Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Avaluació de dissenys de microvehicles aeris fets a mà i implementació d'un quatrirotor

TITULACIÓ: Enginyeria Tècnica Aeronàutica, especialitat Aeronavegació

AUTOR: Marcos Navarrete Rodríguez

DIRECTOR: Joshua Tristancho Martínez

DATA: 18 de desembre de 2014

Título: Evaluación de diseños de microvehículos aéreos hechos a mano e implementación de un quatrirotor
Autor: Marcos Navarrete Rodríguez
Director: Joshua Tristancho Martínez
Fecha: 18 de diciembre de 2014

Resumen

El objeto del presente Proyecto de Final de Carrera es construir, probar y evaluar tres distintos diseños de plantas de potencia hechas a mano. Éstas tendrán una característica común, un vuelo estacionario.

Una vez hecho esto, se alcanzarán distintas conclusiones, que dirigirán el proyecto a una solución específica. Como resultado de esta, se comprará un cuatrirotor comercial de la marca *AMEWI*, modelo *BLASTERx80*. Mediante ingeniería inversa se alcanzará a entender cómo funciona el dispositivo. Posteriormente se implementará en este otro microcontrolador pero igual IMU y planta de potencia que el MAV comercial. A lo largo de este proceso se presentarán una serie de problemas que se tendrán que ir resolviendo.

Finalmente, se determinarán las diferencias entre los Micro Vehículos Aéreos comerciales y los hechos a mano, además de la conveniencia de adquirir unos o fabricar.

Los conocimientos adquiridos a lo largo de la carrera, ayudarán a entender los problemas surgidos, identificarlos y resolverlos, consiguiendo desarrollar el presente proyecto.
Los conocimientos específicos de la carrera que se han necesitado son:

- Electrónica
- Probabilidad y estadística
- Programación C/C++
- Control
- Física

Palabras clave: Cuatrirotor, Micro Vehículo Aéreo (MAV), hecho a mano, Unidad de Medición Inercial (IMU).

Title: Evaluation of handmade micro-air vehicle designs and quadrotor implementation
Author: Marcos Navarrete Rodríguez
Director: Joshua Tristancho Martínez
Date: December, 18th 2014

Overview

The aim of the present Bachelor Final Project is making, testing and evaluating three different Micro Air Vehicle handmade power plants. All of these ones were designed and implemented with tools and materials I had to. The power plants had a common feature, hover flight.

After making and testing I got different conclusions. Those ones led me to get a decision, thus I bought a commercial quadrotor called *AMEWI BLASTERx80*. I tried to understand how the device worked (performing reverse engineering). I implemented the same commercial IMU and power plant, but I set another microcontroller. During that process some problems arose and I had to identify and solved them.

At the end, I saw the differences between commercial and handmade Micro Air Vehicles, their possibilities and future works and if it is advisable to buy commercial or make handmade MAVs.

The knowledge I have been taken for years at the university helped me to develop this project. Understanding and solving the different problems I faced.
The specific knowledge I needed were:

- Electronics
- Probability and Statistics
- Programming C/C++
- Control
- Physics

Keywords: Quad rotor, Micro Air Vehicle (MAV), handmade, Inertial Measurement Unit (IMU)

INDEX

LIST OF FIGURES

LIST OF TABLES

Acknowledgments

Dedico el presente Proyecto de Final de Carrera a mis abuelos (d.e.p.), padres, hermanas y novia por su constante apoyo y estima durante todos estos años.

A todos mis amigos hacerme desconectar cuando más lo necesitaba.

A los profesores que me han enseñado todo lo que sé y que me dieron motivación cuando no la tenía especialmente Eulalia Tramuns, Joshua Tristancho y Óscar Casas.

Trademarks

- Airscoot helicopter is a trademark
- AMEWI BLASTERx80 is a trademark
- ATX-1965 is a trademark of Bestec
- Code Composer Studio is a trademark of Texas Instruments
- eZ430-RF2500 is a trademark of Texas Instruments
- GoPro is a trademark
- HM404-2 is a trademark of Hameg Instruments
- Hubsan is a trademark
- I^2C is a trademark of PHILIPS
- Labview is a trademark of National Instruments
- Matlab is a trademark of Mathworks
- Microsoft is a trademark
- MPU6050 is a trademark of Invensense
- Proteus 8.1 is a trademark of ISIS Professional
- Robobee is a trademark
- Si2300DS is a trademark of Vishay
- STM8S005K6 is a trademark of STMicroelectronics
- T-Hawk is a trademark of Honeywell

ACRONYMS, ABBREVIATIONS AND DEFINITIONS

ACLK	Auxiliary-Clock
AMEWI	A quadrotor device called AMEWI BLASTERX80
Angle of attack:	Angle formed by the airfoil's chord and the fluid (air) speed vector.
Anisoptera:	Suborder rank beside the "ancient dragonflies"
BJT	Bipolar Junction Transistor
C/C++	Language programming
Crystal Oscillator:	Electronic oscillator circuit which uses the mechanical resonance to get a specific frequency
CTOL	Conventional Takeoff and Landing
DB-9	D-subminiature 9, Connection and pin numbers of serial hardware
DB-25	D-subminiature 25, Connection and pin numbers of serial hardware
DCE	Data Communication Equipment
DCO	Digitally-Controlled Oscillator
DTE	Data Terminal Equipment
Handshaking:	Automated process of negotiation that sets the communication parameters between two devices
HyperTerminal:	Computer program which can read Serial-communication data.
I^2C	Inter IC or Interconnected Integrated Circuit (Two wire interface)
IMU	Inertial Measurement Unit
Lift	Force generated by an object moving throughout a fluid as air or water
Lift Coefficient:	No-dimensional variable intrinsic to lift force.
LSB	Least Significant Bit
MAV	Micro Air Vehicle
MCLK	Master-Clock
MEMS	Micro-Electromechanical Systems
MOSFET-N	Transistor NPN
MOSFET-P	Transistor PNP
MSB	Most Significant Bit
NACA	National Advisory Committee for Aeronautics
NOTAR	No Tail Rotor
Null-modem:	Serial Communication without Data Terminal Equipment
Ornithopter:	Aircraft that flies by flapping its wings
PC	Personal Computer
Pin	Microcontroller Outputs
Pitch	Free-motion over aircraft's longitudinal axis
Power plant:	MAV's that takes it throttle/lift forces in order to takeoff and landing.
PWM	Power Width Modulation
Register	Address that commands some specific work into microcontrollers
Roll	Free-motion over aircraft's cross axis
RS-232	Receive-Send 232 standard protocol
SMCLK	Secondary Master-Clock

SMD	Surface-Mounted Device
Swashplate:	Helicopter device that translates rotating motion from power plant axis to the lift surfaces (blades)
UART	Universal Asynchronous Receiver/Transmitter
UAV	Unmanned Air Vehicle
URSS	Union of Soviet Socialist Republics
USAF	United States Air Force
V_{GS}	Tension between Gate and Source in any MOSFET
V_{DS}	Tension between Drain and Source in any MOSFET
VLO	Internal Very-Low-Power Low-Frequency Oscillator
VTOL	Vertical Takeoff and Landing
White Noise:	Random signal with a constant power spectral density
Yaw	Free-motion over aircraft’s vertical axis

INTRODUCTION

Nowadays, there are a lot of Unmanned Air Vehicles, commonly called UAVs. These kinds of devices offer us a bunch of possibilities, as use them in war missions or in surveillance works

The concept gave birth in 1800, although really got UAV's current idea in 1950s in the United States. The United States Air Force (USAF) was worried about a lot of aircraft pilots killed during the wars. The inflexion point was at the moment the secret aircraft U2 was shot down over the URSS land in spy mission. Thus, USAF began a development program of unmanned air vehicles [1], see Figure 1.
Technology advanced and the power plants and size of UAVs were change. Then, a new branch UAVs took place, the Micro Air Vehicles [2], depicted in the Figure 2.

Figure 1 – UAV classified by range, endurance and altitude

These new devices can get an unmanned flight and they have a small size. These micro-UAVs can have a wingspan from centimetres to metres, but not much longer. They were called Micro Air Vehicles, known as MAVs.

Currently, as aircrafts makes the human being live easy, MAVs do it too. That was a good reason to choose the theme of this project.

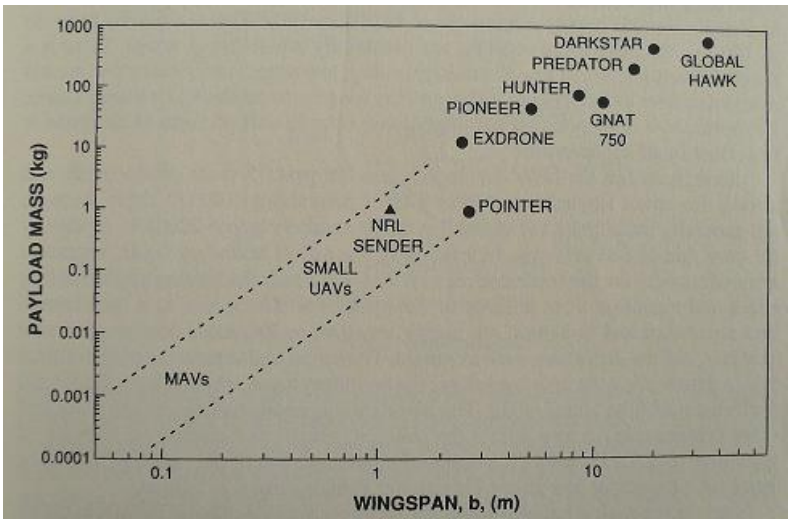


Figure 2 – UAV payload vs. Wingspan

The aim of this Bachelor Final Project is making, testing and evaluating three different Micro Air Vehicles handmade power plants with a common feature, hover flight.
I designed and made these ones with all I could find into any ironmonger's.
I evaluated them and got different preliminary conclusions (see chapter 3). Those ones drove the project to a specific solution. As a result of it, I had to buy a commercial quadrotor called *AMEWI BLASTERx80*. The next step was trying to understand how it works, performing reverse engineering. That was not easy and I had to face other problems.

I solved the problems thanks the knowledge I had learnt for years at the university. Specifically the following subjects:

- Electronics: To choose the best transistor to control the power of DC motors I needed and make the circuitry too.
- Probability and Statistics: To solve noise/vibration problem in the MAV/IMU.
- Programming C/C++ and Pseudo.-C++: To make a program into the compiler called Code Composer Studio for microcontroller and setting IMU and Serial Port communication.
- Control: To develop a program which control the DC motors.
- Physics: To understand Lift vs. Current/ PWM graphs and the flight dynamics.

Finally, I got many conclusions. Getting those ones I became aware of how difficult are the design and implementation in a MAV.

CHAPTER 1. STATE OF THE ART

Micro Air Vehicles have taken an important role in our society. We can use them for enjoying (mini-helicopters[1]), as playing with them, for protection (PD100 Black Hornet[2]), as Army's device or taking them for other uses, as making home videos (GoPro quadrotor[3]). See those ones in the Figure 3.

Figure 3 – Mini-helicopter, US army's MAV and GoPro quadrotor

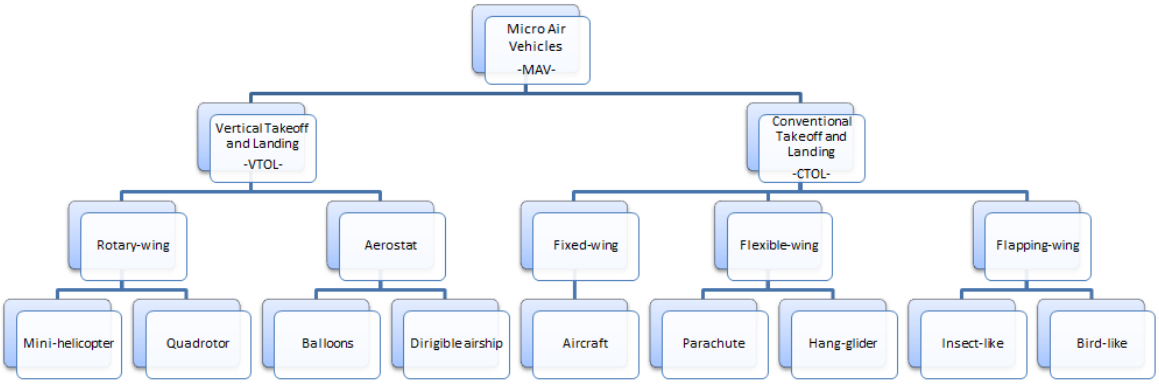
1.1 MAVs state of the art

What is a MAV specifically? The Micro Air Vehicles belong to a branch of Unmanned Air Vehicles. The main difference between MAVs and UAVs is the size and sometimes wings feature (Table 1).

What is an UAV? Unmanned Air Vehicle is a device which does not need anybody inside to control its movements. However, they need to be piloting by a human being or a computer through a radio controlled set on it (Navigation System).

MAVs are a result of different evolution areas of technology. These fields, as materials, electronics or physics improve the features of MAV getting new levels in flight, communication, surveillance…

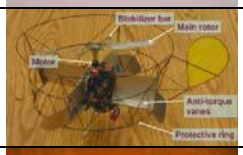
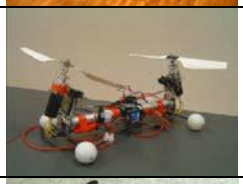
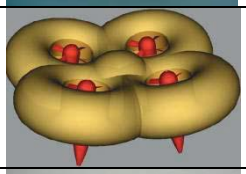
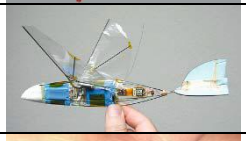
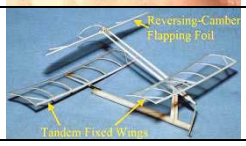
Table 1 – Kind of MAV's wing configuration



I needed to understand what are the pros and cons of the wings features [3], depicted in the following Table 2.

[1] http://www.pixfans.com/el-helicoptero-radiocontrol-mas-pequeno-del-mundo/
[2] http://www.proxdynamics.com/products/pd-100-black-hornet-prs
[3] http://www.banggood.com/Walkera-QR-X350-GPS-GOPRO-Auto-Pilot-RC-Quadcopter-p-80925.html

Table 2 – MAV wing configuration.

Configuration	Picture	Advantages	Drawbacks
Fixed-wing		- Simple mechanics - Silent operation	- No hovering
Single		- Good controllability and maneuverability	- Complex mechanics - Large motor - Long tail boom
Axial rotor		- Compactness - Simple mechanics	- Complex aerodynamics
Coaxial rotor		- Compactness - Simple mechanics	- Complex aerodynamics
Tandem rotors		- Good controllability and maneuverability - No aerodynamics Interference	- Complex mechanics - Large size
Quadrotors		- Good maneuverability - Simple mechanics - Increased payload	- High energy consumption - Large size
Blimp		- Low power consumption - Auto-lift	- Large size - Weak maneuverability
Hybrid		- Good maneuverability - Good survivability	- Large size - Complex design
Bird-like		- Good maneuverability - Low power Consumption	- Complex mechanics - Complex control
Insect-like		- Good maneuverability - Compactness	- Complex mechanics - Complex control
Fish-like		- Multimode mobility - Efficient Aerodynamics	- Complex control - Weak maneuverability

Knowing and understanding what a MAV is, and how many wing configurations are, I just had enough information to choose through the pros and cons features the flight characteristic I want to. Hover flight configuration.

1.2 MAVs with hover capabilities

MAV with hover capability gives us a lot of possibilities; we can control it since the takeoff take place. That is a good reason to choose this specific kind.

We can find thesis, projects, technologic and scientific articles, which have been study for a long time different ways to get a hover fly, searching answers in the environment [4], as *Robobee* (Figure 4), or improving current systems (Figure 5), as *T-Hawk*.

One famous hover flying device is a Micro Air Vehicle called *RoboBee*. This device has a power plant based on flapping-wing and has only $80\ mg$ [5]. It is as tiny as one American cent, depicted in Figure 4

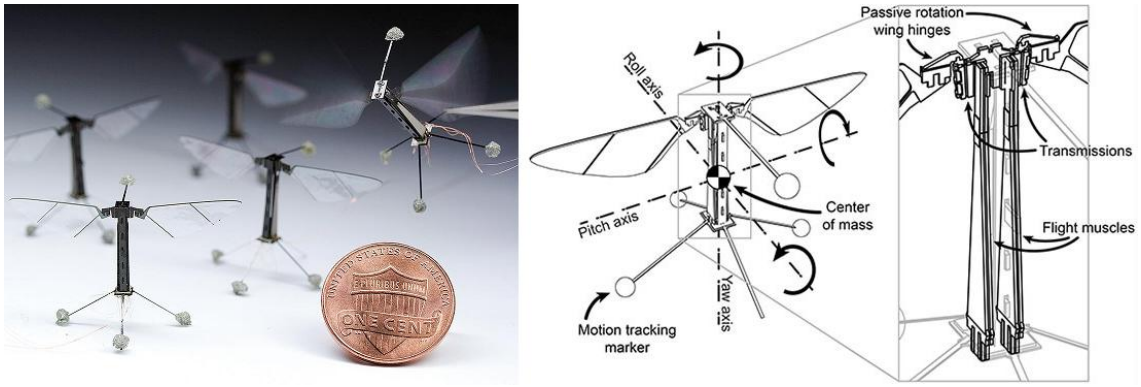


Figure 4 – Miniature flapping-wing robot, *Robobee*[4].

Although, not always is necessary a MAV as tiny as *Robobee*, or use that power plant. There are examples as *Honeywell T-Hawk*[5] prototype (Figure 5), which was designed with a coaxial rotor configuration and big size too. It has a lot of sensors, cameras and different kind of communications inside.

Figure 5 – Honeywell T-Hawk (tarantula hawk) MAV developed by the US Army

[4] http://wyss.harvard.edu/viewpressrelease/110/

[5] http://www.army-technology.com/projects/honeywell-thawk-mav-us-army/

Another good example is the mini-quadrotor called *Hubsan Q4*[6], its size (as you can see in Figure 6) is tiny enough as it could takeoff on your own hand.

Figure 6 – Mini-quadrotor Hubsan Q4[6]

I chose the last power plants to make them with my own hands. The process to make a MAV power plant in this project will be both amateur and handmade.

[6] http://www.hubsan.com/products/HELICOPTER/H111.htm

CHAPTER 2. MICROAIR VEHICLE DESIGN SELECTION

This chapter will explain the three different power plants with hover flight feature I chose. Furthermore, how many pieces I needed, and how I implemented them. The problems and solutions I faced take place in the following pages. The whole chapter will determine the final way of the project.

2.1 Dragonfly design

Before designing the *Dragonfly MAV* we need to know how this kind of MAVs has got the present. I could find a bunch of devices, but they were more or less structurally similar.
Some of the most common MAVs are based on ornithopter's mechanics, like a seagull flight or any other animal flapping their wings. The flight mechanics is based on pushing down the air, as a result of it the device glides through the air. A famous example of this kind of device is in the sketches that Da Vinci drew, called Da Vinci's ornithopter[7] (Figure 7). He kept attention on the birds' flight.

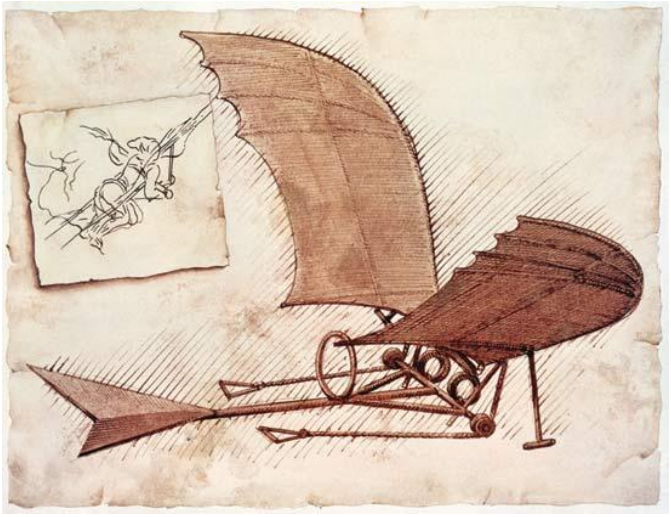


Figure 7 – Da Vinci's ornithopter

I did not want to design an ornithopter device, instead of that; I needed to get an anisoptera flight. Anisoptera flight is different from ornithopter flight; it is not only a common flapping. Flapping is present in the movement, but the axis of wings rotates to obtain an effective flight [6], see Figure 8.

[7] http://www.britannica.com/EBchecked/media/95182/Leonardo-da-Vincis-plans-for-an-ornithopter-a-flying-machine

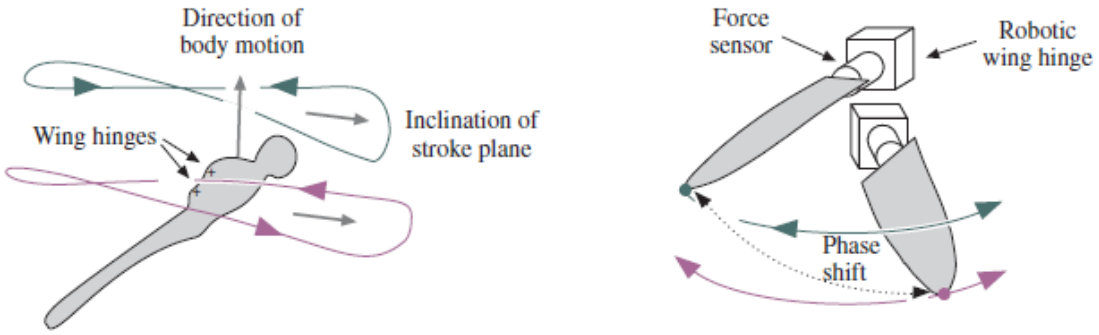


Figure 8 – Anisoptera wing movement

The part that gives movement to wings in ornithopter mechanism is formed by gear wheels, DC motor and set up on a chassis.
The power system was similar to any ornithopter device, as I did not want to change it. The mechanism had pieces which change the rotating movement to vertical and linear movement [7]. The mechanism worked using the same basic mechanic movement as combustion engine[8]; depicted in Figure 9.

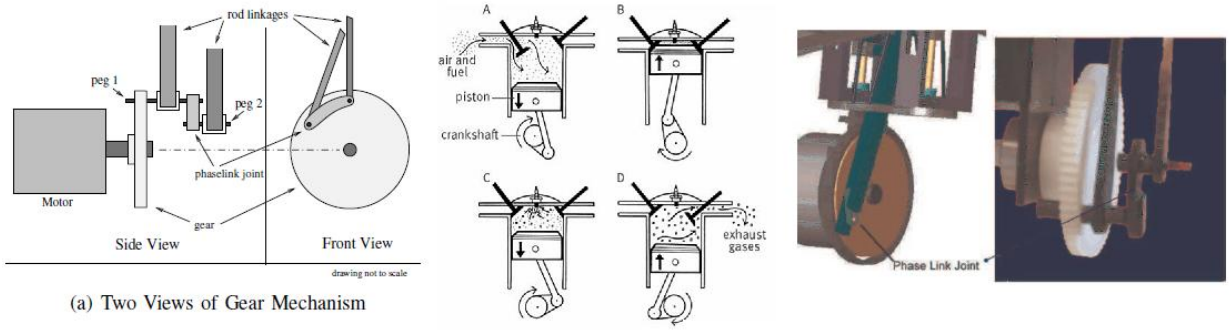


Figure 9 – Ornithopter mechanism and mechanic movement combustion engine

DC motor gave circular movement to the gear wheel; that movement became vertical and linear by a linker (that's the way ornithopter MAVs get the flapping movement). I used this system to obtain the flapping movement but not the rotational one.
I obtained rotational movement using the air drag to change the position along the wing axis(Figure 8 and 13), when the wing got the top or bottom edge of flapping movement, the wing was submitted to air forces and these ones changed its position again. That means, always the wing got the position with less drag and high lift depicted idea in Figure 13.

2.1.1 Dragonfly development objectives

The *Dragonfly Micro Air Vehicle* had the following characteristics:

- Good maneuverability.
- Low power consumption.
- Compactness.

[8] http://astarmathsandphysics.com/gcse-physics-notes/gcse-physics-notes-the-internal-combustion-engine.html

- VTOL – Vertical Takeoff and Landing
- Small
- Easy mechanism

2.1.2 Handmade dragonfly design

The design of this MAV was based on anisoptera flight (dragonfly flight), see Figure 10. Insect-like configuration (Table 2) was not easy making. The pieces of the mechanism were both tiny and light. Materials as plastic or wood gave some solutions. The design needed four flapping wings, two DC motors, two gear wheels, two servomotors, a chassis to set the parts and the battery.

Figure 10 –Dragonfly

I found information about flapping wings mechanics [6], depicted in the Figure 12 and 13, which was a good help to understand and do the MAV. I did some sketches based on this scientific information (see Annex A7).

The first designing critical point was changing circular movement to vertical one. I did a sketch depicted the main mechanism, see Figure 11.

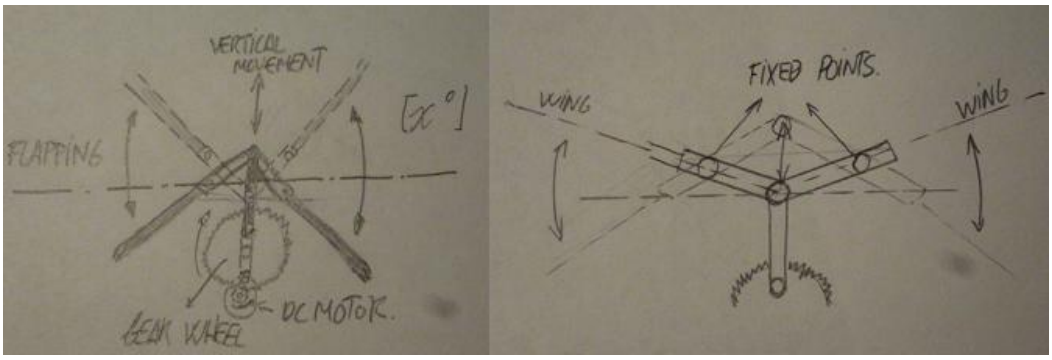


Figure 11 –Mechanism that change circular movement to vertical one

The second movement was based on how work the dragonfly wings. They are a complex movement; I used the Figure 12 to understand these movements.

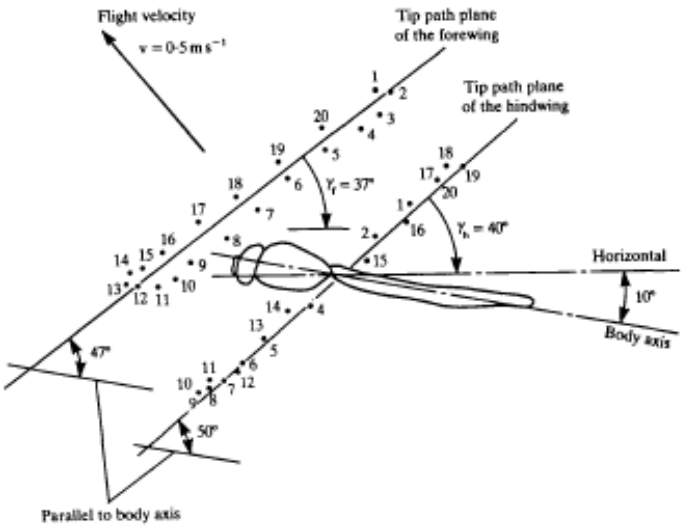


Figure 12 – Orbits of wings

When I got the last both movements, I needed a mechanism which changes the position along wing axis.
Rotational to vertical movement mechanism (Figure 11), work together with linearity actuators of servomotors (Figure 26) and the rotational passive system along axis wing. The first mechanism determines the orbit of wings and the second one change the position of wings along their axis [4] (Figure 11, 12 and 13).

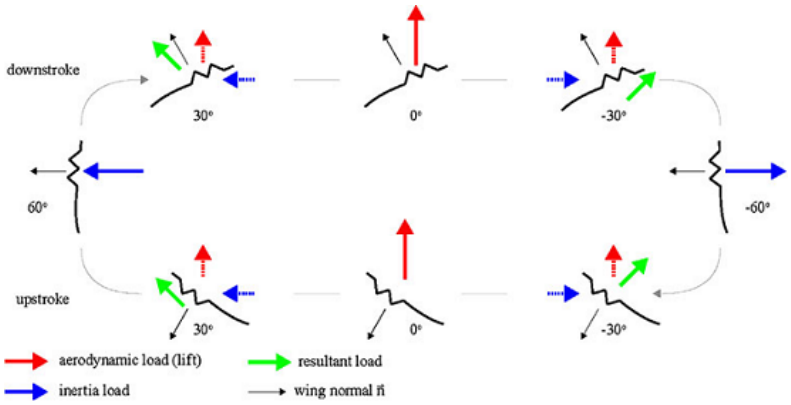


Figure 13 – Flight mechanics dragonfly wing

How dragonfly wings react in a fluid as air was the following step. Dragonfly wing structure is very complex; it is a perfect model in the nature how the evolution works. The wings have irregular airflows [8], see Figure 14. Seeing that the airfoil was complex and I could not make one, I changed that airflow wing to another easier; I set up on the MAV lift surfaces that belonged to helicopter MAV (see Figure 19 and equation (2.1)).

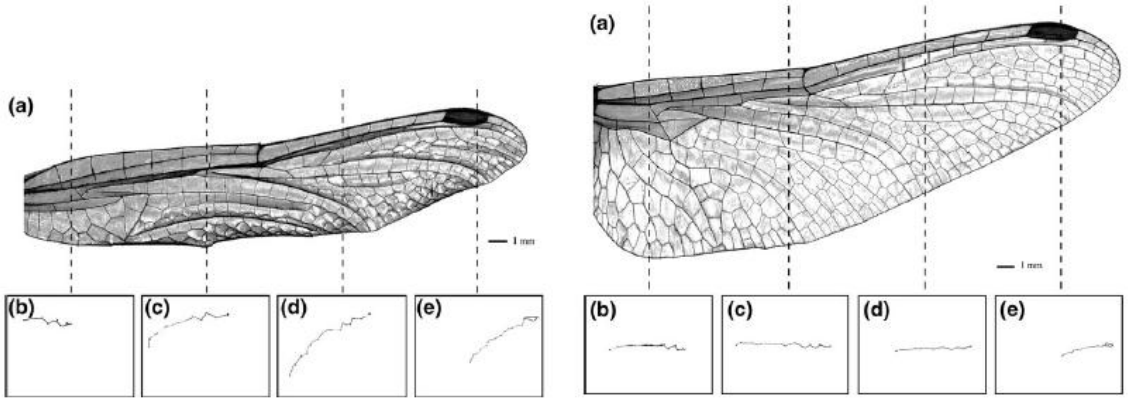


Figure 14 – Airflow of dragonfly wings.

Finally, I designed the chassis. Last parts I described are joined on this one. It fixed, DC motors, gear wheels and wing's mechanism In a very tiny space, see Figure 15.

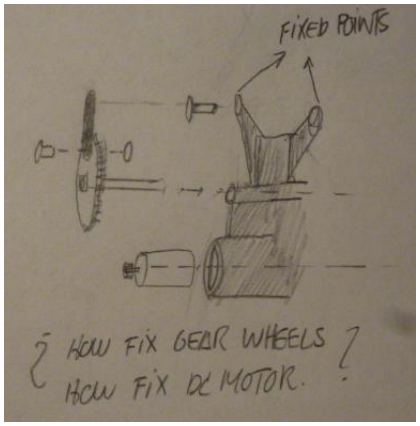


Figure 15 – Sketch chassis

It was a complex component. It needed several holes with exactly sizes and positions. The rest of components were set up in the chassis (formed by one piece). Furthermore, it has to be strong enough to bear crashes and vibrations.

Dragonfly MAV was made by all I could find into an ironmonger's and some special parts in specific stores, as gear wheels or lift surfaces.
As there are no similar MAV commercial I had to do all, chassis and wing's mechanism by myself. Doing all by me had pros and cons:

- Pros: I could design any piece on my own.
- Cons: All pieces I made were imperfect than industrial ones. Handmade parts were also weaker than industrial pieces.

2.1.3 Dragonfly implementation

To implement the dragonfly MAV, I needed material (Figure 16) as:

- Carbon fiber stick

- Wires
- Plastic
- Brass
- 2x DC motor
- 2x Servomotor
- Servomotor gear wheels
- 2x Gear wheel
- Tools (screwdrivers, drill, glue, *Dremmel* drill, …)

Figure 16 – Materials used in *Dragonfly MAV*

Making *Dragonfly MAV* I followed next steps:

- In the first step, I had to insert the DC motor in a piece of plastic and checked if the gear wheel of DC motor fit with any servo's gear set I had. Fortunately one worked.

- In the second one, both gears (DC motor and wheel) had to work together. I set DC motor in the hexagonal plastic piece (Figure 16) making holes with a drill. After several attempts, I was able to attach gear wheel in an exact place.

- In the third one, I was aware of the mechanism did not work rightly, as between DC motor and gear sometimes there was too pressure and sometimes not. The problem was the holes position; it had to be too accurate. After a lot of attempts I got more or less the best point I could, and the mechanism worked.

- In the fourth step, I set a wire between gear wheel and the wing's mechanism, but the fastener was so imperfect and the device had a bunch of vibrations and malfunctioning movement. I couldn't make a chassis, as I had the same problems of accurate with the holes. In addition, the wood was not the best solution.

- Finally, I could not solve the vibrations and the malfunctioning. Best tools would have helped to get my idea, as professional machines. Making the components in wood took much time and those ones were not strong enough.

2.1.4 Dragonfly prototype results

The results were:

- Handmade inefficient mechanism. It had not a smooth movement to reduce the vibrations at high speed.
- Unsuitable tools and material (3D printer).
- Never got the next implementing steps (wings movements), as I could not solve the rotation-linear mechanism.

2.2 Coaxial rotor design

Coaxial Rotor MAV configuration is commonly used in devices called mini-copters, micro-copters or mini-helicopters see Figure 3. The chassis design and coaxial rotor configuration allows a huge vertical stability. On the contrary, they need a lot of power during the takeoff, flight and landing, because their wings need a lot of speed to get enough thrust to flight. As the reader can see in Figure 17 the power plant design is simple but very effective.

The power plant system is made by two gear wheels and two DC motors working together. Each DC motor gives movement to each coaxial rotor. These ones work independently and with each rotor, see Figure 20.

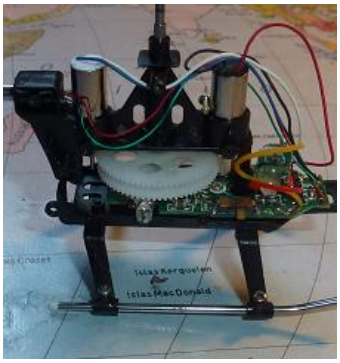


Figure 17 – Power plant commercial mini helicopter

Vertical axis is setting by coaxial rotors; each one is absolutely fixed to the vertical axis and cannot take any movement than rotational. The top rotor has a passive mechanism, which stabilizes the device in case of turbulences. That mechanism has a stick or bar with two well balance loads, this item is called flybar[9]. It works as any gyroscope keeping the blades balance changing their pitch angle. In addition, flybar reduces the effect of cross wing and avoids vibration. *Flybar* (Figure 18) can be connected to either the upper or bottom (Figure 21) rotor.

[9] http://www.robotmarketplace.com/products/0-EFLH2219B.html

Figure 18 – Flybar commercial mini-helicopter

This balanced system takes an important role keeping the best angle of attack (Figure 19) on lift surfaces, getting the highest lift (equation 2.1) each time.

$$L = \frac{1}{2}\rho V^2 S C_L \qquad (2.1)$$

Where:

- Lift (L): It is the force generated by the blades moving through the air [$Newtons$].
- Air density (ρ): It is the density of the air fluid in a specific altitude [$Kg/m3$].
- Speed (V): It is the velocity that the airflow goes through a fluid [m/s].
- Surface (S): It is the whole wing area [m^2].
- Lift Coefficient (C_L): It is a constant with no units that every airfoil has inherently. It takes a specific value in each wing.

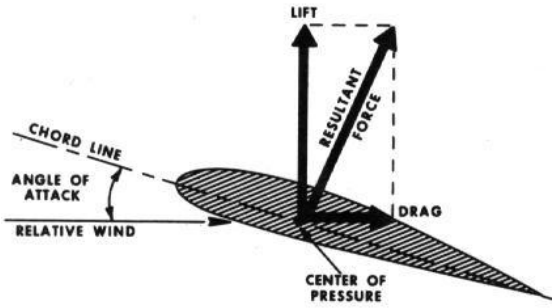


Figure 19 – Airfoil wing forces[10]

2.2.1 Coaxial rotor development objectives

The Coaxial Rotor Micro Air Vehicle had the following characteristics:

- Good controllability and stability
- High consumption.

[10] http://avstop.com/ac/flighttrainghandbook/forcesonanairfoil.html

- Compactness.
- Light
- Small
- Complex mechanism
- VTOL –Vertical Takeoff and Landing- .

2.2.2 Coaxial rotor design

Coaxial rotor was made by all I could find into any ironmonger's, I found some special parts in specific stores, as lift surfaces, Dc motors or gear wheels.

I found some *mini-helicopters* on different commercial surfaces. These ones do not have the mechanism I wanted to (Figure 23).
I could use some parts of commercial mini-helicopters, as wings, gear wheels, chassis parts…the rest of pieces, as actuators inputs, pitch link, rotating plate, no rotating plate…I had to make by myself (explaining in the following paragraphs).

The *Coaxial Rotor MAV* power plant implementation was less complex than *Dragonfly MAV* (see section 2.1 *Dragonfly MAV* design). The mechanism was based on helicopter coaxial rotors (Figure 22). These devices have two pairs of wings working in opposite directions over the same vertical axis. That configuration is due to the Third Newton Law, this explains about action-reaction in a system, the sum of the rotational pair of wings rotational moment has to be null. That means the MAV will not rotate on their vertical axis; see Figure 20.

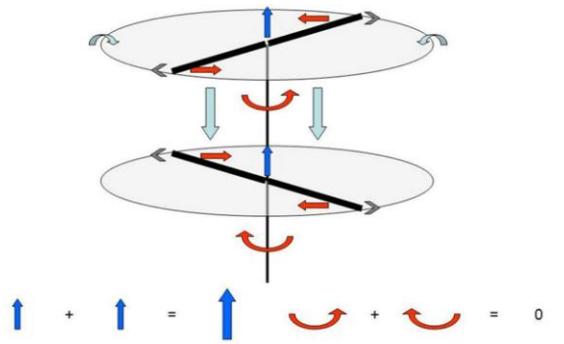


Figure 20 – Coaxial rotor Thrust (blue) and Torque (red)[11]

The materials were the same as *Dragonfly MAV*, but the size parts increased. The new size helped us to design and made the pieces relatively easy.

Although, I had gear wheels and DC motors (a wheel for each DC motor, see Figure 17), technically I had not the same vibration problem of *Dragonfly MAV*, as some *Coaxial Rotor MAV* parts has industrial accuracy.
Coaxial Rotor MAV was more or less similar to *AirScoot helicopter* [9] and *First Coaxial Rotor prototype* (Figure 21). The main difference was that *Coaxial Rotor*

[11] http://www.simhq.com/_air13/air_427a.html

MAV had some features as *Coaxial Rotor Prototype*; it had not set tail rotor for example. Devices with this specific feature are called NOTAR (No Tail Rotor).

Figure 21 – AirScoot helicopter MAV and Coaxial Rotor Prototype-

I had to think thoroughly the design and check it with others coaxial rotor mechanism [10], depicted in Figure 22.

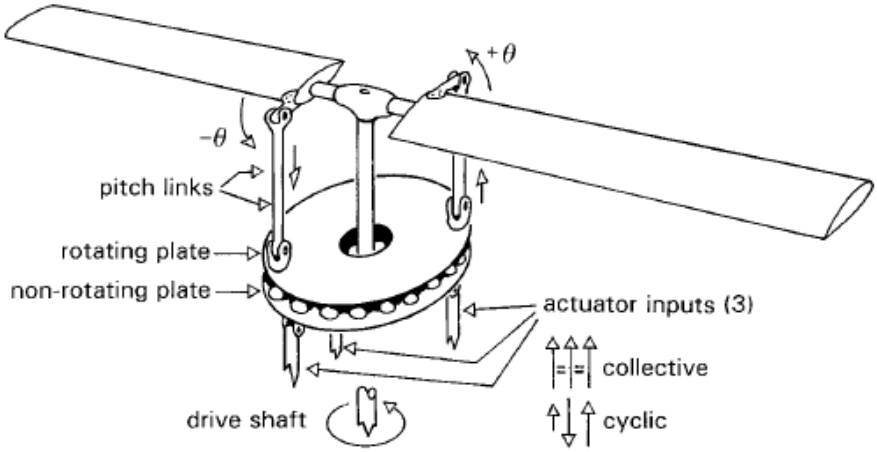


Figure 22 – Coaxial rotor with pitching and rolling control

I made the coaxial rotor MAV design with the same tools and material as *Dragonfly MAV*. But I needed to design that mechanism two times for each DC motor, as in the Figure 22 only there is one rotor.

I did several sketches (Annex A7) about the mechanism; one of it is depicted in Figure 23.

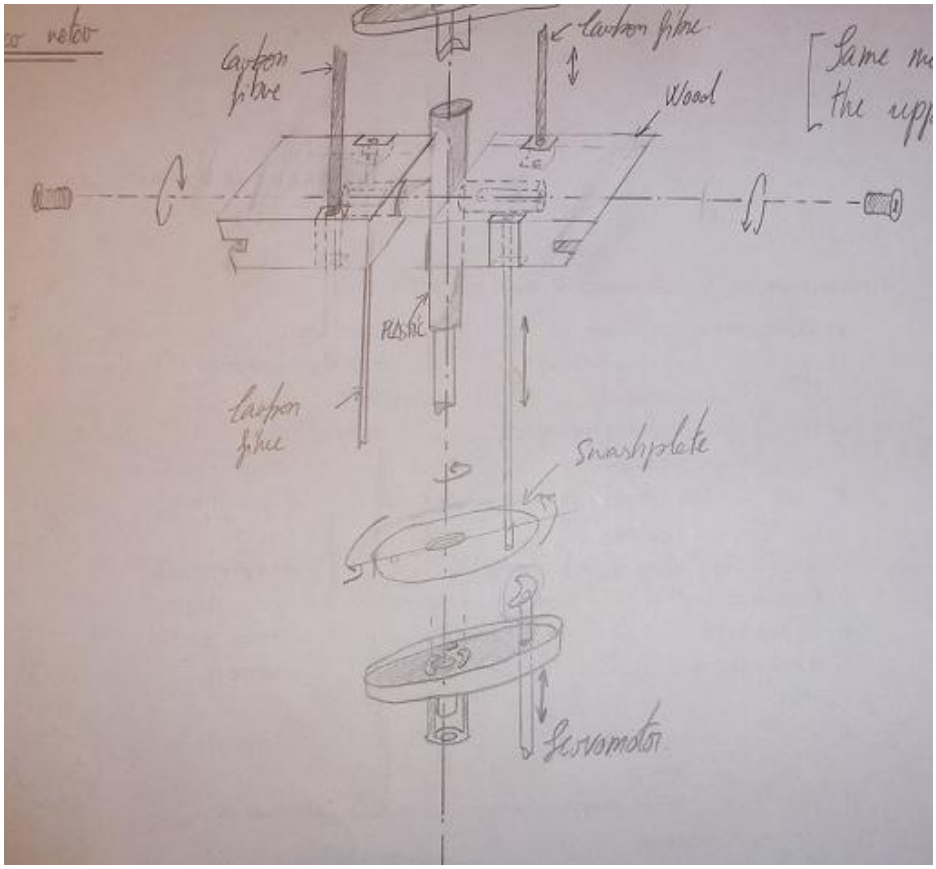


Figure 23 – Coaxial rotor handmade mechanism

That mechanism helped the MAV in the takeoff and landing maneuver reducing the DC motor current. The upper force called Lift (2.1) takes this value in function of other variables, as density, surface and lift coefficient as constant in the same altitude. The only variable which can change at that moment is lift coefficient (C_L). That one will change with if changes the angle of attack, depicted in Figure 24.Thus, at same speed, surface and air density changing the angle of attack we could get same lift at less speed (positive and great angle) or get the same lift with more speed and less angles, but positive too.

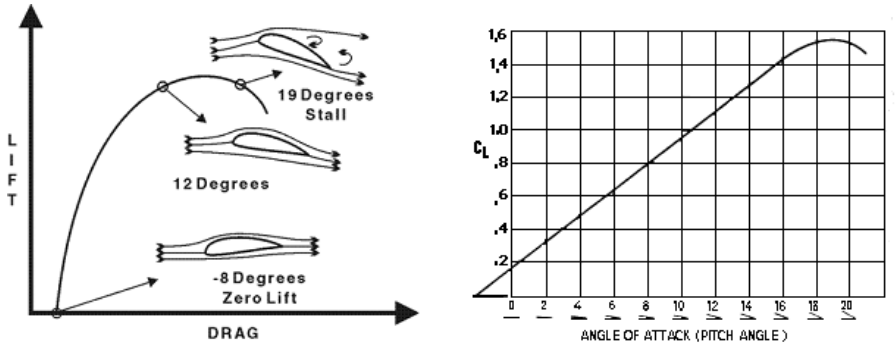


Figure 24 – Relation between Lift, Drag and angle of attack[12]

[12] http://www.insideracingtechnology.com/tech103anglattack.htm

I wanted an optimal flight reducing the consumption, I needed to know how change the angle of attack obtaining same lifts with less power. But, I had to be careful with that, as the device could get rotor stall boundaries [10] due to air disturbance (Figure 25)

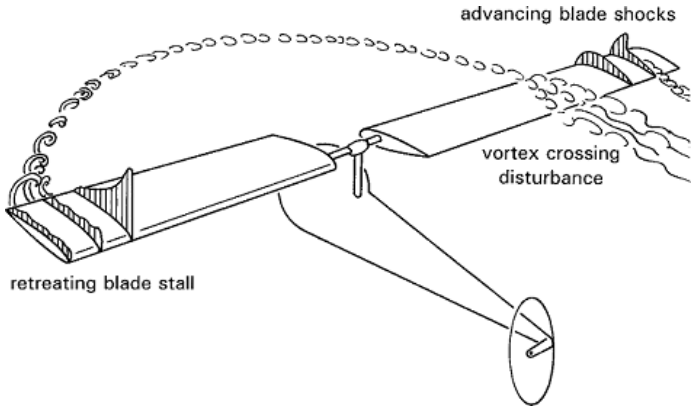


Figure 25 –Rotor stall boundaries

That disturbance is getting when at same speed the blades edge get high speed than other blade parts (uniform circular motion). Those disturbances affect the Thrust as the fluid is not laminar on that moment. Finally, the blades lose lift force as Thrust Coefficient (Lift in circular movement) will be decremented (2.2).

$$T = C_T \rho (\Omega R)^2 \pi R^2 \tag{2.2}$$

Where:

- Air density (ρ) = [$Kg/m3$]
- Rotor speed (Ω) = [$rad/second$]
- Blade radius (R)= [m]
- Thrust coefficient (C_T)

In the following lines I'm going to explain the mechanism parts and their specific function.

Swashplate changes its position due to three servomotors (Figure 26) with linear actuation. The positions are (Figure 27):

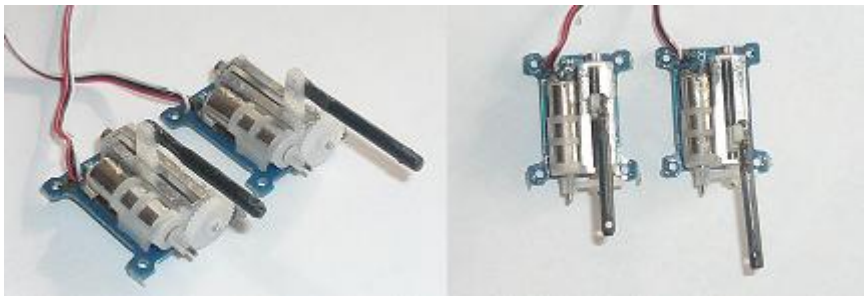


Figure 26 – Servomotor linear actuation which change swashplate position

- All up: Maximum angle of attack (swashplate takes top position).
- All down: Minimum angle of attack (swashplate takes bottom position).
- All neutral: A middle position servo (angle of attack is zero).
- Servo 2 up rest down (negative X axis movement).
- Servo 2 up, Servo 1 and 2 down (X axis movement negative).
- Servo 3 up, rest down (Y axis movement positive).
- Servo 3 and 2 down, Servo 1 up (Y axis movement negative).

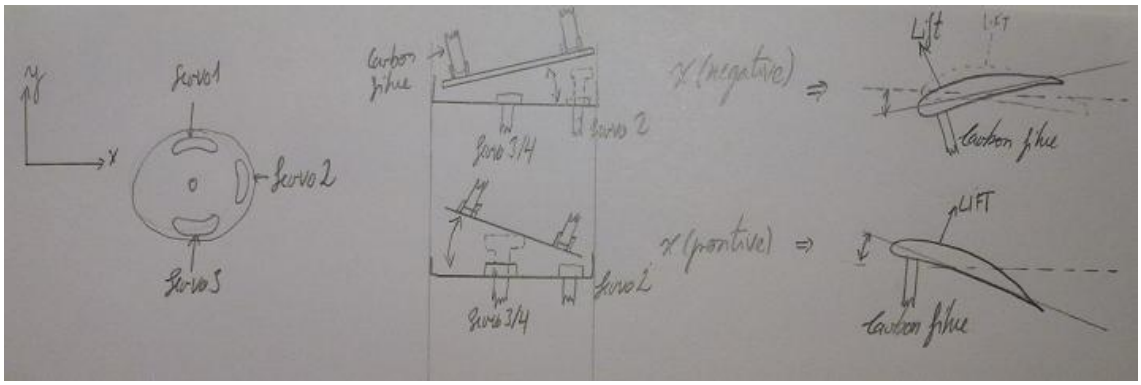


Figure 27 – Coaxial rotor handmade swashplate actuation along X axis

The servos movement change the Lifts/Thrust (angles of attack) surfaces (Figure 27). Consequently the Lift/Thrust vector, which determines the direction of MAV in space change. Thus, the device modifies both position in the space and the consumption.
Finally, I am going to explain the chassis. That structure was made by of wood and brass. Two rectangular wood parts joined by *Superglue* with rectangular pieces of brass (Figure 28).

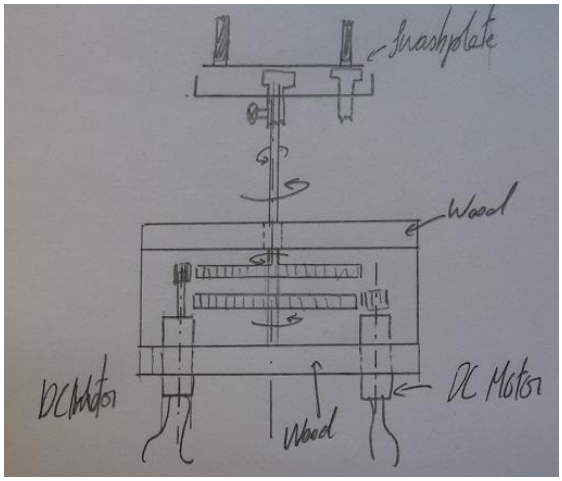


Figure 28 – DC motors into power plant structure

The bottom piece made of wood had two holes to set the DC motors. Those holes had to be set with a great precision, as DC motors gear wheels and coaxial rotors gear wheels had to work correctly (not too pressure, not too loose).

2.2.3 Coaxial rotor implementation

To implement the Coaxial Quadrotor MAV, I needed some specific material as (Figure 29):

- Carbon fiber stick
- Wires
- Plastic
- Brass
- 4x DC motors
- 3x Servomotor
- 4x Gear wheels
- Tools: screwdrivers, hammer, drill, Dremmel drill, *Superglue*…

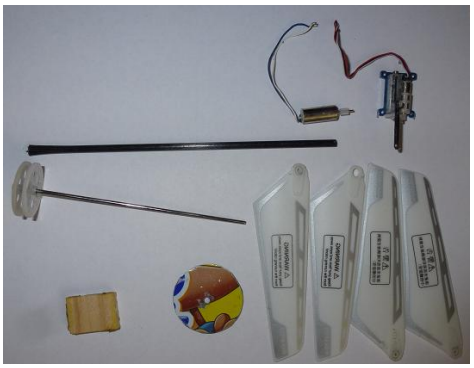


Figure 29 – Coaxial Rotor MAV material

Making Coaxial Rotor MAV I followed next steps:

- In the first step, I cut the brass by a drill *Dremmel* and with that I made the rotating plates, no-rotating plates and some chassis parts. After, I cut carbon fiber sticks doing pitch links and actuator inputs (servomotors) (Figure 22, 23, 26 and 27).

- In the second one, I made wings' affixing with specific shape using a hand saw and wood. I used both a *Dremmel* and a drill (last one to make the holes), see Figure 23 and 30.

- In the third one, I made the shape getting plastic crosses (Figure 23 and 30); setting them into the second one step wood parts. These ones belong to the mechanism which changes the angle of attack, and needed a careful process as it has two parts (Figure 30).

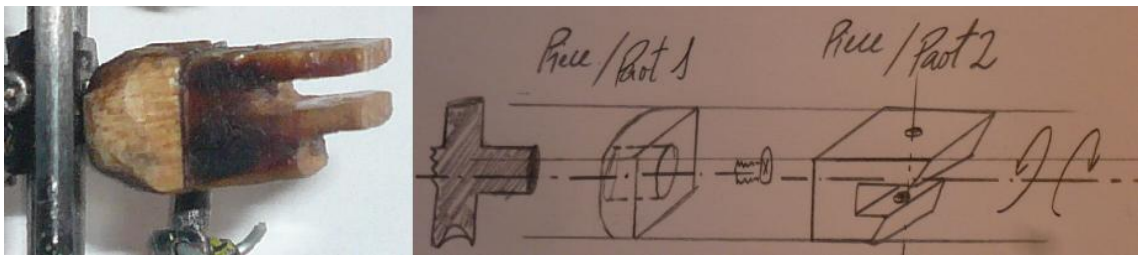


Figure 30 – Mechanism that change angle of attack

The part 1 is joined to plastic crosses by a screw, and they can rotate over the screws' axis.

The part 2 has a specific shape affixing the wings and reducing the vibrations. I joined these parts by glue.

- On the last step, I joined the power plant structure and mechanism, Figure 23, by strength glue, screws and wire. The *Coaxial Rotor MAV prototype* is depicted in Figure 31.

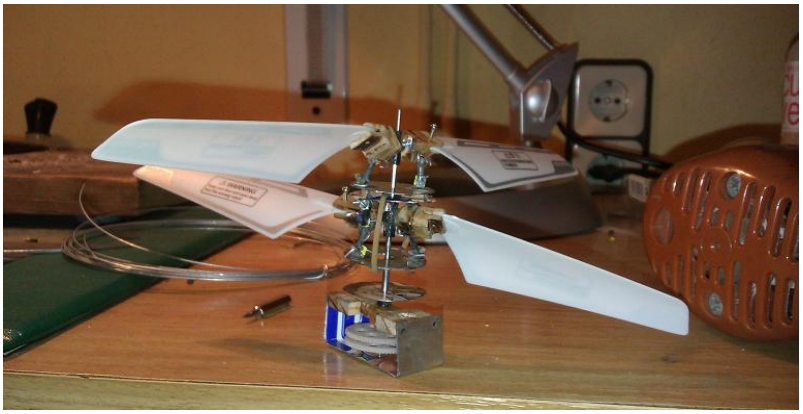


Figure 31 – Coaxial Rotor prototype MAV

2.2.4 Coaxial rotor results

- The mechanism worked, but it was inefficient due to the friction.
- The friction took place between rotating and no-rotating plate at the moment I wanted to change angle of attack.
- We lost a lot of power in vibration and heat.
- The servomotors didn't work well too. They had not enough strength to push to their top position.

A good solution would have been set a bearing into the swashplates, improving the friction coefficient in the rotating parts or making a perfect size parts in order to reduce system friction.

2.3 Quadrotor design

Quadrotors are devices which take an important role nowadays in our society. We can find a width offer range of these devices, in specific stores, commercial surfaces and on the internet (robot trainers).
Quadrotors take a lot of possibilities to explode, enterprises as GoPro and others develops their own quadrotor (Figure 32).

Figure 32 – GoPro quadrotor

The power plant of this kind of MAVs has 4 DC motors. Each one DC motor has on the top a fan. These sets formed by DC motor and fan are put over a strong structure made of plastic or any other material forming a cross shape (90 degrees between branches structure), see Figure 42. That configuration let the owner the MAV control with great precision. Some models can bear a high load too, as their circuits are very tiny and lights. These devices are stable and have a great agility too.
One of the most famous quadrotor MAV is the device called HUBSAN X4 (Figure 33). The configuration is the same I told (90º degrees between branches), but the size is smaller than GoPro quadrotor.

Figure 33 – GoPro[13] and Hubsan x4's size[14]

2.3.1 Quadrotor development objectives

The Quadrotor Micro Air Vehicle had the following characteristics:

- Good control and stability.
- High consumption.
- Compactness.
- Light
- Small
- Easy

[13] http://martybugs.net/gallery/image.cgi?img=GoProHero2-hand
[14] http://www.swiftrc.co.uk/hubsan-x4-quadcopter-4448-p.asp

2.3.2 *Quadrotor design*

Coaxial rotor was made by all I could find into ironmonger's, and some special parts in specific stores, as lift surfaces, Dc motors or gear wheels.

I could find some mini-quadrotors on different store surfaces. The *Quadrotor MAV* power plant was simple and easy to made, but it had not the strength of commercial MAV's structure.

The third MAV power plant belongs to the quadrotor family/ configuration. That configuration is the most common UAV and MAV power plant I made.

I only needed 4 DC motors with their own wings and did a cross structure or chassis. The angle between their branches was 90 degrees, see Figure 34.

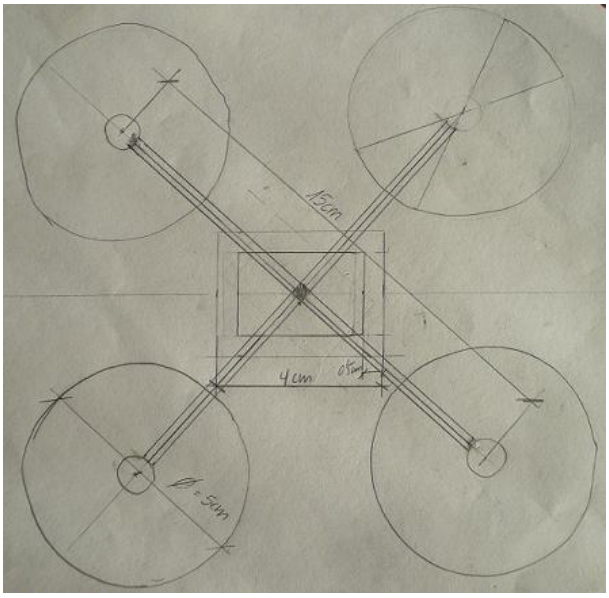


Figure 34 – Quadrotor MAV sketch

Wings are always fixed on the top of DC motors. The current go through DC motors determine the speed of them, and consequently the fans rotates and they make thrust (see Annex A1).
In the following bullets I explain how I made the *Quadrotor MAV* power plant:

Our first problem was solving the DC motors and the rotational moment. I found information about it [9] and how to solve this problem. The Figure 35 explains how many rotational moments the quadrotor has to.

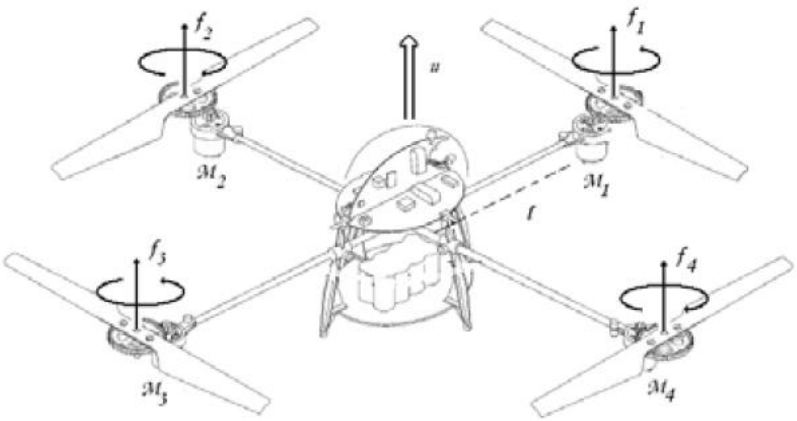


Figure 35 – Quadrotor throttle and rotating direction wings

The 3rd Newton's Law explains how works action and reaction. That means the rotational moment of each motor has to be an opposite rotating direction determines by the set of Equations (2.3).

$$\begin{aligned} f_T &= f_1 + f_2 + f_3 + f_4 \\ f_i &= k_i \omega_i^2 \quad i = 1,...,4 \\ \sum_{i=1}^{4} M_i &= 0 \quad i = 1,...,4 \end{aligned} \tag{2.3}$$

Where:

- f_T is the sum of *thrust forces* of each DC motor-wing set
- k is a constant formed by several variables (air density (ρ), radius(R), lift coefficient (C_T) , pi constant (π)) and rotational acceleration (ω) from thrust equation (2.2).
- M is the sum of moments, it has to be zero in order to get a hover flight.

The next step was needed to know how DC motors-wings worked to get stability. I needed to know how many movements had the MAV. It will get 3 movements (depicted in the Figure 36):

- Rolling (φ)
- Pitching (θ)
- Yawing (ψ)

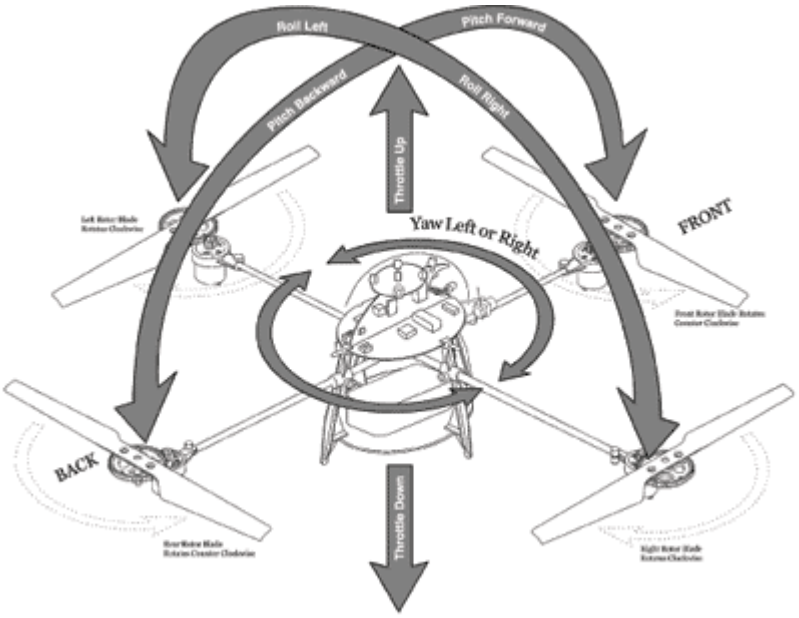


Figure 36 – Quadrotor movements

I could control all this movement through the DC motor speed. In these kinds of devices I could not change the angle of attack. That means, I needed to control the speed of DC motors getting *Pitching* or *Rolling* movement. Furthermore, I had to be careful with that, as if I had had more thrust in order to get either *Pitch* or *Roll* movement; I would have had high *Yaw* (Figure 37)

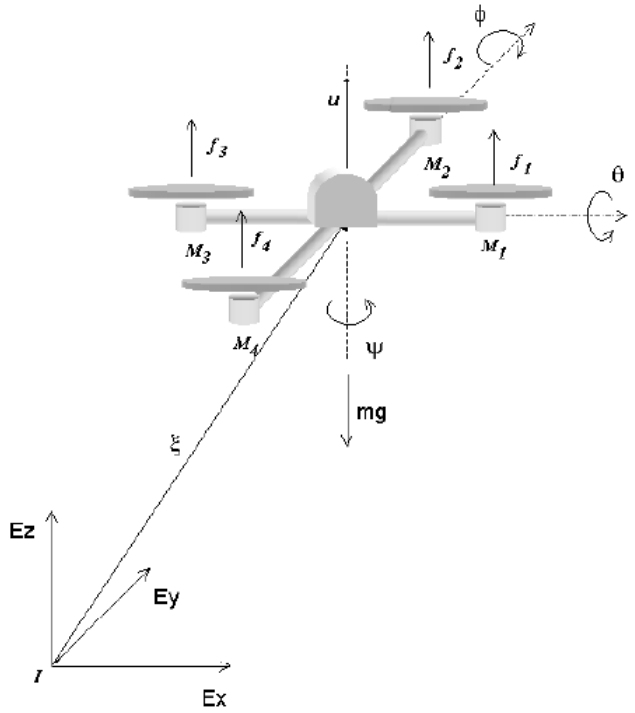


Figure 37 – Quadrotor in an inertial frame

Thus, changing the speed input, I could change the quadrotor position every time I need. Understood its flight too, see Figure 37 and 38.

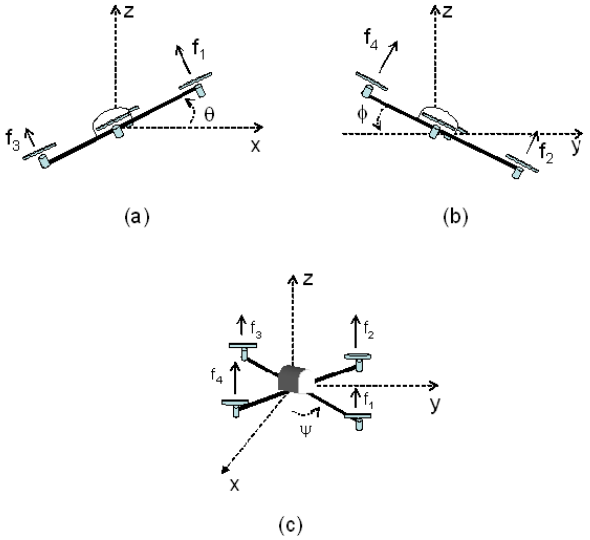


Figure 38 – Pitching, Rolling and Yawing in a quadrotor

The next to last step was drawing a sketch to determine the MAV size (Figure 34). I needed to know the length of each chassis' branch, which determines the MAV static stability. I chose a neutral length (14.5 *cm*).

Finally, I had all I wanted to make an amateur quadrotor. I needed less material than the other MAVs (Figure 39):

- Carbon fiber stick
- Plastic
- 4x DC motor
- 4x Hubsan X4 fans
- Tools: drill, Dremmel drill, knife and glue.

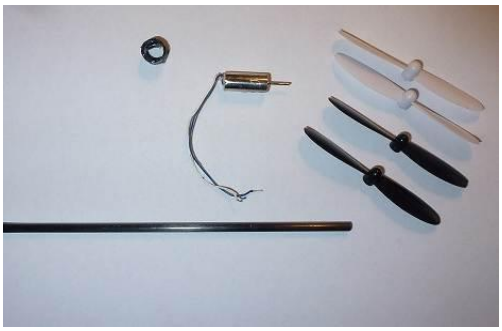


Figure 39 – Quadrotor MAV material

2.3.3 *Quadrotor implementation*

Making Coaxial Rotor MAV I followed next steps:

- First at all, I cut the carbon fiber by Dremmel drill. After, I joined the branches forming a cross and I stack those ones with glue (Figure 40).

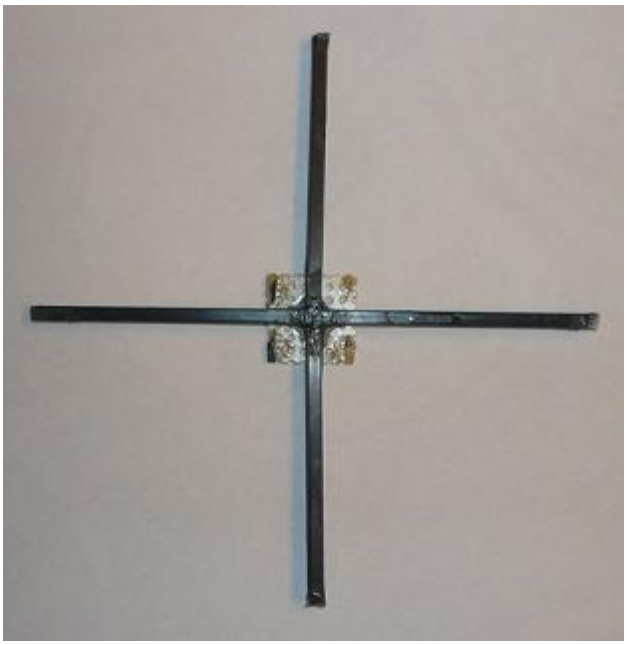


Figure 40 – Quadrotor structure made by carbon fiber

- In the second step, I needed a knife to give the right shape a plastic piece to put inside DC motor. I got the piece from a mini-helicopter structure (Figure 41).

Figure 41 – Piece to set up DC motor

- In the third one, I joined plastics pieces to set DC motors to the cross structure, using *Superglue* (Figure 42).

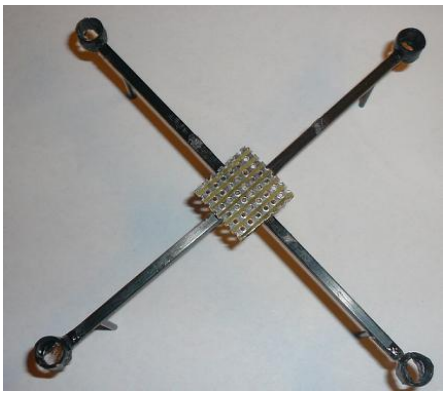


Figure 42 – Quadrotor final structure

- Finally, I cut a cylindrical plastic piece to set into the wings. Because the wing holes are too width for the rotor of DC motor, see Figure 43.

Figure 43 – DC motors feature for setting on DC motors

2.3.4 Quadrotor prototype results

The results of *Quadrotor MAV* power plant (Figure 44) are:

- It worked, but the structure was fragile in the joining parts.
- When it crashed it broke always.
- I needed a lot of time to fix the structure every time it broke.
- During the test configuration I had more problems (see Annex A1).

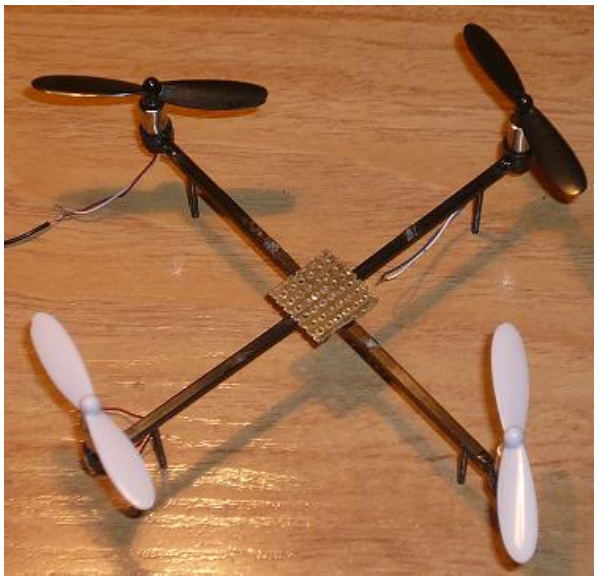


Figure 44 – Handmade *Quadrotor MAV*

CHAPTER 3. EVALUATION OF THE PREVIOUS DESIGN

In this chapter I am going to determine which of last handmade MAVs designs is the best and useful to make it (keeping attention on different fields, as implementation, design, dynamic features…).

3.1 Evaluation of the designs

I tried to make the three specific MAVs; *Dragonfly*, *Coaxial Rotor* and *Quadrotor*. Unfortunately not always they worked on the way I wanted to.

In the first MAV attempt, I tried to build a device based on a dragonfly flight. These kinds of MAVs have a complex aerodynamic and simple mechanism. The MAV worked partially as ornithopter power plant (flapping movement). The device needed another mechanism getting an anisoptera mechanic flight. That means several movements at same time (translation and rotation wings).
That was a challenge; however, the simple mechanism which changes rotational movement to vertical one and the size of the mechanism stopped the *Dragonfly MAV* implementation.

In the second MAV attempt, I tried to build a *Coaxial Rotor MAV.* I understood how the coaxial rotors work properly, and I designed a system relatively small to get this idea (17 cm wingspan).
I employed the same materials as *Dragonfly MAV,* but this time I could get to make the prototype. Mechanically it worked; however, a problem arose when the DC motors started to work (friction in both swashplates). That means the MAV lost power in the rotating movement. That decrement of power was big enough to decrease the speed of wings. Consequently, it lost thrust.

In the last MAV attempt, I tried to make a *Quadrotor MAV*. I had the simplest structure, and it worked well enough. Understanding MAV's reactions in space (dynamics) was complex. Once we got that, everything went well.

In the following bullets you will read pros and cons about making *Dragonfly MAV*, *Coaxial Rotor MAV* and *Quadrotor MAV*.

The *Dragonfly MAV* needs to work:

- Specific components, materials and tools.
- Simulations and dynamic processes.
- Complex mechanics (tiny parts).
- Handmade does not mean precision.

The *Coaxial Quadrotor MAV* needs to work:

- Understanding how the power plant works.
- Small pieces into the mechanism and accurate service.

- Specific components, materials and tools.
- Has a mechanic flight more understandable than *Dragonfly MAV*.

The *Quadrotor MAV* needs to work.

- A structure was easy to build
- Does not need too much time to make it.
- No parts in movement than the rotary wings.
- Be careful with carbon fibers and isolate them (could carry the electricity).

3.2 Summary of the designs

On the following table I am going to show all you the mainly features of the three MAVs I designed and made. The table may answer questions about the difficulty to make one of these devices.

Table 3 – Summary of the proposed handmade MAV designs

	Dragonfly	**Coaxial rotor**	**Quadrotor**
Advantages	-Tiny -Very low power -Possibilities Silent	Aerodynamic Low Power Consumption	Structure Aerodynamic Stability
Drawbacks	Making up Specific tools Complex mechanism	Making up Complex mechanism Friction	Noisy High power consumption
Typical size	10 cm	17 cm	18 cm
Mechanism	Complex	Complex	Simple
Handmade skills	High	High	Low
Flight mechanics	Complex	Complex	Easy
Service	High	High	Low
Power DC motor	Low	Low	Middle
Wing's movement	Flapping	Circular	Circular
Professional tools	Yes	Yes	No

3.3 Airframe selection

After last attempts trying to make MAVs, getting the *Quadrotor MAV* and facing problems I had with it (see Annex A1). I understood I could not ever make strength and perfect mechanism, getting a smooth movement as commercial MAVs. The *Quadrotor MAV* was a good solution, but I had problems with its fragility and with the accelerometer I set on it.

Thus, solving these problems I bought a commercial quadrotor MAV called *AMEWI BLASTERX80*. The device has an industrial precision and strength plastic chassis, big DC motor-wings and a battery too. The circuit of the device has different electronic elements. I could buy and try to implement these on the

new Micro Air Vehicle. In addition, it has a quadrotor power plant (the option I chose).

Figure 45 –Quadrotor MAV called *AMEWI BLAXTERX80*

This device helped to understand how it works; the power plant was perfect to work on it. Understanding how it works helped to learn the system and why I had to use some electronic elements instead other ones (see chapter 2, part 2.3 Quadrotor design).
If I tried to do the same process with the *Quadrotor MAV* instead of *AMEWI BLASTERx80*, it would have been a failure, as every time the MAV crashes I would have had to repair again and again (see 2.3.4 Quadrotor prototype results). In addition, the circuit of commercial MAV gave us some ideas to solve different problems that I had to face through the project.

CHAPTER 4. QUADROTOR MAV IMPLEMENTATION

This chapter explains how many different ways I took to understand how works *AMEWI BLASTERx80*. As any other MAV circuit, *BLASTERx80* has a microcontroller, Inertial Measurement Unit, a Communication Chip (communicating the owner to MAV) and passive electronics elements as capacitors, resistors, Crystal clocks…, see Figure 46.

Figure 46 – Circuit Quadrotor MAV

Each circuit element keeps in touch with different fields of technology, and I needed to know each one.

4.1 MSP430 microcontroller and wireless solution

AMEWI BLASTERx80 has a microcontroller called *STM8S005K6* [15], working together with the IMU called *MPU-6050* [16] and with radio communication chip too. I used another microcontroller with similar features, but I bought the same IMU as commercial MAV. That was due to I had a lot of problems with *ADXL330K* accelerometer (see Annex A1), and *MPU-6050* seemed to be a good solution.

The device called *eZ430-RF2500* (Figure 47) developed by Texas Instruments was set by a microcontroller chip and wireless communication chip and aerial. I chose this one instead *STM8S005K6* as it had wireless communication.

The *eZ430-RF2500* [17] is a USB-based MSP430 microcontroller with a wireless development tool. The device provides hardware and software using *MSP430F2274* microcontroller and *CC2500* 2.4GHz wireless transceiver.

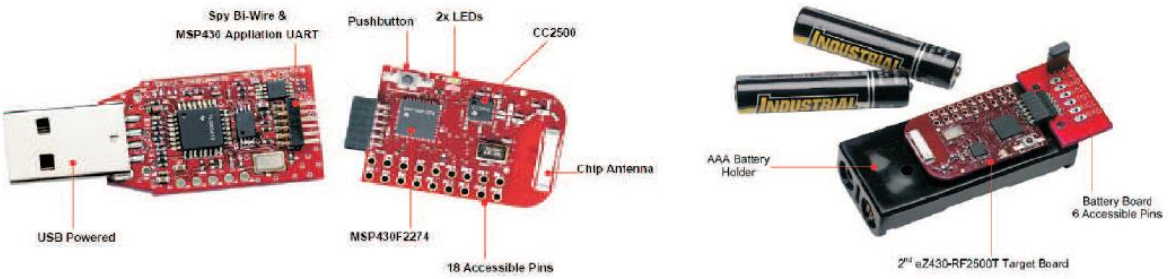


Figure 47 – eZ430-RF2500 Development Kit Components

The *eZ430-RF2500* features (see Annex A4) include:

- USB debugging and programming interface.
- 21 available development pins (see Annex A5).
- Ultra–low-power MSP430 MCU with 16-MHz performance.
- Two general-purpose digital I/O pins connected to green and red LEDs for visual feedback.
- Reset push button for user feedback.
- Expansion board (using the device remotely).

The board has available pins E/S, everyone has different possibilities as General Purpose digital I/O pin, I2C communication, Timer A and B Capture/Compare, Serial Communication, SPI communication, Analog-Digital-Converter…

The microcontroller MSP430F2274 belongs to the branch MSP430 microcontroller family. That means you can employ C/C++ programming language, using registers [18] to configure its applications (see code). Every Timer, Analog-Digital-Converter, I2C communication…need to be set by their own registers (Figure 48)

```
BCSCTL2 = SELM_0 + DIVM_1; //MCLK=DCOCLK, MCLK/2
```

7	6	5	4	3	2	1	0
SELMx		DIVMx		SELS	DIVSx		DCOR(1)(2)
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

SELMx	Bits 7-6	Select MCLK. These bits select the MCLK source.	
	0	00	DCOCLK
	1	01	DCOCLK
	2	10	XT2CLK when XT2 oscillator present on-chip. LFXT1CLK or VLOCLK when XT2 oscillator not present on-chip.
	3	11	LFXT1CLK or VLOCLK
DIVMx	Bits 5-4	Divider for MCLK	
		00	/1
		01	/2
		10	/4
		11	/8
SELS	Bit 3	Select SMCLK. This bit selects the SMCLK source.	
		0	DCOCLK
		1	XT2CLK when XT2 oscillator present. LFXT1CLK or VLOCLK when XT2 oscillator not present
DIVSx	Bits 2-1	Divider for SMCLK	
		00	/1
		01	/2
		10	/4
		11	/8
DCOR	Bit 0	DCO resistor select. Not available in all devices. See the device-specific data sheet.	
		0	Internal resistor
		1	External resistor

Figure 48 – Basic Clock System Control Register 2 (BCSCTL2)

The register has either two or one bytes, not always you will be able to write and read (*rw*) as in Figure 48. There are registers which you only can read (*r*) or write (*w*). Behind the registers there are steps that you need to read and understand how each application works. The application has to follow different steps and works in specific part of memory. I had to set the registers due to these sequences [18].

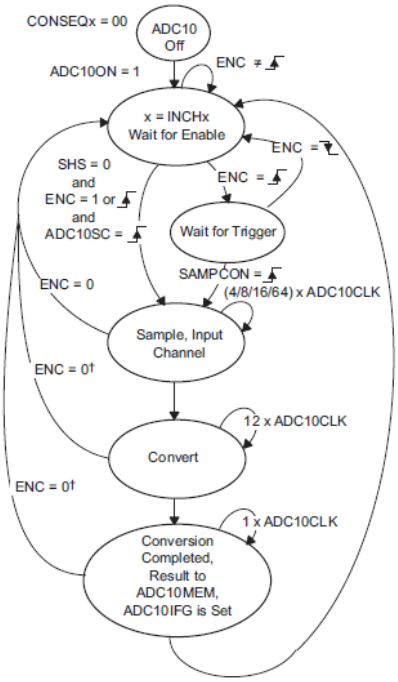


Figure 49 – Analog-Digital-Converter's sequence channel

Each tool or application needs a time execution. The time execution is set by the clocks of the device. Those ones set the speed working of the microcontroller. Exactly it has 3 clocks:

- Masterclock (*MCLK*)
- Secondary Masterclock (*SMCLK*)
- Auxiliary Master Clock (*ACLK*)

The SMCLK and ACLK can set the Masterclock. I have different possibilities to change the processor speed getting the best execution order for our application. The Figure 50 shows how the three clocks affect whole microcontroller system.

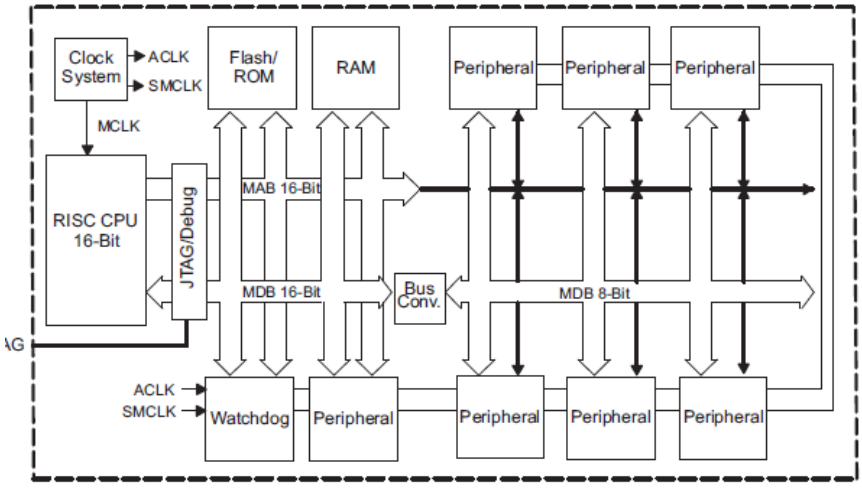


Figure 50 – MSP430 architecture

The Auxiliary Masterclock (ACLK) needs an external Crystal oscillator, which gives to the microcontroller $32\ kHz$ of frequency (you can divide that $32\ kHz$ getting more values). It is another way to get a specific speed which sometimes we need.
This microcontroller has a Very Low Oscillator too (see code configuration). That one gets $12\ kHz$, but you cannot change this value, as you can do with ACLK or SMCLK.

```
BCSCTL3 |= LFXT1S_2;                    // LFXT1 = VLO
```

You can modify in the same program Masterclock speed, configuring the Digitally-Controlled Oscillator and the Basic Clock System. In this project I have not an external Crystal as we have to set SMCLK as MCLK. Setting the SMCLK we need the registers of Digital Clock Oscillator and Basic Clock System Control Register 2, see Table 5.

Table 4 – DCO calibration data

Label	Description	Offset
CALBC1_1MHZ	Value for the BCSCTL1 register for 1 MHz, T_A = 25°C	0x07
CALDCO_1MHZ	Value for the DCOCTL register for 1 MHz, T_A = 25°C	0x06
CALBC1_8MHZ	Value for the BCSCTL1 register for 8 MHz, T_A = 25°C	0x05
CALDCO_8MHZ	Value for the DCOCTL register for 8 MHz, T_A = 25°C	0x04
CALBC1_12MHZ	Value for the BCSCTL1 register for 12 MHz, T_A = 25°C	0x03
CALDCO_12MHZ	Value for the DCOCTL register for 12 MHz, T_A = 25°C	0x02
CALBC1_16MHZ	Value for the BCSCTL1 register for 16 MHz, T_A = 25°C	0x01
CALDCO_16MHZ	Value for the DCOCTL register for 16 MHz, T_A = 25°C	0x00

Those registers can set the SMCLK directly, another way is setting the DCO by the RSEL and DCO registers, but DCO oscillator is not very accuracy getting a specific value. For example, if we set the speed at *1 MHz*, we will get *1.2 MHz*,, for that reason I configured the SMCLK by the following code.

```
BCSCTL1 = CALBC1_1MHZ; // Set DCO
DCOCTL = CALDCO_1MHZ;
```

To understand the clocks sequences, is important to read the block diagram application too Figure 51.

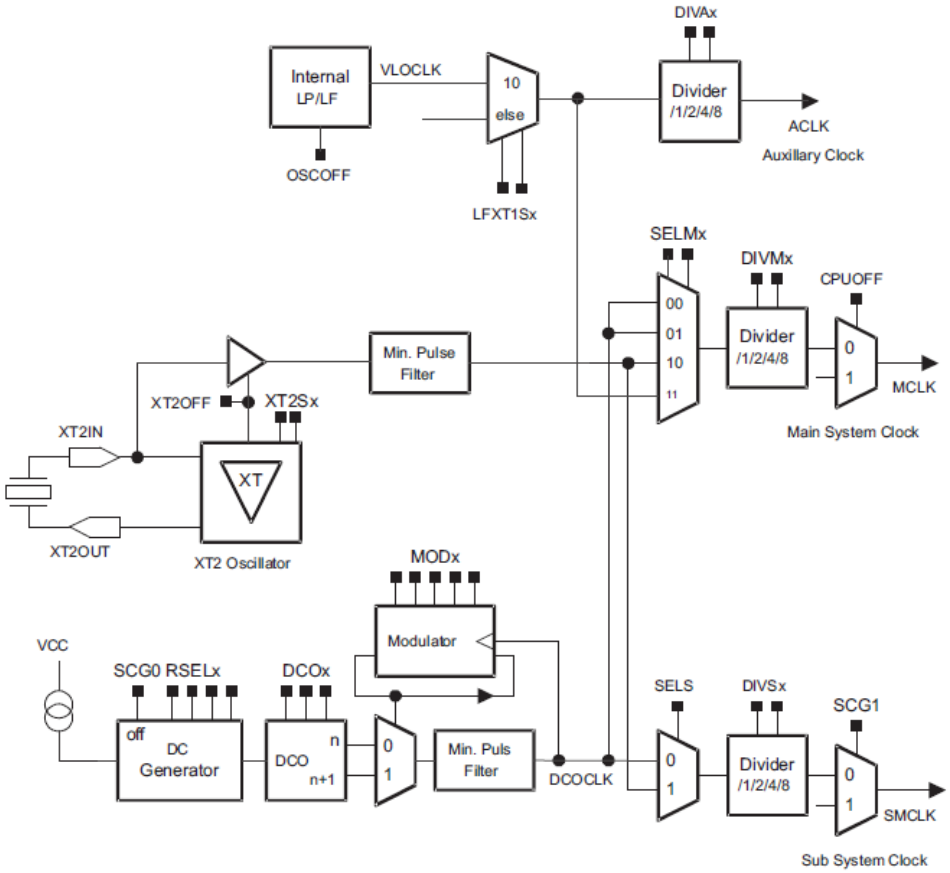


Figure 51 – Basic Clock Module + Block Diagram microcontroller

Into the microcontroller you can program interruptions too. It is useful if you are processing important data, you could stop the system during this specific data

process; when the process is over the device turn back to the last process before the interruption (see interruption code).

```
__bis_SR_register(LPM0_bits + GIE);       // Enter LPM0, interrupts enabled

#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void)
{
}
```

I did not need that application, but it is important you know it. It is useful when the microprocessor is over loading; you only need to set the device in Low Power Mode (Table 5).

Table 5 – Low Power Mode configuration

Mode	CPU and Clocks Status
Active	CPU is active, all enabled clocks are active
LPM0	CPU, MCLK are disabled, SMCLK, ACLK are active
LPM1	CPU, MCLK are disabled. DCO and DC generator are disabled if the DCO is not used for SMCLK. ACLK is active.
LPM2	CPU, MCLK, SMCLK, DCO are disabled. DC generator remains enabled. ACLK is active.
LPM3	CPU, MCLK, SMCLK, DCO are disabled. DC generator disabled. ACLK is active.
LPM4	CPU and all clocks disabled

Another important feature of *eZ430-RF2500* is the Universal Asynchronous Receiver-Transmitter (UART) communication via USB. I configured the communication RS-232 in order to send and receive data from other devices. RS-232 was set with the following characteristics (Figure 52).

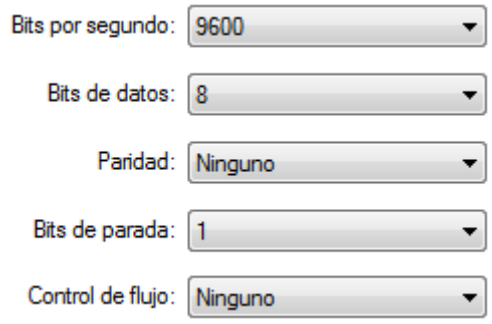


Figure 52 – Values for setting up Serial communication

In the following Figure 53 is depicted the pins of USB Debugging Interface, where TX (transmit) and RX (receive) pins carry Serial data to the computer in this case.

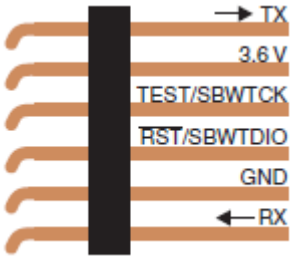


Figure 53 – eZ430-RF2500 USB Debugging Interface pins

4.2 MSP430 programming environment

Each microcontroller needs an interface to load the program by the programmer. That one is called compiler. The *eZ430-RF2500* can work with a compiler called *Code Composer Studio* (Figure 54) developed by *Texas Instruments* (in a CD with *eZ430-RF2500* device set).

Figure 54 – Code Composer Studio

The program has a C++ programming environment; I needed time to adapt to it as the getting started information did not help to understand the microcontroller. The C/C++ programming has bit differences with other C/C++ compilers, which sometimes did not compile instructions that compilers can. The internal device memory restricts to do some operations.

The Code Composer Studio has two different compiler windows:

- The first one is the programming window, which compiles code errors too
- The second one is the loading window, the program uploads to *eZ430-RF2500* device, and the programmer can modify the code too.

At the moment the new project is created; the code is written and it has no errors or warnings; the programmer can debug the code and uploading to the device with the next button (Figure 55).

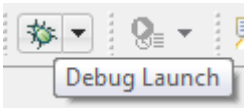


Figure 55 – Debug Launch button

That button, get the programmer from C/C++ window to Debug window. Last one has sub-windows inside of it (Figure 56):

- Code window: Where C/C++ code is (green line window).
- Variables window: You can see how change the variables of the program (right hand window).
- Error and Console window: Any code error arises on this window (bottom window).
- Disassembly window: Show you how the code works in assembler language (Middle window).

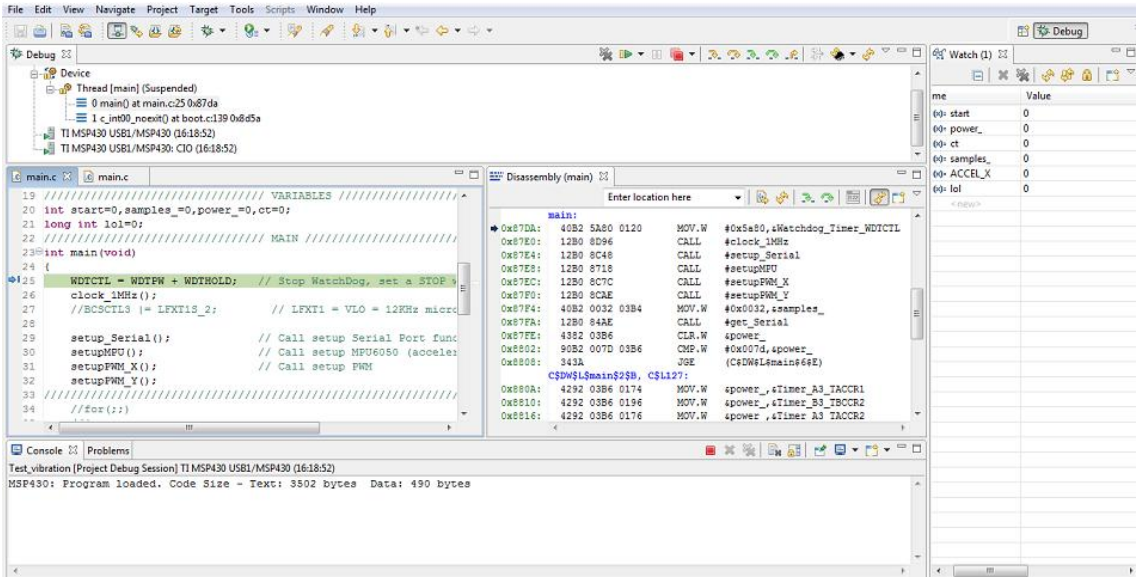


Figure 56 – Code Composer Studio Debug window

To execute the program, the programmer only needs to push the green start button. Once the button is pushed, the programmer will be in any other C/C++ compiler. Step into, step over, assembly step into, assembly step over… can be found in the main tools bar (Figure 57).

Figure 57 – Code Composer Studio Debug tool bar

To enjoy all *eZ430-RF2500* applications, the programmer has to download from Texas Instruments webpage the *code examples* of your device model, in this case *MSP430F22x2 EXAMPLES*. Into this file there are code examples about Analog-Digital-Converter, Pulse Width Modulation, Serial communication…

To understand how it works the programmer needs to paste the program on **.c file*, load in the pre-compiler the device's library (see the following code), and read the user's guide registers and schemes. Finally, the program will be compiling.

```
#include <msp430x22x4.h>
```

4.3 Pulse Width Modulation

The outputs of *AMEWI BLASTERx80's* are 4 DC motors, but I needed to control them. I could control the speed of DC motors by an input signal called Pulse Width (Figure 58).

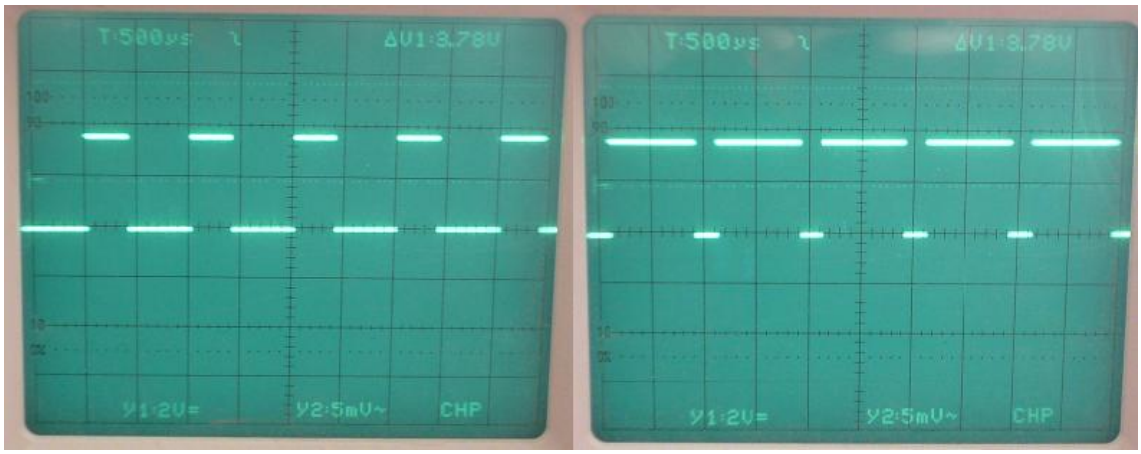


Figure 58 – PWM 1 kHz frequency

PWM is a common solution to control the DC motor's speed. It is a square wave where we can change the current pass through the motor expanding the PWM duty cycle (width). This change is not synonym of frequency modification; the PWM frequency is always constant.

DC motors can work with different PWM frequencies. Our DC motors are brushless, which doesn't need brushes to change the rotor's polarity and rotate. These kinds of DC motors are:

- Cheap
- Light
- Less complex due to electronic control
- Low service.

I chose a 40 kHz frequency in order to avoid the noise for us (*15* to *20 kHz* human being earing frequency); the MOSFET-N called *Si2300DS* could work with that frequency and the DC motors worked correctly too.
Getting this frequency I looked for in the C/C++ MPS430F22X2 EXAMPLES and find the PWM using the Timers of microcontroller, see next code.

```
TACCTL1 = OUTMOD_7;                          // TACCR1 reset/set
TACCR1 = 100;                               // TACCR1 PWM duty cycle
TACTL = TASSEL_2 + ID_3 + MC_1;// SMCLK, SMCLK/8, up mode counter
```

Timer is a microcontroller tool, a counter. That one starts to count from 0 to 65535 (as the microcontroller works with 16-Bit Timer_A and Timer_B [19]) and it has modes which will change the counter shape. You can see in the Figure 59 the mode Continuous and Up/Down.

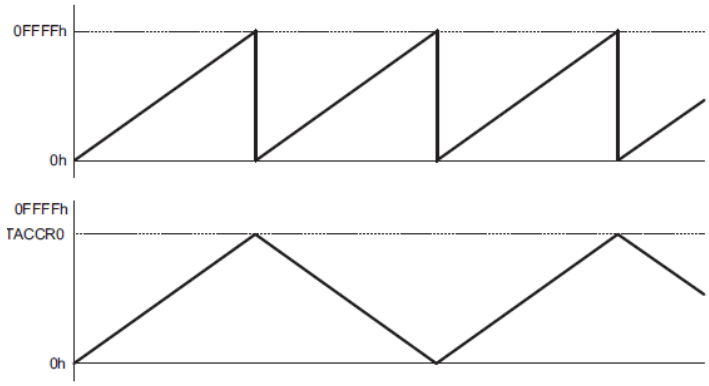


Figure 59 – Timer Mode continuous and Up/Down

That shapes help us to determine the outmode getting our PWM (Figure 60). The counter can be programmed as Capture or Compare. The first one records the counter value (speed communications or time measurements) and the second one is normally used to generate PWM output signals or interruptions at specific time intervals (Figure 60).

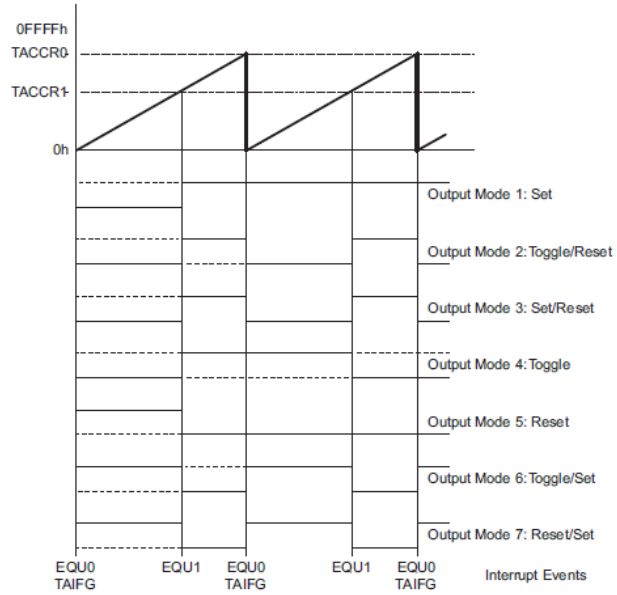


Figure 60 – Outputs of Timer in Up Mode

Once the programmer gets the mode and output mode, he has to set the MCLK. That set is explaining in the section 4.1, see next code.

```
BCSCTL1 = CALBC1_8MHZ;      //Set SMCLK
DCOCTL = CALDCO_8MHZ;
TACTL = TASSEL_2 + ID_0 + MC_1;   // SMCLK, up mode
TACCR0 = 200;                    // PWM Period = 1 kHz
TACCTL1 = OUTMOD_7;              // TACCR1 reset/set
TACCR1 = 100;                    // TACCR1 PWM duty cycle
```

Once the programmer gets *40 KHz*, the Pulse Width Modulation will have to go through to a pin output. That pin will be connected to the gate of the transistor. It is an important element because allows DC motors work with the pulse characteristics (explained in the section 4.4).

To control the width of Pulse Width Modulation, there are two ways:

- Through Analogic-Digital-Converter
- Via RS-232 communication with *Labview*

I got both, the most useful is via *Labview*, and I learnt how to start, send and end data with the RS-232 communication. *Labview* has a lot of tools that you will ever get with a simply ADC.
Sending data from the microcontroller to a program as Labview and vice versa you need the UART tool (specific *eZ430-RF2500*). Before, the communication with *Labview* took place; I had checked if the communication works sending a string and finding the message with a program called HyperTerminal (Figure 61), see the following code.

```
UCA0TXBUF = data ;                    // Sending data
data = UCA0RXBUF;                    //Receiving data
```

This program showed if I was either sending or receiving information through *Serial Port*. *HyperTerminal* only can read strings, then when I sent numbers random symbols arose on the display of *HyperTerminal*.

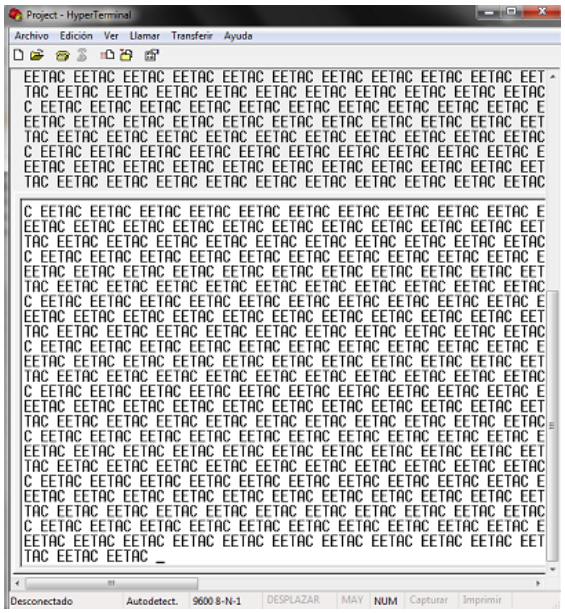


Figure 61 –HyperTerminal display reading a character string

The *MSP430F2274* sent data, but I needed to check if the information was arriving in the correct order. The RS-232 communication only can read/send bytes (Section 4.5). Then, I made a communication send/ receive data protocol (Figure 62).

Figure 62 –Byte composition

Getting that point and knowing that I had to send 16 bits belong to the Analog-Digital Converter from Inertial Measurement Unit [16]; I needed to break those 16 bits into 4 bytes (see Annex A2). The data value of these bytes is positive or negative, that means they are going to take values from -32768 to 32767.

The sending and receiving bytes are composed by:

- Bit check: Useful in MATLAB's communication.
- +/- Number: If this bit value is 1 data belongs to negative numbers, if it is 0, the data belongs to positive numbers.
- Send/Receive Order: These two bits determine the sending and receiving data.
- Data: Essentially data value.

When I got the last protocol, I needed to do a program which controls the input DC motors (PWM). Interface was absolutely necessary, the selected program was *Labview* version 2013 (Figure 63.).

Figure 63 –Labview logotype

Laboratory Virtual Instrument Engineering Workbench is a design platform for a visual programming language developed by National Instruments. It is used for data acquisition, industrial automation and instrument control.

Labview sent to *eZ430-RF2500* these four bytes data information. These ones carried the width of Pulse Width Modulation. The width of PWM control was through a vertical progress bar (Figure 64). I programmed in the C/C++ a code line which changes the 1024 sent to the 125 values that PWM's width had to (read next code).

```
power_1 = getLabview();
power_1 =((unsigned long int) 125* (unsigned long int)power_1) /(unsigned long
int) 1024 ;
```

The 125 value is as a result of the timer calculus to get $40\ kHz$ frequency to control the DC motors.

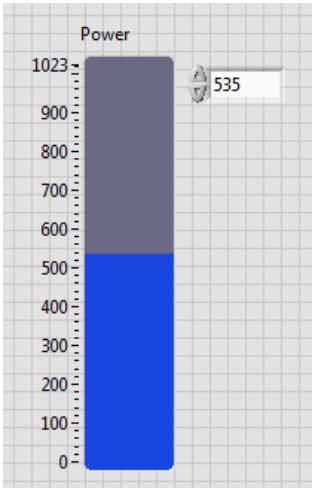


Figure 64 –Labview control PWM's width

4.4 Power plant schematics

It is important to choose each electronic element rightly, which helps PWM arrive to the DC motor and work correctly. Getting that we need elements as transistors, capacitors, resistors…
I designed our schematics using a program called Proteus (Figure 65).This program can design and simulate any electronic circuit you want. In addition, it has a width library with a bunch of electronics elements as Serial ports, resistors, capacitors, amplifiers, microcontrollers…

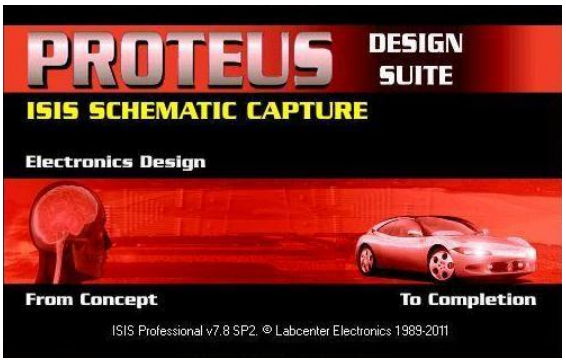


Figure 65 – Proteus schematics

Controlling the current through DC motors is very important. The element I needed was a transistor. The microcontroller cannot provide more than a few mili-Ampers and the DC motors need a couple of *Amperes* to work properly. I needed a specific transistor called MOSFETs. There are two different MOSFETs[15], MOSFET-N and MOSFET-P (Figure 66). I chose to use MOSFETs-N (does not need negative voltage). These elements are different from BJT transistors, because it works with voltage not with current. They allow

[15] http://blog.oscarliang.net/how-to-use-mosfet-beginner-tutorial/

working with high current between Drain-Source and control it with a signal in the through the *Gate* (low current).

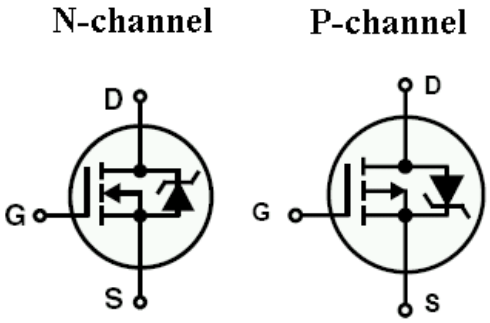


Figure 66 – MOSFET-N and MOSFET-P transistor

The MOSFET-N called *Si2300DS* (Figure 67) is the transistor I set and has a tiny size and support the following all technical requirements too:

Figure 67 – MOSFET-N transistor Si2300DS

- Allows high current with low gate voltage ($3\ V$), see Figure 68. The graph shows the MOSFET-N model Si2300DS has $V_{Drain-Source} = 3.7V$ and $V_{Gate-Source} = 3.65V$, it can allow more than $15\ A$. We only need $0.8\ A$ as maximum for each DC motor.

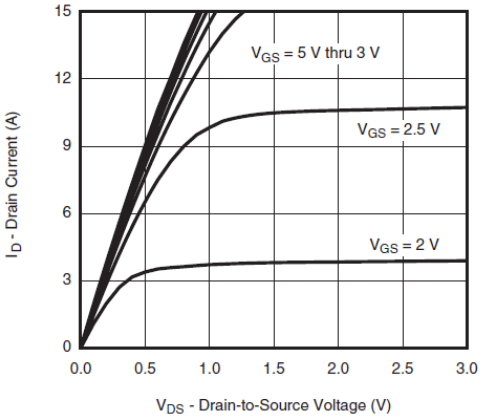


Figure 68 – Drain current vs. Drain to Source Voltage

- Low internal on-Resistor value, see Figure 69: The internal resistor could reduce our tension and burn the element. If our $V_{Gate-Source} = 3.65V$ and the

drain $I_{Drain} = 0.8V$ (maximum ampere in each DC motor); that means the internal resistor might be equal to 0.05Ω.

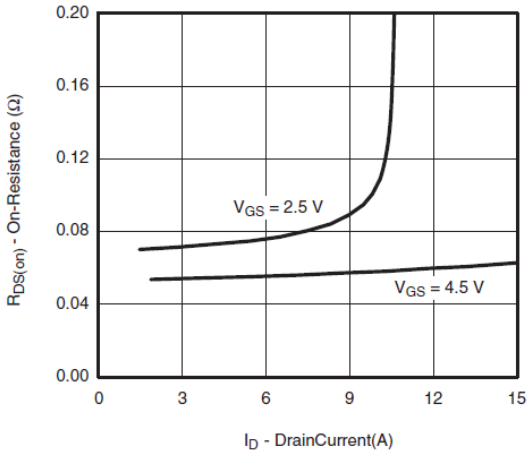


Figure 69 – On-Resistance vs. Drain Current

The transistor with *0.8* top current should not be on fire in any case. I knew what power dissipation was using the following equation (4.1):

$$Tj = T_{case}(\theta_{juntion-case} Power) \tag{4.1}$$
$$Tj = 25^{o}C + (60^{o}C/W * 1.7W) = 127^{o}C < 150^{o}C$$

That Watts will be dissipating in heat and I knew *Si2300DS* top heat [20]. The transistor has a constant that determines how many Celsius degrees heat could get with that amount of Watts (in the junction-case).

$$R_{ThJF} = 60^{o}C/W$$

The junction case will get as maximum *127 ºC*; the datasheet shows us the maximum is *150 ºC*. MOSFET kept its integrity.

Checking the junction-case temperature is the most important in SMD elements. That determines if the electronic element could be either damaged or burned.

- The first graph in the Figure 70 shows if we have $V_{Drain-Source} = 3.7V$ and if the commutation frequency (PWM frequency) is *0.25 µs* (*40 kHz*), we are on the safe area. The junction case can bear the PWM signal with no damages.
  The second graph shows if $I_{Drain} = 0.8V$, the transistor could get the maximum temperature junction-case; but in our system the maximum MOSFET temperature was *127 ºC*.

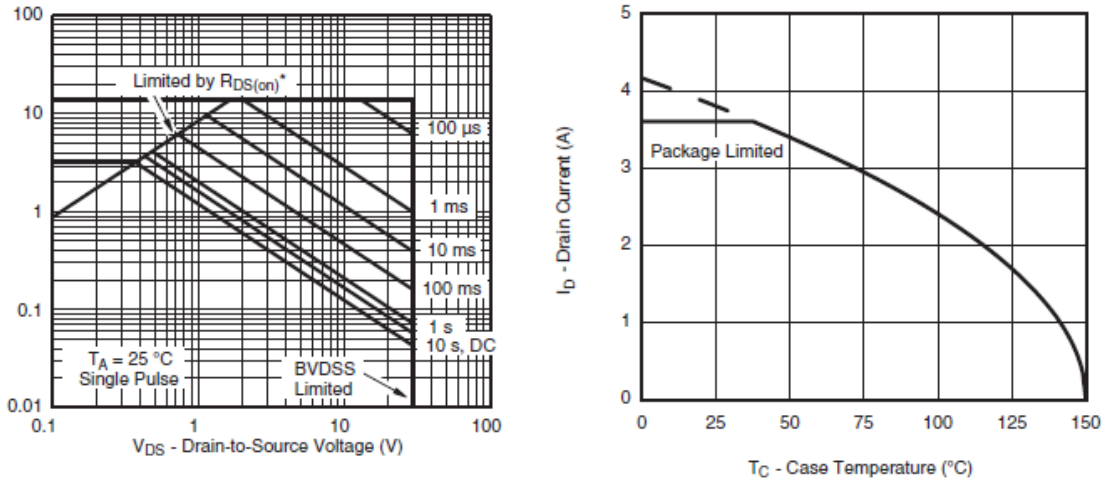


Figure 70 – Junction-to-Case safe area graph and power dissipation temperature graph.

Si2300DS had got all what I wanted for designing the power circuit. I had to be careful doing the circuit design as the scheme has to protect microcontroller pins. The scheme of one DC motor MAV is shown in the following Figure 71.

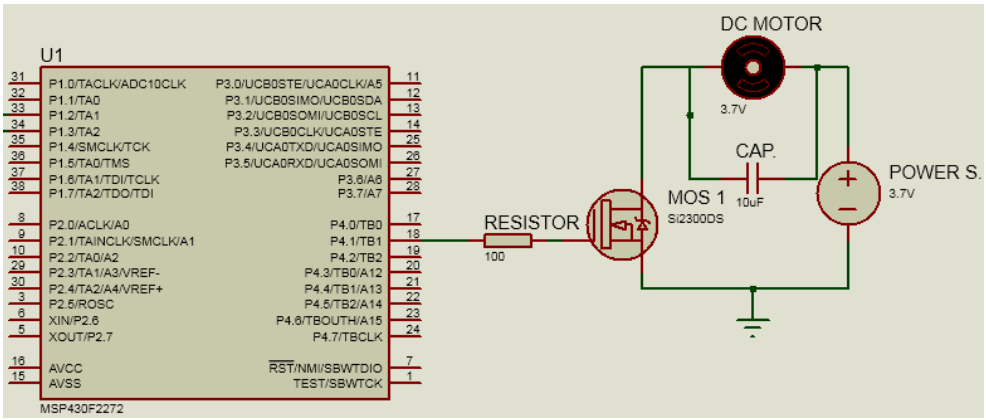


Figure 71 – Scheme of electronic power circuit

We set a resistor in the gate connection for protect it. This resistor protects the microcontroller's pin; it value is 100Ω. That one limits the current through the gate and avoids breaks on it.

The scheme has not protecting by a diode, as after a lot of attempts trying to solve problems between the power circuit and IMU's I2C communication, I decided not to use it as the diode was not the solution with the problem I had (section 5.1). *BLAXTERx80* circuit (Figure 72) had not this element.

Figure 72 – Power circuit DC motors 1 and 2

4.5 UART, RS232, USB, I2C data buses

I set different kind of standard communications in the present project. These took an important role as had to understand how the communications work. I used the following communications.

- I2C: Communication between Inertial Measurement Unit and eZ430-RF2500 microcontroller.
- Serial communication RS-232: Communication between the interface program (*Labview*) and *eZ430-RF2500* device.
- UART: Communication that allowed set Serial Port communication without need hardware connection.

These buses have their own communication protocol and I needed to understand their features.

Inter-Integrated Circuit called I2C is a multi-master, multi-slave, singled ended. This serial computer bus invented by Phillips Semiconductor, and is used in low-speed peripherals.
I2C has two bidirectional lines; these ones are Serial Data Line (SDA) and Serial Clock Line (SCL), see Figure 73. Both need pulled up resistors to work properly.

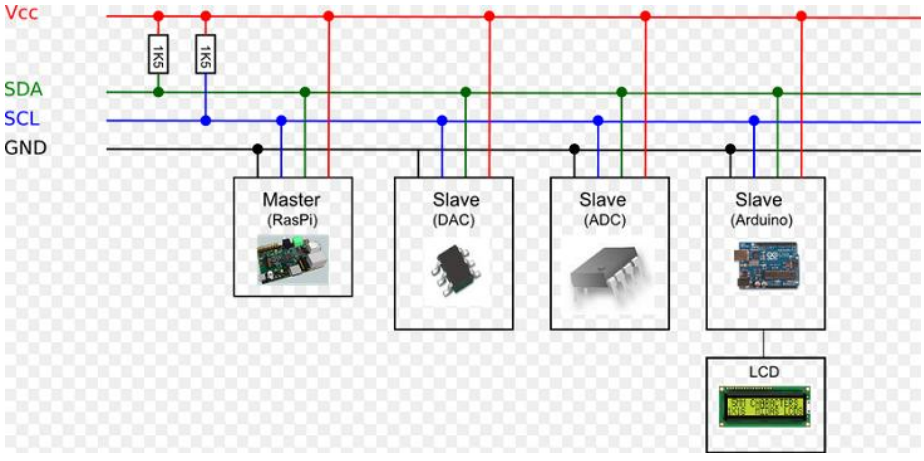


Figure 73 – I2C communication structure.

The devices connect to this bus can be either slaves or masters. Master always generates the clock and initiates the communication to slaves. The slaves receive the clock signal and responds when the master sends them a register address. Slaves can become master when a stop signal is sent to. I2C bus communication is binary, which works with two voltages low and high (voltage depends of microcontroller $3.65V$ to $0V$ (*MSP430F2274*)).
The communication always begins with a change High-to-Low in SDA signal (Figure 74). Then, the clock works. Stopping communication we need a change Low-to-High in SDA signal, consequently clock signal stops.

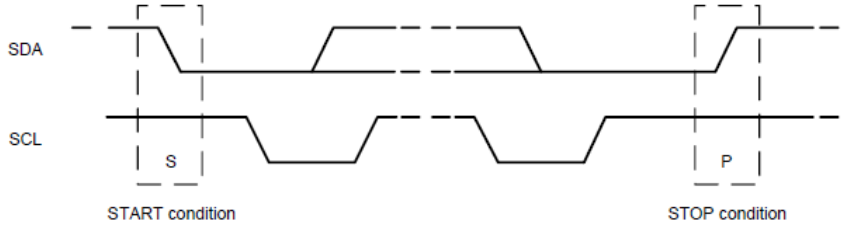


Figure 74 – I2C STAR and STOP condition communication.

If the communication is setting the master sends a 7-bit slave address followed by an 8th bit (the read/write bit). This last bit determines if master is reading or sending address to slave. Then, the master releases the SDA line and waits for the clock device signal; each byte transferred is checked by a signal clock. Sending and receiving bytes has a specific order depicted in the Figure 75 and Table 6:

Single-Byte Read Sequence

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Single-Byte Write Sequence

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Figure 75 – I2C read/write communication protocol.

Table 6 – I2C protocol communication

Signal	Description
S	Start Condition: SDA goes from high to low while SCL is high
AD	Slave I^2C address
W	Write bit (0)
R	Read bit (1)
ACK	Acknowledge: SDA line is low while the SCL line is high at the 9^{th} clock cycle
NACK	Not-Acknowledge: SDA line stays high at the 9^{th} clock cycle
RA	MPU-60X0 internal register address
DATA	Transmit or received data
P	Stop condition: SDA going from low to high while SCL is high

There are a Burst Read/Write sequences for a continuous dropping data [21] too.
I2C registers has 8 bits, and I had to set some of them to access reading their data (Figure 76 and Table 7) as I explained in section 4.1.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-		

Figure 76 – I2C registers MPU6050.

Table 7 – MPU-6050's accelerometer scale range configuration

AFS_SEL	Full Scale Range
0	± 2*g*
1	± 4*g*
2	± 8*g*
3	± 16*g*

The last communication was Universal Asynchronous Receiver Transmitter. UART controls serial devices and serial ports. That one is an interface shaping data (RS-232, RS-422 and RS-485) to output devices. Normally UART is set up at computer mother base and can send data either parallel or serial shape communication.
In this project I used UART when I wanted to send data via RS-232 standard from *eZ430-RF2500* to my Personal Computer (PC). The microcontroller device was set with a chip UART communication called *MSP-FET430UIF*. It is set into the USB connection of *eZ430-RF2500*, see Figure 77.

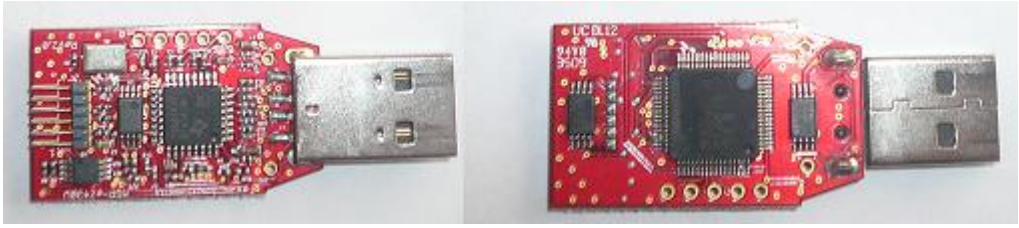


Figure 77 – MSP-FET430UIF.

Communication RS-232 (Recommended Standard 232) is a Serial communication standard. It is a physical interface where the information is sending/receiving one bit at a time. This kind of communication needs for working a Data Terminal Equipment (DTE) and a Data Communication Equipment (DCE) setting communication through this port. There are two hardware connections (Figure 78):

- RS-232 connection with 25 pins (DB-25).
- RS.232 connection with 9 pins (DB-9)

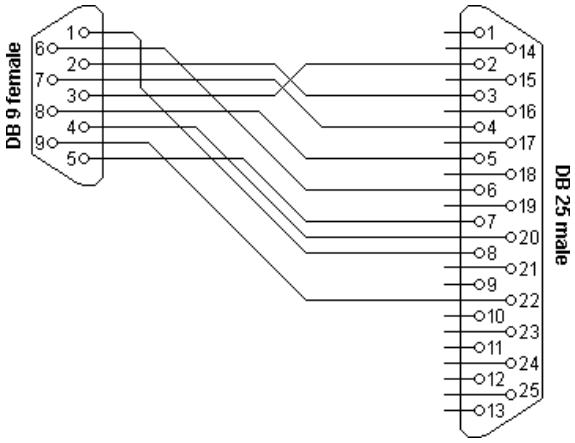


Figure 78 – Pin connection DB-9 and DB-25.

It is not necessary establish connection to a DTE, if the connection is set between them, this communication is called *null modem*.
Interface is designing to send data in short wires ($15\ m$. maximum). It can work either synchronous or asynchronous mode. The synchronous communication needs an external clock to send and receive the data. The second one does not need any clock communication. Thus, the Data Communication Equipment will need a little protocol to receive the information in the order it needs.

The communication will be half duplex. That means, the data will be sending in both directions, sending and receiving (not at the same time).To set RS-232 communication we need to know the values of (Figure 52):

- Speed: It uses two voltage levels High and Low; the data rate will be in bits per second bps ($Bauds$); at high speed it turns inefficient due to the bit framing.
- Data bits: How many bits you will send.

- Parity: Detects errors in transmission. *None* the detection is handled by the communication protocol.
- Stop bit: It is sent at the end of every character, which allows the receiving signal hardware to detect the end of a character and to resynchronize the character stream.
- Flux control: Serial port uses signals in the interface to pause and resume data transmission; in addition, it set parameters of communication (*handshake*).

This communication is especially useful in the communications of the following devices: printers, industrial field buses, GPS receivers, LED text displays, satellite phones…

4.6 Inertial Measurement Unit

Inertial Measurement Unit is an electronic device which takes gravitational forces, velocity and orientation measures. IMU has set accelerometers and gyroscopes sensors. Sometimes, it has set magnetometers too. IMU takes an important role in aerial devices; it takes information about the stability and navigation of the aircraft.

Technology has been changed and nowadays we can find IMUs setting on Microelectromechanical Systems (MEMS). These devices have reduced their size from a bunch of centimeters to several millimeters. That change helps us to set it into a small aircrafts as MAVs.
In this project I used a MEMS called *ITG/MPU-6050* (Figure 79).

Figure 79 – ITG/MPU-6050.

That one has the next features [16]:

- I2C communication.
- Digital-output triple axis accelerometer with a programmable full scale range of ±2g, ±4g, ±8g, ±16g, see Figure 80.
- Digital-output triple axis angular rate sensors (gyroscopes) with a programmable full scale range of ±250g, ±500g, ±1000g, ±2000º/sec, see Figure 80.
- Integrated 16-bit ADCs enable simultaneous sampling of gyros and accelerometers.

- Digitally-programmable low-pass filters (gyroscopes and accelerometer).
- Sensor of temperature
- Minimal cross axis sensitivity between the accelerometer and gyroscopes axes.
- 400 kHz Fast Mode I2C for communicating with all registers.

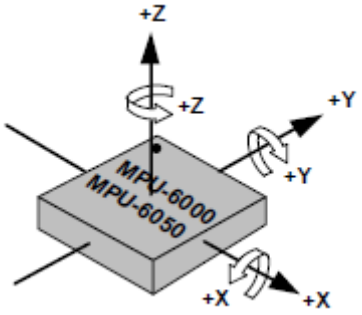


Figure 80 – Orientation of Axes of Sensitivity and Polarity of Rotation.

At the beginning I used an accelerometer called *ADXL330K*, it was set into the *Quadrotor MAV* power plant, and I had a lot of problems, specially filtering. The IMU called *MPU-6050* is the same IMU that *AMEWI BLASTERX80* device. *ITG/MPU6050* gave us the opportunity to understand and program I2C communication and working with an accuracy and strength system.

CHAPTER 5. QUADROTOR MAV FLIGHT TESTS

In this chapter I explain how many problems arise when the *AMEWI* MAV (with MSP430F2274) works. I had two important problems; the first one was the electromagnetic interferences in digital signals (crosstalk) and the second one was the vibration over IMU due to DC motors working. I tried to implement a lot of different solutions.
To check these solutions I needed to set a test system (Figure 81), formed by:

- Oscilloscope (signal check)
- Digital Multimeter (check voltage (*V*) and current (*A*))
- Personal Computer (manage all data information)
- Microcontroller (eZ430-RF2500, see section 4.1)
- Power Source (from old PC)
- Wires (electronic and communication)
- Protoboard

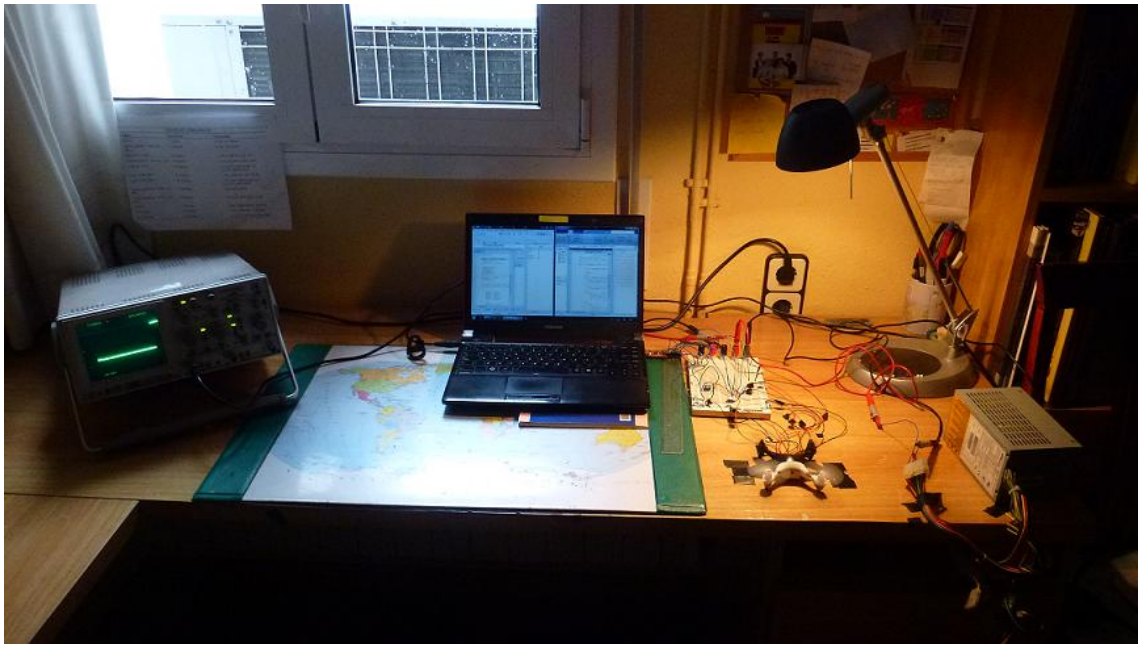


Figure 81 – Amateur test system

5.1 Functional test

Having all specific elements for testing, I only needed to implement a program into the microcontroller in order to fly the device.

In the first step, I had to determine the balance value of IMU. We took 400 samples to determine the exact value of accelerometers stability position on the three space axis via I2C communication, between MEM (slave) and PC (master). I remembered that IMU send 16 bits ADC data shared in two bytes. Getting 400 samples I needed two *For* loops, every loop cycle took the

accelerometer value via I2C communication. The value arrived into two bytes. The first one had the Most Significant Byte (MSB) and the second one had the Least Significant Byte (LSB). Then, I had to join the values (see next code).

```
ACCEL_X_H = ByteRead_I2C_USCI(ACCEL_XOUT_H);    //MSB
ACCEL_X_L = ByteRead_I2C_USCI(ACCEL_XOUT_L);    //LSB
ACCEL_X = (ACCEL_X_H << 8) | ACCEL_X_L;         //MSB + LSB
```

In the second step, I established communication with Labview via RS-232 standard. That program gave the width value of PWM that DC motors needed to work. The data was sent with 1024 values, and was necessary to change it to 125 (top PWM value, see chapter 4.3 Pulse Width Modulation).

The program (see Annex A6) are formed by an infinite loop (either *For* or *While*) and inside of it another one, but it was a *Do-While*. This one had a condition which could break it; this condition was the X and Y axes were completely balanced. If the balanced was reached, the program could read again the new general power (PWM).
The code belonged *Do-While loop* was based on the quadrotor dynamics (section 2.3.2 *Quadrotor MAV* design). If the device had changed the axis X and Y position, the value of the IMU would have changed the width of the PWM. DC motor actuation[16] in function of *MPU-6050* data value (Figure 82) is going to be explained in the next bullets:

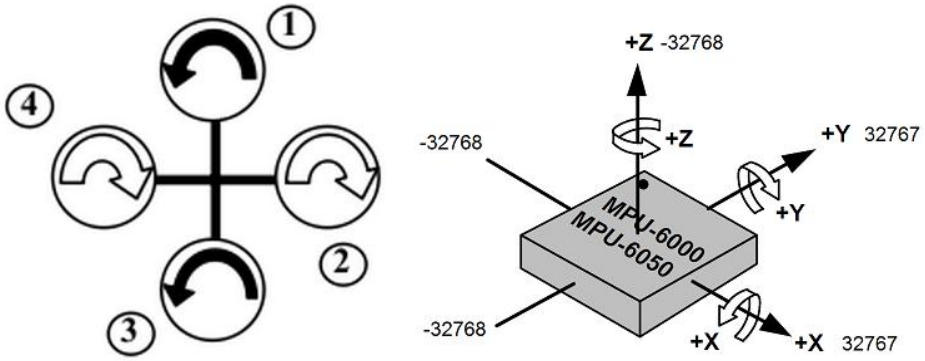


Figure 82 – DC motors structure position and top-bottom axes values MPU-6050.

- X axis:
  - If the data value is less than 0 (balanced point) DC motors 1 and 4 work.
  - If the data value is less than 0 (balanced point) DC motors 2 and 3 work.
- Y axis:
  - If the data value is less than 0 (balanced point) DC motors 3 and 4 work.
  - If the data value is less than 0 (balanced point) DC motors 1 and 2 work.
- Z axis:

[16] http://file.scirp.org/Html/12-7900231_35654.htm

- If the data value is less than 0 (balanced point) DC motors 1, 2, 3 and 4 work.
- If the data value is less than 0 (balanced point) DC motors 1, 2, 3 and 4 work.

Doing a Proportional Derivative programming output to keep the MAV's stability, limiting the width of PWM and getting the equation of Z axis (obtained by Lift vs. Current graph, see Annex A1 as example) I could have got a hover flight.

5.2 Flight test

I loaded the program (see Annex A6) into the microcontroller. When the device was on land the system works. Then, the empiric test took place. For that I tied the *AMEWI BLASTERx80* to a lamp. I did not want the MAV flew without control over a wall or to the floor, see Figure 83.

Figure 83 – Empiric test.

The result was disappointing, the MAV never got a stability flight, the system got freeze every time and the DC motors work random and independently of each other. I tried solutions searching what were the problems. The chronological solutions are going to be explained:

- I removed the Derivative code from the program. Only with the Proportional PWM output part had more or less the same result. I thought the Proportional Derivative code but it was too slow.
  I did not to try to implement an integral as quadrotor power plant response is an Integral itself [11] [14].
  The Proportional Derivative could have worked, but I never really checked as electromagnetic interferences and vibration arose.

- The problem could have been a capacitor in parallel to DC motor that I did not set it up, and *BLASTERX80* has set up (Figure 72). I set the capacitor as the PWM has peaks over 5 times the top value. The capacitor had $10\mu F$ value. I solved the peak (Figure 86), but it had not change the problem.

- Another solution was the DC motor works overloading due to the wings. That overloads demand more current in the stator to surpass the rotor's rotational moment. More current, more electric noise. To reduce this noise we need 3 capacitors in each motor, see Figure 84.

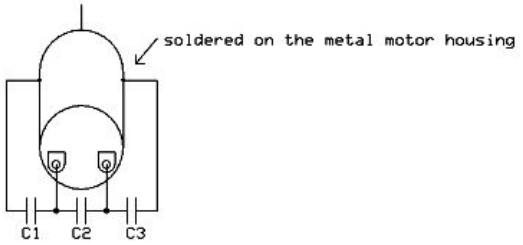


Figure 84 –Capacitor's feature to reduce electric noise.

We changed the capacitor values in a lot of times, but the stability never improved.
In addition, these capacitors configuration worked together with capacitors in parallel with $V_{CC-Ground}$ connection in the IMU and microcontroller in order to avoid noise. I tried this solution (Figure 85) with a bunch of capacitors as electrolytic and ceramics, and changing values too.

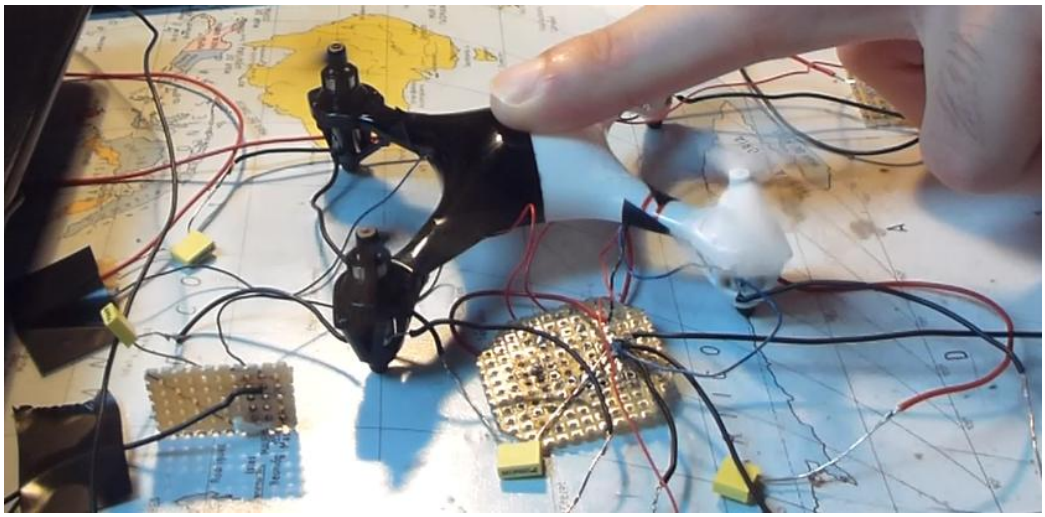


Figure 85 –Capacitor's feature to reduce electric noise (MAV test).

I read all kind of information about DC noise[17], but no-one had such noise as this system. The problem could not be the DC motors current, maybe the microcontroller or the IMU (the power plant system worked rightly when you only use constant PWMs).

[17] http://www2.etown.edu/wunderbot/DOWNLOAD/NPC_T74_Motors/Application_Note_Motor_Noise.pdf

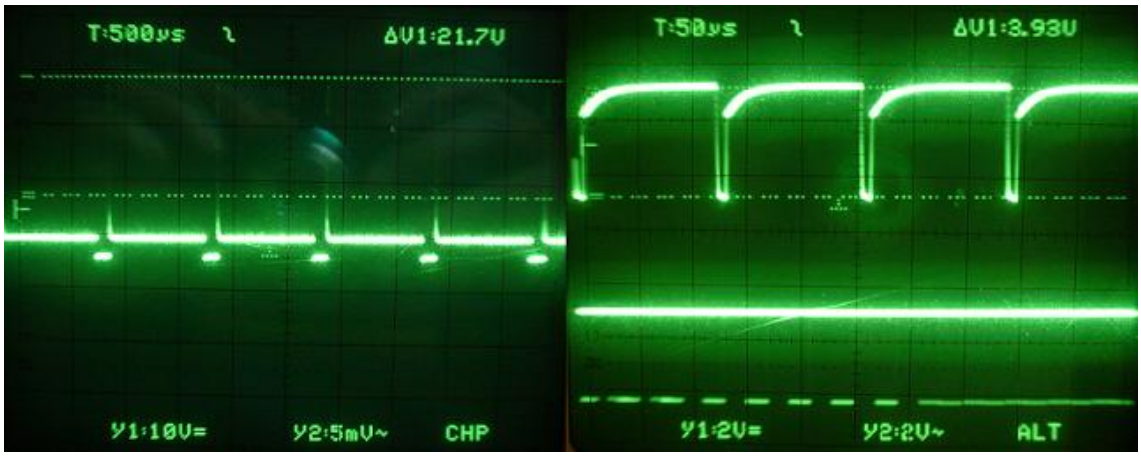


Figure 86 –High peak voltage values (PWM) and signal DC motor with capacitor.

5.3 Electromagnetic interferences

Discarded DC motor noise as problem, I needed to keep attention in the communication between the *MPU-6050* and *MSP430F2274*. I needed to understand scrupulously the communication signal.

When I connected the oscilloscope to I2C bus (either SDA or CLK) as the power plant was working, I saw the communication has a PWM pattern noise, see Figure 87.

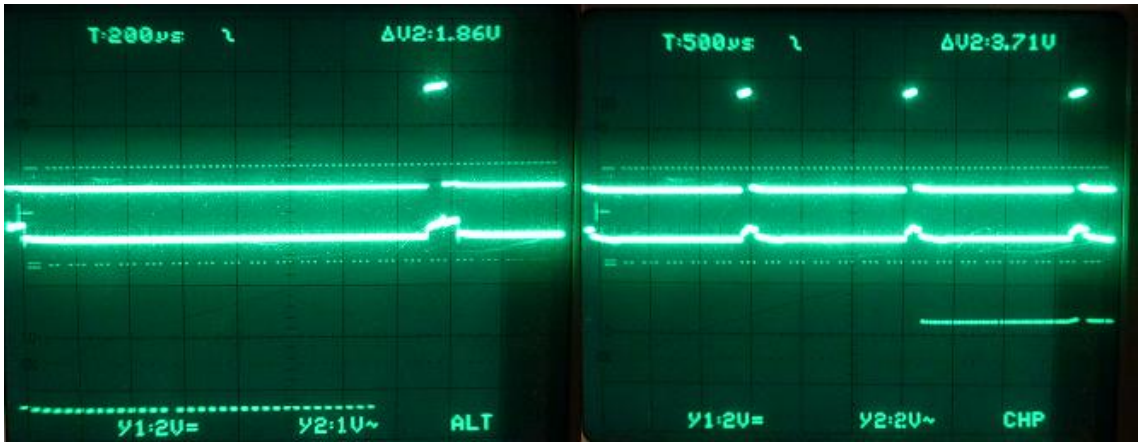


Figure 87 –I2C corrupted by PWM pattern (top signal PWM and bottom one CLOCK).

While PWM width increases got a moment that I2C communication broke (Figure 88) the communication (no clock pulse). The oscilloscope display shows PWM signal (2 top lines) and I2C clock signal (2 bottom lines)

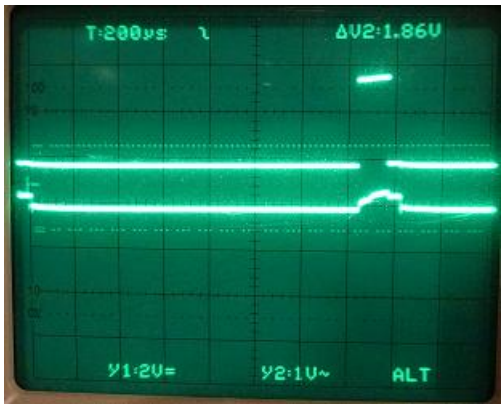


Figure 88 –I2C corrupted by PWM with no clock signal.

The MAV dynamics when it faced the broke communication was an absolutely chaos. The PWM had a random pattern and sometimes was constant in specific width. This phenomenon took place when the system worked in low level PWM's width during several minutes and when the PWM got high width in few seconds (in permanent working). The system kept I2C communication if the power increase is not constant, see section 5.4.
Technically this problem could have the source in the pull up resistors needed by I2C communication. I set two resistors as I explained in section 4.5 and showed in the Figure 73, but it did not solve the problem, as the *eZ430-RF2500* had got internal resistors for this kind of communication.

Finally, I found information about a problem called *crosstalk* [12] It is a phenomenon when a signal transmitted in one circuit creates an undesired effect in another circuit channel (e.g. between microcontroller (PWM) and IMU (I2C communication)). There are three different *crosstalk* causes:

- Capacitive coupling: When a communication signal disturb another communication due to the capacitance between them.
  The solution is reducing this noise with a capacitance between both lines.

- Inductive coupling: Caused by magnetic induction between lines. The solution is setting inductances in order to reduce external magnetic fields.

- Conductive coupling: Common impedance between two circuits (ground reference). It is a sum of impedances of wires and by electrostatic or magnetic effects. The solution is setting resistors and reducing the length of ground reference.

There are other solutions that can reduce the interferences between lines:

- Put away I2C wires from PWM signals. I was impossible as I worked in a Micro Air Vehicle (Figure 89).

Figure 89 – Wires DC motors and communication.

- Set only one wire to ground reference, and with thickness section. But, I worked on a protoboard (Figure 90) and set all with only ground reference thickness wire.

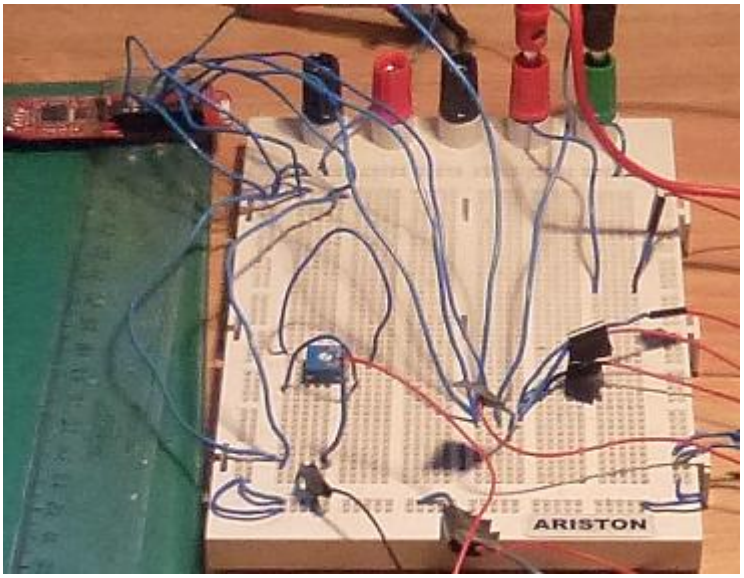


Figure 90 –Protoboard circuit.

- Reduce the length of wires. I needed the quadrotor flying (Figure 89).

- Twist wires. I did it and I set a ferrite to erase the rest of noise. It was not a solution.

Finally, after a lot of attempts I solved the PWM pattern problem (see Figure 91). I isolated the ground of microcontroller and the ground of the power source. I solved the *crosstalk* problem, but not the freeze problem between the microcontroller and the IMU. In my opinion, it could be *loading effect* between devices or capacitive coupling. Or maybe it was due to the microcontroller was designed for Arduino microcontrollers with different internal resistors.

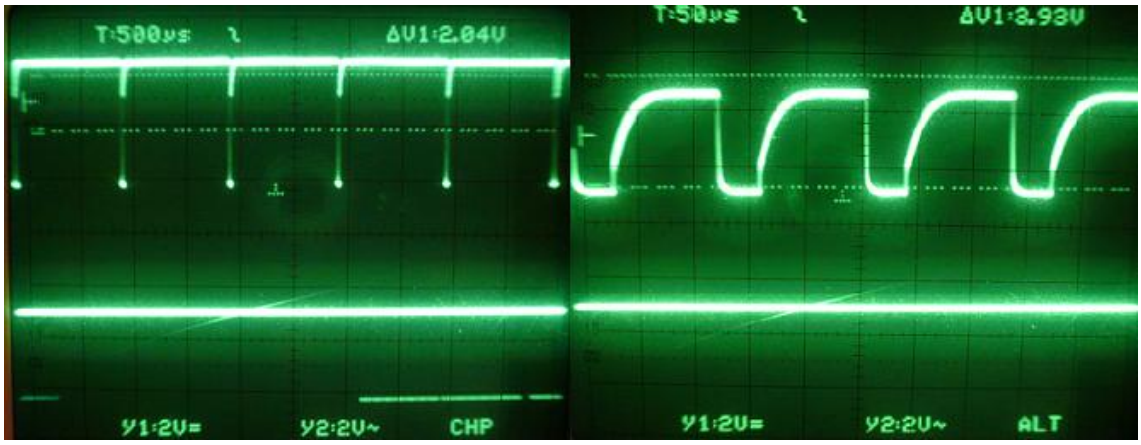


Figure 91 –I2C without interference and PWM signal.

5.4 Vibrations

In this section I discovered the electromagnetic problem trying to solve the vibrations (last section 5.3). I tried to solve the IMU vibrations thinking these could be the solution of the instability problem. The source of vibrations was the DC motors themselves. When they worked the fans which work together with every DC motor made vibration. The result was more DC motor speed, more vibrations.
The accelerometer was under vibration; consequently the width of PWM changed every time and made our MAV unstable. Solving this problem I could get the MAV stability. I chose two ways for solving the vibration vibration:

- Kalman's filter.
- Reducing noise using probability and statistics

In the first solution I needed to find information [13] about how the filter works. Kalman's filter is a linear quadratic estimation; it takes system input values which can contain noise or other inaccuracies. The filter can foresee a future event unknown the future inputs, it works over the time/cycles. Kalman's filter makes a statistically optimal estimate of the system underlying state. I used the discrete Kalman's filter (Figure 92).

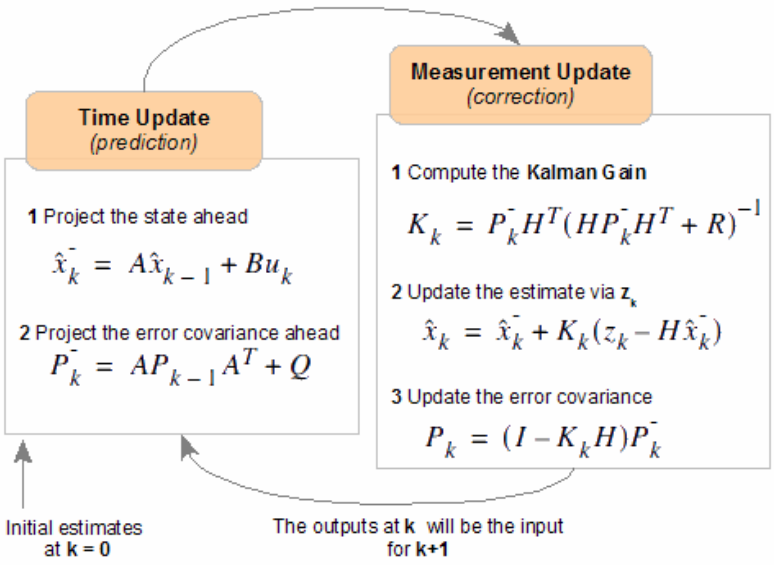


Figure 92 –Discrete Kalman's filter.

Time Update (prediction) X_k (a priori) is the last value known, u_k is the noise (zero, it can be modeling as white noise). P_k (a priori) is the covariance error and Q is the white noise which belongs to the process,
In the *Measurement Update* (correction) K_k is an optimal *Kalman* gain (priori inputs), which takes an important role to reduce the error over the input value. This value changes while time pass (it has a strong relation with covariance error). $\hat{X}_k$ (a posteriori) is the estimated value, and $\hat{P}_k$ (a posteriori) is the covariance error which reduces this value each time as the error is decremented.
Normally the value of *A* and *H* constants are *1*, because in real life is odd you find a system with other values (remember Normal Standard Distribution in real cases).

I implemented the code and uploaded to the microcontroller, finding to solve the vibration problem. The Kalman's filter had a problem, it needed a couple of times or cycles to work and reduce the error (see Figure 93). Those cycles did the BLASTERx80 MAV unstable. This one never got a specific value in a system where the dynamics changes every time the accelerometer value.

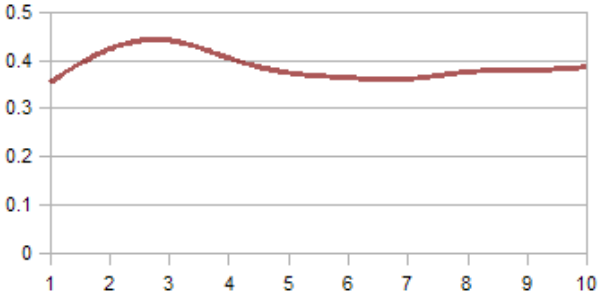


Figure 93 –Kalman's filter get the correct value after several cycles.

A solution could be change the microcontroller speed in order to be faster than the IMU. That could do the microcontroller reading the same IMU value during a couple of cycles, but I would need to know how many cycles the value remains. Unfortunately, Kalman's filter was a solution, but at that moment I did not discovered the *crosstalk* problem.

Finally, unknowing the *crosstalk* problem and thinking the Kalman's filter was too slow. Another solution was reduced the noise through probability and statistics, thus, I did not need cycles getting the correct accelerometer value. Getting that idea, I needed working with a program which gave the possibility to work with a lot of data. The program I chose was MATLAB (Figure 94), as I know this professional program and I used it several times at the university.

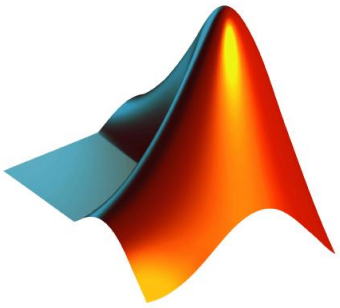


Figure 94 –MATLAB logotype.

To reduce the vibration I had to understand its shape/pattern. I knew the DC motors have similar electrics (resistors and inductances) and physics features (same fan). The DC motors-fans set had more or less the same amount of vibration. Thus, I could characterize the vibration as Gaussian shape (white noise, see Figure 95) theoretically.

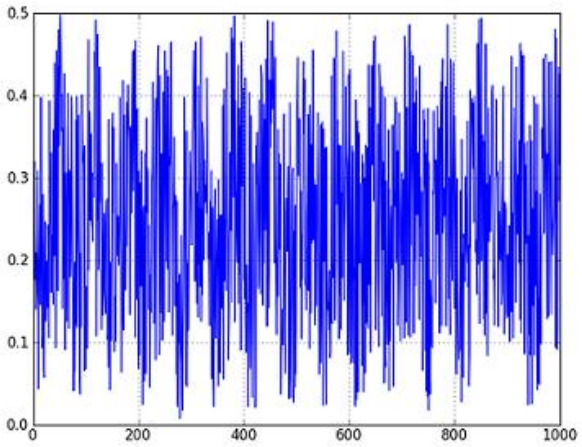


Figure 95 –Gaussian Noise shape[18]

[18] http://pandasplotting.blogspot.com.es/2012/06/autocorrelation-plot.html

Getting that idea, I implemented a program which took communication between the microcontroller and the IMU by RS-232 standard. The program saved every IMU data in a longer vector (1000 values) with only one power value. I did a test and the result was concluding (Figure 96), the vibrations had the same distribution as *white noise*, but I was not concluding.

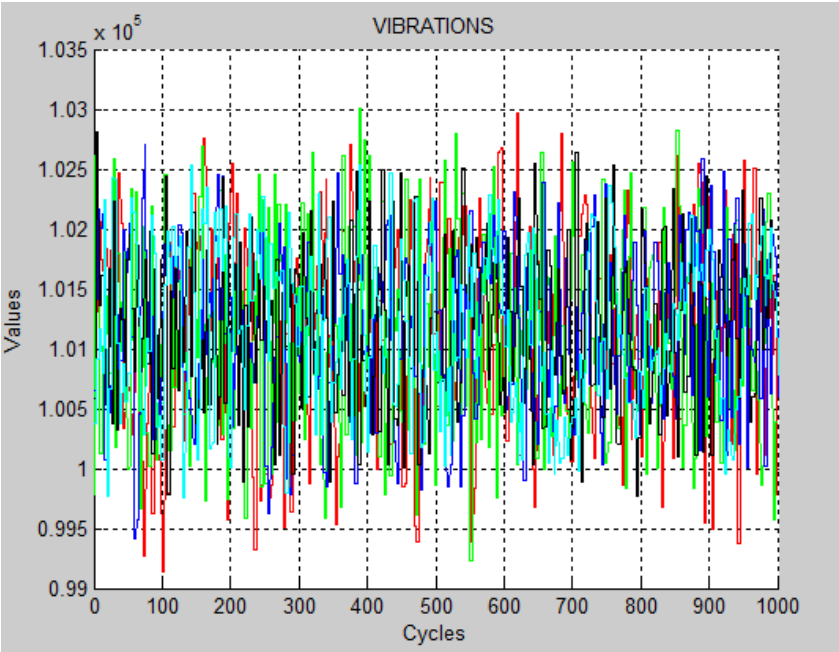


Figure 96 –Vibration pattern PWM duty cycle 80%.

The noise/vibration seemed to be Gaussian distribution. Getting this idea the MATLAB program plotted a histogram in order to see how many values were the most common and how many times the data took each value. After that, I needed plotting the data and determine the distribution (Figure 97). I used the command called *histfit()* (Annex A7). That command makes and histogram and find the mathematic distribution, in this case Normal Standard Distribution.

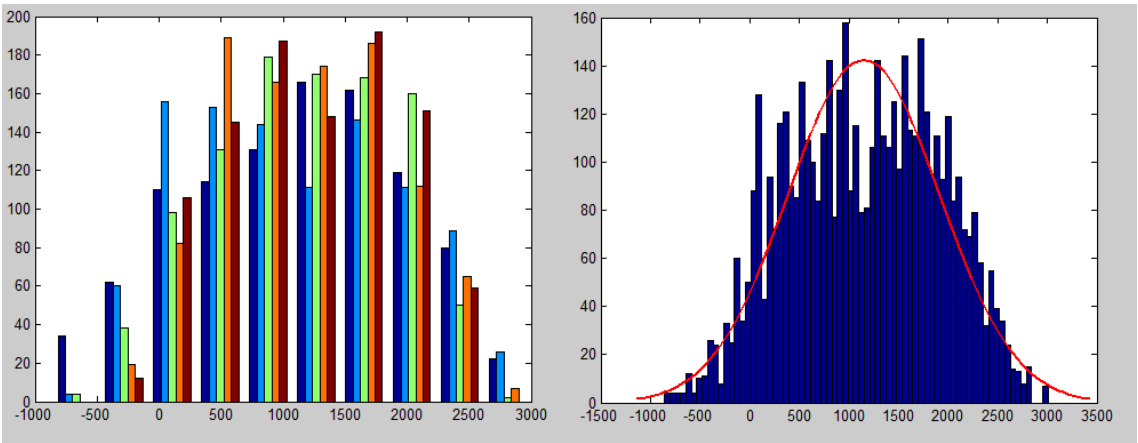


Figure 97 –Histogram and distribution data (90% duty cycle PWM).

To reduce the noise I implemented another program with some code lines from the first test MATLAB program:

- It took the accelerometer data from MPS430 via Serial Port.
- While the PWM changed and DC motors were working, the vibrations values went to a matrix. That matrix took 1000 values for each PWM increment.
- When the matrix was full DC motors, the program started plotting different graphs as:
  - Maximum value vs. common value.
  - Common value vs. minimum value.
  - Maximum value vs. average.
  - Average vs. Common value.

- I could see how change the μ (mean) and σ (variance) value through the power change.
- Using some values from Normal Distribution (Figure 98) I could erase the high values from the average (high top difference was 1500) through the confidence interval with the next values 80%, 85%,90% and 95%.

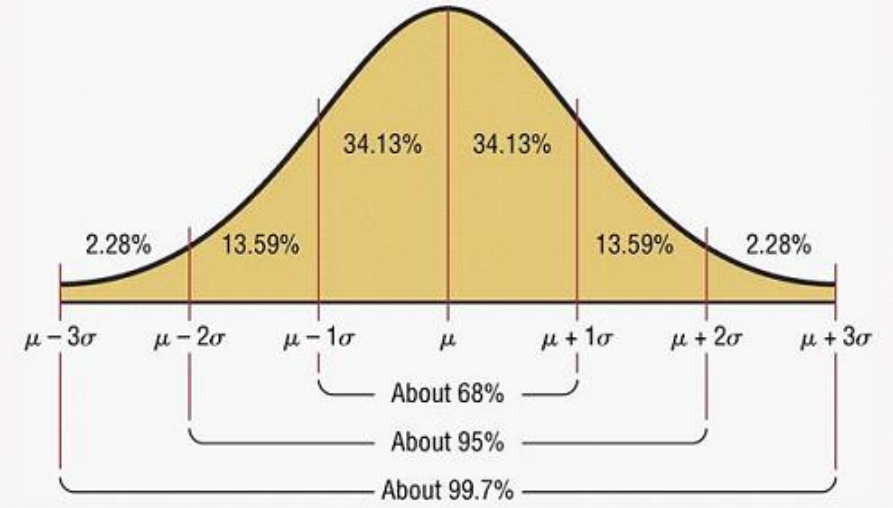


Figure 98 – Normal Standard Distribution[19].

- Obtaining the four different graphs I could see the result and determine which the best was.
- Finally, I could get the equation curve from the graph I chose which determines the confidence interval in every power value.

The program took a couple of hours to fill the matrix due to a lot of values the program had to take (every 1000 values the program needed 3 minutes).
If in the future the MAV loses the balance flight; you could re-calibrate the curve and set it up again into the microcontroller.

[19] http://rchsbowman.wordpress.com/2009/11/29/statistics-notes-%E2%80%93-properties-of-normal-distribution-2/

In the process to avoid vibrations I found the problem explained in the section 5.3 electromagnetic interferences. The communication was broken all time and the device never could complete the calibration test. I got the Figure 96 and 97 due to a lucky strike that PWM didn't change the value and the program took those values.

CHAPTER 6. CONCLUSIONS

In this chapter I am going to show how many conclusions I got. The aim and the results take an important role. How I solved the problems (throughout the project) determine the following concluding arguments.

6.1 General conclusions

After doing different MAVs and trying to understand how works a commercial quadrotor I got different conclusions:

- If you want to do a handmade MAV, you will need specific tools and materials.
- Not always the specific handmade MAV configuration will work, but there are a lot of power plants and you will find the best to make, in my case quadrotor.
- Making a fragile power plant is only a possibility if you developed a system in another similar strength power plant.
- If you are a beginner in MAV world try to learn from MAV trainers. They are sold in commercial surfaces.
- Be careful with electronic circuit, it has to be very well balanced toward interferences.
- Doing reverse engineerable helps to understand how a system works, and develop your own system.
- Interferences into the communication are sometimes complex to solve. There are different factors as length communication, crosstalk…
- There are many solutions to fix the interferences, but sometimes the design of the circuit doesn't help to implement specific solutions.
- Designing something needs face a lot of problems at same time and you will need to isolate the problems one by one.
- Using bachelor knowledge is a good point to start solving problems or facing different possibilities to get the solution
- Sometimes you will be limited by your knowledge and you will learn more about different fields you never seen to get a solution
- Electronics is a complex field, every element spread electromagnetism and you need to isolate each one.
- If you can do a printed circuit board for a prototype you will be able to avoid a lot of interference problems as crosstalk, fail connections or external signals.
- Buying peripheral designed for other microcontrollers will get the system does not work rightly (MSP430F2274 and Arduino's design ITG/MPU-6050).

6.2 Future work

There are many things I could not get:

- To do an efficient Dragonfly MAV mechanism or solving the vibration and friction problem in the Coaxial Rotor MAV device.

- I could not solve why the communication between the microcontroller (*MSP430F2274*) and MEMS (*ITG/MPU-6050*) froze when the DC motors works.
- Getting a hover flight with *AMEWI BLASTERx80* power plant.

But I did many other aims that could help the future engineering student, if he or she wanted to solve the challenges I could not get. I am going to show the problems cannot get malfunction in the system:

- DC motor electric noise
- DC motor PWM's peaks of current
- Pull-up resistors on I2C communication
- External electromagnetism noise.
- DC motors' vibration

Finally, work I did and you will find in this Project:

- MPS430 program to balance the power plant
- Serial communication protocol
- How to do power plant graph (Lift vs. Current)
- Vibrations program
- I2C communication program
- Power plant (MOSFET, DC motors, fans…)

6.3 Environmental impact

Every device will make remains in the process to make it, and more ones when the device will not properly.

On the one hand, I employed many different materials, as plastic, brass, wood and electronic elements. All of them could be submitted in recycling processes. Reusing plastic to make package, wood to make furniture, brass to make can…

On the other hand, the devices of this project could help to optimize energy. The *AMEWI BLASTERx80* is an indoor MAV, which can be used for:

- Keeping a constant temperature in a room or living room.
- To detect if there are people in a salon to switch off the lights.
- Alert in case of emergencies as fire or fumes
- Repeater Wi-Fi signal
- Surveillance.
- Rescue Missions

BIBLIOGRAPHY

[1] Tristancho, J., Mansilla, S, P., **An intelligent Scenario For New Unmanned Aerial Sytems**, Working presentation on 5th International Conference on Intelligent Environments, 2009.
DOI: 10.3233/978-1-60750-034-6-285

[2] Mueller, T. J., Editor, **Fixed and Flapping Wing Aerodynamics For Micro Air Vehicle Applications**, Progress Aeronautics and Astronautics Working (195):3, 2001.
DOI: 10.2514/4.866654

[3] Valanis, K. P., Editor, **Advances in Unmanned Aerial Vehicles**, Intelligent Systems, Control, and Automation: Science and Engineering, (33): 172-174, 2007
DOI: 10.1029/2007GL032508

[4] Jonguerius, S. R., Lentink, D., **Structural Analysis of a Dragonfly Wing**, Experimental Mechanics; 50:1323-1334, 2010.
DOI: 10.1007/s11340-010-9411-x

[5] Ma, K., Chirarattananon, P., Fuller, S., Wood, R., **Controlled flight of a biologically inspired, insect-scale robot**, Journal Science, 05/2013; 340(6132):603-7, 2013
DOI: 10.1126/science.1231806

[6] Azuma, A., Azuma, S., Watanabe, I., Furuta, T., **Flight Mecanichs of a Dragonfly.** Institute of Interdisciplinary Research, Faculty of Engineering, The University of Tokyo, Tokyo, Japan, J. exp. Biol. 116, 79-107, 1985.

[7] DiLeo, C., Deng, X.., **Experimental Testbed and Prototype Development for a Dragonfly-Inspired Robot.** IEEE/RSJ International Conference on Intelligent Robots and Systems, San Diego, USA, Oct. 29 2007-Nov. 2 2007.
DOI: 10.1109/IROS.2007.4399418

[8] Jongerius, S. R., Lentink, D., **Structural Analysis of a Dragonfly Wing.** Experimental Mechanics, (50):1323-1334, 2010.
DOI: 10.1007/s11340-010-9411-x

[9] Castillo, P., Lozano, R., Dzul, A.E., **Modelling and Control of Flying Machines.** Advances in Industrial and Control, 2005.
DOI: 10.1007/1-84628-179-2

[10] Padfield, G. P., **The theory and Application of Flying Qualities and Simulation Modelling.** Helicopter Flight Dynamics, Second Edition, 1996.
DOI: 10.2514/2.4396

[11] Bracke, J.., **Interfaz Labview para programar el sistema de control de quadrotores.** Bachelor Final Project, 2010.
Link:
http://upcommons.upc.edu/pfc/bitstream/2099.1/10490/1/memoria.pdf

[12] Balcells, J., Daura, F.,Esparza, R., Pallás, R., **Electromagnetic Interferences in electronic systems.** Marcombo Editorial, 1992.

[13] Castañeda, J. A., Nieto, M. A., Ortiz, V. A., **Analysis and application of the Kalman filter to a signal with random noise.** University oftechnology Pereira, Scientia et Technica, (18):267-274, April 2013.

[14] Mayorga, R. A., **Navigation System for Quadrotor Air Vehicles.** Final Bachelor Project. University Polytechnic of Catalonia, 2009.

[15] http://pdf1.alldatasheet.es/datasheet-pdf/view/527757/STMICROELECTRONICS/STM8S005K6.html (Jun. 2012)

[16] http://www.invensense.com/mems/gyro/documents/PS-MPU-6000A-00v3.4.pdf (Sep. 2013)

[17] http://www.datasheetarchive.com/dl/Datasheet-081/DASF0035135.pdf (Sep. 2008)

[18] http://www.ti.com/lit/ug/slau144j/slau144j.pdf (Jul. 2013)

[19] http://www.ti.com.cn/cn/lit/ds/symlink/msp430f2274.pdf (Aug. 2012)

[20] http://www.vishay.com/docs/65701/si2300ds.pdf (Jan. 2010)

[21] http://invensense.com/mems/gyro/documents/RM-MPU-6000A.pdf (Sep. 2012)

[22] http://www.analog.com/static/imported-files/data_sheets/ADXL330.pdf (2007)

ANNEXES

TÍTOL DEL TFC: Avaluació de dissenys de microvehicles aeris fets a mà i implementació d'un quatrirotor

TITULACIÓ: Enginyeria Tècnica Aeronàutica, especialitat Aeronavegació

AUTOR: Marcos Navarrete Rodríguez

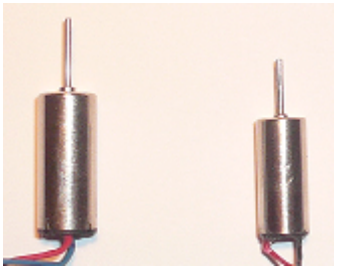
DIRECTOR: Joshua Tristancho Martínez

DATA: 24 de novembre de 2013

ANNEX A. QUADROTOR MAV POWER TEST

A.1 Power test Quadrotor MAV

There was a briefly time period I used the *Quadrotor MAV*. It was set up by an accelerometer called ADXL330 designed by Analog Devices [22**¡Error! No se encuentra el origen de la referencia.**]. That one is a small, thin, low power, complete 3-axis accelerometer with signal conditioned voltage outputs (ADC). The measures had a minimum full-scale range of ± 3g. I only needed to connect their outputs to microcontroller and programming the ADC (see Annex A6).
The result was unsatisfactory as the size of DC motors was too small and working rightly they consume 3 Amperes, more than same the *AMEWI BLASTERx80* ones. In my opinion, the high current through the tiny (see the next picture) DC motors generated a huge electromagnetic fields which affects the accelerometer working.

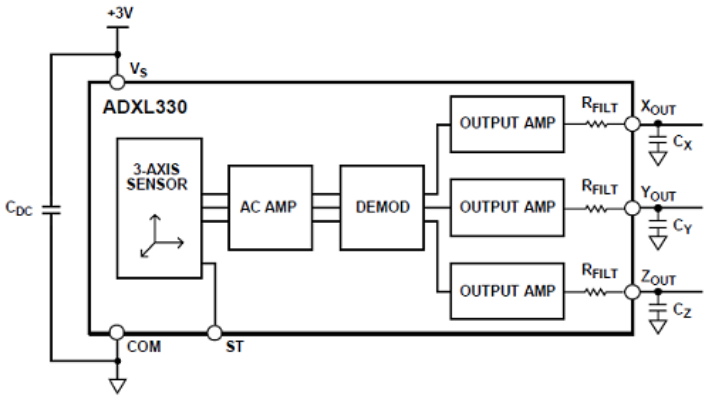


BLASTER X80's DC motor vs. Quadrotor MAV's DC motor.

Consequently the PWMs were unstable and inefficient. The problem could remain in the circuitry due to the ground reference was an aerial and the *bandwidth* was fixed in *400MHz*. Getting that bandwidth was set up capacitors according the next table.and block diagram scheme. The bandwidth feature affects the amount of radiation that could affect the circuit of *ADXL330K* (the ground reference was an aerial that took the electromagnetism of DC motors).

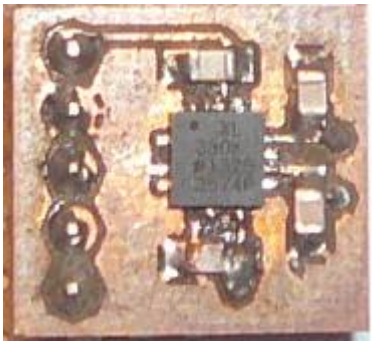
Table – Filter Capacitor Selection

Bandwidth (Hz)	Capacitor (µF)
1	4.7
10	0.47
50	0.10
100	0.05
200	0.027
500	0.01



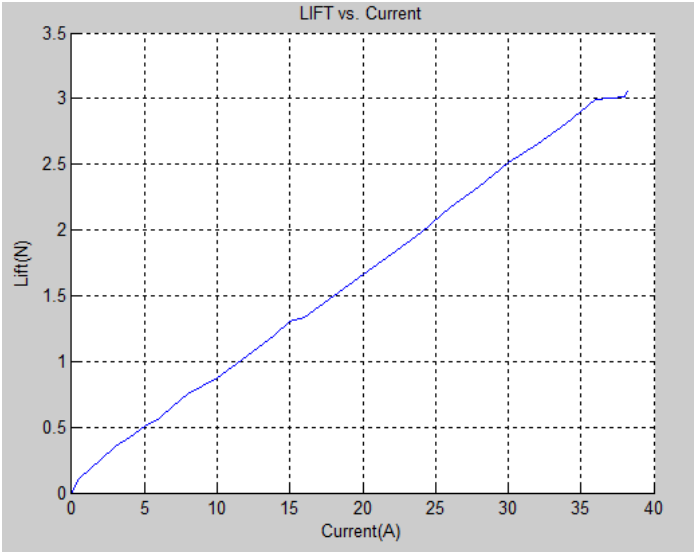
Functional block diagram (ADXL330).

Before I found that problem, I did a power test. With the test I did two graphs (MATLAB), which showed how changed the current/PWM's width versus de lift force generated by the *Quadrotor MAV*'s fans. Using the MATLAB program, I found the polynomial function of the curve and understood the relation between the variables. That curve equation determines the hover flight in the vertical axis (Z).



Circuit (ADXL330K).

Lift versus current through DC motor (following graph):



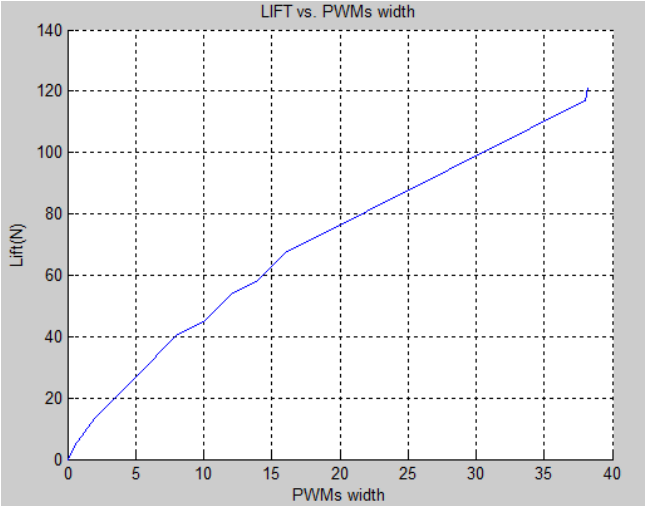
Lift vs. Current (ADXL330K).

The curve equation *first(x)* of last graph with 95% of confidence bounds took the following values:

```
Linear model Poly1:
first(x) = p1*x + p2
Coefficients (with 95% confidence bounds):
  p1 =      0.07934   (0.07827, 0.08041)
  p2 =      0.08798   (0.06603, 0.1099)
```

Curve equation Lift vs. Current graph.

Lift versus PWM's width (following graph):



Lift vs. PWM's width.

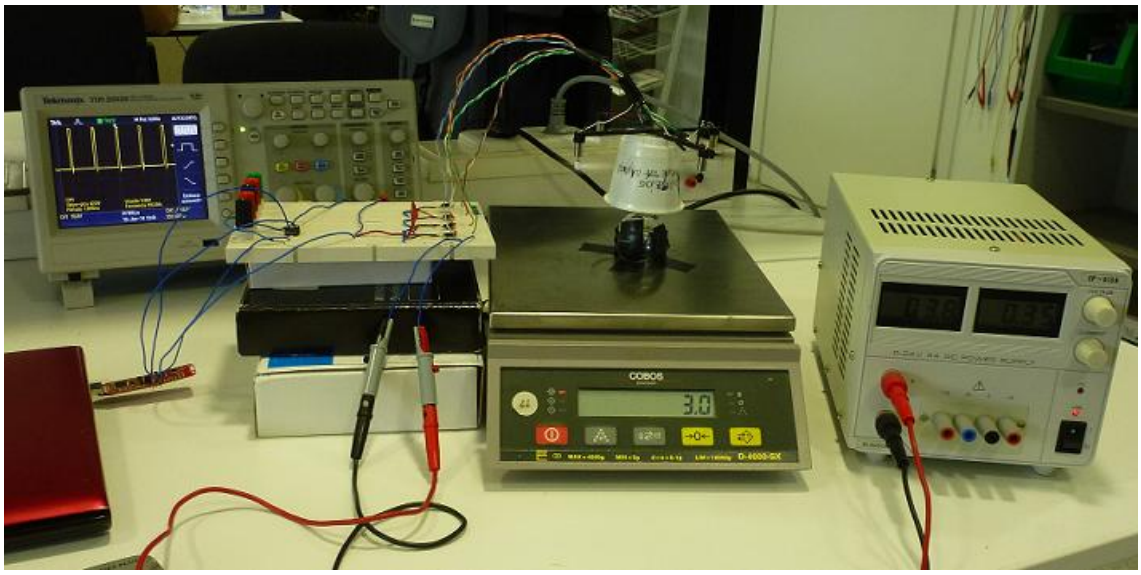
The curve equation *second(x)* of last graph with 95% of confidence bounds confidence bounds took the following values:

```
Linear model Poly1:
second(x) = p1*x + p2
Coefficients (with 95% confidence bounds):
  p1 =       2.942   (2.768, 3.117)
  p2 =       12.32   (8.743, 15.89)
```

Curve equation Lift vs. PWM's width.

Surprisingly, both equations were the linear. The first one shows the consumption, useful if we have to choose a battery or determine the whole consumption. The second one shows how change the width of the PWM every time the lift changes.
Unfortunately, the chassis was fragile and that was the reason I chose *AMEWI BLASTERx80*.

The test took place in a laboratory with different tools (Power source, oscilloscope and a scale) showed in the next picture.



Power system test.

I needed to know the *Quadrotor MAV* mass. That one using the graphs Lift vs. PWM's width helped to know the width and current the device would have needed to.

Microcontroller and chassis mass.

Battery and DC motor mass.

Fans mass.

Quadrotor MAV element's weight

Element	MSP430	Chassis	Battery	DC motors	Fans
Mass	1.7 gr.	2.8 gr.	4.3 gr.	4.8 gr.	1.2 gr.

The total mass was *14.8gr*. The device mass was conditioned by Baterry and DC motors mass.

A.2 Communication protocol MSP430F2274 to other devices.

The information data of Inertial Measurement Unit had to be submitted with a data process. The system needed different bytes with IMUS's data in order to read them and in the correct order.
The microcontroller and programs had the same code for reading and sending all kind of information data. In this document is explained how I did this.

The data information is stored in 16-bits (-32768 to 32767), I needed to break that in four parts dividing the information for each bit position value, see the following table.

-32768	1000	0000	0000	0000
32767	0111	1111	1111	1111
	4069	**256**	**16**	**1**

Dividing data information.

I set a *head-byte*, which determines the check byte, the negative and positive numbers and the order of sending/receiving, see the distribution in the next figure.

Byte structure.

Getting the head-byte I needed to determine each value, explained in the following table (each byte has bottom and top value, that one depends the head-byte and the minimum and maximum value break data).

NEGATIVE DATA VALUES									POSITIVE DATA VALUES								
0	1	0	0	0	0	0	0	64	0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	1	1	79	0	0	0	0	1	1	1	1	15
0	1	0	1	0	0	0	0	80	0	0	0	1	0	0	0	0	16
0	1	0	1	1	1	1	1	95	0	0	0	1	1	1	1	1	31
0	1	1	0	0	0	0	0	96	0	0	1	0	0	0	0	0	32
0	1	1	0	1	1	1	1	111	0	0	1	0	1	1	1	1	47
0	1	1	1	0	0	0	0	112	0	0	1	1	0	0	0	0	48
0	1	1	1	1	1	1	1	127	0	0	1	1	1	1	1	1	63

Byte order sending/receiving.

Where:

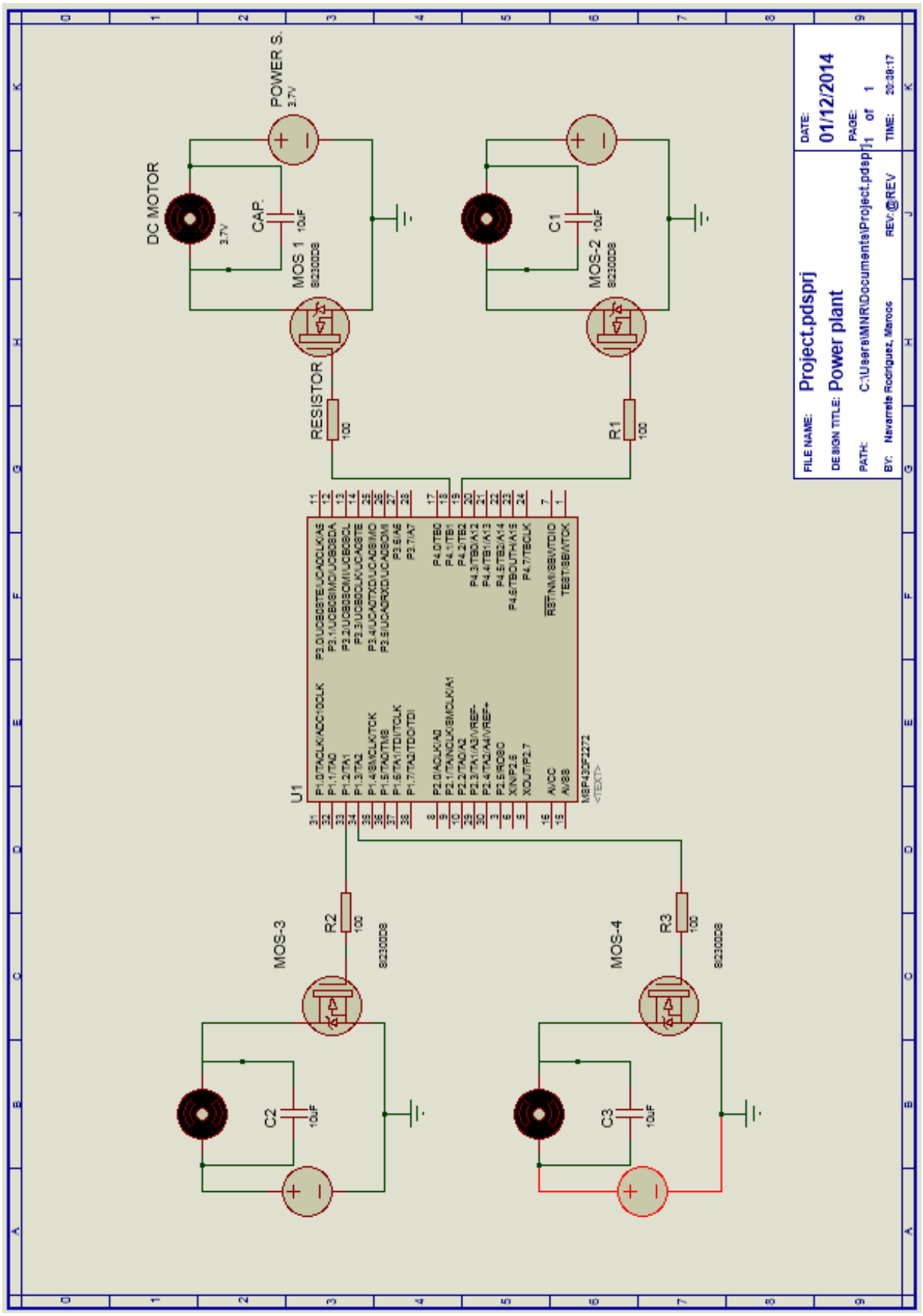
- The green cell is the check byte; it belongs to a check MATLAB protocol. I thought maybe some bytes were losing during communication RS.232. The problem was the crosstalk (see 5.3 Electromagnetic interferences).

- The second byte's cell in red colour determines if the value data is positive (0) or negative (1).
- The third and fourth one in yellow colour set the byte's order (00:1, 01:1, 10:2, 11:3).

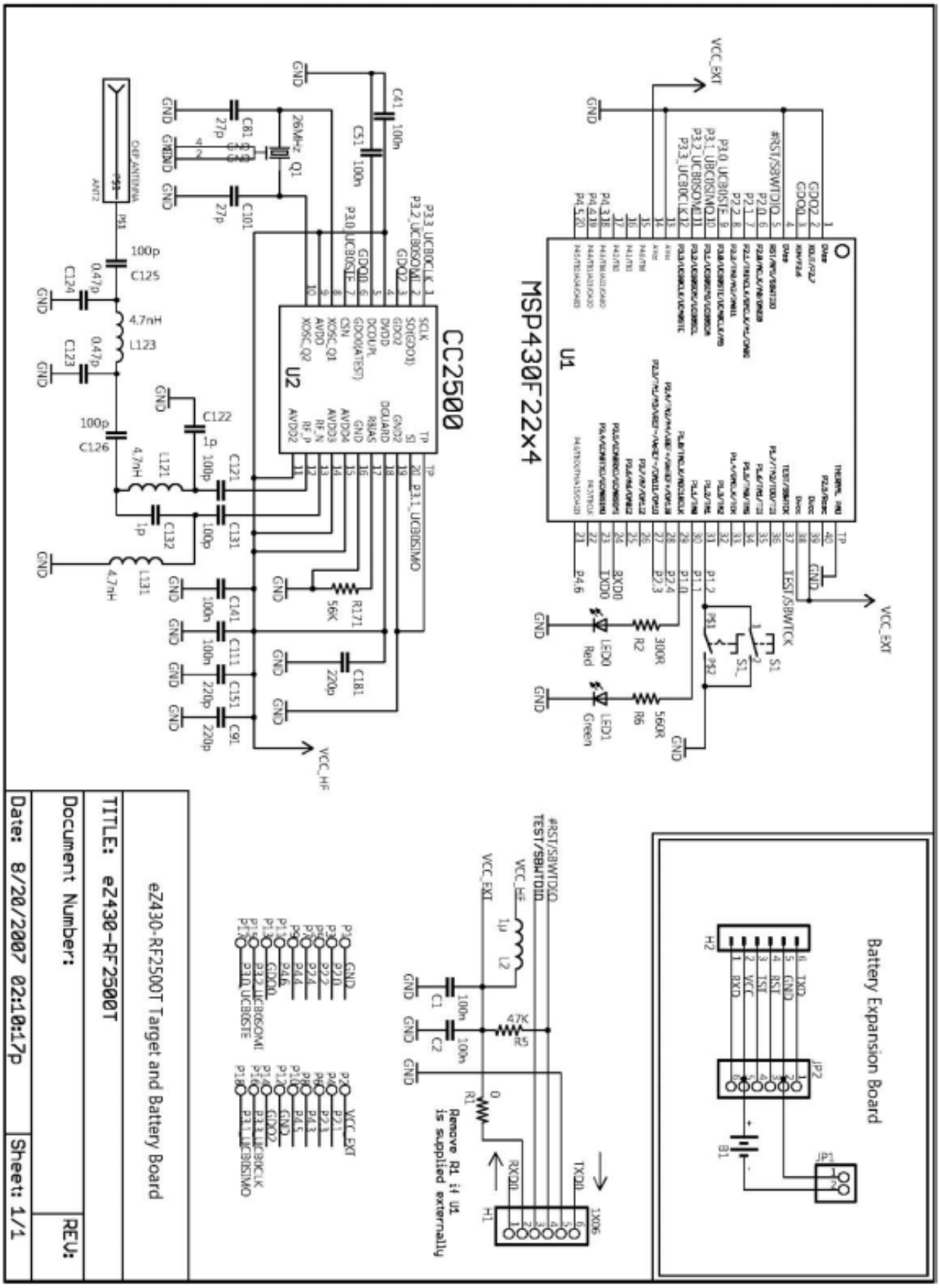
- The rest cells in blue contain the data itself. The data has top value (1111) and bottom value (0000) which determines the final decimal value of the byte (white cells).

```
if (buffer[count_buffer] <= 63 && buffer[count_buffer] >= 48 && count == 0)
                {
                          powerByte1 = (buffer[count_buffer] - 48) * 4096;
                          count = 1;
                }

                if (buffer[count_buffer]<= 47 && buffer[count_buffer] >= 32 &&
count == 1 )
                {
                          powerByte2 = (buffer[count_buffer] - 32) * 256;
                          count =2;
                }

                if (buffer[count_buffer] <= 31 && buffer[count_buffer] >= 16 &&
count == 2 )
                {
                          powerByte3 = (buffer[count_buffer] - 16) * 16;
                          count =3;
                }
                if (buffer[count_buffer] <= 15 && buffer[count_buffer] >= 0  &&
count == 3 )
                {
                          powerByte4 = buffer[count_buffer];
                          count =4;
                          power  =  powerByte1  +  powerByte2  +  powerByte3  +
powerByte4;
                          break;
                }
```

A.3 Power circuit.



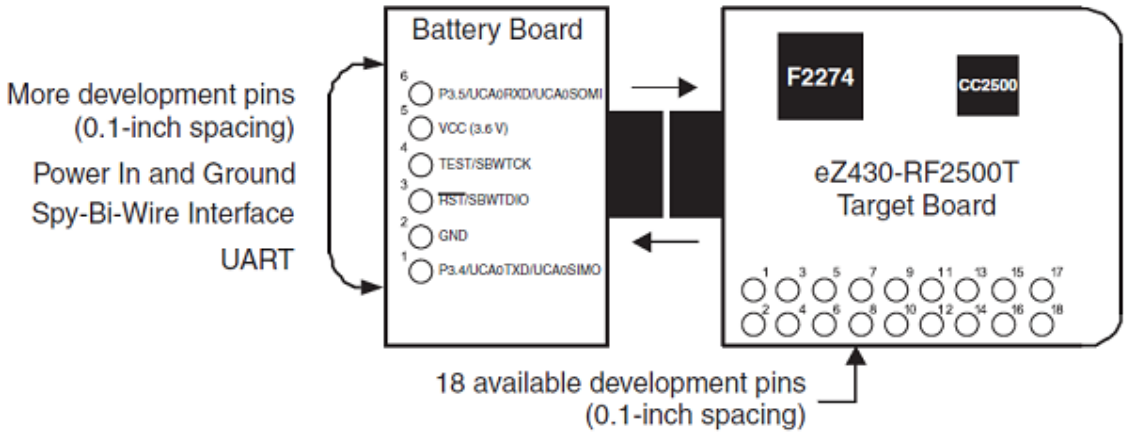
A.4.Electronic scheme MPS430F2274



A.5. Pin configuration

The *eZ430-RF2500* Target Board Pinouts has 18 pins and Battery Board Pinout 6 ones. The 18 pins which belong to target board are available to us (setting different wires and devices). In the present project we need to set 8 pins (without ADC) or 9 pins (with ADC):

- Pin 6-7-9-10 to set timers getting 4 PWMs.
- Pin 15 to set SCL I2C clock in I2C mode.
- Pin 18 to set SDA I2C data in I2C mode.
- Pin 1 and 6 set for transmitting and receiving data in UART mode (Battery Board).
- Pin 8 to set Analog-Digital Converter.



Ez430-RF2500T pins.

Pin	Function	Description
1	GND	Ground reference
2	VCC	Supply voltage
3	P2.0 / ACLK / A0 / OA0I0	General-purpose digital I/O pin / ACLK output / ADC10, analog input A0
4	P2.1 / TAINCLK / SMCLK / A1 / A0O	General-purpose digital I/O pin / ADC10, analog input A1 Timer_A, clock signal at INCLK, SMCLK signal output
5	P2.2 / TA0 / A2 / OA0I1	General-purpose digital I/O pin / ADC10, analog input A2 Timer_A, capture: CCI0B input/BSL receive, compare: OUT0 output
6	P2.3 / TA1 / A3 / VREF– / VeREF– / OA1I1 / OA1O	General-purpose digital I/O pin / Timer_A, capture: CCI1B input, compare: OUT1 output / ADC10, analog input A3 / negative reference voltage output/input
7	P2.4 / TA2 / A4 / VREF+ / VeREF+ / OA1I0	General-purpose digital I/O pin / Timer_A, compare: OUT2 output / ADC10, analog input A4 / positive reference voltage output/input
8	P4.3 / TB0 / A12 / OA0O	General-purpose digital I/O pin / ADC10 analog input A12 / Timer_B, capture: CCI0B input, compare: OUT0 output
9	P4.4 / TB1 / A13 / OA1O	General-purpose digital I/O pin / ADC10 analog input A13 / Timer_B, capture: CCI1B input, compare: OUT1 output
10	P4.5 / TB2 / A14 / OA0I3	General-purpose digital I/O pin / ADC10 analog input A14 / Timer_B, compare: OUT2 output
11	P4.6 / TBOUTH / A15 / OA1I3	General-purpose digital I/O pin / ADC10 analog input A15 / Timer_B, switch all TB0 to TB3 outputs to high impedance
12	GND	Ground reference
13	P2.6 / XIN (GDO0)	General-purpose digital I/O pin / Input terminal of crystal oscillator
14	P2.7 / XOUT (GDO2)	General-purpose digital I/O pin / Output terminal of crystal oscillator
15	P3.2 / UCB0SOMI / UCB0SCL	General-purpose digital I/O pin USCI_B0 slave out/master in when in SPI mode, SCL I2C clock in I2C mode
16	P3.3 / UCB0CLK / UCA0STE	General-purpose digital I/O pin USCI_B0 clock input/output / USCI_A0 slave transmit enable
17	P3.0 / UCB0STE / UCA0CLK / A5	General-purpose digital I/O pin / USCI_B0 slave transmit enable / USCI_A0 clock input/output / ADC10, analog input A5
18	P3.1 / UCB0SIMO / UCB0SDA	General-purpose digital I/O pin / USCI_B0 slave in/master out in SPI mode, SDA I2C data in I2C mode

Table 2. Battery Board Pinouts

Pin	Function	Description
1	P3.4 / UCA0TXD / UCA0SIMO	General-purpose digital I/O pin / USCI_A0 transmit data output in UART mode (UART communication from 2274 to PC), slave in/master out in SPI mode
2	GND	Ground reference
3	$\overline{\text{RST}}$ / SBWTDIO	Reset or nonmaskable interrupt input Spy-Bi-Wire test data input/output during programming and test
4	TEST / SBWTCK	Selects test mode for JTAG pins on Port1. The device protection fuse is connected to TEST. Spy-Bi-Wire test clock input during programming and test
5	VCC (3.6V)	Supply voltage
6	P3.5 / UCA0RXD / UCA0SOMI	General-purpose digital I/O pin / USCI_A0 receive data input in UART mode (UART communication from 2274 to PC), slave out/master in when in SPI mode

Pin description.

A.6. Programming code

Vibration main program

```
////////////////////////////// TEST VIBRATIONS ///////////////////////////////
/* The program send via Serial communication (RS-232) accelerometer values to
 * MATLAB, and then it works with the data.
 * Both programs MATLAB and Code Composer Studio take communication and should
 * take attention if there is some problem every time they send and receive
 * bytes*/
////////////////////////////// PRE-COMPILER //////////////////////////////////
#include <msp430x22x4.h>
#include "MPU6050.h"
#include "I2C.h"
#include "Com_UART.h"
#include "PWM.h"
////////////////////////////// FUNCTIONS DECLARATION /////////////////////////
void setup_Serial();
int get_Serial();
void send_To_Serial(long int);
void setupPWM_X();
void setupPWM_Y();
////////////////////////////////// VARIABLES /////////////////////////////////
int start=0,samples_=0,power_=0,ct=0;
long int lol=0;
////////////////////////////////// MAIN //////////////////////////////////////
int main(void)
{
      WDTCTL = WDTPW + WDTHOLD;   // Stop WatchDog, set a STOP when the CPU is
trap
      clock_1MHz();
      //BCSCTL3 |= LFXT1S_2;        // LFXT1 = VLO = 12KHz microprocessor work
frequency

      setup_Serial();             // Call setup Serial Port function
      setupMPU();                           // Call setup MPU6050
(accelerometer)
      setupPWM_X();                    // Call setup PWM
      setupPWM_Y();

      samples_ = 100;            //Number of samples using for the While loop
      get_Serial();                    //Waiting condition to start loop

      for(power_=0;power_<=100; power_=power_+ 4)
      {
//every time that the counter gets the sample's value
            TACCR1 = power_;
            TBCCR2 = power_;           // Modify PWM's width andconsequently
determine the motor DC current
            TACCR2 = power_;
            TBCCR1 = power_;

            ct=1;                             //Every time the While loop
finish the counter 'ct' take value 1,synchronism MATLAB
            while(ct<=samples_)
            {
                        send_To_Serial(average_Accel_X(0,1)); //Send to
MATLAB the accelerometer value
                  start = get_Serial(); //Check if the value arrived to
MATLAB or not
                  if (start == 130){}   //If the value didn't arrive,
```

```
buffer empty []
                        if (start == 131){}   //If the value take zero value
[0]
                        if (start == 133){}
                        if (start == 132){ct++;} //Value arrived correctly to
MATLAB, increase the counter (at same time MATLAB)
                }
        }
}
```

Pulse Width Modulation

```
#ifndef PWM_H_
#define PWM_H_

#include "MPU6050.h"
#include "Com_UART.h"

void setupPWM_X()                           //PWM X axis
{
  BCSCTL1 = CALBC1_8MHZ;                              //Set MCLK (8 MHz)
  DCOCTL = CALDCO_8MHZ;

  P2DIR |= BIT3 + BIT4;                     // P2.3 and P2.4 output
  P2SEL |= BIT3 + BIT4;                     // P2.3 and P2.4 TA1/2 otions

  TACTL = TASSEL_2 + ID_0 + MC_1;           // SMCLK, up mode
  TACCR0 = 200;                             // PWM Period
  TACCTL1 = OUTMOD_7;                       // TACCR1 reset/set MOTOR1
  TACCTL2 = OUTMOD_7;                       // TACCR2 reset/set MOTOR2
}

void setupPWM_Y()                                           //PWM X axis
{
  BCSCTL1 = CALBC1_8MHZ;
  DCOCTL = CALDCO_8MHZ;

  P4DIR |= BIT4 + BIT5;
  P4SEL |= BIT4 + BIT5;

  TBCTL = TBSSEL_2 + ID_0 + MC_1;
  TBCCR0 = 200;
  TBCCTL1 = OUTMOD_7;
  TBCCTL2 = OUTMOD_7;
}

#endif /*PWM_H_*/
```

I2C communication

```
#ifndef IC2_H_
#define IC2_H_

void Init_I2C_USCI(unsigned char addr);
unsigned char ByteRead_I2C_USCI(unsigned char address);
void ByteWrite_I2C_USCI(unsigned char address, unsigned char Data);

unsigned char TransmittedData, ReceivedData, WriteRegister;
int phase, WriteMode;

void Init_I2C_USCI(unsigned char addr)
{
       P3DIR |= 0x0f;
       P3SEL |= BIT1 + BIT2;                     // Assign I2C pins to USCI_B0
       UCB0CTL1 |= UCSWRST;                      // Enable SW reset
       UCB0CTL0 = UCMST+UCMODE_3+UCSYNC;         // I2C Master, synchronous mode
       UCB0CTL1 = UCSSEL_2+UCSWRST;              // Use SMCLK, keep SW reset
       UCB0BR0 = 40;                             // fSCL = SMCLK/40 = ~400kHz
       UCB0BR1 = 0;
       UCB0I2CSA = addr;                         // Set slave address
       UCB0CTL1 &= ~UCSWRST;                     // Clear SW reset, resume
operation
}
void ByteWrite_I2C_USCI(unsigned char Adress, unsigned char Value){
    while (UCB0CTL1 & UCTXSTP);                //check if STOP is sent
       WriteMode = 1;                          //important for the ISR procedure
       WriteRegister = Adress;
       TransmittedData = Value;
       phase = 0;                          //important for ISR procedure

       IE2 &= ~UCB0RXIE;                   //clear RX interrupt-enable...
       IE2 |= UCB0TXIE;                    //... and set TX interrupt-enable

    UCB0CTL1 |= UCTR + UCTXSTT;                //send START condition + address
+ Write
    __bis_SR_register(CPUOFF + GIE);           //go into a LPM and wait for ISR

       WriteMode = 0;
       return;
}

unsigned char ByteRead_I2C_USCI(unsigned char Register){
              TransmittedData = Register;
              WriteMode = 0;                           //important for ISR
procedure
              phase = 0;                           //important for ISR procedure
              IE2 &= ~UCB0RXIE;                    //clear RX interrupt-enable
              IE2 |= UCB0TXIE;                     //set TX interrupt-enable

           while (UCB0CTL1 & UCTXSTP);                 //ensure STOP condition
was sent
           UCB0CTL1 |= UCTR + UCTXSTT;                 //send START condition +
address + Write
           __bis_SR_register(CPUOFF + GIE);            //go into a LPM and wait
for ISR

           while (UCB0CTL1 & UCTXSTP);                 //ensure STOP was sent
              return ReceivedData;
}

#pragma vector = USCIAB0TX_VECTOR
```

```
__interrupt void USCIAB0TX_ISR(void)
{
	if(WriteMode)
	{
		if(phase == 0)
		{
			UCB0TXBUF = WriteRegister;		//put the register address
on the bus
			phase = 1;

		}
		else if (phase == 1)
		{
			phase = 2;
			UCB0TXBUF = TransmittedData;		//put data on the
bus

		}
		else if(phase == 2)
		{
			IFG2 &= ~(UCB0TXIFG + UCB0RXIFG); //disable RX and TX
interrupts
			UCB0CTL1 |= UCTXSTP;		//send STOP condition
			__bic_SR_register_on_exit(CPUOFF);	//exit LPM
		    WriteMode = 0;
		    phase = 0;
		}
	}
	else
	{
		//Read-mode
		if(phase == 0){
			UCB0TXBUF = TransmittedData;		//put register
address on the bus
			phase = 1;

	}
	else if (phase == 1)
		{
			IE2 &= ~UCB0TXIE;		//disable TX interrupts
			IE2 |= UCB0RXIE;		//enable RX interrupts

			UCB0CTL1 &= ~UCTR;		//clear UCTR bit (means
READ mode on next START condition)
		    UCB0CTL1 |= UCTXSTT;		//send START condition

		    while (UCB0CTL1 & UCTXSTT);		//wait for START to
be sent
		    UCB0CTL1 |= UCTXSTP;		//send STOP condition and
wait for the sensor-data
			phase = 2;

		}
		else if (phase == 2)
		{
			ReceivedData = UCB0RXBUF;		//read sensor-data from
the bus
			IFG2 &= ~(UCB0TXIFG + UCB0RXIFG); //disable interrupts
			__bic_SR_register_on_exit(CPUOFF);	//exit LPM

		}
```

```
        }
}
#endif /*I2C_H_*/
```

UART Communication

```
#ifndef COM_UART_H_
#define COM_UART_H_

int power = 0, count = 0, receive = 0;
int powerByte1,  powerByte2,  powerByte3,  powerByte4;
long int data, quotient, rest, send_negative, send;
int count_send, count_buffer;
int buffer[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int data_receive;

void setup_Serial()
{
  BCSCTL1 = CALBC1_1MHZ;                              // Set clock
  DCOCTL = CALDCO_1MHZ;

  P3SEL = BIT4 + BIT5;                                        // USCI_A0
transmit data output (2274 to PC)

  UCA0CTL1 |= UCSSEL_2;                       // CLK = SMCLK
  UCA0BR0 = 104;                              // 9600 bauds
  UCA0BR1 = 0;
  UCA0MCTL = UCBRS0;                          // Modulation
  UCA0CTL1 &= ~UCSWRST;                       // **Initialize USCI state
machine**
  IE2 &= ~UCB0RXIE;
  IE2 &= ~UCB0TXIE;                           // Enable USCI_A0 TX and RX
interrupt
}

int get_Serial()
{
              count_buffer = 0;
              while(count_buffer < 16) //16 position vector = 4 bytes
              {
                     while (!(IFG2&UCA0RXIFG));
                     receive = UCA0RXBUF;
                     buffer[count_buffer]=receive;
                     count_buffer++;
              }

           count_buffer = 0;
              while(count_buffer < 16)
           {
                     if (buffer[count_buffer] <= 63 && buffer[count_buffer] >=
48 && count == 0)
               {
                           powerByte1 = (buffer[count_buffer] - 48) * 4096;
                           count = 1;
               }

               if (buffer[count_buffer]<= 47 && buffer[count_buffer] >= 32 &&
count == 1 )
               {
                           powerByte2 = (buffer[count_buffer] - 32) * 256;
                           count =2;
               }

               if (buffer[count_buffer] <= 31 && buffer[count_buffer] >= 16 &&
count == 2 )
               {
                           powerByte3 = (buffer[count_buffer] - 16) * 16;
```

```
                        count =3;
                }
                if (buffer[count_buffer] <= 15 && buffer[count_buffer] >= 0  &&
count == 3 )
                {
                        powerByte4 = buffer[count_buffer];
                        count =4;
                        power = powerByte1 + powerByte2 + powerByte3 +
powerByte4;
                        break;
                }
                count_buffer++;
        }
        count = 0;
    return power;
}
////////////////////////////////////////////////////////////////////////////
void send_To_Serial(long int send)
{
        count_send = 0;
        while(count_send < 4)//Almenos 32 ya que MATLAB los lee de 12 en 12
         {
              if (send < 0)
              {
                    send_negative = -(send);

                       quotient = send_negative / 4096;
                          data = quotient + 112;
                          rest = send_negative -(4096 * quotient);
                          UCA0TXBUF = data;
                          while (!(IFG2&UCA0TXIFG));

                          quotient = rest / 256;
                          data = quotient + 96;
                          rest = rest -(256 * quotient);
                          UCA0TXBUF = data;
                          while (!(IFG2&UCA0TXIFG));

                          quotient = rest / 16;
                          data = quotient + 80;
                          rest = rest -(16 * quotient);
                          UCA0TXBUF = data;
                          while (!(IFG2&UCA0TXIFG));

                          data = rest + 64;
                          UCA0TXBUF = data;
                          while (!(IFG2&UCA0TXIFG));
              }

              if (send >= 0)
              {
                          quotient = send / 4096;
                          data = quotient+ 48;
                          rest = send -(4096 * quotient);
                          UCA0TXBUF = data;
                          while (!(IFG2&UCA0TXIFG));

                          quotient = rest / 256;
                          data = quotient + 32;
                          rest = rest -(256 * quotient);
                          UCA0TXBUF = data;
                          while (!(IFG2&UCA0TXIFG));
```

```
                            quotient = rest / 16;
                            data = quotient + 16;
                            rest = rest -(16 * quotient);
                            UCA0TXBUF = data;
                            while (!(IFG2&UCA0TXIFG));

                            data = rest;
                            UCA0TXBUF = data;
                            while (!(IFG2&UCA0TXIFG));
                }
                count_send++;
        }
}
#endif /*COM_UART_H_*/
```

MPU-6050 configuration and reading data

```
#ifndef MPU6050_H_
#define MPU6050_H_

#include "I2C.h"

#define MPU6050_ADDRESS    0x68
#define PWR_MGMT_1      0x6B
#define SIGNAL_PATH_RESET 0x68
#define CONFIG              0x1A
#define GYRO_CONFIG        0x1B
#define SMPLRT_DIV         0x19
#define TEMP_OUT_H      0x41
#define TEMP_OUT_L      0x42
#define GYRO_XOUT_H     0x43
#define GYRO_XOUT_L     0x44
#define GYRO_YOUT_H     0x45
#define GYRO_YOUT_L     0x46
#define GYRO_ZOUT_H     0x47
#define GYRO_ZOUT_L     0x48
#define ACCEL_CONFIG 0x1C
#define ACCEL_XOUT_H    0x3B
#define ACCEL_XOUT_L 0x3C
#define ACCEL_YOUT_H 0x3D
#define ACCEL_YOUT_L 0x3E
#define ACCEL_ZOUT_H 0x3F
#define ACCEL_ZOUT_L 0x40
#define FIFO_EN                   0x23
#define INT_STATUS         0x3A

short int average_Accel_X();
short int average_Accel_Y();
short int average_Accel_Z();
long int average_Gyro_X();
short int average_Gyro_Y();
short int average_Gyro_Z();
short int temp();

long int averageX=0, averageY=0,averageZ=0, averageTemp=0;
long int sum_i=0, SUM_i=0;
long int dataX[40];

long int  ACCEL_X_H=0,ACCEL_X_L=0, ACCEL_X=0;
long int ACCEL_X_16g=0, ACCEL_Y_16g=0, ACCEL_Z_16g=0;
long int  ACCEL_Y_H=0, ACCEL_Y_L=0, ACCEL_Y=0;
long int  ACCEL_Z_H=0,ACCEL_Z_L=0, ACCEL_Z=0;
short int TEMP_H=0,TEMP_L=0, TEMP=0;

long int  GYRO_X_H=0, GYRO_X_L=0, GYRO_X=0;
long int  GYRO_Y_H=0, GYRO_Y_L=0, GYRO_Y=0;
long int  GYRO_Z_H=0, GYRO_Z_L=0, GYRO_Z=0;

long int GYRO_X_16g=0, GYRO_Y_16g=0, GYRO_Z_16g=0;

short int before_value_max=0, before_value_min=0;

volatile int i=0,ii=0;
unsigned long int number_counts=40;
int average, now;

long int max=0, min=0;
```

```
long int max_absolute = 0, min_absolute = 0;
long int errorMax=0, errorMin=0, accel_ref=0;
long int error_X_H=0, error_X_L=0, error_Y_H=0, error_Y_L=0, error_Z_H=0,
error_Z_L=0;

void clock_1MHz();
void clock_16MHz();
long int max_func_X(long int);
long int min_func_X(long int);
long int max_func_Y(long int);
long int min_func_Y(long int);
long int max_func_Z(long int);
long int min_func_Z(long int);
long int error_max();
long int error_min();
////////////////////////////////MPU6050////
void setupMPU()
{
  void clock_1MHz();

  Init_I2C_USCI(MPU6050_ADDRESS);
  ByteWrite_I2C_USCI(PWR_MGMT_1,0x18);
  ByteWrite_I2C_USCI(SIGNAL_PATH_RESET,0x07);
  ByteWrite_I2C_USCI(CONFIG,0x06); //44KHz 0x03 // 21Hz 0x04
  ByteWrite_I2C_USCI(GYRO_CONFIG, 0x00); //+-250º/s
  ByteWrite_I2C_USCI(ACCEL_CONFIG,0x00);
  //0x00 (2g) //0x08 (4g) // 0x10 (8g) // 0x18 (16g)
  //ByteWrite_I2C_USCI(SMPLRT_DIV, 0x00);
  ByteWrite_I2C_USCI(TEMP_OUT_H,0x00);
  ByteWrite_I2C_USCI(TEMP_OUT_L,0x00);
  ByteWrite_I2C_USCI(GYRO_XOUT_H,0x00);
  ByteWrite_I2C_USCI(GYRO_XOUT_L,0x00);
  ByteWrite_I2C_USCI(GYRO_YOUT_H,0x00);
  ByteWrite_I2C_USCI(GYRO_YOUT_L,0x00);
  ByteWrite_I2C_USCI(GYRO_ZOUT_H,0x00);
  ByteWrite_I2C_USCI(GYRO_ZOUT_L,0x00);


  ByteWrite_I2C_USCI(ACCEL_XOUT_H,0x00);
  ByteWrite_I2C_USCI(ACCEL_XOUT_L,0x00);
  ByteWrite_I2C_USCI(ACCEL_YOUT_H,0x00);
  ByteWrite_I2C_USCI(ACCEL_YOUT_L,0x00);
  ByteWrite_I2C_USCI(ACCEL_ZOUT_H,0x00);
  ByteWrite_I2C_USCI(ACCEL_ZOUT_L,0x00);

  ByteWrite_I2C_USCI(FIFO_EN, 0x00);
  ByteWrite_I2C_USCI(INT_STATUS, 0x01);
}

short int average_Accel_X(int average, int now)
{
    clock_1MHz();

       if(average == 1 && now == 0)
       {
              SUM_i=0;
              for(ii=0;ii<=9;ii++)
              {
                     sum_i=0;
                     for(i=0;i<=39;i++)
                     {
```

```
                        ACCEL_X_H = ByteRead_I2C_USCI(ACCEL_XOUT_H);
                        ACCEL_X_L = ByteRead_I2C_USCI(ACCEL_XOUT_L);
                        ACCEL_X = (ACCEL_X_H << 8) | ACCEL_X_L;

                        ACCEL_X_16g = (short int) ACCEL_X;
                        dataX[i] = (short int) ACCEL_X;

                        if(i==0 && ii==0)
                        {
                               before_value_max = ACCEL_X_16g;
                               before_value_min = ACCEL_X_16g;
                        }
                        if(before_value_max < ACCEL_X_16g)
                        {
                               before_value_max = ACCEL_X_16g;
                        }
                        if(before_value_min > ACCEL_X_16g)
                        {
                               before_value_min = ACCEL_X_16g;
                        }

                        sum_i = (long int) sum_i + (long int)dataX[i];
                    }
                       max_absolute = before_value_max;
                       min_absolute = before_value_min;

                       if(max_absolute < before_value_max)
                       {
                                max_absolute = before_value_max;
                       }
                       if(min_absolute > before_value_min)
                       {
                                min_absolute = before_value_min;
                       }

                           averageX = (long int) sum_i/(long int)number_counts;
                     SUM_i = (long int)SUM_i +(long int)averageX;
              }

              error_X_H = max_absolute;
              error_X_L = min_absolute;

              averageX = (long int) (SUM_i)/ (long int)10;
       }
       //////////////////
       if (average==0 && now==1)
       {
              ACCEL_X_H = ByteRead_I2C_USCI(ACCEL_XOUT_H);
              ACCEL_X_L = ByteRead_I2C_USCI(ACCEL_XOUT_L);
              ACCEL_X = (ACCEL_X_H << 8) | ACCEL_X_L;
              averageX = (short int)ACCEL_X;

       }
    return averageX;
}

short int average_Accel_Y(int average, int now)
{
       clock_1MHz();

       if(average == 1 && now == 0)
       {
```

```
		SUM_i=0;
		for(ii=0;ii<=9;ii++)
		{	sum_i=0;

			for(i=0, sum_i=0; i<=39; i++)
			{
				ACCEL_Y_H = ByteRead_I2C_USCI(ACCEL_YOUT_H);
				ACCEL_Y_L = ByteRead_I2C_USCI(ACCEL_YOUT_L);
				ACCEL_Y = (ACCEL_Y_H << 8) | ACCEL_Y_L;

			 ACCEL_Y_16g = (short int) ACCEL_Y;
				dataX[i] = (short int) ACCEL_Y;

				if(i==0 && ii==0)
				{
					before_value_max = ACCEL_Y_16g;
					before_value_min = ACCEL_Y_16g;
				}
				if(before_value_max < ACCEL_Y_16g)
				{
					before_value_max = ACCEL_Y_16g;
				}
				if(before_value_min > ACCEL_Y_16g)
				{
					before_value_min = ACCEL_Y_16g;
				}

				sum_i = (long int) sum_i + (long int)dataX[i];
			}
			   max_absolute = before_value_max;
			   min_absolute = before_value_min;

			   if(max_absolute < before_value_max)
			   {
					max_absolute = before_value_max;
			   }
			   if(min_absolute > before_value_min)
			   {
					min_absolute = before_value_min;
			   }

				averageY = (long int) sum_i/(long int)number_counts;
			 SUM_i = (long int)SUM_i +(long int)averageX;
		}
		error_Y_H = max_absolute;
		error_Y_L = min_absolute;

		averageX = (long int) (SUM_i)/ (long int)10;
	}
	//////////////////
	if (average==0 && now==1)
	{
		ACCEL_Y_H = ByteRead_I2C_USCI(ACCEL_YOUT_H);
		ACCEL_Y_L = ByteRead_I2C_USCI(ACCEL_YOUT_L);
		ACCEL_Y = (ACCEL_Y_H << 8) | ACCEL_Y_L;
		averageY = (short int) ACCEL_Y;

	}

	return averageY;
}
```

```
short int average_Accel_Z(int average, int now)
{
	clock_1MHz();

	if(average == 1 && now == 0)
	{
		SUM_i=0;
		for(ii=0;ii<=9;ii++)
		{	sum_i=0;

			for(i=0, sum_i=0; i<=39; i++)
			{
				ACCEL_Z_H = ByteRead_I2C_USCI(ACCEL_ZOUT_H);
				ACCEL_Z_L = ByteRead_I2C_USCI(ACCEL_ZOUT_L);
				ACCEL_Z = (ACCEL_Z_H << 8) | ACCEL_Z_L;

				ACCEL_Z_16g =(short int) ACCEL_Z;
				dataX[i] =(short int) ACCEL_Z;

			       if(i==0 && ii==0)
			       {
					before_value_max = ACCEL_Z_16g;
					before_value_min = ACCEL_Z_16g;
			       }
			       if(before_value_max < ACCEL_Z_16g)
			       {
					before_value_max = ACCEL_Z_16g;
			       }
			       if(before_value_min > ACCEL_Z_16g)
			       {
					before_value_min = ACCEL_Z_16g;
			       }

			       sum_i = (long int) sum_i + (long int)dataX[i];
			}
		   max_absolute = before_value_max;
			min_absolute = before_value_min;

			if(max_absolute < before_value_max)
			{
				max_absolute = before_value_max;
			}
			if(min_absolute > before_value_min)
			{
				min_absolute = before_value_min;
			}

			    averageZ = (long int) sum_i/(long int)number_counts;
			SUM_i = (long int)SUM_i +(long int)averageZ;
		}
		error_Z_H = max_absolute;
		error_Z_L = min_absolute;

		averageZ = (long int) (SUM_i)/ (long int)10;
	}
	//////////////
	if (average==0 && now==1)
	{
		ACCEL_Z_H = ByteRead_I2C_USCI(ACCEL_ZOUT_H);
		ACCEL_Z_L = ByteRead_I2C_USCI(ACCEL_ZOUT_L);
		ACCEL_Z = (ACCEL_Z_H << 8) | ACCEL_Z_L;
		averageZ = (short int) ACCEL_Z;
```

```
	}

	return averageZ;
}

short int temp(int average, int now)
{
	clock_1MHz();

	if(average == 1 && now == 0)
	{SUM_i=0;
		for(ii=0;ii<=9;ii++)
		{	sum_i=0;

			for(i=0, sum_i=0; i<=39; i++)
			{
		  TEMP_H = ByteRead_I2C_USCI(TEMP_OUT_H);
		  TEMP_L = ByteRead_I2C_USCI(TEMP_OUT_L);
		  TEMP = (TEMP_OUT_H << 8) + TEMP_OUT_L;

		  dataX[i] = (short int) TEMP;
		  sum_i = (long int) sum_i + (long int)dataX[i];
			}
				averageX = (long int) sum_i/(long int)number_counts;
			 SUM_i = (long int)SUM_i +(long int)averageX;
		}

		averageTemp = (long int) (SUM_i)/ (long int)10;
	}
	if (average==0 && now==1)
	{
		TEMP_H = ByteRead_I2C_USCI(TEMP_OUT_H);
		TEMP_L = ByteRead_I2C_USCI(TEMP_OUT_L);
		TEMP = (TEMP_OUT_H << 8) | TEMP_OUT_L;

		averageTemp =(short int)TEMP;
		//Temperature in degrees C = (TEMP_OUT Register Value)/340 + 36.53
		averageTemp = ((short int)averageTemp/ (short int)430)+36.35;
		averageTemp =(short int) averageTemp - 32;
		averageTemp =(short int) averageTemp / 1.8;
	}

	return averageTemp;
}
////////////////////////////////GYROSCOPES/
long int average_Gyro_X(int average, int now)
{
    if(average == 1 && now == 0)
	{
		SUM_i=0;
		for(ii=0;ii<=9;ii++)
		{
			sum_i=0;
			for(i=0;i<=39;i++)
			{

				GYRO_X_H = ByteRead_I2C_USCI(GYRO_XOUT_H);
				GYRO_X_L = ByteRead_I2C_USCI(GYRO_XOUT_L);
				GYRO_X = (GYRO_X_H << 8) | GYRO_X_L;

				GYRO_X_16g = (short int) GYRO_X + 32768;
```

```
                        dataX[i] = (short int) GYRO_X + 32768;

                        if(i==0 && ii==0)
                        {
                              before_value_max = GYRO_X_16g;
                              before_value_min = GYRO_X_16g;
                        }
                        if(before_value_max < GYRO_X_16g)
                        {
                              before_value_max = GYRO_X_16g;
                        }
                        if(before_value_min > GYRO_X_16g)
                        {
                              before_value_min = GYRO_X_16g;
                        }

                        sum_i = (long int) sum_i + (long int)dataX[i];
                  }
                     max_absolute = before_value_max;
                     min_absolute = before_value_min;

                     if(max_absolute < before_value_max)
                     {
                              max_absolute = before_value_max;
                     }
                     if(min_absolute > before_value_min)
                     {
                              min_absolute = before_value_min;
                     }

                        averageX = (long int) sum_i/(long int)number_counts;
                   SUM_i = (long int)SUM_i +(long int)averageX;
            }

            error_X_H =  max_absolute;
            error_X_L =  min_absolute;

            averageX = (long int) (SUM_i)/ (long int)10;
      }
      ////////////////////
      if (average==0 && now==1)
      {
            GYRO_X_H = ByteRead_I2C_USCI(GYRO_XOUT_H);
            GYRO_X_L = ByteRead_I2C_USCI(GYRO_XOUT_L);
            GYRO_X = (GYRO_X_H << 8) | GYRO_X_L;
            averageX = (short int)GYRO_X;
            averageX = averageX - (short int)GYRO_X;
      }
   return averageX;
}
//////////////////////
short int average_Gyro_Y(int average, int now)
{
   if(average == 1 && now == 0)
      {
            SUM_i=0;
            for(ii=0;ii<=9;ii++)
            {
                  sum_i=0;
                  for(i=0;i<=39;i++)
                  {
```

```
                        GYRO_Y_H = ByteRead_I2C_USCI(GYRO_YOUT_H);
                        GYRO_Y_L = ByteRead_I2C_USCI(GYRO_YOUT_L);
                        GYRO_Y = (GYRO_Y_H << 8) | GYRO_Y_L;

                        GYRO_Y_16g = (short int) GYRO_Y + 32768;
                        dataX[i] = (short int) GYRO_Y + 32768;

                        if(i==0 && ii==0)
                        {
                              before_value_max = GYRO_Y_16g;
                              before_value_min = GYRO_Y_16g;
                        }
                        if(before_value_max < GYRO_Y_16g)
                        {
                              before_value_max = GYRO_Y_16g;
                        }
                        if(before_value_min > GYRO_Y_16g)
                        {
                              before_value_min = GYRO_Y_16g;
                        }

                        sum_i = (long int) sum_i + (long int)dataX[i];
                  }
                     max_absolute = before_value_max;
                     min_absolute = before_value_min;

                     if(max_absolute < before_value_max)
                     {
                              max_absolute = before_value_max;
                     }
                     if(min_absolute > before_value_min)
                     {
                              min_absolute = before_value_min;
                     }

                        averageX = (long int) sum_i/(long int)number_counts;
                   SUM_i = (long int)SUM_i +(long int)averageX;
            }

            error_X_H =  max_absolute;
            error_X_L =  min_absolute;

            averageX = (long int) (SUM_i)/ (long int)10;
      }
      ////////////////////
      if (average==0 && now==1)
      {
            GYRO_Y_H = ByteRead_I2C_USCI(GYRO_YOUT_H);
            GYRO_Y_L = ByteRead_I2C_USCI(GYRO_YOUT_L);
            GYRO_Y = (GYRO_Y_H << 8) | GYRO_Y_L;
            averageX = (short int)GYRO_Y;
            //averageX = averageX + 32768;
      }
    return averageX;
}
//////////////////////////
short int average_Gyro_Z(int average, int now)
{
    if(average == 1 && now == 0)
      {
            SUM_i=0;
            for(ii=0;ii<=9;ii++)
```

```
            {
                  sum_i=0;
                  for(i=0;i<=39;i++)
                  {

                        GYRO_Z_H = ByteRead_I2C_USCI(GYRO_ZOUT_H);
                        GYRO_Z_L = ByteRead_I2C_USCI(GYRO_ZOUT_L);
                        GYRO_Z = (GYRO_Z_H << 8) | GYRO_Z_L;

                        //GYRO_X_16g = GYRO_X + 32768;
                        GYRO_Z_16g = (short int) GYRO_Z + 32768;
                        dataX[i] = (short int) GYRO_Z + 32768;

                        if(i==0 && ii==0)
                        {
                              before_value_max = GYRO_Z_16g;
                              before_value_min = GYRO_Z_16g;
                        }
                        if(before_value_max < GYRO_Z_16g)
                        {
                              before_value_max = GYRO_Z_16g;
                        }
                        if(before_value_min > GYRO_Z_16g)
                        {
                            before_value_min = GYRO_Z_16g;
                        }

                        sum_i = (long int) sum_i + (long int)dataX[i];
                  }
                     max_absolute = before_value_max;
                     min_absolute = before_value_min;

                     if(max_absolute < before_value_max)
                     {
                              max_absolute = before_value_max;
                     }
                     if(min_absolute > before_value_min)
                     {
                              min_absolute = before_value_min;
                     }

                         averageX = (long int) sum_i/(long int)number_counts;
                   SUM_i = (long int)SUM_i +(long int)averageX;
            }

            error_X_H =  max_absolute;
            error_X_L =  min_absolute;

            averageX = (long int) (SUM_i)/ (long int)10;
      }
      ////////////////////
      if (average==0 && now==1)
      {
            GYRO_Z_H = ByteRead_I2C_USCI(GYRO_ZOUT_H);
            GYRO_Z_L = ByteRead_I2C_USCI(GYRO_ZOUT_L);
            GYRO_Z = (GYRO_Z_H << 8) | GYRO_Z_L;
            averageX = (short int)GYRO_Z;
            //averageX = averageX + 32768;
      }
    return averageX;
}
```

```
////////////
long int error_Accel_Max_X()
{
       errorMax = (long int)error_X_H-(long int)averageX;
       return errorMax;

}
long int error_Accel_Min_X()
{
    errorMin = (long int)averageX-(long int)error_X_L;
    return errorMin;
}
long int error_Accel_Max_Y()
{
       errorMax = (long int)error_Y_H-(long int)averageY;
       return errorMax;

}
long int error_Accel_Min_Y()
{
    errorMin = (long int)averageY-(long int)error_Y_L;
    return errorMin;
}
long int error_Accel_Max_Z()
{
       errorMax = (long int)error_Z_H-(long int)averageZ;
       return errorMax;

}
long int error_Accel_Min_Z()
{
    errorMin = (long int)averageZ-(long int)error_Z_L;
    return errorMin;
}
//////////////////GYRO///
long int error_Gyro_Max_X()
{
       errorMax = (long int)error_X_H-(long int)averageX;
       return errorMax;

}
long int error_Gyro_Min_X()
{
    errorMin = (long int)averageX-(long int)error_X_L;
    return errorMin;
}
long int error_Gyro_Max_Y()
{
       errorMax = (long int)error_Y_H-(long int)averageY;
       return errorMax;

}
long int error_Gyro_Min_Y()
{
    errorMin = (long int)averageY-(long int)error_Y_L;
    return errorMin;
}
long int error_Gyro_Max_Z()
{
       errorMax = (long int)error_Z_H-(long int)averageZ;
       return errorMax;
```

```
}
long int error_Gyro_Min_Z()
{
    errorMin = (long int)averageZ-(long int)error_Z_L;
    return errorMin;
}
//////////////TIMER///
void clock_1MHz()
{
       BCSCTL1 = CALBC1_1MHZ;
    DCOCTL = CALDCO_1MHZ;
    return;
}
void clock_8MHz()
{
       BCSCTL1 = CALBC1_8MHZ;
    DCOCTL = CALDCO_8MHZ;
    return;
}
void clock_12MHz()
{
       BCSCTL1 = CALBC1_12MHZ;
    DCOCTL = CALDCO_12MHZ;
    return;
}
void clock_16MHz()
{
       BCSCTL1 = CALBC1_16MHZ;
    DCOCTL = CALDCO_16MHZ;
    return;
}

#endif /*MPU6050_H_*/
```

Main Hover flight MAV

```
#include <msp430x22x4.h>
#include <math.h>
#include "MPU6050.h"
#include "I2C.h"
#include "Com_UART.h"
#include "PWM.h"
#include "Kalman.h"

long int av_accel_X=0, av_accel_Y=0,av_accel_Z=0, av_accel_X_max_error=0;
long int av_accel_X_min_error=0, av_accel_Y_max_error=0,
av_accel_Y_min_error=0;
long int av_accel_Z_max_error=0, av_accel_Z_min_error=0;
long int av_gyro_X=0, av_gyro_X_max_error=0, av_gyro_X_min_error =0;
long int av_gyro_Y=0, av_gyro_Y_max_error=0, av_gyro_Y_min_error =0;
long int av_gyro_Z=0, av_gyro_Z_max_error=0, av_gyro_Z_min_error =0;
short int av_Temp=0;

int data_ADC = 0;
int axisX=0, axisY=0;
int  K_x = 0;
unsigned short int power_1=0;
int samples;

void setup_Serial();
int get_Serial();
void send_To_Serial(long int);
int ADC();
int test_vibrations();

int Kalman_accel_x();
int Kalman_accel_y();
void setupPWM_X();
void setupPWM_Y();
void rotational_speed();

void motor1(long int, long int, int, long int, long int, long int, long int,
int, int);

long int valorX_1=0, valorX_2=0, angulo=0, rotationalspeed=0;

short int a1=0, b1=0, c1=0;

int start=0,samples_=0,columns_=0,power_=0,ct=0;

int main(void)
{
       WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

       clock_1MHz();
       setup_Serial();
       setupMPU();
       setupPWM_X();
       setupPWM_Y();

  //av_accel_X = average_Accel_X(0,1);
  av_accel_X = average_Accel_X(1,0);
  av_accel_X_max_error = error_Accel_Max_X();
  av_accel_X_min_error = error_Accel_Min_X();

  //av_accel_Y = average_Accel_Y(0,1);
  av_accel_Y = average_Accel_Y(1,0);
```

```
  av_accel_Y_max_error = error_Accel_Max_Y();
  av_accel_Y_min_error = error_Accel_Min_Y();

  //av_accel_Z = average_Accel_Z(0,1);
  av_accel_Z = average_Accel_Z(1,0);
  av_accel_Z_max_error = error_Accel_Max_Z();
  av_accel_Z_min_error = error_Accel_Min_Z();

  for(;;)
  {
        power_1 = get_Serial();
        power_1 =((unsigned long int) 125* (unsigned long int)power_1)
/(unsigned long int) 1024 ;
        counter = 0;
        do{
                    motor1(av_accel_X, av_accel_Y, power_1,
av_accel_X_max_error, av_accel_X_min_error, av_accel_Y_max_error,
av_accel_Y_min_error, axisX, axisY); //axisX (1,0)

     }while(!(a >= (av_accel_X - 1000) && a <=(av_accel_X + 1000) && b >=
(av_accel_Y - 1000) && b <=(av_accel_Y + 1000)));

  }

}
/*
int ADC()
{
  ADC10CTL0 = ADC10SHT_3 + ADC10ON; // ADC10ON, interrupt enabled
  ADC10AE0 |= 0x01;                          // P2.0 ADC option select

  ADC10CTL0 |= ENC + ADC10SC;  //Enable data conversion + start sample and
conversion
  data_ADC = ADC10MEM;

  return data_ADC;
}
*/
```

PWM DYNAMICS

```
#ifndef PWM_H_
#define PWM_H_

#include "MPU6050.h"
#include "Com_UART.h"

long int a=0, b=0, c=0, sum_a=0;
int potencia_1_X=0,  potencia_2_X=0;
int counter= 0, count;
int axisX, axisY;
short int av_a;
short int vibration_a, vibration_b;
int error_max_x,error_min_x;
long int averageX_max = 0, averageX_min = 0;
long int averageY_max = 0, averageY_min = 0;
//////////////////////////////////////////////
void setupPWM_X()
{
  BCSCTL1 = CALBC1_8MHZ;
  DCOCTL = CALDCO_8MHZ;

  P2DIR |= BIT3 + BIT4;    // P2.3 and P2.4 output
  P2SEL |= BIT3 + BIT4;    // P2.3 and P2.4 TA1/2 otions

  TACTL = TASSEL_2 + ID_0 + MC_1;// SMCLK, up mode BIT4
  TACCR0 = 200;               // PWM Period
  TACCTL1 = OUTMOD_7;         // TACCR1 reset/set MOTOR1
  TACCTL2 = OUTMOD_7;         // TACCR2 reset/set MOTOR2
}

void setupPWM_Y()
{
  BCSCTL1 = CALBC1_8MHZ;
  DCOCTL = CALDCO_8MHZ;

  P4DIR |= BIT4 + BIT5;                          // P1.2 and P1.3 output
  P4SEL |= BIT4 + BIT5;

  TBCTL = TBSSEL_2 + ID_0 + MC_1;
  TBCCR0 = 200;                          // PWM Period (1Khz 125)
  TBCCTL1 = OUTMOD_7;                        // TACCR1 reset/set
  TBCCTL2 = OUTMOD_7;                        // TACCR2 reset/set BIT3
}

void motor1(long int averageX, long int averageY, int power, long int
av_X_max_error,
       long int av_X_min_error,long int av_Y_max_error, long int
av_Y_min_error, int axisX, int axisY)
{
  if(counter == 0)
  {
       potencia_1_X = power;
       potencia_2_X = potencia_1_X;
    counter = 1;
  }
  if( a >= (averageX_min) && a <= (averageX_max))
  {
             TACCR1 = potencia_1_X;
             TBCCR2 = potencia_1_X;
           TACCR2 = potencia_1_X;
             TBCCR1 = potencia_1_X;
```

```
  }
//////// X AXIS
        while((a < (averageX_min)) || (a > (averageX_max)))
        {

                        if(a < (averageX_min))// && potencia_1_X <=50)
                        {
                           potencia_1_X = 30; //potencia_1_X + 1;

                           TACCR1 = potencia_1_X; //Motor Negro 0
                           TBCCR2 = potencia_1_X; //Motor Negro 1
                        }
                        if(a < (averageX_min))// && potencia_2_X >=1)
                        {
                           potencia_2_X = 0; //potencia_2_X - 1;
                           TACCR2 = potencia_2_X; //Motor Blanco 1
                           TBCCR1 = potencia_2_X; //Motor Blanco 0
                        }
                        if(a > (averageX_max))//&& potencia_1_X >=1)
                        {
                        potencia_1_X = 0;//potencia_1_X - 1;
                        TACCR1 = potencia_1_X; //Motor Negro 0
                        TBCCR2 = potencia_1_X; //Motor Negro 1
                        }
                        if(a > (averageX_max))//&& potencia_2_X <=50)
                        {
                        potencia_2_X =30;// potencia_2_X + 1;
                        TACCR2 = potencia_2_X; //Motor Blanco 1
                           TBCCR1 = potencia_2_X; //Motor Blanco 0
                        }
                        break;
        }
  }

//////////////// Y AXIS (BLANCO-NEGRO / NEGRO-BLANCO)////

        while((b < (averageY_min)) || (b > (averageY_max)))
        {
               if(b < (averageY_min))//&& potencia_1_X <=50)
               {
                                          potencia_1_X =30;// potencia_1_X +
1;
                                          TBCCR1 = potencia_1_X; //Motor
Blanco 1
                                          TBCCR2 = potencia_1_X; //Motor
Negro 1
               }
               if(b < (averageY_min))// && potencia_2_X >=1)
               {
                                          potencia_2_X =0;//  potencia_2_X -
1;
                                          TACCR1 = potencia_2_X;  //Motor
Negro 0
                                          TACCR2 = potencia_2_X;  //Motor
Blanco 0
               }
               if(b > (averageY_max))// && potencia_1_X >=1)
               {
                                       potencia_1_X = 30;// potencia_1_X - 1;
                                       TACCR1 = potencia_1_X; //Motor Negro 0
                                       TACCR2 = potencia_1_X; //Motor Blanco
0
```

```
                }
                if(b > (averageY_max))//&& potencia_2_X <=50)
                {
                                        potencia_2_X =0;// potencia_2_X + 1;
                                        TBCCR1 = potencia_2_X;  //Motor Blanco
1
                                           TBCCR2 = potencia_2_X;  //Motor
Negro 1
                }
                break;
        }
  }
#endif /*PWM_H_*/
```

MATLAB MAIN

```
function Project()

%This function takes a 1000 accelerometer values given by the
%microcontroller. The power increased every loop 'for' cycle after
the 1000
%accelerometer values.
%Also, it has a little protocol for not losing synchoronous between
both
%loop 'For' cycles MATLAB and microncontroller (ez430-RF2500(T.I.))

%%%%%%%%%%%%%%%%%%%%%%%% DECLARATION VARIABLES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
samples = 100; %Samples that the function takes from microcontroller
row = 0; %Determine the row increase of accelerometer matrix data.
ct = 1; % While loop counter
power=0;
%%%%%%%%%%%%%%%%%%%%%%% SERIAL (RS-232) COMMUNICATION
%%%%%%%%%%%%%%%%%%%%%
s = serial('COM3'); %Create an 's' object to open the Port COM5
(micro)
set(s,'BaudRate',9600); %Set Baudrate object 's'
fopen(s); %Open Port to initiate communication

%%%%%%%%%%%%%%%%%%%%%%%%%%%%% TAKE VALUES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Serial_send(s,0); %Send zero to start MSP430 loop
while power<=90 %For loop, which increases the PWM's width every time
                  %the 1000 values have been taken.
     %Serial send(s,power);
     row = row + 1; %Row increment when the 1000 values have been
taken
     ct = 1;
     while(ct<=samples) %While loop 1000 values

           while s.BytesAvailable >= 16 %If bytes available to read
where less
                                        %than 16 the program couldn't
read data
            break;
           end
           error = Serial_receive(s);%Call read bytes function
           if error == 0 %buffer empty,cannot count cycle and take
another value
               Serial_send(s,130); %Send error this to
microcontroller
           end
           if error == 1 %buffer vector is zero, cannot count the
cycle
               Serial_send(s,131);%Send this error to microcontroller
           end
           if error == 2 %buffer vector is one, cannot count the
cycle
               Serial_send(s,133);%Send this error to microcontroller
           end
           if error ~= 1
           if error ~= 0
           if error ~= 2
                 matrix(row,ct)= error; %Matrix where the program
keeps the values
                 Serial_send(s,132); %Send arrival data check to
```

```
'micro'
                ct = ct + 1; %Increased the counter
                %save('matrix.mat','matrix'); %Save matrix's values
to work after with them
           end
           end
           end
           %if error ~= 0
            %   if error ~= 1
             %    matrix(row,ct)= error; %Matrix where the program
keeps the values
              %   Serial_send(s,132); %Send arrival data check to
'micro'
              %   ct = ct + 1; %Increased the counter
              %   save('matrix.mat','matrix'); %Save matrix's values
to work after with them
              % end
         % end
     end
     power = power + 4;
end
save('matrix.mat','matrix'); %Save matrix's values to work after with
them
%%%%%%%%%%%%%%%%%%%%%% SERIAL (RS-232) COMMUNICATION
%%%%%%%%%%%%%%%%%%%%
fclose(s) %Close Serial Port COM5
delete(s) %Delete object 's'
end
```

MATLAB SERIAL SEND

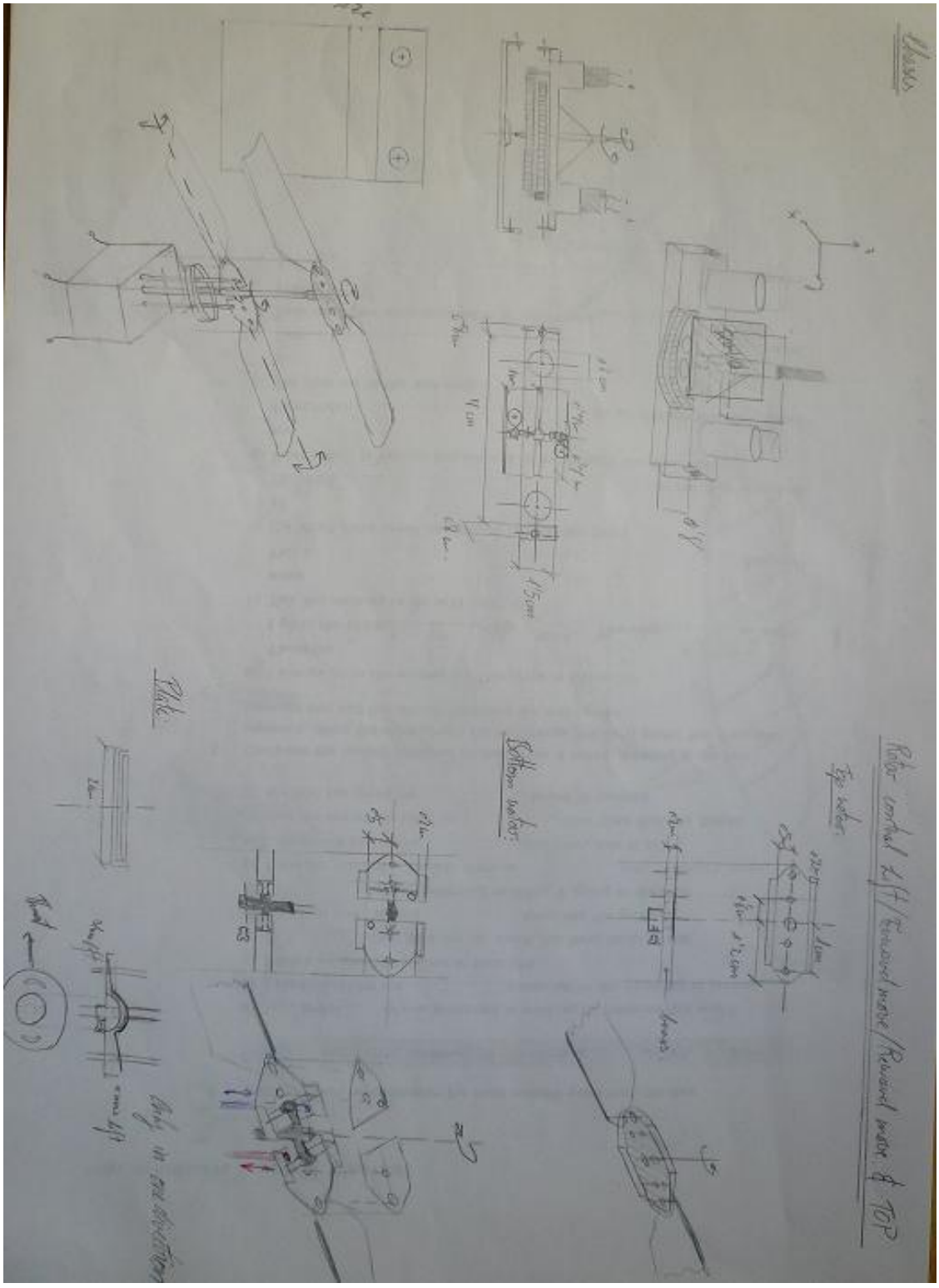
```
function Serial_send(s,send)
%The function takes the error value to check if the data from MSP430
arrive
%correctly.
%As in Serial_receive send every data shared in 4 bytes.

send_count = 0;

    while(send_count<4) %Send every check 32 times

            quotient = floor(send/4096);
            data = quotient + 48;
            rest =floor(send -(4096 * quotient));
            fwrite(s, data, 'uint8');

            quotient = floor(rest/ 256); %floor round float value
            data = quotient + 32;
            rest = floor(rest -(256 * quotient));
            fwrite(s, data, 'uint8');

            quotient =floor(rest/16);
            data = quotient + 16;
            rest = floor(rest -(16 * quotient));
            fwrite(s, data, 'uint8');

            data = floor(rest);
            fwrite(s, data, 'uint8');

            send_count = send_count + 1;
    end
end
```

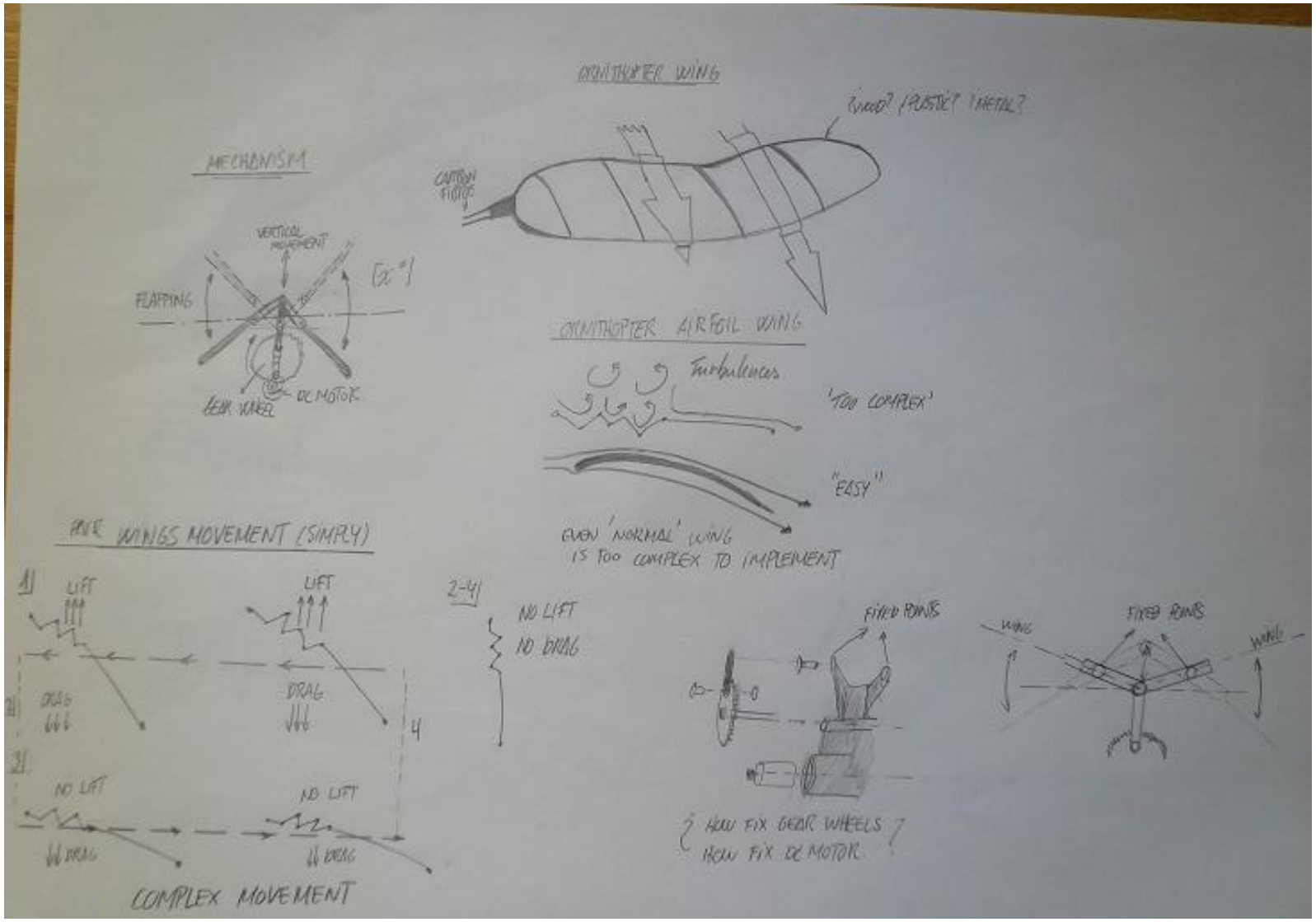
MATLAB MATRIX DATA (INCOMPLETE)

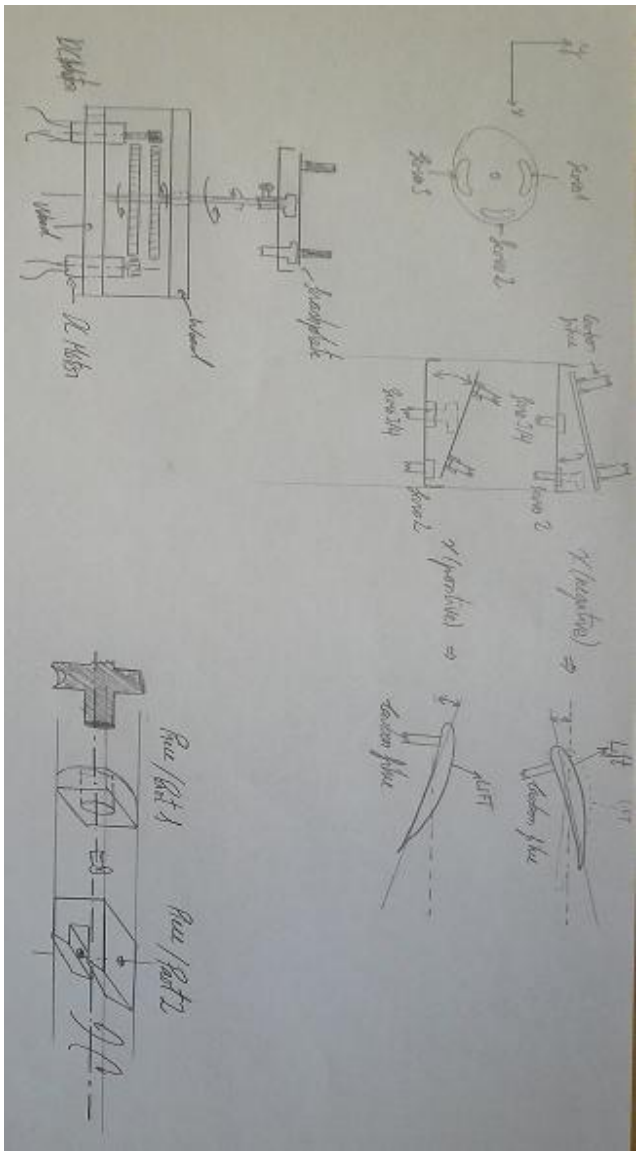
```
function Matrix_data()
%This function will work with matrix's vibrationdata

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VARIABLES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
samples = 1000;
val_power = 100;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% LOAD MATRIX
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load('matrix.mat','matrix');
matrix_n = 40000 + matrix; %Sum 400000 values to avoid negative
numbers

%%%%%%%%%%%%%%%%%%%%%%%%%%%% ACCELEROMETER DATA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for x=1:1:val_power
    for y=1:1:samples
         new_value = matrix_n(x,y);
         if y == 1
            min_val = new_value;
            max_val = new_value;
         end
         if max_val < new_value
            max_val = new_value;
         end
         if min_val > new_value
            min_val = new_value;
         end
    end
    max(x,1) = max_val; %Maximum values in each row (power)
    min(x,1) = min_val; %Minimum values in each row (power)
end
[common] = mode(matrix_n,2); %The most repeated value in each power
average = int32(mean(matrix_n,2)); %Average value data each row
(power)

%%%%%%%%%%%%%%%%%%%%%%% INCREMENT BETWEEN VALUES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Amax_common = max - common;%How change the differences
Acommon_min = common - min;
Amax_average = max - average;
Aaverage_common = average - common;

%%%%%%%%%%%%%%%%%%%%%%%%%%%% PROBABILITY
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
matrix_vert = matrix'; %Matrix change row for columns and vice-versa

for n=1:1:val_power
    for count=1:1:samples
        vector_matrix(count,1) = matrix(n,count);%Take matrix column
value to a vector
    end
        mu_sigma = fitdist(vector_matrix,'Normal');%Needs a numeric
column vector
        Amu(n) = mu_sigma.mu;%Mu'svalues NA~(0,1)
        Asigma(n) = mu_sigma.sigma%Sigma's values NA~(0,1)
end
%%%%%%%%%%%%%%%%%%%%%%%% CONFIDENCE INTERVAL
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% 95%
for n=1:1:val_power
    interval_conf_95(n) = norminv([0.025 0.975],Amu(n),Asigma(n));
end
% 90%
for n=1:1:val_power
    interval_conf_90(n) = norminv([0.05 0.95],Amu(n),Asigma(n));
end
% 85%
for n=1:1:val_power
    interval_conf_85(n) = norminv([0.075 0.925],Amu(n),Asigma(n));
end
% 80%
for n=1:1:val_power
    interval_conf_80(n) = norminv([0.1 0.9],Amu(n),Asigma(n));
end









end
```

A.7. Sketches




ORNITHOPTER WING
(WOOD? / PLASTIC? METAL?)
MECHANISM
VERTICAL MOVEMENT
FLAPPING
REAR WHEEL
DC MOTOR
ORNITHOPTER AIRFOIL WING
Turbulences
'TOO COMPLEX'
"EASY"
EVEN 'NORMAL' WING IS TOO COMPLEX TO IMPLEMENT
FIVE WINGS MOVEMENT (SIMPLY)
1)
LIFT
LIFT
DRAG
DRAG
2)
NO LIFT
NO LIFT
2-4)
NO LIFT
NO DRAG
COMPLEX MOVEMENT
FIXED PARTS
? HOW FIX REAR WHEELS
HOW FIX DC MOTOR ?
WING
FIXED PARTS
WING


Rear Left
Rear Right