DRONE STATIONNAIRE

PROJET ELECTRONIQUE PEIP 2 LEHYAN FLOURIOT G3

**Rapport de séance 5 :**

Pour commencer, j’ai réalisé un test de vol avec le « codefinal3 » que j’ai publié sur GitHub.

Voila le compte-rendu de ce test :

Le drone décolle légèrement car je lui ai demandé d’assigner des valeurs maximales de moteur relativement faibles, il se stabilise en fonction des valeurs d’inclinaison.

Jusque-là tout parait très prometteur, MAIS il y a un problème réellement encombrant.

Ces étapes de stabilisations sont beaucoup trop lentes donc le drone fait des à-coups trop importants pour avoir l’équilibre. Quand il se redresse sur la gauche, il doit se redresser sur la droite etc… Cela fait des boucles sans fin donc il ne se stabilise jamais vraiment et lutte continuellement.

Pour pallier ce problème il va falloir augmenter d’une manière considérable la vitesse des informations que renvoie le MPU6050 et donc par continuité la vitesse des corrections d’inclinaisons.

J’ai pu comprendre après quelques recherches que je ne peux pas utiliser les librairies que j’utilisé tel que « I2C » etc… car elles sont trop lentes. Je vais donc utiliser la librairie « Wire » de l’Arduino.

Je ne vais plus avoir besoin de brancher le INT du MPU6050.

Donc il va falloir changer les configurations du composant MPU6050 pour augmenter son taux de rafraichissement. Pour cela nous devons changer des valeurs précises qui sont détaillées sur ce lien :

<https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf>

En gros, on change la vitesse de transmission et la précision des capteurs accéléromètre et gyroscopique avec ce genre de schéma :

SENSIBILITE DU GYROSCOPE :

Wire.beginTransmission(0x68); // Start communication with MPU

Wire.write(0x1B); // Request the GYRO\_CONFIG register

Wire.write(0x08); // Apply the desired configuration to the register : +-500°/s

Wire.endTransmission(); // End the transmission

SENSIBILITE DE L’ACCELEROMETRE :

Wire.beginTransmission(0x68); // Start communication with MPU

Wire.write(0x1C); // Request the ACCEL\_CONFIG register

Wire.write(0x10); // Apply the desired configuration to the register : +-8g

Wire.endTransmission(); // End the transmission

FREQUENCE D’HORLOGE :

Wire.beginTransmission(0x68); // Start communication with MPU

Wire.write(0x6B); // Request the PWR\_MGMT\_1 register

Wire.write(0x00); // Apply the desired configuration to the register

Wire.endTransmission(); // End the transmission

CHANGEMENT DU FILTRE PASSE BAS (conseillée) :

Wire.beginTransmission(0x68); // Start communication with MPU

Wire.write(0x1A); // Request the CONFIG register

Wire.write(0x03); // Apply the desired configuration to the register : DLPF about 43Hz

Wire.endTransmission(); // End the transmission

Ce qui donne finalement dans une bonne fonction propre :

#define MPU\_ADDRESS 0x68 // I2C address of the MPU-6050

/\*\*

\* Configure la plage de fonctionnement du gyroscope et de l'accéléromètre :

\* - accéléromètre: +/-8g

\* - gyroscope: 500°/sec

\*

\* @return void

\*/

void setupMpu6050Registers() {

// Configure power management

Wire.beginTransmission(MPU\_ADDRESS); // Start communication with MPU

Wire.write(0x6B); // Request the PWR\_MGMT\_1 register

Wire.write(0x00); // Apply the desired configuration to the register

Wire.endTransmission(); // End the transmission

// Configure the gyro's sensitivity

Wire.beginTransmission(MPU\_ADDRESS); // Start communication with MPU

Wire.write(0x1B); // Request the GYRO\_CONFIG register

Wire.write(0x08); // Apply the desired configuration to the register : ±500°/s

Wire.endTransmission(); // End the transmission

// Configure the acceleromter's sensitivity

Wire.beginTransmission(MPU\_ADDRESS); // Start communication with MPU

Wire.write(0x1C); // Request the ACCEL\_CONFIG register

Wire.write(0x10); // Apply the desired configuration to the register : ±8g

Wire.endTransmission(); // End the transmission

// Configure low pass filter

Wire.beginTransmission(MPU\_ADDRESS); // Start communication with MPU

Wire.write(0x1A); // Request the CONFIG register

Wire.write(0x03); // Set Digital Low Pass Filter about ~43Hz

Wire.endTransmission(); // End the transmission

}

Nous devons donc désormais lire les valeurs brutes que le capteur stocke.

Pour ce faire, nous devons requêter 14 registres numérotés de 59 à 72 comprenant également une information de température (c’est toujours bien).

#include <Wire.h>

#define MPU\_ADDRESS 0x68 // I2C address of the MPU-6050

int acc\_raw[3] = {0,0,0};

int gyro\_raw[3] = {0,0,0};

int temperature = 0;

void readSensor() {

Wire.beginTransmission(MPU\_ADDRESS);// Start communicating with the MPU-6050

Wire.write(0x3B); // Send the requested starting register

Wire.endTransmission(); // End the transmission

Wire.requestFrom(MPU\_ADDRESS,14); // Request 14 bytes from the MPU-6050

// Wait until all the bytes are received

while(Wire.available() < 14);

acc\_raw[X] = Wire.read() << 8 | Wire.read(); // Add the low and high byte to the acc\_raw[X] variable

acc\_raw[Y] = Wire.read() << 8 | Wire.read(); // Add the low and high byte to the acc\_raw[Y] variable

acc\_raw[Z] = Wire.read() << 8 | Wire.read(); // Add the low and high byte to the acc\_raw[Z] variable

temperature = Wire.read() << 8 | Wire.read(); // Add the low and high byte to the temperature variable

gyro\_raw[X] = Wire.read() << 8 | Wire.read(); // Add the low and high byte to the gyro\_raw[X] variable

gyro\_raw[Y] = Wire.read() << 8 | Wire.read(); // Add the low and high byte to the gyro\_raw[Y] variable

gyro\_raw[Z] = Wire.read() << 8 | Wire.read(); // Add the low and high byte to the gyro\_raw[Z] variable

}

Nous avons des valeurs « brutes », ce ne sont pas des degrés facilement lisibles. Il nous faut donc les convertir en une valeur compréhensible.

Pour avoir des valeurs en degré par seconde, il va falloir diviser les données brutes du gyroscope par 65.5.

Malheureusement, on peut constater que les valeurs sont légèrement décalées par rapport à la réalité. Il faut donc estimer puis compenser ce léger décalage. Pour cela, il faut prendre beaucoup de mesures et calculer leur valeur moyenne, cela nous donnera une correction à appliquer sur chaque axe.

// On déclare quelques constantes pour la lisibilité

#define X 0

#define Y 1

#define Z 2

float gyro\_offset[3] = {0,0,0};

int gyro\_raw[3] = {0,0,0};

/\*\*

\* Calibrage du MPU6050 : on prend 2000 échantillons et on calcule leur valeur moyenne.

\* Durant cette étape, le drone doit rester sur une surface horizontal sans bouger.

\*

\* @return void

\*/

void calibrateMpu6050()

{

int max\_samples = 2000;

for (int i = 0; i < max\_samples; i++) {

readSensor();

gyro\_offset[X] += gyro\_raw[X];

gyro\_offset[Y] += gyro\_raw[Y];

gyro\_offset[Z] += gyro\_raw[Z];

// Just wait a bit before next loop

delay(3);

}

// Calculate average offsets

gyro\_offset[X] /= max\_samples;

gyro\_offset[Y] /= max\_samples;

gyro\_offset[Z] /= max\_samples;

}

Il reste plus qu’a soustraire ces valeurs aux valeurs brutes, puis les diviser par le facteur de sensibilité que l’on a défini.

void loop()

{

readSensor();

real\_value[X] = (gyro\_raw[X] - gyro\_offset[X]) / SSF\_GYRO;

real\_value[Y] = (gyro\_raw[Y] - gyro\_offset[Y]) / SSF\_GYRO;

real\_value[Z] = (gyro\_raw[Z] - gyro\_offset[Z]) / SSF\_GYRO;

}

Cependant j’aimerai des degrés et pas des degrés par secondes.

Il va donc falloir une étape de plus :

On va devoir intégrer et sommant plusieurs valeurs par secondes. Nous allons partir sur une fréquence de 250Hz (conseillée car 200Hz est trop faible pour une stabilisation efficace). On va sommer les valeurs toutes les 4ms, et donc devoir diviser la valeur finale par 250.

Dans l’Arduino on le mettra comme ça :

angle[PITCH] += (gyro\_raw[X] - gyro\_offset[X]) / (250 \* SSF\_GYRO);

angle[ROLL] += (gyro\_raw[Y] - gyro\_offset[Y]) / (250 \* SSF\_GYRO);

Pour le Pitch, il faut mettre un sinus (sinon ça pose des problèmes lors de plans inclinés) :

pitch\_angle += roll\_angle \* sin(yaw\_raw)

Le code ressemble donc à cela au final pour récupérer très rapidement les valeurs du MPU6050 :

#include <Wire.h>

#define MPU\_ADDRESS 0x68 // I2C address of the MPU-6050

#define SSF\_GYRO 65.5 // Sensitivity Scale Factor of the gyro

#define X 0 // X axis

#define Y 1 // Y axis

#define Z 2 // Z axis

#define FREQ 250 // Sampling frequency

int acc\_raw[3] = {0,0,0};

int gyro\_raw[3] = {0,0,0};

int period;

int temperature = 0;

unsigned long int loop\_timer;

unsigned long int now; // Exists just to reduce the calls to micros()

float gyro\_angle[3] = {0,0,0};

float gyro\_offset[3] = {0,0,0};

void setup() {

// Setup MPU registers

setupMpu6050Registers();

// Calibrate MPU6050 calculating average offset of each axis

calibrateMpu6050();

// Calculate period in µs

period = (1000000/FREQ);

// Init loop timer

loop\_timer = micros();

}

void loop() {

readSensor();

calculateGyroAngles();

// Do nothing until sampling period is reached

while((now = micros()) - loop\_timer < period);

loop\_timer = now;

}

/\*\*

\* Calculate angles with gyro data

\*/

void calculateGyroAngles()

{

// Substract offsets

gyro\_raw[X] -= gyro\_offset[X];

gyro\_raw[Y] -= gyro\_offset[Y];

gyro\_raw[Z] -= gyro\_offset[Z];

// Angle calculation

gyro\_angle[X] += gyro\_raw[X] / (FREQ \* SSF\_GYRO);

gyro\_angle[Y] += (gyro\_raw[Y]) / (FREQ \* SSF\_GYRO);

// Transfer roll to pitch if IMU has yawed

gyro\_angle[Y] -= gyro\_angle[X] \* sin(gyro\_raw[Z] \* (PI / (FREQ \* SSF\_GYRO \* 180)));

gyro\_angle[X] += gyro\_angle[Y] \* sin(gyro\_raw[Z] \* (PI / (FREQ \* SSF\_GYRO \* 180)));

}

Avec les fonctions définis avec les codes précédents.

Ce n’est pas fini, on constate qu’avec cette méthode, les valeurs finissent par dérivent et finissent par devenir aberrantes.

C’est ici qu’intervient l’accéléromètre !

On va mesurer les valeurs de l’accéléromètre et les combiner avec celles du gyroscope, car les vibrations ne nous permettent pas d’utiliser uniquement l’accéléromètre.

Valeurs de l’accéléromètre :

#define X 0 // X axis

#define Y 1 // Y axis

#define Z 2 // Z axis

int acc\_raw[3] = {0,0,0};

float acc\_angle[3] = {0,0,0};

float acc\_total\_vector;

/\*\*

\* Calculate angles using the 3 axis accelerometer.

\*/

void measureAccelerometerAngles()

{

// Calculate total 3D acceleration vector : √(X² + Y² + Z²)

acc\_total\_vector = sqrt(pow(acc\_raw[X], 2) + pow(acc\_raw[Y], 2) + pow(acc\_raw[Z], 2));

// To prevent asin to produce a NaN, make sure the input value is within [-1;+1]

if (abs(acc\_raw[X]) < acc\_total\_vector) {

acc\_angle[X] = asin((float)acc\_raw[Y] / acc\_total\_vector) \* (180 / PI); // asin gives angle in radian. Convert to degree multiplying by 180/pi

}

if (abs(acc\_raw[Y]) < acc\_total\_vector) {

acc\_angle[Y] = asin((float)acc\_raw[X] / acc\_total\_vector) \* (180 / PI);

}

}

Donc pour des mesures très précises, nous allons prendre 99.96% du gyroscope et 0.04% de l’accéléromètre.

pitch\_angle = pitch\_angle\_gyro \* 0.9996 + pitch\_angle\_acc \* 0.0004;

roll\_angle = roll\_angle\_gyro \* 0.9996 + roll\_angle\_acc \* 0.0004;

Par contre, lors de démarrage sur plan incliné, il faut initialiser les valeurs de gyroscope avec celle de l’accéléromètre en considérant que lors du démarrage le drone ne bouge pas !

if (initialized) {

pitch\_angle = pitch\_angle\_gyro \* 0.9996 + pitch\_angle\_acc \* 0.0004;

roll\_angle = roll\_angle\_gyro \* 0.9996 + roll\_angle\_acc \* 0.0004;

} else {

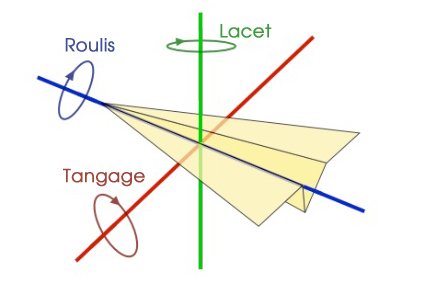
pitch\_angle\_gyro = pitch\_angle\_acc;

roll\_angle\_gyro = roll\_angle\_acc;

initialized = true;

}

On pose ensuite une référence pour les sens de rotations de la sorte :



* **aile gauche vers le haut : roulis (roll) positif**
* **nez vers le haut : tangage (pitch) positif**
* **nez vers la droite : lacet (yaw) positif**

Ces valeurs sont super importantes.

Le code final est carrément super gros, donc je le met sur le GitHub nommé « IMU »pour Inertial Measurement Unit .