Hp_d.Denominator);
%assing the value of each coeff
a1=den(2);
a2=den(3);
b1=num(2);
b2=num(3);
%sampling time over the time constant
e=exp(-0.005/5);
%asing the numerator and denominator as in formula
Hr_ve_num=(1-e)*den;
Hr_ve_den=[b1+b2, -(b1+b2)*e -(1-e)*b1, -(1-e)*b2];
Hr=tf(Hr_ve_num,Hr_ve_den,0.005);

%% helps me to compute faster the requrence formulas
num=Hr_ve_num/Hr_ve_den(1)
den=Hr_ve_den/Hr_ve_den(1)
```

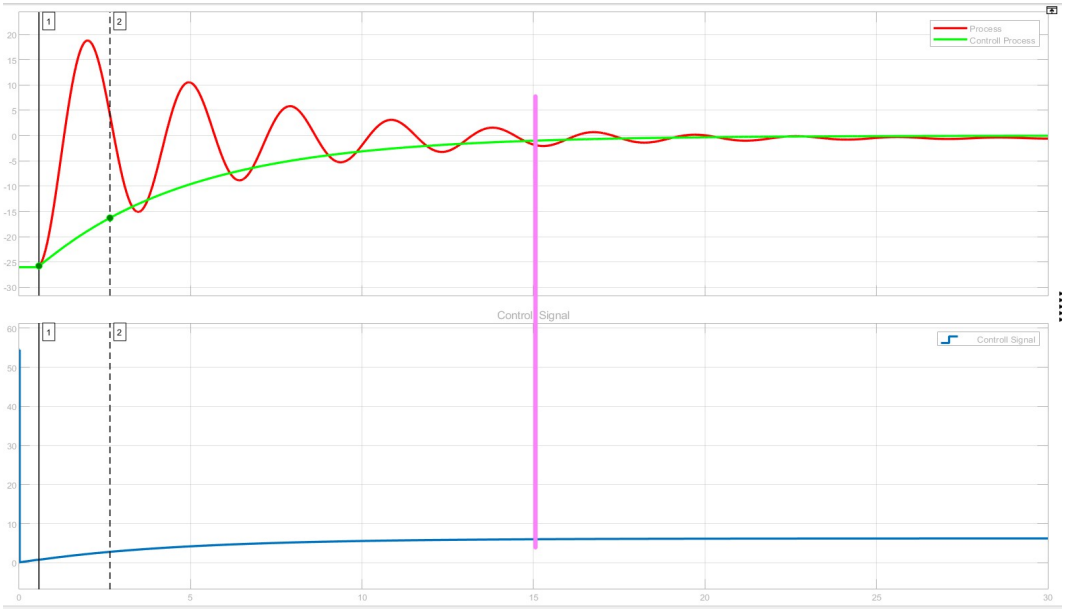
Listing 2.3: First controller

Again the recurrence relation, after I discretize :

ck=2.09481071269372*ek-4.18542313386844*ek1+ 2.09085131141609*ek2+ 0.999000499833375*ck1+0.00049990766245 6721*ck2 +0.00049959250416 8257*ck3;

### 2.3.4 Results


QNET VTOL Quick Start
V2.0
105
ELVIS Device 2
Dev1
Stop
Pitch (deg)
Time (samples)
Reference (degrees)
Controller sampling time (ms)
disturbance
Disturbance Amplitude (degrees)
Voltage (V)
Time (samples)
Voltage (V)
Evaluation Software
24°C Sunny
ENG
10:35 AM
5/15/2024


Process
Controll Process
Control Signal
Controll Signal

## 2.4 Conclusions

I am very glad that I finished the project. I had some challenges with it, because I didn't knew the small tricks, like: don't try to make it perfectly from start, because in Simulink works perfectly, and when you do the real tests it's opposite.

# Chapter 3

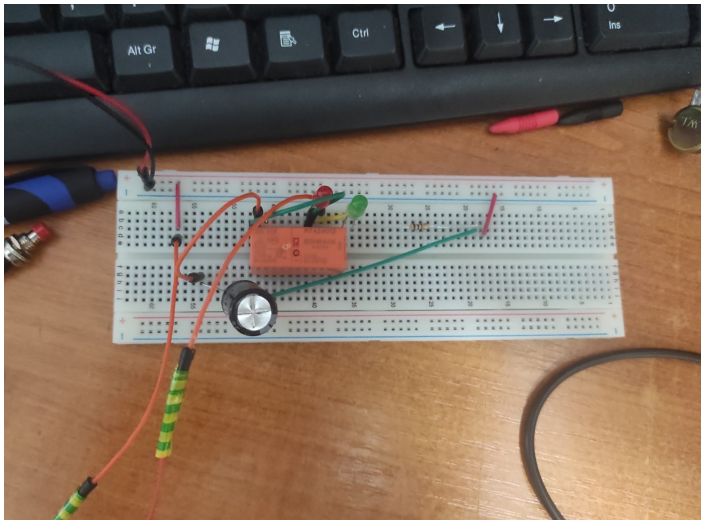
# Basic circuits

## 3.1 Description

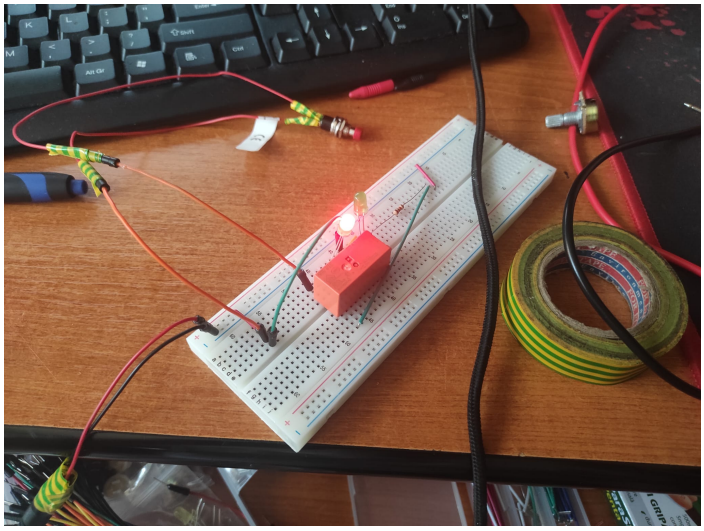
This were the first things I did after I receive your email. I had implemented some circuits like a relay,an oscillator using the relay, testing transistors. I have reached the halfway point of the book, including amplification for a speaker. Here I got some delays regarding the components. The circuits use a few programmable unijunction transistors, which are hard to find here, so I had to wait for them 2 weeks and after I didn't have time to build them.

I also attach some picture with the circuits down and also some video clips with them working on Github.

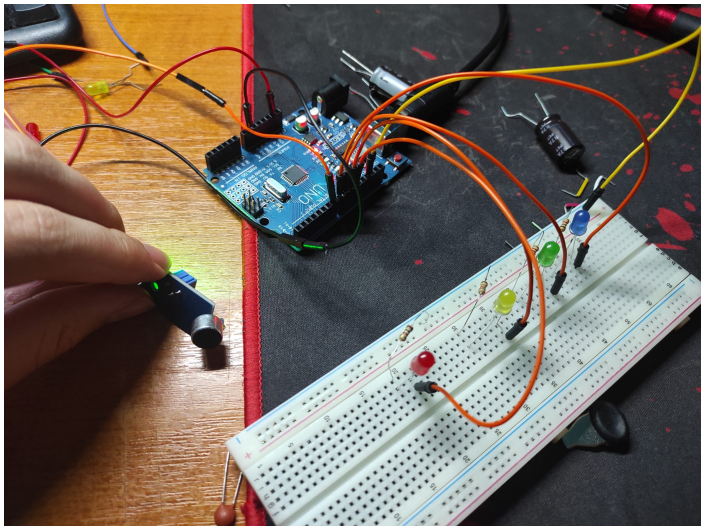
## 3.2 Pictures



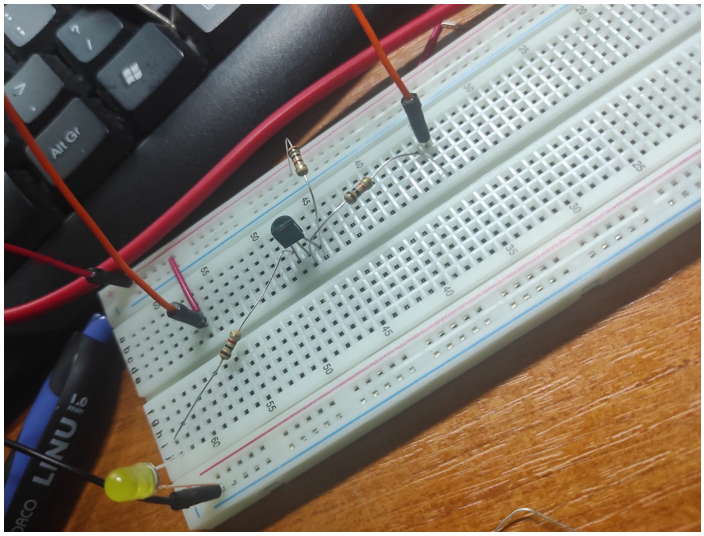

(a) Panel used for Identification.



(b) Transfer function compared with the real process.



(c) The simple controller.




(d) Vogel-Edgar Controller.

Figure 3.1: More images are on Github.

# Chapter 4

# What I learn this Semester

The most important things that I learned this semester is how to work with PLCs. I worked with Siemens(300 series) and Schneider. I made some small programs in Tia Portal and EcoStruxure using ladder logic and Grafcet.

Another technical domain with witch I made contact was Hydropneumatic control equipment, at the laboratory we used some old equipment, but the basics have been learned.Also, the course teacher took as to see some new equipment's at Emerson. There I saw some Differential Pressure Flow Meters, I had the chance to program it using HART protocol.

Real time system, was the discipline witch give me the opportunity to think at a bigger scale. At RTS we were taught to be more disciplined when we start a new project, and how to make/read UML diagrams. Also, we taught how to program in Java using Petri nets. Including some advance programming like Object Enhanced Time Petri Nets Framework.