Master Thesis

Institut Supérieur de l'Aéronautique et de l'Espace
National ICT Australia

Stabilization of a four rotors UAV

Hugo MERIC

Supervisor : Max OTT
School Supervisor : Emmanuel LOCHIN

Acknowledgement

First of all, I would like to thank my two supervisors Emmanuel Lochin and Max Ott for their support all along my internship. Emmanuel helped me to find my internship and Max accepted to supervise me for this project. Moreover Max always had good ideas to make me move on when I was stucked.

Then I would like to thank all the people who helped me during my six month at NICTA (Guillaume, Olivier, Rodney, Sebastien, Thierry...). Without their support, many points would have been very hard to overcome. They also provided an external opinion, which brought new ideas for the project.

Finally, I wanted to thank all the NICTA organisation for the good atmosphere at work, and especially Rema and Prashanty who welcomed me.

Table of Contents

Chapter 1

Introduction

1.1 NICTA

My end-of-studies internship took place at NICTA in Sydney at the Australian Technology Park laboratory (Figure 1.1). NICTA is an independent company in the business of research, commercialisation and research training established in 2002 by the Australian Government. With five laboratories in four cities and over 700 people, it is the largest organisation in Australia dedicated to Information and Communications Technology (ICT) research.

NICTA has four reseach themes:

- Embedded Systems
- Networked Systems
- Making Sense of Data
- Managing Complexity

The goal of most projects is commercialisation (selling the technology...). If you want to know more about NICTA, refer to [18].

1.2 THE PROJECT

The development of uninhabited aerial vehicles (UAVs) is today increasing. They have many practical applications due to their high maneuverability and their ability to takeoff and land vertically: aerial photography, crowd control, military tactical surveillance... More applications are presented on [17].

The first successful quadrotors flew in the 1920's, but no practical quadrotors have been built until recently. This is due to the difficulty to control four motors simultaneously

Figure 1.1 ATP Building

to stabilize the aircraft. Recent advances in microprocessor capabilities, power storage and Micro ElectroMechanical systems (MEMs) sensors have accelerated the development of that kind of aircraft. Many studies are now working on control algorithms in order to obtain autonomous flights of UAVs. For instance, the Stanford Testbed of Autonomous Rotorcraft for Multi-Agent Control (STARMAC) project has very good results with various algorithms.

My end-of-studies internship took place in that research field. The project's goal was to design an autonomous four rotors vehicle which could be commanded by a mobile phone via WiFi or bluetooth. The NSW firemen have already expressed their interest in that kind of rotorcraft to help them in their work. The project started last year with two students who built the aircraft and wrote the code to control it without stability augmentation systems. Thus the quadrotor could not fly due to stabilization problems. The aim of my internship was to find and implement different control algorithms in order to get a stable flight. The internship was from March to August 2009. Here is the initial planning:

Period	**Objectives**
March	Knowledge of the project
April- mid June	Attitude Control
mid June - mid August	Altitude Control
mid August - end August	Writing documentation and final report

Table 1.1 Initial planning

1.3 PRESENTATION OF THE QUADROTOR

The NICTA quadrotor, Figure 1.2, is a four rotors aircraft which weights around 0.6kg and has an envergure of 80cm. The payload of the aircraft is composed by the battery, a printed circuit board (PCB), sensors and an OpenMoko FreeRunner. Openmoko is a project dedicated to delivering mobile phones with an open source software stack. The FreeRunner runs a Linux kernel and has one WiFi interface, one I^2C interface and two 3-axis accelerometers. The distribution is OM2007.2 which uses the 2.6.24 Linux kernel. The OpenMoko is used to stabilize the aircraft. It runs the control algorithms using the data of the sensors. All the code is in C language.

The I^2C bus links the phone to the PCB. All the commands calculated by the FreeRunner are send via this bus to the PCB. Its goal is to provide enought power to the motors.

Several sensors are necessary for the project. The OpenMoko already has two 3-axis accelerometers. Thus we can obtain the attitude of the aircraft, roll and pitch angles. The yaw angle is given by a compass with tilt compensation. Five ultrasonic sensors are used, one to get the altitude and the others give the distance to the nearest obstacles.

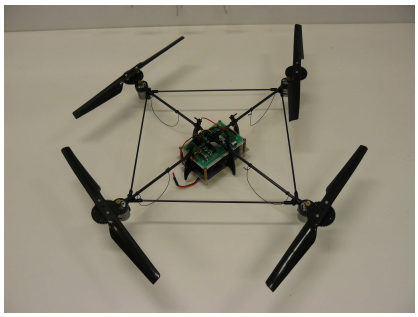


Figure 1.2 NICTA's quadrotor

1.4 FLIGHT PRINCIPLE

Basically it is the shape of the four propellers which makes the quadrotor able to fly. The curved shape accelerates the air flow at the extrados, unlike the intrados where the flow is slowed. The Bernoulli's principle tells us there is a decrease of pressure at the extrados and an increase of pressure at the intrados, and then a vertical force is created. This force is called lift. On each propeller, there is a letter (A or B) to indicate if the propeller must turn clockwise or not. It is to be sure that the lift is directed in the upper vertical.

The four propellers produce their own lift. The global lift compensates the weight and the quadrotor can take off. By changing the speed of a propeller, it modifies the lift generated by itself and it is possible to control the aircraft: move on the left or right, spin on itself and go up and down. On Figure 1.3 to move right, motors 1 and 3 have the same velocity,

but motor 2 turns faster than 4. Thus the lift produced by 2 is bigger than the one of 4 and the quadrotor is moving right.

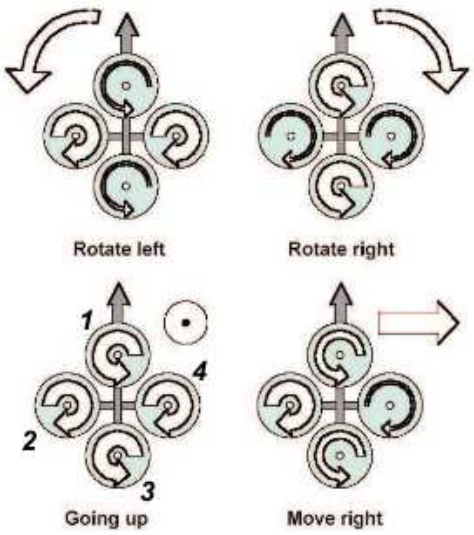


Figure 1.3 Quadrotor motion

1.5 ORGANISATION OF THIS REPORT

This report is organised as follows:

- chapter 2 presents the different parts of the aircraft;
- chapter 3 proposes a dynamic model for the quadrotor;
- chapter 4 studies different control algorithms;
- chapter 5 describes the implementation of the controller on the OpenMoko;
- chapter 6 presents conclusions and future directions for the project.

Chapter 2

System overview

This chapter introduces the different elements of the aircraft: structure, OpenMoko, I^2C bus and the components. It completes the hardware [13] and software [8] reports of the previous year.

2.1 STRUCTURE OF THE AIRCRAFT

The aircraft structure is here to hold the payload. The structure has to be light because the quadrotor has a limited lift. But it also required to be strong to protect the payload in case of accidents. Last year, a first structure has been designed (Figure 2.1).But this structure has few drawbacks. First of all, the phone's battery is not hold, so it is possible that the battery moves and then the phone and the PCB have no more power supply. In that case, the battery must be reset. Thus we see the second drawback. To reset the battery, you need to dismantle the structure and then reassemble it. It is pretty long to do because the structure is not very convenient. Finally the structure must be editable because in the future, more components should be added (video camera...), but it is quite difficult with the current structure.

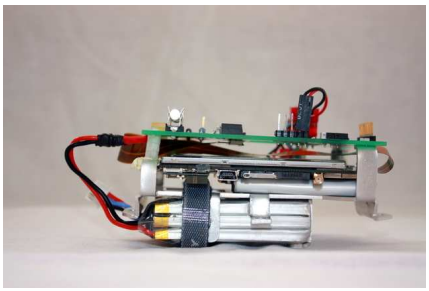


Figure 2.1 Old structure

Thus I decided to make an other structure. To avoid the first drawback, I used the shell of the phone. I removed all the useless parts of the shell to make it lighter. Then I searched for what kind of structure was used in the other quadrotor projects. It seems that a floor-structure is one of the best solution (structure used in the STARMAC project for example). It is very easy to dismantle and reassemble, and there is no problem for adding new components. I designed the pieces using Catia [16].

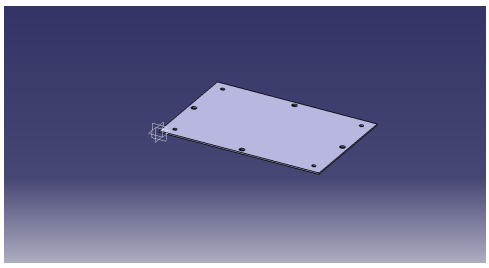


Figure 2.2 Design of a part using Catia

The structure has been made at the workshop of the University of New South Wales. It is in aluminium. I already have experience with aluminium, I know it is light, strong enought and easy to drill. In fact, last year I designed the structure for a model car and I used aluminium.

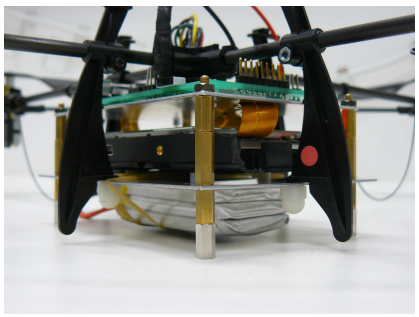


Figure 2.3 New structure

The last point was to make the metal piece which carries the printed board nonconductive, some tape has been used for that. Although the new structure is more convenient, several improvements could be done,

- make a hole to see the screen of the OpenMoko;
- design an attachment for the big battery;
- improve the anti-reverse system as in [3].

2.2 OPENMOKO FREERUNNER

The OpenMoko has two main functions. First of all, it is used because of the two 3-axis accelerometers in it. The data of one accelerometer help us to determine the attitude (pitch and roll angle) of the aircraft. The orientation and the data structure of the accelerometers can be found on [11]. Unfortunately, the two accelerometers can't work together (driver problem?) therefore we use the device */dev/input/event3* for the project which is more convenient due to its orientation.

The second function is to run the control algorithms. It's very easy to program the OpenMoko. It just requires to install the good compiler and then you can develop applications for the phone. See the Appendix C for more details.

The FreeRunner has a WiFi interface. It is possible to connect to the OpenMoko using this interface and then control the aircraft. At the end of the report, you will find in the appendix several tutorials about the OpenMoko, Appendix A B C D. Moreover the OpenMoko project is very well documented on the Internet.

Figure 2.4 OpenMoko FreeRunner

2.3 THE PRINTED CIRCUIT BOARD

The PCB is here to link the mobile phone to the different components. It is also used to provide power to the four motors. All the documentation is in the hardware report of Benedikt Sieghartsleitner [13].

2.4 I2C BUS AND COMPONENTS

2.4.1 I2C bus

The I^2C bus is a multi-master serial computer bus invented by Philips that is used to connect components to a motherboard, embedded system or a cellphone. The I^2C protocol is very simple. For this project, the I^2C bus was chosen to communicate with the different components.

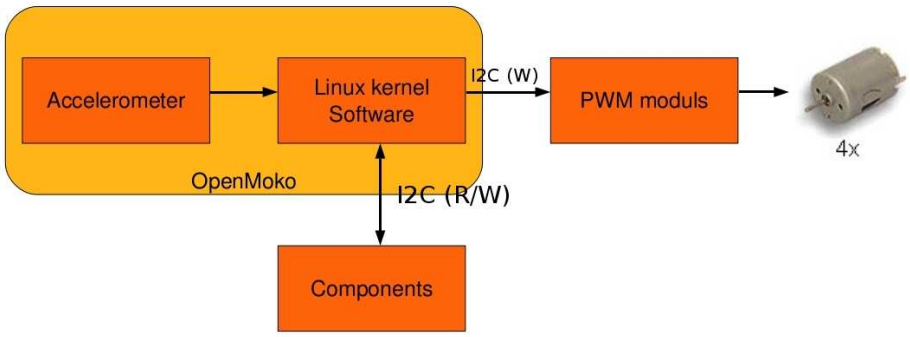


Figure 2.5 i2c communication

Several tools are available on the Internet to test the I^2C bus. During the project, I used the lm-sensors tools from [9]. For instance, you have *i2cdetect* which probes the I^2C bus and return the address of the probed chips. This function is very useful when you have a new component: see if the component is detected and get its address (or check if it is the same address as in the datasheet).
The site also provides the i2c-dev.h file which is used in the project. This file contains low level functions to communicate with I^2C devices.
To use their tools, you just need to modify the makefile, instead of compiling with gcc, you use openmoko-gcc (see Appendix C). All the documentation is also available on their website [9].

NICTA also has the Philips I^2C evaluation board. This board allows the user to easily test I^2C devices. Using the expert mode, you can send all the possible signals and get data if needed. I used the evaluation board to test the compass with tilt compensation. I expose few problems with the evaluation board in section 2.4.3.

I will now present the different devices using the I^2C bus and quickly explain how they work. All the code is available on the NICTA wiki page.

2.4.2 PWM moduls and Ultrasonic sensors

The PWM moduls are used to vary the duty cycle of PWM-signals. The speeds of the propellers are controlled by these signals. The ultrasonic sensors detect the distance to the closest object, it is used to get the altitude of the aircraft and also to avoid collision. The

technical details are in the datasheet of the chips and in the report of Matthias Kazengruber [8].

It is already mentionned in Matthias' report but the minimum distance measured by the ultrasonic sensors is about 20cm. It is a problem for the altitude control. In fact, the controller has no idea of the altitude of the aircraft at the take off and landing, which are the critical part of the flight. I think these sensors should be changed in the future.

2.4.3 Compass with tilt compensation

During the first flight tests, I noticed that the aircraft tend to spin around its vertical axis. To compensate this phenomenon, a controller has been established. The input of the controller is the yaw angle which is given by the compass. At the beginning it was impossible to get the yaw angle. I thought of using the second accelerometer to get it, but as I have already mentioned, the two accelerometers can't work together. Last year, the two students decided to use a simple compass. The results were not very good because of two effects: the motors induce too much noise and, when the compass is not horizontal, its output is far away from the reality.

To get the yaw angle, most quadrotors projects use gyroscopes. So I looked for gyroscopes with I^2C interface. Unfortunately, the only I found where provided without carriage board, it means I have to design a small printed board and make it. It takes a lot of time, so I searched an other solution. I finally found a compass with tilt compensation, I^2C interface and carriage board called HMC6343 [20]. It is possible to get the angle to the magnetic north and also the pitch/roll angles (the chip has a 3 axis accelerometer to get the tilt). The protocol used here is not the same as for the other chips. First, we can get the address of the chip using i2c-detect or in the datasheet. It is written that the slave address is 0x32 for write operation and 0x33 for read operation.

$$0x33 = \mathbf{0011\ 001}1$$
$$0x32 = \mathbf{0011\ 001}0$$

By taking the seven most significant bits and adding a zero to complete in one octet, we obtain the address of the compass 0**001 1001**=0x19. To get the yaw angle, the first step is to send the command 0x50 using the *i2c_smbus_write_byte* function (see i2c-dev.h) and then to read six bytes in a row. To do that, you need an i2c_rdwr_ioctl_data structure and then complete the fields with the good values. The I^2C bus can also crash if one field is bad, in that case the mobile has to be restarted.

The frequence to read the yaw angle is only 5Hz but it is not a problem because the dynamics is not so fast as the pitch/roll angles.

About the I^2C evaluation board, I used it at the beginning to see if the compass was working. I noticed two problems. First of all, the board is supposed to provided 3.3 or 5V depending of the jumper position. The compass only need a 3.3V supply, but the board can't supply it. There is a drop of the voltage. I used an other source to provide the good voltage. The second problem is when I was using the expert mode. There is an address field for the address of the chip. But when I put 0x19, I got an error 'address not

acknowledged'. In fact, in the field address, you have to write the read or write address (here 0x32 or 0x33) and then it is working. During the experiment, it is also possible to visualize the I^2C protocol with an oscilloscope (SDA and SCL signals) as on Figure 2.6.

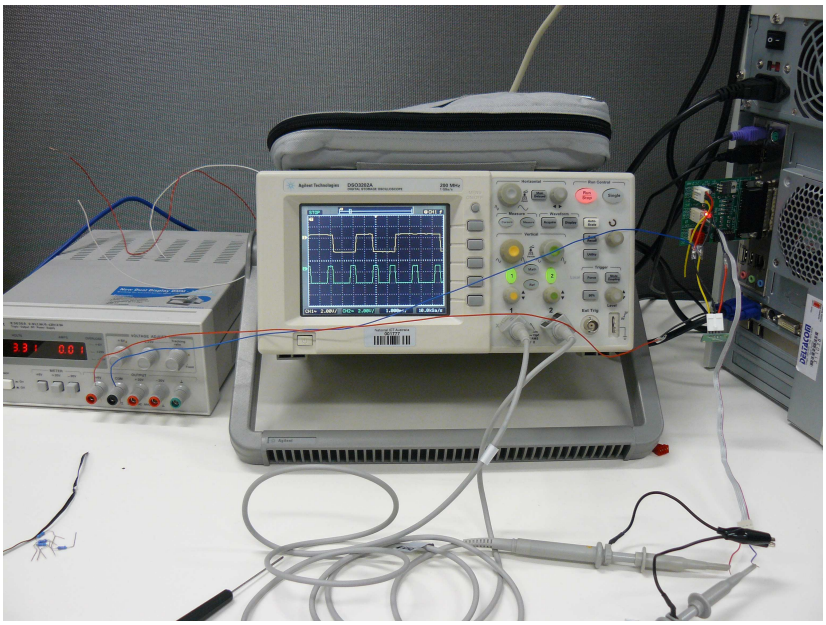


Figure 2.6 Test of the compass with the evaluation board

2.5 CONCLUSION

All the parts of the quadrotor have been well presented and described. More details can be found in [8] or [13]. Several modifications have been done to the aircraft: structure, components... It is still possible to change or improve few points. It depends of the application, the cost...

The next chapter proposes a mathematical model for the aircraft's dynamic. This model is the basis for the design of controllers studied in the chapter 4.

Chapter 3

Modelling the quadrotor

Modelling the quadrotor is the first step before trying to stabilize it. Once the dynamic equations obtained, it will be possible to run some simulations using Matlab to test the control algorithms, before running the flight tests. During the modelling, the most difficult part is to be close of the real system. In fact, all the physical effects can't be modelled but the equations have to be pertinent in order to make the Matlab simulations useful.

3.1 MODEL ASSUMPTIONS AND NOTATIONS

Modelling the quadrotor is pretty hard and it requires few assumptions to make it easier. Thus we will assume,

- the structure is rigid.
- the structure is symmetrical.
- the lift and the drag are proportionnal to the square of the propeller speed.

Symbol	Definition
m	mass of the aircraft
b	thrust factor
d	drag factor
$I_{x,y,z}$	body inertia
l	radius of the aircraft
Ω_i	rotor speed
I_{rotor}	propeller inertia

Table 3.1 Notations

3.2 EULER ANGLES

The Euler angles are used to describe the orientation of a rigid body in a 3-dimensional Euclidean space. The orientation is given by the composition of three rotations from a reference frame.

For this case, I use the following order for the rotations (the order is important because all the calculations depend of the convention used),

$$XYZ \xrightarrow{\psi} \alpha\beta Z \xrightarrow{\theta} x\beta\gamma \xrightarrow{\phi} xyz$$

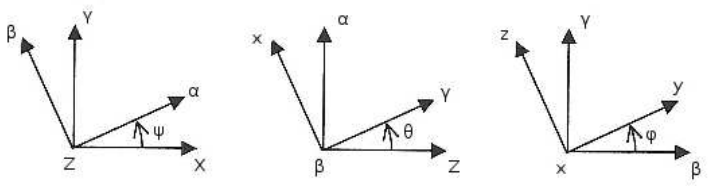


Figure 3.1 Euler angles

We have thus defined three angles,

- the roll angle ϕ which represents a rotation around $\mathbf{x}$ ($-\pi \leq \phi \leq \pi$);
- the pitch angle θ which represents a rotation around $\mathbf{y}$ ($-\pi/2 \leq \theta \leq \pi/2$);
- the yaw angle ψ which represents a rotation around $\mathbf{z}$ ($-\pi \leq \psi \leq \pi$).

Each angle has its own associated rotation matrix. These matrix are given by,

$$R_\phi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \cos(\phi) & \sin(\phi) \end{pmatrix} \qquad R_\theta = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \qquad R_\psi = \begin{pmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Now we can define the global rotation matrix for the quadrotor,

$$R = R_\psi R_\theta R_\phi = \begin{pmatrix} \cos(\psi)\cos(\theta) & \cos(\psi)\sin(\theta)\sin(\phi) - \sin(\psi)\cos(\theta) & \cos(\psi)\sin(\theta)\cos(\phi) + \sin(\psi)\sin(\phi) \\ \sin(\psi)\cos(\theta) & \sin(\psi)\sin(\theta)\sin(\phi) + \cos(\psi)\cos(\theta) & \sin(\psi)\sin(\theta)\cos(\phi) - \sin(\phi)\cos(\psi) \\ -sin(\theta) & \cos(\phi)\sin(\phi) & \cos(\theta)\cos(\phi) \end{pmatrix}$$

However this matrix can be obtained by calculating the coordinates of the aircraft frame in the rigid frame,

$$\begin{cases} \vec{\alpha} = \cos\psi\vec{X} - \sin\psi\vec{Y} \\ \vec{\beta} = \cos\psi\vec{Y} - \sin\psi\vec{X} \\ \vec{\gamma} = \cos\psi\cos\theta\vec{X} + \cos\theta\sin\psi\vec{Y} - \sin\theta\vec{Z} \\ \vec{x} = \cos\theta\cos\psi\vec{X} + \cos\theta\sin\psi\vec{Y} - \sin\theta\vec{Z} \\ \vec{y} = (-\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi)\vec{X} + (\cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi)\vec{Y} + \sin\phi\cos\theta\vec{Z} \\ \vec{z} = (\sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi)\vec{X} + (-\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi)\vec{Y} + \cos\phi\cos\theta\vec{Z} \end{cases}$$

Now it is easy to have the expression of the rotation vector in the aircraft frame,

$$\begin{aligned}\vec{\omega} &= \dot{\psi}\vec{Z} + \dot{\theta}\vec{\beta} + \dot{\phi}\vec{x} \\ &= (\dot{\phi}\cos\theta\cos\psi - \dot{\theta}\sin\psi)\vec{X} + (\dot{\theta}\cos\psi + \dot{\phi}\cos\theta\sin\psi)\vec{Y} + (\dot{\psi} - \dot{\theta}\sin\theta)\vec{Z}\end{aligned} \tag{3.1}$$

3.3 DYNAMIC MODEL

Aircraft configuration

To study the quadrotor dynamic, we need to define two frames and the aircraft configuration. I use that configuration for the project,

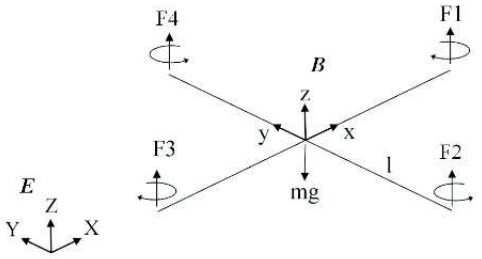


Figure 3.2 Aircraft configuration

Thus the roll angle (about x-axis) is controlled using motors 2 and 4, the pitch angle (about the y-axis) with motors 1 and 3. The yaw required the four motors.

Position equations

The position equations come with the Newton's second law of motion,

$$m\vec{a} = \vec{P} + \vec{T} \tag{3.2}$$

where m is the mass of the aircraft, $\vec{P}$ its weight and $\vec{T}$ represents the lift. Using the lift hypothesis, we can write $\vec{T} = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2)\vec{z} = T\vec{z}$. Thus (3.2) becomes in the rigid frame,

$$\boxed{m\begin{pmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{pmatrix}_{(X,Y,Z)} = mg\begin{pmatrix}0\\0\\-1\end{pmatrix}_{(X,Y,Z)} + T\begin{pmatrix}\cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi\\ \sin\psi\sin\theta\cos\phi - \sin\phi\cos\psi\\ \cos\theta\cos\phi\end{pmatrix}_{(X,Y,Z)}} \tag{3.3}$$

Angular equations

For these equations, we will use the Euler-Lagrange formalism. The Euler-Lagrange equations of the rotorcraft are given by,

$$\Gamma_i = \frac{d}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} = F(q, \dot{q}) \tag{3.4}$$

where $L = T - V$ represents the Lagrangian, T the kinetic energy, V the potential energy, q the generalized coordinates vector and F the generalized forces. Here the generalized coordinates are angular parameters, so the generalized forces are homogenous to torques.

The first step is to calculate the kinetic energy. The aircraft is supposed to be symmetrical so its inertia matrix is diagonal. Moreover, we already have the expression of the rotation vector in the aircraft frame (3.1). Thus, we obtain,

$$T = \frac{1}{2}I_x(\dot{\phi} - \dot{\psi}\sin\theta)^2 + \frac{1}{2}I_y(\dot{\theta}\cos\phi + \dot{\psi}\sin\phi\cos\theta)^2 + \frac{1}{2}I_z(\dot{\theta} - \dot{\psi}\cos\phi\cos\theta)^2$$

The potential energy is given by,

$$V = mgz$$

Now we can compute the three Lagrangians $\Gamma_\psi, \Gamma_\theta, \Gamma_\phi$. I will develop how to obtain Γ_ϕ, it is similar for the two others lagrangians. Thus,

$$\frac{\partial L}{\partial \dot{\phi}} = I_x(\dot{\phi} - \dot{\psi}\sin\theta)$$

Then,

$$\frac{d}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{\phi}}\right) = I_x(\ddot{\phi} - \ddot{\psi}\sin\theta - \ddot{\psi}\dot{\theta}\cos\theta)$$

We assume that the angles are small so we have $\cos\alpha \approx 1$ and $\sin\alpha \approx 0$, and we only keep the first order terms. So it is possible to simplify the expression,

$$\frac{d}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{\phi}}\right) = I_x\ddot{\phi}$$

The second term of the lagrangian is,

$$\frac{\partial L}{\partial \phi} = I_y(\dot{\theta}\cos\phi + \dot{\psi}\sin\phi\cos\theta)(-\dot{\theta}\sin\phi + \dot{\psi}\cos\phi\cos\theta) + I_z(\dot{\theta}\sin\phi - \dot{\psi}\cos\phi\cos\theta)(\dot{\theta}\cos\phi + \dot{\psi}\sin\phi\cos\theta)$$

And after simplifying,

$$\frac{\partial L}{\partial \phi} = (I_y - I_z)\dot{\theta}\dot{\psi}$$

Finally, we obtain,

$$\Gamma_\phi = I_x\ddot{\phi} - (I_y - I_z)\dot{\psi}\dot{\theta}$$

The two others lagrangians are computed the same way as this one. The results are,

$$\begin{cases} \Gamma_\phi = I_x\ddot{\phi} - (I_y - I_z)\dot{\psi}\dot{\theta} \\ \Gamma_\theta = I_y\ddot{\theta} - (I_z - I_x)\dot{\phi}\dot{\psi} \\ \Gamma_\psi = I_z\ddot{\psi} - (I_x - I_y)\dot{\phi}\dot{\theta} \end{cases}$$

The final step is to calculate the nonconservatives torques acting on the rotorcraft. Nonconservatives torques result firstly from the lift,

$$\tau_x = bl(\Omega_4^2 - \Omega_2^2) \qquad \tau_y = bl(\Omega_3^2 - \Omega_1^2)$$

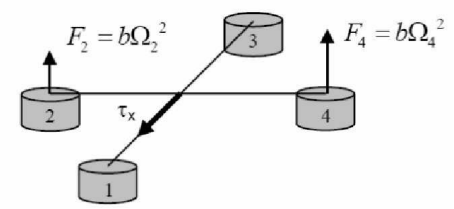


Figure 3.3 Lift torque due to motors 2 and 4

Then, each rotor blades drag which generates an other torque,

$$\tau_z = d(\Omega_1^2 + \Omega_3^2 - \Omega_2^2 - \Omega_4^2)$$

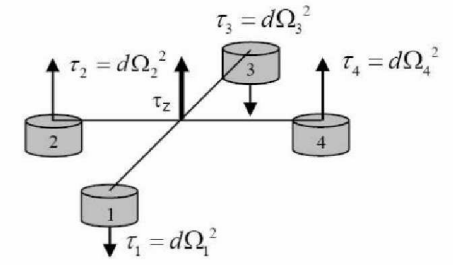


Figure 3.4 Drag torque

Finally, the last torque is due to the gyroscopic effects,

$$\tau_x = I_{rotor}\omega_y(\Omega_1 + \Omega_3 - \Omega_2 - \Omega_4) \qquad \tau_y = I_{rotor}\omega_x(-\Omega_1 - \Omega_3 + \Omega_2 + \Omega_4)$$

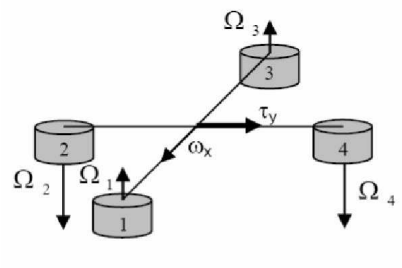


Figure 3.5 Gyroscopic torque (motors and rotation around x-axis)

By equalizing the lagrangians to the torques (3.4), we obtain the angular equations,

$$\boxed{\begin{array}{rcl} \ddot{\phi} & = & \frac{I_{rotor}}{I_x}\dot{\theta}(\Omega_3+\Omega_1-\Omega_2-\Omega_4)+\frac{I_y-I_z}{I_x}\dot{\theta}\dot{\psi}+\frac{bl}{I_x}(\Omega_4^2-\Omega_2^2) \\ \ddot{\theta} & = & \frac{I_{rotor}}{I_y}\dot{\phi}(-\Omega_3-\Omega_1+\Omega_2+\Omega_4)+\frac{I_z-I_x}{I_y}\dot{\phi}\dot{\psi}+\frac{bl}{I_y}(\Omega_3^2-\Omega_1^2) \\ \ddot{\psi} & = & \frac{I_x-I_y}{I_z}\dot{\theta}\dot{\phi}+\frac{d}{I_z}(\Omega_1^2-\Omega_2^2+\Omega_3^2-\Omega_4^2) \end{array}} \tag{3.5}$$

3.4 CALCULATION OF COEFFICIENTS

Now we have the full dynamic model of the aircraft. To run simulations, the last point is to determine all the coefficients who appear in the model. These coefficients have been introduced previously in the Section 3.1.

Body inertia

In this part, we will see how to obtain the $I_{x,y,z}$ coefficients. The structure is supposed to be symmetrical so the inertia matrix is diagonal. It is too complicated to use the definition of $I_{x,y,z}$ so the aircraft needs to be modelled to simplify the calculation. The four motors are modelled by cylinders of radius r and length L, the payload (OpenMoko, battery, printed board) is modelled by a rectangular parallelepiped with dimensions a,w and h.

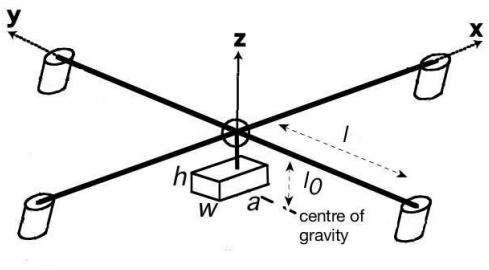


Figure 3.6 Inertia model

The different inertia moments will be calculated at the carbon fiber cross intersection point. The Huygens formula is also used. For the motor 1, we have,

$$\begin{array}{rcl} I_{x,C} & = & \frac{mr^2}{4}+\frac{mL^2}{12}+m(\frac{L}{2})^2 \\ & = & \frac{mr^2}{4}+\frac{mL^2}{3} \\ I_{y,C} & = & \frac{mr^2}{4}+\frac{mL^2}{12}+m(\frac{L}{2})^2+ml^2 \\ & = & \frac{mr^2}{4}+\frac{mL^2}{3}+ml^2 \\ I_{z,C} & = & \frac{mr^2}{2}+ml^2 \end{array}$$

For $I_{x,C}$ we use the Huygens theorem once to translate the moment from the center of gravity of the motor to the point of intersection of the motor vertical axis and the aircraft x-axis. For $I_{y,C}$ the Huygens theorem is used twice. The same calculation is done for the other motors.

Now we will study the payload inertia. Using the results for a parallelepiped,

$$\begin{aligned} I_{x,C} &= \frac{m(w^2+h^2)}{12} + ml^2 \\ I_{y,C} &= \frac{m(a^2+h^2)}{12} + ml^2 \\ I_{z,C} &= \frac{m(w^2+a^2)}{12} \end{aligned}$$

Finally the inertia about each axis is calculated by summation. For example, the inertia about the x-axis is,

$$I_x = I_{x,C,motor1} + I_{x,C,motor2} + I_{x,C,motor3} + I_{x,C,motor4} + I_{x,C,payload}$$

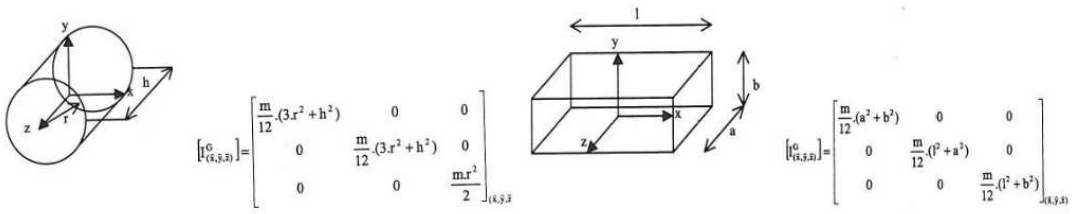


Figure 3.7 Basic shapes inertia

Propeller inertia

Each propellers are modelled by a parallelepiped. The formula above is used to calculate the inertia.

Maximum propellers velocity

This factor is used later in the matlab model and it is also good to have an idea of its value. To get the maximum propellers velocity, we need the maximum speed of the motors and the gear ratio. In the documentation of the motors, we find 7370r/min at maximum efficiency. By counting the teeth of the gears, we have $n_{rotor} = 56teeth$ and $n_{motor} = 10teeth$, and using the following formula we have the propellers velocity,

$$\frac{\omega_{motor}}{\omega_{rotor}} = \frac{n_{rotor}}{n_{motor}}$$

This result is theoretic and it is probably not the real maximum velocity (frictions...). A better thing would be to calculate the speed experimentally, but it requires specific tools.

Drag factor

To determine the drag factor, an easy experiment is carried out. The aircraft is hold in a way that it can turn around the z-axis. Thus the same command is send to two opposite motors. The quadrotor begin to turn around the z-axis.

In this experimentation, the pitch and roll angles are null, thus (3.5) becomes,

$$\ddot{\psi} = \frac{d(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2)}{I_z}$$

By integrating twice, we obtain the yaw law (initial conditions are equal to zero),

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

The drag factor is obtained by timing the time to make a turn on itself,

$$d = \frac{2\pi I_z}{\Omega^2 t_{2\pi}^2}$$

where Ω is the speed of the two rotors and $t_{2\pi}$ the timing.

Thrust factor

The thrust factor is normally given in the datasheet of the propeller. Unfortunatelly, there is no datasheet for our blades. The only information I can get is that the Draganflyer V Ti provides about 6 g/W (for information the X6 rotor blades provide about 14g/W). Otherwise an experiment could be done to determine the thrust factor. We assume that $F = b\Omega^2$. Thus, it is sufficient to measure the maximum weight one motor can carry at maximum rotor velocity. Here again the experiment is pretty easy but requires equipment and the approximation of the maximum propellers velocity can distort the results. I decided to use the coefficient used in one project with a similar aircraft [15].

Results

The values of all coefficients are given in Table 3.2.

3.5 MOTORS MODEL

The transfer function of a motor is a second order one,

$$H(p) = \frac{\Omega(p)}{U(p)} = \frac{K}{K^2 + (R + Lp)(f + Jp)}$$

where K is the motor gain (in volt.s.rad^{-1}), R the internal resistance of the motor, L the inductance, f the friction and J the rotor inertia. Using the fact that $R \ll L$ and $J \ll f$ (not homogenous but in value) then we can simplify the transfer function to a first order,

$$H(p) = \frac{K}{K^2 + RJp} = \frac{\frac{1}{K}}{1 + \tau p}$$

Symbol	Value	Unit (SI)
m	0.638	kg
b	3.13e-5	$m \cdot kg \cdot rad^{-2}$
d	3.2e-6	$m \cdot kg \cdot rad^{-2}$
I_x	0.2839	$m^2 \cdot kg$
I_y	0.3066	$m^2 \cdot kg$
I_z	0.0439	$m^2 \cdot kg$
l	0.21	m
$\Omega_i max$	140	$rad \cdot s^{-1}$
I_{rotor}	8.3e-5	$m^2 \cdot kg$

Table 3.2 Coefficients of the model

τ represents the time constant of the motor. In that model, the motor gain is given in the datasheet and τ is determined experimentally. As I did not have the tools to carry out the experiment, I take an average value for τ.

The motor model cannot be used like that in the future matlab model. The output of the controller is the desired velocity for each motors, these values are then transformed in integer between 0 and 32. We work with these values and not a voltage. I decided to change the transfert function. There is still a lowpass effect but the motor gain is useless. I kept the denominator of the transfert function for the lowpass effect and modify the gain to obtain the values in the good range. Thus the dynamic of the motor is well conserved.

3.6 TRANSPORT DELAY

The quadrotor model must also take into account the transport delay. The delay make the stabilization harder and sometimes impossible if it is too long. So it is very important not to forget this factor for the future model.

Here there is two sources of delay: software and hardware. The software delay is due to the time of computation of the control algorithms. It depends of the code, the frequency of the CPU... To determine the value of the delay, I run the control algorithm in a while loop and I check the start and end time. I do this few times and I take the upper limit. The while loop computes 10000 times the algorithm and it takes in the worst case around 72 seconds. Thus we can assume (because of the size of the loop) that one loop takes on average 7.2ms.

The other delay is due to the hardware. When a command is given to the printed board, how much time does it take before sending the command to the four motors? To determine that delay, I send one command to one motor and I use an oscilloscope to get the delay. The oscilloscope needs to be configured to use the trigger function to detect upper front. By reading the value on the oscilloscope, we see that the hardware delay is less than 2ms, which is very small. Now we can sum the two results to obtain the total delay. All the code I used for these experiments can be found in my source code (projects Delay_Hardware

and Delay_Software). The total delay will be used in the matlab model of the aircraft to determine the differents coefficients for the stabilization.

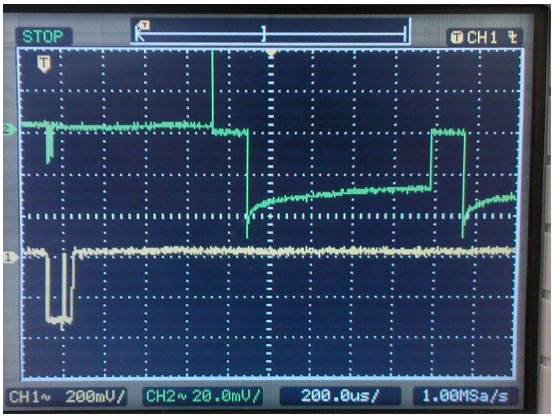


Figure 3.8 Bottom: SDA signal, Top: PWM signal

3.7 CONCLUSION

We finally have a dynamic model of the aircraft. This model is not supposed to be unique. Some phenomenons have not been included like the aerodynamic effects (see [7]) but in our case (slow velocity and no aerodynamic disturbance in an indoor environment), this model should be sufficient.

The next step is to study the attitude stabilization using the angular equations obtained with the Euler-Lagrange formalism. The current mathematical model is going to be implemented with Matlab (simulink) to run simulations before flight tests on the testbed.

Chapter 4

Control algorithms

This chapter presents the control algorithms I used to stabilize the aircraft using the mathematical model previously described. First of all, I will present relative work about the stabilization of quadrotor and then the work I've done during my internship.

4.1 RELATED WORK

A lot of projects are now working on quadrotors due to the number of practical applications. I present hereafter the most relevant ones.

At the ISAE, several groups of students worked on a quadrotor since three years [5] [12] [3]. They built the structure of the aircraft and used a MicroPilot board to run the control algorithms. A big part of the project was dedicated to design a good structure. However, after two years of project, the aircraft could not take off because it was still too heavy. Last year, they finally succeeded to reduce its weight and it was able to take off. Thus they worked on the control algorithm. They started with a Proportional-Integral-Derivative controller (PID controller) for controlling the attitude of the aircraft. The altitude is commanded by the controller using a remote. They modified the attitude controller to a PIDD controller which includes a derivative action on the velocity error. Unfortunately, the aircraft was not able to fly.

The Swiss Federal Institute of Technology Zurich has a department which works on flying robots. Their quadrotor is very similar to NICTA's one (I use their thrust factor). They succeed to make it fly using PID and LQR controllers [14], but the PID obtained the better results. They use two different controllers for the attitude and the altitude.

The Standford Testbed of Autonomous Rotorcraft for Multi-Agent Control (STARMAC, see [1]) is the most advanced project I've seen until now. They build all the rotorcraft and make it fly using different algorithms. Now several aircrafts fly together and they also succeed recently to do backflips. But the most important point is that they also use

two different controllers for the attitude and the altitude control. In fact, the altitude stabilisation is more difficult that it appears because of ground effect, aerodynamic disturbances (see the introduction of [14]). Publications and amazing videos are available on the STARMAC website [19].

Before running algorithms on the aircraft, I decided to do a matlab model for running simulations. I know simulations are not always relevant, but it helps. It is possible to compare different algorithms, to have an idea of the coefficients to use and if the equations of the model are pretty close to the reality, it gives us a good idea of what will happen on the testbed. Due to my initial research, I decided to use two controllers, one for the attitude and one for the altitude.

4.2 ATTITUDE CONTROLLER

The attitude controller consists of controlling the yaw-pitch-roll angles. It is necessary to have done this step before working on the altitude control. I decided to use a Proportionnal-Integral-Derivative controller (PID) which is supposed to work for indoor flights: low velocities and small aerodynamic disturbances [7] [2]. Moreover PID controller are easy to program and do not require a lot of calculations, which is a good point because the CPU is limited.

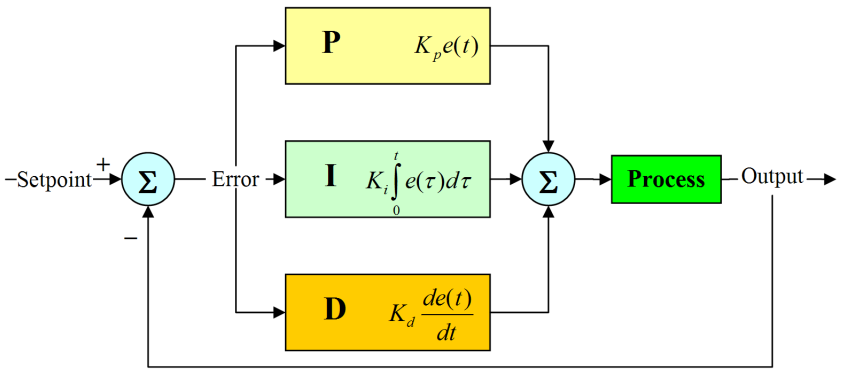


Figure 4.1 PID controller structure

For my matlab model I used the aircraft configuration defined in section 3.3. Now three controllers have to be done, one for each angles, which means nine coefficients to determine. The model use the different coefficients calculated in the chapter 3. Two parameters appear to be very important for the control of the aircraft: the delay and the noise. The delay has been studied in Section 3.6. The noise is due to the accuracy of the sensors, the mechanical and electronic noises. A noise block exits in Simulink to model this phenomenon.

The main idea for designing a controller in a near-hover situation is to neglect the gyroscopic and inertia effect [2]. The model (3.5) is then given by,

$$\ddot{\phi} = \frac{bl}{I_x}(\Omega_4^2 - \Omega_2^2) \qquad \ddot{\theta} = \frac{bl}{I_y}(\Omega_3^2 - \Omega_1^2) \qquad \ddot{\psi} = \frac{d}{I_z}(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2)$$

where,

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

I also introduce $U_4 = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2)$ which represents the full thrust. $U_{1,2,3}$ are used to control the attitude of the aircraft. The output of the controller gives us the value of $U_{1,2,3,4}$ and by solving the system (4.1), the motors speeds are known.

$$\begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{pmatrix} = \begin{pmatrix} 0 & -b & 0 & b \\ -b & 0 & b & 0 \\ d & -d & d & -d \\ b & b & b & b \end{pmatrix} \begin{pmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{pmatrix} \tag{4.1}$$

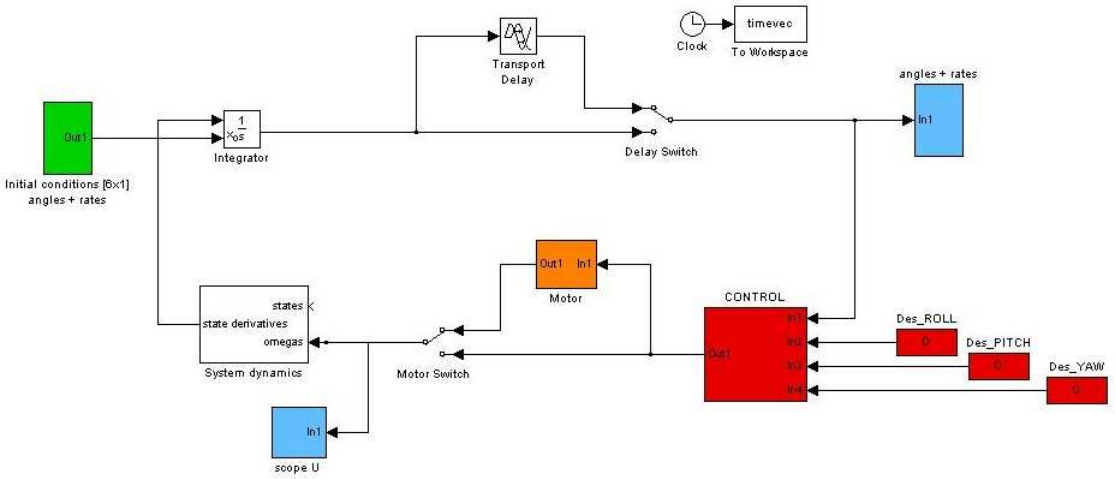


Figure 4.2 Matlab model

I obtained the best results for the three controllers with a proportionnal action on the angle and velocity errors, which corresponds globally to a PD controller. For example, with the yaw angle ψ,

$$\begin{aligned} k_p(0 - v_\psi) &= k_p(\frac{d\psi_d}{dt} - \frac{d\psi}{dt}) \\ &= k_p \frac{e_\psi}{dt} \end{aligned}$$

where ψ_d is the desired angle and the desired velocity is 0 rad/s. Figure 4.3 presents the results obtained with a model including the motor dynamic and noise. We can see that the results for the pitch and roll angles are good. For the yaw angle, we have oscillations

after a few seconds. It is not really a problem in our case because we do not want to drive the aircraft. Moreover I notice that a slight change in the coefficients modify all the dynamic of the aircraft: response time, damping... But now we have an idea of the order of magnitude of the different factors.

I also would like to point out that I used a simplified model for designing the controller, but during the simulations, Matlab runs the full model as described in (3.5).

Controller	Proportional	Derivative
Pitch	0.9	0.8
Roll	0.9	0.8
Yaw	0.6	0.5

Table 4.1 Attitude controller's coefficients values

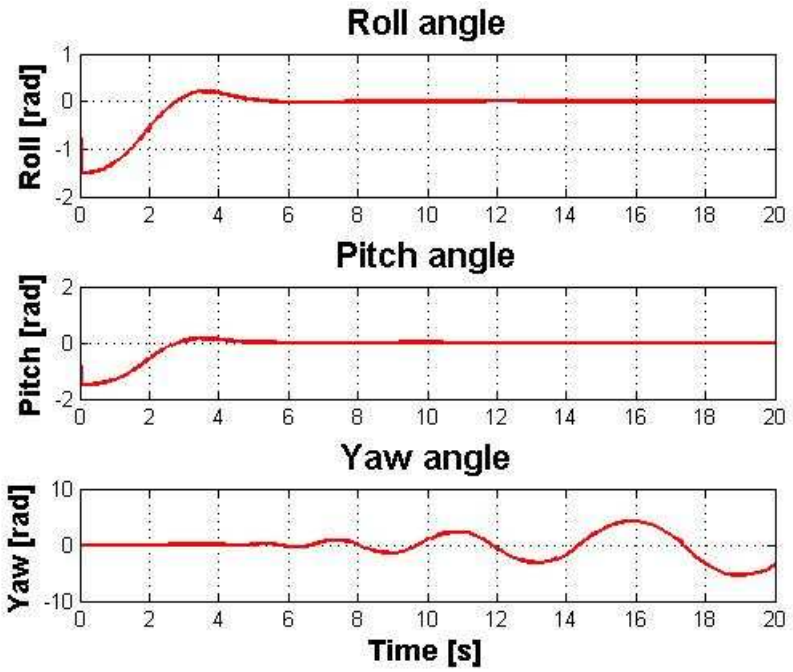


Figure 4.3 Attitude simulation results

4.3 ALTITUDE CONTROLLER

Even if I do not work experimentally on the altitude controller, I propose two different algorithms to test when the attitude control will be achieved.

4.3.1 PD controller

The first controller is presented in [1]. From the position equations (3.2),

$$\ddot{z} = -g + \cos\theta\cos\phi\frac{T}{m}$$

The proposed control is given by,

$$T = \frac{mg}{\cos\theta\cos\phi}(k_p z + k_d \dot{z})$$

This controller only requires one altitude sensor and is very easy to implement. I try this first controller in a matlab model with the attitude controller, and the results are good (without the motor dynamic and noise). As for the attitude controller, the dynamic is very sensitive of the controller factors. The coefficients used are given in Figure 4.2.

Controller	Proportional	Derivative
Pitch	0.8	0.4
Roll	0.8	0.4
Yaw	0.8	0.5
Z	0.6	0.5

Table 4.2 Full controller's coefficients values

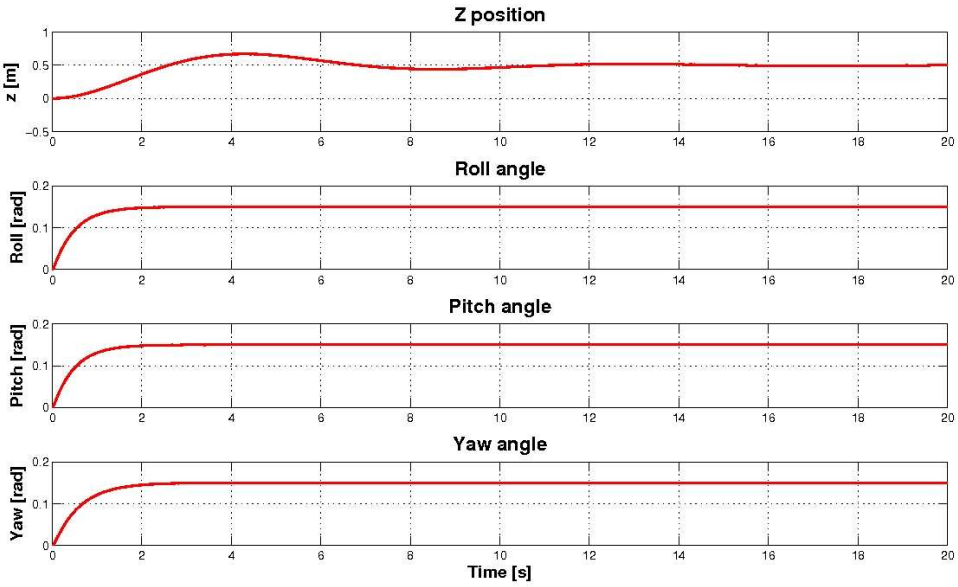


Figure 4.4 Full model simulation results

4.3.2 Reinforcement learning control

For the STARMAC project, several altitude controllers have been designed. I studied the algorithm presented in [14] using reinforcement learning. The goal of the method is to use a reinforcement learning algorithm to search for an optimal control policy.

Reinforcement learning is a sub-area of machine learning concerned with how an agent ought to take actions in an environment so as to maximize some notion of long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the agent ought to take in those states.

The control policy is,

$$\pi = \omega_1 + \omega_2(z - z_d) + \omega_3\dot{z} + \omega_4\ddot{z}$$

The goal of the following algorithm is to determined the vector $\mathbf{w} = {}^t(\omega_1, \omega_2, \omega_3, \omega_4)$ to optimize the policy.

The first step is to approximate the system as a stochastic Markov model using flight data. The method to get the markov model is described in [14]. Unfortunately, there is a problem in the formula used to estimate the subsequent state. Using the same notation as the paper, the estimate is given by,

$$\hat{\mathbf{S}}(t+1) = ({}^tXWX)^{-1t}X^tW\mathbf{x} + \mathbf{v}$$

We can already see that the estimate is computed using X (the input training samples matrix) and not Y (outputs matrix). Now we will check the dimensions of the matrix. I introduce $q = n_s + n_u + 1$, then the dimension of $({}^tXWX)^{-1t}X^tW\mathbf{x}$ is $q \times 1$. But the dimension of $\hat{\mathbf{S}}(t+1)$ is $n_s \times 1$. So the formula is false. The only possibility to get the good dimension is to introduce Y in the estimate formula. I tried to get the good formula but I did not succeed.

However, I got an idea to overcome this problem. First of all, I modified the $\mathbf{S}$ vector by removing the battery level which is anyway quite hard to get here. Then I computed the estimate this way,

$$\hat{\mathbf{S}}(t+1) = \begin{pmatrix} z(t+1) = z(t) + \int_t^{t+1} \dot{z}(t)\,dt \\ \dot{z}(t+1) = \dot{z}(t) + \int_t^{t+1} \ddot{z}(t)\,dt \\ \ddot{z}(t+1) = -\frac{1}{m}T(t) + mg \end{pmatrix}$$

I just use the position equation and integrate twice. Now we can continue as in [14]. A reward function is defined,

$$R(\mathbf{S}, \mathbf{S_{ref}}) = -c_1(r_z - r_{z,ref})^2 - c_2\dot{r_z}^2$$

This function will help us to determine the optimal control policy by rewarding the states close to the setpoint.

The last step is to run the policy itereation. The algorithm uses some random trajectories and look how the defined policy works. To determine if a policy is better than an other, the reward function is used. If it is well implemented (good initial values...), the algorithm should converge to $\mathbf{w_{best}}$. I guess Matlab is the best tool to run the algorithm.

The STARMAC project obtained very good results with that algorithm. Unfortunately, I could not implement it (still working on the attitude controller) and I think it is hard to run a good simulation.

4.3.3 Conclusion about altitude controller

I have presented two altitude controllers. I only simulated one of them so I can't tell which one is the best. Once the attitude will be controlled, I think the PD controller should be tried first. This controller is easier to realize than the second one and the swiss team had good results with it. In my opinion, The STARMAC algorithm should take a lot of time before obtaining positive results (very sensitive to initial values, coefficients...).

4.4 CONCLUSION

Using the model described in Chapter 3 and Matlab, I succeeded to run simulations with good results as shown on Figure 4.4. Four PD controllers have been designed for the full stabilization: attitude and altitude.

The next step is to implement the Attitude controller and test it on the testbed. It will be possible to see if the simulations were good and to adjust the coefficients of the PD algorithms.

Chapter 5

Implementation and flight tests

Once the different simulations done, it is time to implement the control algorithms on the aircraft and test them on the testbed. I will now explain the structure of the program and show the results of the flight tests.

5.1 IMPLEMENTATION

I will just explain the implementation of the attitude controller because I did not code the altitude controller. The program is quite simple. First, I defined a structure (see the file Controller.h) which contains all the data required for the PID controller. Then the program works with four threads,

- the main thread: it is the first thread launched. This thread gets the coefficients of the three controllers, starts the three others threads, computes the outputs of the controllers using the datas in the structure and writes the command values on the PWM moduls;
- the accelerometer thread: its function is to get the data from the accelerometer and calculates the pitch and roll angles;
- the compass thread: this thread gets the data from the compass;
- the input thread: it listens the keyboard events and lets the user commands the aircraft (start and stop the aircraft, control the motors...).

It is possible to check if the program is running well in the terminal. Moreover it is very easy to write into files to get some data (attitude, angles velocitys...) and then check whatever you want. I used gnuplot (install the gnuplot package) on Ubuntu and Matlab when I was running Windows for drawing curves.

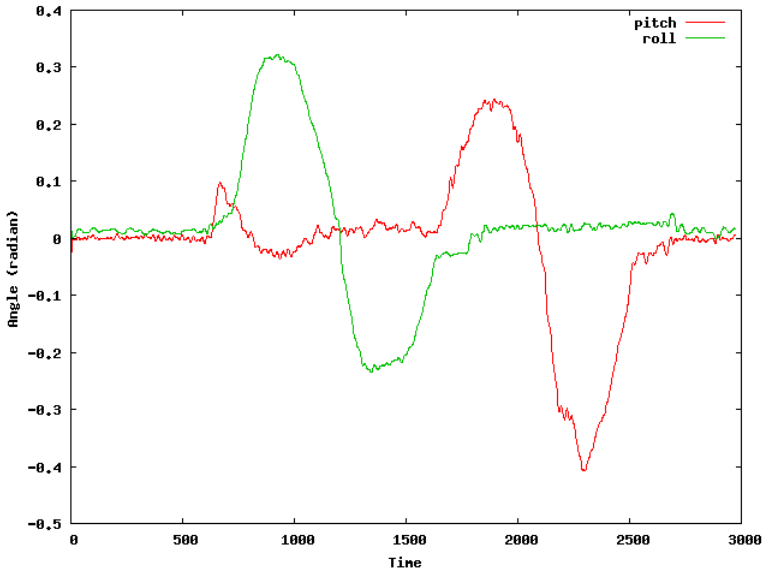


Figure 5.1 Pitch and roll curves obtained with gnuplot

5.1.1 Main function

As already mentioned, the main thread has several functions. To run the program on the OpenMoko, you type in the terminal, *Attitude 0 0 9 8 9 8 4 4* (for example). The two first coefficients are the pitch and roll setpoints. The others coefficients are the PD coefficients, proportionnal and derivative for the three angles. The main thread reads the coefficients and store them.

Then it starts the three other threads. It supposed that the main thread is never killed, otherwise it will kill the other threads and the program stops. It's why the main thread called a function with an infinite loop.

The called function is the altitude stabilization function. It computes the outputs of the controllers using the PD coefficients and the data in the controller structure, and finally writes the commands to the four motors on the PWM moduls via the I^2C bus. The interface with the bus is very easy due to the i2c-dev.h file from the lm-sensors website [9].

5.1.2 Accelerometer control

The accelerometer thread gets the data from the accelerometer and calculates the pitch and roll angles. To understand how the OpenMoko accelerometers work, see [11].

Without treatment, the data are useless due to the static error and the noise. The static error can be observed with an easy experiment. Put the mobile on a table (z-axis vertical) and then get the data from the accelerometers. Normally, the accelerations on the x and y axis are supposed to be null and equal to 1000mg on the z-axis, but it is never the case. Last year, the students calculated the average value on each axis and then compensate it with static coefficients. The fact is that the average values change and then

a static compensation is not enought. Now I use a dynamic compensation, each time the attitude controller is launched, it gets datas from the accelerometer and then computes the compensation vector. It just requires few seconds at the beginning of the program and it is more accurate than a static compensation.

Then the other problem is that the data are very noisy. Two different noises can be observed. We have a white noise as for many experimental values, and also it is possible to see some diracs when the motors are running. Diracs can be easily removed as it is only one point each time. A threshold is sufficient, but a moving average can be added if necessary. For the white noise, it requires a lowpass filter. Using Matlab, it is possible to design that kind of filters. All the available specifications of the filter can be chosen: order, cutoff frequency, attenuation... To design the filter, I use directly experimental data that I import in Matlab. Thus it is possible to see directly the effect of the filter. The fft function is also very useful to see the behaviour of the filter in the frequency domain.

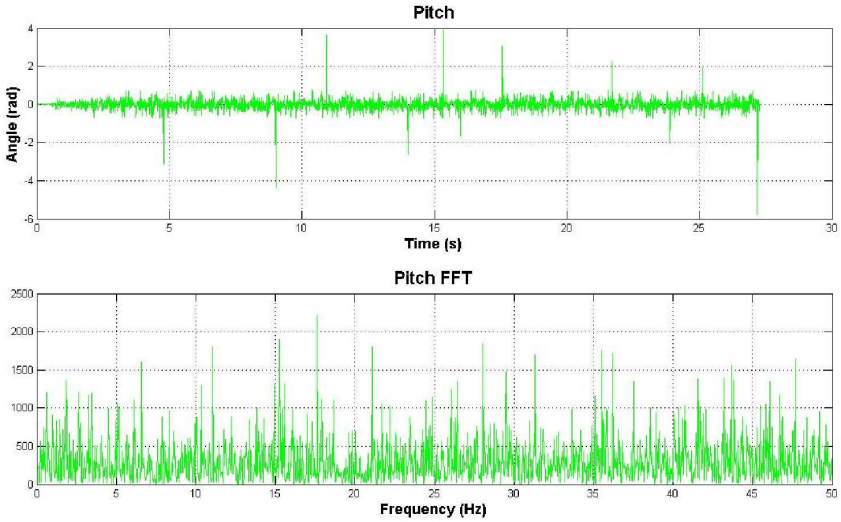


Figure 5.2 Experimental datas imported in Matlab

Now, a Butterworth filter is used to filter the data. It is an Infinite Impulse Response (IIR) filter and its recurrent relation is,

$$s(n) + \sum_{k=1}^{M} a_k s(n-k) = \sum_{k=0}^{N} e(n-k) \tag{5.1}$$

where e is the input signal (non filtered) and s the output signal. The coefficients of (5.1) are computed by Matlab.

This filter gives good results for the angles but the velocitys are still very noisy. I'm now still working on the filtering. The goal is to obtain useful data without slowing too much the dynamic of the aircraft. In fact, the implementation of filters slows the time reaction of the quadrotor. So a comprise has to be done between the time reaction and the acceptable noise.

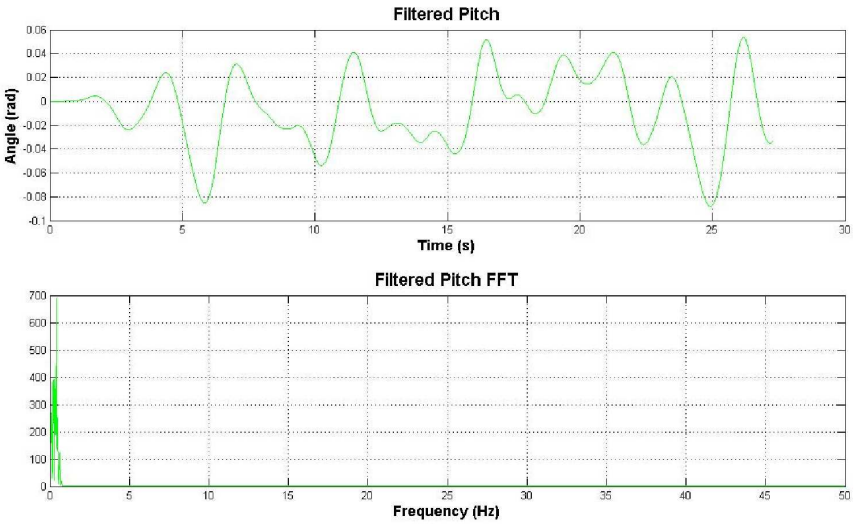


Figure 5.3 Butterworth filter (order 5, cutoff frequency 0.01Hz)

5.1.3 Compass and Input thread

The compass thread handles the communication with the HMC6343 compass. Unlike the pitch and the roll, the user does not mention a desired yaw angle. When the attitude controller program is launched, the first value gets from the compass becomes the reference value, and after the program works with that value.

Finally, the input thread listens the keyboard event. Thus it is possible to control the aircraft. In my program, the input thread controls three things,

- 's' pressed: after the initialization (get the PD coefficients, compute the compensation vector...), it starts the controller loop.
- 'p' or 'm' pressed: it lets the user control the thrust of all the motors.
- 'q' pressed: it shuts down all the motors and exits the program. It is a quick stop in case of the aircraft has a problem.

A lot of other functions can be added in function of what you want to do. Moreover, it is possible to use the terminal with no buffer (no need to press the key and then 'enter'), which is better in case of emergency. Unfortunately, some problems can appear during a flight test: the WiFi connection and the I^2C bus are likely to break down.

5.2 FLIGHT TESTS

5.2.1 Testbed

Before doing the flight tests, I build a testbed for several reasons,

- for the safety of the users and the people around;

- to avoid any damages of the aircraft;
- to test the attitude controller, the aircraft needed to have three degrees of freedom (pitch, roll and yaw).

Moreover, the aircraft has to lie flat when the stabilization program starts (initialization of the compensation vector). I decided to use a hook on the ceiling. On one side I hang the aircraft with rope, and on the other side, I hold the rope to control the altitude of the aircraft. Thus after the initialization is done, I can hang the aircraft in the air and launch the program.

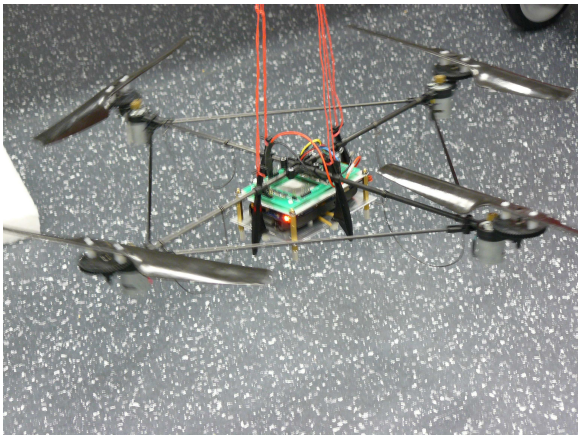


Figure 5.4 Testbed

5.2.2 Flight test results

Once the testbed build and the code implemented and uploaded on the OpenMoko, it is time to test the algorithms. Matlab gives us an idea of the coefficients to use but it is only a model so the coefficients have to be adjusted. By playing with the coefficients, we see how they act on the aircraft: oscillations...

Flight tests show if the code is correct. For example, when there is a non zero pitch angle, do the motors act to compensate in the good way ? It is possible to see if the dynamic of the aircraft is fast enought.

Using a PD controller for the pitch and the roll, the best I have is an oscillating response. An easy experiment shows that there are filtering problems. I put the aircraft between two boxes and I work only with one angle (pitch or roll). The quadrotor should normally stabilize. I can see that sometimes it is close to the setpoint, and suddenly one motor races, which generates oscillations. That problem comes from the data from the OpenMoko. It is the problem I have already exposed earlier and I am still working on it. Otherwise I have also tried a PID controller with a small integral effect. It does not work. The oscillations are amplified and the behaviour of the aircraft is diverging. Moreover the dynamic was very slow, so it makes the stabilization even harder.

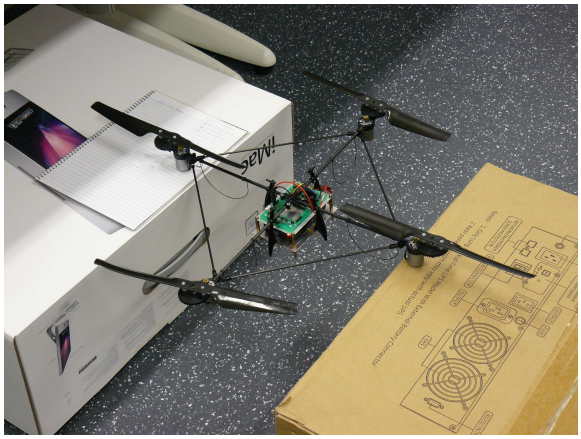


Figure 5.5 One axis stabilization

For the yaw angle, the new compass seems to work well. But the algorithm to stabilize the angle is very slow. Normally the frequency to get data from the compass is 5Hz. It is not high but the stabilization of the yaw is not as critical as for the pitch and roll. But here the response to a yaw disturbance is really too long. I think this is due to the sheduling of all the threads. Thus the response time is increased. Several things can be done to schedule the different tasks, refer to Section 5.3.

The last thing I'd like to point out is about the code. The output of the controller is the desired velocity for each motors. This velocity is expressed in rad/s. This velocity is changed in an integer,

$$32 \times \left\lfloor \frac{desiredvelocity}{maxvelocity} \right\rfloor$$

Strangely it is possible to modify the dynamic by choosing the floor, ceiling or nearest integer function. What I really want to point out here is that the maximum propeller velocity is around 150rad/s. But if you use that value in the code, the results are very bad because the propellers begin to turn too fast. It means the values written in the PWM moduls are too high. I modify the code and increase the value of the maximum velocity to 700 rad/s and the results are much better. Thus it is also possible to use that speed and not only the PID coefficients to change the behaviour of the aircraft.

5.3 ISSUES

In that section, I will present some issues. First of all, there is the filtering problem. I am currently working on that point because I think it is the major problem. It is very important to have good data. The control of such aircraft is very hard. It requires a good estimation of the state and a quick response to perturbations. I use Matlab to design a lowpass filter which fits to our case. Otherwise, I have others ideas but I will be too short in time to test them.

An other issue is about the scheduling of the thread. If nothing is done in the program to schedule the threads, it is the OpenMoko scheduler which handles that. With only two threads (main and accelerometer threads), Figure 5.1 shows one possible behaviour.

Listing 5.1 Scheduling without policy

```
1  Begin controller
   going to write on motor 0
3          End Accelopenmoko
           Begin Accelopenmoko
5          End Accelopenmoko
           Begin Accelopenmoko
7          End Accelopenmoko
           Begin Accelopenmoko
9          End Accelopenmoko
           Begin Accelopenmoko
11 finished writing on motor 0, going to write on motor 1
           End Accelopenmoko
13         Begin Accelopenmoko
           End Accelopenmoko
15         Begin Accelopenmoko
           End Accelopenmoko
17         Begin Accelopenmoko
           End Accelopenmoko
19         Begin Accelopenmoko
   finished writing on motor 1, going to write on motor 2
21 finished writing on motor 2, going to write on motor 3
           End Accelopenmoko
23         Begin Accelopenmoko
           End Accelopenmoko
25         Begin Accelopenmoko
           End Accelopenmoko
27         Begin Accelopenmoko
           End Accelopenmoko
29         Begin Accelopenmoko
   finished writing on motor 3
31 End controller
```

We notice that the main thread is always cut by the other one. For example, at the line 11, we see that the command has been written for the motor 0, but before writting on the motor 1, the accelerometer thread gets new values. This behaviour is normal because nothing have been done to specify how to assign the tasks. Then I tried to schedule the different threads using mutex. I finally obtain Figure 5.2.

Listing 5.2 Scheduling with one policy

```
1  Begin controller
   going to write on motor 0
```

```
finished writing on motor 0, going to write on motor 1
finished writing on motor 1, going to write on motor 2
finished writing on motor 2, going to write on motor 3
finished writing on motor 3
End controller

Begin accelmoko
2       3       0       1
End accelmoko
```

The accelerometer thread gets the attitude (2 3 1 0 corresponds to the three accelerations and the synchronization event) and then the main thread writes the four commands. That's task assignment seemed more suitable, but the flight tests showed it was not working. The reaction of the aircraft was too slow and it was not possible to stabilize. I found that result really strange. Finally, I came back to the first point where the OpenMoko scheduler assigns the ressources. It is possible to try others scheduling policies and see which one get the best results.

The last point I'd like to discuss here is the integral effect in the PID controller. My Matlab model does not use any integral action but I try to add one for several flight tests. The advantage of the integral term is to remove the static error, but the dynamic of the integral is also slower. At the beginning, I compute the integral by summing the current error with the old integral. The results were not very good. It appears that keeping a history of all the previous errors is not a good idea. So I only calculate the integral using with a finite numbers of errors. The results were much better. I think that, even if the matlab model is good, an integral term helps for the stabilization. An other track is to use different coefficients to compute the integral. Thus the weights can give more significance to the oldest or newest points.

5.4 CONCLUSION

As in most projects, the implementation of the controller had been the longest part in my internship. In fact, it is often easy to get good results in simulations, but unmodeled effects appear during the flight tests. Moreover others problems are discovered, for instance the insufficient filtering here.

Chapter 6

Conclusion

6.1 RETROSPECTIVE OF MY INTERNSHIP

The first step of my internship was to get used to the aircraft and learn C language. Using the code of the previous year and tutorials on the internet, the C learning was not difficult at all. Then I designed a new structure to have a more convenient aircraft.

Then I began to work on the mathematical model of the aircraft. A good model is essential before designing a controller. I think the model is pretty good for the application wanted: stabilized flight in an indoor space. Once the model obtained, I implemented it with Matlab to test different control algorithms. Then I got some good results in simulations.

The flight tests was the final step. I could see some problems during the tests: filtering, access to the yaw angle, scheduling... Most of them were already known but some were new.

About my initial planning, I see now that it was very optimistic. I did not have the time to work experimentally on the altitude controller as I am still working on the attitude stabilization. This delay is mostly due to technical problems. For example I had few problems with the printed circuit board. Moreover experimentations are slowed by the phone and its technical problems.

6.2 CONTRIBUTION TO THE PROJECT

Today the quadrotor is not able to fly. It can just make little 'jumps', but it does not reverse anymore.

One of the main contribution is the dynamic model obtained in Chapter 3 which is a very good start for the stabilization problem. It is now possible to design controllers using

mathematical equations. I proposed in Chapter 4 different control algorithms based on that model. Some of these algorithms were tested and implemented.

I also solved several problems: new structure, adding a compass to get the yaw angle... Even if all the problems have not been resolved (filtering problem), tools have been made to help for the development of the project. For instance it is now easier to design filters because it is possible to configure and test them directly on experimental data. A lot of code is also available: programs to test all the components, controller implementation...

6.3 FUTURE WORK

During my internship, I read a lot of papers or works about quadrotors. I will just expose here the ideas, which may solve some problems. First of all, I think it will be good to build a new printed board with new components. My idea is to use at least two gyroscopes, one for the pitch and one for the roll angle. All the 'big' projects used gyroscopes (STARMAC...). Moreover accelerometers can not see the difference between tilt and translation. I carried out an experiment where the Openmoko was lying on a table and I moved it. A program computed the attitude using the accelerometer data. The aircraft sees pitch and roll angles

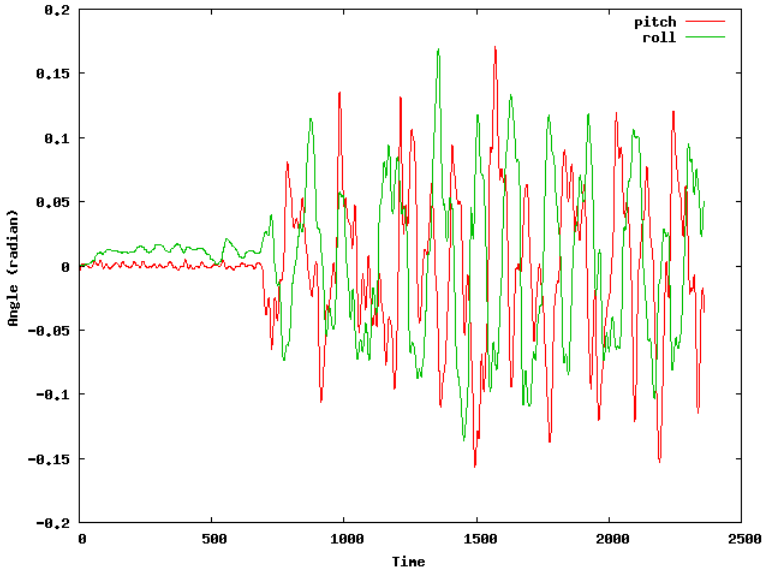


Figure 6.1 Variations of the angles due to accelerations

variying but they were always equal to zero. It is the main reason of using gyroscopes. However gyroscopes have problems too, the big one is that they tend to drift with the time. It is why an accelerometer is often working with them to compensate that drift.

Then more powerful motors could also be used. With the current motors, we can't increase the payload: camera or anything else. But more powerful motors mean more energy and less life time due to the battery storage.

About the filtering, the best results are obtained with a Kalman filter. That kind of filter is not easy to understand but it appears to work better than any lowpass filter.

My last point is about the mobile phone. The OpenMoko was a well-founded choice. But it has few drawbacks: WiFi and I^2C can breakdown, the accelerometers can't work together... Maybe a new platform can be considered. I know very good boards exist for that purpose (see the MikroKopter website).

List of Figures

List of Tables

Bibliography

[1] S. Bouabdallah, P. Murrieri, and R. Siegwart. Design and control of an indoor micro quadrotor. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems, New Orleans, USA, 2004, 2004.

[2] S. Bouabdallah, A. Noth, and R. Siegwart. Pid vs lq control techniques applied to an indoor micro quadrotor. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems, Sendai, Japan, 2004, 2004.

[3] Antoine Claveyrollas, Jean Corbisier, Sébastien Fitte, and Florian Rigaud. Stabilisation du drone quadrirotor. 2007-2008.

[4] Xbird forum. http://forum.xbird.fr/.

[5] Alexis Frenot, Anthony Gossmann, and Romaric Guillerm. Stabilisation d'un quadrirotor. 2005-2006.

[6] Gabriel Hoffmann, Dev Gorur Rajnarayan, Steven L. Waslander, David Dostal, Jung Soon Jang, and Claire J. Tomlin. The stanford testbed of autonomous rotorcraft for multi-agent control. In *the Digital Avionics System Conference 2004*, Salt Lake City, UT, November 2004.

[7] Gabriel M. Hoffmann, Haomiao Huang, Steven L. Waslander, and Claire J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Hilton Head, SC, August 2007. AIAA Paper Number 2007-6461.

[8] Matthias Katzengruber. Semi-autonomous control of an indoor model helicopter. 2009.

[9] lm-sensors website. http://lm-sensors.org/.

[10] P. McKerrow. Modelling the draganflyer four-rotor helicopter. 2004.

[11] Wiki OpenMoko. http://wiki.openmoko.org/wiki/Accelerometer_data_retrieval.

[12] Adeline Serrecourt and Julien Chapuis. Stabilisation d'un quadrirotor. 2006-2007.

[13] Benedikt Sieghartsleitner. Semi-autonomous control of an indoor model helicopter. 2009.

[14] Steven L. Waslander, Gabriel Hoffmann, Jung Soon Jang, and Claire J. Tomlin. Multi-agent x4-flyer testbed design: Integral sliding mode vs. reinforcement learning. In *IROS 2005*, Edmonton, AB, Canada, August 2005. IEEE.

[15] Automous Systems Lab website. http://asl.epfl.ch/research/projects/VtolIndoorFlying/indoorFlying.php.

[16] Catia website. http://www.3ds.com/products/catia/welcome/.

[17] Draganfly website. http://www.draganfly.com/uav-helicopter/draganflyer-x6/applications/.

[18] NICTA website. http://nicta.com.au/.

[19] STARMAC website. http://hybrid.eecs.berkeley.edu/index.php?section=37.

[20] Compass with tilt compensation. http://www.sparkfun.com/commerce/product_info.php?products_id=8656.

Chapter A

HowTo start the quadrotor

I will explain here how to start the OpenMoko and the aircraft. It is pretty easy but you need to be careful when you connect the batteries. For instance a capacitor could explode if the big battery (for the motors) is not well pluged.

- Turn on the openmoko: push the power button until OpenMoko logo appears on the screen.
- Wait until it is fully booted (pretty long, around 2 minutes).
- Connect to the OpenMoko, refer to Appendix B.
- Check if the I2C bus and the accelerometer are working. In the terminal, launch the command *dmesg* to check the I2C bus state. If you obtain the following line,

  ```
  s3c2440-i2c s3c2440-i2c: cannot get bus (error -110)
  ```

  you could have a problem with the I2C bus. To check the accelerometer, launch the command *hexdump /dev/input/event3*. If nothing happens, then the accelerometer is not working. In both cases (I2C or/and accelerometer problems), you need to reboot. You can use the reboot command. But it is often not enough, so you need to shut down the mobile, remove the battery, replace it and then restart. If everything is ok, then enter the command *echo 400 > /sys/bus/spi/drivers/lis302dl/spi0.1/sample_rate* to speed up the accelerometers. Now you can run your program on the OpenMoko.
- Plug the circuit board supply and switch on the power supply using the button (green led is on).
- Before wiring the big battery, I used to launch my program to be sure that the four motors are off, then I plug the big battery.

To get more information about the accelerometers, see [11].
The last thing I want to point out is the power supply of the PCB. This power supply comes from the battery of the mobile phone. Two wires are linked from the PCB to the

battery. Last year, they soldered the wires to the terminals. I really do not like that solution (soldering on the phone...). I prefer doing nodes with the wires and the good terminals (plus or minus). It works well and it is safe. I used a screwdriver to make the nodes.

Chapter B

HowTo connect to the OpenMoko

You have three different ways to connect to the OpenMoko. You can choose the way you connect depending on what you are doing on the aircraft. The wifi connection is the most useful but it can sometimes crash so the usb connection can be preferred when the OpenMoko doesn't need to be free.

B.1 USB CONNECTION

- Turn on the openmoko and wait until it is fully booted.
- Plug in the usb wire.
- The standart ip address of the openmoko is 192.168.0.202. Set up the usb0 device of your computer,
  sudo ifconfig usb0 192.168.0.200 netmask 255.255.255.0
  Try a ping to check if it works,
  ping 192.168.0.202
- Use ssh to connect to the openmoko,
  ssh root@192.168.0.202
  If you have a problem with a RSA key, try to remove the file /.ssh/knownhosts

Then you are in the mobile terminal and you can use it like any other terminal.

B.2 WIFI CONNECTION

For the wifi connection, I used a router with the following IP and MAC address: 192.168.1.1 and 00:14:BF:3A:60. On my personal laptop, the WiFi interface is eth1. Here again you have two different scenarios, which depends if you have modified the /etc/network/interfaces file. Let assume the interfaces file has not been modified,

- Connect to the OpenMoko using the usb wire (see the above method).

- On the phone via ssh, configure the eth0 device for WiFi ,
  iwconfig eth0 essid linksys
  ifconfig eth0 192.168.1.2 netmask 255.255.255.0
- On the computer,
  sudo iwconfig eth1 essid linksys
  sudo ifconfig eth1 192.168.1.3 netmask 255.255.255.0
  Try a ping to check if it works,
  ping 192.168.1.2
- Connect to the mobile via ssh,
  ssh root@192.168.1.2

A faster way to connect using WiFi is to modify the /etc/network/interfaces file. In this file, you need to modify the WiFi part like that,

```
# Wireless interfaces
iface wlan0 inet dhcp
wireless_mode managed
wireless_essid any

auto eth0
iface eth0 inet static
wireless_mode managed
wireless_essid linksys
address 192.168.1.2
netmask 255.255.255.0

iface atml0 inet dhcp
```

Now when you turn on the OpenMoko, it will automatically connect to the router and all you need to do is,

- On the computer,
  sudo iwconfig eth1 essid linksys
  sudo ifconfig eth1 192.168.1.3 netmask 255.255.255.0
  Try a ping to check if it works,
  ping 192.168.1.2
- Connect to the mobile via ssh,
  ssh root@192.168.1.2

B.3 DOWNLOAD/UPLOAD DATAS

Once connected to the mobile, it is possible to download or upload datas. Using scp, it is very easy. The scp command works like that,

scp source_path destination_path

For example to download datas from the OpenMoko (using the WiFi connection),

scp root@192.168.1.2:/home/root/pitch.csv .

The point means it will copy the file in the current directory. For uploading,

scp /my/path/myApplication root@192.168.1.2:/home/root

Chapter C

HowTo program for the OpenMoko

C.1 INSTALL THE ENVIRONMENT

On this project, all the code is in C language. To get executable files which run on the OpenMoko, you need at least to install the good compiler (openmoko-gcc). Go on this page http://wiki.openmoko.org/wiki/Toolchain and install the paquets in the Prerequisites section. Then go on http://andreasdalsgaard.blogspot.com/2008/07/openmoko-development-in-5-minutes.html (the link is given on previous page in the section Downnloading and installing). Then you just need to do the step 1. Normally you now have a directory /usr/local/openmoko. The compiler use for this project is /usr/local/openmoko/arm/arm-angstrom-linux-gnueabi/bin/gcc.
Basically you can now program (using text editor) and compile code (using Makefile) for the OpenMoko. I prefer using an IDE to program because I find it is more user friendly. I decided to install Eclipse using the paquet manager. Then you need the C plugin called eclipse-cdt. Once installed, the last step is to configure the compiler. On your C project, do a right click, then Properties. In the C/C++ build section, change the path of the GCC C Compiler and GCC C Linker. You can also add libraries here (pthread...).

C.2 LISTINGS OF ALL MY CODE

C language

- project OpenMoko: it is the code of Matthias. I modified his code to clean it (replace all the include "file.c" by include "file.h"...). See his report to have more details.
- project Attitude: it is my attitude stabilization program.
- project AccelData: this project gets the data from the accelerometer. It is useful at the beginning to see how the accelrometer works. Then it is used to test new filters. I also write a function which computes the difference between two times (struct timeval).

- project Compass: it is the program to test the compass with tilt compensation. Run it with the command *Compass*.
- project UsData: like the previous project, it is for testing the ultrasonic sensors.
- project Delay_Hardware and Delay_Software : I wrote these projects to determine the delay to include in my matlab model. The Delay_Hardware project must be used with a oscilloscope (watch the SDA signal and the motor signal).
- others projects: just some code I wrote to help me during the internship. It is not very useful now.

Matlab

- fft.m: this m-file is used to design filters and test it on experimental datas. We can also see the results in the frequency domain.
- Simulink models: I made several models. To run a simulation, the principle is always the same,
  - run the const_global.m file (file with all the constants)
  - run the simulation
  - run the plot_all.m file to visualize the results

  For each block in the model, there is one or several associated m-files.

Chapter D

Compile a kernel for the OpenMoko

Compiling a kernel for the OpenMoko is pretty hard for people who know nothing about kernel compilation (like me). You may need this tutorial if you get a new mobile or if you want to change the current linux distribution. Here I just give the different steps I follow to obtain my kernel. It is possible that it won't work in the future because the Makefile is different or something else but anyway that could help.

- Download the MokoMakefile, http://wiki.openmoko.org/wiki/MokoMakefile.
- Installed the required packages (see package requirement on the previous webpage).
- Create a directory for the MokoMakefile, for me it was,

  /home/hugo/Documents/moko

- In this directory, do the command, *make update-common* and *make setup*.
- Before building the image, check the following text files to add the patch line:

  .../moko/fso-milestone5/openembedded/packages/linux/linux-openmoko-2.6.24_git.bb
  .../moko/fso-milestone5/openembedded/packages/linux/linux-openmoko-2.6.28_git.bb

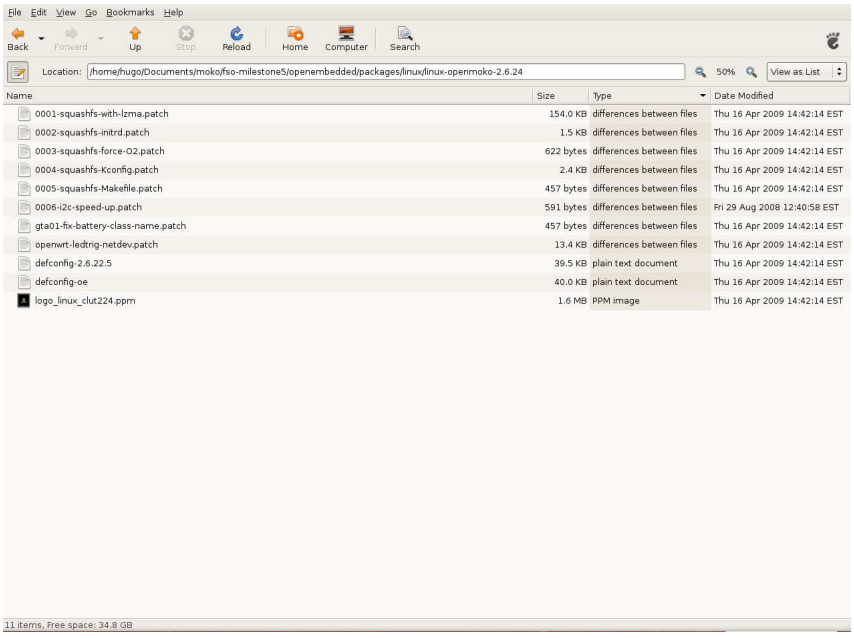
  Here is an example,

```
require linux.inc
require linux-openmoko.inc

DESCRIPTION = "The Linux kernel for the Openmoko devices GTA01
    (Neo1973) and GTA02 (Neo FreeRunner)"

KERNEL_RELEASE = "2.6.24"
KERNEL_VERSION = "${KERNEL_RELEASE}"

OEV = "oe5"
PV = "${KERNEL_RELEASE}-${OEV}+gitr${SRCREV}"
PR = "r1"
```

```
SRC_URI = "\
git://git.openmoko.org/git/kernel.git;protocol=git;branch=
    stable \
\
file://0001-squashfs-with-lzma.patch;patch=1 \
file://0002-squashfs-initrd.patch;patch=1 \
file://0003-squashfs-force-O2.patch;patch=1 \
file://0004-squashfs-Kconfig.patch;patch=1 \
file://0005-squashfs-Makefile.patch;patch=1 \
file://0006-i2c-speed-up.patch;patch=1 \
\
file://openwrt-ledtrig-netdev.patch;patch=1 \
file://gta01-fix-battery-class-name.patch;patch=1 \
\
file://defconfig-oe \
"
S = "${WORKDIR}/git"
```

- Paste the patch i2c-speedup.patch in the both files,

```
.../moko/fso-milestone5/openembedded/packages/linux/linux-
    openmoko-2.6.24
.../moko/fso-milestone5/openembedded/packages/linux/linux-
    openmoko-2.6.28
```



- On the moko makefile page, you read: "MokoMakefile no longer builds the QEMU-based Neo1973 emulator." this makes the compilation failed. So you need to install qemu via apt-get and in the local.conf file (.../moko/fso-milestone5/conf) add the following lines:

```
ENABLE_BINARY_LOCALE_GENERATION = "0"
```

```
ASSUME_PROVIDED += "qemu-native"
PARALLEL_MAKE = "-j 3"
```

The last line is an option for compiling faster if the computer can.

- Make the image, *make fso-gta02-milestone5-image*
- Few hours later, if the compilation suceeds, you find your new kernel image and root file system in the directory,

```
/home/hugo/Documents/moko/fso-milestone5/tmp/deploy/glibc/images
```