Institut de Robòtica
i Informàtica Industrial

POLYTECH
SORBONNE

SORBONNE
UNIVERSITÉ

Control in Thrust of a Drone Motor

Intern: Kevin BECQUET

Internship Supervisor: Joan Solà

RÉSUMÉ

Durant ce stage il m'a été demandé de réaliser une preuve de concept pour l'implémentation d'un contrôle moteur par la force de poussée que ce dernier génère. Ce projet prend place dans un projet d'ampleur plus importante : l'implémentation d'un drone agile. La réalisation de ce contrôleur se divise en deux parties : premièrement le choix des composants et leurs connexions, puis secondement, l'implémentation.

Pour assembler ce contrôleur il faudra installer un capteur sous le moteur pour mesurer la poussée qu'il crée. Ensuite, le capteur est connecté à un amplificateur de tension pour transcrire ses mesures dans une échelle plus lisible. Enfin, la sortie de l'amplificateur est connectée à un microcontrôleur qui reçoit également une commande de force que l'on veut appliquer au moteur.
Ce microcontrôleur est en charge de l'exécution des lois de commande permettant d'envoyer au moteur les signaux adéquats pour lui faire produire la force de poussée souhaitée.

L'implémentation des lois de commande de déroule de la façon suivante. Le microcontrôleur commence par acquérir la commande envoyée par l'utilisateur et la valeur de la force ressentie par le capteur. Ensuite à partir de ces données, le microcontrôleur détermine le signal à envoyer au contrôleur du moteur à l'aide d'un contrôleur PID numérique couplé à un terme de commande prédictive (ou feedforward).

Enfin, pour vérifier l'efficacité du contrôler nous avons implémenté un outil de visualisation puis réalisé une série de tests s'étant révélé satisfaisante.

THANKS

I want to thank all the people who helped me in the realization of this internship.

First, I want to thank my tutor, Joan Solà for the time he spent during both the realization of this project and the redaction of this report and for the knowledge he shared with me all along my internship.

I also want to thank all the laboratory's mobile robotics team for their welcome and their expertise in particular Josep Martí who helped me a lot on a problem of register modification, critical to the success of the project, Patrick Grosch who taught me how to use several workshop instruments and Hugo Duarte who helped me a lot both during the internship in the understanding of the way ROS works and outside of it making my arrival to the city easy and sharing his experience of it.

Thanks to all of these people I learned a lot during my internship and I'm grateful for all the experience I acquired doing it.

SUMMARY

Vocabulary:

PWM : Pulse Width Modulation, squared signal with a variable duty cycle

ESC : Electronic Speed Controller, component that allows the conversion between a PWM signal and a three phased signal usable by the motor in order to spin.

INTRODUCTION

I did my internship in the Mobile Robotics team of the Institut de Robòtica i Informàtica Industrial (IRI), a Robotics laboratory located in Barcelona (Spain). The goal of this internship was to implement a system that allows the control of a motor by the thrust it produces.

Motivations of the project

The laboratory is currently working on a project: the realization of an agile drone. This drone wants to be able to follow complex trajectories and thus react quickly. To make this drone react quickly, they use a model predictive control that sends commands in force to the drone and in particular its motors' force.

For now, the motors are commanded in open loop. The only thing that is controlled is the signal sent to the motor. The force produced by the motor is calculated using the motor speed and models of aerodynamics and of the propeller attached to the motor.

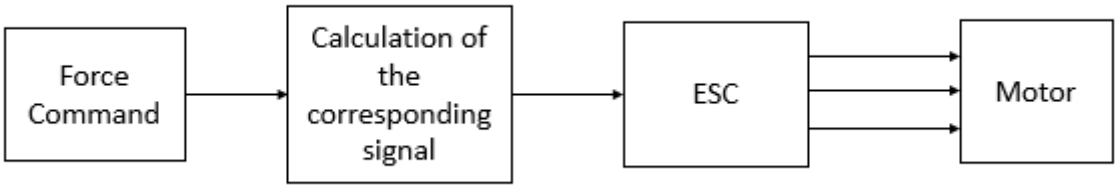

Figure 1 - Schematic of the current way to send commands to the motor

However, there could be air perturbations or other things that the models do not take in consideration making the force determination not accurate enough.

Thus, to bypass those calculations, we want to create a closed loop controller that will make the motor produce the desired force. By putting a sensor right under the motor, we allow the possibility of a feedback of the forces applied to it.

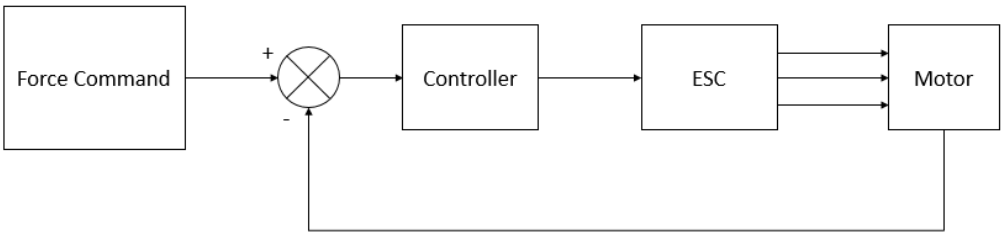

Figure 2 - Desired way to send commands to the motor

Objectives

So, the goal is to close the loop from the current system to control force.

This loop has to have a dynamic that is faster than the global system that sends the force commands not to be the part that limits the drone's overall performance.

Moreover, as we want a fine control of the drone's trajectories, we don't want any error between the command and the real thrust.

Method

This project can be split in two parts: the hardware conception and the software implementation.

For the hardware part, we can use a more detailed version of the figure 2 by putting in parallel the hardware components that will be used to build this controller. The components

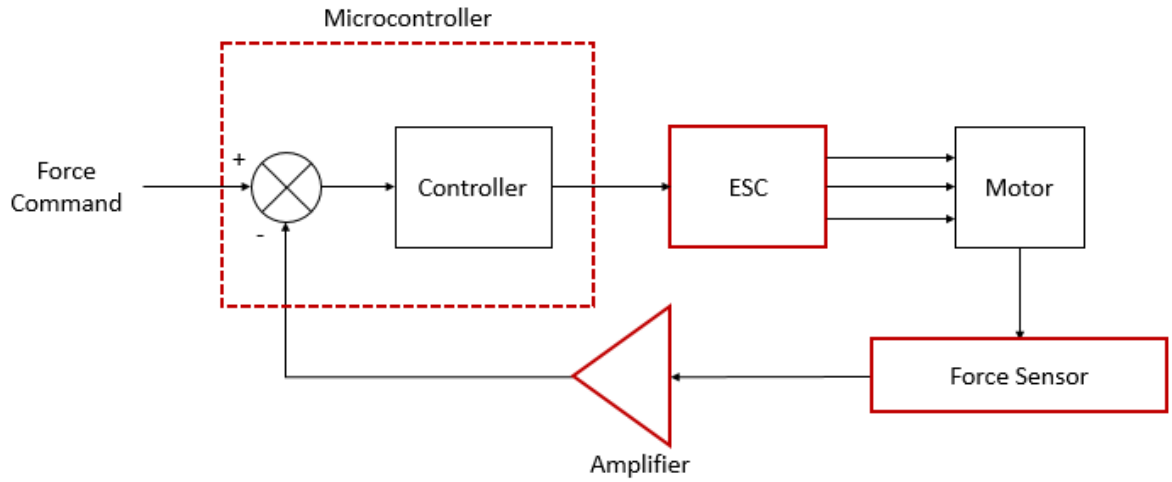

Figure 3 - Desired system explicating the components used in it

Concerning the software part, we need the microcontroller to implement these for tasks:

- Receive force commands from the UAV main controller
- Get the thrust values from the sensor and filter them
- Implement the controller and tune it
- Send the corresponding signal to the ESC to make the motor produce the desired thrust
- Send the data to the UAV main controller to confirm the results we get.

Finally, the tests were realized in the laboratory the motor caged for safety reasons and fixed to a benchmark to get an external measurement of the force it produces. To get these measurements we fixed the motor to a benchmark, a tool able to collect several data like the rotation speed, the torque or the thrust of the motor.

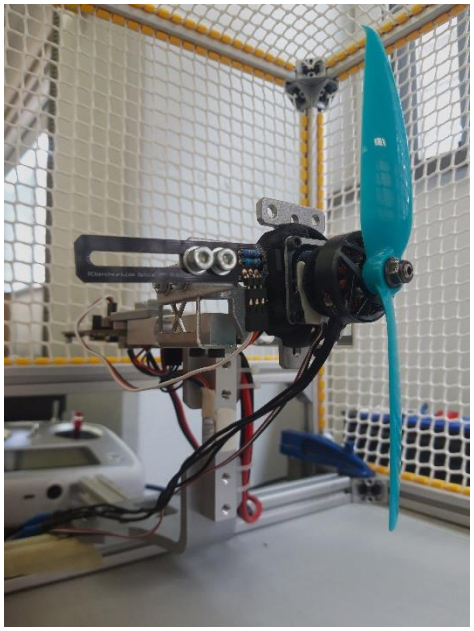

Figure 4 - Motor installed on the benchmark

1-HARDWARE

This part is consecrated to the hardware components used during this project, their explanation and how they are connected together.

To make a control in thrust of the motor we will need to put a force sensor to know how much force it produces. Then we will also need an amplifier to make the sensor's signal readable by the microcontroller which will compute the signal to send to the ESC, the component that

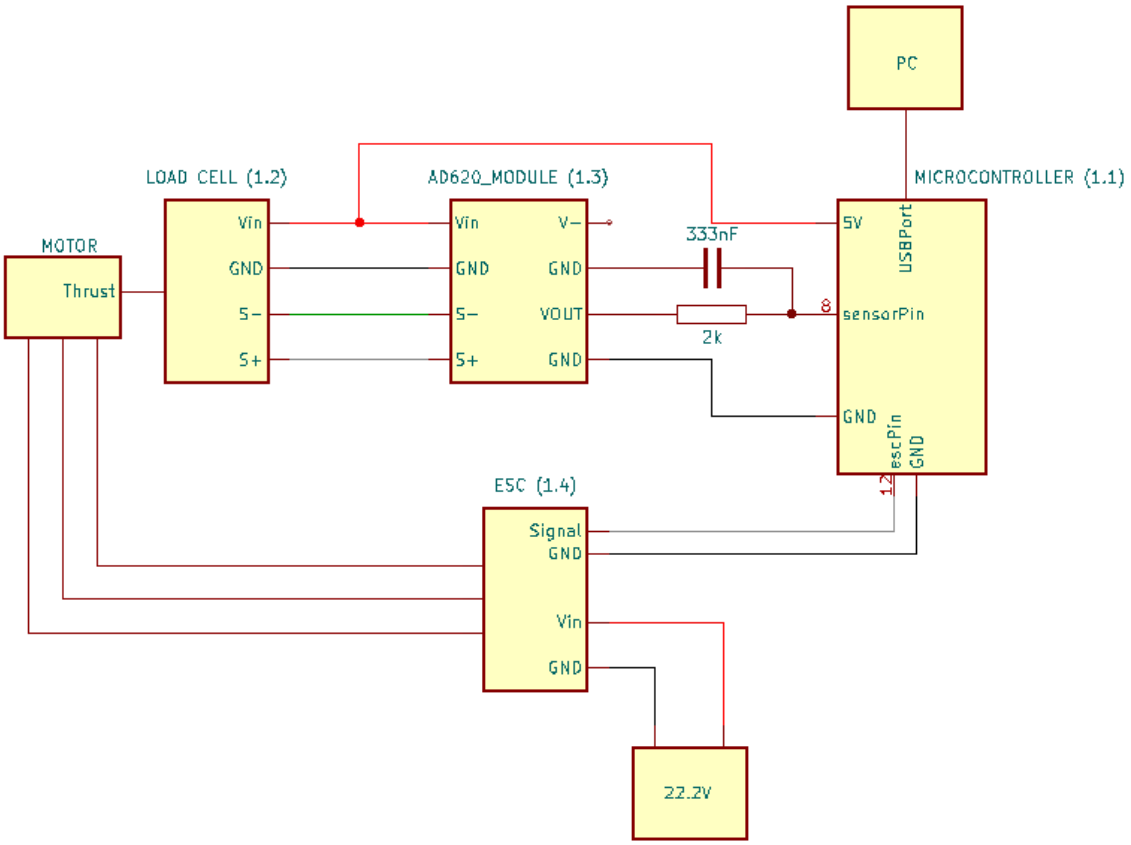

Figure 5- Block schematic of the system

1.1-Microcontroller

The microcontroller used in this project in a STM32F103CB mounted on an OpenCM9.04 board. It will allow the implementation of the software part and the communication between the other components. The microcontroller is connected to a pc with an USB wire.

1.2-Force Sensor

To build a thrust controller, we need, in a first part, to know what force the motor produces. The motor (with the propeller) can produce a maximum thrust of 20N. This motor cannot produce negative forces. Therefore, we want our system to be able to measure precisely the forces applied to the motor in a range from -5N to 25N.

The sensor used to do so is a load cell. It is built fixing a Wheatstone full bridge over an aluminium.

The load cell is fixed at the base of the motor by a printed fixation piece and some screws:

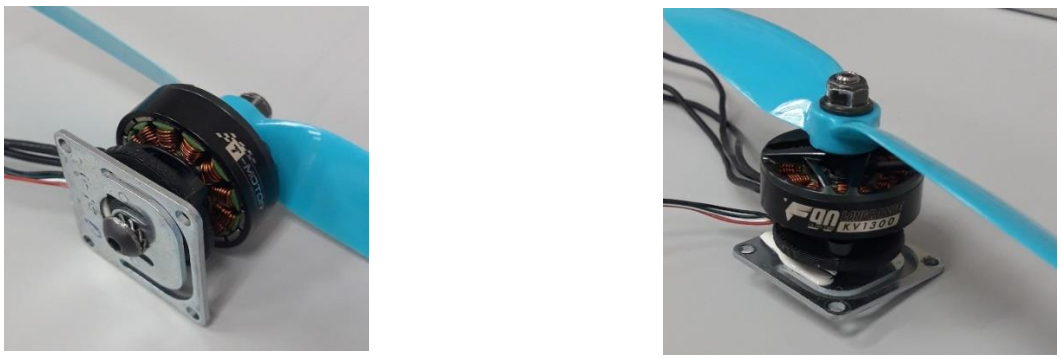

Figure 6 - Fixation between the motor and the load cell

1.2.1-Wheatstone bridge

A Wheatstone bridge is an electronic circuit composed by 4 strain gauges acting like variable resistors.

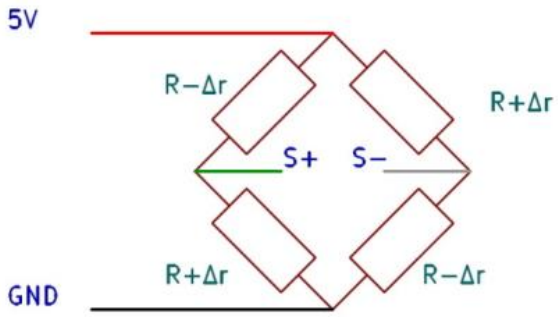

Figure 7 - Wheatstone full-bridge schematic

When no force is applied on the sensor the 4 strain gauges have the same resistive value. However, when a force is applied to the sensor, the aluminium plaque will bend under this force causing two of the strain gauges to be in compression, decreasing their resistivity while the two other will be in tension, increasing their resistivity.

The output signal can be determined using Kirchoff's laws:

$$V_s^+ = \frac{R + \Delta r}{2R} \cdot V_{in} \;\; ; \;\; V_s^- = \frac{R - \Delta r}{2r} \cdot V_{in}$$

$$V_s = V_s^+ - V_s^- = \frac{\Delta r}{R} \cdot V_{in}$$

We have now a linear relation between the output of the sensor and the force applied to it.

1.2.2-Pros and Cons

This sensor provides a voltage that varies linearly according to the force applied to it. However, when powered with a 5V supply, the scale of the voltage variation is around 1mV each 10N. This sensor will need to be amplified to be usable in our system.

1.3-Amplifier

The purpose of this component is to make the signal from the load cell usable to the microcontroller.

1.3.1-HX711

The first amplifier that has been tried is the HX711 ADC amplifier. This one was at the first sight very practical to use: there are tutorials and a library on the Arduino IDE allowing an easy reading, calibration, treatment of the load cell's output signal.

However, as it is a digital converter, the acquisition speed of the thrust was limited by this component: its sampling rate only reaches 80Hz at its maximum. It was way too slow to be usable on a drone.

1.3.2-AD620

Thus, the component used to amplify the load cell signal is a module based on the AD620 instrumentation amplifier.

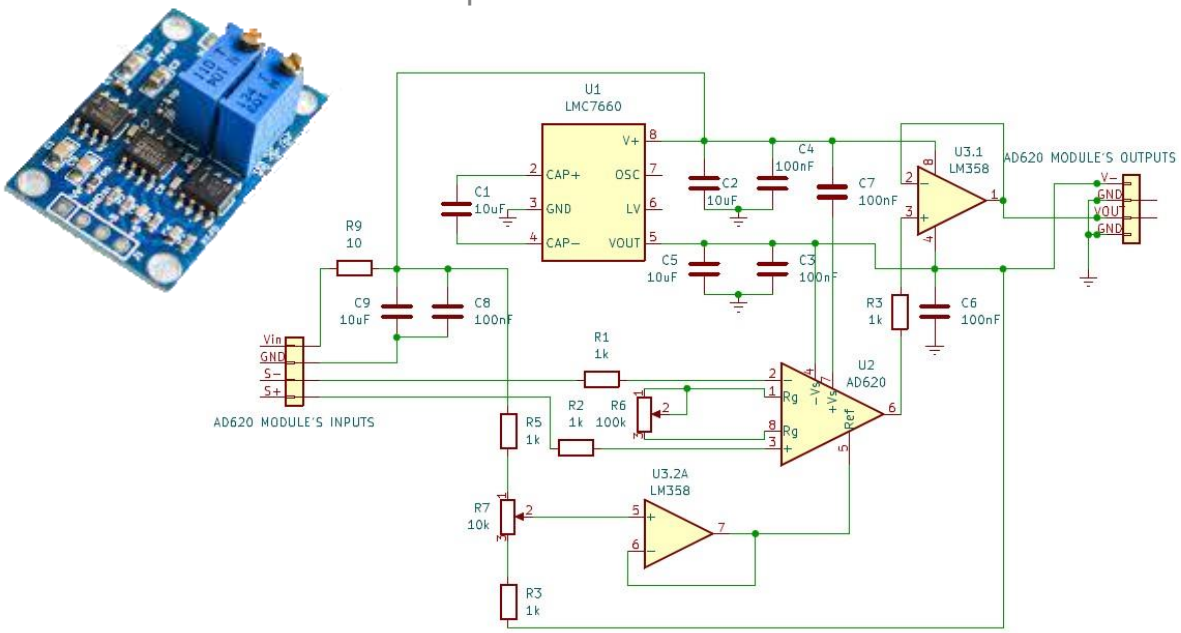

Figure 8 - Picture and electrical schematic of the AD620 module

This amplifier is an analog voltage amplifier. So the problem of sampling rate is solved as the digital conversion will now be done by the microcontroller, which is able to sample up to a maximum frequency of 10kHz.

The module possesses two potentiometers. One tunes the gain between the input and the output while the other tunes the offset of the value. Indeed, a calibration of the component is necessary to use it. But before the calibration, we can notice a noise non-negligible in the output signal.

1.3.2.a-Output filter

On this module the AD620 amplifier is powered up by a +5V/-5V source. The negative voltage is provided by a voltage inverter present on the board. However, the voltage it creates is not stable creating an undesired oscillation around 4kHz on the output of the module:

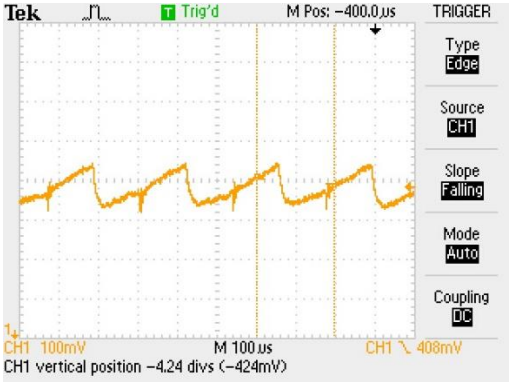

Figure 9 - Output voltage of the amplifier module with a steady signal in input (0V)

To fix this problem, we have to realize a low pass filter to reduce these oscillations.

This filter built is an RC circuit and has to respect 3 criteria:

- Suppress the oscillations to be useful
- Let the output signal being fast not to smoothen the motor's dynamic
- Not having a too high impedance to still being able to trust the microprocessor's reading

The first condition means that we have to make the ratio between the capacitor and the resistor so that:

$$f_c = \frac{1}{2\pi RC} < 4kHz$$

With the second condition the capacitor needs to stay small otherwise its discharge time will impact the measurement.

And the last condition makes necessary the use of a small resistor. Otherwise its values won't be negligible anymore in front of the microcontroller's impedance.

With these conditions. We ended up using a 400Ω resistor and a 100nF capacitor.

1.3.2.b-Potentiometers Calibration

The microcontroller has analog input pins that can handle voltage from 0V to 3.3V. So we need the amplifier to provide the biggest gain possible while staying in that range for all the force values between -5N and 25N that the sensor receives.

Tuning

The output voltage of the module follow this formula : $V_{out} = (G - 1) \cdot V_{in} + V_{ref}$

By using an external tool measuring the force and a voltmeter, we will start by tuning the potentiometer (R6 on the schematic) dealing with the gain. The goal is to have a voltage range the as close to 3V as possible when we apply a force range of 30N to the sensor.

Once the gain is tuned, we need to tune the offset potentiometer (R7 on the schematic) so the corresponding signal for -5N gives a value over 0V to allow the microcontroller to read it.

Replacement

Now that the potentiometers are tuned, we could start using it to get the force.

However, these potentiometers are very sensitive and the values of their resistors are very likely to change just by manipulating the system. So, we'll have to replace them by resistors to have constant resistor values.

In the case of the offset, the potentiometer acts like to resistors in serial that the values are variable but their sum is constant. So, to find the value of each resistor, we need to measure the voltages of the two resistors and solve the following system:

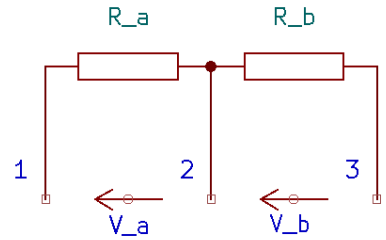

Figure 10 - Potentiometer Model

$$\begin{cases} R_a + R_b = R_7 \\ \dfrac{R_a}{R_b} = \dfrac{V_a}{V_b} \end{cases} ; R_7 = 10k\Omega \; and \; \frac{V_a}{V_b} = 1.63$$

When the resistors are found, use the resistor with the closest value you have.

$$\Leftrightarrow \qquad R_a = 3.9k\Omega \; and \; R_b = 6.2k\Omega$$

For the gain, only one of the two parts of the potentiometer is used. The protocol to find the resistors is still the same but only one resistor will be put in the board.

Thus, the resistor used is : $R_g = 27\Omega$

Now, the amplifier module is ready to be used. Here is the schematic of the amplifier with all the added modifications.

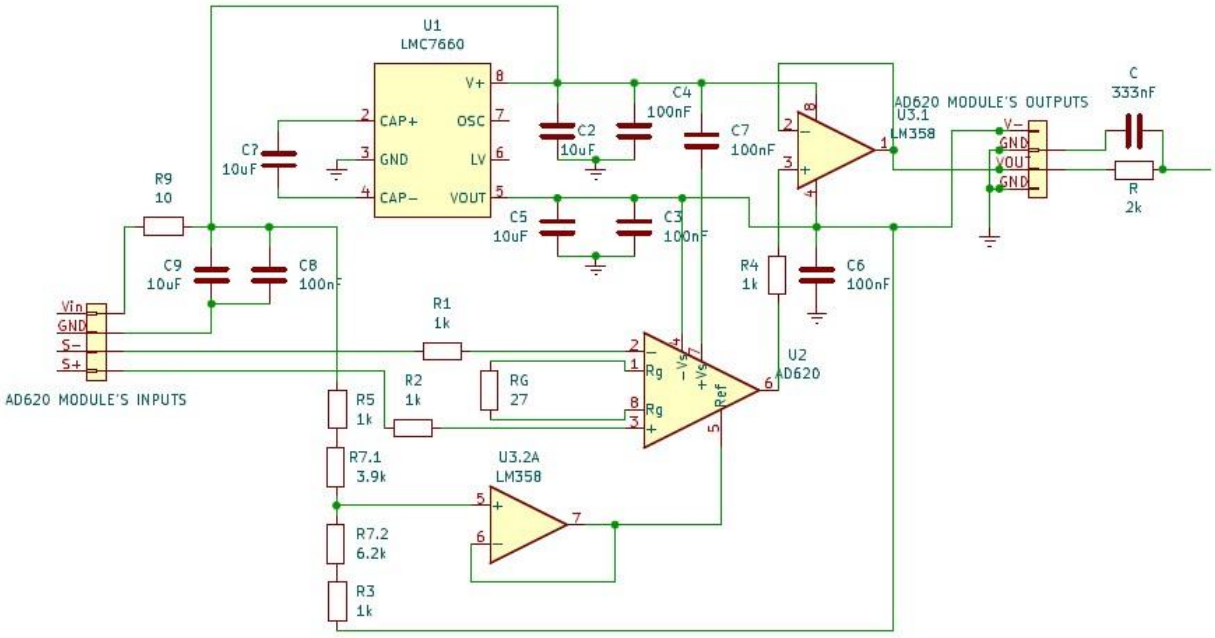

Figure 11 - AD620 schematic after modifications showing the fixed resistors that substitute the potentiometers and the output RC filter

1.4-ESC

The ESC is in charge of making the motor spin according to the command we send it.

Its input signal is a PWM signal and it generates in the output a three-phase signal from it that is usable by the motor to spin.

The PWM protocol used by the ESC is the OneShot125.

1.4.1- OneShot125

The microcontroller will have to generate a PWM signal that respects this protocol to make the motor spin.
OneShot125 is defined by the length of the throttle of one PWM period. This throttle has to be between 125µs, the minimum pulse that can be sent corresponding to a non-spinning motor, and 250µs, the maximum pulse, that corresponds to a motor spinning at full power. As the maximum pulse length is 250µs, we can send signals to the ESC at a rate of 4kHz.

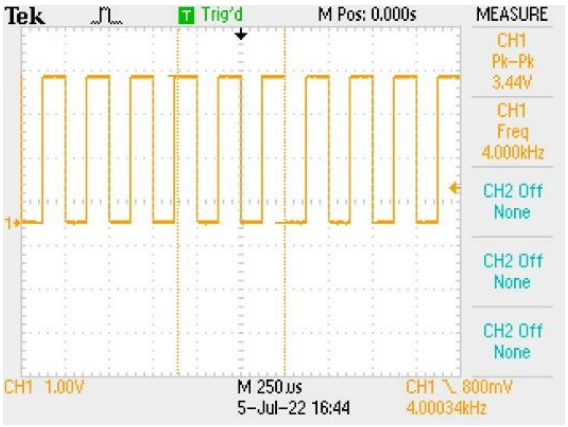

Figure 12- OneShot125 Signal at 4kHz at its minimum value

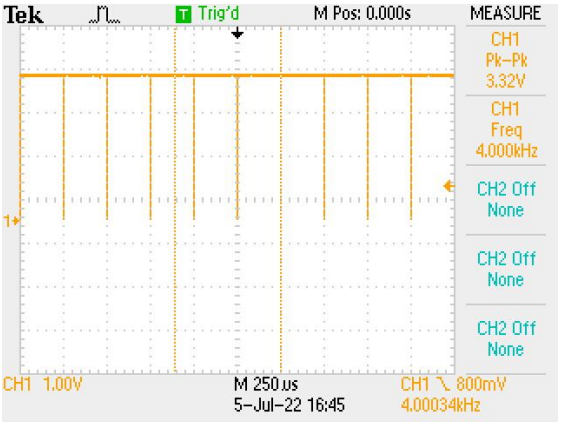

Figure 13 - OneShot125 Signal at 4kHz at its maximum value

1.5-Wiring

The different components are powered up in a certain way.

- The load cell and the amplifier are powered up by the 5V supply that can provide the microcontroller.
- The ESC powered up by a 22.2V power supply (to replicate the 6*3.7V provided by the drone's battery).

For the wiring, the following figures describe the electric connection between the components.

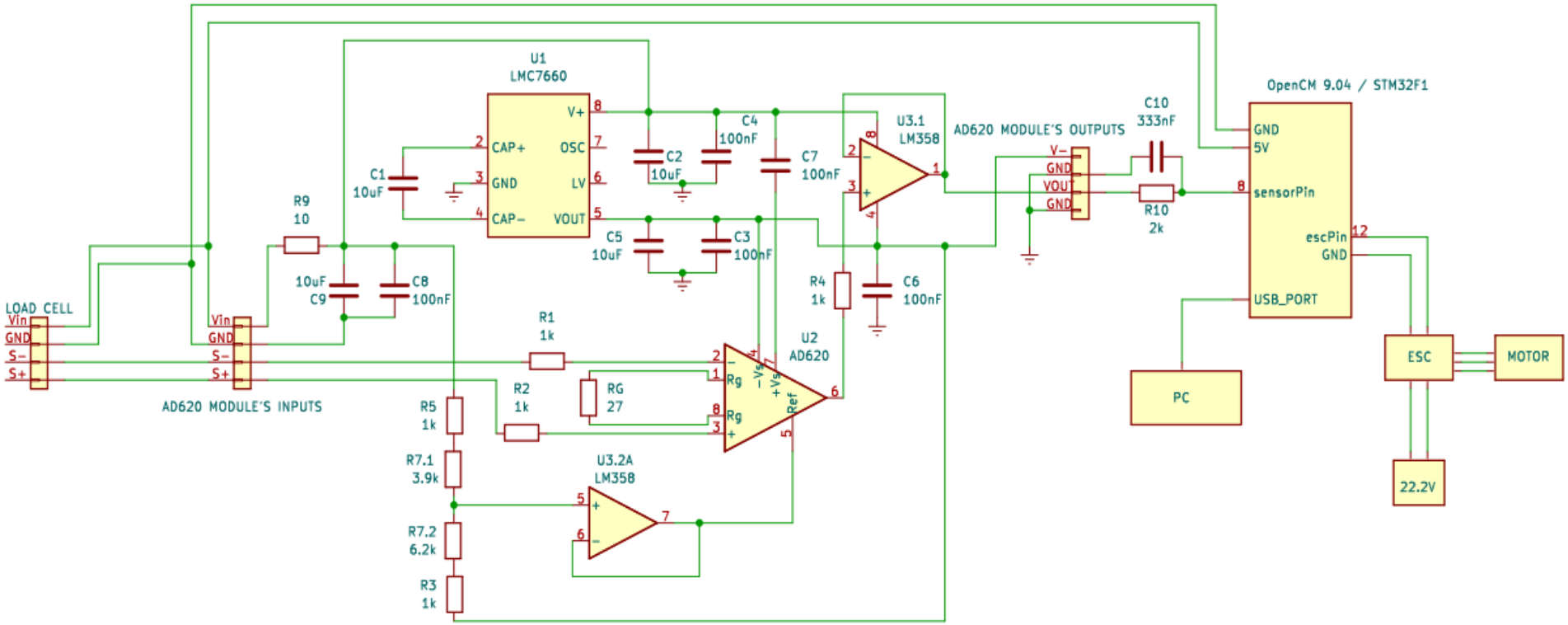

Figure 14 - Electronic schematic of the thrust controller (with the modification of the A620 module)

2-SOFTWARE

This part describes how the microcontroller manages to send to the ESC the command that will make the motor produce the thrust we want.

To make this happen, the microcontroller, in each iteration, make these tasks:

- Check if a new user set point command has been received
- Read the thrust signal from the amplifier and treat it to be usable
- Compute the output force with a PID controller with feedforwarded thrust
- Send the PWM command signal to the ESC
- Send several data to the PC (output thrust, command, …)

But first let's see how the algorithm is built.

2.1-Algorithm structure

As the ESC can receive signal at a frequency of 4kHz, we want the microcontroller's algorithm go to the same rate to be able to modify the signal sent to the motor as fast as we can.

To do so, we will use one the microcontroller's hardware timer. This timer is able to count a defined amount of time and execute an interruption when the time passed.

2.1.1-Timer interruption

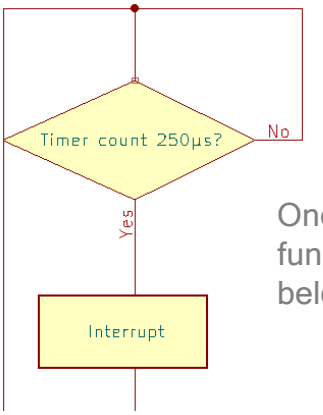

Figure 15- Schematic of the microcontroller's algorithm

The program uses on of the hardware timer of the microcontroller to make an iteration of the algorithm every 250µs.

Once the timer counted 250µs, the program enters the interruption function where it executes one iteration of the code described below.

2.1.2-Interruption function

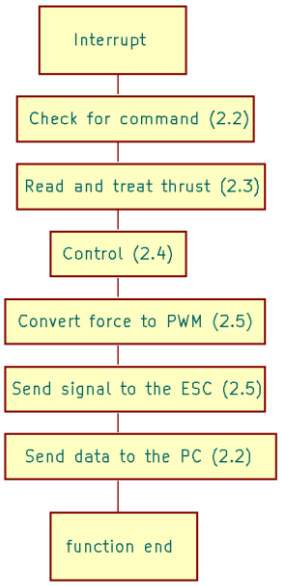

Figure 16 - Details of the tasks done in the interruption

This sums up all the task the microcontroller needs to do in an interval of 250µs.

After measuring the time passed in the interruption, we noticed that this function spends around 35µs to execute all its tasks. So, the time passed in the interruption is not problematic.

2.2- Communication with the PC

This part will detail the communication between the microcontroller and the PC; How the command messages are sent and how the data are received.

The communication between the system and the computer is made with the microcontroller through the USB serial port.

2.2.1-Receiving commands

The force command is sent by writing a message through the USB port. In a first time, this message sent to the microcontroller meant to be only force command.

However, this method showed its limits quickly as we wanted to tune the controller or tare the force sensor frequently. Thus the messages sent were no longer command messages but configuration messages that now contains:

- mode
- Force command
- Vibration filter coefficient
- Controller's coefficients (Kp, Ki, Kd, Kff in that order)

The mode parameter determines what has to be modified in the parameters:

Mode	*Action*
0	Send a command to the ESC
1	Send a controller configuration
2	Disable the controller and stop the motor
3	Tare the load cell

2.2.2-Sending data to the PC

As for the command, sending measurements started with only the measured thrust. But quickly, we wanted to see more data than that. The data sent by the controller ended up being:

- The measured thrust
- The command
- The output thrust of controller
- Its different part (proportional, integral, derivative and feedforward)
- The pulse length encoded over 10bits sent to the ESC

These data have been useful to check if the results of the controller were coherent and how should the controller be tuned according to its responses.

2.3 Real-time Visualization

Here is described the tool that has been implemented to make easier both the command sending and data displaying processes.

2.3.1-Plotter implementation

The Arduino tools like the serial monitor or the serial plotter can be uneasy and uncomfortable to work with as the speed which the data are sent is quite high.

Moreover, the graph of the serial plotter is shared by all the measurements that are printed. Thus, it is impossible to see properly two different instances of measurement at the same time.

For example, we wanted to be sure that period of the timer was constant but at the same time see if the sensor was working properly. The period was on microseconds so around 250µs and the sensor value was between -5V and 25V. Putting them on the same graph doesn't allow to see properly the variations of neither of the two signals.

Thus, a more performant tool has been implemented. It uses ros2 and its tools rqt and plotjuggler.

ROS node

First we build a ROS node. Its purpose will be to receive the data sent by the microcontroller through the USB port and publish it in a topic named thrust_measure.

This node will also have secondary goal: subscribe to another topic to collect a configuration message and structure it to make it understandable by the microcontroller.

Plotjuggler

This tool can subscribe to a ROS topic and plot the data that are in it. It can plot each data on separate graphs and their scale can be set manually which allows a way better visualization of the gathered data.

RQT

Rqt is a graphical user interface which it is possible to publish message on a topic.

Publish messages from the terminal can be tedious as there are a lot of details that need to be precise each time. Thus, using this tool allows a more comfortable and more fluid use of the controller.

2.4-Read and condition thrust measurements

Reading the voltage from the amplifier consist only in using the Arduino's analogRead() that converts an analog Voltage between 0 and 3.3V into its value over 10 bits (from 0 to 1023). Once this signal is read, it needs to be mapped into the wanted unit and treated to be usable.

2.4.1-Input Voltage / Force Mapping

Once the voltage is read, it is necessary to convert it back into a force unit.

As said earlier, the relation between the output voltage of the amplifier and the force applied to the motor is linear.

Thus, it is possible to define the mapping function between them using a reference point and a slope.

Find the slope

To determine the slope, we take several measurements of the force we apply to the motor and its corresponding analogRead() output.

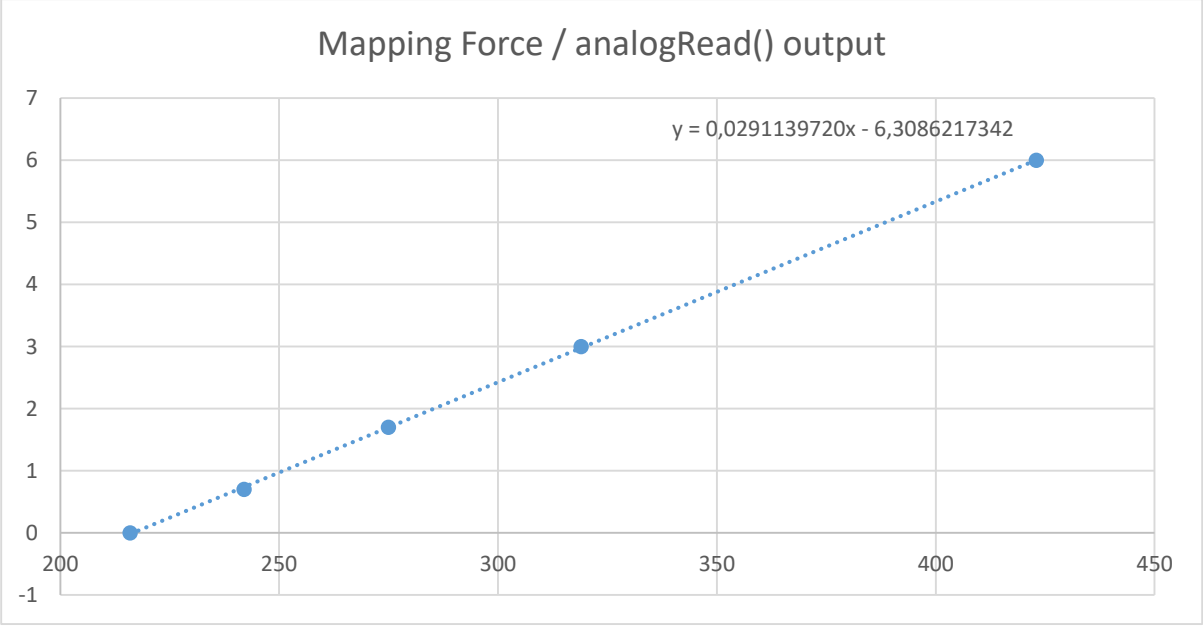

Figure 17 - Mapping of the analogRead output values into their corresponding force

Find the reference point

The reference point is the couple (0N, analogRead() at 0N).

We used this method to be able to calibrate the load cell easily.

2.4.2-Vibration filtering

The microcontroller is now able to display the force applied to the motor. However, when the motor is spinning, the signal is very noised due to the vibration created by the motor. This makes impossible to differentiate the thrust variation of small amplitude (<0.2N). Variations of this scale are not negligible as they could change the behavior of the drone in flight.

Thus, it is necessary to implement a filter that will decrease the impact of vibration on the signal.

This filter has to be fast enough to allow all the interruption function to be executed in less than 250µs.

After trying median and averaging filters the result in the given timing was not that good, the noise was still too important to have a useful signal.

Finally, the filter implemented is a 1st order low pass filter such as:

$$y(k) = \alpha x(k) + (1 - \alpha) y(k - 1)$$

It is way faster and allow us to have a noise in a scale of $\pm 0.01N$.

2.5-Implementation of the controller

The controller implemented is a PID controller with a feedforward term.

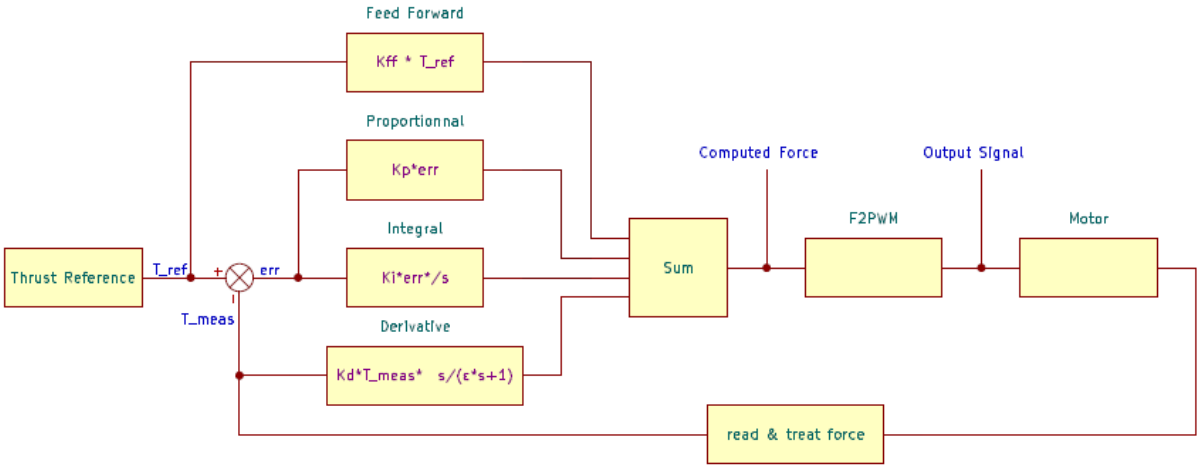

Figure 18- Detailed schematic of the controller

2.5.1-Proportional

The proportional part of a PID controller computes the error between the measurement and the command and send it to the output.

2.5.2-Derivative

The derivative part of a PID controller is used to fasten the stabilization of the force around the command. However, its use increases the noise in the signal.

Here the derivative used is an approximation of a real derivative:

$$real\ derivative\ :\ K_d \cdot \frac{df(t)}{dt} \Rightarrow K_d \cdot s \cdot F(s)$$

$$approximated\ derivative\ :\ K_d \cdot \frac{df(t)}{dt} \Rightarrow \frac{K_d \cdot s}{\epsilon s + 1} \cdot F(s)\ ;\epsilon \ll 1$$

This approximation is made to limit to increasing of the noise allowing us to have a greater derivative part without increasing too much the noise.

To implement this in the program it is necessary to convert this Laplacian expression in the Z-domain.

Using Euler backward transform, we replace: $s \rightarrow \frac{1-z^{-1}}{T_s}$ where T_s is the sampling period of the system: $T_s = 250 \cdot 10^{-6} s$

Thus, we get: $K_d \cdot \frac{1-z^{-1}}{T_s+\epsilon-\epsilon \cdot z^{-1}}$ which, converted in a recursive form corresponds to :

$$Der_{out}(k) = \frac{\epsilon \cdot Der_{out}(k-1) + K_d\left(T_{meas}(k) - T_{meas}(k-1)\right)}{T_s + \epsilon}$$

Where $T_{meas}(k)$ is the thrust measurement and $Der_{out}(k)$, its derivative at $t = k \cdot T_s$

2.5.3-Integral

The Integral part of a PID controller nullifies the error between the command and the output of the system.

In digital control the integral is made by adding successively the current error in the integral sum:

$$Int_{out} = Int_{out} + Ki \cdot err$$

In case the system is saturated and cannot reach the command sent to it,

An anti-windup has been added: when the computed Thrust passes over 14N, or when it decreases under 0N (the motor cannot provide forces over 14N nor negative forces), the integration is stopped meaning that the program stops adding terms in the integration sum:

```
if ((*thrust_compute <= ANTIWINDUP_THRESHUP) && (*thrust_compute >= ANTIWINDUP_THRESHDOWN)) {

   Iout += Ki / FREQ * err; //integration of the error if the output is not saturated
  *thrust_compute = FFout + Pout + Iout + Dout;
}
```

Figure 19 - Code part implementing the integral part with the anti-windup

2.5.4-Feed forward

The feed forward term is the response the system would have given to the command if it was in open loop. It doesn't react with the change in the feedback.

This term is useful in a controller because it shortens the rising time of the system output allowing a faster stabilization of the measurement around the command overall.

2.6-Communication with the ESC

The communication between the microcontroller and the ESC is made using a PWM signal.

Thus to send a force command to the ESC we have to know how to send a PWM signal and how to convert a force into it.

2.6.1-Signal sending

To get the best of the OneShot125 protocol, the signal sent to the ESC is a PWM signal at 4kHz.

We started by using the Arduino function analogWrite() but as its main purpose is to create an analog output using the average of a PWM signal, the result was not as good as expected: the change in the PWM were not synchronized with its period and the results was that some throttles were fused . Moreover, the basic PWM frequency of the analogWrite() is 400Hz. So we needed a way to fix both problems.

To send a PWM signal at this frequency, there are some functions in the OpenCM's board manager that are useful. Indeed, there is a PWM driver implemented in it.

So send the signal at the desired frequency and pulse require only one line of code for each:

```
drv_pwm_setup_freq(pinOut, FREQ);

drv_pwm_set_duty(pinOut, PWM_RES, dc);
```

Where:

- pinOut is the output pin on which the PWM signal is sent,
- FREQ is the desired frequency of the signal
- PWM_RES is the resolution of the duty cycle sent
- dc is the duty cycle encoded over PWM_RES bits

However, there still was glitches in the PWM signal. This was due to the way drv_pwm_set_duty updated the PWM signal sent: to modify the shape of the PWM signal, we have to modify a register of the microcontroller. This function clears the whole register before rewriting the new value. This action makes a glitch in the signal each time the register is modified. We fixed this by modifying the function to stop prevent the register from resetting but only modify the bits that are responsible of the PWM duty cycle.

2.6.3-Force/PWM Mapping

To make the motor produce a certain force we have to know what signal we have to send for it to reacts on a coherent way.

The relation between the length of the throttle and the thrust the motor creates is quadratic. The formula describing it has this form:

$$Thrust = a \cdot (PWM - 511)^2 + b \cdot (PWM - 511)$$

With PWM the length of the throttle encoded over 10bits. 511 is retracted to PWM because it is the minimum throttle that can be sent and the motor it stopped at this value. 511 corresponds to 50% of duty cycle encoded over 10bits.

Then, as we want the corresponding PWM knowing a certain thrust, we have to invert this equation:

let y = Thrust and x = PWM - 511

$$y = a \cdot x^2 + bx + c$$

$$y = a\left(x^2 + \frac{b}{a}x + \frac{c}{a}\right)$$

$$y = a\left(x^2 + 2 \cdot \frac{b}{2a}x + \left(\frac{b}{2a}\right)^2 + \frac{4ac - b^2}{2a}\right)$$

$$y = a\left(\left(x + \frac{b}{2a}\right)^2 + \frac{4ac - b^2}{2a}\right)$$

$$x = -\frac{b}{2a} \pm \sqrt{\frac{y}{a} + \frac{b^2 - 4ac}{2a}}$$

$$as\ c = 0\ and\ x\ is\ increasing\ with\ y$$

$$x = -\frac{b}{2a} + \sqrt{\frac{y}{a} + \frac{b^2}{2a}}$$

$$PWM = -\frac{b}{2a} + \sqrt{\frac{Thrust}{a} + \frac{b^2}{2a}} + 511$$

Now the formula between the wanted Thrust and the signal to send is determine, we have to find the a and b coefficients.

To do so, we take several couples of values by making the motor spin by sending specific signals to the ESC and measuring the thrust it provides.

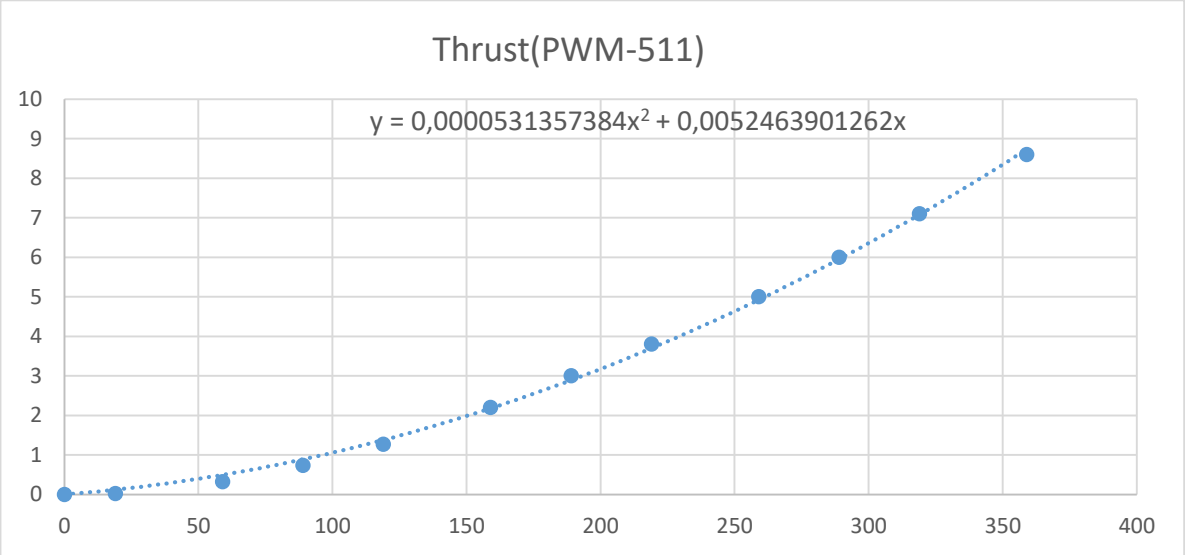

Figure 20 - Throttle length sent to the ESC and their corresponding thrust

a and b are now determined : $a = 5.3135784 \cdot 10^{-5}$; $b = 5.2463901262 \cdot 10^{-3}$.

2.6.2-Calibration

Before use, the ESC needs to be properly calibrated.

The calibration of the ESC is made by sending the maximum signal for a little period of time and the minimum signal right after.

When you power up the ESC it will emit three beeps. After these sounds, send the maximum signal and the minimum one right after. It should emit two more sounds. If it is the case, the ESC is calibrated.

3-TESTS

The test phase has the goal to make the controller work in the way we want to validate the concept. These tests were done in two phases: the controller tuning and the final test demonstrating how much thrust can the motor deploy.

3.1-Tuning

To have the desired response from the controller there were five parameters to tune: the four coefficients of the controller and the one of the vibration filter.

3.1.1-Filter coefficient

Indeed, the way the vibration filter has been made has an impact on the thrust measurement meaning that we don't have a lot of noise from the vibration but at the same time, the change in the measurement are slower. So the goal of the tuning of this parameter is to have the signal the less noised possible while keeping visible the high dynamics of the system.

After some thoughts we decided that it is more important to have a signal cleaner but slower rather than a signal that show all the variations of the thrust but noisier.

3.1.2-Controller tuning

To have the desired response of our system, the controller needs to be tuned accordingly. It is important to know what modify to obtain the result we want: increase the proportional or integral part will decrease the rising time whereas increasing the derivative part will decrease the stabilizing time.

For example, adding a derivative part will increase the noise in the signal, in a such way that in our first try we added a coefficient that was too high causing a such increase in the noise that the signal got out of the limits of what value it can reach. So the motor stopped itself according to the software security implemented in the microcontroller's code.

After a few tries, we decided that, in the context of this project, it is better to have a signal that is slower but less noisy rather than one that stabilizes faster but is noisier.

The following figure shows the thrust output from our system.

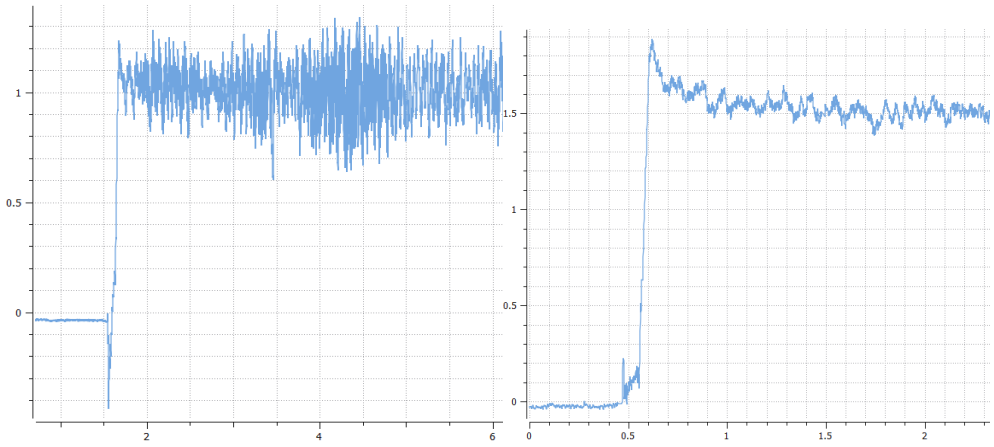

Figure 21 - Output force of the controller using different coefficient values. The x-axis describes the time in seconds and the y-axis the force in Newton

We can see on this figure that the coefficients of the controller can have influence over the shape of the response like its rising time of the output, the size of the overshoot, the stabilization time or the amplitude of the oscillations after the stabilization.

3.2-Final

For the final test, we made a command increasing by 2N until reach 16N.

The controller satisfies our expectations until 14N: the output signal has a short rising time (0.1s), it doesn't have a significant overshoot and its oscillations after the stabilization have a small amplitude(±0.1N).

However, the motor can't produce a force over 14N because of the limits of power it receives. Indeed, it was powered up through the ESC by a power supply that can provide a maximum current of 30A at the ESC's nominal voltage (22.2V). This is the power the motor needs to produce a 14N force. This force allows the drone to create a force able to carry 3 times its weight which is enough to permit it to do complex trajectories.

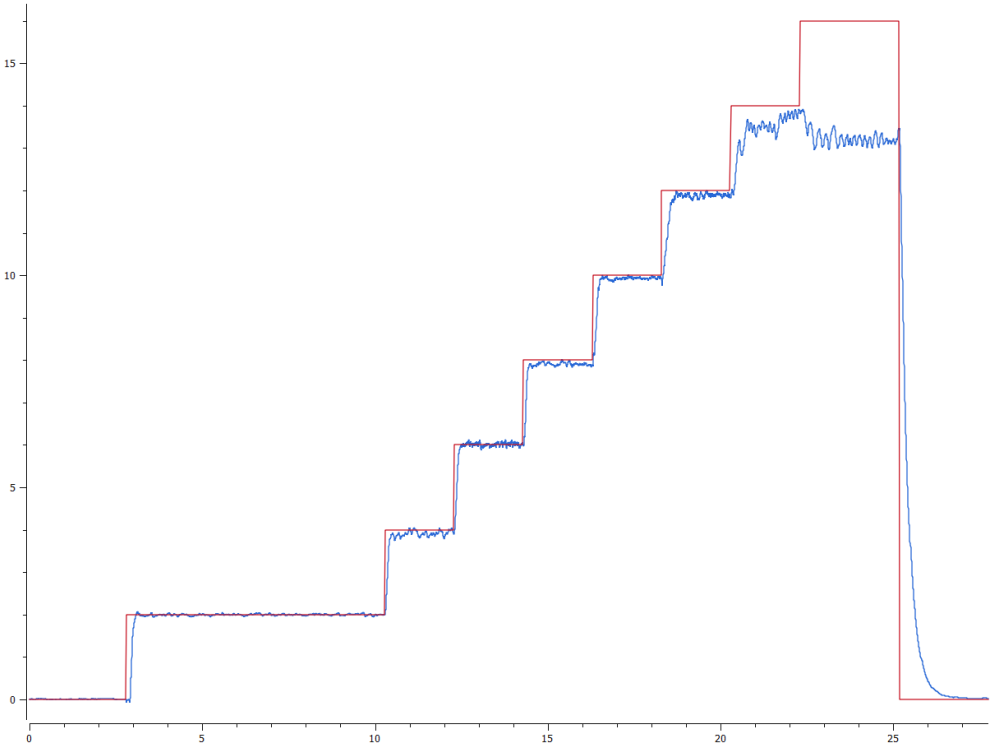

Figure 22 - Results of the final test. The x-axis describes the time in seconds and the y-axis the force in Newton The command has been increase by 2N until 16N. However the motor reached its maximum thrust at 14N.

CONCLUSION

Project's sum-up

To do so we needed in a first time to build a hardware system.

First we need a load cell to measure the forces applied to the motor. This sensor creates a voltage according to the force measured.

Then, we need an amplifier to make the force measurements readable by the microcontroller. On this amplifier a filter is added to make the measurement less noisy (the noise is created by a component in the amplifier module). Moreover, we replaced the potentiometer of the module to make the experiments repeatable.

After, we need a microcontroller to acquire the force measurement, the force command, and compute the force output to send to the ESC.

Finally, we need an ESC to make the motor spin according to the force we want to produce.

Critical analysis

The project of this internship was to implement a control in thrust of a drone motor. It had for goal to be a proof of concept in the purpose of checking if we can bypass some complicated models to compute the force in the drone. Knowing that goal, the project is a success: we managed to make the motor stabilize its thrust by using a sensor. Thus we can now totally bypass the models of aerodynamics or the influence of the shape of the propeller.

However, some things should be upgraded for it to be properly used on a drone.

In a first place, it will need a better filter to smoothen the vibration. Indeed, the one used here works fine removing the high frequency signals but it also removing some of the fast dynamics of the system. We can hear that the motor goes faster at the beginning letting think that there could be a peak in the thrust signal that has been erased by the vibration filter. So it could be interesting to have a filter that remove only the high frequency signals that have small variations in amplitude so that the peaks in thrust remain visible.

Then, as the space under the motors on the drone is limited, reducing the volume of the hardware will be necessary. It can be done by creating a PCB that will contain all the electronics necessary in one board so that the components placing is more optimized than having them on separate boards.

Finally, using a PC and communicate through a USB port is fine for testing the system but is not compatible with the use of the drone: because of the PC, the system is not mobile and the communication by USB port is voluminous knowing we have to command 6 motors in one drone. It would be preferable to use another communication protocol that is adapted with the drone's controllers.