République Tunisienne
Ministère de l'Enseignement Supérieur,
de la Recherche Scientifique
Carthage University
National Engineering School of Carthage

End Year Project

Specialty : Mechatronics engineering

Autonomous drone

Realized by **: BEN SAID Alaeddine**

Supervisor : **Mr MAGHRAOUI Mohamed Nabil**

Academic year : 2019/2020

Acknowledgement

To my mother Raja ,

To my father Seifeddine

To all those who have contributed from near or far , by their advice, suggestions and encouragement to the realization of this work.

Table of contents

List of figures :

General introduction:

Why autonomous drone ?

This idea that has been running in my head for a long time is to create an autonomous UAV network that can cover a country and offers several services such as delivery, emergency response and other services.

Our century is marked by the autonomy we see everywhere autonomous robots, cars ... and now it's the turn for drones.

Tests and prototypes of this type of UAV are in the initial phase, and even major delivery companies are investing in this area. The application of this type of drone must meet all safety standards and must contain multiple sensors and advanced algorithms to safely carry out its mission.

And all this we will discuss in the following chapters:

Chapter I : Global study of civil drones applications

Chapter II : Study and design of an autonomous quadcopter

Chapter III : Quadcopter prototype realization

Chapter I : Global study of civil drones applications

INTRODUCTION

The drone market will experience a real golden age, especially in the professional segment. The need for investors is real, especially during the first rounds if we want to see future giants in the sector emerge

I/Application of civilian drones

Rescued by a drone: at the end of June 2019, more than 100 children living in a remote region of Ghana, Africa, were rescued by the arrival of a Zipline company drone. Suffering from acute diarrhea, the school children, who had neither a local health centre nor adequate means of transport, all received a dose of rehydration salts, a treatment conveyed in record time by the flying apparatus. The proof, if need be, that the best of technology, free from logistical or energy constraints, can render very concrete services, even save lives.

Drone applications come from a wide variety of industries that already deploy drones for their respective purposes. In fact, drone applications have been found in different industry sectors:

- Agriculture
- Arts, Entertainment and Recreation
- Construction
- Educational Services
- Energy
- Health and Social Services
- Information
- Insurance
- Mining, quarrying, and oil and gas extraction
- Public administration
- Professional, scientific and technical services
- Real estate, rental and leasing and industrial plants
- Repair and maintenance
- Safety and security
- Transportation and storage

II / problems and solution

With applications as diverse as agriculture, energy, construction, health, insurance, cinema and, of course, security, isn't there a risk, however, of setting the conditions for a veritable "war of the skies" over our cities and countryside? Here again, technological advances allow us to look to the future with serenity: increasingly systematically equipped with geofencing and anti-collision devices, the new generation UAVs have everything to reassure the authorities and regulators in the skies. Since 2008, the International Civil Aviation Organization (ICAO) has been looking into the issue of UAVs, and should soon have regulations at least as secure as those for aviation.

III/ conclusion

The use of UAVs is becoming more and more important, we can summarize the importance of UAVs in these 5 points:

- Security
- Time gain
- New perspectives
- Ecological
- Innovative

Chaptre 2 Study and design of an autonomous quadcopter

INTRODUCTION

How does a quadcopter fly ?

The propeller is a device made up of several blades arranged regularly around an axis of the engine. When its axis enters in rotation, this system, thanks to the blades oriented according to a certain angle of attack, takes support on the fluid, which forms the lift force (the UAV can fly easily provided that the value of the lift force is twice as great as the value of the force of gravity).
For a quadracopter type UAV, the two opposite engines and their propellers must turn clockwise and the other two must turn counter-clockwise.
The UAV requires a set of sensors to maintain its stability and follow the pilot's instructions.

A/Mechanical study

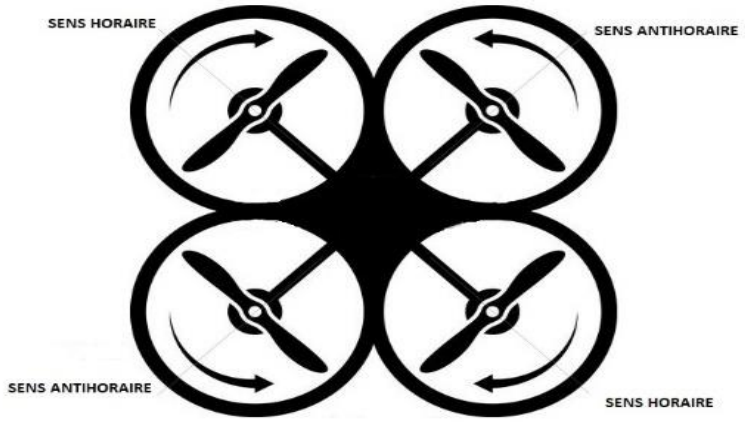


Fig2-1 : direction of rotation of the motors of a quadractor

The forces applied :

By studying the system we find that it is under 4 forces:

- The Weight
- lift
- Thrust
- drag

1/The Weight :

Weight is the force of gravity exerted by the Earth on the system, applied to the center of gravity and intensity:

P = m*g (2.1)

Aerodynamic force (lift+ drag) :

The rotation of the four engines provides an aerodynamic resultant force resulting from the flow of air over the blade surfaces, which is an inclined force directed from the bottom to the top, applied to the center of gravity. This force is generally decomposed into two forces: **lift** and **drag.**

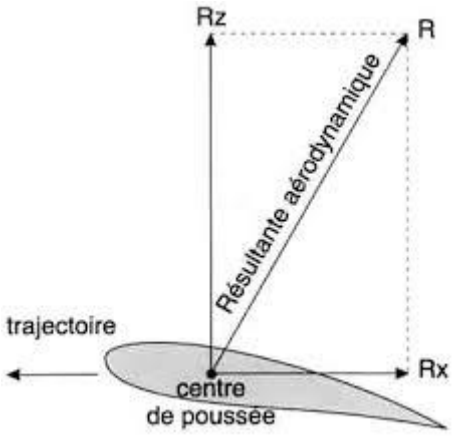


Fig2-2 : components of the aerodynamic force

2/ **The lift :**

The lift is a vertical force , when it compensates at least the weight, it allows the quadcopter to rise. its equation is linear and it is in the following form:

$$Fz = \frac{1}{2}*\rho*S*Cz*V^2 \quad (2.2)$$

V : speed of the propeller in m/s **V= Ω*r (2.3)**

Ω : rotation frequency in rad/s.

r : the distance between the center of propeller and the point of force (lift) in meter

ρ : air density in kg/m^3 1.3

S : surface area of the propeller in m² S= L*R

R : propeller radius

Cz : coefficient of lift without unit

Total lift of the quadracopter :

According to the propeller speed formula (2.3), the speed varies along the propeller i.e. the speed on the axis of rotation is zero, but it is maximum at the tip.

Then to obtain the total lift of the propeller it is enough to integrate the infinitesimal forces on infinitely small surfaces along the radius of the propeller.

So: $Fzth= 1/6*\rho*S*Cz*\Omega^2*R^3 \quad (2.4)$

Knowing that we have 4 propellers and each propeller contains 2 blades so the total lift force of the quadracopter: $Fzt= 4/3*\rho*S*Cz*\Omega^2*R^3 \quad (2.5)$

3/Drag :

The drag is the force that opposes the propeller movement. It is the resultant of the friction of the air on the propeller, it is opposite to the trajectory.

It is the lateral component of the force produced by the movement of the propeller.

The drag equation is linear in the following form:

$$Fx = ½*\rho*S*Cx*V^2 \quad (2.6)$$

V : speed of propeller in m/s

ρ : air density in kg/m^3

S : the projected reference surface of the wing according to OX in m^2

S = E*R (with E height or width on the OX OZ plane)

Cx : drag coefficient without unit

Then to obtain the total drag of the propeller it is enough to integrate the infinitesimal forces on infinitely small surfaces along the radius of the propeller (same method with the lift).

$$Fxth= 1/6*\rho*S*Cx*\Omega^2*R^3 \quad (1.7)$$

Knowing that we have 4 propellers and each propeller contains 2 blades so the total lifting force of the quadracopter**:** $Fxt= 4/3*\rho*S*Cx*\Omega^2*R^3$ **(1.8)**

4/The thrust:

The thrust is the force that drives the quadrocopter to move in the desired direction. The thrust or more precisely the thrust force is the result of a small increase in rotation at the level of one of 4 engines or even two engines. this increase gives us an angle of inclination that occurs when we want to increase the speed of lateral movement while keeping the aircraft in flight.

How to increase the efficiency of the quadracopter according to the formulas obtained from these forces?

1/ according to the gravity formula (2.1) the maximum weight of the system must be reduced

2/ according to the lift formula (2.2) to increase it we can play on the following factors: Increasing the propeller size, increasing the rotation speed, increasing the number of blades of a propeller.

Possible movements :

1/ Throttle : it simply corresponds to the ascent/descent (either by increasing or reducing the speed of the 4 engines).

2/ Yaw : this movement is used to make the quadracoaster turn on itself. It is obtained either by increasing the speed of the propellers clockwise and proportionally decreasing the speed of the propellers counterclockwise, or vice versa.

3&4/ Roll and Pitch : these movements are quite similar and aim at tilting the UAV on one axis or another. This movement is obtained by increasing the speed of two propellers on the same side and by proportionally lowering the speed of the two opposite propellers.

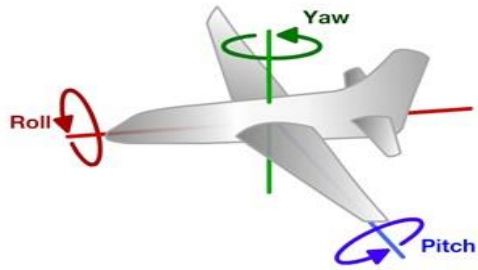


fig2-3 : Rotation axes of drone

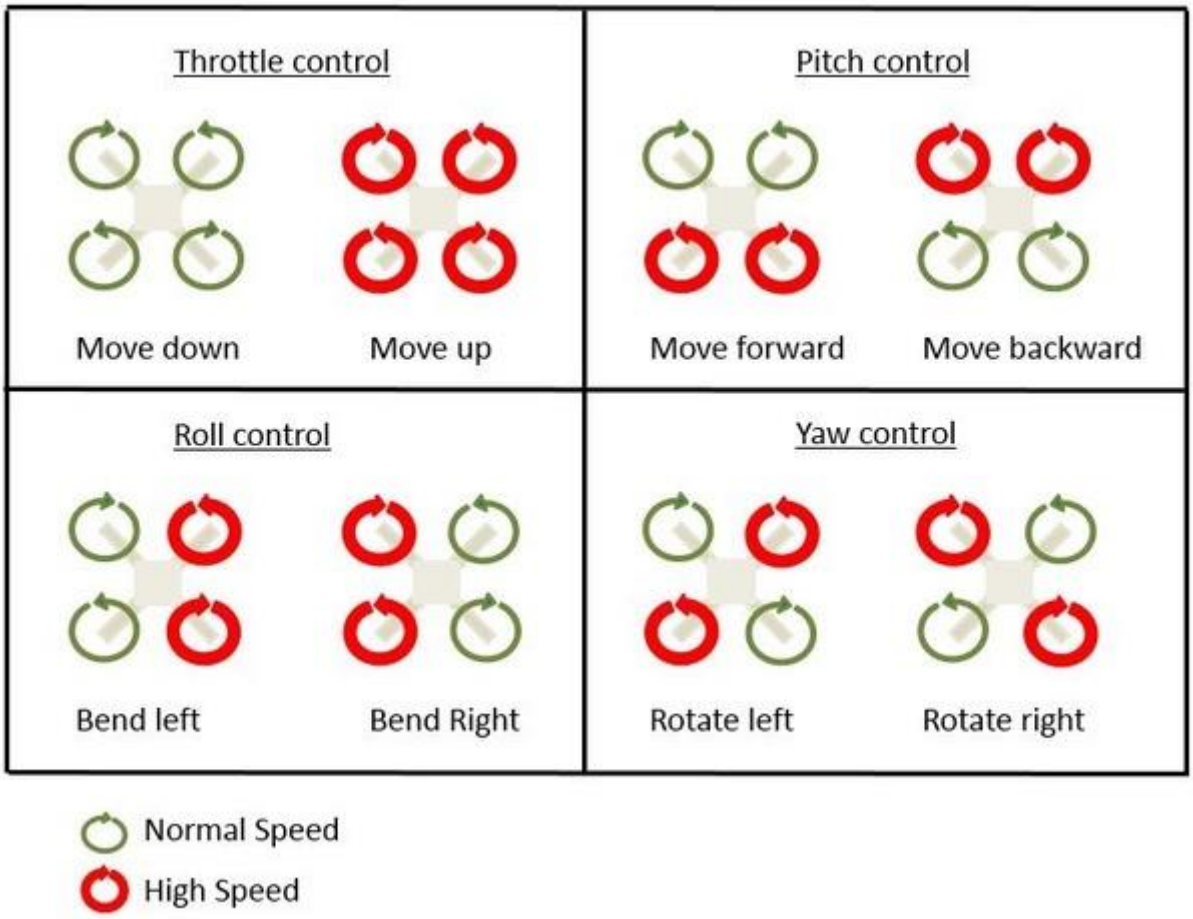


Fig 2-4 : possible movements of quadcopter

B/Electrical study :

Synoptic diagram

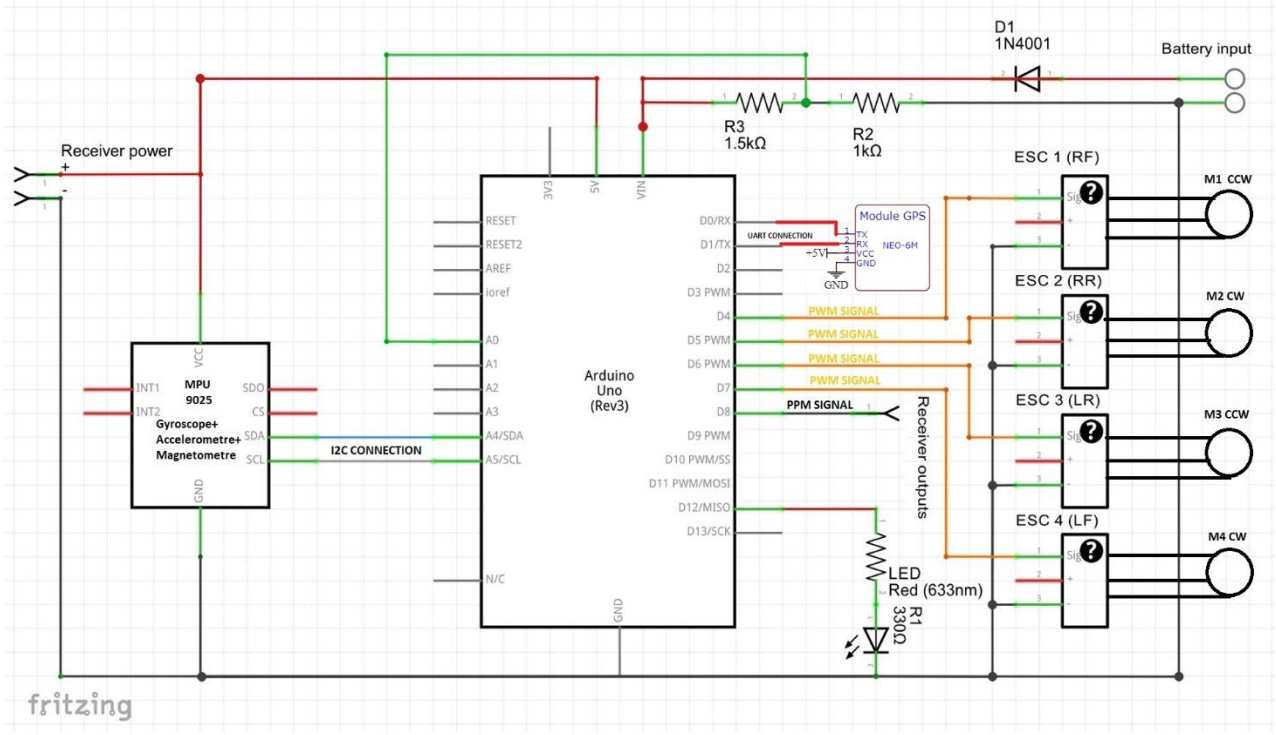


Fig2-5 Synoptic diagram

I- Radio Control Part :

A radio control is an instrument for controlling an aerodynamic system remotely. Une radiocommande se compose de deux éléments :

A radio control consists of two parts:

1/The transmitter:

Our transmitter must be able to send 6 channels to the receiver for a long distance and high speed to control the stability of the UAV, the possible movements and other functions for the UAV.

Our transmitter contains 4 potentiometers with 2 switches, a microcontroller and a 2.4GHz radio module.

***Microcontroller :**

we choose the Atmega8; it is a microcontroller of 8 bits, it contains 6 analog pins and the rest are digital, only one interface that supports the communication protocol SPI, I2C, UART, a clock of 16Mhz, 8KO of program memory, 1KO byte of SRAM, 512 byte EEPROM and it is supplied with 5V.

***Radio module:**

We choose the NRF24L01 with a 2.4GHz band antenna, its speed is between 250Kbps up to 2Mbps, it contains 4 pins for the SPI communication protocol and it is supplied with a 3.3v voltage.

***4 potentiometers** to simulate the movements (Throttle, Yaw , Roll , Pitch)

***2 switch buttons** to simulate other functions

Transmitter algorithm :

1/Setup of the radio module (by choosing the correct baud rate, set the connection mode between the radio module and the microcontroller, set the special key between the transmitter and the receiver).

2/ read the values of the different potentiometers and switches and set them between 0 and 255.

3/send data using the radio module

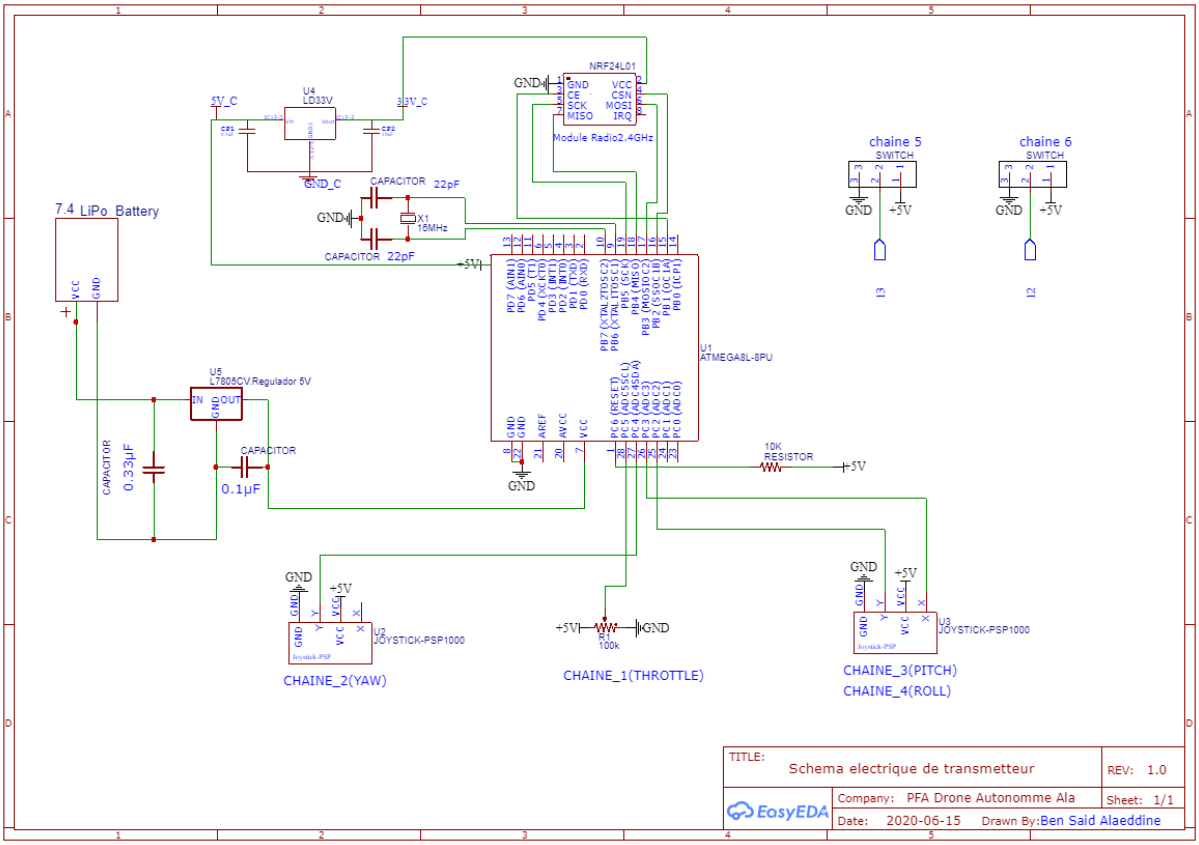


Fig2-6 Transmitter schematic

2/The receiver :

It contains almost the same components as the transmitter (Atmega8 and radio module nrf24L01).

The role of the receiver is to send the data sent by the transmitter to the flight controller using a TIMER type interrupt the receiver will generate a PPM type signal outgoing on a single pin from the receiver to the flight controller.

This PPM signal contains the values of all the channels of the radio control as shown in the following picture

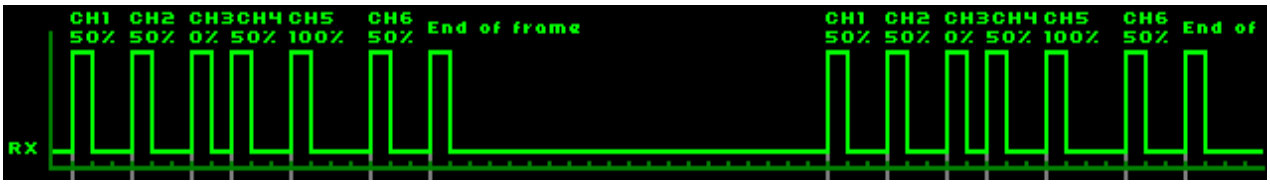


Fig2-7 example of a PPM signal

Receiver algorithm:

1/Setup of the radio module (by choosing the correct baud rate, set the connection mode between the radio module and the microcontroller, set the special key between the transmitter and the receiver).

2/Receiving data

3/using the TIMER interrupt we will generate a PPM signal that contains all the values of the 6 channels one after the other.

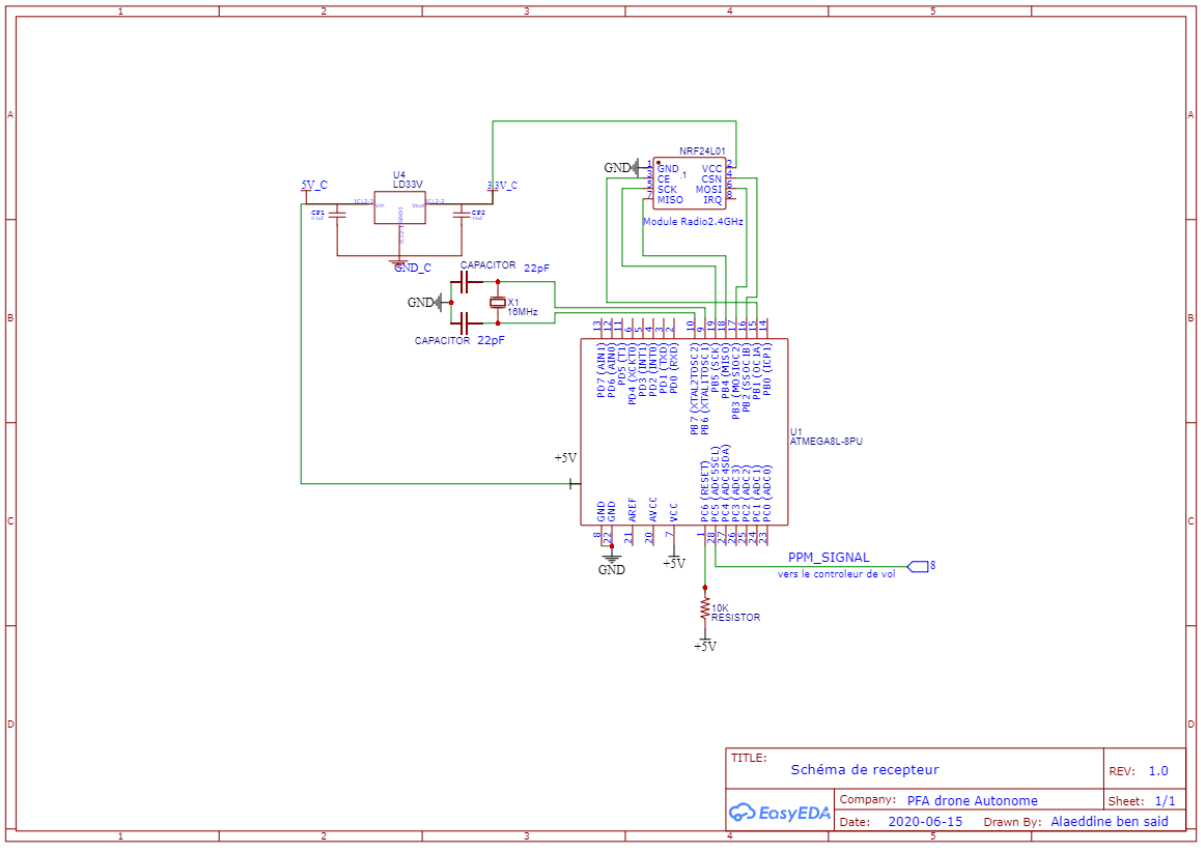


Fig2-8 receiver schematic

II/Power section :

1/ brushless motor

Les moteurs sans balais sont les moteurs les plus populaires pour les drones .
Brushless motors are the most popular motors for UAVs .

The brushless motor or brushless motor is an electric machine of the synchronous machine category with the rotor consisting of permanent magnets and the stator consisting of coils,
It is simply based on the interaction created between the magnets and the electromagnetic field created.
Seen from the outside, it operates on direct current. However, an electronic control system must switch the current in the stator windings.
The use of this type of motor is justified by its reliability, long service life, less noise and maintenance, low price and especially the control of its speed.

- **The KV number of a brushless motor**
  The KV number indicated on the motor translates the rotation per minute according to the following formula :
  Rotation Per Minute = KV * Input Voltage
  In this project a 2200 KV motor with a 12 volt battery was chosen.
  So the maximum RPM that can supply this motor is 2200*12 = 26400 rotation/min.

Characteristics of the motor chosen for this project

- Reference A2212/13T
- KV 2200
- 2-3 Lipo cells
- Max current 13 A for 60 secondes
- Dimension 2.8*2.8*3.2 mm
- Propulsion generate 300gr-800gr

2/Electronic Speed Controller

An electronic speed controller or ESC is an electronic circuit that controls and regulates the speed of an electric motor. It consists of a transistor assembly and a microcontroller to switch between the 3 different inputs of the motor in a well synchronized and precise way.
16 | P a g e
The electronic speed controller receives a PWM signal from the flight controller between 1000µs and 2000µs (1000µs means that the motor dosen't run and 2000µs means that the motor is at maximum rotation).
The electronic speed controller contains a Hall Effect sensor to properly synchronize the commutation between the three motor phases.

Caractéristique du ESC choisit lors de ce projet :

- Référence 30A BLDC ESC
- Supports 30A continuous output , 40 A for 10 seconds
- Dimensions 55mm x 26mm x 13mm
- Programmable in different modes

3/ Batteries (LiPo battery)

A Lipo battery is a Lithium-polymer type battery. This type is the most used for drones, RC cars and other vehicles, it offers a better power to weight ratio than all other batteries.

The battery is characterized by :

- Number of cells
- Capacity
- Discharge rate (C-Rate)

Each cell has a nominal voltage of 3.7 v (4.2 for a fully charged cell).
For this project we make the parallel connection between two 11.1v batteries(3s) to increase the intensity (i.e. the flight time).
The two batteries are identical Lipo type, capacity 3000mAh and with C-rate C=20.

→ Maximum discharge rate of a single battery = capacity * (C-rate)
=3000*20 = 60.A (this is the maximum current Can be debited by the battery)

→ The capacity of a battery is the flow of current that a battery can generate continuously for one hour (3000mAh i.e. 3A continuous current in one hour until the battery is exhausted).

4/ Power distribution board

This board allows the distribution of the power supply for the 4 brushless motors and their ESCs
It also powers the flight controllerWe add also a voltage divider (two resistors 1kΩ and 1.5 kΩ) to measure the battery voltage and a Schottky diode to protect the power circuit during code upload.

III/ Command part :

flight controller

The flight controller is the brain of our system. It manages many parameters to make its piloting possible. It is generally accompanied by many external sensors and their data are processed by the microcontroller.

1/Arduino Uno

For this project we chose the Arduino Uno based on the Atmega328P which is an 8-bit microcontroller with a 16MHz clock, it contains 6 PWM channels, 32K Byte of program memory, 1KBytes EEPROM, 2KBytes SRAM, a single interface for SPI communication and a single interface for I2C communication.
We will use the Arduino IDE to program this microcontroller.
The microcontroller will receive the various data from the sensors and then it will perform the calculations necessary to maintain the quadcopter's stability and the various flight modes.

Sensors

2/MPU 9250

It is a MEMS (micro electromechanical systems) inertial sensor of 9 axes, powered with 5v. It consists of a gyroscope of 3 axes, an accelerometer of 3 axes and a magnetometer of 3 axes, the MPU9250 communicates with the microcontroller using the communication protocol I2C.

The gyroscope :

It is a vibrating structure gyroscope, which uses a vibrating structure to determine the speed of rotation.

The gyroscope allows the determination of the rotation speed according to 3 axes (roll, pitch, yaw).
The gyroscope consumes about 3.2mA .
The analog-to-digital conversion of the signal is on 16 bits .
According to the datasheet the output of the gyroscope is expressed in degrees per second.
Some register must be configured in the MPU9250 to activate the gyro and select the Full-Scale Range for a value of +/-500 deg/sec.

The accelerometer :

An accelerometer is a sensor that, attached to a mobile or any other object, allows the linear acceleration to be measured. We talk about accelerometer even when it is actually 3 accelerometers that calculate linear accelerations along 3 orthogonal axes.
The accelerometer consumes about 450µA.
The analog-to-digital conversion of the signal is on 16 bits .
Accelerometer data are expressed in G (this is the gravitational constant about 9.81 m/s2).
Some register must be set in the MPU9250 to activate the accelerometer and select the Full-Scale Range for a value of +/-4 g.
You can use the accelerometer to check if the axes (Roll, Pitch, Yaw) are inverted or not during the flight and to eliminate the noise.

The magnetometer:

A magnetometer is a device used to measure the intensity or direction of a magnetic field (in our case it is the earth's magnetic field).
The magnetometer consumes about 280µA.
The analog-to-digital conversion of the signal is on 14 bits .
There is a single full scale range +/- 4800 µT.
The main role of a magnetometer for a quadcopter is to behave like a compass so we must convert the magnetic north indicated by the magnetometer to the geographic north and this margin of variation between these two north varies according to the geographic position as indicated on the map in the appendix (in Tunisia the difference between the geographic north and the magnetic north is 3°).

3/The barometer (BMP280) :

The BMP280 barometer uses MEMS technology, making it capable of measuring atmospheric pressure in a small flexible structure.

How does it work ?

It contains a diaphragm formed through a capacitive plate in contact with the atmosphere, the atmospheric pressure is detected through the deformation of the diaphragm due to the resulting pressure.
The higher the pressure, the more the diaphragm moves, resulting in a higher barometer reading.
The role of this sensor is to measure the altitude to help the UAV to keep the altitude or other features for an autonomous UAV.
It is well known that the altitude is inversely proportional to the measured pressure.

Altitude calculation is simply done with the barometric levelling formula

$P = P_0 e^{\frac{-\mu g h}{RT}}$,

where

μ - Molar mass of earth air, 0,0289644 kg/mol

g - Gravitational acceleration ,9,80665 m/(s*s)

h - Altitude difference, mètre

R - Universal gas constant for air, 8,31432 N·m /(mol·K)

T - Air temperature, K

P_0 correspond to the standard atmospheric pressure 0 meter above sea level

The BMP280 uses the SPI communication protocol.
The accuracy of this sensor is ±0.12 hPa (equivalent to ±1 m).
The BMP280 consumes approximately 2.7µA at a sampling rate of 1Hz.

4/ GPS module (NEO-6 u-blox 6) :

Global Positioning System is an essential module for the autonomy part of the quadracopter, the GPS module gives us several data such as longitude, latitude, altitude, real time, number of satellites connected to the module, for this project we will use just the longitude and latitude to locate the position of the quadracopter.

How does it work ?

To define a position in space, you need three coordinates (x, y and z). GPS data also includes a fourth variable: time. Therefore, four satellites with atomic clocks are needed to obtain a position, as well as a GPS receiver that will decode and calculate the signals received.
Each satellite emits an electromagnetic wave of known speed. This wave, carrying a "pseudo-random" code, is emitted at a specific time. The receiver then calculates the transmission time, i.e. the time needed for its signal (which carries the same pseudo-random code) to be in phase with the signal emitted by the satellite. By multiplying this time by the speed, it obtains the distance that separates it from the satellite.
At the end of this calculation, the receiver has a first piece of information: it is on a circle centered on the satellite. By repeating this procedure with a second satellite, it can again be located on a second circle centered on the second satellite. By repeating the operation a third time and looking for the area of intersection between these three circles, the position on the Earth is obtained. The fourth satellite is used to determine the offset between the time of the gps receiver and the exact time provided by the satellites, in order to refine the position. The greater the number of satellites received, the better the accuracy.
The update rate for this GPS module is 1Hz (every second).
The communication protocol used for this module is UART .

C / Operation of a quadracopter (normal mode) :

With the help of an interruption on a rising edge the flight controller reads all the values of the channels sent by the receiver in the form of a PPM signal, these values are the setpoint, the microcontroller then converts these values between the interval of 1000µs and 2000µs (1000µs i.e. motor at rest and 2000µs i.e. motor at its maximum speed) and then it sends for each ESC the well determined pulse in the form of a PWM signal to manage the desired movements of the pilot.

We send for all the ESCs the same value of channel1 (Throttle) and then to carry out the other movements (Roll, Pitch, Yaw) we increase the pulses according to the configuration of the motors.

Control of the quadcopter :

The quadracopter cannot maintain its stability easily because of the wind, the center of gravity is not in the center of the system, the 4 engines and their ESC are not perfectly identical, hence the risk of being crushed.

For this purpose a gyroscope is used to determine the angular velocity along the 3 axes of the system, the difference between the values of the radio control (setpoint) and the position of the quadracopter measured by the gyroscope represents the error of this system. So we will implement 3 PID regulators (one regulator for each axis) to maintain stability and correct the pulses sent to the ESCs.

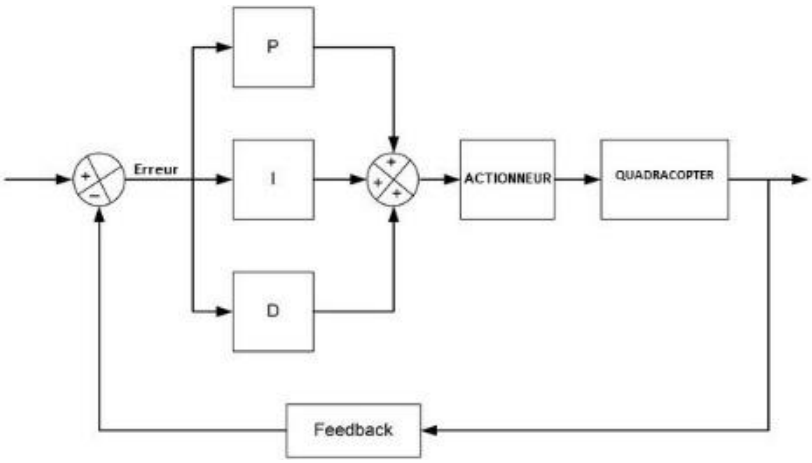


Fig2-9 closed loop control

P gain : determines how hard the flight controller works to correct the error and reach the desired flight path

I gain : determines how hard the flight controller works to maintain the UAV's attitude against external forces, such as wind and off-center CG.

D gain : works like a damper and reduces over-correction and overshoot caused by the P term. Since a damper prevents the suspension from bouncing, adding gain D can reduce oscillations caused by excessive P gain.

The coefficients P, I and D depend on the mechanical design of the quadractor, the hardware used .The determination of these values requires several tests to find the corresponding values for a stable flight.

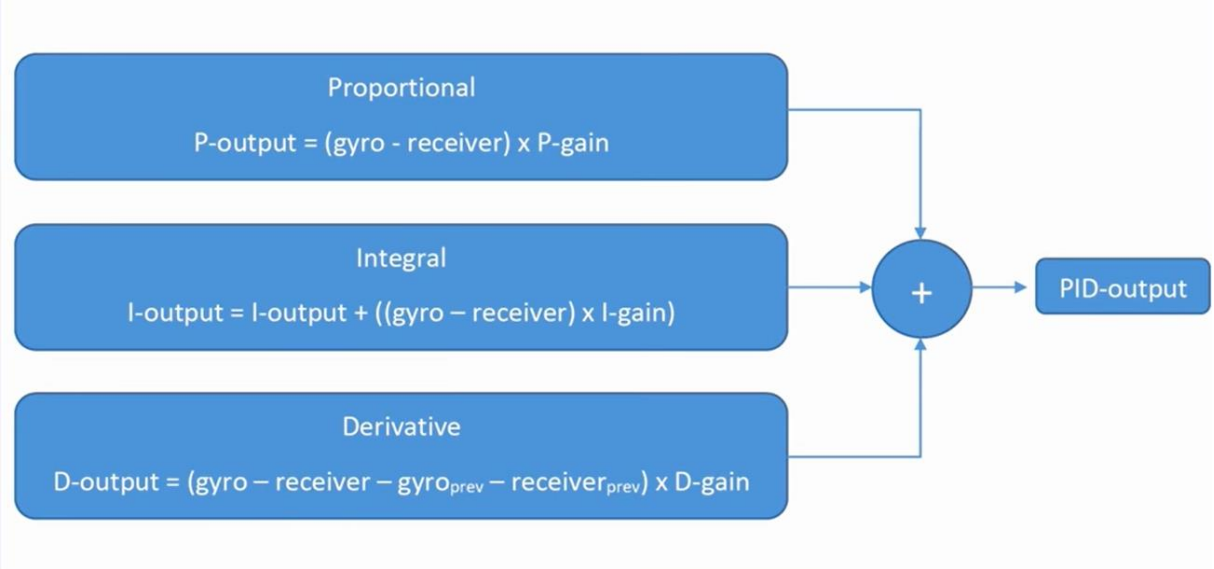


Fig2-10 PID equations

We send for all the ESCs the same value of channel1 (Throttle) and then to realize the other movements (Roll, Pitch, Yaw) we increase the pulses for the corresponding ESCs (as shown in the following equation).

Pulse for ESC 1 (right-front - CCW)

esc_1 = throttle - pid_output_pitch + pid_output_roll - pid_output_yaw

Pulse for ESC 2 (right-rear - CW)

esc_2 = throttle + pid_output_pitch + pid_output_roll + pid_output_yaw

Pulse for ESC 3 (left-rear – CCW)

esc_3 = throttle + pid_output_pitch - pid_output_roll - pid_output_yaw

Pulse for ESC 4 (left-front - CW)

esc_4 = throttle- pid_output_pitch - pid_output_roll + pid_output_yaw

D/Operation of a quadcopter (autonomous mode) :

In order for the UAV to fly completely autonomously, it must know its destination, its current position, its direction and altitude.

There are several autonomous functions (hold position, hold altitude, safe return, autonomous mission from point A to point B).

Above all, the radio control must contain a switch button that allows switching between normal mode and autonomous mode or vice versa to ensure safety.

1/Select an altitude to fly

The autonomous UAV must fly on a well chosen altitude in order to avoid collisions with buildings or other obstacles according to the geography of the place.
So the first step is the elevation of the UAV until the barometer indicates the acquisition of the necessary altitude (thus for the take-off procedure).

2/Orientation to the destination

The shortest route to point B must be chosen (i.e the shortest straight line between point A and B).
For the choice of this direction the bearing angle between these two points must be calculated.
The bearing : is the determination of the angle that makes, in the horizontal plane, the line of an observer towards an object with that of a fixed direction of reference. In general, the bearing is relative to the geographic North: positive in the clockwise direction.

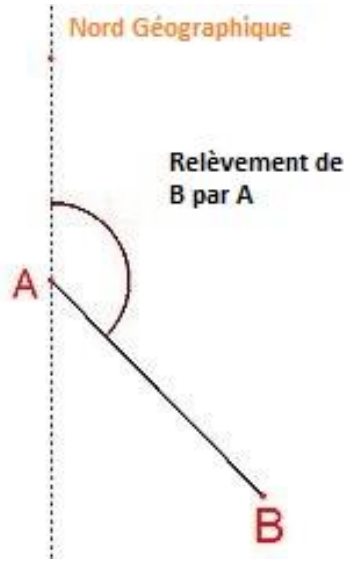


Fig2-11 Bearing angle between A and B

The bearing angle can be calculated with the help of software, This angle is stored in the flight controller memory, using the magnetometer the quadcopter will orient itself in order to maintain this orientation in relation to magnetic north.

3/ Path to follow

The coordinates (Longitude, Latitude) of point B (destination point) are also stored in the flight controller's memory.
During the flight the quadcopter processes data from the GPS module which indicates its current position,
The comparison between the destination and the current position is made, the UAV makes the decision to move forward or not.

Chaptre 3 Quadcopter prototype realization

Introduction

In this chapter we have presented the stages of construction and the realization of the quadracopter. The objective of this project is to build a first version of an autonomous quadcopter with the lowest cost to test the code and its efficiency

I/Realization of the hardware part

1/Radiocontrol

Transmitter

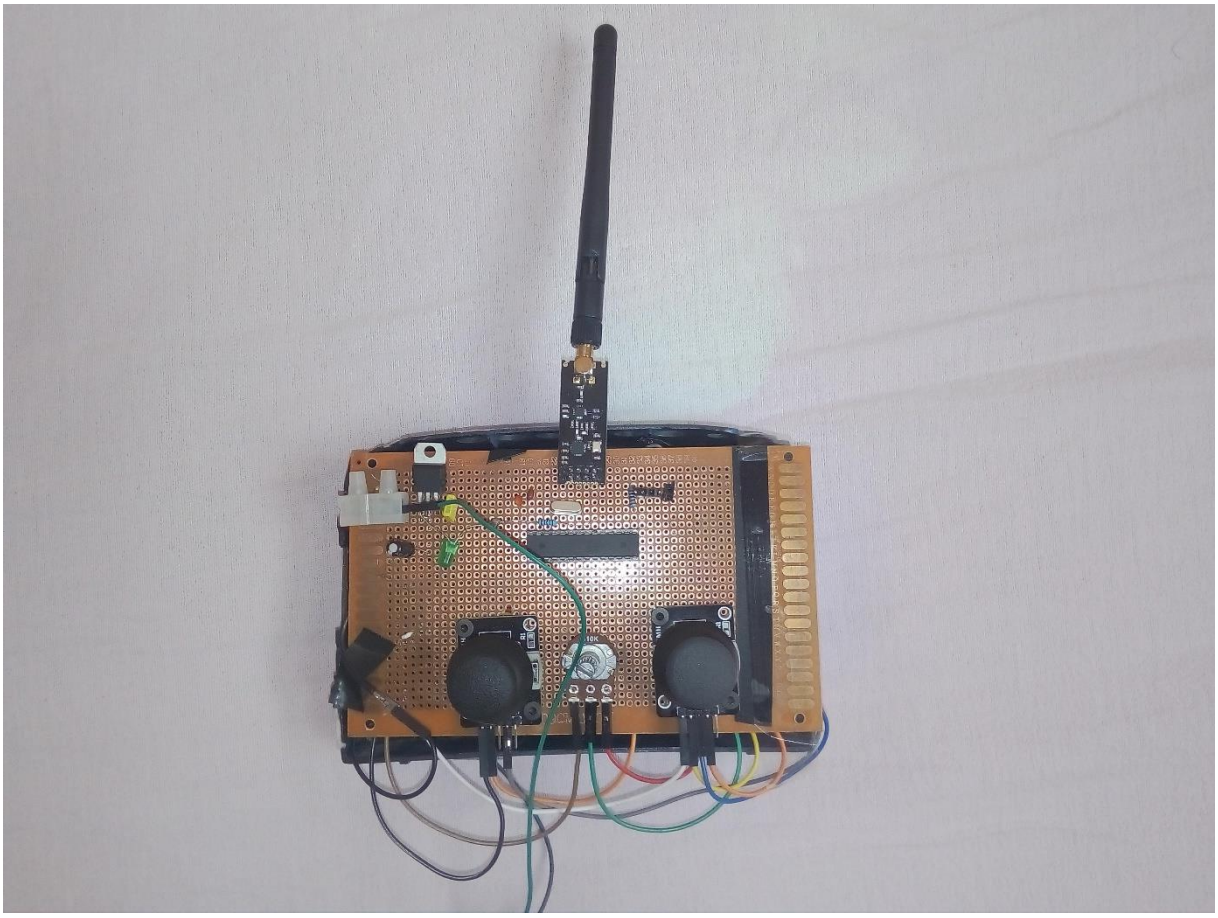


Fig 3-1 : design of a transmitter based on Atmge8 + NRF24L01

Receiver

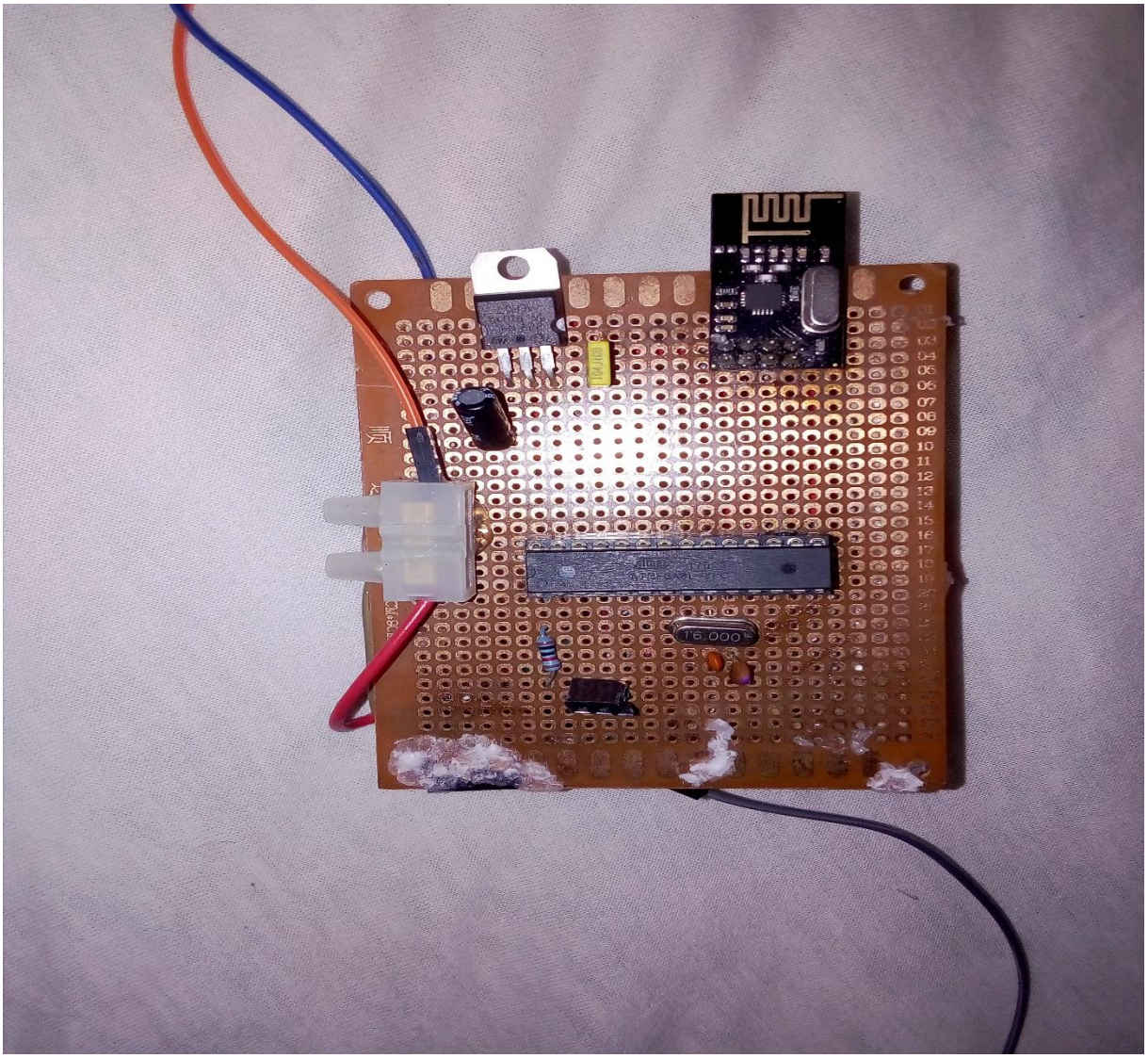


Fig 3-2 : design of a receiver based on Atmge8 + NRF24L01

2/frame of the quadcopter :

Plywood is chosen as a material because it is cheaper and more resistant than other materials such as plastic.

Fig3-3 The design of the drone is drawn on the plywood before cutting it.

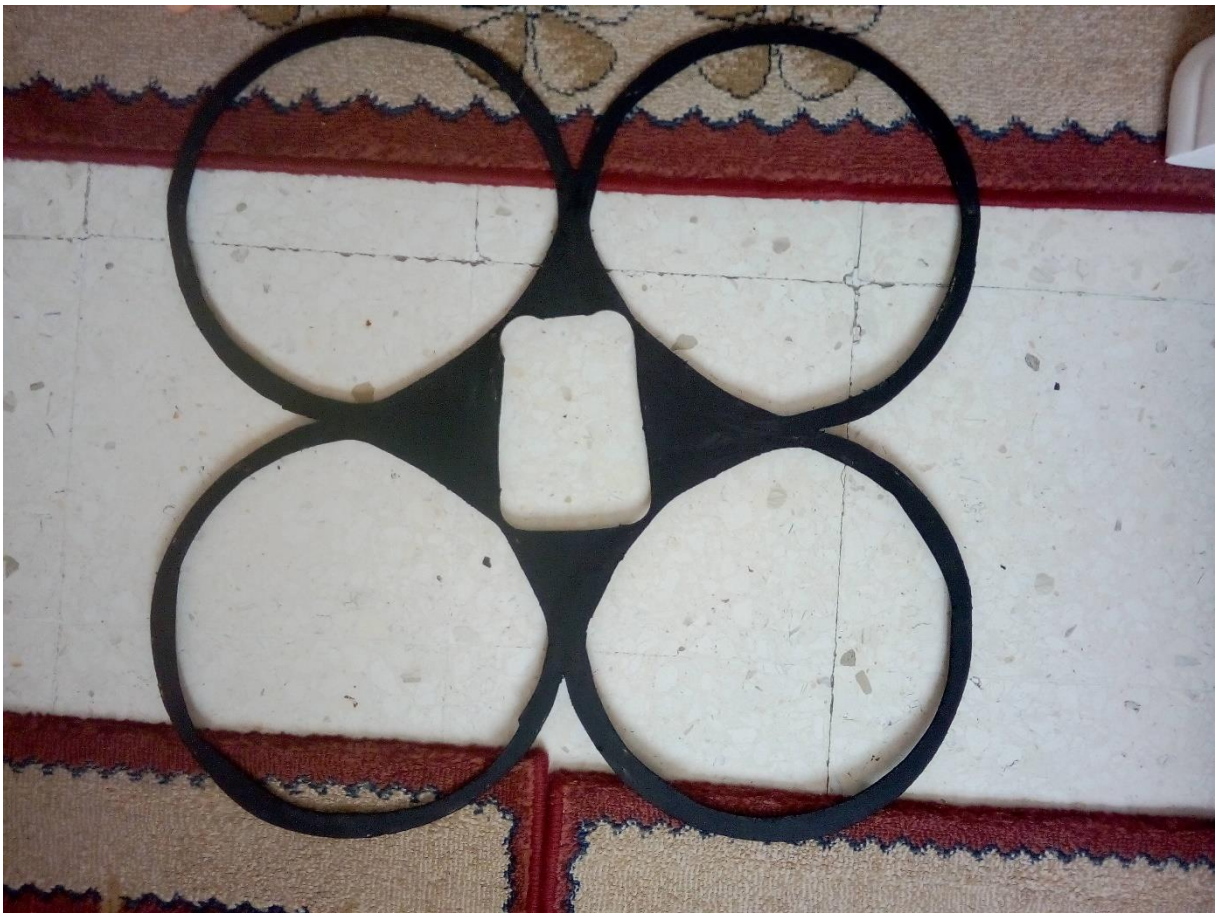


Fig 3-4 upper part of the frame

Fig3-5 lower part of the chassis (mounting of the motor and ESC speed controllers)

A light material such as pumice is inserted between the upper and lower parts and the two parts are joined together with tape.
this structure is for protection so the propellers will not represent any danger for the pilot and also to protect them during the crash.
This shape, which looks like a Kort Nozzle, allows to generate a higher lift force than other propellers without a Kort Nozzle.

Fig3-6 assembly

3/Power distribution board :

This board allows the power distribution for the ESCs speed controllers and their brushless motor, as well as the flight controller.

It also contains a voltage divider to determine the battery level and a Schottky diode for the protection of the Arduino board.

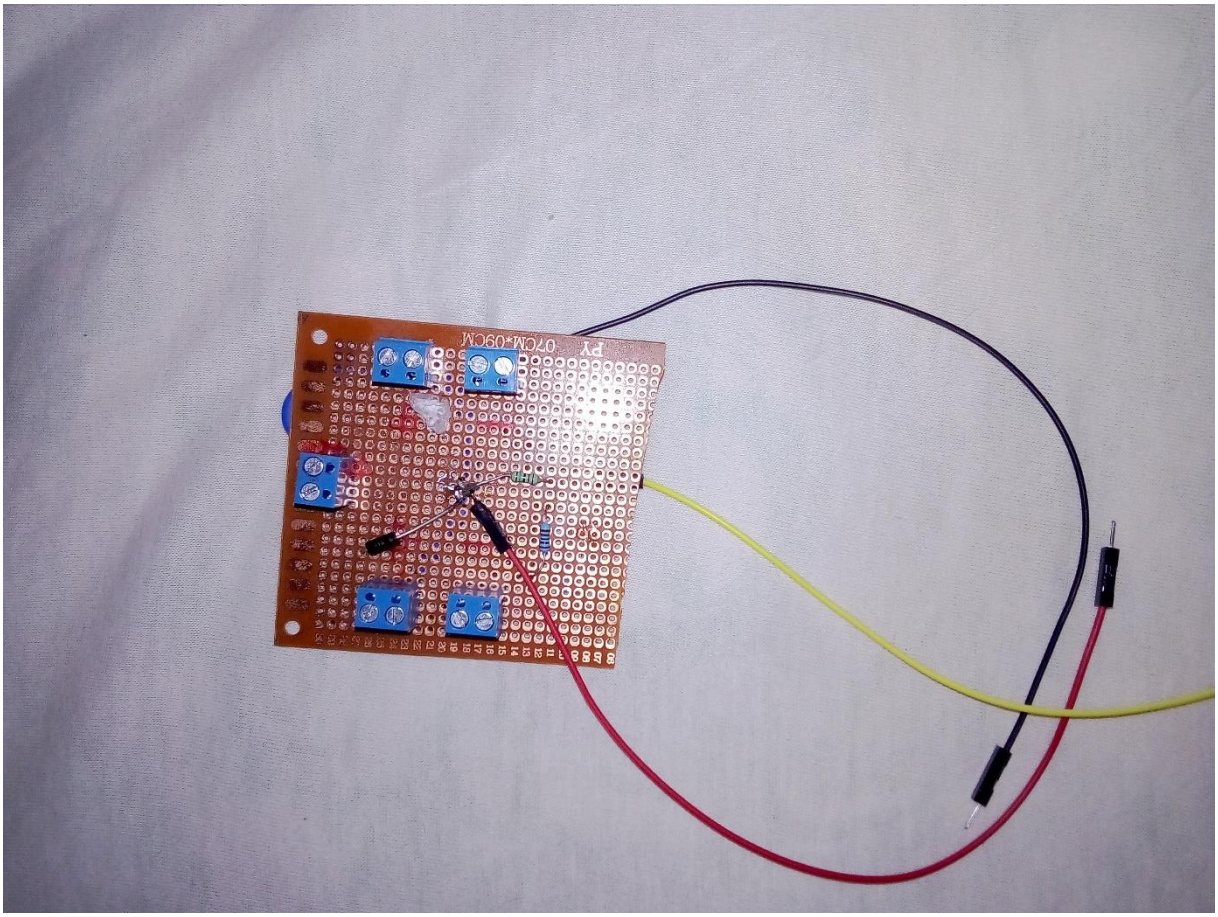


Fig 3-7 Power distribution board

4/Connecting batteries :

Two LiPO batteries are connected in parallel in order to increase the amperage, which leads to an increase in the quadcopter's flight time.

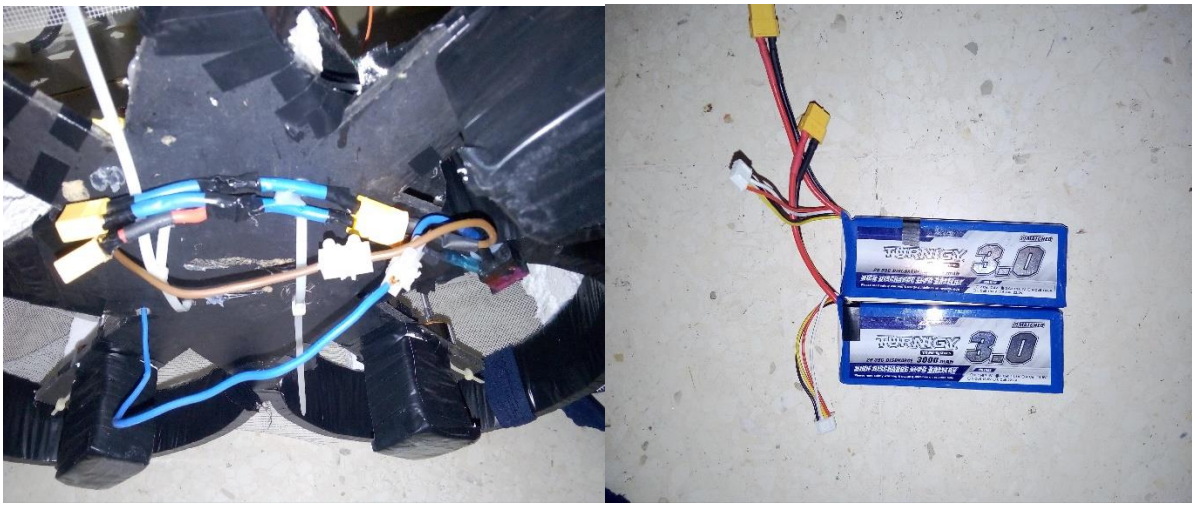


Fig3-8 Parallel battery wiring

5/Brushless motor

Fig3-9 brushless motor (A2212/6T 2200KV)

6/Electronics speed control (ESC)

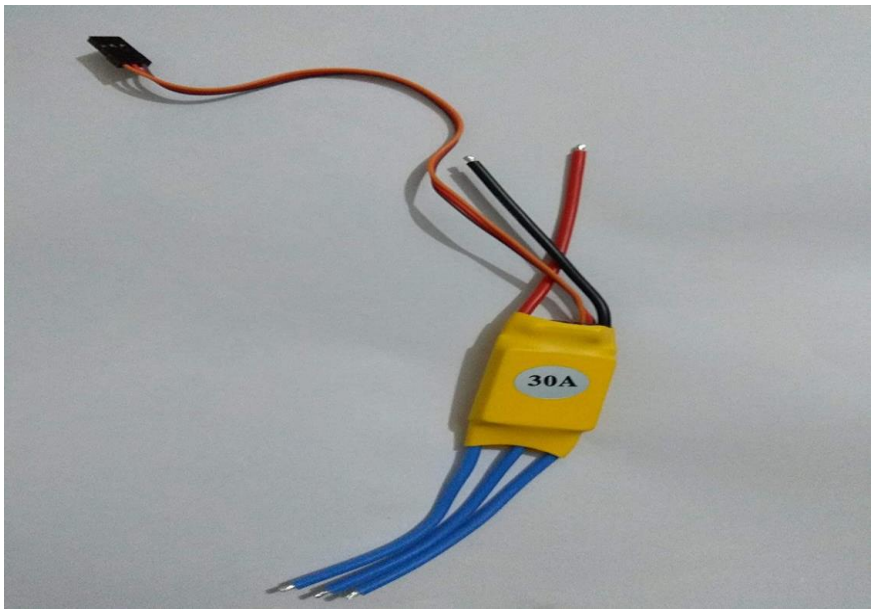


Fig3-10 ESC 30A BLDC ESC

7/MPU 9025 (Gyroscope + accelerometer + magnetometer)

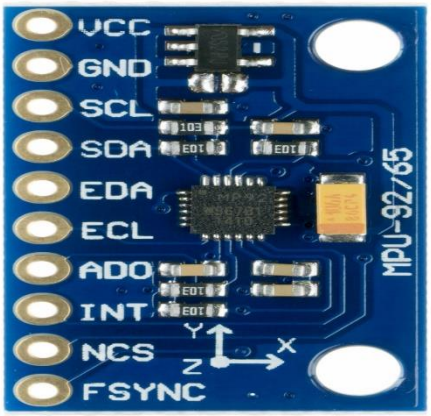


Fig3-11 MPU9025

The point in the microcontroller of the MPU9025 must point towards the left rear engine, to respect the direction of the axes already indicated in the code of the quadcopter.

8/Propeller Balancing

The propellers used are ABS 8x4 propellers.

For a quadractopter we have 4 propellers (2 clockwise and the others counter-clockwise) we have to do a balancing procedure (it consists in checking that the two have the same weight) in order to reduce the noise and the strong vibration that can affect the stability of the flight.

Fig 3-12 an unbalanced propeller

The solution is to make the two blades of the propeller of the same weight, for example we can glue some pieces of tape on the part of low mass as shown in the figure below.

Fig3-13 adding tape to the right low mass blade

Fig3-14 Result after balancing

Fig3-15 quadcopter ready to fly

II/Realization of the software part

The code is written on the Arduino IDE development environment. We try to avoid libraries and predefined functions as much as possible so that our code can allocate less memory in the microcontroller

Algorithm of a flight controller :

1/ The SETUP part :

- Define global variables
- Define the gains in PID values for each axis (following the test results)
- Initialization of communication protocols
- Initialization of the different sensors
- Calibration of the gyroscope (by reading several samples and then obtaining the average)
- Check that the values of channels are correct before starting the main program.

2/ The main code (an infinite loop)

- Convert the channels values (sent by receiver) in a range [1000µs-2000µs].
- Apply an 80%-20% filter for gyroscope data
- Activate or deactivate the motors according to very precise conditions (for activation, channel 1 must be at lowest point and channel2 at the highest point, to stop the motor, channel1 must be at the lowest point and channel2 at the lowest point).

- Store the inclination of the UAV desired by the pilot following the values of channel 2, 3 and 4 (YAW , ROLL , PITCH).
- Calculate the PID to ensure the quadcopter stability and the pilot's setpoint
- Calculate the battery level
- Send the final PWM pulse determined for each ESC speed controller.

3/ Routine of interruption :

This is a rising edge interrupt for the PPM signal sent by the receiver.
This interruption calculates the time between two rising edges to assign each channel its value, it also contains a synchronization procedure to ensure the succession of channels in the same order.

Conclusion

For the first version this drone seems to be satisfied for the work we have done.
But we must always think about improving and increasing the performance of our project.
Among the solutions that must be implemented as soon as possible, is to manufacture a printed circuit board that includes all the sensors with the receiver and flight controller to reduce noise and avoid short circuits during the vibration of the quadcopter.
Making a 3d printed frame (for example in carbon fiber) so that our quadcopter is well balanced around the center of gravity
We must also improve the microcontroller and work with the STM32F4 or STM32F7 which are characterized by a higher clock speed, larger memory space and other characteristics stronger than the Atmega328.

General conclusion :

At the end of this project we have worked with different engineering fields ; aerodynamics, radio control, programming, control theory and other areas that are strongly related to our study.

We learn many new concepts and skills during the realization of this project and especially the application of what we have learned during these two years of study.

This first version of the quadcopter seems very satisfied during the first flight tests with non-professional equipment.

This project does not stop here, we will improve this version with the use of other advanced microcontroller and sensors to have a professional version.

Bibliographies

A guide to understanding Lipo Batteries

https://www.robotshop.com/media/files/pdf/hyperion-g5-50c-3s-1100mah-lipo-battery-User-Guide.pdf

Applications of Magnetoresistive Sensors in Navigation Systems (Michael J. Caruso) Honeywell Inc.

https://www.generationrobots.com/media/module%20boussole%203%20axes%20HMC5883L/29133-Compass-Module-Application-Note-2.pdf

Applications of Magnetic Sensors for Low Cost Compass Systems(Michael J. Caruso) Honeywell, SSEC

https://www.electronicwings.com/download/attachment=Fri-04-17-17-36-09.Applications-of-Magnetic-Sensors-for-Low-Cost-Compass-Systems.pdf

Atmega8 datasheet

https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf

Atmega328p datasheet

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

NRF24L01 datasheet

https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf

MPU9025 datasheet

https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf

30A BLDC ESC datasheet

https://www.optimusdigital.ro/index.php?controller=attachment&id_attachment=451

LM7805 5v Voltage regulator

https://www.sparkfun.com/datasheets/Components/LM7805.pdf

AMS1117 3.3v Voltage regulator

http://www.advanced-monolithic.com/pdf/ds1117.pdf

Netographie

https://www.youtube.com/user/loraan

http://www.electronoobs.com/

https://www.youtube.com/user/Painless360/featured

https://www.youtube.com/channel/UC3c9WhUvKv2eoqZNSqAGQXg

https://www.ngdc.noaa.gov/geomag/WMM/image.shtml

http://www.brokking.net/

https://oscarliang.com/

https://www.movable-type.co.uk/scripts/latlong.html

https://www.igismap.com/map-tool/bearing-angle

APPENDIX A

Radio control code

1/transmitter code

MY_Drone_transmitter_4CH | Arduino 1.8.5

File Edit Sketch Tools Help

MY_Drone_transmitter_4CH

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

const uint64_t pipeOut = 0xE8E8F0F0E1LL; //IMPORTANT: The same as in the receiver

RF24 radio(9, 10); // select  CSN  pin

// The sizeof this struct should not exceed 32 bytes
// This gives us up to 32 8 bits channals
struct MyData {
  byte throttle;
  byte yaw;
  byte pitch;
  byte roll;
};

MyData data;

void resetData()
{
  //This are the start values of each channal
  // Throttle is 0 in order to stop the motors
  //127 is the middle value of the 10ADC.

  data.throttle = 0;
  data.yaw = 127;
  data.pitch = 127;
  data.roll = 127;
```

```
void setup()
{
  //Start everything up
  radio.begin();
  radio.setAutoAck(false);
  radio.setDataRate(RF24_250KBPS);
  radio.openWritingPipe(pipeOut);
  resetData();
  Serial.begin(9600);

}

/**************************************************/

// Returns a corrected value for a joystick position that takes into account
// the values of the outer extents and the middle of the joystick range.
int mapJoystickValues(int val, int lower, int middle, int upper, bool reverse)
{
  val = constrain(val, lower, upper);
  if ( val < middle )
    val = map(val, lower, middle, 0, 128);
  else
    val = map(val, middle, upper, 128, 255);
  return ( reverse ? 255 - val : val );  //variable = (condition) ? optionA : optionB;
}


void loop()
{

  data.throttle = mapJoystickValues( analogRead(A5), 0, 511, 1023, false );
  data.yaw      = mapJoystickValues( analogRead(A4),  0, 510, 1023, false );
  data.pitch    = mapJoystickValues( analogRead(A3), 0, 507, 1023, false);
  data.roll     = mapJoystickValues( analogRead(A2), 0, 512, 1023, false);

  radio.write(&data, sizeof(MyData));
}
```

2/Receiver code

The code receives the values sent by the transmitter, it will send it to the flight controller in the form of a PPM signal.


```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

////////////////////// PPM CONFIGURATION/////////////////////////
#define channel_number 4  //set the number of channels
#define sigPin A5  //set PPM signal output pin on the arduino
#define PPM_FrLen 27000  //set the PPM frame length in microseconds (1ms = 1000µs)
#define PPM_PulseLen 400  //set the pulse length
/////////////////////////////////////////////////////////////////

int ppm[channel_number];

const uint64_t pipeIn =  0xE8E8F0F0E1LL;

RF24 radio(9, 10);

// The sizeof this struct should not exceed 32 bytes
struct MyData {
  byte throttle;
  byte yaw;
  byte pitch;
  byte roll;

};

MyData data;
```

```
void resetData()
{
  // 'safe' values to use when no radio input is detected
  data.throttle = 0;
  data.yaw = 127;
  data.pitch = 127;
  data.roll = 127;


  setPPMValuesFromData();
}

void setPPMValuesFromData()
{
  ppm[0] = map(data.throttle, 0, 255, 1000, 2000);
  ppm[1] = map(data.yaw,      0, 255, 1000, 2000);
  ppm[2] = map(data.pitch,    0, 255, 1000, 2000);
  ppm[3] = map(data.roll,     0, 255, 1000, 2000);

   }

/**************************************************/

void setupPPM() {
  pinMode(sigPin, OUTPUT);
  digitalWrite(sigPin, 0);  //set the PPM signal pin to the default state (off)

  cli();
```

```
  cli();
  TCCR1A = 0; // set entire TCCR1 register to 0
  TCCR1B = 0;

  OCR1A = 100;  // compare match register (not very important, sets the timeout for the first interrupt)
  TCCR1B |= (1 << WGM12);  // turn on CTC mode
  TCCR1B |= (1 << CS11);  // 8 prescaler: 0,5 microseconds at 16mhz
  TIMSK |= (1 << OCIE1A); // enable timer compare interrupt
  sei();
}

void setup()
{
  resetData();
  setupPPM();

  // Set up radio module
  radio.begin();
  radio.setDataRate(RF24_250KBPS); // Both endpoints must have this set the same
  radio.setAutoAck(false);

  radio.openReadingPipe(1,pipeIn);
  radio.startListening();
  Serial.begin(9600);
}
```

```
unsigned long lastRecvTime = 0;

void recvData()
{
  while ( radio.available() ) {
    radio.read(&data, sizeof(MyData));
    lastRecvTime = millis();
  }
}

/**************************************************/

void loop()
{
  recvData();
  Serial.println(data.throttle);

  unsigned long now = millis();
  if ( now - lastRecvTime > 1000 ) {
    // signal lost?
    resetData();
  }

  setPPMValuesFromData();
}

/**************************************************/
```

Interrupt routine for PPM signal generation

```
#define clockMultiplier 2
ISR(TIMER1_COMPA_vect){
  static boolean state = true;
   TCNT1 = 0;
  if ( state ) {
    //end pulse
    PORTC = PORTC & ~B00100000;
    // turn pin 2 off. Could also use:   digitalWrite(sigPin,0) ;
    OCR1A = PPM_PulseLen * 2 ;
    state = false;
  }
  else {
    //start pulse
    static byte cur_chan_numb;
    static unsigned int calc_rest;
    state = true;
    if(cur_chan_numb >= channel_number) {
      PORTC = PORTC & ~B00100000;
      OCR1A = 35000 ;
      cur_chan_numb = 0;
    }
    else {
       OCR1A = (ppm[cur_chan_numb] - PPM_PulseLen) * 2;
       PORTC = PORTC | B00100000;
       calc_rest += ppm[cur_chan_numb];
      cur_chan_numb++;
    }
  }
}
```

APPENDIX B

Flight controller code (normal mode)

Before uploading the code we must first test all the sensors and motors to verify their operation, we must also do a calibration(synchronisation) for the motors and its ESC so that they start and stop at the same time.

Global variables of the code

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//PID gain and limit settings
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
float pid_p_gain_roll = 1.3 ;
float pid_i_gain_roll = 0.05 ;
float pid_d_gain_roll = 15  ;               )
int pid_max_roll = 400;                    //Maximum output of the PID-controller (+/-)

float pid_p_gain_pitch = pid_p_gain_roll;
float pid_i_gain_pitch = pid_i_gain_roll;
float pid_d_gain_pitch = pid_d_gain_roll;
int pid_max_pitch = pid_max_roll;          //Maximum output of the PID-controller (+/-)

float pid_p_gain_yaw = 4 ;
float pid_i_gain_yaw = 0.02 ;
float pid_d_gain_yaw = 0 ;
int pid_max_yaw = 400;                     //Maximum output of the PID-controller (+/-)

//Declaring global variables
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
byte eeprom_data[36];
byte highByte, lowByte;
int receiver_input_channel_1, receiver_input_channel_2, receiver_input_channel_3, receiver_input_channel_4;
int counter_channel_1, counter_channel_2, counter_channel_3, counter_channel_4, loop_counter;
int esc_1, esc_2, esc_3, esc_4;
int throttle, battery voltage;
```

```
int cal_int, start, gyro_address;
int receiver_input[5];
unsigned long timer_channel_1, timer_channel_2, timer_channel_3, timer_channel_4, esc_timer, esc_loop_timer;
unsigned long timer_1, timer_2, timer_3, timer_4, current_time;
unsigned long loop_timer;
double gyro_pitch, gyro_roll, gyro_yaw;
double gyro_axis[4], gyro_axis_cal[4];
float pid_error_temp;
float pid_i_mem_roll, pid_roll_setpoint, gyro_roll_input, pid_output_roll, pid_last_roll_d_error;
float pid_i_mem_pitch, pid_pitch_setpoint, gyro_pitch_input, pid_output_pitch, pid_last_pitch_d_error;
float pid_i_mem_yaw, pid_yaw_setpoint, gyro_yaw_input, pid_output_yaw, pid_last_yaw_d_error;
unsigned long  ch[4] ;
unsigned long  t[5] ;
int pulse = 0;
```

Setup routine :

It consists in initializing the communication protocols, the sensors, calibrating them (we take the average of several readings, this operation is marked by the flashing of the LED) and checking if the flight controller receives a correct signal from the radio control.

```
void setup(){
  //Serial.begin(57600);
  //Read EEPROM for fast access data.
  for(start = 0; start <= 35; start++)eeprom_data[start] = EEPROM.read(start);
  gyro_address = eeprom_data[32];                                   //Store the gyro address in the variable.
  Serial.begin(9600);
  Wire.begin();                                                     //Start the I2C as master.
  DDRB |= B00111110;                                                //Configure digital poort 9,10,11,12 and 13 as output.

  //Use the led on the Arduino for startup indication.
  digitalWrite(13,HIGH);                                            //Turn on the warning led.
  //Check the EEPROM signature to make sure that the setup program is executed
  while(eeprom_data[33] != 'J' || eeprom_data[34] != 'M' || eeprom_data[35] != 'B')delay(10);
  set_gyro_registers();                                             //Set the specific gyro registers.////////////////////
  for (cal_int = 0; cal_int < 1250 ; cal_int ++){                   //Wait 5 seconds before continuing.////////////////////
    PORTB |= B00011110;                                  //Set digital poort 9, 10, 11 and 12 high.
    delayMicroseconds(1000);                                        //Wait 1000us.
   PORTB &= B11100001 ;                          //Set digital poort 9, 10, 11 and 12 low.
    delayMicroseconds(3000);                                        //Wait 3000us.
  }  ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

  //Let's take multiple gyro data samples so we can determine the average gyro offset (calibration).
    for (cal_int = 0; cal_int < 2000 ; cal_int ++){                   //Take 2000 readings for calibration.////////////////
    if(cal_int % 15 == 0)digitalWrite(13, !digitalRead(13));        //Change the led status to indicate calibration.
    gyro_signalen();                                                //Read the gyro output.
    gyro_axis_cal[1] += gyro_axis[1];                               //Ad roll value to gyro_roll_cal.
    gyro_axis_cal[2] += gyro_axis[2];                               //Ad pitch value to gyro_pitch_cal.
    gyro_axis_cal[3] += gyro_axis[3];                               //Ad yaw value to gyro_yaw_cal.
```

```
  //Now that we have 2000 measures, we need to devide by 2000 to get the average gyro offset.
  gyro_axis_cal[1] /= 2000;                                         //Divide the roll total by 2000.
  gyro_axis_cal[2] /= 2000;                                         //Divide the pitch total by 2000.
  gyro_axis_cal[3] /= 2000;                                         //Divide the yaw total by 2000.*/////////////////////////////////////////////////
  cli();
  PCICR |= (1<<PCIE0);
  PCMSK0 |= (1<<PCINT0);
  sei() ;
  //Wait until the receiver is active and the throtle is set to the lower position.
  while(receiver_input_channel_1 < 990 || receiver_input_channel_1 > 1020 || receiver_input_channel_2 < 1400){
    receiver_input_channel_1 = convert_receiver_channel(1);     //Convert the actual receiver signals for throttle to the standard 1000 - 2000us
    receiver_input_channel_2 = convert_receiver_channel(2);     //Convert the actual receiver signals for yaw to the standard 1000 - 2000us
    receiver_input_channel_3 = convert_receiver_channel(3);
    receiver_input_channel_4 = convert_receiver_channel(4);
    start ++;                                                   //While waiting increment start whith every loop.
    //We don't want the esc's to be beeping annoyingly. So let's give them a 1000us puls while waiting for the receiver inputs.
    PORTB |= B00011110;
    delayMicroseconds(1000);                                    //Wait 1000us.
    PORTB &= B11100001 ;
    delay(3);                                                   //Wait 3 milliseconds before the next loop.
    if(start == 125){                                           //Every 125 loops (500ms).
      digitalWrite(13, !digitalRead(13));                       //Change the led status.
      start = 0;                                                //Start again at 0.
    }
  }
  start = 0;                                                    //Set start back to 0.*/
```

Void Loop

This is the place of the main code, it is an infinite loop that calculates the PWM signals to be sent to each ESC, all the calculations of the close loop control , the storage of the sensor data, the reading of the values of channels, the calculation of the battery level, the conditions for the START and stop of the motors.

```
void loop(){

  receiver_input_channel_1 = convert_receiver_channel(1);      //Convert the actual receiver signals for pitch to the standard 1000 - 2000us.
  receiver_input_channel_2 = convert_receiver_channel(2);      //Convert the actual receiver signals for roll to the standard 1000 - 2000us.
  receiver_input_channel_3 = convert_receiver_channel(3);      //Convert the actual receiver signals for throttle to the standard 1000 - 2000us.
  receiver_input_channel_4 = convert_receiver_channel(4);      //Convert the actual receiver signals for yaw to the standard 1000 - 2000us.

  gyro_roll_input = (gyro_roll_input * 0.8) + ((gyro_roll/ 57.14286 ) * 0.2);           //Gyro pid input is deg/sec.//57.14286
  gyro_pitch_input = (gyro_pitch_input * 0.8) + ((gyro_pitch/ 57.14286 ) * 0.2);        //Gyro pid input is deg/sec.
  gyro_yaw_input = (gyro_yaw_input * 0.8) + ((gyro_yaw/57.14286 ) * 0.2);              //Gyro pid input is deg/sec.

  //For starting the motors: throttle low and yaw left (step 1).
  if((receiver_input_channel_1 < 1050) && (receiver_input_channel_2 > 1950)){start = 1;
  }
  //Stopping the motors: throttle low and yaw right.
  if(receiver_input_channel_1 < 1050 && receiver_input_channel_2 < 1050)start = 0;
  if(receiver_input_channel_1 > 2060 || receiver_input_channel_2 >2060 || receiver_input_channel_3>2060 || receiver_input_channel_4>2060 )start = 0 ;

  //The PID set point in degrees per second is determined by the roll receiver input.
  //In the case of deviding by 3 the max roll rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).
  pid_roll_setpoint = 0;
  //We need a little dead band of 16us for better results.
  if(receiver_input_channel_4 > 1508)pid_roll_setpoint = (receiver_input_channel_4 - 1508)/3.0;
  else if(receiver_input_channel_4 < 1492)pid_roll_setpoint = (receiver_input_channel_4 - 1492)/3.0;

  //The PID set point in degrees per second is determined by the pitch receiver input.
  //In the case of deviding by 3 the max pitch rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).


   //The PID set point in degrees per second is determined by the pitch receiver input.
   //In the case of deviding by 3 the max pitch rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).
   pid_pitch_setpoint = 0;
   //We need a little dead band of 16us for better results.
   if(receiver_input_channel_3 > 1508)pid_pitch_setpoint = (receiver_input_channel_3 - 1508)/3.0;
   else if(receiver_input_channel_3 < 1492)pid_pitch_setpoint = (receiver_input_channel_3 - 1492)/3.0;

   //The PID set point in degrees per second is determined by the yaw receiver input.
   //In the case of deviding by 3 the max yaw rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).
   pid_yaw_setpoint = 0;
   //We need a little dead band of 16us for better results.
   if(receiver_input_channel_1 > 1050){ //Do not yaw when turning off the motors.
     if(receiver_input_channel_2 > 1508)pid_yaw_setpoint = (receiver_input_channel_2 - 1508)/3.0;
     else if(receiver_input_channel_2 < 1492)pid_yaw_setpoint = (receiver_input_channel_2 - 1492)/3.0;
   }
   //PID inputs are known. So we can calculate the pid output.
  calculate_pid();          ////////////////////////////////////////////////////////////////////
   //The battery voltage is needed for compensation.
   //A complementary filter is used to reduce noise.
   0.09853 = 0.08 * 1.2317.
   battery_voltage = battery_voltage * 0.92 + (analogRead(0) + 65) * 0.09853;
   //Turn on the led if battery voltage is to low.
  if(battery_voltage < 1030 && battery_voltage > 600)digitalWrite(13, HIGH);
   throttle = receiver_input_channel_1;                                        //We need the throttle signal as a base signal.
```

Calculates control values for each ESC

```
  if (start == 1){                                                          //The motors are started.
    if (throttle > 1800) throttle = 1800;                                   //We need some room to keep full control at full throttle
    esc_1 = throttle - pid_output_pitch + pid_output_roll - pid_output_yaw ; //Calculate the pulse for esc 1 (front-right - CCW)
    esc_2 = throttle+ pid_output_pitch + pid_output_roll + pid_output_yaw ; //Calculate the pulse for esc 2 (rear-right - CW)
    esc_3 = (throttle+ pid_output_pitch - pid_output_roll - pid_output_yaw ; //Calculate the pulse for esc 3 (rear-left - CCW)
    esc_4 = (throttle- pid_output_pitch - pid_output_roll + pid_output_yaw ; //Calculate the pulse for esc 4 (front-left - CW)
    if (battery_voltage < 1240 && battery_voltage > 800){                   //Is the battery connected?
      esc_1 += esc_1 * ((1240 - battery_voltage)/(float)3500);              //Compensate the esc-1 pulse for voltage drop.
      esc_2 += esc_2 * ((1240 - battery_voltage)/(float)3500);              //Compensate the esc-2 pulse for voltage drop.
      esc_3 += esc_3 * ((1240 - battery_voltage)/(float)3500);              //Compensate the esc-3 pulse for voltage drop.
      esc_4 += esc_4 * ((1240 - battery_voltage)/(float)3500);              //Compensate the esc-4 pulse for voltage drop.
    }
    if (esc_1 < 1100) esc_1 = 1100;                                         //Keep the motors running.
    if (esc_2 < 1100) esc_2 = 1100;                                         //Keep the motors running.
    if (esc_3 < 1100) esc_3 = 1100;                                         //Keep the motors running.
    if (esc_4 < 1100) esc_4 = 1100;                                        //Keep the motors running.
    if(esc_1 > 2000)esc_1 = 2000;                                           //Limit the esc-1 pulse to 2000us
    if(esc_2 > 2000)esc_2 = 2000;                                           //Limit the esc-2 pulse to 2000us.
    if(esc_3 > 2000)esc_3 = 2000;                                           //Limit the esc-3 pulse to 2000us.
    if(esc_4 > 2000)esc_4 = 2000;                                           //Limit the esc-4 pulse to 2000us.
  }
  else{
    esc_1 = 1000;                                                           //If start is not 2 keep a 1000us pulse for ess-1.
    esc_2 = 1000;                                                           //If start is not 2 keep a 1000us pulse for ess-2.
    esc_3 = 1000;                                                           //If start is not 2 keep a 1000us pulse for ess-3.
    esc_4 = 1000;                                                           //If start is not 2 keep a 1000us pulse for ess-4.
  }
```

```
  ,
  //All the information for controlling the motor's is available.
  //The refresh rate is 250Hz. That means the esc's need there pulse every 4ms.
  while(micros() - loop_timer < 4000);                                   //We wait until 4000us are passed.
  loop_timer = micros();                                                 //Set the timer for the next loop.
  PORTB |= B00011110 ;
  timer_channel_1 = esc_1 + loop_timer;                                  //Calculate the time of the faling edge of the esc-1 pulse.
  timer_channel_2 = esc_2 + loop_timer;                                  //Calculate the time of the faling edge of the esc-2 pulse.
  timer_channel_3 = esc_3 + loop_timer;                                  //Calculate the time of the faling edge of the esc-3 pulse.
  timer_channel_4 = esc_4 + loop_timer;                                  //Calculate the time of the faling edge of the esc-4 pulse.

  while(PORTB >= 2){                                                    //Stay in this loop until output 4,5,6 and 7 are low.
    esc_loop_timer = micros();                                           //Read the current time.
    if(timer_channel_1 <= esc_loop_timer)PORTB &= B11111101;             //Set digital output 4 to low if the time is expired.
    if(timer_channel_2 <= esc_loop_timer)PORTB &= B11111011;             //Set digital output 5 to low if the time is expired.
    if(timer_channel_3 <= esc_loop_timer)PORTB &= B11110111;             //Set digital output 6 to low if the time is expired.
    if(timer_channel_4 <= esc_loop_timer)PORTB &= B11101111;             //Set digital output 7 to low if the time is expired.
  }
}
```

Routine interruption

it is a rising edge interrupt of the PPM signal coming from the receiver in order to transform the value of each channel in the interval [1000µs-2000µs] and store them.

```
ISR(PCINT0_vect)
{
 if ( digitalRead(8)== HIGH )
 {
  switch (pulse)
  {
   case 1:
   t[1] = micros();
   ch[0] = t[1] - t[0]  ;
   receiver_input[1] = ch[0] ;
   pulse++;
   if (ch [0] > 30000 )
   {
    t[0] = t[1];
    pulse=1;
   }
   break;

   case 2:
   t[2] = micros();
   ch[1] = t[2] - t[1] ;
   receiver_input[2] = ch[1] ;
   pulse++;
 if (ch [1] > 30000)
   {
    t[0] = t[2];
    pulse=1;
   }
```

```
    }
    break;
    case 3:
    t[3] = micros();
    ch[2] = t[3] - t[2]  ;
    receiver_input[3] = ch[2] ;
    pulse++;

    if (ch [2] > 30000)
    {
     t[0] = t[3];
     pulse = 1;
    }
    break;
    case 4:
     t[0] = micros();
    ch[3] = (t[0] - t[3])-17900 ;
    receiver_input[4] = ch[3] ;
    pulse = 1 ;
    break;
    default:
    t[0] = micros();
    pulse++;
    break;
   }
  }
 }
 ////////////////////////////////
```

Function for reading gyroscope data

```
void gyro_signalen(){
  if(eeprom_data[31] == 1){
    Wire.beginTransmission(gyro_address);                    //Start communication with the gyro
    Wire.write(0x43);                                        //Start reading @ register 43h and auto increment with every read
    Wire.endTransmission();                                  //End the transmission
    Wire.requestFrom(gyro_address,6);                        //Request 6 bytes from the gyro
    while(Wire.available() < 6);                             //Wait until the 6 bytes are received
    gyro_axis[1] = Wire.read()<<8|Wire.read();               //Read high and low part of the angular data
    gyro_axis[2] = Wire.read()<<8|Wire.read();               //Read high and low part of the angular data
    gyro_axis[3] = Wire.read()<<8|Wire.read();               //Read high and low part of the angular data
  }

  if(cal_int == 2000){
    gyro_axis[1] -= gyro_axis_cal[1];                        //Only compensate after the calibration
    gyro_axis[2] -= gyro_axis_cal[2];                        //Only compensate after the calibration
    gyro_axis[3] -= gyro_axis_cal[3];                        //Only compensate after the calibration
  }
  gyro_roll = gyro_axis[eeprom_data[28] & 0b00000011];
  if(eeprom_data[28] & 0b10000000)gyro_roll *= -1;
  gyro_pitch = gyro_axis[eeprom_data[29] & 0b00000011];
  if(eeprom_data[29] & 0b10000000)gyro_pitch *= -1;
  gyro_yaw = gyro_axis[eeprom_data[30] & 0b00000011];
  if(eeprom_data[30] & 0b10000000)gyro_yaw *= -1;

}
```

Function for the calculation of PID (the formulas are the same as in chapter II)

```
void calculate_pid(){
  //Roll calculations
  pid_error_temp = gyro_roll_input - pid_roll_setpoint;
  pid_i_mem_roll += pid_i_gain_roll * pid_error_temp;
  if(pid_i_mem_roll > pid_max_roll)pid_i_mem_roll = pid_max_roll;
  else if(pid_i_mem_roll < pid_max_roll * -1)pid_i_mem_roll = pid_max_roll * -1;

  pid_output_roll = pid_p_gain_roll * pid_error_temp + pid_i_mem_roll + pid_d_gain_roll * (pid_error_temp - pid_last_roll_d_error);
  if(pid_output_roll > pid_max_roll)pid_output_roll = pid_max_roll;
  else if(pid_output_roll < pid_max_roll * -1)pid_output_roll = pid_max_roll * -1;

  pid_last_roll_d_error = pid_error_temp;

  //Pitch calculations
  pid_error_temp = gyro_pitch_input - pid_pitch_setpoint;
  pid_i_mem_pitch += pid_i_gain_pitch * pid_error_temp;
  if(pid_i_mem_pitch > pid_max_pitch)pid_i_mem_pitch = pid_max_pitch;
  else if(pid_i_mem_pitch < pid_max_pitch * -1)pid_i_mem_pitch = pid_max_pitch * -1;

  pid_output_pitch = pid_p_gain_pitch * pid_error_temp + pid_i_mem_pitch + pid_d_gain_pitch * (pid_error_temp - pid_last_pitch_d_error);
  if(pid_output_pitch > pid_max_pitch)pid_output_pitch = pid_max_pitch;
  else if(pid_output_pitch < pid_max_pitch * -1)pid_output_pitch = pid_max_pitch * -1;

  pid_last_pitch_d_error = pid_error_temp;


  //Yaw calculations
  pid_error_temp = gyro_yaw_input - pid_yaw_setpoint;
  pid_i_mem_yaw += pid_i_gain_yaw * pid_error_temp;
  if(pid_i_mem_yaw > pid_max_yaw)pid_i_mem_yaw = pid_max_yaw;
  else if(pid_i_mem_yaw < pid_max_yaw * -1)pid_i_mem_yaw = pid_max_yaw * -1;

  pid_output_yaw = pid_p_gain_yaw * pid_error_temp + pid_i_mem_yaw + pid_d_gain_yaw * (pid_error_temp - pid_last_yaw_d_error);
  if(pid_output_yaw > pid_max_yaw)pid_output_yaw = pid_max_yaw;
  else if(pid_output_yaw < pid_max_yaw * -1)pid_output_yaw = pid_max_yaw * -1;

  pid_last_yaw_d_error = pid_error_temp;
}
```

Function to convert channels values and their axis if inverted or not

```
int convert_receiver_channel(byte function){
  byte channel, reverse;                                             //First we declare some local variables
  int low, center, high, actual;
  int difference;

  channel = eeprom_data[function + 23] & 0b00000111;                 //What channel corresponds with the specific function
  if(eeprom_data[function + 23] & 0b10000000)reverse = 1;            //Reverse channel when most significant bit is set
  else reverse = 0;                                                  //If the most significant is not set there is no reverse

  actual = receiver_input[function];                                  //Read the actual receiver value for the corresponding function/////////////////
  low = (eeprom_data[channel * 2 + 15] << 8) | eeprom_data[channel * 2 + 14];  //Store the low value for the specific receiver input channel
  center = (eeprom_data[channel * 2 - 1] << 8) | eeprom_data[channel * 2 - 2]; //Store the center value for the specific receiver input channel
  high = (eeprom_data[channel * 2 + 7] << 8) | eeprom_data[channel * 2 + 6];   //Store the high value for the specific receiver input channel

  if(actual < center){                                               //The actual receiver value is lower than the center value
    if(actual < low)actual = low;                                    //Limit the lowest value to the value that was detected during setup
    difference = ((long)(center - actual) * (long)500) / (center - low);       //Calculate and scale the actual value to a 1000 - 2000us value
    if(reverse == 1)return 1500 + difference;                        //If the channel is reversed
    else return 1500 - difference;                                   //If the channel is not reversed
  }
  else if(actual > center){                                                                  //The actual receiver value is higher than the center value
    if(actual > high)actual = high;                                  //Limit the lowest value to the value that was detected during setup
    difference = ((long)(actual - center) * (long)500) / (high - center);      //Calculate and scale the actual value to a 1000 - 2000us value
    if(reverse == 1)return 1500 - difference;                        //If the channel is reversed
    else return 1500 + difference;                                   //If the channel is not reversed
  }
  else return 1500;
}
```

Function to initialize the sensor 9025

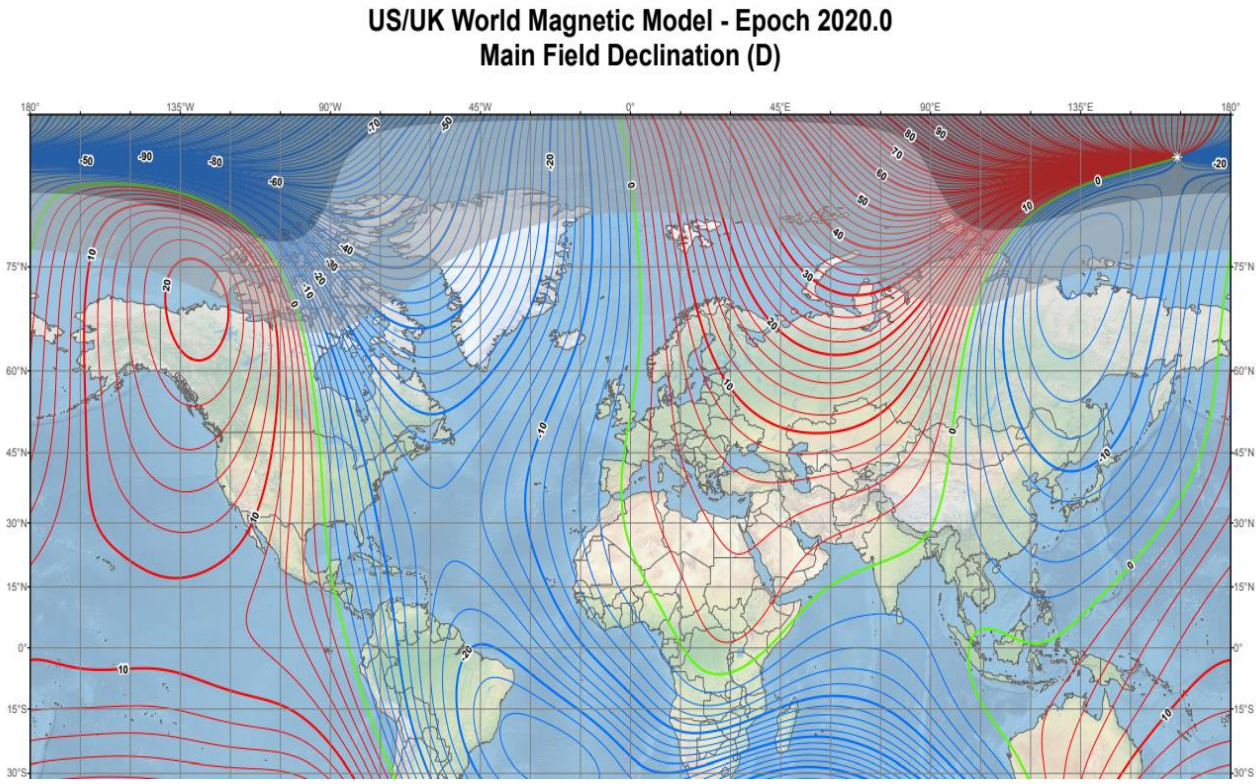
```
void set_gyro_registers(){
  if(eeprom_data[31] == 1){
    Wire.beginTransmission(gyro_address);                //Start communication with the address found during search.
    Wire.write(0x6B);                                    //We want to write to the PWR_MGMT_1 register (6B hex)
    Wire.write(0x00);                                    //Set the register bits as 00000000 to activate the gyro
    Wire.endTransmission();                              //End the transmission with the gyro.

    Wire.beginTransmission(gyro_address);                //Start communication with the address found during search.
    Wire.write(0x1B);                                    //We want to write to the GYRO_CONFIG register (1B hex)
    Wire.write(0x08);                                    //Set the register bits as 00001000 (500dps full scale)
    Wire.endTransmission();                              //End the transmission with the gyro

    Wire.beginTransmission(gyro_address);                //Start communication with the address found during search
    Wire.write(0x1B);                                    //Start reading @ register 0x1B
    Wire.endTransmission();                              //End the transmission
    Wire.requestFrom(gyro_address, 1);                   //Request 1 bytes from the gyro
    while(Wire.available() < 1);                         //Wait until the 6 bytes are received
    if(Wire.read() != 0x08){                             //Check if the value is 0x08
      digitalWrite(13,HIGH);                             //Turn on the warning led
      while(1)delay(10);                                 //Stay in this loop for ever
    }
    Wire.beginTransmission(gyro_address);                //Start communication with the address found during search
    Wire.write(0x1A);                                    //We want to write to the GYRO_CONFIG register (1B hex)
    Wire.write(0x03);                                    //Set the register bits as 00001000 (500dps full scale)
    Wire.endTransmission();                              //End the transmission with the gyro
  }
```

APPENDIX C

This map represents the difference in degree between magnetic north and geographical north and it varies according to the geographical position (for example in Tunisia it is 3°).



This map represents the inclination of the direction of the magnetic field in relation to the earth's surface, we must take into account this value of inclination according to our geographical position or we work to determine the geographical north using the magnetometer.

US/UK World Magnetic Model - Epoch 2020.0
Main Field Inclination (I)