People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering

Department of Electronics

Final Year Project Report Presented in Partial Fulfillment of
the Requirements for the Degree of

MASTER

In **Control**

Option: **Control Engineering**

Title:

Design and Implementation of a Quadcopter Controller

Presented by:

- **TEKFA Sonia**

Supervisor:

Dr. A.ZITOUNI

Registration number……./2018

Dedication

To my amazing parents for their eternal support, devotion and inspiration,

To my lovely family,

To my kind friends,

SONIA

Acknowledgment

All praise be to Allah, the lord of the universe, the most merciful, and the most beneficent.

I extend my gratitude to all the teachers (KHOUAS, OUADI and HARRICHE) for their help and contribution to this work.

A special thanks for the director of the institute MR.Kheldoun and my supervisor Mr. ZITOUNI for his guiding support and help.

Many thanks are given to all the professors and workers at the IGEE for guiding us during my curriculum.

I am infinitely grateful to my family, particularly our parents for their patience, unwavering support, continuous encouragement and belief in us throughout my whole life. I would have never made this far without them beside me, every step of the way.

I would like to express my gratitude to Dr. Bouchahdane who helped with all the necessary materials to achieve this project.

Special Thanks to Wameedh and Inelectronics Students Clubs who gave us a place to work.

I would like to thank also AliouaKarima, BensidiAissa Elyes and abd el aziz touhami who helped in this project completion.

I would like also to express my greatest thankfulness to friends from IGEE and ENP who have been there to support and encourage me.

Abstract:

The aim of this project is to model, design and implement on a microcontroller a set of control algorithms for a quad copter. The reason behind developing these algorithms is that a quad copter must track accurately a desired path and ensure stable attitude and position at stationary flight. The developed model is based on EULER-LAGRANGE method for rotational motions and on NEWTON-EULER method to compensate external forces such as wind effects. A simulation on MATLAB of the dynamic model has been performed before any real tests to assess the correct functioning of the controller and the embedded module. Two control techniques have been used along this project. The first is based on the PID, and the second on the LQ control. The simulation results validation are based on estimating our system performance (position and angles). Both controllers permit to stabilize the quad copter with a different efficiency. From the simulation results it has been shown that the LQ control give better performance. This result has been confirmed by building and testing each controller in a drone quad.

Table of content:

List of figures

List of tables:

List of Acronyms

BLDC	Brush-Less Direct Current
DC	Direct Current
ESC	Electronic Speed Controller
GCS	Ground Control Station
IMU	Inertial Measurement Unit
LiPo	Lithium Polymer
PCB	Printed Circuit Board
PID	Proportional, Integral, Derivative
LQR	Linear Quadratic Regulator
Pitch	Rotation around the side-to-side axis
PSU	Power Source Unit
PWM	Pulse Width Modulation
RC	Remote control
RF	Radio frequency
Roll	Rotation around the front-to-back axis
UAV	Unmanned Aerial Vehicle
Yaw	Rotation around the vertical axis
RPM	Revolutions per minute
VTOL	Vertical Take Off and Landing
SPI	Serial Peripheral Interface

List of symbols

e: landmark

b: mobile mark

F: pushing force

Ω: angular velocity in the fixed landmark

V: linear velocity in the fixed landmark

R: rotation matrix

T: transformation matrix

ξ: Position vector

x: coordinates of center of gravity G of the quad rotor according to X_e

y: coordinates of center of gravity G of the quad rotor according to Y_e

z: coordinates of center of gravity G of the quad rotor according to Z_e

Φ: roll angle

θ: pitch angle

Ψ: Lace angle

M: total mass

G: gravity

ω: rotational velocity of motors

ω_d: desired rotational velocity of motors

τ : Motor input torque

U: control input

x: state variable

x_d: Desired state

e: error

t: time variable

List of symbols

W: weight of quad copter

b: lift coefficient

F: drag force

C_d: drag coefficient

ρ: Air density

$F_{disturbance}$: Disturbance force

$\dot{V}$: First derivative of linear velocity

J: symmetric inertia matrix of dimension (3*3)

K: drag coefficient

Mp: pushing moment

M gyr: gyroscopic moment

M_d: Moments due to drag force

Ω: Angular velocity vector

L: Inductance

E.M.F: electro-motive force

V: source voltage

i: armature curent

T_e : Electric torque

K_f: Friction constant

J: rotor inertia

ω_i: Angular velocity

K_t : Electric torque constant

K_p: Proportional coefficient

K_i: Integral coefficient

K_d: Derivative coefficient

General Introduction

General Introduction:

Since the end of the last century, the interest for aerial vehicles control has grown up, and recently due to the development of technology, they became more accessible and cheaper, as microprocessors witnessed a reduction in dimension and price with enough energy to accomplish various tasks. Remotely controlled drones have been developed and their applications seem limited: as for military applications, contribution to geo-archeological researches (Oczipka and al, 2009), interventions in hostile environment or natural disasters (Apvrille, Tanzi, & Dugelay, 2014), or in agriculture (Tripicchio, Satler, Dabisias, Ruffaldi, & Avizzano, 2015) and many other fields.

Research in Autonomous Aerial Vehicles (UAV) is multidisciplinary. In fact, it groups various fields such as aerodynamics, signal and image processing, control, mechanics and real time data processing...

In this project we are interested in a particular VTOL configuration: the quad copter. The interest comes not only from its nonlinear dynamics, which represent an attractive control problem, but also from the design issue. Integrating the sensors, actuators and intelligence into a lightweight vertically flying system with a decent operation time is not trivial.

The objective of this project is to construct a stable quad copter guided via an RC (Radio Command), and develop a dynamic model used for two control approaches around the equilibrium point (hovering). First, a classical approach by Proportional Integral Derivative, then a modern approach Linear Quadratic control.

For a drone to accomplish its mission, it should have embedded material that executes algorithms (modeling, control and stable navigation), based on these algorithms behavior and position controls are generated.

Once the quad copter's flight parameters are defined, they will be used as control inputs. In this project we will be limited to linear control due to the embedded material that cannot treat complex algorithms.

We notice that, modeling the quad copter is the critical point in control algorithms implementation, since they are based on the dynamic model chosen; the more the actuators models and their effects are taken into consideration the closer control algorithms are to reality.

Our project is structured as follows:

Chapter 1: Quad copter principle of operation and modeling:

In this chapter, the operating principle of a quad copter is presented with a detailed description on the possible motions and flight modes. Based on kinematic laws, on forces and moments we will present a model of the quad copter. This will help elaborating a good MATLAB-Simulink model.

Chapter 2: Stabilization of the drone on the three axes:

In this part, a dynamic model will be given based on bibliographic references, and a Simulink model will help understand the need to implement a controller for the system. Two control approaches will be used a Proportional Integral Derivative controller and a Linear Quadratic one, where simulation results will permit performance comparison.

Chapter 3: Realization of a stable quad copter:

In this chapter, we will give the details about the different components of the quad copter construction. Then, present the way of controlling the motors and testing different components. Finally, apply control laws application on the implemented system.

We conclude with results and perspectives of our work.

Chapter 1

Quadcopter Principle of Operation and Modeling

Introduction:

A quad copter is a multi-copter consisting of four rotary wings motors. It is a multi-directional machine with a Vertical Take Off and vertical Landing (VTOL) able to perform stationary flights driven by four fixed pitch blades coupled to DC motors. The latter supplied by a Lithium Polymer, the embedded electronic chip treats various data of different sensors and applies control on the varying parameters of motors depending on the move to follow.
The four rotors are generally placed at crossing extremities while the electronic part is generally placed at its center. The figure below represents a picture of our quad copter drone

Figure 1.1-Quad copter drone overview

In order to avoid a yaw movement (the device turning on itself), it is necessary for two motors to rotate in one direction and that the other two rotate in the opposite direction. Moreover, to steer the device, motors rotating in the same direction must be placed opposite to each other.

the quad copter can be equipped with several sensors of different functions: an ultrasonic sensor that is actually located under the drone and pointing to the ground, a tri-axis accelerometer (measuring along three orthogonal directions of space), a three axes gyro meter, we can also add to it a GPS antenna and two cameras one pointing towards the front of the drone and the other towards the ground. To operate a quad copter it is enough to vary the power of the engines so it gets up or down, so it tilts from left to right (roll) or moves from backward to forward (pitch) or rotates on itself (yaw). the quad copter has six degrees of

freedom, three movements of translation and three rotation movements, those six degrees of freedom shall be controlled by means of only four triggers, so it is a sub actuated system[1].

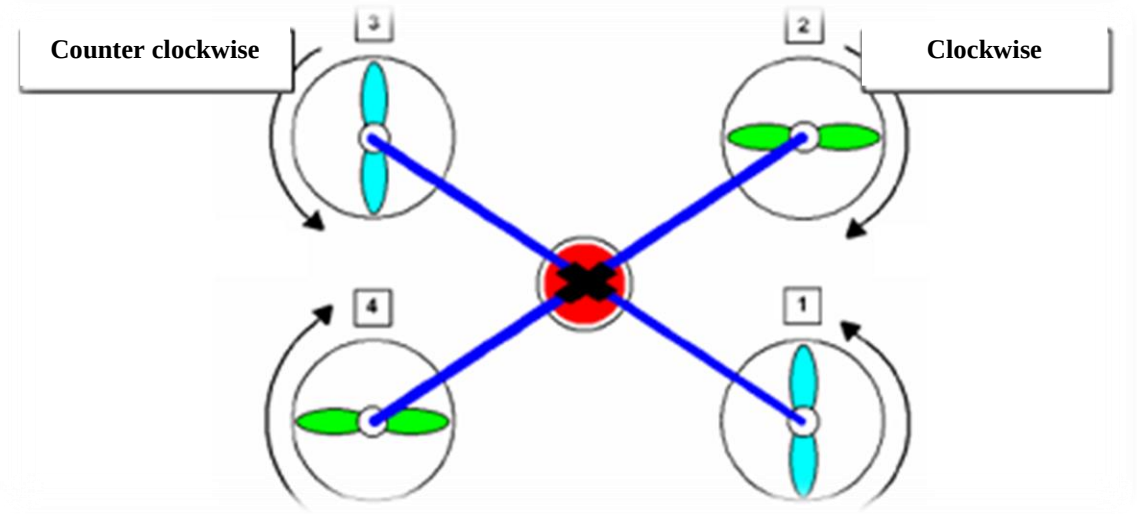


Figure 1.2-Rotational directions of Quad copter's rotors

I-1 Possible movements of a quad copter:

As defined in the previous part, we recall that the quad copter is a machine flywheel with four rotors placed at the ends of a cross. These four rotors provide the three possible movements for a quad rotor: the yaw, pitch and roll. [1]

I-1-A-**Rolling motion**:

In aeronautics, the rotation around the x-axis is called roll. Applying a thrust difference between the rotor two and the rotor four (generates a torque around the x-axis) varies the roll angle. This movement is coupled with a translation movement along the y-axis. Figure1.3- shows a roll motion

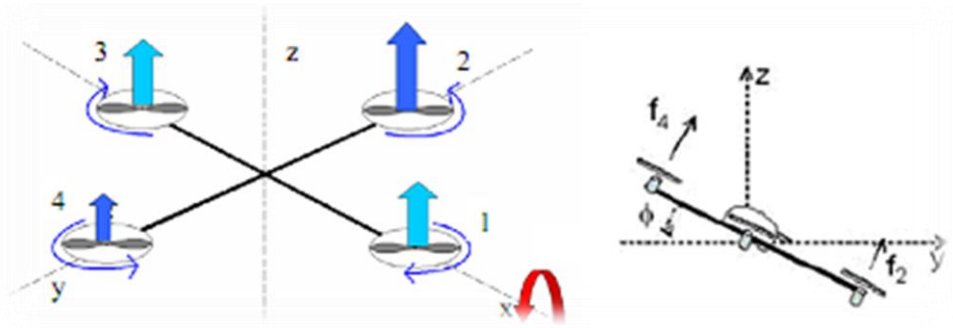


Figure 1.3-Roll motion

I-1-B- **Pitching motion:**

The rotation around the y-axis is called pitching in aeronautics. A variation of the pitch angle is achieved through a difference of thrust between the rotor one and three (generates a torque around the y-axis). This movement is coupled with a translation motion along x-axis.

Figure1.4- shows a pitch motion

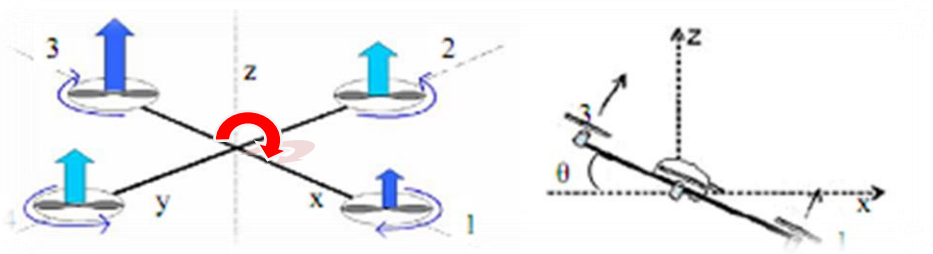


Figure 1.4- Pitching motion

I-1-C- Lace motion (Yaw):

The rotation around the z-axis is called the aeronautical yaw. When the engines rotate at equal speed anti-rotation torque is zero and the VTOL does not rotate. To modify the angle of yaw, it is necessary to apply a difference of speed between the couple rotors (one, three) and (two, four). This movement is not a direct result of the thrust but by the reactive couples produced by the rotation of the rotors. The direction of the pushing force does not shift during the motion but the increase in the lift force in a pair of rotors implies the decrease by equal amount of the other pair to ensure that all pushing force stays the same.

Figure1.5- shows a lace motion

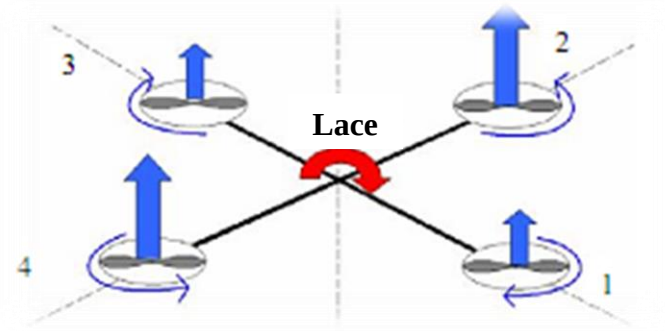


Figure 1.5- Lace motion

Indeed, projection of aerodynamic forces exerted by the air on the pale, shows that a rotor always tends to rotate the quad copter in the opposite direction of rotation.

I-2-Flight Modes

Based on the possible motions, the drone can perform three modes of flight: [4]

- Vertical flight.
- Hovering.
- Translation flight.

I-2-A-Vertical flight

In vertical flight, the aerodynamic result and the total weight are two forces having the same direction but of opposite [16]. The helicopter can go up or down, depending on the aerodynamic effect is either greater or less than the weight of the device.

Figure1.6- shows how a vertical flight is obtained

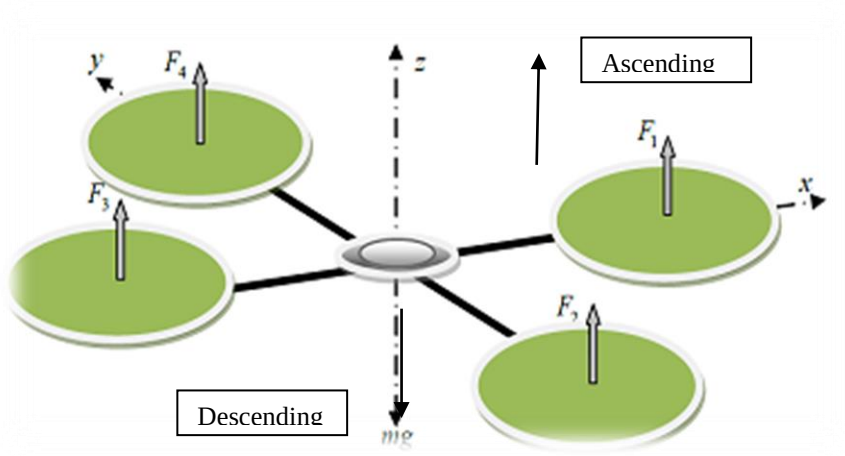


Figure 1.6-Vertical flight

I-2-B-Hovering flight

When the force of lift and that of gravity are equal and opposite, and the quad copter remains motionless, we are in a stationary flight (hovering).

Figure1.7 shows how a hovering flight is obtained.

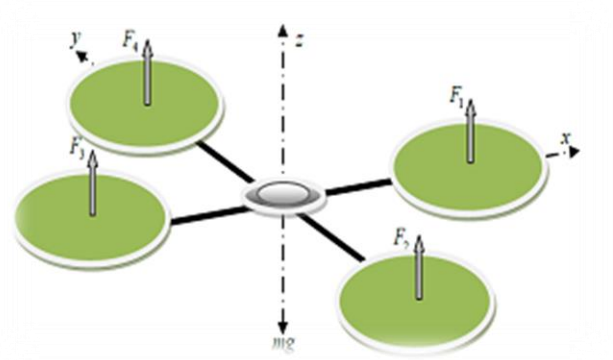


Figure 1.7- Hovering flight[1]

I-2-C-Translation flight

The translation flight corresponds to the navigation of the quad copter on a horizontal plane. It is ensured by the pitch and roll movements.

Figure1.8 shows how a translation flight is obtained.

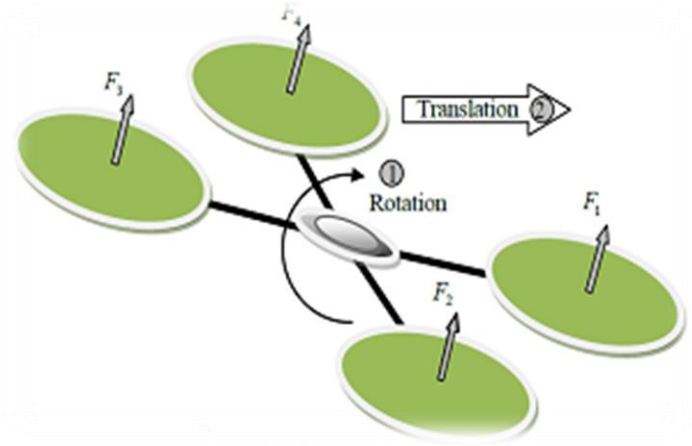


Figure 1.8- Translation flight[1]

I-3 Definition of benchmarks

To describe the flight dynamics of quad copter, a set of landmarks and basic notations must be defined. The first landmark is the inertial landmark $R_o = \{O, E_x, E_y, E_z\}$ or reference mark. This inertial landmark is linked to the Earth, and can be considered Galilean.

Next, we consider R_g= $\{G, E_1^g, E_2^g, E_3^g\}$ a local coordinate system originating from center of gravity of the drone.

The angles of Euler θ, Ψ, and Φ respectively pitch, roll, and yaw, are used to determine the orientation of the reference mark of the helicopter relative to the inertial mark.

I-4 Model hypotheses:

Modeling a quad copter is a sensitive task as its dynamic is largely nonlinear and fully coupled. Actually, a huge number of physical parameters affect our system, for that, it is necessary to make some work hypotheses and assumptions so to eliminate the less important, cited as: [2]

- The quad copter is supposed to be rigid and symmetric that induces that the inertia matrix is supposed to be diagonal.
- The pales are supposed to be rigid in order to neglect their deformation while rotating.
- The mass of the quad copter is one kilogram for a span of one meter.
- The lift and drag of each motor are proportional to the square of the speed which is very similar to the aerodynamic behavior of the real system.
- From the moment the quad copter is in flight, we will only use the speed relative to the speed of rotation of hovering motors.

The model is governed by the equations of mechanics which make it evolve on 6 axes. It undergoes acceleration forces of different types:

The lift: generated by the four rotating motors, it allows the drone to climb if it offsets at least the drag. It is written in the form: $\tau_x = bl(\Omega'^2_4 - \Omega'^2_2)$ on the axis of roll and $\tau_y =$

$bl(\Omega^2_3 - \Omega^2_1)$ on the pitch axis with $\boldsymbol{\Omega^2_i}$ the speed of each motor squared in (rad / s) ², **b** the lift coefficient in (kg.m / rad²) and **l** the half-span of the quad copter in meters.

The drag: resulting from the friction of the air on the quad copter, it is parallel and opposite to the trajectory. Its expression is: $\tau_z = d((\Omega^2_3 + \Omega^2_1) - (\Omega^2_4 + \Omega^2_2))$ with **d** the drag coefficient in kg.m² / rad².

The gyroscopic effect: when the quad copter is rotating on two axes, this force appears on the third axis and tends to resist the movements of the quad copter. It is:

$$\tau_y = I_{rotor}\omega_x(-\Omega^2_1 - \Omega^2_3 + \Omega^2_2 + \Omega^2_4) \;\&\; \tau_x = I_{rotor}\omega_y(\Omega^2_1 + \Omega^2_3 - \Omega^2_2 - \Omega^2_4)$$

With $\mathbf{I_{rotor}}$: the moment inertia of the motor in kg.m² and $\boldsymbol{\omega_x}$ the angular velocity along the x-axis in rad / s.

By projecting the three previous forces and adding the effect on the acceleration of the moments of inertia on each axis, the quad copter then reacts in roll, pitch and yaw in the following manner:

$$\ddot{\Psi} = \frac{d(\Omega_1^2 + \Omega_3^2 - \Omega_2^2 - \Omega_4^2)}{I_z} + \frac{(I_x + I_y)}{I_z}\dot{\phi}\dot{\Theta} \quad (1.1)$$

Equation of angular velocity along the yaw axis.

$$\ddot{\phi} = \frac{I_{rotor}\ \dot{\Theta}(\Omega_1 + \Omega_3 - \Omega_2 - \Omega_4)}{I_x} + \frac{(I_y - I_z)}{I_x}\dot{\Psi}\dot{\Theta} + \frac{bl(\Omega_4^2 - \Omega_2^2)}{I_x} \quad (1.2)$$

Equation of the angular velocity along the roll axis.

$$\ddot{\Theta} = \frac{-I_{rotor}\ \dot{\phi}\ (\Omega_1 + \Omega_3 - \Omega_2 - \Omega_4)}{I_y} + \frac{(I_z - I_x)}{I_y}\dot{\Psi}\dot{\phi} + \frac{bl(\Omega_3^2 - \Omega_1^2)}{I_y} \quad (1.3)$$

Equation of the angular velocity along the pitch axis.

Also, the quad copter undergoes acceleration on the three axes of space. These accelerations depend on the lift generated by the engines which is $\boldsymbol{T_i}$ known as $T_i = b \cdot \Omega_i^2$ with $\mathbf{i} \in \{1, 2, 3, 4\}$ number of each engine.

From the principle of dynamics: $\vec{F} = m \cdot \vec{g}$ with $\vec{F} = \sum T_i$ and projected in the three dimensions of space, these accelerations are summarized as follows:

$$\ddot{x} = \frac{(\cos\Psi \sin\theta + \sin\Psi \sin\phi)}{m} \sum_1^4 T_i \quad (1.4)$$

Equation of acceleration along the x-axis

$$\ddot{y} = \frac{(\cos\phi \sin\Psi \sin\theta - \cos\Psi \sin\theta)}{m} \sum_1^4 T_i \quad (1.5)$$

Equation of the acceleration along the y-axis

$$\ddot{z} = -g + \frac{(\cos\phi \cos\theta)}{m} \sum_1^4 T_i \quad (1.6)$$

Equation of the acceleration along the z-axis

More details will be developed in the next section.

I-5-Development of the model according to Lagrange-Euler:

The model will be developed according to a Lagrangian approach, i.e. according to the kinetic and potential energies. [3]

The Lagrange equation is written:

$$\Gamma_i = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \left(\frac{\partial L}{\partial q_i}\right) \quad (1.7)$$

With: $$L = T - V \quad (1.8)$$

With q_i: generalized coordinates.

Γ_i: Generalized forces given by non-conservative forces.

T: total kinetic energy.

V: total potential energy.

Yaw, pitch and roll angles (aeronautical angles) are defined in the following way (Figure 1.9)

-Rotation of $\phi(t)$ around the x-axis (roll angle with $-\pi/2 < \phi < \pi/2$ -)

-Rotation of θ (t) around the y-axis (pitch angle with $-\boldsymbol{\pi}/\mathbf{2} < \theta < \boldsymbol{\pi}/\mathbf{2}$)

-Rotation of Ψ(t) around the z axis (yaw angle with $-\boldsymbol{\pi} < \boldsymbol{\Psi} < \boldsymbol{\pi}$)

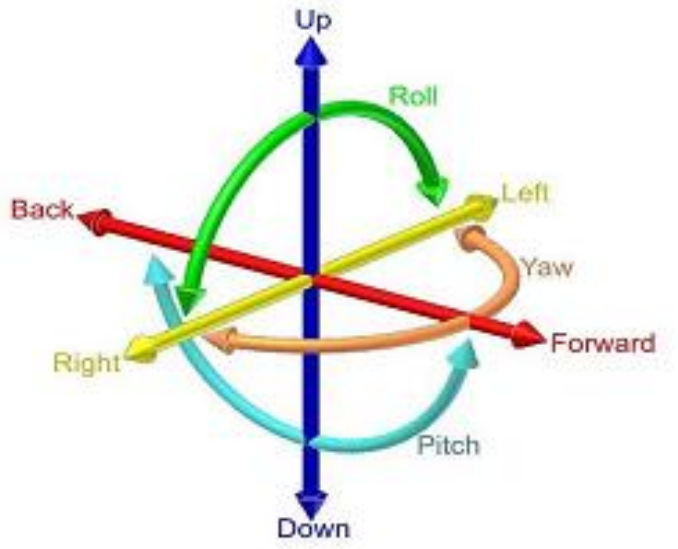


Figure 1.9- Euler rotations and angles

Note: the entriesϕ(t) , θ (t), and Ψ(t) will be denoted ϕ,θ, Ψ and for simplification.

To describe the position and orientation of the drone in the R_0 marker, a parameterization in yaw, pitch, and roll is used [27]. The configuration of the device is described by means of three elementary rotations defined by the three angles of rotation.

In order to make the transition from the marker to the mark, it is necessary to carry out three rotations around the three axes:

$$R(E_X, E_y, E_z) \xrightarrow{H_\Psi} R(U, V, E_Z) \xrightarrow{H_\theta} R(E_1^g, V, W) \xrightarrow{H_\phi} R(E_1^g, E_2^g, E_3^g) \quad (1.10)$$

Or:

$R_0 = \{O, E_X, E_Y, E_Z\}$ Is the base landmark, and $R_G = \{G, E_1^g, E_2^g, E_3^g\}$ is the local landmark.

$R = \{U, V, E_Z\}$ And $R = \{E_1^g, V, W\}$ are intermediate bases andH_Ψ, H_θ, H_ϕ the matrices of orthogonal rotation.

$$R_{(x,\phi)} = \begin{pmatrix} 1 & 0 & 0 \\ \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \end{pmatrix} \quad (1.11)$$

$$R_{(y,\theta)} = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \quad (1.12)$$

$$R_{(z,\Psi)} = \begin{pmatrix} \sin\Psi & -\sin\Psi & 0 \\ \sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (1.13)$$

By multiplying the three matrices we obtain. :

$$R_{(\phi,\theta,\Psi)} = \begin{pmatrix} \cos\Psi\cos\theta & \cos\Psi\sin\theta\sin\phi - \cos\phi\sin\Psi & \cos\phi\cos\Psi\sin\theta + \sin\phi\sin\Psi \\ \sin\Psi\cos\theta & \sin\Psi\sin\theta\sin\phi + \cos\Psi\cos\phi & \sin\phi\sin\Psi\cos\phi - \sin\theta\cos\Psi \\ -\sin\theta & \cos\theta\ \sin\phi & \cos\theta\cos\phi \end{pmatrix} \quad (1.14)$$

Let $[\vec{X}, \vec{Y}, \vec{Z}]$ an orthonormal base constitute a fixed reference. If the quad copter undergoes three successive rotations according to the aeronautical angles one, then has:

$$r_{x,y,z}(x,y,z) = R_{(\phi,\theta,\Psi)} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (1.15)$$

So the coordinates as a function of **t** become:

(For simplification the entries ϕ (t), θ (t) and Ψ (t) will be denoted ϕ, θ and Ψ)

$$r_x(x,y,z) = (\cos\Psi\cos\theta)x + (\cos\Psi\sin\theta\sin\phi - \cos\phi\sin\Psi)y + (\cos\phi\cos\Psi\sin\theta + \sin\phi\sin\Psi)z \quad (1.16)$$

$$r_y(x,y,z) = (\sin\Psi\cos\theta)x + (\sin\Psi\sin\theta\sin\phi + \cos\Psi\cos\phi)y + (\sin\phi\sin\Psi\cos\phi - \sin\theta\cos\Psi)z \quad (1.17)$$

$$r_z(x,y,z) = (-\sin\theta)\,x + (\cos\theta\ \sin\phi)y + (\cos\theta\cos\phi)z \quad (1.18)$$

By derivation we obtain the corresponding speeds

$$v_x(x,y,z) = V_{xx}X + V_{xy}Y + V_{xz}Z = (V_{xx} \quad V_{xy} \quad V_{xz}) \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad (1.19)$$

$$v_Y(x,y,z) = V_{yx}X + V_{yy}Y + V_{yz}Z = (V_{yx} \quad V_{yy} \quad V_{yz}) \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad (1.20)$$

$$v_z(x,y,z) = V_{zx}X + V_{zy}Y + V_{zz}Z = (V_{zx} \quad V_{zy} \quad V_{zz}) \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad (1.21)$$

So the square of the standard of speed is:

$$V^2(x,y,z) = V_x^2 + V_y{}^2 + V_z^2 \quad (1.22)$$

$$V^2(x,y,z) = (V_{xx} \quad V_{xy} \quad V_{xz})\, A \begin{pmatrix} V_{xx} \\ V_{xy} \\ V_{xz} \end{pmatrix} + (V_{yx} \quad V_{yy} \quad V_{yz}) A \begin{pmatrix} V_{yx} \\ V_{yy} \\ V_{yz} \end{pmatrix} +$$

$$(V_{zx} \quad V_{zy} \quad V_{zz}) A \begin{pmatrix} V_{zx} \\ V_{zy} \\ V_{zz} \end{pmatrix} \quad (1.23)$$

With: $A = \begin{pmatrix} x^2 & xy & xz \\ yx & y^2 & yz \\ zx & zy & z^2 \end{pmatrix}$ (1.24)

I-6 Expression of kinetic energy:

Kinetic energy can be expressed as a function of mass and the square of the speed, like:

$$T = \frac{1}{2} m v^2 \quad (1.25)$$

Or in more detail and assuming that the system is perfectly symmetrical, and thus the inertia products are null and that the inertia matrix of the quad copter is diagonal, simplify the equation of kinetic energy:

$$T = \frac{1}{2} I_x \left(\dot{\phi} - \dot{\Psi} \sin\theta\right)^2 + \frac{1}{2} I_y \left(\dot{\theta} \cos\phi + \dot{\Psi} \sin\phi \cos\theta\right)^2 + \frac{1}{2} I_z \left(\dot{\theta} \sin\phi - \dot{\Psi} \cos\phi \cos\theta\right)^2 \quad (1.26)$$

With

$$I_x = \frac{1}{2}\int (y^2 + z^2) dm,\ I_y = \frac{1}{2}\int (x^2 + z^2) dm,\ I_z = \frac{1}{2}\int (y^2 + x^2) dm \quad (1.27)$$

I-7 Expression of potential energy

$$V = g \int (-\sin\theta\, x + \sin\phi \cos\theta\, y + \cos\phi \cos\theta\, z) dm \quad (1.28)$$

$$V = g \int x dm(-g \sin\theta) + \int y dm(g \sin\phi \cos\theta) + \int z dm\, (\cos\phi \cos\theta) \quad (1.29)$$

The equations of movements are then given by:

Roll equation: $\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\boldsymbol{\phi}}}\right) - \left(\frac{\partial L}{\partial \boldsymbol{\phi}}\right) = \tau_{\boldsymbol{\phi}}$ (1.30)

Equation of pitch: $\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \left(\frac{\partial L}{\partial \theta}\right) = \tau_{\theta}$ (1.31)

Equation of the lace: $\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\boldsymbol{\Psi}}}\right) - \left(\frac{\partial L}{\partial \boldsymbol{\Psi}}\right) = \tau_{\boldsymbol{\Psi}}$ (1.32)

With

$\boldsymbol{\tau}$: generalized forces given by the non-conservative forces.

For the roll:

$$\frac{\mathrm{d}}{\mathrm{dt}}\left(\frac{\partial \mathrm{L}}{\partial \dot{\phi}}\right) - \left(\frac{\partial \mathrm{L}}{\partial \phi}\right) = \ddot{\phi}\mathrm{I} - \ddot{\Psi}\sin \mathrm{I_x} - \dot{\Psi}\dot{\theta}\cos\theta\left(\mathrm{I_x} + (\mathrm{I_y} - \mathrm{I_z}(2\cos\phi^2 - 1))\right) + \dot{\theta^2}\frac{1}{2}\sin 2\phi(\mathrm{I_y} - \mathrm{I_z}) - \dot{\Psi}^2\frac{1}{2}\sin 2\phi\cos\theta^2(\mathrm{I_y} - \mathrm{I_z}) + \int \mathrm{ydm}\,(-\mathrm{gcos}\,\phi\cos\theta) + \mathrm{zdm}\ (\mathrm{g}\sin\phi\cos\theta) \quad (1.33)$$

For pitching:

$$\frac{\mathrm{d}}{\mathrm{dt}}\left(\frac{\partial \mathrm{L}}{\partial \dot{\theta}}\right) - \left(\frac{\partial \mathrm{L}}{\partial \theta}\right) = \ddot{\theta}(\mathrm{I_y}\cos^2\phi + \mathrm{I_z}\sin^2\phi) + \ddot{\Psi}\frac{1}{2}\sin 2\phi\cos\theta(\mathrm{I_y} - \mathrm{I_z}) + \dot{\Psi}^2\frac{1}{2}\sin 2\theta(-\mathrm{I_x} + \mathrm{I_y}\sin^2\phi + \mathrm{I_z}\cos^2\phi\ )\dot{\phi}\dot{\theta}\sin 2\phi(\mathrm{I_z} - \mathrm{I_z}) + \dot{\Psi}\dot{\phi}\cos\theta\,(\cos 2\theta\,(\mathrm{I_y} - \mathrm{I_z})\mathrm{I_x}) + \int \mathrm{xdm}(-\mathrm{g}\cos\theta) - \int \mathrm{ydm}(\mathrm{g}\sin\phi\sin\theta) - \int \mathrm{zdm}\,(\cos\phi\sin\theta) \quad (1.34)$$

For the lace:

$$\left(\frac{\partial \mathrm{V}}{\partial \dot{\Psi}}\right) = 0 \quad (1.35)$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\Psi}}\right) = \Psi\left(cos^2\theta\left(\mathrm{I_z}\cos^2\phi + \mathrm{I_y}\sin^2\phi\right) + \sin^2\theta\mathrm{I_x}\right) - \phi\sin\theta\mathrm{I_x} + \theta\,\frac{1}{2}\sin 2\phi\cos\theta(\,\mathrm{I_y} - \mathrm{I_z}\,) + \Psi\theta\sin 2\theta\left(\mathrm{I_x} - \mathrm{I_z}\cos^2\phi + \mathrm{I_y}\sin^2\phi\right) - \Psi\phi\sin 2\phi\cos^2\theta(\mathrm{I_y} - \mathrm{I_z}\,) + \theta\phi\cos\theta(\mathrm{I_x} + (2\cos^2\phi - 1)\left(\mathrm{I_y} - \mathrm{I_z}\right) - \theta\frac{1}{2}\,\sin 2\phi\,\sin\theta(\mathrm{I_y} - \mathrm{I_z}\,) \quad (1.36)$$

I-8 Expression of non-conservative forces

• **The lift:** The motors create torques in direction X and Y axes (Figure 1.10).

$$\tau_x = bl\,(\Omega^2_4 - \Omega^2_2),\ \tau_y = bl(\Omega^2_3 - \Omega^2_1) \quad (2.24) \qquad (1.37)$$

With:

-**b**: is the constant uniting the thrust and rotational speed of an engine and is the half span of the quad copter.

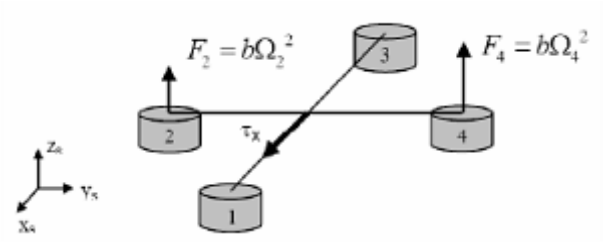


Figure 1.9- Rotations due to Lift of rolling motors

• **The drag**: The propellers create a vertical torque (Figure 1.11):

$$\tau_z = d((\Omega^2_3 + \Omega^2_1) - (\Omega^2_4 + \Omega^2_2)) \qquad (1.38)$$

-**d**: is a constant connecting the drag and speed of an engine.

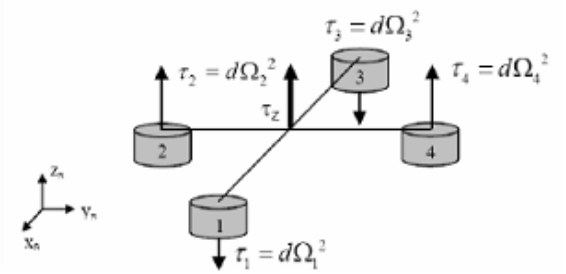


Figure 1. 10- Yaw rotations due to propeller drag.

• **The gyroscopic effects** (Figure 1.12) are due to the propellers during a rotation around of the X and Y axis of the solid reference [27]:

$$\tau_y = I_{rotor}\omega_x\left(-\Omega^2_1 - \Omega^2_3 + \Omega^2_2 + \Omega^2_4\right) \qquad (1.39)$$

$$\tau_x = I_{rotor}\omega_y\left(\Omega^2_1 + \Omega^2_3 - \Omega^2_2 - \Omega^2_4\right) \qquad (1.40)$$

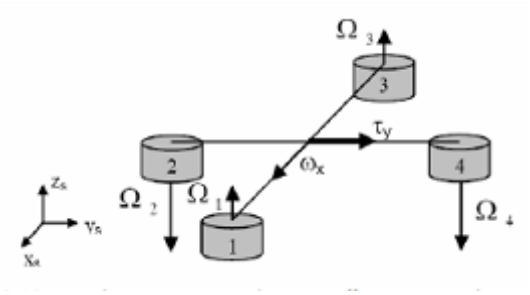


Figure 1.11- Pitch rotations due to gyroscopic effects of propellers.

Adding all the torques related to the different effects we obtain:

$$\tau_y = I_{rotor}\omega_x\left(-\Omega^2{}_1 - \Omega^2{}_3 + \Omega^2{}_2 + \Omega^2{}_4\right) + bl(\Omega^2{}_4 - \Omega^2{}_2) \quad (1.41)$$

$$\tau_x = I_{rotor}\omega_y\left(\Omega^2{}_1 + \Omega^2{}_3 - \Omega^2{}_2 - \Omega^2{}_4\right) + bl(\Omega^2{}_3 - \Omega^2{}_1) \quad (1.42)$$

$$\tau_z = d(\Omega^2{}_3 + \Omega^2{}_1 - \Omega^2{}_4 - \Omega^2{}_2) \quad (1.43)$$

Calculation of coefficients b and d:

Calculation of b:

$$F_i = b\Omega_i^2 \quad (1.44)$$

With:

F: is the force generated by the motor propeller set, expressed in Newton.

Ω : is the rotational speed of the helix expressed in radians per second.

Calculation of d:

For the calculation of the drag coefficient, the following experiment must be performed:

The quad copter is placed on a support allowing free rotation along the axis vertical (Z) (yaw movement only), then a known power set point is entered on two of the four engines (here engine one and three) then we set a timer to know the duration made by the drone to perform a quarter turn ($\boldsymbol{\Psi = \frac{\pi}{2}}$).

The relationship between yaw angle and engine speed is as follows:

$$\ddot{\Psi} = \frac{d(\Omega^2{}_3+\Omega^2{}_1-\Omega^2{}_4-\Omega^2{}_2)}{I_z} \quad (1.45)$$

Due to the assembly, the only possible movement is the lace. We thus obtain the following relation:

$$\frac{I_x-I_y}{I_z}\dot{\phi}\dot{\Theta} = 0 \quad (1.46)$$

By integrating this function twice, we get the law of the yaw angle:

$$\Psi = \frac{d(\Omega_1^2+\Omega_3^2-\Omega_2^2-\Omega_4^2)}{I_z}\frac{t^2}{2} \quad (1.47)$$

For the experiment, it is necessary to put the engines one and three at half-power and the engines two and four at the stop. One quarter turn following **z**. The measure of **t** is when the speed of rotation following z is constant: $\frac{\pi}{2} = \frac{d2\Omega^2}{I_z}\frac{t^2}{2} => d = \frac{\pi I_z}{2\Omega^2 t^2}$ (1.48)

The inertial effect of the system is related to the interactions of the movements.

The controls of the roll and pitch movements intervene on the lace (yaw) through the term of inertia. Similarly, the yaw and the roll act on the pitch due to inertial effects and gyroscopic effects.

In the same way the yaw and the pitch act on the roll, so the three rotations are clearly coupled $\boldsymbol{\omega}$ is the angular velocity.

$$\boldsymbol{\Gamma_\phi = I_x\dot{\omega}_x + (I_z - I_y)\omega_y\omega_z} \quad (1.49)$$

$$\boldsymbol{\Gamma_\theta = I_y\dot{\omega}_y + (I_x - I_z)\omega_x\omega_z} \quad (1.50)$$

$$\boldsymbol{\Gamma_\Psi = I_z\dot{\omega}_z + (I_y - I_x)\omega_x\omega_y} \quad (1.51)$$

If one adds up the accelerations of the thrust, drag, the effect of inertia, of the gyroscopic effect. Applying the approximation of small angles, in the solid reference arrives at the following three equations:

$$\ddot{\Psi} = \frac{d(\Omega_1^2 + \Omega_3^2 - \Omega_2^2 - \Omega_4^2)}{I_z} + \frac{(I_x + I_y)}{I_z}\dot{\phi}\dot{\Theta} \quad (1.52)$$

$$\ddot{\phi} = \frac{I_{rotor}\ \dot{\Theta}(\Omega_1 + \Omega_3 - \Omega_2 - \Omega_4)}{I_x} + \frac{(I_y - I_z)}{I_x}\dot{\Psi}\dot{\Theta} + \frac{bl(\Omega_4^2 - \Omega_2^2)}{I_x} \quad (1.53)$$

$$\ddot{\Theta} = \frac{-I_{rotor}\ \dot{\phi}\ (\Omega_1 + \Omega_3 - \Omega_2 - \Omega_4)}{I_y} + \frac{(I_z - I_x)}{I_y}\dot{\Psi}\dot{\phi} + \frac{bl(\Omega_3^2 - \Omega_1^2)}{I_y} \quad (1.54)$$

These three relations give the angular accelerations along the X, Y and Z axes of the ground according to the speed of each propeller motor torque. Through these three relationships and the motors dynamics we can design a quad copter drone model under Simulink.

I-9-**Development of the mathematical model according to Newton-Euler:** [1]

Using the Newton-Euler formulation, the equations are written in the following form:

$$\begin{cases} \dot{X} = V, \quad \dot{R} = RS(\Omega) \\ m\dot{V} = W + F_P + F_t + F_{disturbance} \\ J\dot{\Omega} = -\Omega \wedge \Omega J + M_a - M_f - M_{gm} \end{cases} \quad (1.55)$$

With:

X: is the position vector of the quad copter.

m: the total mass of the quad copter.

Ω : The angular velocity expressed in the fixed reference.

R: The rotation matrix.

$\wedge$: Vector product.

J: Symmetric inertia matrix of dimension (3x3), it is given by: $J = \begin{bmatrix} I_X & 0 & 0 \\ 0 & I_Y & 0 \\ 0 & 0 & I_Z \end{bmatrix}$

S(Ω): is the asymmetric matrix for a velocity vector: $\Omega = \begin{bmatrix} \Omega_1 \\ \Omega_2 \\ \Omega_3 \end{bmatrix}$,given by:

$$S(\Omega) = \begin{bmatrix} 0 & -\Omega_3 & \Omega_2 \\ \Omega_3 & 0 & -\Omega_1 \\ -\Omega_2 & \Omega_1 & 0 \end{bmatrix}$$

F_P: is the total force generated by the four rotors, it is given by:

$$\boldsymbol{F_P = R \times \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} b\omega_i^2 \end{bmatrix}}$$

F_t : The drag force along the axes (x, y, z), it is given by:

$$\boldsymbol{F_t = \begin{bmatrix} -K_{ftx} & 0 & 0 \\ 0 & -K_{fty} & 0 \\ 0 & 0 & -K_{ftz} \end{bmatrix} \dot{X}}$$

$\boldsymbol{K_{ftx}, K_{fty}, K_{ftz}}$: Translational drag coefficients.

W: force of gravity, it is given by:

$$W = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix}$$

M_a: Moment caused by push and drag forces.

$$\mathrm{M_a} = \begin{bmatrix} \mathrm{bl}(\Omega^2_4 - \Omega^2_2) \\ \mathrm{bl}(\Omega^2_3 - \Omega^2_1) \\ \mathrm{d}(\Omega^2_3 + \Omega^2_1 - \Omega^2_4 - \Omega^2_2) \end{bmatrix}$$

M_f: Moment resulting from aerodynamic friction, and is given by:

$$\mathrm{M_f} = \begin{bmatrix} \boldsymbol{K_{fx}\Omega_1} \\ \boldsymbol{K_{fy}\Omega_2} \\ \boldsymbol{K_{fz}\Omega_3} \end{bmatrix} \text{ Eq.I.5.34}$$

With:$\boldsymbol{K_{fx}}$, $\boldsymbol{K_{fy}}$, $\boldsymbol{K_{fz}}$, The coefficients of aerodynamic friction.

Translational motion equations:

By neglecting the disturbance force we will have:

$$m\dot{V} = W + F_p + F_t \tag{1.56}$$

We replace each force by its formula, we find:

$$m\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} \cos\phi\, sin\theta \sin\Psi + sin\Psi \sin\phi \\ \cos\phi\, sin\theta \sin\Psi - \cos\Psi \sin\phi \\ (\cos\phi\, cos\theta) \end{bmatrix} \sum_{i=1}^{4} F_i - \begin{bmatrix} K_{ftx}\dot{x} \\ K_{fty}\dot{y} \\ K_{ftz}\dot{z} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{1.57}$$

We then obtain the differential equations that define the translation movement

$$\begin{cases} \ddot{x} = \frac{1}{m}(\cos\phi\, sin\theta \sin\Psi + \sin\Psi \sin\phi)\sum_{i=1}^{4} F_i - \frac{K_{ftx}}{m}\dot{x} \\ \ddot{y} = \frac{1}{m}(\cos\phi\, sin\theta \sin\Psi - \cos\Psi \sin\phi)\sum_{i=1}^{4} F_i - \frac{K_{fty}}{m}\dot{y} \\ \ddot{z} = \frac{1}{m}(\cos\phi\, cos\theta)\sum_{i=1}^{4} F_i - \frac{K_{ftz}}{m}\dot{z} - g \end{cases} \quad (1.58)$$

Rotational motion equations:

We have:

$$\boldsymbol{J\dot{\Omega}} = -\boldsymbol{\Omega} \wedge \boldsymbol{\Omega J} + \boldsymbol{M_a} - \boldsymbol{M_f} - \boldsymbol{M_{gm}} \quad (1.59)$$

We replace each moment by the corresponding formula, we find:

$$\begin{bmatrix} I_X & 0 & 0 \\ 0 & I_Y & 0 \\ 0 & 0 & I_Z \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{\phi}} \\ \ddot{\theta} \\ \ddot{\boldsymbol{\Psi}} \end{bmatrix} = -\begin{bmatrix} \dot{\boldsymbol{\phi}} \\ \dot{\theta} \\ \dot{\boldsymbol{\Psi}} \end{bmatrix} \wedge \begin{bmatrix} I_X & 0 & 0 \\ 0 & I_Y & 0 \\ 0 & 0 & I_Z \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{\phi}} \\ \dot{\theta} \\ \dot{\boldsymbol{\Psi}} \end{bmatrix} +$$

$$\begin{bmatrix} bl(\Omega^2_4 - \Omega^2_2) \\ bl(\Omega^2_3 - \Omega^2_1) \\ d(\Omega^2_3 + \Omega^2_1 - \Omega^2_4 - \Omega^2_2) \end{bmatrix} - \begin{bmatrix} \boldsymbol{K_{fx}\dot{\phi}^2} \\ \boldsymbol{K_{fy}}\dot{\theta}^2 \\ \boldsymbol{K_{fz}\dot{\Psi}^2} \end{bmatrix} - \begin{bmatrix} \overline{\Omega_r}\, J_r\dot{\theta} \\ J_r\overline{\Omega_r} \quad \dot{\phi} \\ 0 \end{bmatrix}$$

We then obtain the differential equations defining the rotational movement:

$$I_X\ddot{\phi} = -\dot{\Psi}\dot{\phi}(I_Z - I_Y) - \overline{\Omega_r}\, J_r\dot{\theta} - K_{fx}\dot{\phi}^2 + bl(\Omega^2_4 - \Omega^2_2) \quad (1.61)$$

$$I_y\ddot{\theta} = -\dot{\Psi}\dot{\phi}(I_Z - I_x) - J_r\overline{\Omega_r}\dot{\phi} - K_{fy}\dot{\theta}^2 + bl(\Omega^2_3 - \Omega^2_1) \quad (1.62)$$

$$I_Z\ddot{\Psi} = -\dot{\Psi}\dot{\phi}(I_x - I_y) - K_{fz}\dot{\Psi}^2 + d(\Omega^2_3 + \Omega^2_1 - \Omega^2_4 - \Omega^2_2) \quad (1.63)$$

With:

$$\overline{\Omega_r} = \omega_1 + \omega_3 - \omega_2 - \omega_4 \quad (1.64)$$

As a result, the complete dynamic model governing the quad copter is as follows:

$$\begin{cases} \ddot{\phi} = -\dot{\Psi}\dot{\phi}\left(\frac{I_Z - I_Y}{I_x}\right) - \overline{\Omega_r}\frac{J_r}{I_x}\dot{\theta} - \frac{K_{fx}}{I_x}\dot{\phi}^2 + \frac{l}{I_x}u_2 \\ \ddot{\theta} = -\dot{\Psi}\dot{\phi}\left(\frac{I_Z - I_x}{I_y}\right) - \frac{J_r}{I_y}\overline{\Omega_r}\dot{\phi} - \dot{\theta}^2\frac{K_{fy}}{I_y} + \frac{l}{I_y}u_3 \\ \ddot{\Psi} = -\dot{\Psi}\dot{\phi}\left(\frac{I_x - I_y}{I_z}\right) - \frac{K_{fz}}{I_z}\dot{\Psi}^2 + \frac{l}{I_z}u_4 \\ \ddot{x} = -\frac{K_{ftx}}{m}\dot{x} + \frac{1}{m}u_x u_1 \\ \ddot{y} = -\frac{K_{fty}}{m}\dot{y} + \frac{1}{m}u_y u_1 \\ \ddot{z} = -\frac{K_{fyz}}{m}\dot{z} - g + \frac{cos\phi\cos\theta}{m}u_1 \end{cases} \quad (1.65)$$

$$\begin{cases} u_x = \cos\boldsymbol{\phi}\sin\theta\, cos\boldsymbol{\Psi} + \sin\boldsymbol{\Psi}\sin\boldsymbol{\phi} \\ u_y = \cos\boldsymbol{\phi}\sin\theta\, cos\boldsymbol{\Psi} + \cos\boldsymbol{\Psi}\sin\boldsymbol{\phi} \end{cases} \quad (1.66)$$

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} b & b & b & b \\ 0 & -lb & 0 & lb \\ -lb & 0 & lb & 0 \\ d & -d & d & -d \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \quad (1.67)$$

I-10 Study of engine dynamics:

We present an equivalent circuit to the DC motor that has a well-known model that connects electrical and mechanical quantities. This model is composed of a resistance **R [Ohm],** an inductance **L [H]** and a voltage generator **e [V].** Resistance represents the losses of Joule due to the flow of current in the copper conductor. Its value depends on the geometry and the material characteristics such as the resistivity of the wire, the length and the section. The behavior of the inductor derives from the shape of the motor wires that are wound up to middle of the rotor. Finally, the generator e provides a voltage called the electromotive force (E.M.F) back, proportional to the speed of the motor. The model is represented in the figure: Figure I.13 and Figure I.14 [6]

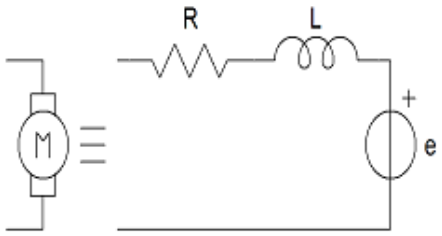


Figure1. 12- Equivalent electric model of the DC motor**. [6]**

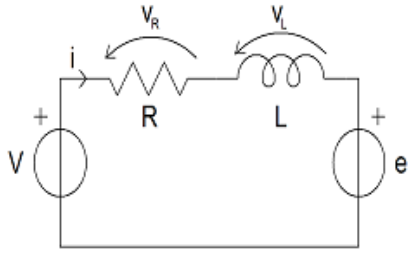


Figure1.13- DC motor equivalent circuit. [6]

Applying Kirchhoff's law, the following equation is obtained:

$$V_S = Ri + L\frac{di}{dt} + e \tag{1.68}$$

At steady state (state whose DC current is zero frequency)

$$V_S = Ri + L\frac{di}{dt} \tag{1.69}$$

Therefore, for the transient state, Eq.I.69 is rearranged to have the e.m.f, as shown in the equation below: [7]

$$e = -Ri - L\frac{di}{dt} + V_S \tag{1.70}$$

With:

Vs: Voltage source.

i: frame current.

Similarly, taking into account the mechanical properties of the DC motor, and from Newton's second law on motion:

$$J\frac{d\omega_m}{dt} = \sum T_i \tag{1.71}$$

Or

$$T_e = k_f\omega_m + J\frac{d\omega_m}{dt} \tag{1.72}$$

With:

$\boldsymbol{T_e}$: The electric torque.

$\boldsymbol{k_f}$: The friction constant.

$\boldsymbol{J}$: Inertia of the rotor.

$\boldsymbol{\omega_m}$: The angular velocity.

The electric torque and the e.m.f could be written as: [7]

$$e = k_e\omega_m \quad and \quad T_e = k_t\omega_m \tag{1.73}$$

With:

$\boldsymbol{k_e}$: The constant of the return e.m.f.

$\boldsymbol{k_t}$: The constant of the electric torque.

Replacing in Eq I.7.3 and Eq I.7.4 we find the following equations:

$$\frac{di}{dt} = -i\frac{R}{L} - \frac{k_e}{L}\omega_m \ + \frac{1}{L}\omega_m \tag{1.74}$$

$$\frac{d\omega_m}{dt} = i\frac{k_t}{J} - \frac{k_f}{L}\omega_m \tag{1.75}$$

In the rest of this work, the Laplace transform is used to evaluate the two equations Eq.I.74 and Eq.I.75, (all initial conditions are assumed to be zero). [7]

Applying the Laplace transform to Eq I.74 and Eq I.75:

$$si = -i\frac{R}{L} - \frac{k_e}{L}\omega_m \ + \frac{1}{L}V_s \tag{1.76}$$

$$s\omega_m = i\frac{k_t}{J} - \frac{k_f}{L}\omega_m \tag{1.77}$$

From Eq I.7.77 we will have the following equation:

$$i = \frac{s\omega_m + \frac{k_f}{L}\omega_m}{\frac{k_t}{J}} \tag{1.78}$$

We replace the ***i*** in the Eq I.76 with the relation found in the Eq I.78 one finds:

$$\left(\frac{s\omega_m + \frac{k_f}{L}\omega_m}{\frac{k_t}{J}}\right)\left(\frac{R}{L} + s\right) = -\frac{k_e}{L}\omega_m \ + \frac{1}{L}V_s \tag{1.79}$$

Finally, Eq.I.79 becomes:

$$V_s = \left(\frac{s^2JL + sk_fL + sRJ + k_fR + k_ek_t}{k_t}\right)\omega_m \tag{1.80}$$

The transfer function is thus obtained by using the ratio between the angular velocity and voltage source.

$$G(s) = \frac{\omega_m}{V_s} = \left(\frac{k_t}{s^2JL + sk_fL + sRJ + k_fR + k_ek_t}\right) \tag{1.81}$$

Considering the following assumptions: [7]

- The friction constant is low and tends to zero, this implies

$\rightarrow RJ \gg k_fL$

$\rightarrow k_ek_t \gg k_fR$

The transfer function can be written:

$$G(s) = \left(\frac{\frac{1}{k_e}}{1+\frac{RJ}{k_e k_t}s+\frac{L}{Rk_e k_t}\frac{RJ}{s^2}} \right) \quad (1.82)$$

From Eq I.82, the following time constants are acquired:

→ The magnetic time constant:

$$\tau_m = \frac{RJ}{k_e k_t} \quad (1.83)$$

→ The electric time constant: $\tau_e = \frac{L}{R}$

Finally, replacing the time constant in Eq I.82, we obtain:

$$G(s) = \left(\frac{\frac{1}{k_e}}{\tau_m \tau_e s^2+\tau_m s+1} \right) \quad (1.85)$$

The Eq I.85 represents the transfer function between the motor input which is the e.m.f and its output which represents the speed. In the next chapter we will use these equations, in order to build the SIMULINK model of the quad copter.

I-11 Conclusion:

In this chapter a description of the operating principle of the drone (quad copter) is presented with a detailed description about its possible motions as well as flight modes, together with jargon (pitch, roll, and yaw) of correct flight, which designates the input signals and their corresponding output responses.

Based on the laws of kinematics (Euler / Lagrange), this chapter presents a dynamic modeling of the quad copter drone. The use of Newton-Euler's formalism allowed us to establish the dynamic model of a quad copter. In addition, the complexity of the model and the non-linearity can be clearly seen.

Knowing the relationships between the thrusts of different engines and angular accelerations, a model can be established under MATLAB Simulink.

In the following sections, we will make simulations in open loop on the model of quad copter in order to demonstrate the instability of the open-loop system and the need for a successful closed-loop regulator that is based on a precise estimate of state [1]

The purpose is to determine the type of regulators to be put in place to ensure good stabilization of the drone on the three axes of yaw, pitch and roll.

Chapter 2

Stabilization of the drone on the three axes

2.1. Introduction:

Four commands drive the quad copter: power (permits to control the altitude) lace pitch and roll. These commands are sent to all motors as follows: power and lace commands are sent to all motors for control whereas the pitch and the roll commands are obtained controlling only two motors (1 and 3 for pitch, 2 and 4 for the roll)

The quad copter's subsystem is made by mechanical equations of the dynamic model discussed in the previous chapter. The corrections used to enhance the performance of our quad copter are the proportional integral derivative corrector (controller) and the Linear Quadratic control.

However, it is still possible to study the type of correctors to set up in order to obtain a satisfactory dynamic.

2.2. Diagram of Simulink Modeling.

Knowing the relationships between the thrusts of different engines and angular accelerations, one can establish a model under Simulink (Figure 2.1).

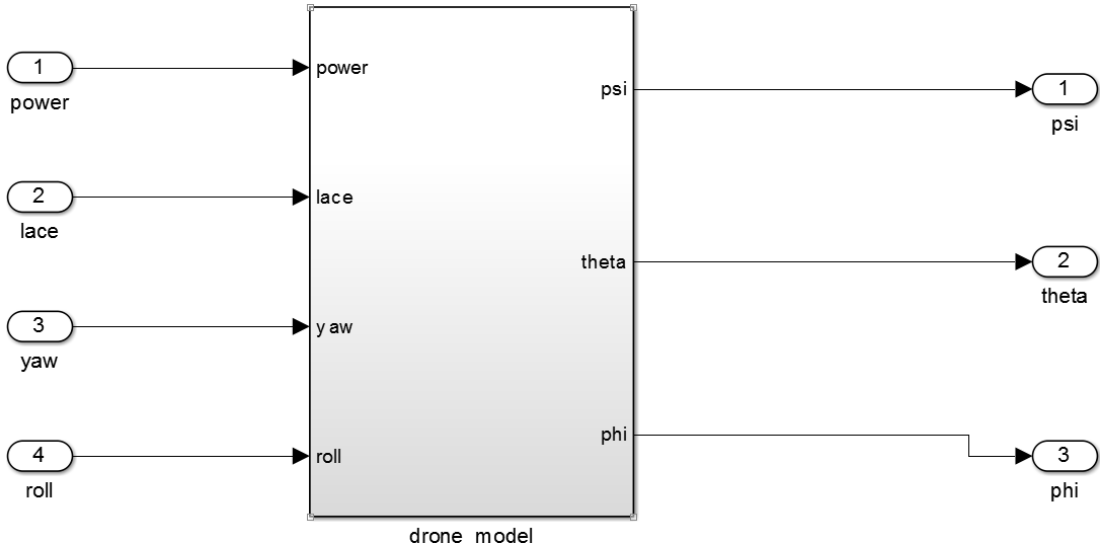


Figure 2 -1 Diagram of the Simulink model

The Quad copter subsystem models the dynamics of our drone (Figure 2.1), it includes three other subsystems:

- Push and drag.
- Gyroscopic effect.
- Inertia effect.

Blocks [R2D] whose Simulink scheme represents the transformation blocks of Radian angles to degree. Figure 2.2 Set of quad copter subsystems. Figure 2.3 shows the modeling of thrust and drag.

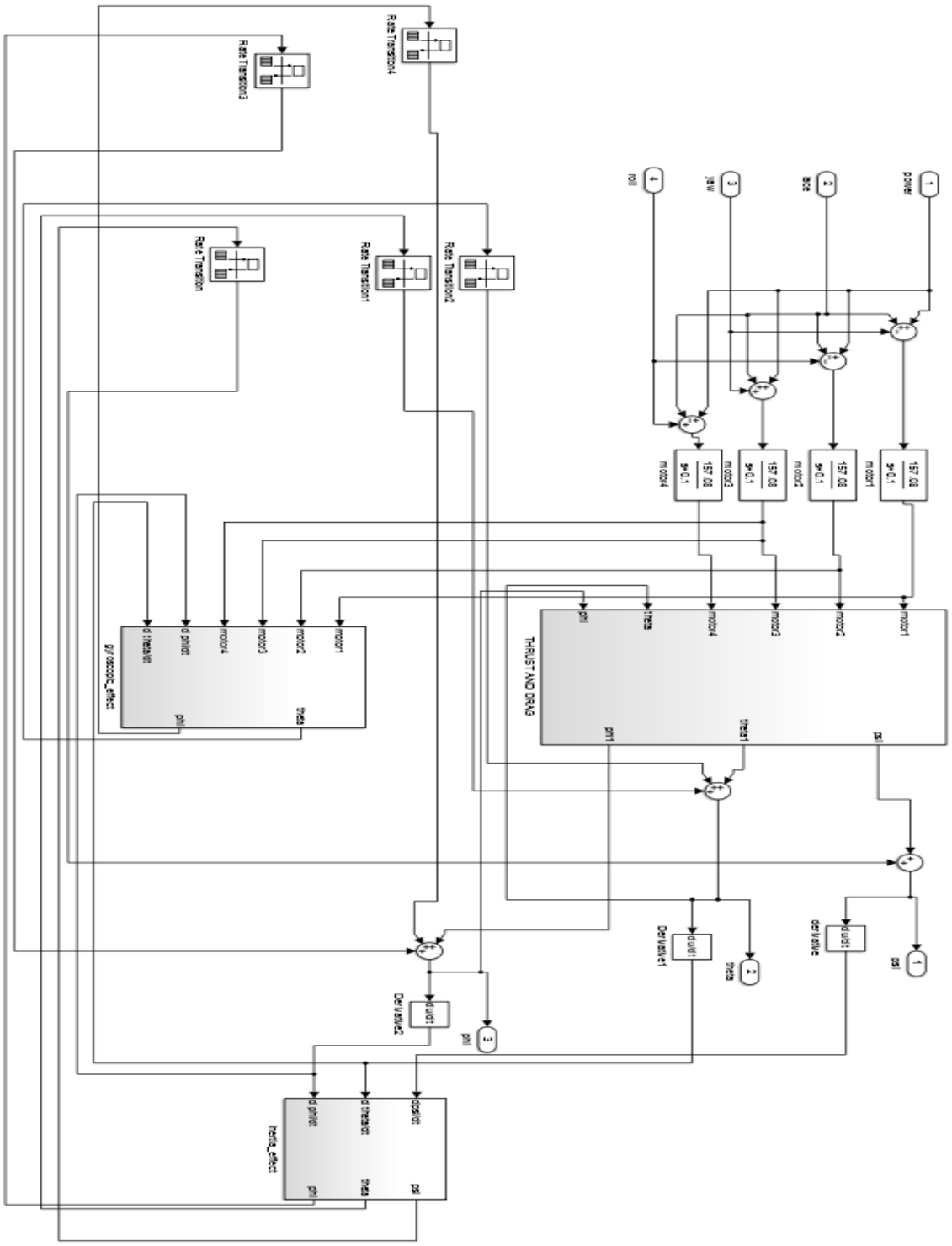


Figure 2.2 Set of quad copter subsystems under Simulink

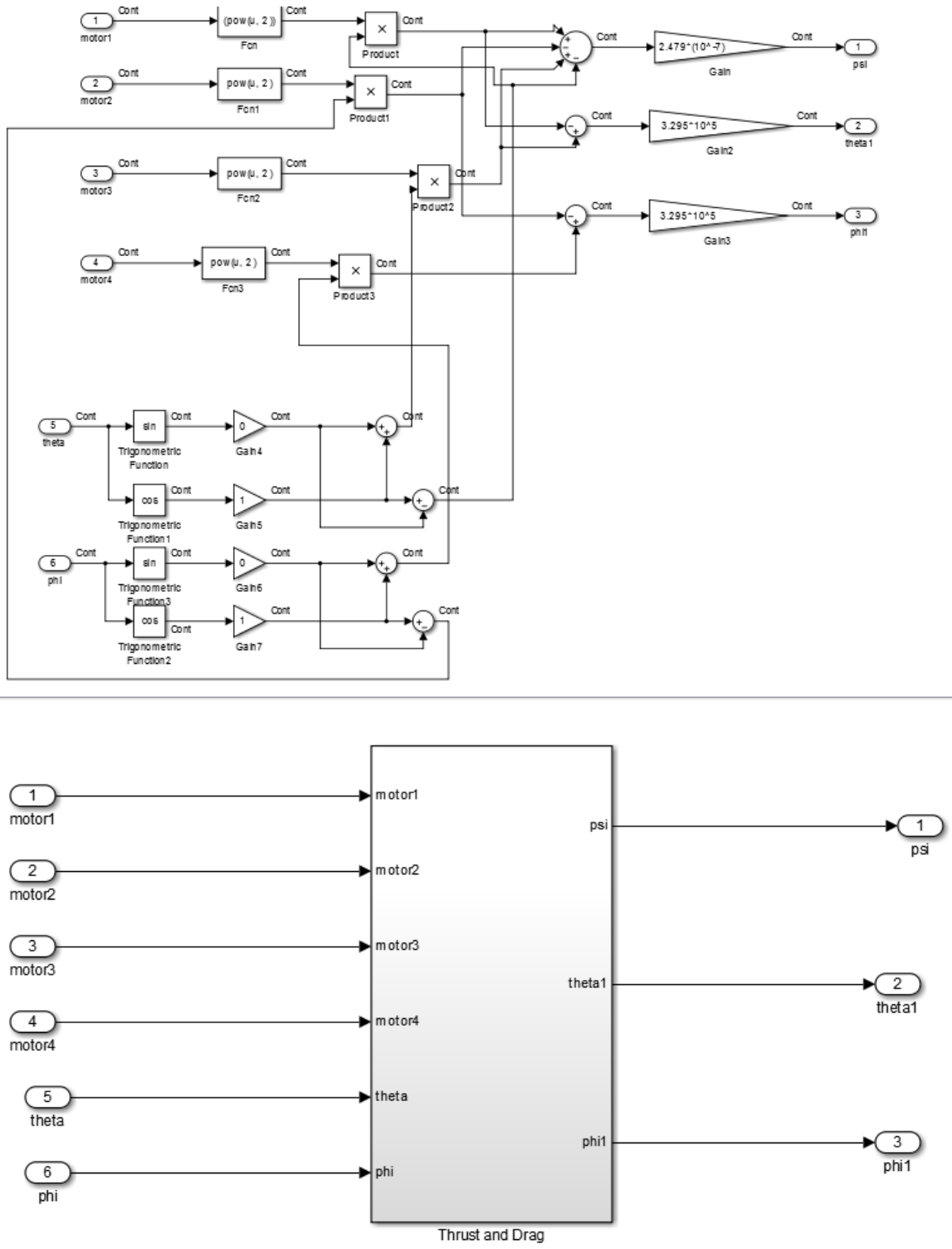


Figure2.2 Modeling of thrust and drag.

The figures (Figure 2.4) and (Figure 2.5) show us the modeling of the effect of inertia, and the gyroscopic effect, under Simulink.

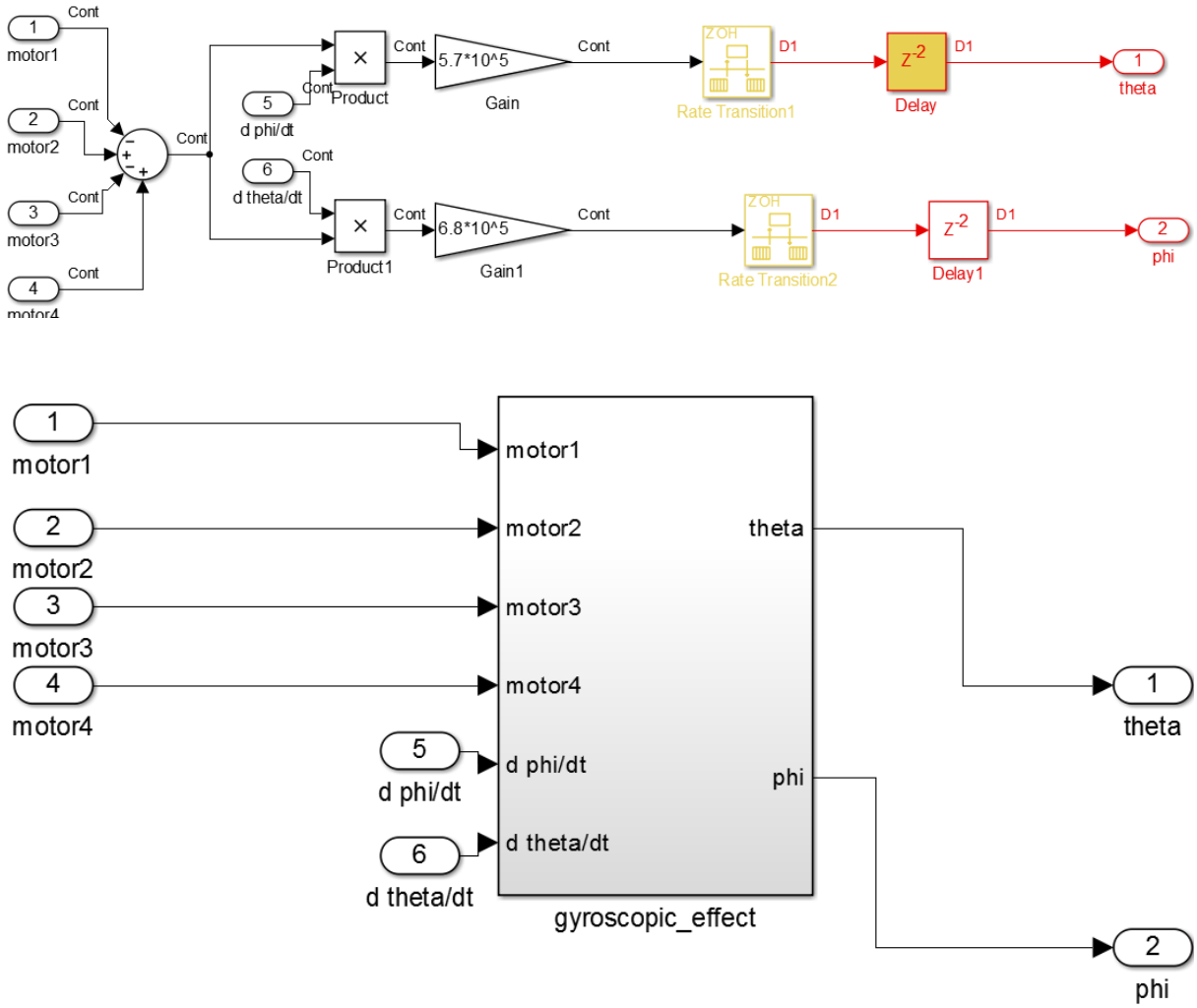


Figure2. 3 Modeling of the gyroscopic effect

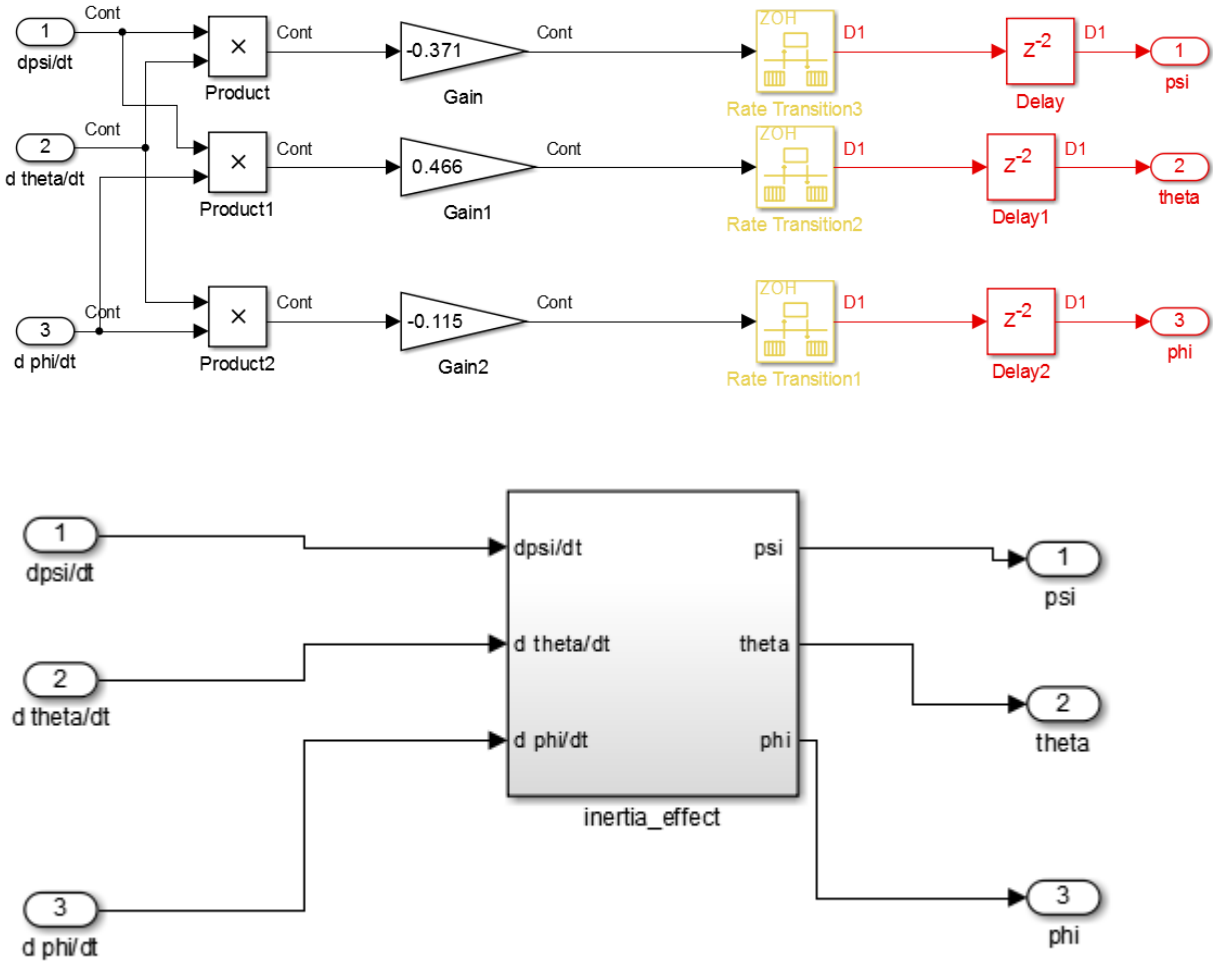


Figure2.4 Model for the inertia effect.

2.3. Quad copter state equations:

The obtained model describes the system's equations. For control study, it is necessary to rewrite our system under state equations so to implement it in a control loop.

As our system is nonlinear, we can write the dynamic model equations as: $\dot{X} = f(X, U)$

Where: U is the input vector, and X is the state vector chosen as:

$$X = \begin{bmatrix} \dot{\Phi} & \Phi & \theta & \dot{\theta} & \Psi & \dot{\Psi} & z & \dot{z} & x & \dot{x} & y & \dot{y} \end{bmatrix}^T \qquad (2.1)$$

$$\begin{aligned} x_1 &= \Phi & x_7 &= z \\ x_2 &= \dot{x}_1 = \dot{\Phi} & x_8 &= \dot{x}_7 = \dot{z} \\ x_3 &= \theta & x_9 &= x \\ x_4 &= \dot{x}_3 = \dot{\theta} & x_{10} &= \dot{x}_9 = \dot{x} \\ x_5 &= \Psi & x_{11} &= y \\ x_6 &= \dot{x}_5 = \dot{\Psi} & x_{12} &= \dot{x}_{11} = \dot{y} \end{aligned} \tag{2.2}$$

$$U = [U_1 \quad U_2 \quad U_3 \quad U_4] \tag{2.3}$$

Where:

U_1 Power command

U_2 Roll command

U_3 Pitch command

U_4 Lace command

$$\begin{cases} U_1 = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ U_2 = b(-\Omega_2^2 + \Omega_4^2) \\ U_3 = b(\Omega_1^2 - \Omega_3^2) \\ U_4 = \mathrm{d}(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \end{cases} \tag{2.4}$$

Under stationary flight conditions and neglecting disturbances during the vertical flight, we can say that the angular velocities of the quad copter are equal to the derivatives of Euler angles ($\dot{\Phi}, \dot{\theta}, \dot{\Psi}$) = (p, q, r) [24].

Based on the dynamic model developed previously, we can write:

$$f(X,U)=\begin{pmatrix} \dot{\Phi} \\ \dot{\theta}\dot{\Psi}a_1+\dot{\theta}a_2\Omega_r+bU_2 \\ \dot{\theta} \\ \dot{\Phi}\dot{\Psi}a_3-\dot{\Phi}a_4\Omega_r+b_2U_3 \\ \dot{\Psi} \\ \dot{\theta}\dot{\Phi}a_5+b_3U_4 \\ \dot{z} \\ g-(\cos\Phi\cos\theta)\frac{1}{m}U_1 \\ \dot{x} \\ u_x\frac{1}{m}U_1 \\ \dot{y} \\ u_y\frac{1}{m}U_1 \end{pmatrix} \tag{2.5}$$

With:

$$\begin{aligned} a_1&=(I_y-I_x) \\ a_2&=\frac{J_r}{I_x}b_1=\frac{l}{I_x} \\ a_3&=\frac{I_z-I_x}{I_y}b_1=\frac{l}{I_y} \\ a_4&=\frac{J_r}{I_y}b_1=\frac{1}{I_z} \\ a_5&=\frac{I_x-I_y}{I_z} \end{aligned} \tag{2.6}$$

$$\begin{aligned} U_x&=(\cos\Phi\sin\theta\cos\Psi+\sin\Phi\sin\Psi) \\ U_y&=(\cos\Phi\sin\theta\sin\Psi-\sin\Phi\cos\Psi) \end{aligned} \tag{2.7}$$

We clearly notice that Euler angles and their derivatives to time are independent of translation components whereas rotations' are dependent. One can then divide the system into two subsystems:

-Angular rotations subsystem.

-Linear translations subsystem.

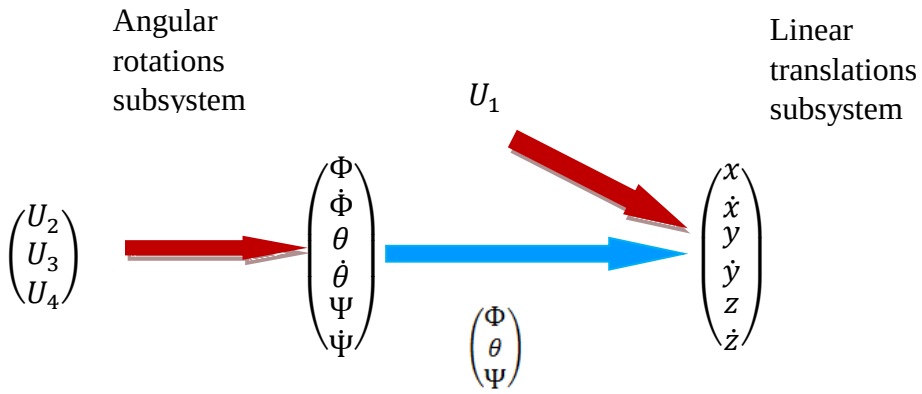


Figure 2.6 Angular rotations and linear translations system

2.4. Simulation in open loop:

We inject directly the rotation velocities to the system. To show the dynamic behavior of the system and the effect of the rotation velocities on the quad copter movements, we take the four cases below:

- First case: $\omega_1{}^2 = \omega_2{}^2 = \omega_3{}^2 = \omega_4{}^2 = 670\ rps^2$
- Second case: $\omega_1{}^2 = \omega_3{}^2 = 556\ rps^2 \ \ and \ \ \omega_2{}^2 = \omega_4{}^2 = 819\ rps^2$
- Third case: $\omega_2{}^2 = \omega_3{}^2 = 662\ \ rps^2 \ \ \ and \ \ \ \omega_1{}^2 = \omega_4{}^2 = \ 702\ rps^2$
- Fourth case: $\omega_1{}^2 = \omega_2{}^2 = 649\ rps^2\ and \ \ \ \omega_3{}^2 = \omega_4{}^2 = 714\ rps^2$

Where control inputs are:

$$\begin{cases} M_x = dU_2 \\ M_y = dU_3 \\ M_z = dU_4 \\ F_p = U_1 \end{cases} \tag{2.8}$$

$$\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} b & b & b & b \\ 0 & -lb & 0 & lb \\ -lb & 0 & lb & 0 \\ d & -d & d & -d \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \tag{2.9}$$

2.4.1. Simulation results:

1st case:

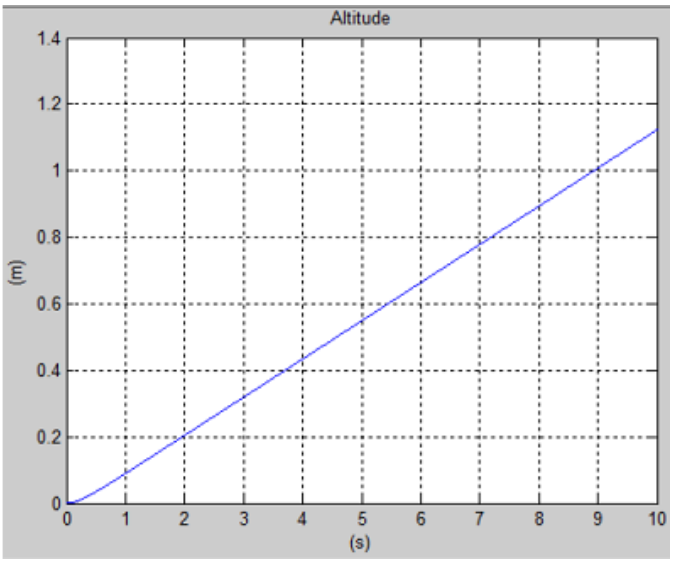


Figure 2.7- vertical movement along Z

2nd case:

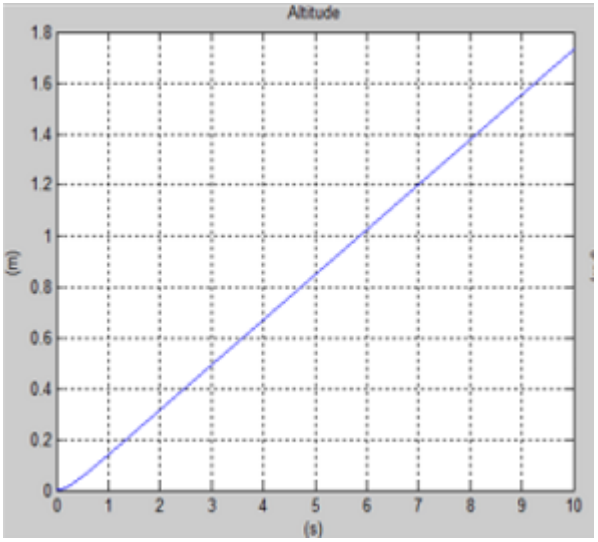


Figure 2.8.a Lace motion (altitude)

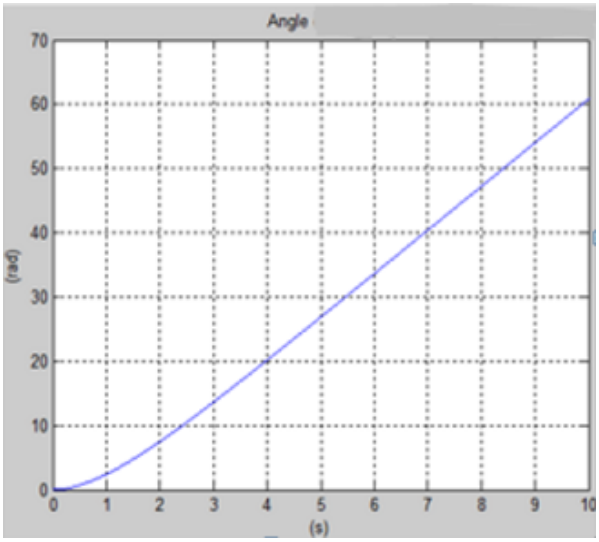


Figure 2.8.b Lace motion (angle)

3rd case:

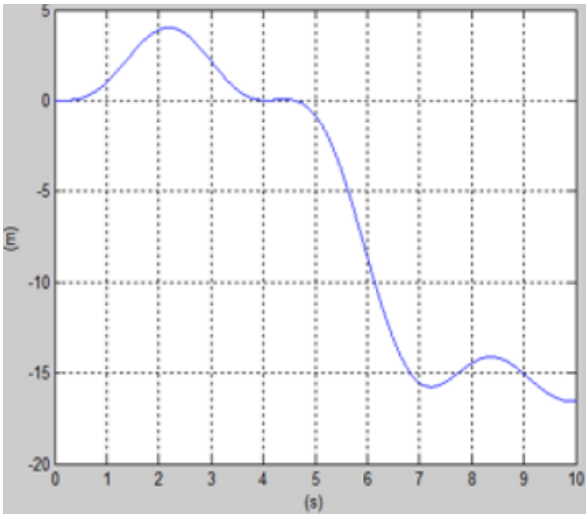


Figure 2.9.a Roll motion (altitude)

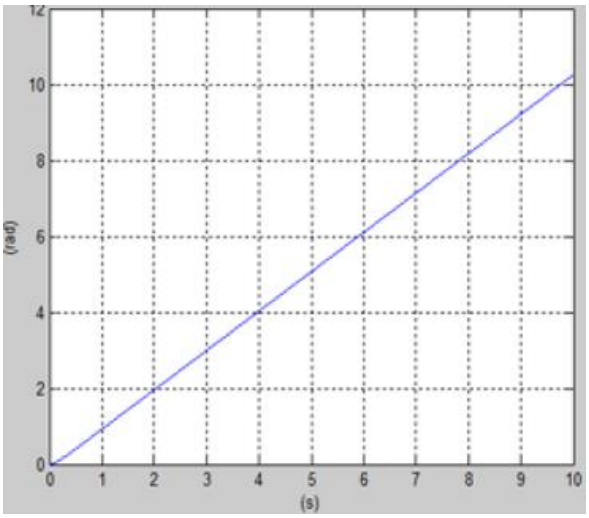


Figure 2.9.b Roll motion (angle)

4th case:

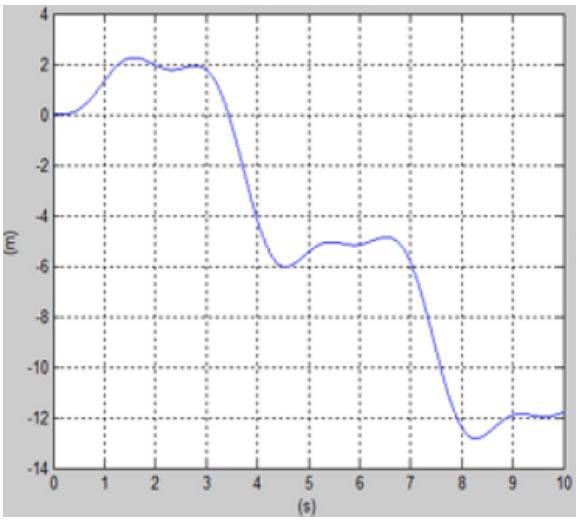


Figure 2.10.a Pitch motion (altitude)

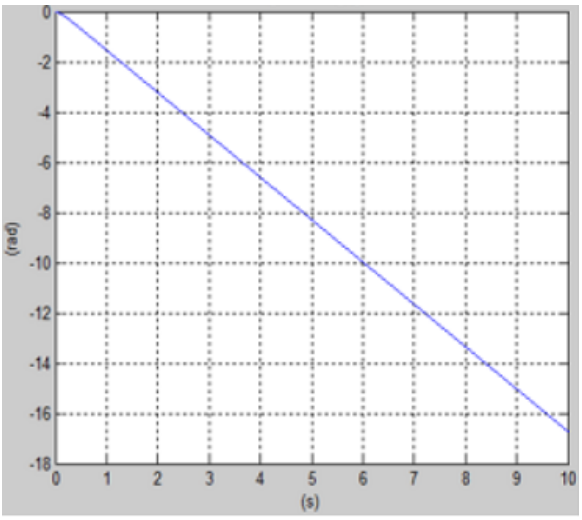


Figure 2.10.b Pitch motion (angle)

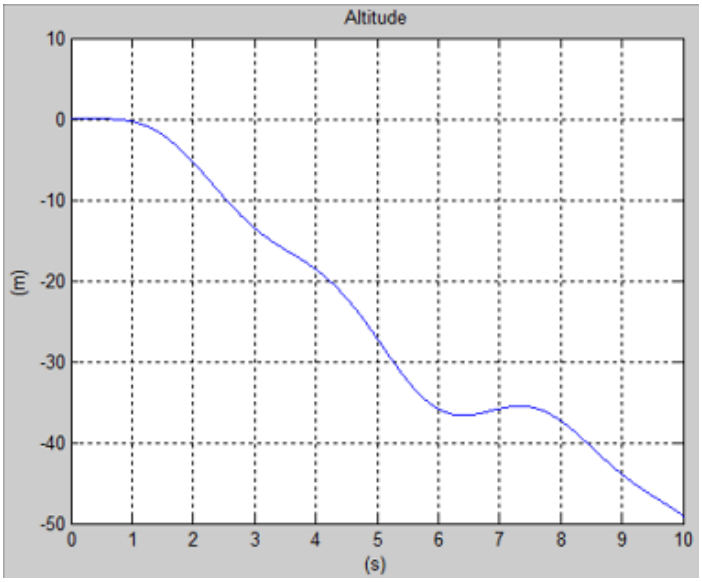


Figure 2.11 Altitude variations during Lace, Roll and Pitch motions

2.4.2. Discussion:

We notice (form Figure 2.7) that the quad copter makes a vertical translation due to lift forces created by U_1, meaning that translation along Z-axis is provoked by U_1.

We notice (from Figure 2.8) that the quad copter rotates around the Z-axis (lace motion) after varying angular velocity of rotors 1 and 3 Vs rotor 2 and 4, meaning Lace(yaw) motion is due to U_4.

We notice (from Figure 2.10) that the quad copter rotates around the Y-axis (Pitch motion), this rotation provokes a translation along X-axis, meaning that U_3 provokes the rotation around Y-axis that causes a translation along X-axis.

According to figure 2.9, we notice that the quad copter rotates around the X-axis (Roll motion), this rotation provokes a translation along Y-axis, we also have a rotation around the Y-axis making a vertical translation and thus we can say that the translation around the Y-axis is due to rotations around X-axis due to U_2 effect.

2.4.3. Specifications:

The objective of this study is to determine control parameters that satisfy constraints as follows:

1-Speed: response time < 5 sec.

2- Overshoot: no overshoot.

3- Precision: no error

4-Constraint we take the references x_r, y_r, z_r equal to '1' so to get an angle between x and z of reference ϕ_d, θ_d are defined based on the non holonomic principleφ_d equal to$\frac{\pi}{4}$, as it is shown from the scheme:

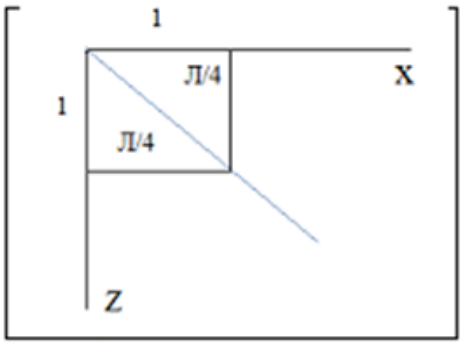


Figure 2.12. References

We consider that control parameters are correct if the criteria in the table (2.1) are respected:

Requirements	Criterion	Level
Obtain a behavioral attitude	Speed	Tr < 5%
	Overshoot	< 5%
	Constraint	$x_r = y_r = z_r = 1$ and $\varphi_d = 0$

Table 2-1.Criteria of specifications

These criteria are chosen based on experiments, since if we obtain values far from the suggested ones we will have a vibrating system or not well controlled (too slow), mostly in altitude where it can influence guiding the quad copter due to the strongly coupled dynamic of the quad copter.

The choice of values is based on knowing the critical point of the process. Experimentally, we loop the process with a simple proportional regulator, where we raise the gain until the system oscillates in a permanent way; we are then at the limit of stability. The first suggested values give a short rise time but with an overshoot.

2.5. Quad copter attitude and position control:

The objective of this part is to modify the open loop model so to control the attitude (ϕ, θ, Ψ) and the position z. A linear control is used, the PID controller.

2.5.1. Linearization:

In order to apply previously mentioned control, linearization around a functioning point is necessary.

We take this vector as a functioning point:

$$[\phi \quad \theta \quad \Psi \quad \dot{\phi} \quad \dot{\theta} \quad \dot{\Psi} \quad x \quad y \quad z \quad \dot{x} \quad \dot{y} \quad \dot{z}] =$$
$$[0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0] \tag{2.10}$$

The linear model is described by the following set of equations:

$$\begin{cases} \ddot{\phi} = \frac{l}{I_X} U_2 \\ \ddot{\theta} = \frac{l}{I_Y} U_3 \\ \ddot{\Psi} = \frac{l}{I_Z} U_4 \\ \ddot{x} = u_x \frac{1}{\mathrm{m}} \mathrm{U}_1 \\ \ddot{y} = u_y \frac{1}{\mathrm{m}} \mathrm{U}_1 \\ \ddot{z} = -g + \frac{1}{\mathrm{m}} \mathrm{U}_1 \end{cases} \tag{2.11}$$

Applying LAPLACE transform, we get:

$$\begin{cases} \phi(s) = \frac{l}{I_X s^2} U_2 \\ \theta(s) = \frac{l}{I_Y s^2} U_3 \\ \Psi(s) = \frac{l}{I_Z s^2} U_4 \\ x(s) = u_x \frac{1}{\mathrm{m}s^2} \mathrm{U}_1 \\ y(s) = u_y \frac{1}{\mathrm{m}s^2} \mathrm{U}_1 \\ (z+g)(s) = \frac{1}{\mathrm{m}s^2} \mathrm{U}_1 \end{cases} \quad (2.12)$$

We notice that each transfer state function is a double integrator multiplied by a gain.

2.5.2. Controls:

We will start the control synthesis by a decoupled system as it is simpler, then apply the regulators found on a decoupled system.

The synthesis will be made taking into consideration system's dynamics and motors saturation effect, as it is an important practical factor.

For that we will control each command with its corresponding state. After that, we will adjust the control gains according to the complete nonlinear model described by:

$\Rightarrow$ For the angles:

$$\begin{cases} \ddot{\theta} = -\dot{\Psi}\dot{\phi}\left(\frac{\mathrm{I_Z}-\mathrm{I_x}}{\mathrm{I_y}}\right) - \frac{\mathrm{J_r}}{\mathrm{I_y}}\overline{\Omega_r}\dot{\phi} - \dot{\theta}^2\frac{K_{fy}}{\mathrm{I_y}} + \frac{l}{\mathrm{I_y}}u_3 \\ \ddot{\Psi} = -\dot{\Psi}\dot{\phi}\left(\frac{\mathrm{I_x}-\mathrm{I_y}}{\mathrm{I_z}}\right) - \frac{K_{fz}}{I_z}\dot{\Psi}^2 + \frac{l}{\mathrm{I_z}}u_4 \\ \ddot{\phi} = -\dot{\Psi}\dot{\phi}(\frac{\mathrm{I_Z}-\mathrm{I_Y}}{\mathrm{I_x}}) - \overline{\Omega_r}\frac{\mathrm{J_r}}{\mathrm{I_x}}\dot{\theta} - \frac{K_{fx}}{I_x}\dot{\phi}^2 + \frac{l}{\mathrm{I_x}}u_2 \end{cases} \quad (2.13)$$

$\Rightarrow$ For the z:

We obtain it from equation, the representation of the control u_1 in equation 2.2 and the linear transfer function in equation (2.5):

$$G_T(s) = \frac{\sum \omega_m}{V_s} = \sum\left(\frac{\frac{1}{k_e}}{1+s\tau_m+s^2\tau_m\tau_e}\right) \quad (2.14)$$

$$u_1 = b(\omega_1+\omega_2+\omega_3+\omega_4) = b\sum\omega_m \quad (2.15)$$

$$(z+g)(s) = \frac{1}{ms^2} U_1 \quad (2.16)$$

We get:

$$(z+g)(s) = \frac{1}{ms^2} \sum \omega_m \quad (2.17)$$

Where $\boldsymbol{\omega_m}$ is a single motor output and $\sum \boldsymbol{\omega_m}$ represents the real output of the four motors which is the output of Brushless motors in our Simulink model.

We see that controlling motors' velocity, we will be having u_1 that permits to control the z, it is sufficient to divide by the mass and multiply –b (axis configuration where the z is oriented downward) the motors velocity state feedback.

$\Rightarrow$ For x, y, θ_d, ϕ_d we used the non holonomic effect (see Appendix I)

2.5.3. Control laws synthesis

We suggest in this section, a control strategy was developed [1] and is based on two loops (an internal loop and an external one), the internal loop contains:

- Roll control (ϕ)
- Pitch control (θ)
- Lace control (Ψ)
- Altitude control (z)

The external loop includes two controls of position x and y. It generates the desired roll and pitch movements ϕ_d, θ_d passing by the correction bloc.

We will use PD regulators or their gains so to determine the six controls: Roll, Pitch, Yaw, X, Y, Z.

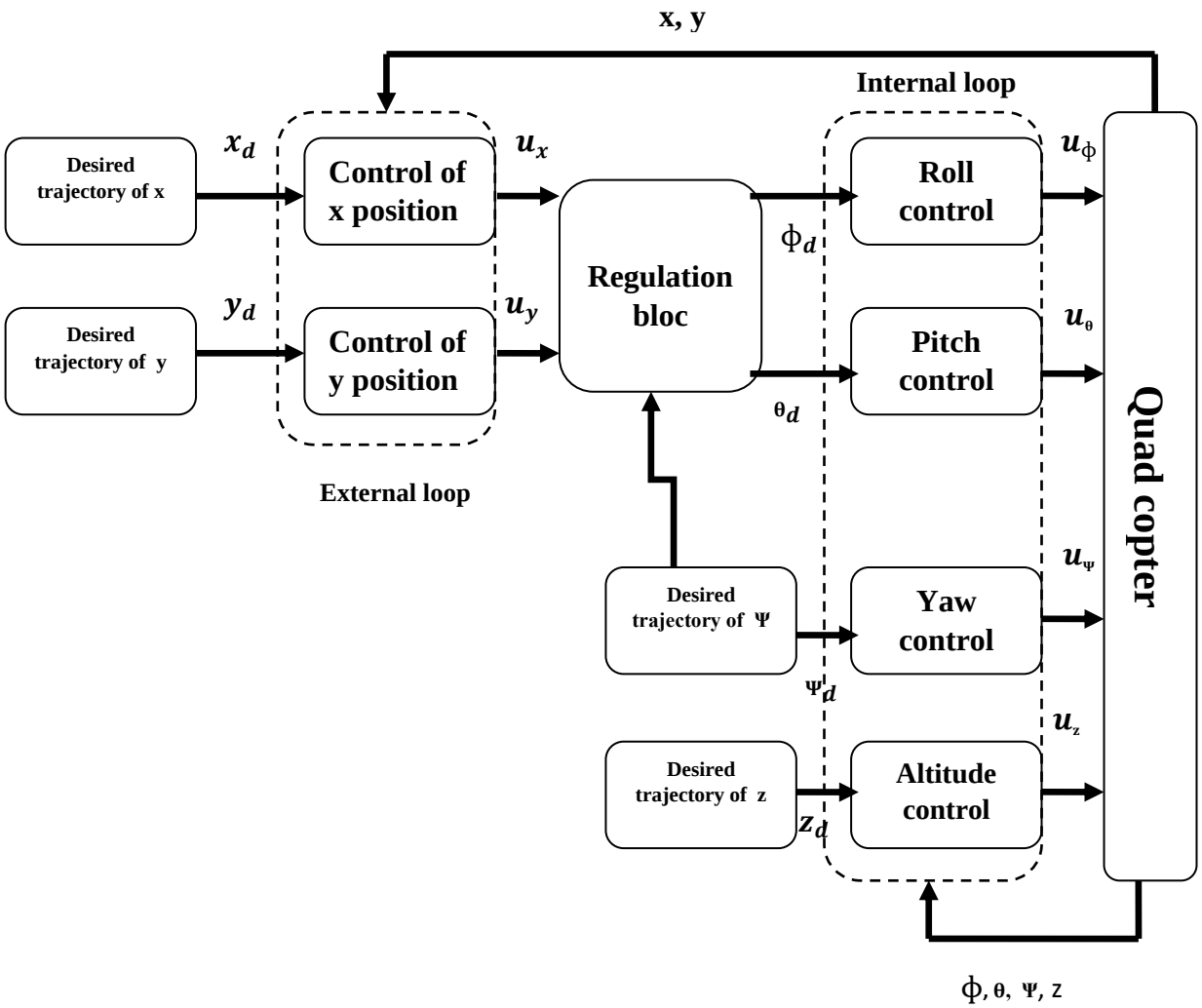


Figure 2.13. Structure of quad copter control [1]

2.6. PID controller:

PID controller is used in control so to stabilize mechanical systems. In order to apply a PID control, we observe the difference between the desired value around which one would stabilize its system, the constraint and the real observed value on the system; this difference is called the error.

- Proportional action: apply a K_p gain on the error.
- Integral action: we integrate the error, and multiply the result of integration byK_i.

- Derivative action: we derive the error then multiply the result by K_d.

In our case we applied a PID control on pitch, roll and lace angles. Figure 2.14 represents SIMULINK model and the entire PID control system.

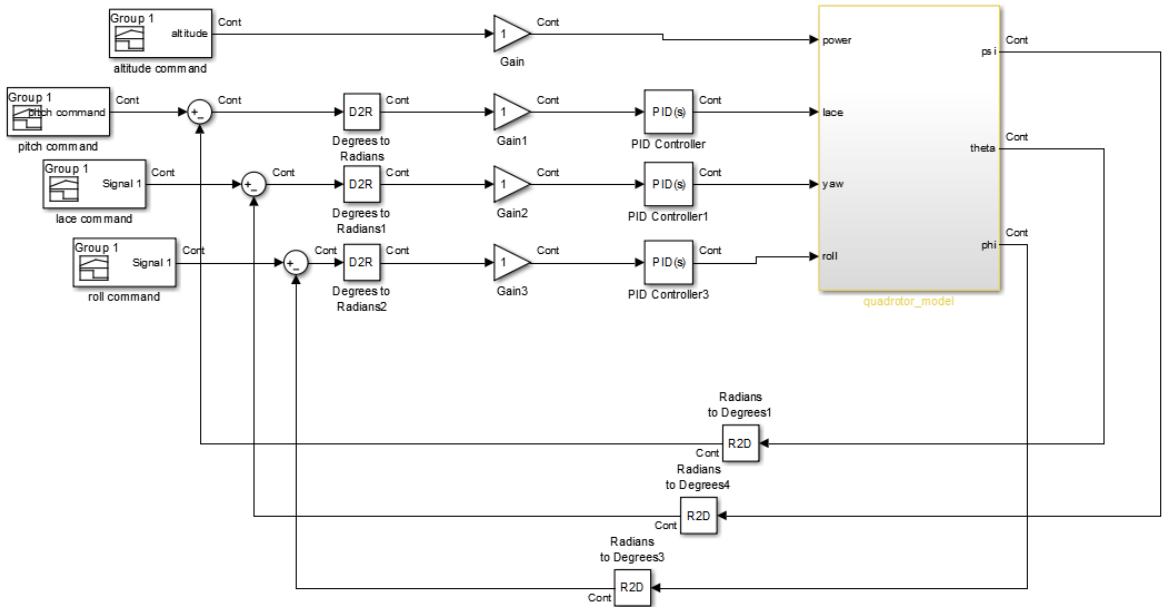


Figure2.14. Scheme of Simulink model with PID control

The four inputs are recombined so to control the motors. The results are interesting only near the equilibrium position (roll, pitch and lace near zero). But in the case of an initial orientation far from equilibrium state, PID regulator cannot stabilize a nonlinear system. (See more details about PID controller parameters effect in APPENDIX A).

Once we finished modeling, we could then test our controller to determine PID coefficients (PID tuning) by MATLAB-Simulink **(APPENDIX H)**, in the next section we will see and discuss the obtained results.

2.6.1. Simulation results:

Initial values of roll and pitch angles were taken from non holonomic bonds (see Appendix I), whereas yaw initial angle has been taken at 45° and X=Y=Z=1

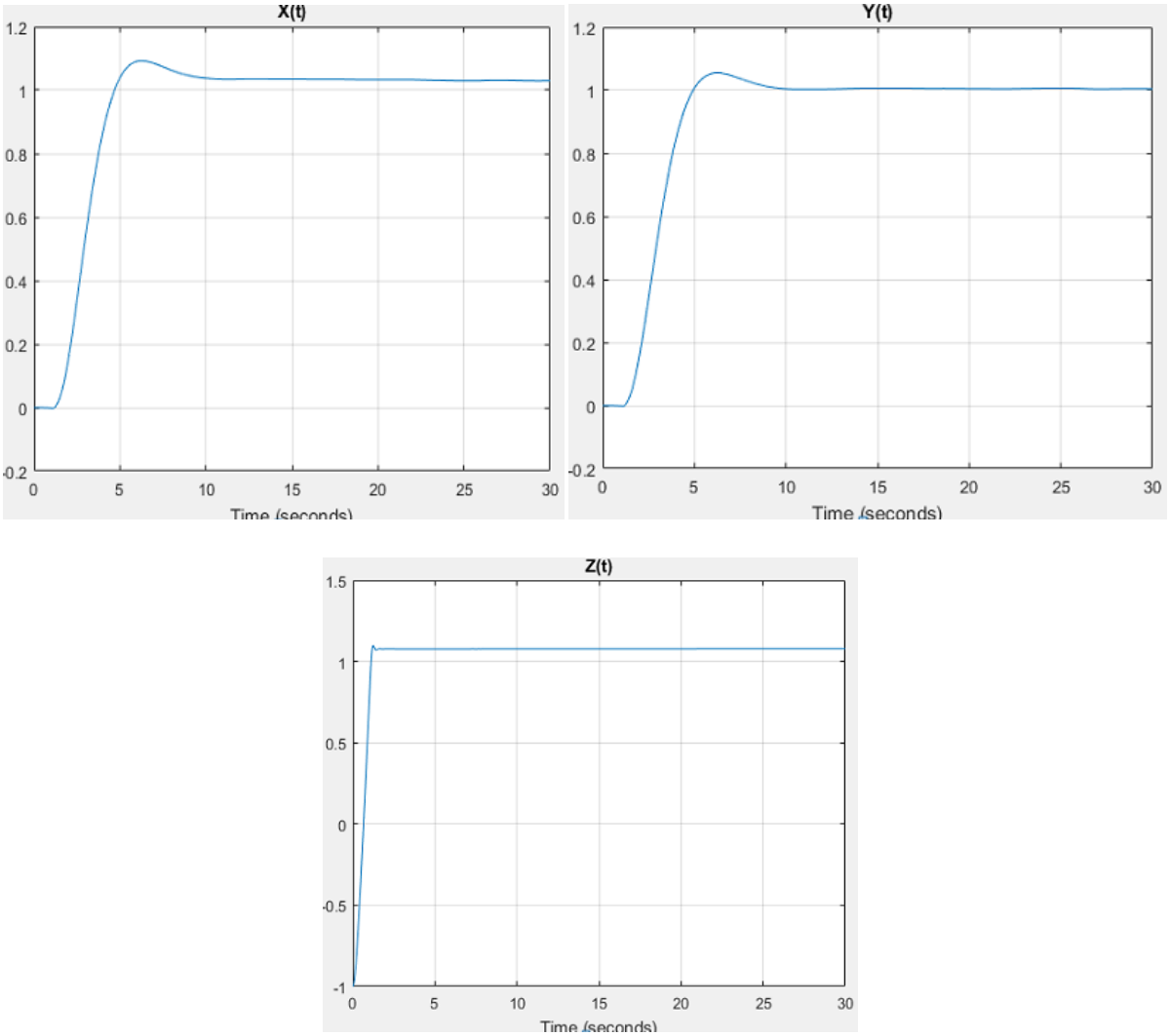
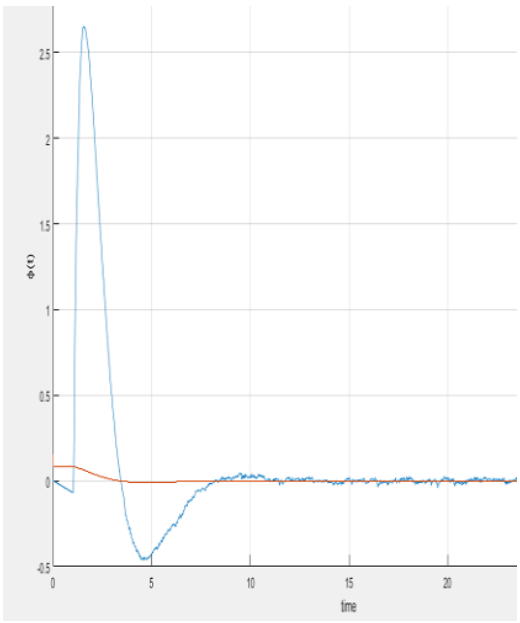


Figure 2-15 PID Controller position response



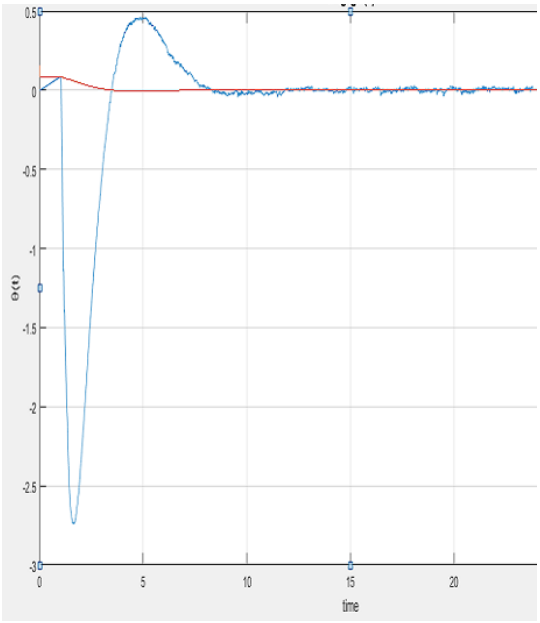


Figure 2-16-a PID Controller response on ϕ(t) **Figure 2-26-a PID Controller response onθ(t)**

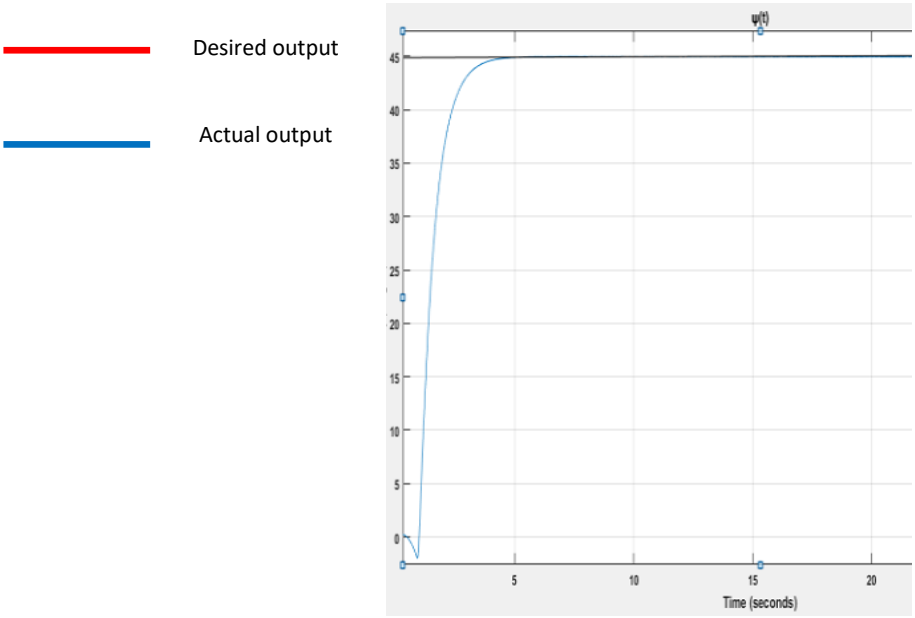


Figure 2-36 PID Controller behavioral response on ψ(t)

2.6.2. Discussion:

We notice that position responses are better of the open loop ones (more stable response), also we see that Z response is faster than the X and Y ones due to state equations relating positions.

Based on these results, even if the PID controller does not take into consideration torques between translation dynamics and rotational dynamics but results present a remarkable improvement in response where T_r and e_{ss} are reduced.

The purpose for which a PID controller has been implemented was reached, criteria of specifications met, we can slightly vary the PID parameters with the existing trade off precision, rapidity and stability to correct the difference between the theoretical application and the real implementation

2.7. Linear Quadratic control application:

Optimal control theory acts on dynamic system with minimal cost. In case the differential equations system is a linear system or a linearized system around an equilibrium point, we can represent it as follows:

$$\begin{cases} \dot{X} = AX + BU \\ Y = CX + DU \end{cases} \quad (2.18)$$

A linear quadratic regulator consists of looking for a matrix K_c as a state feedback control as:

$$u(t) = -K_c x(t) \quad (2.19)$$

It makes the system stable and minimizes the quadratic criterion:

$$J = \int_0^{\infty} (x^T Q x + u^T R u)\, dt \quad (2.20)$$

Where the weighting matrices satisfy the conditions:

$$R = R^T \geq 0 \; and Q = Q^T \geq 0 \quad (2.21)$$

In this case the solution is:

$$K_c = R^{-1} B^T P \quad (2.22)$$

Where P obeys to RICATTI equation:

$$-PA - A^T P + PBR^{-1}B^T - Q = \dot{P} \quad (2.23)$$

Under steady state the solution P is:

$$PA + A^T P - PBR^{-1}B^T + Q = 0 \quad (2.24)$$

2.7.1. Riccati equation solving:

From the general Riccati equation (whereA_r, B_r, C_r and D_r of dimensions respectively n*n, n*m and m*m):

$$XB_rX + XA_r - D_rX - C_r = 0 \tag{2.25}$$

We construct first a Hamiltonian:

$$H = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \tag{2.26}$$

Applying equation (3.16) to (3.13) yields to:

$$H = \begin{bmatrix} A & -BR^{-1}B^T \\ -Q & -A \end{bmatrix} \tag{2.27}$$

We can show that the 2.n Eigen values corresponding to the Hamiltonian matrix are constituted from the n Eigen values of the closed loop matrix $A - BK_C$ and their opposite. Then, if λ is an Eigen value of H, $-\lambda$ is also an Eigen value.

There exists n Eigen value whose real part is negative. Let $\boldsymbol{\Lambda} = diag(\boldsymbol{\lambda_1}, \dots, \boldsymbol{\lambda_n})$the associated matrix to the Eigen values. If a matrix T of dimension 2n*n is constructed from Eigen vectors corresponding to H Eigen vectors then we get:

$$HT = T\boldsymbol{\Lambda} \tag{2.28}$$

We can then have P that verifies equation (2.13).

The LQ optimal control method obliges us to linearize the system under the form:

$$\dot{X} = AX + BU \tag{2.29}$$

For our system, linearization only around the equilibrium point will lead us to a model far from the real one system. [24]

2.7.2. Linearization around an equilibrium point:

$$\frac{d}{dt}\delta x = J_z(x_0, u_0)\delta x + J_u(x_0, u_0)\delta u \tag{2.30}$$

$$J_x(x,u) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \cdot & \cdots & \cdot \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.31}$$

$$J_x(x,u) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \cdot & \cdots & \cdot \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{I_x} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{I_y} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} & 0 & 0 \end{pmatrix} \tag{2.32}$$

Jacobians do not depend on state variables nor on input variables. They are constant which permits us to write:

$$x = \begin{pmatrix} \dot{\Phi} \\ \ddot{\Phi} \\ \dot{\theta} \\ \ddot{\theta} \\ \dot{\Psi} \\ \ddot{\Psi} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \Phi \\ \dot{\Phi} \\ \theta \\ \dot{\theta} \\ \Psi \\ \dot{\Psi} \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{I_z} \end{pmatrix} \begin{pmatrix} U_2 \\ U_3 \\ U_4 \end{pmatrix} \tag{2.33}$$

With:

$$\begin{pmatrix} U_2 \\ U_3 \\ U_4 \end{pmatrix} = \begin{pmatrix} b(-\Omega_2^2 + \Omega_4^2) \\ b(\Omega_1^2 - \Omega_3^2) \\ \mathrm{d}(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \end{pmatrix} \tag{2.34}$$

This means that we only considered drag and pitch motors effects and neglected gyroscopic effect.

2.7.3 Simulation results:

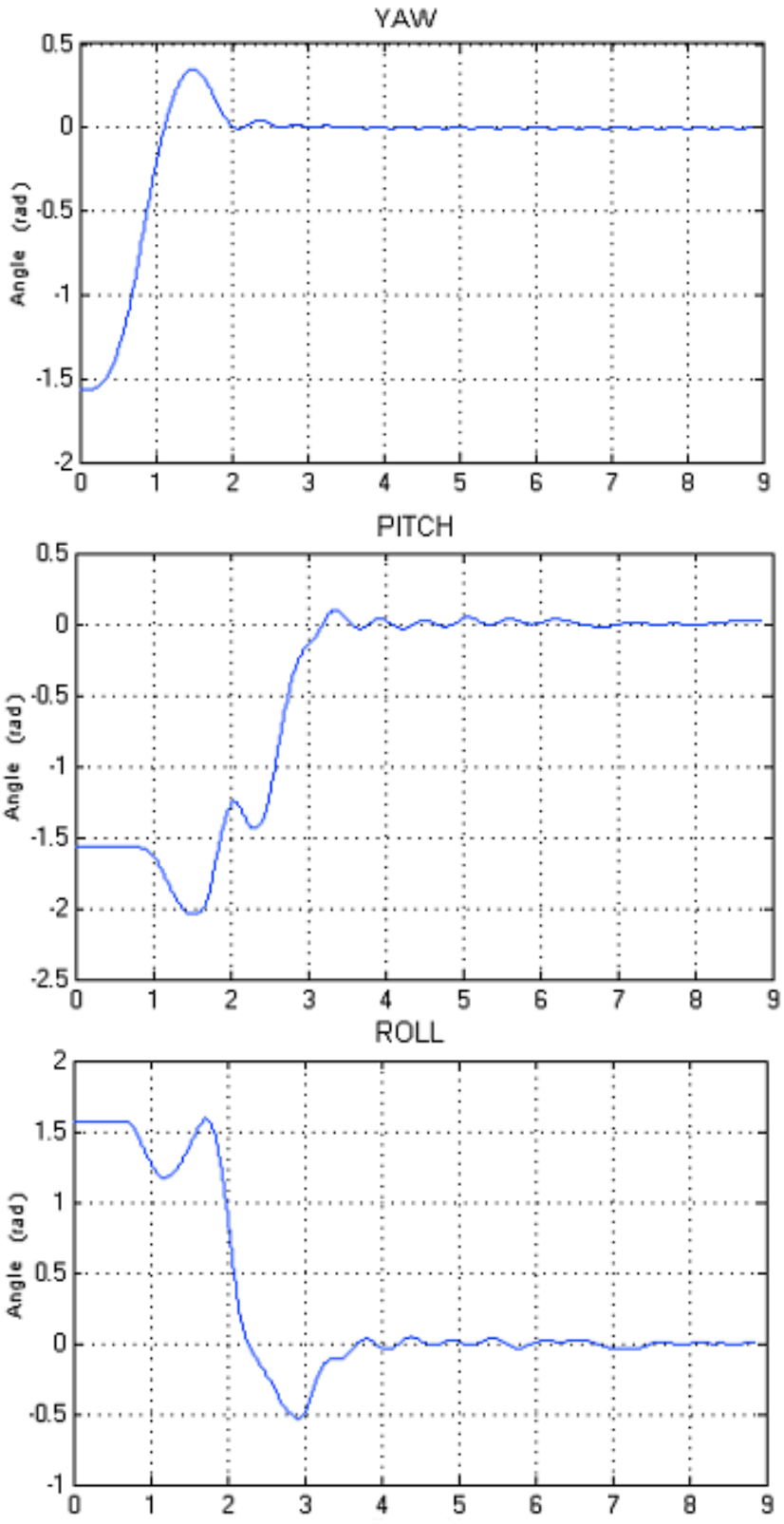


Figure 2-17 Simulation: The system has to stabilize the orientation angles starting from π/2 with an LQ controller

2.7.4. Discussion:

Results are satisfactory, even if we start with a critical initial angles orientation of $\frac{\pi}{2}$, all angles go back to 0°

Simulation gives good results around the equilibrium. But the system is linearized around the equilibrium point, so the regulator can compensate only small angles.

2.7.5. LQR Second approach:

Optimization of our system to a large scale of flight, needs linearization around each state, each coupled term is represented twice, fixing and varying a state at each instant.

$$\dot{X}^T = [\dot{\phi} \quad \ddot{\phi} \quad \dot{\theta} \quad \ddot{\theta} \quad \dot{\psi} \quad \ddot{\psi}]^T \tag{2.35}$$

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{I_{yy}-I_{zz}}{2I_{xx}}\dot{\psi} & 0 & \frac{I_{yy}-I_{zz}}{2I_{xx}}\dot{\theta} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{I_{zz}-I_{xx}}{2I_{yy}}\dot{\psi} & 0 & 0 & 0 & \frac{I_{yz}-I_{xx}}{2I_{yy}}\dot{\phi} \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \frac{I_{xx}-I_{yy}}{2I_{zz}}\dot{\theta} & 0 & \frac{I_{xx}-I_{yy}}{2I_{zz}}\dot{\phi} & 0 & 0 \end{pmatrix} \tag{2.36}$$

$$B = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{l}{I_{xx}} & 0 & 0 & \frac{J_r}{I_{xx}}\dot{\theta} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{l}{I_{yy}} & 0 & 0 & \frac{J_r}{I_{yy}}\dot{\theta} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{I_{zz}} & 0 \end{pmatrix} \tag{2.37}$$

The matrices A and B are now being adapted through the drone trajectory; the linearization is thus more faithful.

2.7.6. LQ controller synthesis and simulation

Q and R matrices choice is crucial and very sensitive but deterministic for the controlled system behavior.

Let K the optimal value that minimizes J and P. K_c can be calculated with LQR Matlab function. Results can be seen in the following figures:

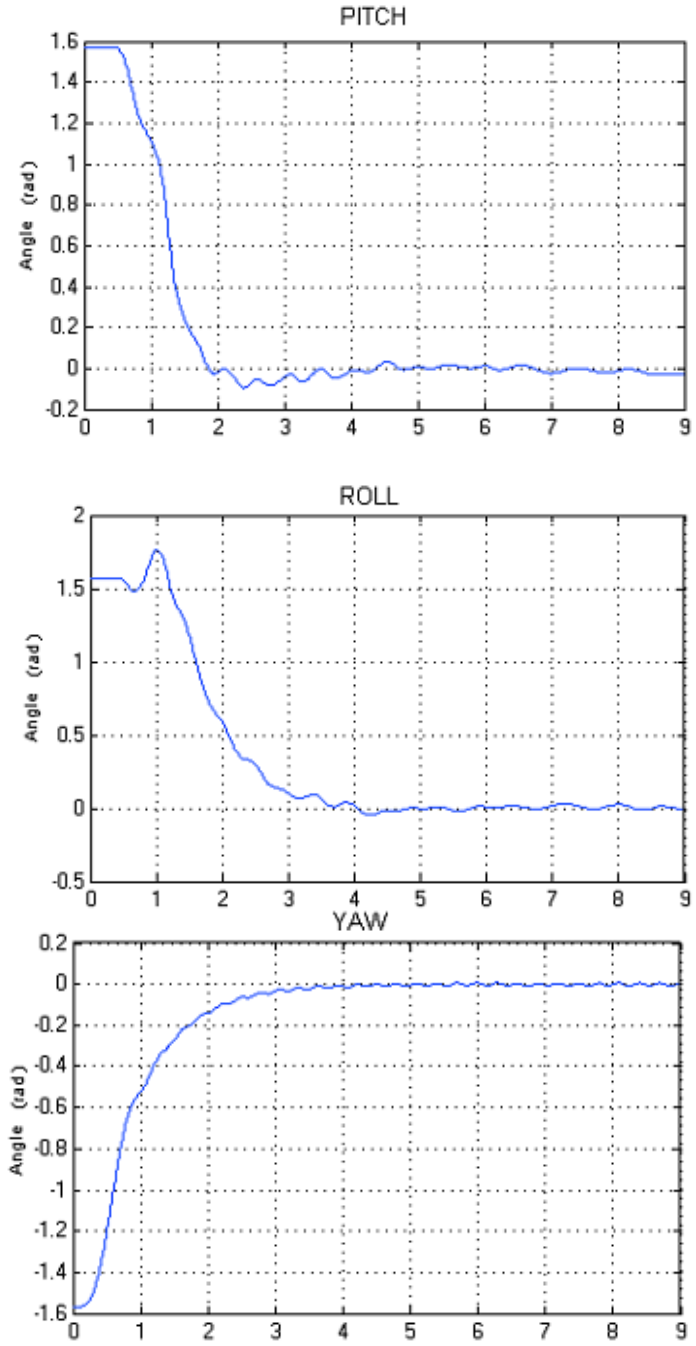


Figure 2-18 Simulation: The system has to stabilize the orientation angles starting from π/2 with an LQ controller

2.7.7. Discussion:

We notice that the stabilization is correct we reduced settling time and we have less overshoot in all motions, however a small steady state error remains because of unmodeled effects and the systematic slight deference in the propulsion groups.

2.8. Comparison between the two approaches:

Both control systems gave satisfactory simulation results. Near hover, they are fast and efficient; PID has the advantage of being easy to manipulate and tune even in real implementation on a microcontroller, it is more intuitive.

However, the Linear Quadratic stability approach minimizes commands, the less commands we have the less saturation problems we get, and it is well adapted to aerial vehicles. Its disadvantage is that it is less intuitive in manipulation than the PID.

2.9. Conclusion

The work presented in this chapter focuses on behavioral stability of our quad copter. We elaborated a control using two controllers, PID approach and LQ approach.

Simulation results on the quad copter model show that these control approaches gave acceptable results in terms of stability and performance, a comparison based on the obtained results was performed where the LQ approach gave better performances. However both controllers are limited due to model simplification and external disturbances, where the main advantage is their simplicity.

Chapter 3

Implementation and Experimental results

3.1 Introduction

So far, we have seen how to develop a mathematical model of the quad copter. In this part, we are going to list all the needed parts that are required to implement an operational quad copter implementing a suitable controller (see Appendix **G** for more details about Quad copter building steps).

Our drone is composed of four rotors, as its name indicates « quad copter ».

The mechanical structure is realized by a plastic frame. Control is made by an arduino card. The treatment of various sensors data is processed by a C program developed for ATMEGA processor integrated in ARDUINO.

A bloc of sensors with six degrees of freedom (DOF) is related to the ARDUINO. This bloc contains an accelerometer that measures acceleration on the three orthogonal axes and a gyroscope that measures rotations rate on the three axes too.

Our electronic configuration does not contain a Global Positioning System (GPS) thus the exact position of the quad copter cannot be determined, which affects position control and tracking.

3.2 Hardware design

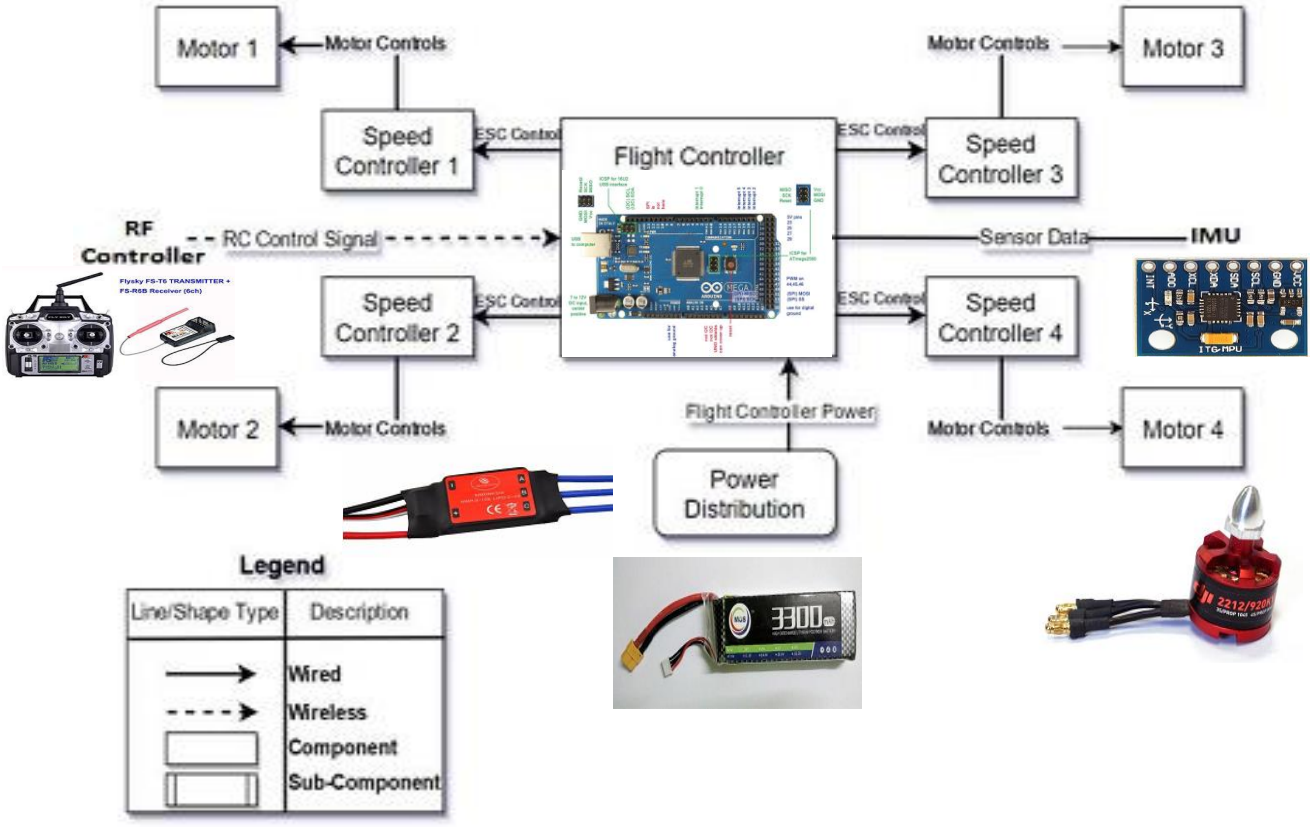


Figure 3-1 block diagram of the quad copter architecture

A block diagram of the quadcopter architecture is shown in Figure 3.1 above. Table 3.1 accompanies the block diagram and lists all of the communication interfaces between the various subsystems.

The flight controller receives:

- Control signal from the Ground Control Station wirelessly,
- Power through the power distribution board and interfaces with the flight controller sensors as well as the four Electronic Speed Controllers (ESCs).
- The four ESCs drive each of the four motors with a three phase power signal.

Signal	Description	Protocol
Flight Controller Power	The power distribution system on the quad copter provides a 5 VDC (volts direct current) power supply to the flight controller.	DC
ESC Control	The flight controller sends a pulse width modulated signal between one millisecond and two milliseconds to the ESC for each motor to control the movement of the quad copter	PWM
Motor Controls	The ESCs send a three-phase trapezoidal wave with varying frequencies to their respective motors to drive them	Three-Phase Power

Table 3-1 Communication interfaces between various subsystems

3.2.1 The battery

In this project, we used a MOS 3S LiPo battery 11.1v 3300 mAh 25C to power the drone (see figure 3.2). LiPo batteries are the standard power source for quad copters. The battery provides power for all on-board. The motors consume the biggest amount of power in the system.

Figure 3-2 LiPo battery

3.2.2. Process plant:

3.2.2.1 Brushless DC motors

Brushless DC motors (BLDC) are synchronous motors. They are lighter and provide more torque and have longer lifetime compared to brushed motors. In our project, we Used DJI 2212 920kv brushless motors (see figure 3.3). Controlling the motors is essential to balance and lift the quadcopter. The easiest way to program the motor controllers is to use ESC programming cards.

Figure 3-3 Brushless DC motor

3.2.2.2 Electronic Speed Controller (ESC)

Electronic Speed Controller is an electronic circuit that controls the speed and direction of a motor. We used 4pcs SimonK 30A brushless (see figure 3.4). It is characterized by handling the max current rating to the motors and providing them with the right voltage. ESCs need to be calibrated in order to know the limits of the transmitter and then control precisely the speed of the motors.

Figure 3-4 Electronic speed controller

3.2.3 Flight controller system hardware

In this section, we are going to introduce the different components used to build the flight controller.

3.2.3.1 Inertial Measurement Unit (IMU)

Inertial Measurement Unit (IMU) is an electronic analog device based on the combination of accelerometers and gyroscopes, and sometimes magnetometers and barometers. In our project, we used GY 521 MPU 6050 3-axes analog gyro-sensors accelerometer module (see figure 3.5).

-**The accelerometer:** measures linear acceleration in up to three axes (X, Y and Z). It plays a major role in allowing a remotely controlled drone to remain stable in the air. It should be mounted on the flight controller so that the linear axes line up with the main axes of the Quadcopter.

-**The gyroscope:** detects angular changes on up to three angular axes (psi, phi, and theta). It should be mounted so that its rotational axes line up with the axes of the Quadcopter.

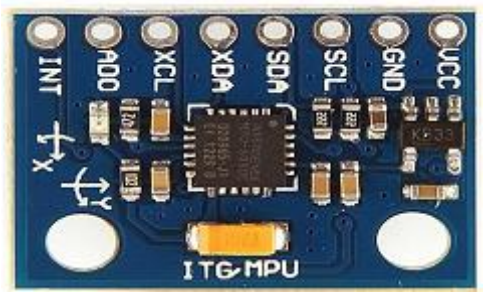


Figure 3-5 Inertial Measurement Unit

3.2.3.2 Flight Controller

It is a programmable microcontroller with specific sensors, microprocessors and output/input pins. It needs to take inputs from the user via RC (Radio receiver), from an accelerometer and a gyroscope. Its outputs are the four ESCs. We use it to stabilize and control the quadcopter. In our project, we used the arduino Mega 2560 as micro controller. The arduino mega 2560 is a microcontroller board based on the ATmaga2560. It contains everything needed to support the microcontroller. It has 54 digital input/output pins,

14 pins of them can be used as PWM outputs, 16 analog inputs, 4 hardware serial ports (UARTs), a power jack, a 16 MHz crystal oscillator, a USB connection, an ICSP header and a reset button (see figure 3.6). To power it, you can connect it to a battery or an AC-to-DC adapter or simply to a computer with a USB cable.

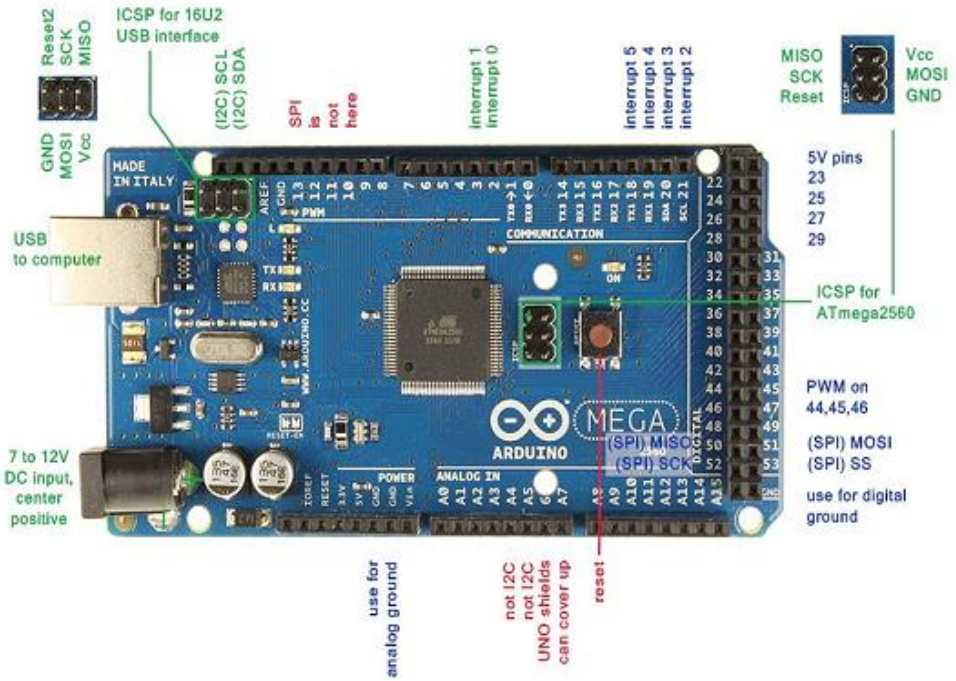


Figure 3-6 ARDUINO ATMEGA 2560

3.2.3.3 Ground control station (GCS)

Ground control station (GCS) is the area-based control center that provides an effective control for especially aircrafts and facilitates the human control of the quadcopter which can be used to control and monitor single or multiple UAV missions from take-off to landing at the same time (See figure 3.6). The remote controller used is the Sky Fly –T6, the arduino then reads the values and processes the data, maps them.

Figure 3-7 Fly Sky-T6 (Ground control station)

3.2.4 Communication and Protocol

A good communication between the FS-T6 and the Quadcopter is essential to have the Quadcopter working properly. All the sensors must report the status of the flying machine to the monitor within the FS-T6 and this latter should respond to those data and make clear report to the operator. The communication is via RF modules.

3.2.4.1 Radio Frequency (RF)

A radio frequency module (RF) is a small electronic device used to transmit or receive radio signals between 2 devices wirelessly. In our project, we used 2 nRF24L01+ module (see figure 3.8).

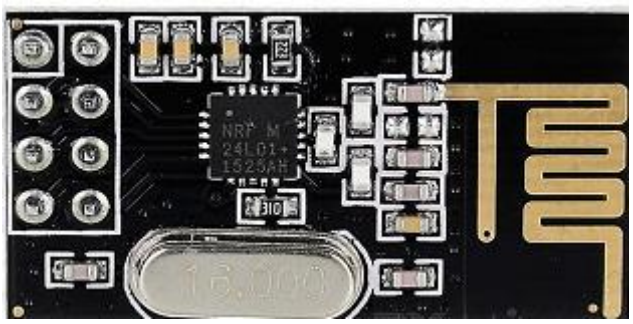


Figure 3-8 Radio Frequency module

3.2.4.2 Quad copter navigation from FS-t6

A continuous communication with the ground station is necessary for the quadcopter to localize itself in its outdoor environment. Arduino UNO reads analog signals which are digital values between 0 and 1023 converted via an ADC, then the nRF24 transmits data to the receiver and the receiver on the quadcopter receives the data sent, then, it moves through the different desired locations.

3.3 Software design

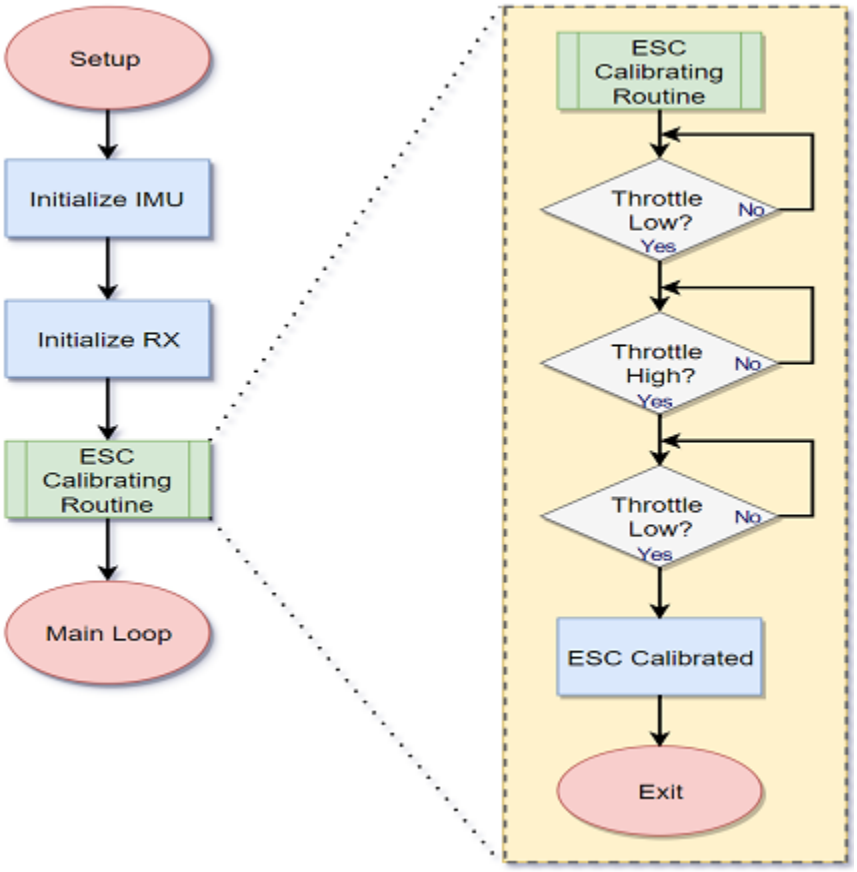


Figure 3 -9 Flow chart of software design from setup to main loop with ESC calibrating routine description

When the power is turned on, the drone enters the setup process which executes initialization and calibration routines for different modules, first we initialize the IMU by defining the I2C communication protocol, setting the correct interrupt pin and waiting for IMU to respond, after that we initialize the RF module by accessing the SPI port, defining the correct pipe address and start listening for incoming packets, this being done, we start the ESC calibration process which is essential to map the max& min signal range of the ESCs, we start by giving a the max throttle value which the ESCs responds to by three beeps after that we assert min value the ESCs make a single long beep indicating that calibration process is done.

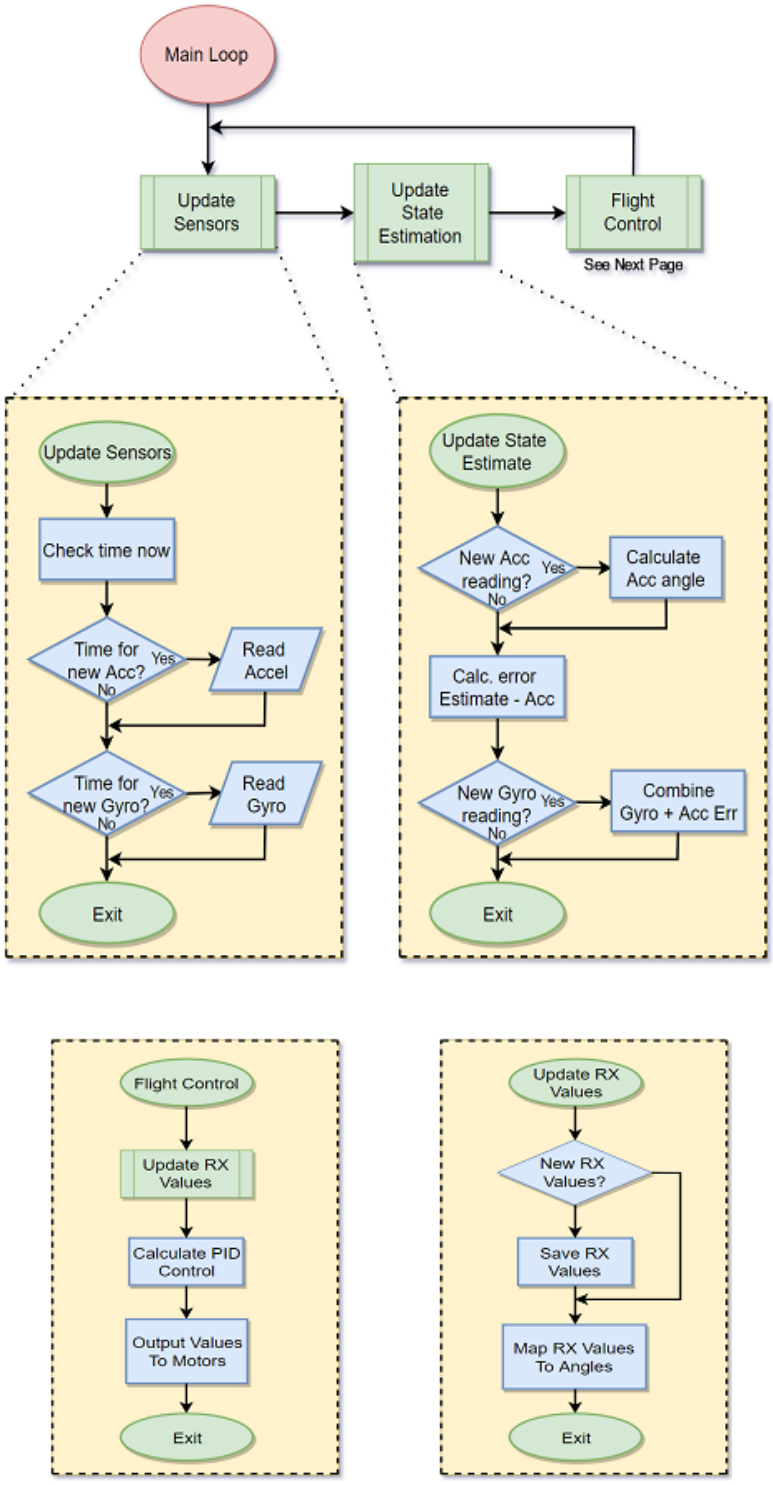


Figure 3-10 Flow chart of main loop program

In the main loop there are 4 functions running: update_sensors(), update_state(), update_RX(), flight_control().

• update_sensors(): this function is executed each 10ms to read FIFO register of IMU and update current drone orientation in space.

• update_state(): this function combine gyroscope reading with accelerometer reading to reduce the effects of vibration on our system, it uses a low pass filter (20 kHz) to remove fluctuation in IMU readings .

• update_RX(): this function is used to check if there is a new radio command sent and process the data.

• flight_control(): this is the core function that takes as an input RX commands and IMU reading processes them and outputs control signal to the motors.

Then, we go to the main loop where flight controller will read sensors and the state estimation data that are updated and map the RX values to angles then it will calculate the PID values and output them to the motors.

3.4 Programming and experimental results

After implementing the flight controller, we are going to study and discuss the results of the implementation by trying a flight test.

3.4.1 Components test

At the beginning, we tested the motors with the ESCs one by one using a computer power supply unit that delivers a maximum of 10 Amps on 12 Volts (see figure 3.10), which was enough to test 1 motor. We used the power supply unit because we encountered a problem with the battery.

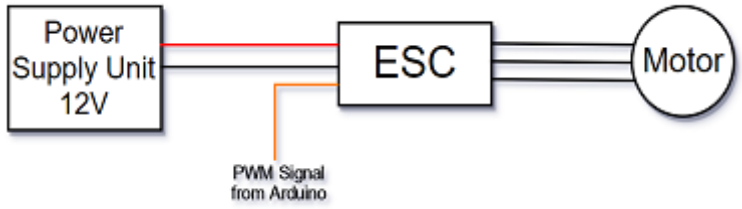


Figure 3-11 One DC motor test diagram

After that, we tested the gyroscope and the accelerometer of the MPU6050, we faced some problems with Wire library we were using, but we fixed them by using another library MPU 6050.

3.4.1.2 Radio Communication testing

Establishing a good communication link between the drone and remote controller is an essential parameter for implementing the flight controller, so we began by setting up the RF modules and preparing for the tests.

3.4.1.2.1. Basic data transmission & range test

We began by wiring the RF modules to the arduino boards according to SPI protocol (Serial Peripheral Interface), after which we set a common pseudo-pipeline address between the RF modules to hook them on the same channel, after that we began testing by sending a simple packet of data while changing the data transfer rate to get max range (radius), the results are in Table 3.2.

Data Rate	Range
2MBbps	$\cong 9m$
1MBps	$\cong 16m$
250KBps	$\cong 27m$

Table 3-1 data range transmission test

Note 1: During all experiment Power Amp was set to max.

Note 2: Those RF modules have their antenna integrated on PCB.

From Table 2, results are coherent with theory which says that we get max range with lowest possible data rate, in our case it's 250KBPS.

3.4.1.2.2 Control data transmission & safety measures

Control data being sent to drone consists of an array of 5 elements ordered in the Table 3.3.

Array[id]	Value
0	Throttle value
1	Pitch angle
2	Yaw angle
3	Roll angle
4	Arming/Disarming

Table 3-2transmitted control data

3.4.2 Flight tests

In order to test our flight controller and the algorithms (See codes in appendices D, E, F) we implemented, we did few tests on the quadcopter with and without propellers.

3.4.2.1 First test without propellers

We tested the motors without propellers and the result is that the motors are controlled correctly and also they respond to the IMU.

3.4.2.2Second test with propellers

In order to test the capability of the motors to lift the quadcopter and for security reasons, we have attached it to a table by threads.

We tested the motors with propellers and we remarked that the system was not stable due to PID coefficients which were not tuned.

3.4.3 PID tuning

After having the quadcopter semi flying, we went through tuning PID values. How to tune PID?

By changing each of the PID coefficients of ROLL then PITCH then YAW and see their effects on the quadcopter until the system stabilizes and thus we will get the quad copter stable at flight.(see Appendix A)

3.4.4 Third test after tuning PID coefficients

We performed our test after trying to tune PID. We first power up the quadcopter, and then increase the thrust gradually using the throttle FS-T6 until the quadcopter lifts for a small altitude and then hold it for a while to see the stability of the system. The quad copter flies with vibrations and it is not stable. We keep on trying other values for PID, we notice some improvement in the stability of the quadcopter.

3.5 Difficulties and problems

At the beginning, we had a problem with the RF module, the commands are sent but the RF receiver does not receive them. The problem was in the pins and also with the program. After working on it, it works well. We had a problem with the IMU, it did not work correctly. The problem was in the IMU, the arduino reads values from FIFO Buffer very fast. Fortunately, we found a solution which is making a delay in the function of the IMU which is Update_Sensors.

3.6 Conclusion

In this chapter, we covered the hardware and software implementation of the quadcopter. We have realized a controller to stabilize the implemented quad copter, even

though all faced difficulties but we could accomplish an acceptable stable flight using the PID controller.

General Conclusion

General Conclusion:

Results:

First we developed a dynamic model of a quad copter based on forces and moments laws acting on the drone which was considered as a rigid system with six degrees of freedom.

After modeling, we treated the most important part which is control laws synthesis that ensures the quad copter stability. Control has been elaborated using two linear control methods, the first by a PID controller and the second by an LQ controller.

Modeling and control algorithms were grouped in the same model, that is divided into two sub systems: a dynamic model in a simulation environment and an embedded subsystem in an electronic card.

Finally, we obtained a stable hovering flight implementing the simple PID controller, unfortunately LQ controller implementation failed to accomplish a stable flight due to complexity of choosing Q and R matrices.

Faced Difficulties:

At first, given the nature of the project, it was difficult for us determine the extent of the task. The biggest difficulty encountered during this project was the stabilization of the drone.

Some of the sensors required the use of communication buses of the type SPI and I^2C that we did not know and that were difficult to implement.

Finally, in general terms, we have poorly estimated the duration to be attributed to some tasks, which ultimately did not allow us to finish all we wanted undertake.

Perspectives:

We can say that the predefined objectives of this project were met, but one can suggest more ideas to develop:

1-Improve the dynamic model of the quad copter where real robot parameters can be identified in a more precise way, to reduce uncertainties related to linearization and permit a model closer to the real one .

2- Add a navigation algorithm; this will permit a better estimation of flight parameters.

3- Using nonlinear control to enhance the response and ensure stability and robustness.

4- We can integrate guiding algorithms using GPS and different sensors (ultrasound, barometers…), an embedded system for image processing that can offer a new panel of activities (objects detection, tracking...)

5- Addition of decision algorithm related to artificial intelligence will make the drone more autonomous.

BIBLIOGRAPHY:

[1] KHEBBACHE HICHAM, «TOLERANCE AUX DEFAUTS VIA LA METHODE BACKSTEPPING DESSYSTEMES NON LINEAIRES», MSC, UNIVERSITY FERHAT ABBAS DE SETIF,2012.

[2] SAM ZINE LAABIDINE ET BOUKKEBEL ABED, « COMMANDE D'UN QUADROTOR PAR RESEAUX DE NEURONES», FINAL YEAR PROJECT, POLYTEVHNIQUE MILITARY SCHOOL, 2014.

[3] S.BOUABDALLAH, « DESIGN AND CONTROL OF QUADROTOR WITH APPLICATION TO AUTONOMOUSFLIGHT », PHD THESIS, EPFL, 2007.

[4] TSOURDOS A. AND WHITE B. A., "LATERAL ACCELERATION CONTROL DESIGN FOR AN LPV MISSILEMODEL", PROCEEDINGS OF THE EUROPEAN CONTROL CONFERENCE ECC'99, KARLSRUHE, GERMANY.SEPTEMBER 1999.

[5] ØYVIND MAGNUSSEN ET KJELL EIVIND SKJØNHAUG «MODELING, DESIGN AND EXPERIMENTALSTUDY FOR A QUADCOPTER SYSTEM CONSTRUCTION», MASTER THESIS, UNIVERSITY OF AGDER, 2011.

[6] TOMMASO BRESCIANI« MODELLING, IDENTIFICATION AND CONTROL OF A QUADROTOR HELICOPTER » DEPARTMENT OF AUTOMATIC CONTROL, LUND UNIVERSITY, OCTOBER 2008.

[7] OLUDAYO JOHN OGUNTOYINBO «PID CONTROL OF BRUSHLESS DC MOTOR AND TOBOT TRAJECTORYPLANNING AND SIMULATION WITH MATLAB/SIMULINK», HONOR THESIS, VASA UNIVERSITY, 2009.

[8] ADAM POLAK « APM MULTICOPTER DEVELOPMENT KIT FOR SIMULINK », PUBLICATION,ARIZONA STATE UNIVERSITY, 2013

[9] TOMMASO BRESCIANI « MODELLING, IDENTIFICATION AND CONTROL OF A QUADROTOR HELICOPTER », MASTER THESIS, DEPARTMENT OF AUTOMATIC CONTROL LUND UNIVERSITY, 2008.

[10] JOHAN FOGELBERG « NAVIGATION AND AUTONOMOUS CONTROL OF A HEXACOPTER IN INDOOR ENVIRONMENTS », MSC THESIS , DEPARTMENT OF AUTOMATIC CONTROL LUND UNIVERSITY, 2013.

[11] EDOUARD LAROCHE. « IDENTIFICATION ET COMMANDE ROBUSTE DE SYSTEMES ELECTROMECANIQUES ». ROBOTICS. UNIVERSITI LOUIS PASTEUR , STRASBOURG I, 2007.

[12] HTTPS://WWW.MATHWORKS.COM/HELP/ROBUST/REF/NCFSYN.HTML

[13] ALBERTO BEMPORAD «AUTOMATIC CONTROL 2 LOOP SHAPING »UNIVERSITY OF TRENTO ACADEMIC YEAR 2010-2011

[14] LAIB KHALED, MAAMRIA DJAMALEDDINE,

[15] NIZAR ZEIN EDDINE «MODÉLISATION ET CONTRÔLE D'UN QUADRIROTOR "BIRITOS"» FINAL YEAR PROJECT, LEBANESE UNIVERISTY, ENGINEERING FACULTY MECHANICAL DEPT, 2011-2012

[16] M. ORSAG AND S. BOGDAN. « HYBRID CONTROL OF QUADROTOR ». PROCEEDINGS OF IEEE, 17TH MEDITERRANEAN CONFERENCE ON CONTROL AND AUTOMATION, PP. 1239-1244, MAKEDONIA PALACE, THESSALONIKI, GREECE, 2009.

[17] SOCIETE DMS - DEPARTEMENT STI « DOSSIER TECHNIQUE, AR.DRONE DE PARROT » SITE : WWW.DMSEDUCATION.COM EMAIL:INFO@DMSEDUCATION.COM

[18] HTTP://WWW.MOTEURINDUSTRIE.COM/BRUSHLESS/TECHNIQUE.HTML

[19] GILLES BROCARD « MOTEUR BRUSHLESS À ROTOR EXTERNE, (BRUSHLESS OUTRUNNER) » SITE WEB HTTP://CLAUDE.GUENIFFEY.FREE.FR

[20] HTTP://BLOG.PATRICKMODELISME.COM/POST/QU-EST-CE-QU-UNE-BATTERIE-LIPO

[21] HTTPS://WWW.ABSOLU-MODELISME.COM/VARIATEURS-DE-VITESSE-ET-CONTROLEURS-BRUSHLESS

[22] HTTPS://FR.WIKIPEDIA.ORG

[23] R. LOZANO, P. CASTILLO, S. SALAZAR, D.LARA « STABILISATION DE VEHICULES AERIENS A DECOLLAGE VERTICAL : THEORIE ET APPLICATION »

[24] CHERIET REDA, «COMMANDE ET STABILISATION D'ATTITUDE D'UN DRONE DE TYPEQUADRIROTOR », FINAL YEAR PROJECT, IAB, 2013.

[25] ADAM UFFORD « DEVELOPMENT AND IMPLEMENTATION OF GUIDANCE, NAVIGATION AND CONTROL SYSTEMS FOR AN AUTONOMOUS AIR VEHICLE » MASTER OF SCIENCE INMECHANICAL ENGINEERING, TEXAS TECH UNIVERSITY, 2009.

[26] ALAIN VUILLE « ETUDE ET EXPERIMENTATION DE PETITS QUADRICOPTERES DESTINES A DESMISSIONS D'OBSERVATION », BACHELOR , HIGHER SCHOOL OF ENGINEERING DUCANTON DE VAUD , 2014.

[27].Y. BRIERE "STABILISATION D'UN QUADRIROTOR". ENSICA 2006.

[28] ALVIN NG « DESIGN AND BUILD OF SWARM QUADROTOR UAVS AT UGS », IMAV2014,UNIVERSITY OF GLASGOW SINGAPORE, 2014.

[29] ROBERT HARTLEY, « APM2 SIMULINK BLOCKSET » MATLAB CENTRAL, 13 NOVEMBER, 2012.

[30] HTTPS://WWW.MATHWORKS.COM/MATLABCENTRAL/FILEEXCHANGE/39037-APM2-SIMULINK-BLOCKSET

VIDÉOS:

1) QUADCOPTER SIMULATION AND CONTROL MADE EASY - MATLAB AND SIMULINK HTTPS://WWW.YOUTUBE.COM/WATCH?V=FPJZSQMQDVK

2) QUADCOPTER/DRONE/UAV - DYNAMIC MODELLING AND TRAJECTORY CONTROL IN MATLAB/SIMULINK

HTTPS://WWW.YOUTUBE.COM/WATCH?V=72PF4XIRKEK

LOGICIELS:

1) MATLAB 2015B.

2) MATLAB 2014B.

Appendix A: PID Approach

A-1 Introduction:

PID stands for Proportional, Integral, Derivative, it is part of flight controller software that reads the data from sensors and calculates how fast the motors should spin in order to retain the desired rotation speed of the aircraft.

The goal of the PID controller is to correct the "error", the difference between a measured value (gyro sensor measurement), and a desired set-point (the desired rotation speed). The "error" can be minimized by adjusting the control inputs in every loop, which is the speed of the motors.

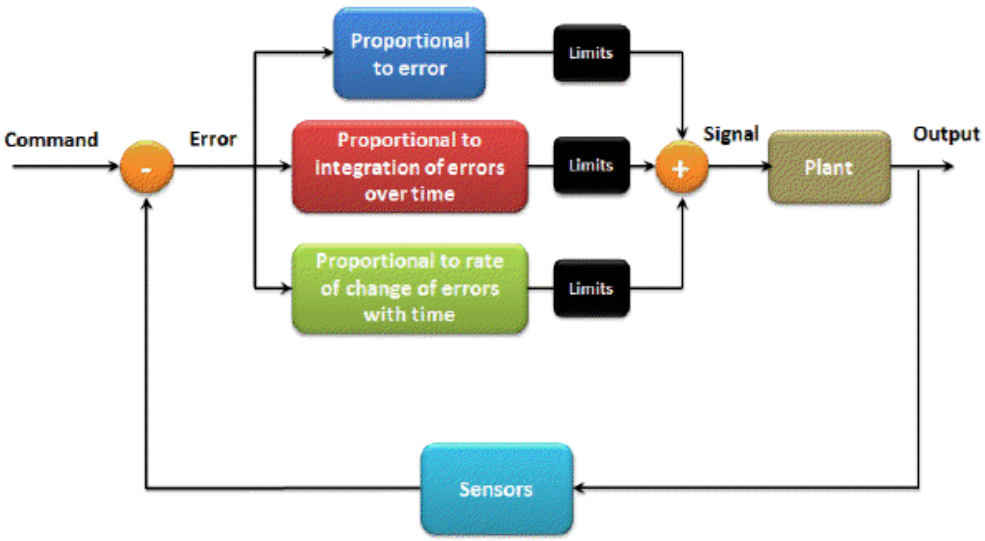


Figure 2- Quad copter PID diagram

There are 3 values in a PID controller; they are the P term, I term, and D term:

- "P" looks at present error – the further it is from the set-point, the harder it pushes.
- "D" is a prediction of future errors – it looks at how fast you are approaching a set-point and counteracts P when it is getting close to minimize overshoot.
- "I" is the accumulation of past errors, it looks at forces that happen over time; for example if a quad constantly drifts away from a set-point due to wind, it will spool up motors to counteract it.

A-2 Effect of each PID parameter:

Altering PID values affects a quad copter's behavior in different ways.

P Gain

P gain determines how hard the flight controller works to correct error and achieve the desired flight path (i.e. where the pilot wants the quad to go by moving the transmitter sticks).

Think of it as sensitivity and responsiveness setting. The sharp response provided with a high P gain can even make it feel like you have increased your rates. Generally speaking, higher P gain means sharper control while low P gain means softer control. If P is too high, the quad copter becomes too sensitive and tends to over-correct, eventually it will cause overshoots, and you will have high frequency oscillations. You can lower P to reduce the oscillations, but reduce it too much and your quad copter will start to feel loosely fitting.

I Gain

I term determines how hard the FC works to hold the drone's attitude against external forces, such as wind.

Think of it as the stiffness setting in the stall motion of your quad copter, and how well it holds its attitude. When I is too low you might find yourself having to correct the quad's flying path a lot more with your sticks, especially when you are active with the throttle. When I gain gets too high, your quad copter will be overly constrained by this, and start to feel stiff and unresponsive. It's similar to having a slower reaction and a decreased P gain. Excessive I gain in extreme cases can create a low frequency oscillation.

Another issue that I gain can address or improve is "throttle dips". In the real world, no two ESC's, motors or propellers are identical, thus they will provide different levels of thrust even when spinning in the same environment. When you do a punch out and immediately lower your throttle, one motor might increase and decrease RPM faster than the others; this will cause an unwanted dip movement. You can increase I gain to "fix" these tiny details in the flight performance.

D Gain

D gain works as a damper and reduces the over-correcting and overshoots caused by P term. Adding D gain can “soften” and counteract the oscillations caused by excessive P gain, as well as minimizing prop wash oscillations. When D is too low, your quad will have bad bounce-backs at the end of a flip or roll, and you will also experience the worst prop wash oscillations in vertical descents. Increasing D gain can improve these problems; however, an excessive D value can introduce vibration in your quad copter because it amplifies the noise in the system. Eventually this will lead to motor overheat and quad oscillation. Another side effect of excessive D term is the decrease in the quad’s response.

P on Roll

Cruise around, with good P, the control should feel precise and the quad should follow your sticks very closely. Try to do some sharp turns (where you use both the throttle and roll), if P is too low the quad will dip to one side (like a wobble), but when P is too high, you will get fast oscillations. When P is right, you should get minimum oscillations when doing sharp turns.

P on Pitch

Do a split-S (where you move both the throttle and pitch sticks), and as you increase throttle to recover, pay attention to the pitch movement. If the quad pitches up more than it should, then P is probably too low. but if you get some fast oscillations then you need to decrease P.

Fine tune it until you get to a point where the quad feels very responsive and nimble, making sure there is no excessive amount of vibration. Also listen to your motors; twitching motors are a sign of excess P gain.

D on Roll and Pitch

When you do aggressive maneuvers like flips and rolls, you will probably notice some prop wash and overshoots at the end of the move. Increasing D gain can help reduce that. Be careful because excessive D gain can introduce oscillations to your quad, and also make motors run hot, so use just enough to minimize prop wash. Another sign of too much D term is fast oscillations at the end of a roll or flip.

I on Roll

Bank your quad to the left and right to see if it's holding the angle well. You want it to just stay at the same attitude as you release the stick. If the quad can't hold the angle then I gain is too low and needs increasing. Do the same for pitching forward.

Your quad can drift with the wind, so you can increase I on a windy day depending how bad it is. I recommend setting I term just high enough to stay level, excessively high I gain can result in a stiff, robotic feeling.

P on Yaw

Yaw PID needs to be tuned separately. Default values usually work pretty well on all setups. Spin quickly on the yaw axis and see how it stops, if you get fast oscillations then decrease P, but if the quad dips one side, and then increase P.

Excessive Yaw P won't cause as much vibration as on roll and pitch because yaw movement is much weaker on a quad copter (lacking yaw authority). Look for any twitching and oscillations on the yaw axis. Another sign of Yaw P being too high, is the quad copter gaining altitude when doing rapid yaw movements. When Yaw P is right the spin should be clean.

Appendix B: LQR Approach

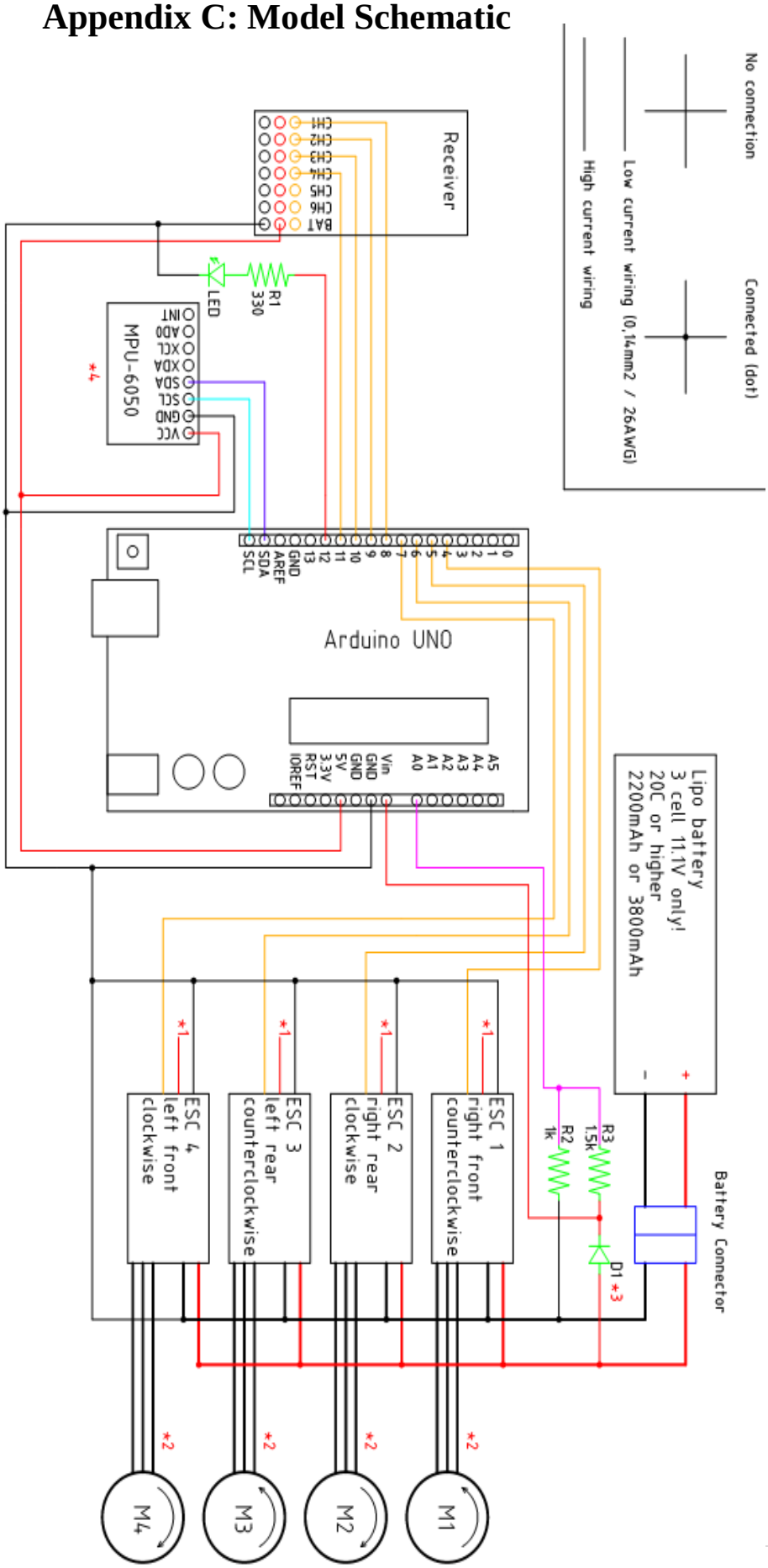
B-1 Classic LQR

Hoffmann et al. used this technique in the attitude loop. At low thrust levels, the control was satisfactory but at higher thrust levels, performance was degraded due to vibrations. A solution to this problem is to apply lower costs on attitude deviations by varying the matrix Q but this degrades tracking performance. A good compromise has to be found. Castillo implemented iteratively from simulation results LQR controller to make the quad rotor hover correctly. The feedback was applied to y and φ. Cowling is using the same kind of controller on x, y, z and ψ to follow a reference trajectory. However, his LQR controller has been designed with a model linearized at the hover. His simulation shows a flight path quite consistent with the reference trajectory.

B-2 State dependent LQR

Bouabdallah has already implemented such kind of controller in the closed loop system to stabilize the angular attitude of the UAV. His method was adapted through the robot trajectory. Indeed, in order to optimize the system for a larger flight envelope than the hover configuration, he has linearized the state space representation around each flight condition. Then, he has applied the classical techniques to get the associated LQR control gains at any state. As he didn't take into account the actuators dynamics, he obtained only average performance in his flight experiment. This technique has been called state dependent Riccati equation control.

Appendix C: Model Schematic



Do not connect the +5 bec wire	**Switch the two of three wires to reverse the motors**	**1N4001 or similar**	**The L3G4200D can also be used**
***1**	***2**	***3**	***4**

Appendix D: ESC calibration code

```
//Send the following characters / numbers via the serial monitor to change the mode
//
//r = print receiver signals.
//a = print quadcopter angles.
//1 = check rotation / vibrations for motor 1 (right front CCW).
//2 = check rotation / vibrations for motor 2 (right rear CW).
//3 = check rotation / vibrations for motor 3 (left rear CCW).
//4 = check rotation / vibrations for motor 4 (left front CW).
//5 = check vibrations for all motors together.


#include <Wire.h>                                         //Include the Wire.h library so we can communicate with the gyro.
#include <EEPROM.h>                                       //Include the EEPROM.h library so we can store information onto the EEPROM

//Declaring global variables
byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
byte eeprom_data[36], start, data;
boolean new_function_request,first_angle;
volatile int receiver_input_channel_1, receiver_input_channel_2, receiver_input_channel_3, receiver_input_channel_4;
int esc_1, esc_2, esc_3, esc_4;
int counter_channel_1, counter_channel_2, counter_channel_3, counter_channel_4;
int receiver_input[5];
int loop_counter, gyro_address, vibration_counter;
int temperature;
long acc_x, acc_y, acc_z, acc_total_vector[20], acc_av_vector, vibration_total_result;
unsigned long timer_channel_1, timer_channel_2, timer_channel_3, timer_channel_4, esc_timer, esc_loop_timer;
```

```
unsigned long zero_timer, timer_1, timer_2, timer_3, timer_4, current_time;

int acc_axis[4], gyro_axis[4];
double gyro_pitch, gyro_roll, gyro_yaw;
float angle_roll_acc, angle_pitch_acc, angle_pitch, angle_roll;
int cal_int;
double gyro_axis_cal[4];

//Setup routine
void setup(){
  Serial.begin(57600);                                                    //Start the serial port.
  Wire.begin();                                                           //Start the wire library as master
  TWBR = 12;                                                              //Set the I2C clock speed to 400kHz.

  //Arduino Uno pins default to inputs, so they don't need to be explicitly declared as inputs.
  DDRD |= B11110000;                                                      //Configure digital poort 4, 5, 6 and 7 as output.
  DDRB |= B00010000;                                                      //Configure digital poort 12 as output.

  PCICR |= (1 << PCIE0);                                                  // set PCIE0 to enable PCMSK0 scan.
  PCMSK0 |= (1 << PCINT0);                                                // set PCINT0 (digital input 8) to trigger an interrupt on state change.
  PCMSK0 |= (1 << PCINT1);                                                // set PCINT1 (digital input 9)to trigger an interrupt on state change.
  PCMSK0 |= (1 << PCINT2);                                                // set PCINT2 (digital input 10)to trigger an interrupt on state change.
  PCMSK0 |= (1 << PCINT3);                                                // set PCINT3 (digital input 11)to trigger an interrupt on state change.

  for(data = 0; data <= 35; data++)eeprom_data[data] = EEPROM.read(data);  //Read EEPROM for faster data access

  gyro_address = eeprom_data[32];                                         //Store the gyro address in the variable.
  set_gyro_registers();                                                   //Set the specific gyro registers.

  //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(500);                                                           //Wait for 500ms.
    digitalWrite(12, !digitalRead(12));                                   //Change the led status to indicate error.
  }
  wait_for_receiver();                                                    //Wait until the receiver is active.
  zero_timer = micros();                                                  //Set the zero_timer for the first loop.

  while(Serial.available())data = Serial.read();                          //Empty the serial buffer.
  data = 0;                                                               //Set the data variable back to zero.
}

//Main program loop
void loop(){
  while(zero_timer + 4000 > micros());                                    //Start the pulse after 4000 micro seconds.
  zero_timer = micros();                                                  //Reset the zero timer.

  if(Serial.available() > 0){
    data = Serial.read();                                                 //Read the incomming byte.
    delay(100);                                                           //Wait for any other bytes to come in
    while(Serial.available() > 0)loop_counter = Serial.read();            //Empty the Serial buffer.
    new_function_request = true;                                          //Set the new request flag.
    loop_counter = 0;                                                     //Reset the loop_counter variable.
    cal_int = 0;                                                          //Reset the cal_int variable to undo the calibration.
    start = 0;                                                            //Set start to 0.

    first_angle = false;                                                  //Set first_angle to false.
    //Confirm the choice on the serial monitor.
    if(data == 'r')Serial.println("Reading receiver signals.");
    if(data == 'a')Serial.println("Print the quadcopter angles.");
    if(data == 'a')Serial.println("Gyro calibration starts in 2 seconds (don't move the quadcopter).");
    if(data == '1')Serial.println("Test motor 1 (right front CCW.)");
    if(data == '2')Serial.println("Test motor 2 (right rear CW.)");
    if(data == '3')Serial.println("Test motor 3 (left rear CCW.)");
    if(data == '4')Serial.println("Test motor 4 (left front CW.)");
    if(data == '5')Serial.println("Test all motors together");

    //Let's create a small delay so the message stays visible for 2.5 seconds.
    //We don't want the ESC's to beep and have to send a 1000us pulse to the ESC's.
    for(vibration_counter = 0; vibration_counter < 625; vibration_counter++){           //Do this loop 625 times
      delay(3);                                                           //Wait 3000us.
      esc_1 = 1000;                                                       //Set the pulse for ESC 1 to 1000us.
      esc_2 = 1000;                                                       //Set the pulse for ESC 1 to 1000us.
      esc_3 = 1000;                                                       //Set the pulse for ESC 1 to 1000us.
      esc_4 = 1000;                                                       //Set the pulse for ESC 1 to 1000us.
      esc_pulse_output();                                                 //Send the ESC control pulses.
    }
    vibration_counter = 0;                                                //Reset the vibration_counter variable.
  }

  receiver_input_channel_3 = convert_receiver_channel(3);                 //Convert the actual receiver signals for throttle to the standard 1000 - 2000us.
  if(receiver_input_channel_3 < 1025)new_function_request = false;        //If the throttle is in the lowest position set the request flag to false.
```

```
  if(data == 0 && new_function_request == false){                                  //Only start the calibration mode at first start.
    receiver_input_channel_3 = convert_receiver_channel(3);                        //Convert the actual receiver signals for throttle to the standard 1000 - 2000us.
    esc_1 = receiver_input_channel_3;                                              //Set the pulse for motor 1 equal to the throttle channel.
    esc_2 = receiver_input_channel_3;                                              //Set the pulse for motor 2 equal to the throttle channel.
    esc_3 = receiver_input_channel_3;                                              //Set the pulse for motor 3 equal to the throttle channel.
    esc_4 = receiver_input_channel_3;                                              //Set the pulse for motor 4 equal to the throttle channel.
    esc_pulse_output();                                                            //Send the ESC control pulses.
  }

  ////////////////////////////////////////////////////////////////////////////////////////////////
  //When user sends a 'r' print the receiver signals.
  ////////////////////////////////////////////////////////////////////////////////////////////////
  if(data == 'r'){
    loop_counter ++;                                                               //Increase the loop_counter variable.
    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.

    if(loop_counter == 125){                                                       //Print the receiver values when the loop_counter variable equals 250.
      print_signals();                                                             //Print the receiver values on the serial monitor.
      loop_counter = 0;                                                            //Reset the loop_counter variable.
    }

    //For starting the motors: throttle low and yaw left (step 1).
    if(receiver_input_channel_3 < 1050 && receiver_input_channel_4 < 1050)start = 1;

    //When yaw stick is back in the center position start the motors (step 2).
    if(start == 1 && receiver_input_channel_3 < 1050 && receiver_input_channel_4 > 1450)start = 2;
    //Stopping the motors: throttle low and yaw right.
    if(start == 2 && receiver_input_channel_3 < 1050 && receiver_input_channel_4 > 1950)start = 0;

    esc_1 = 1000;                                                                  //Set the pulse for ESC 1 to 1000us.
    esc_2 = 1000;                                                                  //Set the pulse for ESC 1 to 1000us.
    esc_3 = 1000;                                                                  //Set the pulse for ESC 1 to 1000us.
    esc_4 = 1000;                                                                  //Set the pulse for ESC 1 to 1000us.
    esc_pulse_output();                                                            //Send the ESC control pulses.
  }

  ////////////////////////////////////////////////////////////////////////////////////////////////
  //When user sends a '1, 2, 3, 4 or 5 test the motors.
  ////////////////////////////////////////////////////////////////////////////////////////////////
  if(data == '1' || data == '2' || data == '3' || data == '4' || data == '5'){     //If motor 1, 2, 3 or 4 is selected by the user.
    loop_counter ++;                                                               //Add 1 to the loop_counter variable.
    if(new_function_request == true && loop_counter == 250){                       //Wait for the throttle to be set to 0.
      Serial.print("Set throttle to 1000 (low). It's now set to: ");               //Print message on the serial monitor.
      Serial.println(receiver_input_channel_3);                                    //Print the actual throttle position.
      loop_counter = 0;                                                            //Reset the loop_counter variable.
    }
    if(new_function_request == false){                                             //When the throttle was in the lowest position do this.
      receiver_input_channel_3 = convert_receiver_channel(3);                      //Convert the actual receiver signals for throttle to the standard 1000 - 2000us.
      if(data == '1' || data == '5')esc_1 = receiver_input_channel_3;              //If motor 1 is requested set the pulse for motor 1 equal to the throttle channel.
      else esc_1 = 1000;                                                           //If motor 1 is not requested set the pulse for the ESC to 1000us (off).
      if(data == '2' || data == '5')esc_2 = receiver_input_channel_3;              //If motor 2 is requested set the pulse for motor 1 equal to the throttle channel.
      else esc_2 = 1000;                                                           //If motor 2 is not requested set the pulse for the ESC to 1000us (off).
      if(data == '3' || data == '5')esc_3 = receiver_input_channel_3;              //If motor 3 is requested set the pulse for motor 1 equal to the throttle channel.
      else esc_3 = 1000;                                                           //If motor 3 is not requested set the pulse for the ESC to 1000us (off).
      if(data == '4' || data == '5')esc_4 = receiver_input_channel_3;              //If motor 4 is requested set the pulse for motor 1 equal to the throttle channel.
      else esc_4 = 1000;                                                           //If motor 4 is not requested set the pulse for the ESC to 1000us (off).

      esc_pulse_output();                                                          //Send the ESC control pulses.

      //For balancing the propellors it's possible to use the accelerometer to measure the vibrations.
      if(eeprom_data[31] == 1){                                                    //The MPU-6050 is installed
        Wire.beginTransmission(gyro_address);                                      //Start communication with the gyro.
        Wire.write(0x3B);                                                          //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.
        acc_x = Wire.read()<<8|Wire.read();                                        //Add the low and high byte to the acc_x variable.
        acc_y = Wire.read()<<8|Wire.read();                                        //Add the low and high byte to the acc_y variable.
        acc_z = Wire.read()<<8|Wire.read();                                        //Add the low and high byte to the acc_z variable.

        acc_total_vector[0] = sqrt((acc_x*acc_x)+(acc_y*acc_y)+(acc_z*acc_z));     //Calculate the total accelerometer vector.

        acc_av_vector = acc_total_vector[0];                                       //Copy the total vector to the accelerometer average vector variable.

          for(start = 16; start > 0; start--){                                     //Do this loop 16 times to create an array of accelrometer vectors.
            acc_total_vector[start] = acc_total_vector[start - 1];                 //Shift every variable one position up in the array.
            acc_av_vector += acc_total_vector[start];                              //Add the array value to the acc_av_vector variable.
          }

          acc_av_vector /= 17;                                                     //Divide the acc_av_vector by 17 to get the avarage total accelerometer vector.

          if(vibration_counter < 20){                                              //If the vibration_counter is less than 20 do this.
            vibration_counter ++;                                                  //Increment the vibration_counter variable.
            vibration_total_result += abs(acc_total_vector[0] - acc_av_vector);    //Add the absolute difference between the avarage vector and current vector to the
          }
          else{
            vibration_counter = 0;                                                 //If the vibration_counter is equal or larger than 20 do this.
            Serial.println(vibration_total_result/50);                             //Print the total accelerometer vector divided by 50 on the serial monitor.
            vibration_total_result = 0;                                            //Reset the vibration_total_result variable.
          }
        }
      }
    }
    ////////////////////////////////////////////////////////////////////////////////////////////////
    //When user sends a 'a' display the quadcopter angles.
    ////////////////////////////////////////////////////////////////////////////////////////////////
    if(data == 'a'){

      if(cal_int != 2000){
        Serial.print("Calibrating the gyro");
```

```
    for (cal_int = 0; cal_int < 2000 ; cal_int ++){                          //Take 2000 readings for calibration.
      if(cal_int % 125 == 0){
        digitalWrite(12, !digitalRead(12));   //Change the led status to indicate calibration.
        Serial.print(".");
      }
      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.
      //We don't want the esc's to be beeping annoyingly. So let's give them a 1000us puls while calibrating the gyro.
      PORTD |= B11110000;                                                    //Set digital poort 4, 5, 6 and 7 high.
      delayMicroseconds(1000);                                               //Wait 1000us.
      PORTD &= B00001111;                                                    //Set digital poort 4, 5, 6 and 7 low.
      delay(3);                                                              //Wait 3 milliseconds before the next loop.
    }
    Serial.println(".");
    //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.
  }
  else{
    ///We don't want the esc's to be beeping annoyingly. So let's give them a 1000us puls while calibrating the gyro.
    PORTD |= B11110000;                                                      //Set digital poort 4, 5, 6 and 7 high.
    delayMicroseconds(1000);                                                 //Wait 1000us.
```

```
PORTD |= B11110000;                                                  //Set digital poort 4, 5, 6 and 7 high.
delayMicroseconds(1000);                                             //Wait 1000us.
PORTD &= B00001111;                                                  //Set digital poort 4, 5, 6 and 7 low.

//Let's get the current gyro data.
gyro_signalen();

//Gyro angle calculations
//0.0000611 = 1 / (250Hz / 65.5)
angle_pitch += gyro_pitch * 0.0000611;                               //Calculate the traveled pitch angle and add this to the angle_pitch variable.
angle_roll += gyro_roll * 0.0000611;                                 //Calculate the traveled roll angle and add this to the angle_roll variable.

//0.000001066 = 0.0000611 * (3.142(PI) / 180degr) The Arduino sin function is in radians
angle_pitch -= angle_roll * sin(gyro_yaw * 0.000001066);             //If the IMU has yawed transfer the roll angle to the pitch angel.
angle_roll += angle_pitch * sin(gyro_yaw * 0.000001066);             //If the IMU has yawed transfer the pitch angle to the roll angel.

//Accelerometer angle calculations
acc_total_vector[0] = sqrt((acc_x*acc_x)+(acc_y*acc_y)+(acc_z*acc_z));       //Calculate the total accelerometer vector.

//57.296 = 1 / (3.142 / 180) The Arduino asin function is in radians
angle_pitch_acc = asin((float)acc_y/acc_total_vector[0])* 57.296;    //Calculate the pitch angle.
angle_roll_acc = asin((float)acc_x/acc_total_vector[0])* -57.296;    //Calculate the roll angle.

if(!first_angle){
  angle_pitch = angle_pitch_acc;                                     //Set the pitch angle to the accelerometer angle.
  angle_roll = angle_roll_acc;                                       //Set the roll angle to the accelerometer angle.
  first_angle = true;

      first_angle = true;
    }
    else{
      angle_pitch = angle_pitch * 0.9996 + angle_pitch_acc * 0.0004;     //Correct the drift of the gyro pitch angle with the accelerometer pitch angle.
      angle_roll = angle_roll * 0.9996 + angle_roll_acc * 0.0004;        //Correct the drift of the gyro roll angle with the accelerometer roll angle.
    }

    //We can't print all the data at once. This takes to long and the angular readings will be off.
    if(loop_counter == 0)Serial.print("Pitch: ");
    if(loop_counter == 1)Serial.print(angle_pitch ,0);
    if(loop_counter == 2)Serial.print(" Roll: ");
    if(loop_counter == 3)Serial.print(angle_roll ,0);
    if(loop_counter == 4)Serial.print(" Yaw: ");
    if(loop_counter == 5)Serial.println(gyro_yaw / 65.5 ,0);

    loop_counter ++;
    if(loop_counter == 60)loop_counter = 0;
  }
 }
}


//This routine is called every time input 8, 9, 10 or 11 changed state.
ISR(PCINT0_vect){
```

```
  current_time = micros();
  //Channel 1=========================================
  if(PINB & B00000001){                                        //Is input 8 high?
    if(last_channel_1 == 0){                                   //Input 8 changed from 0 to 1.
      last_channel_1 = 1;                                      //Remember current input state.
      timer_1 = current_time;                                  //Set timer_1 to current_time.
    }
  }
  else if(last_channel_1 == 1){                                //Input 8 is not high and changed from 1 to 0.
    last_channel_1 = 0;                                        //Remember current input state.
    receiver_input[1] = current_time - timer_1;                 //Channel 1 is current_time - timer_1.
  }
  //Channel 2=========================================
  if(PINB & B00000010 ){                                       //Is input 9 high?
    if(last_channel_2 == 0){                                   //Input 9 changed from 0 to 1.
      last_channel_2 = 1;                                      //Remember current input state.
      timer_2 = current_time;                                  //Set timer_2 to current_time.
    }
  }
  else if(last_channel_2 == 1){                                //Input 9 is not high and changed from 1 to 0.
    last_channel_2 = 0;                                        //Remember current input state.
    receiver_input[2] = current_time - timer_2;                //Channel 2 is current_time - timer_2.
  }
  //Channel 3=========================================
  if(PINB & B00000100 ){                                       //Is input 10 high?
    if(last_channel_3 == 0){                                   //Input 10 changed from 0 to 1.
      last_channel_3 = 1;                                      //Remember current input state.
      timer_3 = current_time;                                  //Set timer_3 to current_time.
    }
  }
  else if(last_channel_3 == 1){                                //Input 10 is not high and changed from 1 to 0.
    last_channel_3 = 0;                                        //Remember current input state.
    receiver_input[3] = current_time - timer_3;                //Channel 3 is current_time - timer_3.
  }
  //Channel 4=========================================
  if(PINB & B00001000 ){                                       //Is input 11 high?
    if(last_channel_4 == 0){                                   //Input 11 changed from 0 to 1.
      last_channel_4 = 1;                                      //Remember current input state.
      timer_4 = current_time;                                  //Set timer_4 to current_time.
    }
  }
  else if(last_channel_4 == 1){                                //Input 11 is not high and changed from 1 to 0.
    last_channel_4 = 0;                                        //Remember current input state.
    receiver_input[4] = current_time - timer_4;                //Channel 4 is current_time - timer_4.
  }
}

//Checck if the receiver values are valid within 10 seconds
void wait_for_receiver(){
  byte zero = 0;                                               //Set all bits in the variable zero to 0
  while(zero < 15){                                            //Stay in this loop until the 4 lowest bits are set
    if(receiver_input[1] < 2100 && receiver_input[1] > 900)zero |= 0b00000001;  //Set bit 0 if the receiver pulse 1 is within the 900 - 2100 range

    if(receiver_input[1] < 2100 && receiver_input[1] > 900)zero |= 0b00000001;  //Set bit 0 if the receiver pulse 1 is within the 900 - 2100 range
    if(receiver_input[2] < 2100 && receiver_input[2] > 900)zero |= 0b00000010;  //Set bit 1 if the receiver pulse 2 is within the 900 - 2100 range
    if(receiver_input[3] < 2100 && receiver_input[3] > 900)zero |= 0b00000100;  //Set bit 2 if the receiver pulse 3 is within the 900 - 2100 range
    if(receiver_input[4] < 2100 && receiver_input[4] > 900)zero |= 0b00001000;  //Set bit 3 if the receiver pulse 4 is within the 900 - 2100 range
    delay(500);                                                //Wait 500 milliseconds
  }
}

//This part converts the actual receiver signals to a standardized 1000 - 1500 - 2000 microsecond value.
//The stored data in the EEPROM is used.
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[channel];                            //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(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;
}

void print_signals(){
  Serial.print("Start:");
  Serial.print(start);

  Serial.print("  Roll:");
  if(receiver_input_channel_1 - 1480 < 0)Serial.print("<<<");
  else if(receiver_input_channel_1 - 1520 > 0)Serial.print(">>>");
  else Serial.print("-+-");
  Serial.print(receiver_input_channel_1);

  Serial.print("  Pitch:");
  if(receiver_input_channel_2 - 1480 < 0)Serial.print("^^^");
  else if(receiver_input_channel_2 - 1520 > 0)Serial.print("vvv");
  else Serial.print("-+-");
  Serial.print(receiver_input_channel_2);

  Serial.print("  Throttle:");
  if(receiver_input_channel_3 - 1480 < 0)Serial.print("vvv");
  else if(receiver_input_channel_3 - 1520 > 0)Serial.print("^^^");
  else Serial.print("-+-");
  Serial.print(receiver_input_channel_3);

  Serial.print("  Yaw:");
  if(receiver_input_channel_4 - 1480 < 0)Serial.print("<<<");
  else if(receiver_input_channel_4 - 1520 > 0)Serial.print(">>>");
  else Serial.print("-+-");
  Serial.println(receiver_input_channel_4);
}

void esc_pulse_output(){
  zero_timer = micros();
  PORTD |= B11110000;                                                   //Set port 4, 5, 6 and 7 high at once
  timer_channel_1 = esc_1 + zero_timer;                                 //Calculate the time when digital port 4 is set low.
  timer_channel_2 = esc_2 + zero_timer;                                 //Calculate the time when digital port 5 is set low.
  timer_channel_3 = esc_3 + zero_timer;                                 //Calculate the time when digital port 6 is set low.
  timer_channel_4 = esc_4 + zero_timer;                                 //Calculate the time when digital port 7 is set low.

  while(PORTD >= 16){                                                   //Execute the loop until digital port 4 to 7 is low.
    esc_loop_timer = micros();                                          //Check the current time.
    if(timer_channel_1 <= esc_loop_timer)PORTD &= B11101111;            //When the delay time is expired, digital port 4 is set low.
    if(timer_channel_2 <= esc_loop_timer)PORTD &= B11011111;            //When the delay time is expired, digital port 5 is set low.
    if(timer_channel_3 <= esc_loop_timer)PORTD &= B10111111;            //When the delay time is expired, digital port 6 is set low.
    if(timer_channel_4 <= esc_loop_timer)PORTD &= B01111111;            //When the delay time is expired, digital port 7 is set low.
  }
}

void set_gyro_registers(){
  //Setup the MPU-6050
  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(0x1C);                                                   //We want to write to the ACCEL_CONFIG register (1A hex)
    Wire.write(0x10);                                                   //Set the register bits as 00010000 (+/- 8g full scale range)
    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(12,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 CONFIG register (1A hex)
    Wire.write(0x03);                                      //Set the register bits as 00000011 (Set Digital Low Pass Filter to ~43Hz)
    Wire.endTransmission();                                //End the transmission with the gyro

  }
}

void gyro_signalen(){
  //Read the MPU-6050
  if(eeprom_data[31] == 1){
    Wire.beginTransmission(gyro_address);                  //Start communication with the gyro.
    Wire.write(0x3B);                                      //Start reading @ register 43h and auto increment with every read.
    Wire.endTransmission();                                //End the transmission.
    Wire.requestFrom(gyro_address,14);                     //Request 14 bytes from the gyro.
    while(Wire.available() < 14);                          //Wait until the 14 bytes are received
    acc_axis[1] = Wire.read()<<8|Wire.read();              //Add the low and high byte to the acc_x variable.
    acc_axis[2] = Wire.read()<<8|Wire.read();              //Add the low and high byte to the acc_y variable.
    acc_axis[3] = Wire.read()<<8|Wire.read();              //Add the low and high byte to the acc_z variable.
    temperature = Wire.read()<<8|Wire.read();              //Add the low and high byte to the temperature variable.
    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];     //Set gyro_roll to the correct axis that was stored in the EEPROM.
  if(eeprom_data[28] & 0b10000000)gyro_roll *= -1;         //Invert gyro_roll if the MSB of EEPROM bit 28 is set.
  gyro_pitch = gyro_axis[eeprom_data[29] & 0b00000011];    //Set gyro_pitch to the correct axis that was stored in the EEPROM.
  if(eeprom_data[29] & 0b10000000)gyro_pitch *= -1;        //Invert gyro_pitch if the MSB of EEPROM bit 29 is set.
  gyro_yaw = gyro_axis[eeprom_data[30] & 0b00000011];      //Set gyro_yaw to the correct axis that was stored in the EEPROM.
  if(eeprom_data[30] & 0b10000000)gyro_yaw *= -1;          //Invert gyro_yaw if the MSB of EEPROM bit 30 is set.

  acc_x = acc_axis[eeprom_data[29] & 0b00000011];          //Set acc_x to the correct axis that was stored in the EEPROM.
  if(eeprom_data[29] & 0b10000000)acc_x *= -1;             //Invert acc_x if the MSB of EEPROM bit 29 is set.
  acc_y = acc_axis[eeprom_data[28] & 0b00000011];          //Set acc_y to the correct axis that was stored in the EEPROM.
  if(eeprom_data[28] & 0b10000000)acc_y *= -1;             //Invert acc_y if the MSB of EEPROM bit 28 is set.
  acc_z = acc_axis[eeprom_data[30] & 0b00000011];          //Set acc_z to the correct axis that was stored in the EEPROM.
  if(eeprom_data[30] & 0b10000000)acc_z *= -1;             //Invert acc_z if the MSB of EEPROM bit 30 is set.
```

Appendix E : SETUP code

```
#include <Wire.h>               //Include the Wire.h library so we can communicate with the gyro
#include <EEPROM.h>             //Include the EEPROM.h library so we can store information onto the EEPROM

//Declaring Global Variables
byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
byte lowByte, highByte, type, gyro_address, error, clockspeed_ok;
byte channel_1_assign, channel_2_assign, channel_3_assign, channel_4_assign;
byte roll_axis, pitch_axis, yaw_axis;
byte receiver_check_byte, gyro_check_byte;
volatile int receiver_input_channel_1, receiver_input_channel_2, receiver_input_channel_3, receiver_input_channel_4;
int center_channel_1, center_channel_2, center_channel_3, center_channel_4;
int high_channel_1, high_channel_2, high_channel_3, high_channel_4;
int low_channel_1, low_channel_2, low_channel_3, low_channel_4;
int address, cal_int;
unsigned long timer, timer_1, timer_2, timer_3, timer_4, current_time;
float gyro_pitch, gyro_roll, gyro_yaw;
float gyro_roll_cal, gyro_pitch_cal, gyro_yaw_cal;


//Setup routine
void setup(){
  pinMode(12, OUTPUT);
  //Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as inputs
  PCICR |= (1 << PCIE0);    // set PCIE0 to enable PCMSK0 scan
  PCMSK0 |= (1 << PCINT0);  // set PCINT0 (digital input 8) to trigger an interrupt on state change
  PCMSK0 |= (1 << PCINT1);  // set PCINT1 (digital input 9)to trigger an interrupt on state change
  PCMSK0 |= (1 << PCINT2);  // set PCINT2 (digital input 10)to trigger an interrupt on state change
  PCMSK0 |= (1 << PCINT3);  // set PCINT3 (digital input 11)to trigger an interrupt on state change
  Wire.begin();             //Start the I2C as master
  Serial.begin(57600);      //Start the serial connetion @ 57600bps
  delay(250);               //Give the gyro time to start
}
//Main program
void loop(){
  //Show the YMFC-3D V2 intro
  intro();

  Serial.println(F(""));
  Serial.println(F("======================================================"));
  Serial.println(F("System check"));
  Serial.println(F("======================================================"));
  delay(1000);
  Serial.println(F("Checking I2C clock speed."));
  delay(1000);

  TWBR = 12;                    //Set the I2C clock speed to 400kHz.

  #if F_CPU == 16000000L        //If the clock speed is 16MHz include the next code line when compiling
    clockspeed_ok = 1;          //Set clockspeed_ok to 1
  #endif                        //End of if statement

  if(TWBR == 12 && clockspeed_ok){
    Serial.println(F("I2C clock speed is correctly set to 400kHz."));
  }
  else{
    Serial.println(F("I2C clock speed is not set to 400kHz. (ERROR 8)"));
    error = 1;
  }

  if(error == 0){
    Serial.println(F(""));
    Serial.println(F("======================================================"));
    Serial.println(F("Transmitter setup"));
    Serial.println(F("======================================================"));
    delay(1000);
    Serial.print(F("Checking for valid receiver signals."));
    //Wait 10 seconds until all receiver inputs are valid
    wait_for_receiver();
    Serial.println(F(""));
  }
  //Quit the program in case of an error
  if(error == 0){
    delay(2000);
    Serial.println(F("Place all sticks and subtrims in the center position within 10 seconds."));
    for(int i = 9;i > 0;i--){
      delay(1000);
      Serial.print(i);
      Serial.print(" ");
    }
    Serial.println(" ");
    //Store the central stick positions
```

```
    center_channel_1 = receiver_input_channel_1;
    center_channel_2 = receiver_input_channel_2;
    center_channel_3 = receiver_input_channel_3;
    center_channel_4 = receiver_input_channel_4;
    Serial.println(F(""));
    Serial.println(F("Center positions stored."));
    Serial.print(F("Digital input 08 = "));
    Serial.println(receiver_input_channel_1);
    Serial.print(F("Digital input 09 = "));
    Serial.println(receiver_input_channel_2);
    Serial.print(F("Digital input 10 = "));
    Serial.println(receiver_input_channel_3);
    Serial.print(F("Digital input 11 = "));
    Serial.println(receiver_input_channel_4);
    Serial.println(F(""));
    Serial.println(F(""));
  }
  if(error == 0){
    Serial.println(F("Move the throttle stick to full throttle and back to center"));
    //Check for throttle movement
    check_receiver_inputs(1);
    Serial.print(F("Throttle is connected to digital input "));
    Serial.println((channel_3_assign & 0b00000111) + 7);
    if(channel_3_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));
    else Serial.println(F("Channel inverted = no"));
    wait_sticks_zero();
    Serial.println(F(""));
    Serial.println(F(""));
    Serial.println(F("Move the roll stick to simulate left wing up and back to center"));
    //Check for throttle movement
    check_receiver_inputs(2);
    Serial.print(F("Roll is connected to digital input "));
    Serial.println((channel_1_assign & 0b00000111) + 7);
    if(channel_1_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));
    else Serial.println(F("Channel inverted = no"));
    wait_sticks_zero();
  }
  if(error == 0){
    Serial.println(F(""));
    Serial.println(F(""));
    Serial.println(F("Move the pitch stick to simulate nose up and back to center"));
    //Check for throttle movement
    check_receiver_inputs(3);
    Serial.print(F("Pitch is connected to digital input "));
    Serial.println((channel_2_assign & 0b00000111) + 7);
    if(channel_2_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));
    else Serial.println(F("Channel inverted = no"));
    wait_sticks_zero();
  }
  if(error == 0){
    Serial.println(F(""));
    Serial.println(F(""));
    Serial.println(F("Move the yaw stick to simulate nose right and back to center"));
    //Check for throttle movement
    check_receiver_inputs(4);
    Serial.print(F("Yaw is connected to digital input "));
    Serial.println((channel_4_assign & 0b00000111) + 7);
    if(channel_4_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));
    else Serial.println(F("Channel inverted = no"));
    wait_sticks_zero();
  }
  if(error == 0){
    Serial.println(F(""));
    Serial.println(F(""));
    Serial.println(F("Gently move all the sticks simultaneously to their extends"));
    Serial.println(F("When ready put the sticks back in their center positions"));
    //Register the min and max values of the receiver channels
    register_min_max();
    Serial.println(F(""));
    Serial.println(F(""));
    Serial.println(F("High, low and center values found during setup"));
    Serial.print(F("Digital input 08 values:"));
    Serial.print(low_channel_1);
    Serial.print(F(" - "));
    Serial.print(center_channel_1);
    Serial.print(F(" - "));
    Serial.println(high_channel_1);
    Serial.print(F("Digital input 09 values:"));
    Serial.print(low_channel_2);
    Serial.print(F(" - "));
```

```
    Serial.println(high_channel_2);
    Serial.print(F("Digital input 10 values:"));
    Serial.print(low_channel_3);
    Serial.print(F(" - "));
    Serial.print(center_channel_3);
    Serial.print(F(" - "));
    Serial.println(high_channel_3);
    Serial.print(F("Digital input 11 values:"));
    Serial.print(low_channel_4);
    Serial.print(F(" - "));
    Serial.print(center_channel_4);
    Serial.print(F(" - "));
    Serial.println(high_channel_4);
    Serial.println(F("Move stick 'nose up' and back to center to continue"));
    check_to_continue();
  }

  if(error == 0){
    //What gyro is connected
    Serial.println(F(""));
    Serial.println(F("=========================================================="));
    Serial.println(F("Gyro search"));
    Serial.println(F("=========================================================="));
    delay(2000);

    Serial.println(F("Searching for MPU-6050 on address 0x68/104"));
    delay(1000);
    if(search_gyro(0x68, 0x75) == 0x68){
      Serial.println(F("MPU-6050 found on address 0x68"));
      type = 1;
      gyro_address = 0x68;
    }

    if(type == 0){
      Serial.println(F("Searching for MPU-6050 on address 0x69/105"));
      delay(1000);
      if(search_gyro(0x69, 0x75) == 0x68){
        Serial.println(F("MPU-6050 found on address 0x69"));
        type = 1;
        gyro_address = 0x69;
      }
    }

    if(type == 0){
      Serial.println(F("Searching for L3G4200D on address 0x68/104"));
      delay(1000);
      if(search_gyro(0x68, 0x0F) == 0xD3){
        Serial.println(F("L3G4200D found on address 0x68"));
        type = 2;
        gyro_address = 0x68;
      }
    }

    if(type == 0){
    Serial.println(F("Searching for L3G4200D on address 0x69/105"));
    delay(1000);
    if(search_gyro(0x69, 0x0F) == 0xD3){
      Serial.println(F("L3G4200D found on address 0x69"));
      type = 2;
      gyro_address = 0x69;
    }
  }

  if(type == 0){
    Serial.println(F("Searching for L3GD20H on address 0x6A/106"));
    delay(1000);
    if(search_gyro(0x6A, 0x0F) == 0xD7){
      Serial.println(F("L3GD20H found on address 0x6A"));
      type = 3;
      gyro_address = 0x6A;
    }
  }

  if(type == 0){
   Serial.println(F("Searching for L3GD20H on address 0x6B/107"));
    delay(1000);
    if(search_gyro(0x6B, 0x0F) == 0xD7){
      Serial.println(F("L3GD20H found on address 0x6B"));
      type = 3;
      gyro_address = 0x6B;
    }
  }
```

```
  if(type == 0){
    Serial.println(F("No gyro device found!!! (ERROR 3)"));
    error = 1;
  }

  else{
    delay(3000);
    Serial.println(F(""));
    Serial.println(F("======================================================"));
    Serial.println(F("Gyro register settings"));
    Serial.println(F("======================================================"));
    start_gyro(); //Setup the gyro for further use
  }
}

//If the gyro is found we can setup the correct gyro axes.
if(error == 0){
  delay(3000);
  Serial.println(F(""));
  Serial.println(F("======================================================"));
  Serial.println(F("Gyro calibration"));
  Serial.println(F("======================================================"));
  Serial.println(F("Don't move the quadcopter!! Calibration starts in 3 seconds"));
  delay(3000);
  Serial.println(F("Calibrating the gyro, this will take +/- 8 seconds"));
  Serial.print(F("Please wait"));
  //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 % 100 == 0)Serial.print(F("."));                  //Print dot to indicate calibration.
     gyro_signalen();                                             //Read the gyro output.
     gyro_roll_cal += gyro_roll;                                  //Ad roll value to gyro_roll_cal.
     gyro_pitch_cal += gyro_pitch;                                //Ad pitch value to gyro_pitch_cal.
     gyro_yaw_cal += gyro_yaw;                                    //Ad yaw value to gyro_yaw_cal.
     delay(4);                                                    //Wait 3 milliseconds before the next loop.
   }
   //Now that we have 2000 measures, we need to devide by 2000 to get the average gyro offset.
   gyro_roll_cal /= 2000;                                         //Divide the roll total by 2000.
   gyro_pitch_cal /= 2000;                                        //Divide the pitch total by 2000.
   gyro_yaw_cal /= 2000;                                          //Divide the yaw total by 2000.

   //Show the calibration results
   Serial.println(F(""));
   Serial.print(F("Axis 1 offset="));
   Serial.println(gyro_roll_cal);
   Serial.print(F("Axis 2 offset="));
   Serial.println(gyro_pitch_cal);
   Serial.print(F("Axis 3 offset="));
   Serial.println(gyro_yaw_cal);
   Serial.println(F(""));

   Serial.println(F("======================================================"));
   Serial.println(F("Gyro axes configuration"));
   Serial.println(F("======================================================"));
  Serial.println(F("Lift the left side of the quadcopter to a 45 degree angle within 10 seconds"));
  //Check axis movement
  check_gyro_axes(1);
  if(error == 0){
    Serial.println(F("OK!"));
    Serial.print(F("Angle detection = "));
    Serial.println(roll_axis & 0b00000011);
    if(roll_axis & 0b10000000)Serial.println(F("Axis inverted = yes"));
    else Serial.println(F("Axis inverted = no"));
    Serial.println(F("Put the quadcopter back in its original position"));
    Serial.println(F("Move stick 'nose up' and back to center to continue"));
    check_to_continue();

    //Detect the nose up movement
    Serial.println(F(""));
    Serial.println(F(""));
    Serial.println(F("Lift the nose of the quadcopter to a 45 degree angle within 10 seconds"));
    //Check axis movement
    check_gyro_axes(2);
  }
  if(error == 0){
    Serial.println(F("OK!"));
    Serial.print(F("Angle detection = "));
    Serial.println(pitch_axis & 0b00000011);
    if(pitch_axis & 0b10000000)Serial.println(F("Axis inverted = yes"));
    else Serial.println(F("Axis inverted = no"));
    Serial.println(F("Put the quadcopter back in its original position"));
```

```
    Serial.println(F("Move stick 'nose up' and back to center to continue"));
    check_to_continue();

    //Detect the nose right movement
    Serial.println(F(""));
    Serial.println(F(""));
    Serial.println(F("Rotate the nose of the quadcopter 45 degree to the right within 10 seconds")
    //Check axis movement
    check_gyro_axes(3);
  }
  if(error == 0){
    Serial.println(F("OK!"));
    Serial.print(F("Angle detection = "));
    Serial.println(yaw_axis & 0b00000011);
    if(yaw_axis & 0b10000000)Serial.println(F("Axis inverted = yes"));
    else Serial.println(F("Axis inverted = no"));
    Serial.println(F("Put the quadcopter back in its original position"));
    Serial.println(F("Move stick 'nose up' and back to center to continue"));
    check_to_continue();
  }
}
if(error == 0){
  Serial.println(F(""));
  Serial.println(F("=================================================="));
  Serial.println(F("LED test"));
  Serial.println(F("=================================================="));
  digitalWrite(12, HIGH);
  Serial.println(F("The LED should now be lit"));
  Serial.println(F("Move stick 'nose up' and back to center to continue"));
  check_to_continue();
  digitalWrite(12, LOW);
}

Serial.println(F(""));

if(error == 0){
  Serial.println(F("=========================================================="));
  Serial.println(F("Final setup check"));
  Serial.println(F("=========================================================="));
  delay(1000);
  if(receiver_check_byte == 0b00001111){
    Serial.println(F("Receiver channels ok"));
  }
  else{
    Serial.println(F("Receiver channel verification failed!!! (ERROR 6)"));
    error = 1;
  }
  delay(1000);
  if(gyro_check_byte == 0b00000111){
    Serial.println(F("Gyro axes ok"));
  }
  else{
    Serial.println(F("Gyro exes verification failed!!! (ERROR 7)"));
    error = 1;
   }
}

if(error == 0){
  //If all is good, store the information in the EEPROM
  Serial.println(F(""));
  Serial.println(F("=========================================================="));
  Serial.println(F("Storing EEPROM information"));
  Serial.println(F("=========================================================="));
  Serial.println(F("Writing EEPROM"));
  delay(1000);
  Serial.println(F("Done!"));
  EEPROM.write(0, center_channel_1 & 0b11111111);
  EEPROM.write(1, center_channel_1 >> 8);
  EEPROM.write(2, center_channel_2 & 0b11111111);
  EEPROM.write(3, center_channel_2 >> 8);
  EEPROM.write(4, center_channel_3 & 0b11111111);
  EEPROM.write(5, center_channel_3 >> 8);
  EEPROM.write(6, center_channel_4 & 0b11111111);
  EEPROM.write(7, center_channel_4 >> 8);
  EEPROM.write(8, high_channel_1 & 0b11111111);
  EEPROM.write(9, high_channel_1 >> 8);
  EEPROM.write(10, high_channel_2 & 0b11111111);
  EEPROM.write(11, high_channel_2 >> 8);
  EEPROM.write(12, high_channel_3 & 0b11111111);
  EEPROM.write(13, high_channel_3 >> 8);
  EEPROM.write(14, high_channel_4 & 0b11111111);
```

```
  EEPROM.write(15, high_channel_4 >> 8);
  EEPROM.write(16, low_channel_1 & 0b11111111);
  EEPROM.write(17, low_channel_1 >> 8);
  EEPROM.write(18, low_channel_2 & 0b11111111);
  EEPROM.write(19, low_channel_2 >> 8);
  EEPROM.write(20, low_channel_3 & 0b11111111);
  EEPROM.write(21, low_channel_3 >> 8);
  EEPROM.write(22, low_channel_4 & 0b11111111);
  EEPROM.write(23, low_channel_4 >> 8);
  EEPROM.write(24, channel_1_assign);
  EEPROM.write(25, channel_2_assign);
  EEPROM.write(26, channel_3_assign);
  EEPROM.write(27, channel_4_assign);
  EEPROM.write(28, roll_axis);
  EEPROM.write(29, pitch_axis);
  EEPROM.write(30, yaw_axis);
  EEPROM.write(31, type);
  EEPROM.write(32, gyro_address);
  //Write the EEPROM signature
  EEPROM.write(33, 'J');
  EEPROM.write(34, 'M');
  EEPROM.write(35, 'B');


  //To make sure evrything is ok, verify the EEPROM data.
  Serial.println(F("Verify EEPROM data"));
  delay(1000);
if(center_channel_1 != ((EEPROM.read(1) << 8) | EEPROM.read(0)))error = 1;
if(center_channel_2 != ((EEPROM.read(3) << 8) | EEPROM.read(2)))error = 1;
if(center_channel_3 != ((EEPROM.read(5) << 8) | EEPROM.read(4)))error = 1;
if(center_channel_4 != ((EEPROM.read(7) << 8) | EEPROM.read(6)))error = 1;

if(high_channel_1 != ((EEPROM.read(9) << 8) | EEPROM.read(8)))error = 1;
if(high_channel_2 != ((EEPROM.read(11) << 8) | EEPROM.read(10)))error = 1;
if(high_channel_3 != ((EEPROM.read(13) << 8) | EEPROM.read(12)))error = 1;
if(high_channel_4 != ((EEPROM.read(15) << 8) | EEPROM.read(14)))error = 1;

if(low_channel_1 != ((EEPROM.read(17) << 8) | EEPROM.read(16)))error = 1;
if(low_channel_2 != ((EEPROM.read(19) << 8) | EEPROM.read(18)))error = 1;
if(low_channel_3 != ((EEPROM.read(21) << 8) | EEPROM.read(20)))error = 1;
if(low_channel_4 != ((EEPROM.read(23) << 8) | EEPROM.read(22)))error = 1;

if(channel_1_assign != EEPROM.read(24))error = 1;
if(channel_2_assign != EEPROM.read(25))error = 1;
if(channel_3_assign != EEPROM.read(26))error = 1;
if(channel_4_assign != EEPROM.read(27))error = 1;

if(roll_axis != EEPROM.read(28))error = 1;
if(pitch_axis != EEPROM.read(29))error = 1;
if(yaw_axis != EEPROM.read(30))error = 1;
if(type != EEPROM.read(31))error = 1;
if(gyro_address != EEPROM.read(32))error = 1;

if('J' != EEPROM.read(33))error = 1;
     if('M' != EEPROM.read(34))error = 1;
     if('B' != EEPROM.read(35))error = 1;

     if(error == 1)Serial.println(F("EEPROM verification failed!!! (ERROR 5)"));
     else Serial.println(F("Verification done"));
  }


  if(error == 0){
    Serial.println(F("Setup is finished."));
    Serial.println(F("You can now calibrate the esc's and upload the YMFC-AL code."));
  }
  else{
   Serial.println(F("The setup is aborted due to an error."));
   Serial.println(F("Check the Q and A page of the YMFC-AL project on:"));
   Serial.println(F("www.brokking.net for more information about this error."));
  }
  while(1);
}

//Search for the gyro and check the Who_am_I register
byte search_gyro(int gyro_address, int who_am_i){
  Wire.beginTransmission(gyro_address);
  Wire.write(who_am_i);
  Wire.endTransmission();
  Wire.requestFrom(gyro_address, 1);
  timer = millis() + 100;
```

```
  while(Wire.available() < 1 && timer > millis());
  lowByte = Wire.read();
  address = gyro_address;
  return lowByte;
}

void start_gyro(){
  //Setup the L3G4200D or L3GD20H
  if(type == 2 || type == 3){
    Wire.beginTransmission(address);                    //Start communication with the gyro with the address found during search
    Wire.write(0x20);                                   //We want to write to register 1 (20 hex)
    Wire.write(0x0F);                                   //Set the register bits as 00001111 (Turn on the gyro and enable all axis)
    Wire.endTransmission();                             //End the transmission with the gyro

    Wire.beginTransmission(address);                    //Start communication with the gyro (adress 1101001)
    Wire.write(0x20);                                   //Start reading @ register 28h and auto increment with every read
    Wire.endTransmission();                             //End the transmission
    Wire.requestFrom(address, 1);                       //Request 6 bytes from the gyro
    while(Wire.available() < 1);                        //Wait until the 1 byte is received
    Serial.print(F("Register 0x20 is set to:"));
    Serial.println(Wire.read(),BIN);

    Wire.beginTransmission(address);                    //Start communication with the gyro  with the address found during search
    Wire.write(0x23);                                   //We want to write to register 4 (23 hex)
    Wire.write(0x90);                                   //Set the register bits as 10010000 (Block Data Update active & 500dps full scale)
    Wire.endTransmission();                             //End the transmission with the gyro
    Wire.beginTransmission(address);                          //Start communication with the gyro (adress 1101001)
    Wire.write(0x23);                                         //Start reading @ register 28h and auto increment with every read
    Wire.endTransmission();                                   //End the transmission
    Wire.requestFrom(address, 1);                             //Request 6 bytes from the gyro
    while(Wire.available() < 1);                              //Wait until the 1 byte is received
    Serial.print(F("Register 0x23 is set to:"));
    Serial.println(Wire.read(),BIN);

  }
  //Setup the MPU-6050
  if(type == 1){

    Wire.beginTransmission(address);                          //Start communication with the gyro
    Wire.write(0x6B);                                         //PWR_MGMT_1 register
    Wire.write(0x00);                                         //Set to zero to turn on the gyro
    Wire.endTransmission();                                   //End the transmission

    Wire.beginTransmission(address);                          //Start communication with the gyro
    Wire.write(0x6B);                                         //Start reading @ register 28h and auto increment with every read
    Wire.endTransmission();                                   //End the transmission
    Wire.requestFrom(address, 1);                             //Request 1 bytes from the gyro
    while(Wire.available() < 1);                              //Wait until the 1 byte is received
    Serial.print(F("Register 0x6B is set to:"));
    Serial.println(Wire.read(),BIN);

    Wire.beginTransmission(address);                          //Start communication with the gyro
    Wire.write(0x1B);                                         //GYRO_CONFIG register
     Wire.write(0x08);                                         //Set the register bits as 00001000 (500dps full scale)
     Wire.endTransmission();                                   //End the transmission

     Wire.beginTransmission(address);                          //Start communication with the gyro (adress 1101001)
     Wire.write(0x1B);                                         //Start reading @ register 28h and auto increment with every read
     Wire.endTransmission();                                   //End the transmission
     Wire.requestFrom(address, 1);                             //Request 1 bytes from the gyro
     while(Wire.available() < 1);                              //Wait until the 1 byte is received
     Serial.print(F("Register 0x1B is set to:"));
     Serial.println(Wire.read(),BIN);

   }
}

void gyro_signalen(){
  if(type == 2 || type == 3){
    Wire.beginTransmission(address);                          //Start communication with the gyro
    Wire.write(168);                                          //Start reading @ register 28h and auto increment with every read
    Wire.endTransmission();                                   //End the transmission
    Wire.requestFrom(address, 6);                             //Request 6 bytes from the gyro
    while(Wire.available() < 6);                              //Wait until the 6 bytes are received
    lowByte = Wire.read();                                    //First received byte is the low part of the angular data
    highByte = Wire.read();                                   //Second received byte is the high part of the angular data
    gyro_roll = ((highByte<<8)|lowByte);                      //Multiply highByte by 256 (shift left by 8) and ad lowByte
    if(cal_int == 2000)gyro_roll -= gyro_roll_cal;            //Only compensate after the calibration
    lowByte = Wire.read();                                    //First received byte is the low part of the angular data
    highByte = Wire.read();                                   //Second received byte is the high part of the angular data
```

```
    gyro_pitch = ((highByte<<8)|lowByte);                    //Multiply highByte by 256 (shift left by 8) and ad lowByte
    if(cal_int == 2000)gyro_pitch -= gyro_pitch_cal;         //Only compensate after the calibration
    lowByte = Wire.read();                                   //First received byte is the low part of the angular data
    highByte = Wire.read();                                  //Second received byte is the high part of the angular data
    gyro_yaw = ((highByte<<8)|lowByte);                      //Multiply highByte by 256 (shift left by 8) and ad lowByte
    if(cal_int == 2000)gyro_yaw -= gyro_yaw_cal;             //Only compensate after the calibration
  }
  if(type == 1){
    Wire.beginTransmission(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(address,6);                             //Request 6 bytes from the gyro
    while(Wire.available() < 6);                             //Wait until the 6 bytes are received
    gyro_roll=Wire.read()<<8|Wire.read();                    //Read high and low part of the angular data
    if(cal_int == 2000)gyro_roll -= gyro_roll_cal;           //Only compensate after the calibration
    gyro_pitch=Wire.read()<<8|Wire.read();                   //Read high and low part of the angular data
    if(cal_int == 2000)gyro_pitch -= gyro_pitch_cal;         //Only compensate after the calibration
    gyro_yaw=Wire.read()<<8|Wire.read();                     //Read high and low part of the angular data
    if(cal_int == 2000)gyro_yaw -= gyro_yaw_cal;             //Only compensate after the calibration
  }
}

//Check if a receiver input value is changing within 30 seconds
void check_receiver_inputs(byte movement){
  byte trigger = 0;
  int pulse_length;
  timer = millis() + 30000;
  while(timer > millis() && trigger == 0){
    delay(250);
    if(receiver_input_channel_1 > 1750 || receiver_input_channel_1 < 1250){
      trigger = 1;
      receiver_check_byte |= 0b00000001;
      pulse_length = receiver_input_channel_1;
    }
    if(receiver_input_channel_2 > 1750 || receiver_input_channel_2 < 1250){
      trigger = 2;
      receiver_check_byte |= 0b00000010;
      pulse_length = receiver_input_channel_2;
    }
    if(receiver_input_channel_3 > 1750 || receiver_input_channel_3 < 1250){
      trigger = 3;
      receiver_check_byte |= 0b00000100;
      pulse_length = receiver_input_channel_3;
    }
    if(receiver_input_channel_4 > 1750 || receiver_input_channel_4 < 1250){
      trigger = 4;
      receiver_check_byte |= 0b00001000;
      pulse_length = receiver_input_channel_4;
    }
  }
  if(trigger == 0){
    error = 1;
    Serial.println(F("No stick movement detected in the last 30 seconds!!! (ERROR 2)"));
  }
  //Assign the stick to the function.
  else{
    if(movement == 1){
      channel_3_assign = trigger;
      if(pulse_length < 1250)channel_3_assign += 0b10000000;
    }
    if(movement == 2){
      channel_1_assign = trigger;
      if(pulse_length < 1250)channel_1_assign += 0b10000000;
    }
    if(movement == 3){
      channel_2_assign = trigger;
      if(pulse_length < 1250)channel_2_assign += 0b10000000;
    }
    if(movement == 4){
      channel_4_assign = trigger;
      if(pulse_length < 1250)channel_4_assign += 0b10000000;
    }
  }
}

void check_to_continue(){
  byte continue_byte = 0;
  while(continue_byte == 0){
    if(channel_2_assign == 0b00000001 && receiver_input_channel_1 > center_channel_1 + 150)continue_byte = 1;
    if(channel_2_assign == 0b10000001 && receiver_input_channel_1 < center_channel_1 - 150)continue_byte = 1;
    if(channel_2_assign == 0b00000010 && receiver_input_channel_2 > center_channel_2 + 150)continue_byte = 1;
    if(channel_2_assign == 0b10000010 && receiver_input_channel_2 < center_channel_2 - 150)continue_byte = 1;
    if(channel_2_assign == 0b00000011 && receiver_input_channel_3 > center_channel_3 + 150)continue_byte = 1;
    if(channel_2_assign == 0b10000011 && receiver_input_channel_3 < center_channel_3 - 150)continue_byte = 1;
    if(channel_2_assign == 0b00000100 && receiver_input_channel_4 > center_channel_4 + 150)continue_byte = 1;
    if(channel_2_assign == 0b10000100 && receiver_input_channel_4 < center_channel_4 - 150)continue_byte = 1;
    delay(100);
   }
   wait_sticks_zero();
 }

 //Check if the transmitter sticks are in the neutral position
 void wait_sticks_zero(){
   byte zero = 0;
   while(zero < 15){
     if(receiver_input_channel_1 < center_channel_1 + 20 && receiver_input_channel_1 > center_channel_1 - 20)zero |= 0b00000001;
     if(receiver_input_channel_2 < center_channel_2 + 20 && receiver_input_channel_2 > center_channel_2 - 20)zero |= 0b00000010;
     if(receiver_input_channel_3 < center_channel_3 + 20 && receiver_input_channel_3 > center_channel_3 - 20)zero |= 0b00000100;
     if(receiver_input_channel_4 < center_channel_4 + 20 && receiver_input_channel_4 > center_channel_4 - 20)zero |= 0b00001000;
     delay(100);
   }
 }

 //Checck if the receiver values are valid within 10 seconds
 void wait_for_receiver(){
   byte zero = 0;
   timer = millis() + 10000;
   while(timer > millis() && zero < 15){
```

```
    if(receiver_input_channel_1 < 2100 && receiver_input_channel_1 > 900)zero |= 0b00000001;
    if(receiver_input_channel_2 < 2100 && receiver_input_channel_2 > 900)zero |= 0b00000010;
    if(receiver_input_channel_3 < 2100 && receiver_input_channel_3 > 900)zero |= 0b00000100;
    if(receiver_input_channel_4 < 2100 && receiver_input_channel_4 > 900)zero |= 0b00001000;
    delay(500);
    Serial.print(F("."));
  }
  if(zero == 0){
    error = 1;
    Serial.println(F("."));
    Serial.println(F("No valid receiver signals found!!! (ERROR 1)"));
  }
  else Serial.println(F(" OK"));
}

//Register the min and max receiver values and exit when the sticks are back in the neutral position
void register_min_max(){
  byte zero = 0;
  low_channel_1 = receiver_input_channel_1;
  low_channel_2 = receiver_input_channel_2;
  low_channel_3 = receiver_input_channel_3;
  low_channel_4 = receiver_input_channel_4;
  while(receiver_input_channel_1 < center_channel_1 + 20 && receiver_input_channel_1 > center_channel_1 - 20)delay(250);
  Serial.println(F("Measuring endpoints...."));
  while(zero < 15){
    if(receiver_input_channel_1 < center_channel_1 + 20 && receiver_input_channel_1 > center_channel_1 - 20)zero |= 0b00000001;
    if(receiver_input_channel_2 < center_channel_2 + 20 && receiver_input_channel_2 > center_channel_2 - 20)zero |= 0b00000010;
   if(receiver_input_channel_3 < center_channel_3 + 20 && receiver_input_channel_3 > center_channel_3 - 20)zero |= 0b00000100;
   if(receiver_input_channel_4 < center_channel_4 + 20 && receiver_input_channel_4 > center_channel_4 - 20)zero |= 0b00001000;
   if(receiver_input_channel_1 < low_channel_1)low_channel_1 = receiver_input_channel_1;
   if(receiver_input_channel_2 < low_channel_2)low_channel_2 = receiver_input_channel_2;
   if(receiver_input_channel_3 < low_channel_3)low_channel_3 = receiver_input_channel_3;
   if(receiver_input_channel_4 < low_channel_4)low_channel_4 = receiver_input_channel_4;
   if(receiver_input_channel_1 > high_channel_1)high_channel_1 = receiver_input_channel_1;
   if(receiver_input_channel_2 > high_channel_2)high_channel_2 = receiver_input_channel_2;
   if(receiver_input_channel_3 > high_channel_3)high_channel_3 = receiver_input_channel_3;
   if(receiver_input_channel_4 > high_channel_4)high_channel_4 = receiver_input_channel_4;
   delay(100);
  }
}

//Check if the angular position of a gyro axis is changing within 10 seconds
void check_gyro_axes(byte movement){
  byte trigger_axis = 0;
  float gyro_angle_roll, gyro_angle_pitch, gyro_angle_yaw;
  //Reset all axes
  gyro_angle_roll = 0;
  gyro_angle_pitch = 0;
  gyro_angle_yaw = 0;
  gyro_signalen();
  timer = millis() + 10000;
  while(timer > millis() && gyro_angle_roll > -30 && gyro_angle_roll < 30 && gyro_angle_pitch > -30 && gyro_angle_pitch < 30 && gyro_angle_yaw > -30 && gyro_angle_yaw < 30){
    gyro_signalen();
    if(type == 2 || type == 3){
      gyro_angle_roll += gyro_roll * 0.00007;              //0.00007 = 17.5 (md/s) / 250(Hz)
      gyro_angle_pitch += gyro_pitch * 0.00007;
      gyro_angle_yaw += gyro_yaw * 0.00007;
    }
    if(type == 1){
      gyro_angle_roll += gyro_roll * 0.0000611;          // 0.0000611 = 1 / 65.5 (LSB degr/s) / 250(Hz)
      gyro_angle_pitch += gyro_pitch * 0.0000611;
      gyro_angle_yaw += gyro_yaw * 0.0000611;
    }

    delayMicroseconds(3700); //Loop is running @ 250Hz. +/-300us is used for communication with the gyro
  }
  //Assign the moved axis to the orresponding function (pitch, roll, yaw)
  if((gyro_angle_roll < -30 || gyro_angle_roll > 30) && gyro_angle_pitch > -30 && gyro_angle_pitch < 30 && gyro_angle_yaw > -30 && gyro_angle_yaw < 30){
    gyro_check_byte |= 0b00000001;
    if(gyro_angle_roll < 0)trigger_axis = 0b10000001;
    else trigger_axis = 0b00000001;
  }
  if((gyro_angle_pitch < -30 || gyro_angle_pitch > 30) && gyro_angle_roll > -30 && gyro_angle_roll < 30 && gyro_angle_yaw > -30 && gyro_angle_yaw < 30){
    gyro_check_byte |= 0b00000010;
    if(gyro_angle_pitch < 0)trigger_axis = 0b10000010;
    else trigger_axis = 0b00000010;
  }
  if((gyro_angle_yaw < -30 || gyro_angle_yaw > 30) && gyro_angle_roll > -30 && gyro_angle_roll < 30 && gyro_angle_pitch > -30 && gyro_angle_pitch < 30){
    gyro_check_byte |= 0b00000100;
    if(gyro_angle_yaw < 0)trigger_axis = 0b10000011;
    else trigger_axis = 0b00000011;
```

```
  if(trigger_axis == 0){
    error = 1;
    Serial.println(F("No angular motion is detected in the last 10 seconds!!! (ERROR 4)"));
  }
  else
  if(movement == 1)roll_axis = trigger_axis;
  if(movement == 2)pitch_axis = trigger_axis;
  if(movement == 3)yaw_axis = trigger_axis;

}

//This routine is called every time input 8, 9, 10 or 11 changed state
ISR(PCINT0_vect){
  current_time = micros();
  //Channel 1=========================================
  if(PINB & B00000001){                                        //Is input 8 high?
    if(last_channel_1 == 0){                                   //Input 8 changed from 0 to 1
      last_channel_1 = 1;                                      //Remember current input state
      timer_1 = current_time;                                  //Set timer_1 to current_time
    }
  }
  else if(last_channel_1 == 1){                                //Input 8 is not high and changed from 1 to 0
    last_channel_1 = 0;                                        //Remember current input state
    receiver_input_channel_1 = current_time - timer_1;         //Channel 1 is current_time - timer_1
  }
  //Channel 2=========================================
  if(PINB & B00000010 ){                                       //Is input 9 high?
   if(last_channel_2 == 0){                                    //Input 9 changed from 0 to 1
     last_channel_2 = 1;                                       //Remember current input state
     timer_2 = current_time;                                   //Set timer_2 to current_time
   }
 }
 else if(last_channel_2 == 1){                                 //Input 9 is not high and changed from 1 to 0
   last_channel_2 = 0;                                         //Remember current input state
   receiver_input_channel_2 = current_time - timer_2;          //Channel 2 is current_time - timer_2
 }
 //Channel 3=========================================
 if(PINB & B00000100 ){                                        //Is input 10 high?
   if(last_channel_3 == 0){                                    //Input 10 changed from 0 to 1
     last_channel_3 = 1;                                       //Remember current input state
     timer_3 = current_time;                                   //Set timer_3 to current_time
   }
 }
 else if(last_channel_3 == 1){                                 //Input 10 is not high and changed from 1 to 0
   last_channel_3 = 0;                                         //Remember current input state
   receiver_input_channel_3 = current_time - timer_3;          //Channel 3 is current_time - timer_3

 }
 //Channel 4=========================================
 if(PINB & B00001000 ){                                        //Is input 11 high?
   if(last_channel_4 == 0){                                    //Input 11 changed from 0 to 1
     last_channel_4 = 1;                                       //Remember current input state
     timer_4 = current_time;                                   //Set timer_4 to current_time
   }
   else if(last_channel_4 == 1){                                //Input 11 is not high and changed from 1 to 0
     last_channel_4 = 0;                                       //Remember current input state
     receiver_input_channel_4 = current_time - timer_4;        //Channel 4 is current_time - timer_4
   }
 }

//Intro subroutine
void intro(){
  Serial.println(F("======================================================"));
  delay(1500);
  Serial.println(F(""));
  Serial.println(F("Your"));
  delay(500);
  Serial.println(F("  Multicopter"));
  delay(500);
  Serial.println(F("    Flight"));
  delay(500);
  Serial.println(F("      Controller"));
  delay(1000);
  Serial.println(F(""));
  Serial.println(F("YMFC-AL Setup Program"));
  Serial.println(F(""));
  Serial.println(F("======================================================"));
  delay(1500);
  Serial.println(F("For support and questions: www.brokking.net"));
  Serial.println(F(""));
  Serial.println(F("Have fun!"));
}
```

Appendix F: Flight Controller code

```
#include <Wire.h>                         //Include the Wire.h library so we can communicate with the gyro.
#include <EEPROM.h>                       //Include the EEPROM.h library so we can store information onto the EEPROM

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//PID gain and limit settings
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
float pid_p_gain_roll = 1.3;              //Gain setting for the roll P-controller
float pid_i_gain_roll = 0.04;             //Gain setting for the roll I-controller
float pid_d_gain_roll = 18.0;             //Gain setting for the roll D-controller
int pid_max_roll = 400;                   //Maximum output of the PID-controller (+/-)

float pid_p_gain_pitch = pid_p_gain_roll;  //Gain setting for the pitch P-controller.
float pid_i_gain_pitch = pid_i_gain_roll;  //Gain setting for the pitch I-controller.
float pid_d_gain_pitch = pid_d_gain_roll;  //Gain setting for the pitch D-controller.
int pid_max_pitch = pid_max_roll;          //Maximum output of the PID-controller (+/-)

float pid_p_gain_yaw = 4.0;               //Gain setting for the pitch P-controller. //4.0
float pid_i_gain_yaw = 0.02;              //Gain setting for the pitch I-controller. //0.02
float pid_d_gain_yaw = 0.0;               //Gain setting for the pitch D-controller.
int pid_max_yaw = 400;                    //Maximum output of the PID-controller (+/-)

boolean auto_level = true;                //Auto level on (true) or off (false)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Declaring global variables
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
byte eeprom_data[36];
byte highByte, lowByte;
volatile 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];
int temperature;
int acc_axis[4], gyro_axis[4];
float roll_level_adjust, pitch_level_adjust;

long acc_x, acc_y, acc_z, acc_total_vector;
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_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;
float angle_roll_acc, angle_pitch_acc, angle_pitch, angle_roll;
boolean gyro_angles_set;
void setup(){
  //Serial.begin(57600);
  //Copy the EEPROM data for fast access data.
  for(start = 0; start <= 35; start++)eeprom_data[start] = EEPROM.read(start);
  start = 0;                                                          //Set start back to zero.
  gyro_address = eeprom_data[32];                                     //Store the gyro address in the variable.

  Wire.begin();                                                       //Start the I2C as master.

  TWBR = 12;                                                          //Set the I2C clock speed to 400kHz.

  //Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as inputs.
  DDRD |= B11110000;                                                  //Configure digital poort 4, 5, 6 and 7 as output.
  DDRB |= B00110000;                                                  //Configure digital poort 12 and 13 as output.

  //Use the led on the Arduino for startup indication.
  digitalWrite(12,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);

  //The flight controller needs the MPU-6050 with gyro and accelerometer
  //If setup is completed without MPU-6050 stop the flight controller program
  if(eeprom_data[31] == 2 || eeprom_data[31] == 3)delay(10);

  set_gyro_registers();                                               //Set the specific gyro registers.
```

```
for (cal_int = 0; cal_int < 1250 ; cal_int ++){                         //Wait 5 seconds before continuing.
  PORTD |= B11110000;                                                   //Set digital poort 4, 5, 6 and 7 high.
  delayMicroseconds(1000);                                              //Wait 1000us.
  PORTD &= B00001111;                                                   //Set digital poort 4, 5, 6 and 7 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(12, !digitalRead(12));              //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.
  //We don't want the esc's to be beeping annoyingly. So let's give them a 1000us puls while calibrating the gyro.
  PORTD |= B11110000;                                                   //Set digital poort 4, 5, 6 and 7 high.
  delayMicroseconds(1000);                                              //Wait 1000us.
  PORTD &= B00001111;                                                   //Set digital poort 4, 5, 6 and 7 low.
  delay(3);                                                             //Wait 3 milliseconds before the next loop.
}
//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.

PCICR |= (1 << PCIE0);                                                  //Set PCIE0 to enable PCMSK0 scan.
PCMSK0 |= (1 << PCINT0);                                                //Set PCINT0 (digital input 8) to trigger an interrupt on state change.
PCMSK0 |= (1 << PCINT1);                                               //Set PCINT1 (digital input 9)to trigger an interrupt on state change.
PCMSK0 |= (1 << PCINT2);                                               //Set PCINT2 (digital input 10)to trigger an interrupt on state change.
PCMSK0 |= (1 << PCINT3);                                               //Set PCINT3 (digital input 11)to trigger an interrupt on state change.

//Wait until the receiver is active and the throtle is set to the lower position.
while(receiver_input_channel_3 < 990 || receiver_input_channel_3 > 1020 || receiver_input_channel_4 < 1400){
  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
  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.
  PORTD |= B11110000;                                                   //Set digital poort 4, 5, 6 and 7 high.
  delayMicroseconds(1000);                                              //Wait 1000us.
  PORTD &= B00001111;                                                   //Set digital poort 4, 5, 6 and 7 low.
  delay(3);                                                             //Wait 3 milliseconds before the next loop.
  if(start == 125){                                                     //Every 125 loops (500ms).
    digitalWrite(12, !digitalRead(12));                                 //Change the led status.
    start = 0;                                                          //Start again at 0.
  }
}
start = 0;                                                              //Set start back to 0.

//Load the battery voltage to the battery_voltage variable.
//65 is the voltage compensation for the diode.
//12.6V equals ~5V @ Analog 0.
//12.6V equals 1023 analogRead(0).
//1260 / 1023 = 1.2317.
//The variable battery_voltage holds 1050 if the battery voltage is 10.5V.


  loop_timer = micros();                                                //Set the timer for the next loop.

  //When everything is done, turn off the led.
  digitalWrite(12,LOW);                                                 //Turn off the warning led.
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Main program loop
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void loop(){

  //65.5 = 1 deg/sec (check the datasheet of the MPU-6050 for more information).
  gyro_roll_input = (gyro_roll_input * 0.7) + ((gyro_roll / 65.5) * 0.3);   //Gyro pid input is deg/sec.
  gyro_pitch_input = (gyro_pitch_input * 0.7) + ((gyro_pitch / 65.5) * 0.3);//Gyro pid input is deg/sec.
  gyro_yaw_input = (gyro_yaw_input * 0.7) + ((gyro_yaw / 65.5) * 0.3);      //Gyro pid input is deg/sec.


  ////////////////////////////////////////////////////////////////////////////////////////////////////

  ////////////////////////////////////////////////////////////////////////////////////////////////////

  //Gyro angle calculations
  //0.0000611 = 1 / (250Hz / 65.5)
  angle_pitch += gyro_pitch * 0.0000611;                                //Calculate the traveled pitch angle and add this to the angle_pitch variable.
  angle_roll += gyro_roll * 0.0000611;                                  //Calculate the traveled roll angle and add this to the angle_roll variable.

  //0.000001066 = 0.0000611 * (3.142(PI) / 180degr) The Arduino sin function is in radians
  angle_pitch -= angle_roll * sin(gyro_yaw * 0.000001066);              //If the IMU has yawed transfer the roll angle to the pitch angel.
```

```
//Accelerometer angle calculations
acc_total_vector = sqrt((acc_x*acc_x)+(acc_y*acc_y)+(acc_z*acc_z));       //Calculate the total accelerometer vector.

if(abs(acc_y) < acc_total_vector){                                        //Prevent the asin function to produce a NaN
  angle_pitch_acc = asin((float)acc_y/acc_total_vector)* 57.296;          //Calculate the pitch angle.
}
if(abs(acc_x) < acc_total_vector){                                        //Prevent the asin function to produce a NaN
  angle_roll_acc = asin((float)acc_x/acc_total_vector)* -57.296;          //Calculate the roll angle.
}

//Place the MPU-6050 spirit level and note the values in the following two lines for calibration.
angle_pitch_acc -= 0.0;                                                   //Accelerometer calibration value for pitch.
angle_roll_acc -= 0.0;                                                    //Accelerometer calibration value for roll.

angle_pitch = angle_pitch * 0.9996 + angle_pitch_acc * 0.0004;            //Correct the drift of the gyro pitch angle with the accelerometer pitch angle.
angle_roll = angle_roll * 0.9996 + angle_roll_acc * 0.0004;               //Correct the drift of the gyro roll angle with the accelerometer roll angle.

pitch_level_adjust = angle_pitch * 15;                                    //Calculate the pitch angle correction
roll_level_adjust = angle_roll * 15;                                      //Calculate the roll angle correction

if(!auto_level){                                                          //If the quadcopter is not in auto-level mode
  pitch_level_adjust = 0;                                                 //Set the pitch angle correction to zero.
  roll_level_adjust = 0;                                                  //Set the roll angle correcion to zero.
}


//For starting the motors: throttle low and yaw left (step 1).
```

```
//For starting the motors: throttle low and yaw left (step 1).
if(receiver_input_channel_3 < 1050 && receiver_input_channel_4 < 1050)start = 1;
//When yaw stick is back in the center position start the motors (step 2).
if(start == 1 && receiver_input_channel_3 < 1050 && receiver_input_channel_4 > 1450){
  start = 2;

  angle_pitch = angle_pitch_acc;                                          //Set the gyro pitch angle equal to the accelerometer pitch angle when the quadcopter is started.
  angle_roll = angle_roll_acc;                                            //Set the gyro roll angle equal to the accelerometer roll angle when the quadcopter is started.
  gyro_angles_set = true;                                                 //Set the IMU started flag.

  //Reset the PID controllers for a bumpless start.
  pid_i_mem_roll = 0;
  pid_last_roll_d_error = 0;
  pid_i_mem_pitch = 0;
  pid_last_pitch_d_error = 0;
  pid_i_mem_yaw = 0;
  pid_last_yaw_d_error = 0;
}
//Stopping the motors: throttle low and yaw right.
if(start == 2 && receiver_input_channel_3 < 1050 && receiver_input_channel_4 > 1950)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_1 > 1508)pid_roll_setpoint = receiver_input_channel_1 - 1508;
else if(receiver_input_channel_1 < 1492)pid_roll_setpoint = receiver_input_channel_1 - 1492;
```

```
pid_roll_setpoint -= roll_level_adjust;                                   //Subtract the angle correction from the standardized receiver roll input value.
pid_roll_setpoint /= 3.0;                                                 //Divide the setpoint for the PID roll controller by 3 to get angles in degrees.


//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_2 > 1508)pid_pitch_setpoint = receiver_input_channel_2 - 1508;
else if(receiver_input_channel_2 < 1492)pid_pitch_setpoint = receiver_input_channel_2 - 1492;

pid_pitch_setpoint -= pitch_level_adjust;                                 //Subtract the angle correction from the standardized receiver pitch input value.
pid_pitch_setpoint /= 3.0;                                                //Divide the setpoint for the PID pitch controller by 3 to get angles in degrees.

//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_3 > 1050){ //Do not yaw when turning off the motors.
  if(receiver_input_channel_4 > 1508)pid_yaw_setpoint = (receiver_input_channel_4 - 1508)/3.0;
  else if(receiver_input_channel_4 < 1492)pid_yaw_setpoint = (receiver_input_channel_4 - 1492)/3.0;
}

calculate_pid();                                                          //PID inputs are known. So we can calculate the pid output.

//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 < 1000 && battery_voltage > 600)digitalWrite(12, HIGH);


throttle = receiver_input_channel_3;                                        //We need the throttle signal as a base signal.

if (start == 2){                                                            //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.
  }
  if(micros() - loop_timer > 4050)digitalWrite(12, HIGH);                   //Turn on the LED if the loop time exceeds 4050us.

  //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.

  PORTD |= B11110000;                                                       //Set digital outputs 4,5,6 and 7 high.
  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.

  //There is always 1000us of spare time. So let's do something usefull that is very time consuming.
  //Get the current gyro and receiver data and scale it to degrees per second for the pid calculations.
  gyro_signalen();

  while(PORTD >= 16){                                                       //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)PORTD &= B11101111;                //Set digital output 4 to low if the time is expired.
    if(timer_channel_2 <= esc_loop_timer)PORTD &= B11011111;                //Set digital output 5 to low if the time is expired.
    if(timer_channel_3 <= esc_loop_timer)PORTD &= B10111111;                //Set digital output 6 to low if the time is expired.
    if(timer_channel_4 <= esc_loop_timer)PORTD &= B01111111;                //Set digital output 7 to low if the time is expired.
  }
}
```

```
ISR(PCINT0_vect){
  current_time = micros();
  //Channel 1=========================================
  if(PINB & B00000001){                                        //Is input 8 high?
    if(last_channel_1 == 0){                                   //Input 8 changed from 0 to 1.
      last_channel_1 = 1;                                      //Remember current input state.
      timer_1 = current_time;                                  //Set timer_1 to current_time.
    }
  }
  else if(last_channel_1 == 1){                                //Input 8 is not high and changed from 1 to 0.
    last_channel_1 = 0;                                        //Remember current input state.
    receiver_input[1] = current_time - timer_1;                //Channel 1 is current_time - timer_1.
  }
  //Channel 2=========================================
  if(PINB & B00000010 ){                                       //Is input 9 high?
    if(last_channel_2 == 0){                                   //Input 9 changed from 0 to 1.
      last_channel_2 = 1;                                      //Remember current input state.
      timer_2 = current_time;                                  //Set timer_2 to current_time.
    }
  }
  else if(last_channel_2 == 1){                                //Input 9 is not high and changed from 1 to 0.
    last_channel_2 = 0;                                        //Remember current input state.
    receiver_input[2] = current_time - timer_2;                //Channel 2 is current_time - timer_2.
  }
  //Channel 3=========================================
  if(PINB & B00000100 ){                                       //Is input 10 high?
    if(last_channel_3 == 0){                                   //Input 10 changed from 0 to 1.
      timer_3 = current_time;                                  //Set timer_3 to current_time.
    }
  }
  else if(last_channel_3 == 1){                                //Input 10 is not high and changed from 1 to 0.
    last_channel_3 = 0;                                        //Remember current input state.
    receiver_input[3] = current_time - timer_3;                //Channel 3 is current_time - timer_3.

  }
  //Channel 4=========================================
  if(PINB & B00001000 ){                                       //Is input 11 high?
    if(last_channel_4 == 0){                                   //Input 11 changed from 0 to 1.
      last_channel_4 = 1;                                      //Remember current input state.
      timer_4 = current_time;                                  //Set timer_4 to current_time.
    }
  }
  else if(last_channel_4 == 1){                                //Input 11 is not high and changed from 1 to 0.
    last_channel_4 = 0;                                        //Remember current input state.
    receiver_input[4] = current_time - timer_4;                //Channel 4 is current_time - timer_4.
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Subroutine for reading the gyro
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void gyro_signalen(){
  //Read the MPU-6050
  if(eeprom_data[31] == 1){
    Wire.beginTransmission(gyro_address);                      //Start communication with the gyro.
    Wire.write(0x3B);                                          //Start reading @ register 43h and auto increment with every read.
    Wire.endTransmission();                                    //End the transmission.
    Wire.requestFrom(gyro_address,14);                         //Request 14 bytes from the gyro.

    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.

    while(Wire.available() < 14);                              //Wait until the 14 bytes are received.
    acc_axis[1] = Wire.read()<<8|Wire.read();                  //Add the low and high byte to the acc_x variable.
    acc_axis[2] = Wire.read()<<8|Wire.read();                  //Add the low and high byte to the acc_y variable.
    acc_axis[3] = Wire.read()<<8|Wire.read();                  //Add the low and high byte to the acc_z variable.
    temperature = Wire.read()<<8|Wire.read();                  //Add the low and high byte to the temperature variable.
    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];         //Set gyro_roll to the correct axis that was stored in the EEPROM.
  if(eeprom_data[28] & 0b10000000)gyro_roll *= -1;             //Invert gyro_roll if the MSB of EEPROM bit 28 is set.
```

```
  gyro_pitch = gyro_axis[eeprom_data[29] & 0b00000011];                     //Set gyro_pitch to the correct axis that was stored in the EEPROM.
  if(eeprom_data[29] & 0b10000000)gyro_pitch *= -1;                         //Invert gyro_pitch if the MSB of EEPROM bit 29 is set.
  gyro_yaw = gyro_axis[eeprom_data[30] & 0b00000011];                       //Set gyro_yaw to the correct axis that was stored in the EEPROM.
  if(eeprom_data[30] & 0b10000000)gyro_yaw *= -1;                           //Invert gyro_yaw if the MSB of EEPROM bit 30 is set.

  acc_x = acc_axis[eeprom_data[29] & 0b00000011];                           //Set acc_x to the correct axis that was stored in the EEPROM.
  if(eeprom_data[29] & 0b10000000)acc_x *= -1;                              //Invert acc_x if the MSB of EEPROM bit 29 is set.
  acc_y = acc_axis[eeprom_data[28] & 0b00000011];                           //Set acc_y to the correct axis that was stored in the EEPROM.
  if(eeprom_data[28] & 0b10000000)acc_y *= -1;                              //Invert acc_y if the MSB of EEPROM bit 28 is set.
  acc_z = acc_axis[eeprom_data[30] & 0b00000011];                           //Set acc_z to the correct axis that was stored in the EEPROM.
  if(eeprom_data[30] & 0b10000000)acc_z *= -1;                              //Invert acc_z if the MSB of EEPROM bit 30 is set.
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Subroutine for calculating pid outputs
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
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;
}
//This part converts the actual receiver signals to a standardized 1000 - 1500 - 2000 microsecond value.
//The stored data in the EEPROM is used.
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[channel];                                         //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;
}

void set_gyro_registers(){
  //Setup the MPU-6050
  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(0x1C);                                                       //We want to write to the ACCEL_CONFIG register (1A hex)
    Wire.write(0x10);                                                       //Set the register bits as 00010000 (+/- 8g full scale range)
    Wire.endTransmission();                                                 //End the transmission with the gyro

    //Let's perform a random register check to see if the values are written correct
    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(12,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 CONFIG register (1A hex)
    Wire.write(0x03);                                                   //Set the register bits as 00000011 (Set Digital Low Pass Filter to ~43Hz)
    Wire.endTransmission();                                             //End the transmission with the gyro

  }
}
```

Appendix G: Quad copter building steps

G-1 The importance of diode D1 and resistors R2 / R3

The diode D1 protects the USB port of the computer when the Arduino is connected to the computer. This diode has an important function and cannot be excluded.

The resistors divide the flight battery voltage by 2.5. This way it is possible to measure the battery voltage during flight. The LED will light up when the battery voltage gets to low and the motor rpm automatically increase to compensate the dropping battery voltage during flight.

The 1kΩ and 1.5kΩ resistors need to be installed correctly otherwise the quad copter will not fly perfectly.

G-2 The MPU-6050 gyro/accelerometer

The orientation of the gyro is not important as long as the Z-axis is vertical (perpendicular to the surface) and the edges of the gyro are aligned with the edges of the quad copter. The setup software (**Appendix E**) will detect the gyro's orientation and invert the gyro and accelerometer axis when necessary. Mount the gyro with thin double side tape. Don't use foam tape or other damping material. This will decrease the performance.

G-3 The transmitter and receiver

Almost every 4 channel RC transmitter can be used for this project. The most important feature is the used receiver output pulse. The range should be approximately 1000 till 2000 with a 1500 center position.

Check the manual of the specific transmitter / receiver for details of Fly sky FS-T6 6-CH TX Transmitter.

Connect the roll (aileron), pitch (elevator), yaw (rudder) and throttle output of the receiver to the Arduino Uno ports 8, 9, 10 and 11. The order is not important as the setup software will recognize each separate channel. Check the manual of the transmitter / receiver to see which receiver port is connected to the specific function.

The receiver is powered by the +5V output of the Arduino. The connection can be found on the schematic **Appendix C**.

G-4 The ESC's

On the schematic only the ground and the signal wires of the ESC's are connected. This is correct. The +5V from the ESC is not connected because the Arduino gets its power directly from the flight battery via the diode D1.

In some cases the ground of the ESC doesn't have to be connected. Check with a multimeter if the ground of the battery connection is connected to the ground / - of the esc connection wire. If these are connected the ground of the ESC does not need to be connected to the Arduino because they share the same battery ground.

The signal wire of the ESC's are connected to the digital outputs 4, 5, 6 and 7 of the Arduino as shown in the table below. Also check the direction of rotation.

Arduino	Location	Direction of rotation
D4	right front	counter clockwise
D5	right rear	clockwise
D6	left rear	counter clockwise
D7	left front	clockwise

G-5 Run the setup software

Remove the props, don't connect the flight battery and upload the setup program (**Appendix E**) to the Arduino Uno. Open the serial monitor at 57600baud and complete the setup by executing the requested actions.

After the setup is completed all the settings are stored in the EEPROM of the Arduino.

G-5.1 Receiver and gyro check

To make sure that everything is working correct it's necessary to run some basic checks. Remove the props, disconnect the flight battery and upload the ESC calibration program (**Appendix D**) to the Arduino. Open the serial monitor at 57600baud.

G-5.2 Receiver input check

Send the letter 'r' to start the receiver monitor. Now move the sticks and see if the values on the screen correspond with the movements of the sticks.

All the channels should read 1000us till 2000us with a center position of 1500 (+/-8).

G-5.3 Gyro / accelerometer angle check

After the receiver check is completed send the letter 'a' to start the angle check.

Don't move the quad copter because the gyro needs to calibrate itself. After the calibration the roll and pitch angles are shown. The yaw value is the output of the gyro and will go back to zero if the yaw rotation stops.

Check if the angles correspond with the movement of the quad copter:

- Nose up is positive pitch and nose down is negative pitch.
- Left wing up is positive roll and left wing down is negative roll.
- Nose right is positive yaw and nose left is negative yaw.

G-6 Calibrate the ESC's

Electronic speed controllers or ESC's for short are controlled with a 1000us till 2000us pulse. 1000us means off and 2000us means full throttle. To make sure that all the ESC's react the same way it's important to calibrate the 1000us and 2000us point. Without calibration the motors will perform different and the quad copter does not fly well or might even crash.

Remove the props and upload the ESC calibration program (**Appendix D**) to the Arduino. Disconnect the USB cable and follow the instructions in the manual to calibrate the ESC's.

In most cases this is done with the following steps:

- Place the throttle stick in the upper position (full throttle)
- Connect the flight battery
- After some beeps place the throttle stick in the lowest position

- Disconnect the flight battery
- But again, check the manual of your specific ESC for the correct calibration procedure.

G-6.1 Balance the motors and props

Balancing the propellers is incredibly important! Without well balanced props and motors the gyro and accelerometer will produce noise that makes the motors react badly. There is minimal stability and the quad copter can't level itself.

To get the best performance the props and motors need to be balanced perfectly. Putting the gyro / accelerometer on vibration dampeners does not help and can only make things worse.

G-6.2 How to balance the props

Mount the props on the motors and check if the counter clock wise and clock wise props are in the right position. Upload the ESC calibration program and open the Arduino serial monitor at 57600baud. Send '1' via the serial monitor and wait for the response "Test motor 1 (right front CCW.)".

The numbers that are printed on the screen represents the amount of vibration measured by the accelerometer. This is not a standardized value and should only be used to minimize the amount of vibration of your quad copter.

Hold the quad copter firmly down, place the throttle in the lowest position and connect the flight battery. Now slowly increase the throttle until motor 1 starts to spin. Check the direction of rotation and that the prop produces upward thrust. If the motor rotates in the wrong direction you need to switch two of the three motor wires. Put the throttle in the lowest position to stop the motor.

Now hold the motor frame firmly in your hand and increase the throttle to half throttle. Check the numbers on the screen and also memorize the vibrations that you feel with your hand that is holding the motor frame.

Stop the motor and put a small piece of tape on one of the blades and run the test again. Check if the vibrations reduce. If nor try a piece of tape on the other blade. Keep doing this until the motor and prop run as smooth as possible. This can sometimes be a daunting task but the reward is a very stable flying quad copter.

When done with motor 1, send a '2' via the Arduino IDE and start the process again for motor / prop 2. And after that, send a '3' for motor number 3 and a '4' for motor 4.

By sending a '5' all the motors will run together as a final test.

G-7 Upload the flight controller software

Disconnect the flight battery and upload the flight controller software (**Appendix F**) to the Arduino. Disconnect the USB cable and connect the flight battery.

Hold the quad copter firmly in your hand and start the motors with the following sequence:

Start = throttle down and yaw left

Stop = throttle down and yaw right

Increase the throttle up to the point when it almost starts to become weightless. The quad copter should now try to level itself. If you move the quad it should start to counteract the movement until it is level again. When the roll or pitch stick of the transmitter is moved the quad copter should move in the same direction

Appendix H: PID tuning

Tuning a PID is a process of finding the values of proportional, integral and derived gains from a PID controller to achieve the desired performance and meet design requirements.

Setting the PID controller seems easy, but finding a set of gains that guarantees the best performance of our control system is a complex task. Traditionally, PID controllers are set manually or using rule-based methods. The methods of manual adjustment are iterative and require a lot of time and, if they are used on equipment, they can cause damage. The methods based on the rules also have serious limitations: they do not take into account load certain types of models, such as unstable plots, plots of a higher order or plots that have no delay. You can automatically synchronize the PID controllers to obtain the optimal system design and to meet the requirements of design, even for factory models, according to which the methods traditional rules-based cannot handle well.

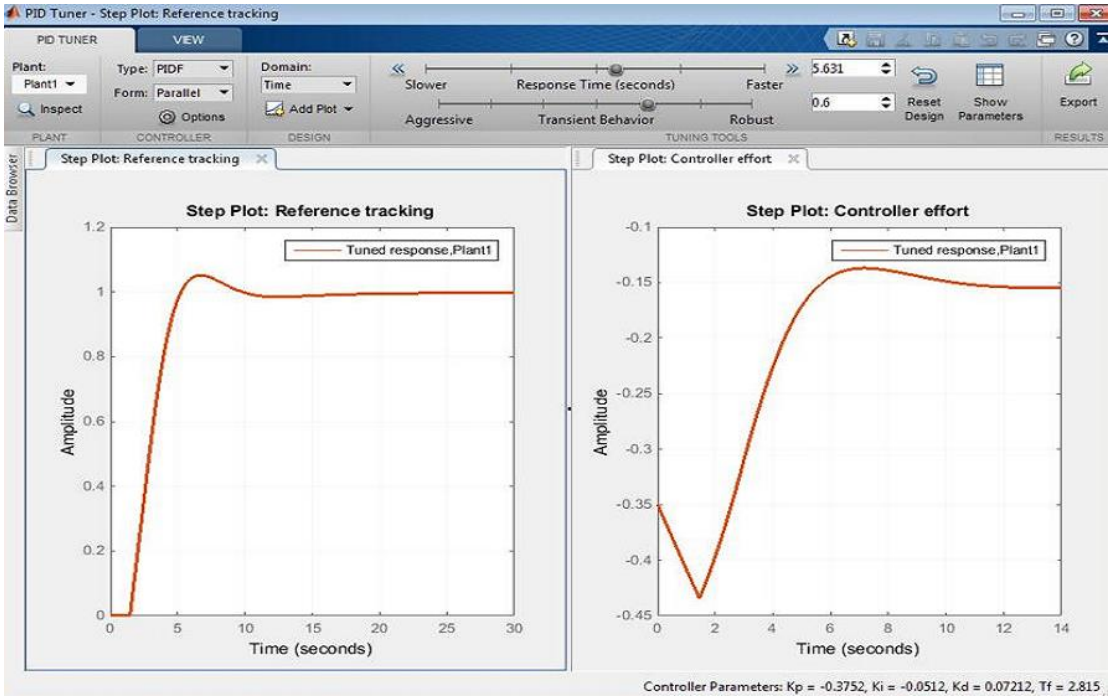


Figure 3- Tuning a PID on SIMULINK /Matlab

Appendix I: Non-Holonomic Effect

The non-holonomic bonds are relations linking the two angles θ and ϕ with Ψ and components of the acceleration vector. These relationships are obtained by manipulating three equations of the dynamic model neglecting the gyroscopic effects in x, y, z:

$$\ddot{x} = \frac{u_1}{m}(C\phi S\theta C\psi + S\psi S\phi)$$

$$\ddot{y} = \frac{u_1}{m}(C\phi S\theta S\psi - C\psi S\phi)$$

$$\ddot{z} = \frac{C\phi C\theta}{m}u_1 - g$$

$$C\phi S\theta = \frac{m}{u_1}(\ddot{x}C\psi + \ddot{y}S\psi)$$

$$C\phi S\theta = \frac{m}{u_1}(-\ddot{x}S\psi + \ddot{y}C\psi)$$

$$C\phi C\theta = \frac{m}{u_1}(\ddot{z} + g)$$

$$tan\theta = \frac{\ddot{x}C\psi + \ddot{y}S\psi}{\ddot{z}+g}$$

$$Sin\phi = \frac{m}{u_1}(\ddot{x}S\psi - \ddot{y}C\psi)$$

$$\ddot{x}^2 + \ddot{y}^2 + (\ddot{z} + g)^2 = \frac{{u_1}^2}{m^2}$$

$$\frac{m}{u_1} = \frac{1}{\sqrt{\ddot{x}^2 + \ddot{y}^2 + (\ddot{z}+g)^2}}$$