





**Control in Thrust of a Drone Motor**

THANKS

SUMMARY

[THANKS 2](#_Toc110507678)

[SUMMARY 3](#_Toc110507679)

[INTRODUCTION 5](#_Toc110507680)

[1-HARDWARE 7](#_Toc110507681)

[1.1-Microcontroller 7](#_Toc110507682)

[1.1.1-How to use the OpenCM9.04 board 7](#_Toc110507683)

[1.2-Force Sensor 7](#_Toc110507684)

[1.2.1-Wheatstone bridge 7](#_Toc110507685)

[1.2.2-Pros and Cons 8](#_Toc110507686)

[1.3-Amplifier 8](#_Toc110507687)

[1.3.1-HX711 8](#_Toc110507688)

[1.3.2-AD620 9](#_Toc110507689)

[1.3.2.a-Output filter 9](#_Toc110507690)

[1.3.2.b-Potentiometers Calibration 10](#_Toc110507691)

[Tuning 10](#_Toc110507692)

[Replacement 10](#_Toc110507693)

[1.4-ESC 11](#_Toc110507694)

[1.4.1- OneShot125 12](#_Toc110507695)

[1.5-Wiring 12](#_Toc110507696)

[2-SOFTWARE 14](#_Toc110507697)

[2.1-Utilization 14](#_Toc110507698)

[2.1.1-Upload the microcontroller’s program 14](#_Toc110507699)

[2.1.2-Use the plotter 14](#_Toc110507700)

[2.2-Algorithm structure 14](#_Toc110507701)

[2.2.1-Timer interruption 14](#_Toc110507702)

[2.2.2-Interruption function 15](#_Toc110507703)

[2.3-Getting thrust measurements 15](#_Toc110507704)

[2.3.1-Input Voltage / Force Mapping 15](#_Toc110507705)

[Find the slope 15](#_Toc110507706)

[Find the reference point 16](#_Toc110507707)

[2.3.2-Vibration filtering 16](#_Toc110507708)

[2.4- Communication with the PC 17](#_Toc110507709)

[2.4.1-Receiving commands 17](#_Toc110507710)

[2.4.2-Sending thrust measurements 17](#_Toc110507711)

[2.4.3-Plotter implementation 18](#_Toc110507712)

[ROS node 18](#_Toc110507713)

[Plotjuggler 18](#_Toc110507714)

[RQT 18](#_Toc110507715)

[2.5-Communication with the ESC 18](#_Toc110507716)

[2.5.1-Signal sending 18](#_Toc110507717)

[2.5.2-Calibration 19](#_Toc110507718)

[2.5.3-Force/PWM Mapping 19](#_Toc110507719)

[2.6-Implementation of the controller 21](#_Toc110507720)

[2.6.1-Proportional 21](#_Toc110507721)

[2.6.2-Derivative 21](#_Toc110507722)

[2.6.3-Integral 22](#_Toc110507723)

[2.6.4-Feed forward 22](#_Toc110507724)

[3-TESTS 23](#_Toc110507725)

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 laboratory is currently working on a project: the realization of an agile drone. That means that we want a drone that reacts quickly. To do so, we need a drone controlled dynamically using the forces interacting with it rather than its position or speed. This is where this internship is located in this project.

The goal of this internship is to implement a control in closed loop of the thrust of a drone motor. In a long term the goal is to implement this controller on each motor of the drone allowing a faster and more stable control of the drone.

The current system uses an open loop for the control sending a command to an ESC that sends itself the signal to the motor that spins accordingly.

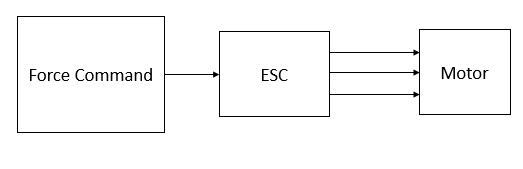


Figure 1 - Schematic of the current way to send commands to the motor

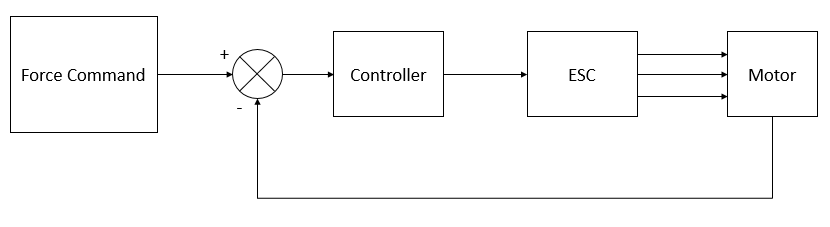


Figure 2 - Desired way to send commands to the motor

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

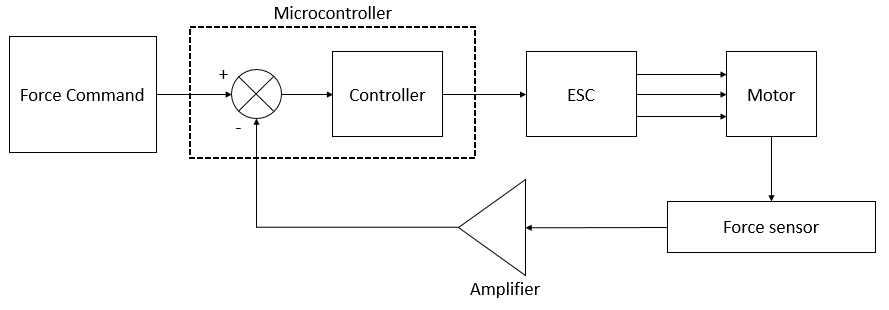


Figure 3 - Desired system explicating the components used in it

Concerning the software part, we need to implement the four following functionalities and make them work together:

* Send commands to the system and make it being able to read them
* 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

We will also want to get the data from the system to confirm the results we get.

1-HARDWARE

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

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.

1.1.1-How to use the OpenCM9.04 board

This board is compatible with the Arduino IDE. To upload a program in the OpenCM9.04, it is necessary to install the Arduino IDE and the OpenCM board Manager.

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. So 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.

1.2.1-Wheatstone bridge

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

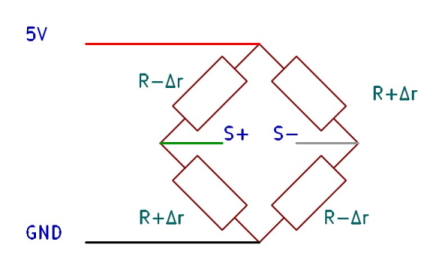


Figure 4 - Wheatstone full-bridge schematic

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 be deformed 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:

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.

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

Figure 5 - Fixation between the motor and the load cell

1.3-Amplifier

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

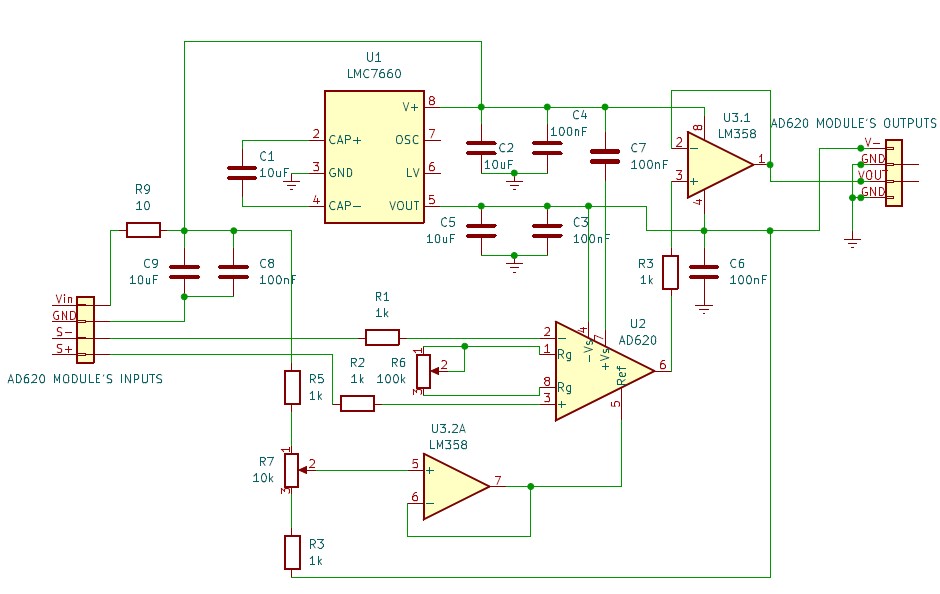
1.3.1-HX711

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 an AD620 module.



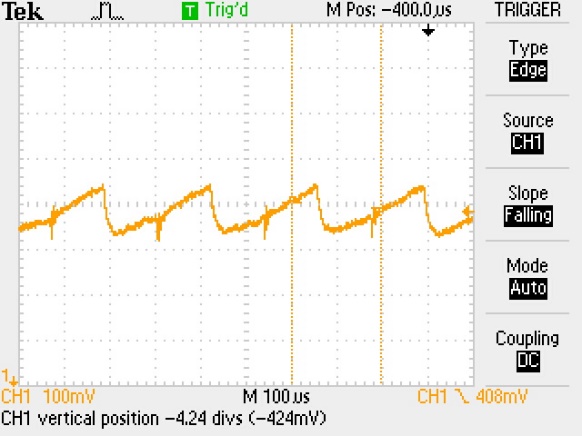
*A – Picture and electrical schematic of the AD620 module*

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 at 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:



*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:

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 2kΩ resistor and a 333nF 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 :

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:

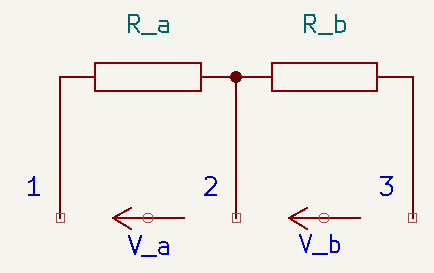


Figure 6 - Potentiometer Model

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

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 :

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

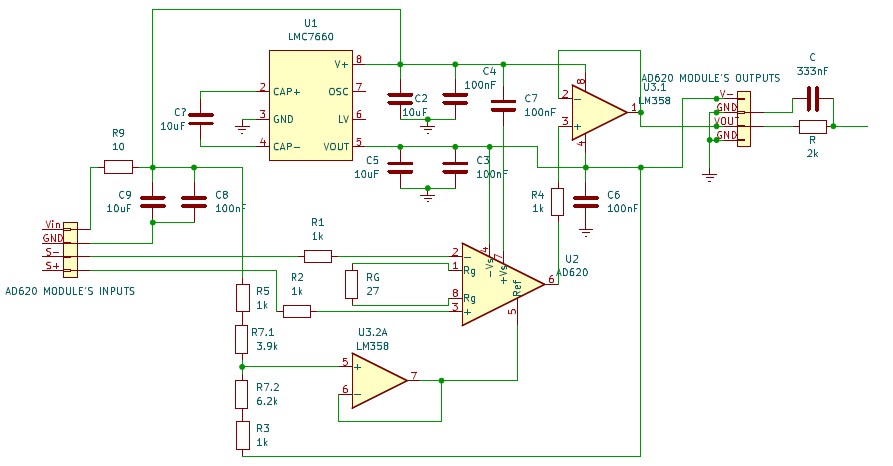


Figure 7 - AD620 schematic after modifications

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 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.

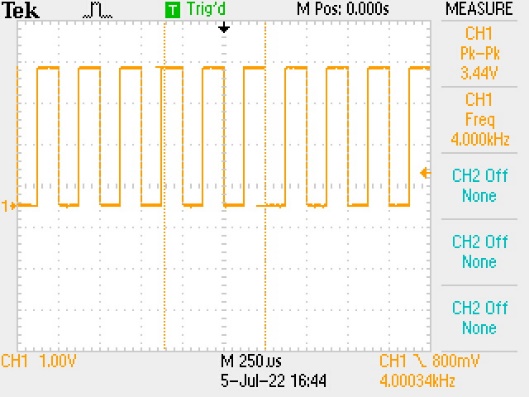
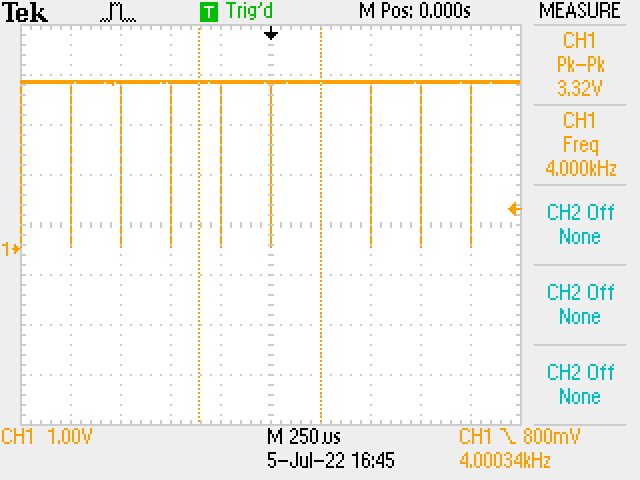
 

Figure 8- OneShot125 Signal at 4kHz

at its minimum value

Figure 9 - 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.

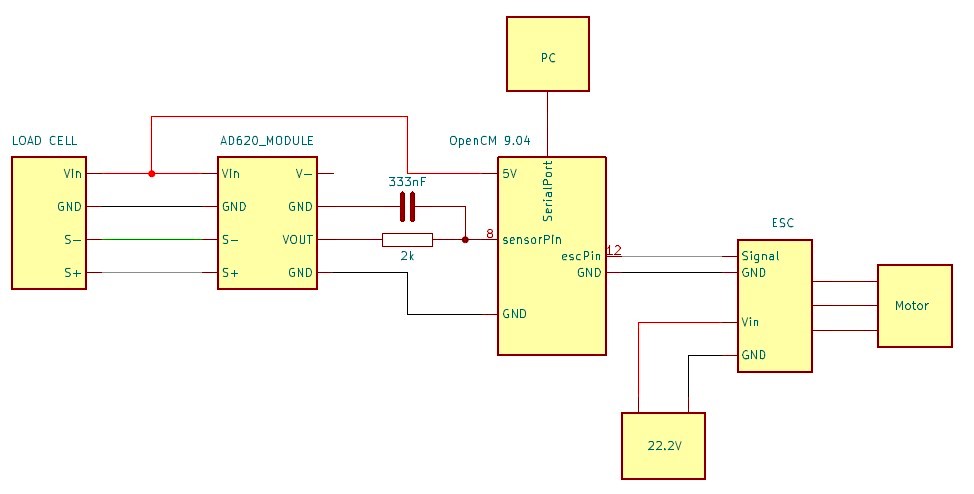


Figure 10 - Block schematic of the system

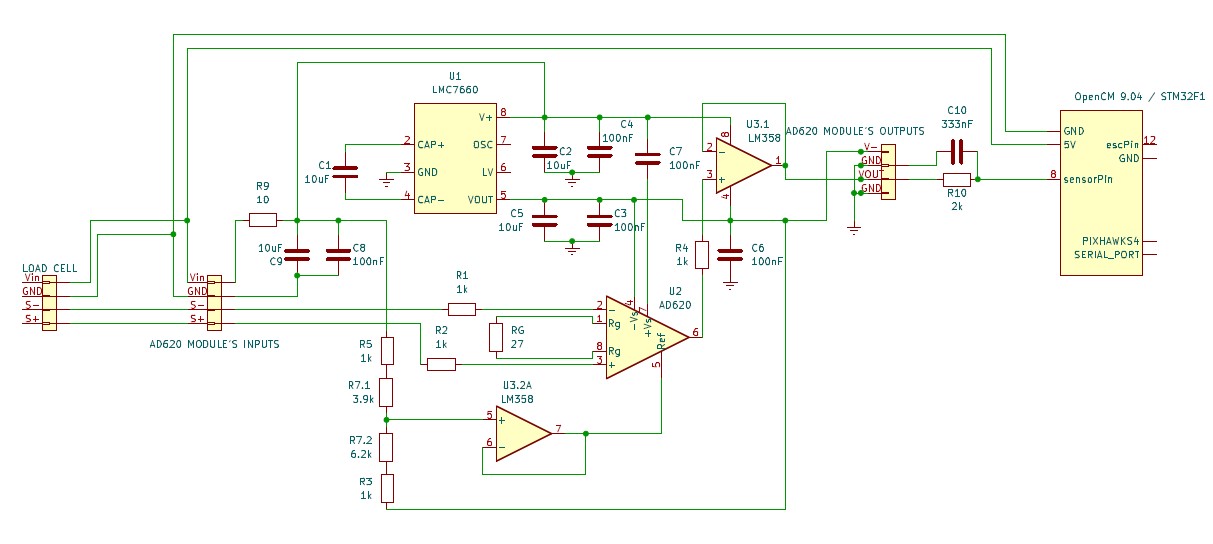


Figure 11 - Electronic schematic of the thrust controller (with the modification of the A620 module)

2-SOFTWARE

2.1-Utilization

2.1.1-Upload the microcontroller’s program

After the wiring done, upload the program “.ino” in the microcontroller by using the Arduino IDE

To use this IDE with an OpenCM board, please follow the steps described here:

<https://emanual.robotis.com/docs/en/parts/controller/opencm904/#arduino-ide>

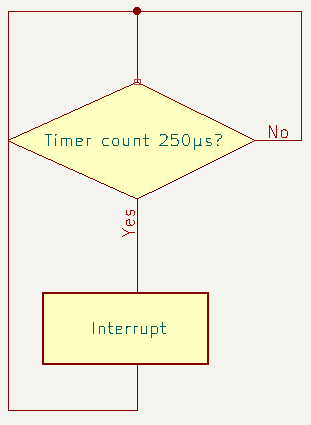
2.1.2-Use the plotter

Some requirements are needed to use the plotter implemented:

* ros2 needs to be installed ()
* Plotjuggler is the plotter used to visualize data. It needs to be installed as well

2.2-Algorithm structure

2.2.1-Timer interruption

As the ESC can receive signal at a frequency of 4kHz,

We want the algorithm to be synchronized with it. To do so, 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 an iteration of the code.

Figure 12- Schematic of the microcontroller's algorithm

2.2.2-Interruption function

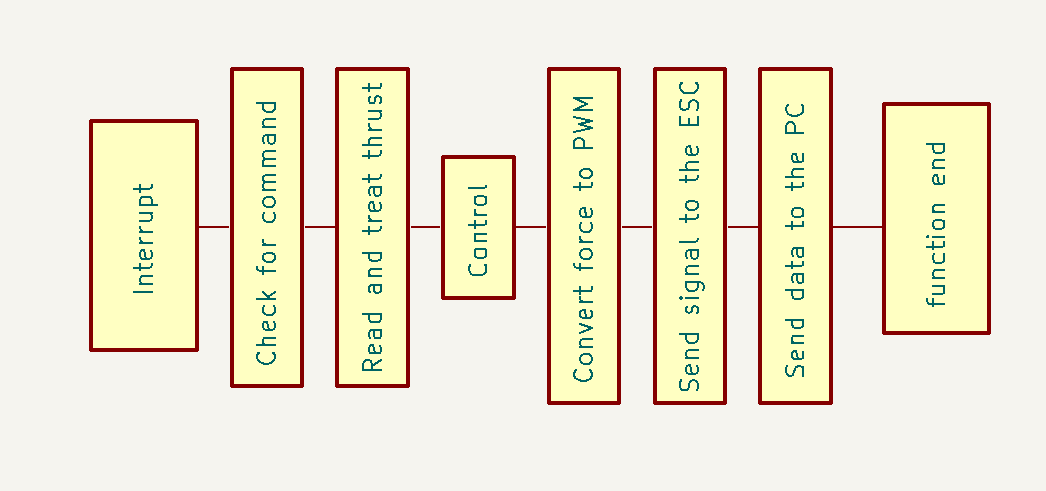


Figure 13 - interruption function of the microcontroller

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.

2.3-Getting 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).

2.3.1-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.

|  |  |
| --- | --- |
| analogRead()  (over 10b) | F (N) |
| 216 | 0 |
| 242 | 0,7 |
| 275 | 1,7 |
| 319 | 3 |
| 423 | 6 |

Figure 14 - 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 tare the load cell easily.

2.3.2-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:

It is way faster and allow us to have a noise in a scale of .

*Insert picture noised / unnoised signal.*

2.4- Communication with the PC

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

2.4.1-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)

Where the mode 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 |

2.4.2-Sending thrust measurements

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.

2.4.3-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.

2.5-Communication with the ESC

2.5.1-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.

2.5.2-Calibration

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.

2.5.3-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:

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

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.

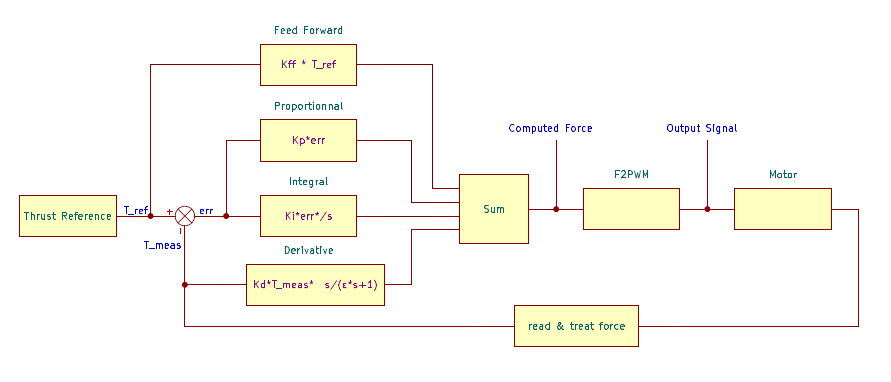
|  |  |
| --- | --- |
| PWM-511 (10bits) | thrust(N) |
| 0 | 0 |
| 19 | 0,02 |
| 59 | 0,32 |
| 89 | 0,73 |
| 119 | 1,27 |
| 159 | 2,2 |
| 189 | 3 |
| 219 | 3,8 |
| 259 | 5 |
| 289 | 6 |
| 319 | 7,1 |
| 359 | 8,6 |

Figure 15 - Throttle length sent to the ESC and their corresponding thrust

a and b are now determined : .

2.6-Implementation of the controller

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



2.6.1-Proportional

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

2.6.2-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:

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 : where Ts is the sampling period of the system :

Thus, we get : which, converted in a recursive form corresponds to :

Where Tmeas(k) is the thrust measurement and Derout(k), its derivative at

2.6.3-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:

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:

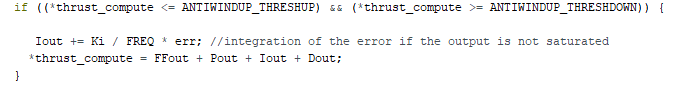


Figure 16 - Code part implementing the integral part with the anti-windup

2.6.4-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.

3-TESTS