DRONE STATIONNAIRE

PROJET ELECTRONIQUE PEIP 2 LEHYAN FLOURIOT G3

Rapport de séance 4 :

Nous en étions à la calibration de l'accéléromètre. J'ai finalement reussi faire cette étape en changeant d'ordinateur, celle-ci avait pour but de me renvoyer des valeurs d' « offset ». Ces précieuses valeurs vont me permettre par la suite de définir le point que je jugerai comme parfaitement horizontale, de sorte à définir mon 0° d'inclinaison. Cela représentera le point ou le drone est « stable » en vol.

J'ai obtenu les valeurs suivantes :

-3961 -927 580 179 -59 7

C'est le résultat du code CalibrageMPU sur le GitHub. Celui-ci ne vient évidemment pas de moi, et il est très correctement commenté donc je ne l'expliquerai pas.

Il est maintenant temps de les implémenter et d'essayer ce fameux MPU6050 (accéléromètre) :

J'implémente ce code que j'ai nommé « codetest » sur GitHub :

C'est la première version du code finale qui sera implémenté dans le drone.

```
/**
 * Usage, according to documentation(https://www.firediy.fr/files/drone/HW-01-V4.pdf) :
 *    1. Plug your Arduino to your computer with USB cable, open terminal, then type 1 to send max throttle to every ESC to enter programming mode
 *    2. Power up your ESCs. You must hear "beep1 beep2 beep3" tones meaning the power supply is OK
 *    3. After 2sec, "beep beep" tone emits, meaning the throttle highest point has been correctly confirmed
 *    4. Type 0 to send 0 throttle
 *    5. Several "beep" tones emits, wich means the quantity of the lithium battery cells (3 beeps for a 3 cells LiPo)
 *    6. A long beep tone emits meaning the throttle lowest point has been correctly confirmed
 *    7. Type 2 to launch test function. This will send 0 to 180 throttle to ESCs to test them
 */

#include <Servo.h>
#include <I2Cdev.h>
#include <MPU6050_6Axis_MotionApps20.h>
#include <Wire.h>
Servo motA, motB, motC, motD;
bool calibre = false;
char data;
```

```
// ---------------------------------------------------------------------------
#define YAW     0
#define PITCH   1
#define ROLL    2
// ---------------------- MPU650 variables ---------------------------------
// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// Orientation/motion vars
Quaternion q;       // [w, x, y, z]        quaternion container
VectorFloat gravity; // [x, y, z]           gravity vector
float ypr[3];       // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

volatile bool mpuInterrupt = false;     // Indicates whether MPU interrupt pin has gone high
// ---------------------------------------------------------------------------

//defintion des variables de puissance pour les moteurs et des angles du drone (yaw pitch roll)

int puissanceA = 40;
int puissanceB = 40;
int puissanceC = 40;
int puissanceD = 40;

float angleYaw;
float anglePitch;
float angleRoll;



// ---------------------------------------------------------------------------
/**
 * Interrup détection routine
 */
void dmpDataReady() {
   mpuInterrupt = true;
}




void setup() {
    Wire.begin();
   TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)

   Serial.begin(57600);

   Serial.println(F("Initializing I2C devices..."));
   mpu.initialize();

   // Verify connection
   Serial.println(F("Testing device connections..."));
   Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

   // Load and configure the DMP
   Serial.println(F("Initializing DMP..."));
```

```
    devStatus = mpu.dmpInitialize();

    // MPU calibration: set YOUR offsets here.
    mpu.setXAccelOffset(-3961);
    mpu.setYAccelOffset(-927);
    mpu.setZAccelOffset(580);
    mpu.setXGyroOffset(179);
    mpu.setYGyroOffset(-59);
    mpu.setZGyroOffset(7);

    // Returns 0 if it worked
    if (devStatus == 0) {
      // Turn on the DMP, now that it's ready
      Serial.println(F("Enabling DMP..."));
      mpu.setDMPEnabled(true);

      // Enable Arduino interrupt detection
      Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0 : #pin2)..."));
      attachInterrupt(0, dmpDataReady, RISING);
      mpuIntStatus = mpu.getIntStatus();

      // Set our DMP Ready flag so the main loop() function knows it's okay to use it
      Serial.println(F("DMP ready! Waiting for first interrupt..."));
      dmpReady = true;

      // Get expected DMP packet size for later comparison
      packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
      // ERROR!
      // 1 = initial memory load failed
      // 2 = DMP configuration updates failed
      // (if it's going to break, usually the code will be 1)
      Serial.print(F("DMP Initialization failed (code "));
      Serial.print(devStatus);
      Serial.println(F(")"));
    }


    motA.attach(3, 1000, 2000); // 1000 and 2000 are very important ! Values can be different with other ESCs.
    motB.attach(5, 1000, 2000);
    motC.attach(6, 1000, 2000);
    motD.attach(9, 1000, 2000);



    displayInstructions();

    //calibrage();

}

void loop() {
  if(calibre==false){
    if (Serial.available()) {
      data = Serial.read();

      switch (data) {
        // 0
        case 48 : Serial.println("Sending 0 throttle et fin de calibrage");
                  motA.write(0);
                  motB.write(0);
                  motC.write(0);
                  motD.write(0);

                  delay(3000);
```

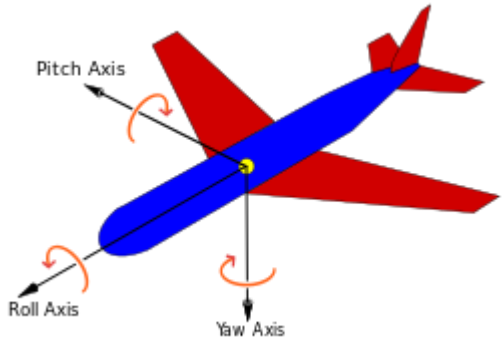
```
                calibre = true;
        break;

        // 1
        case 49 : Serial.println("Sending 180 throttle");
                motA.write(180);
                motB.write(180);
                motC.write(180);
                motD.write(180);
        break;

        // 2
        case 50 : Serial.print("Running test in 3");
                delay(1000);
                Serial.print(" 2");
                delay(1000);
                Serial.println(" 1...");
                delay(1000);
                test();
        break;
      }
   }
 }

 else
 {
   testStationnaire();
   }


}

void calibrage()
{

 Serial.println("Valeur maximal 180");

 motA.write(180);
 motB.write(180);
 motC.write(180);
 motD.write(180);
 delay(6000);

 Serial.println("Valeur minimale 0");

 motA.write(0);
 motA.write(0);
 motA.write(0);
 motA.write(0);
 delay(4000);



}

void test()
{
   for (int i=0; i<=180; i++) {
      Serial.print("Speed = ");
      Serial.println(i);
      motA.write(i);
      motB.write(i);
      motC.write(i);
      motD.write(i);
      delay(200);
   }
```

```
  Serial.println("STOP");
  motA.write(0);
  motB.write(0);
  motC.write(0);
  motD.write(0);
}
void displayInstructions()
{
  Serial.println("READY - PLEASE SEND INSTRUCTIONS AS FOLLOWING :");
  Serial.println("\t0 : Sends 0 throttle");
  Serial.println("\t1 : Sends 180 throttle");
  Serial.println("\t2 : Runs test function\n");
}

void testStationnaire()
{
 // If programming failed, don't try to do anything
  if (!dmpReady) {
    return;
  }

  // Wait for MPU interrupt or extra packet(s) available
  while (!mpuInterrupt && fifoCount < packetSize) {
    // Do nothing...
  }

  // Reset interrupt flag and get INT_STATUS byte
  mpuInterrupt = false;
  mpuIntStatus = mpu.getIntStatus();

  // Get current FIFO count
  fifoCount = mpu.getFIFOCount();

  // Check for overflow (this should never happen unless our code is too inefficient)
  if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
    // reset so we can continue cleanly
    mpu.resetFIFO();
    Serial.println(F("FIFO overflow!"));

    // Otherwise, check for DMP data ready interrupt (this should happen frequently)
  } else if (mpuIntStatus & 0x02) {
    // Wait for correct available data length, should be a VERY short wait
    while (fifoCount < packetSize) {
      fifoCount = mpu.getFIFOCount();
    }

    // Read a packet from FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);

    // Track FIFO count here in case there is > 1 packet available
    // (this lets us immediately read more without waiting for an interrupt)
    fifoCount -= packetSize;

    // Convert Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

    // Print angle values in degrees.


    angleYaw = ypr[YAW] * (180 / M_PI);
    anglePitch = ypr[PITCH] * (180 / M_PI);
    angleRoll = ypr[ROLL] * (180 / M_PI);
```

```
    Serial.print(angleYaw);
    Serial.print("\t");
    Serial.print(anglePitch);
    Serial.print("\t");
    Serial.println(angleRoll);

    //Correction des moteurs

    if (angleRoll < 0){
      puissanceD += 1;
      puissanceC += 1;
      puissanceA -= 1;
      puissanceB -= 1;
      }
    if (angleRoll > 0){
      puissanceD -= 1;
      puissanceC -= 1;
      puissanceA += 1;
      puissanceB += 1;
      }
    if (anglePitch > 0){
      puissanceD += 1;
      puissanceA += 1;
      puissanceC -= 1;
      puissanceB -= 1;
      }
    if (anglePitch < 0){
      puissanceD -= 1;
      puissanceA -= 1;
      puissanceC += 1;
      puissanceB += 1;
      }
    if (puissanceA < 20){puissanceA = 20;}
    if (puissanceB < 20){puissanceB = 20;}
    if (puissanceC < 20){puissanceC = 20;}
    if (puissanceD < 20){puissanceD = 20;}

    if (puissanceA > 60){puissanceA = 60;}
    if (puissanceB > 60){puissanceB = 60;}
    if (puissanceC > 60){puissanceC = 60;}
    if (puissanceD > 60){puissanceD = 60;}

  }
  /*
 motA.write(puissanceA);
 motB.write(puissanceB);
 motC.write(puissanceC);
 motD.write(puissanceD);

 Serial.print(puissanceC);
 Serial.print(" / ");
 Serial.println(puissanceB);

 Serial.print(puissanceD);
 Serial.print(" / ");
 Serial.println(puissanceA);
 */
}
```

Les valeurs d'offset sont insérées dans la partie en ORANGE du code, La fonction Calibrage à pour but de calibrer les ESC lorsque l'on met sous tension le drone, celle-ci ne fonctionne pas encore mais j'y travaille.

En VERT la partie qui imprime les valeurs données par l'accéléromètre sachant qu'elles suivent cette logique :



En BLEU figure la partie qui va demander le plus de travaille, et l'une des seules qui n'est absolument pas aboutit, en revanche j'ai testé les réactions du drone lorsque je change son inclinaison ; les moteurs réagissent, ils prennent donc bien en compte les valeurs de l'accéléromètre MAIS les réactions sont beaucoup trop lentes pour équilibrer le drone. Il me faut donc trouver un moyen pour faire une action proportionnelle à l'inclinaison du drone.

Ce code est brouillon et évolue énormément, d'ailleurs lorsque je publierai le rapport il n'est pas impossible qu'il y ait déjà plusieurs versions de « codetest » sur le GitHub.
Bref en tout cas, il me reste concrètement l'esthétique et l'équilibrage pour terminer mon projet.

Comme je l'ai montré sur les photos précédentes, le drone n'est pas facilement manipulable lorsqu'il s'agit de brancher des fils. DONC, je vais y remédier dans ce rapport.

La carte Arduino est au milieu du châssis donc les pins ne sont pas accessibles et cela commence à fortement m'énerver.

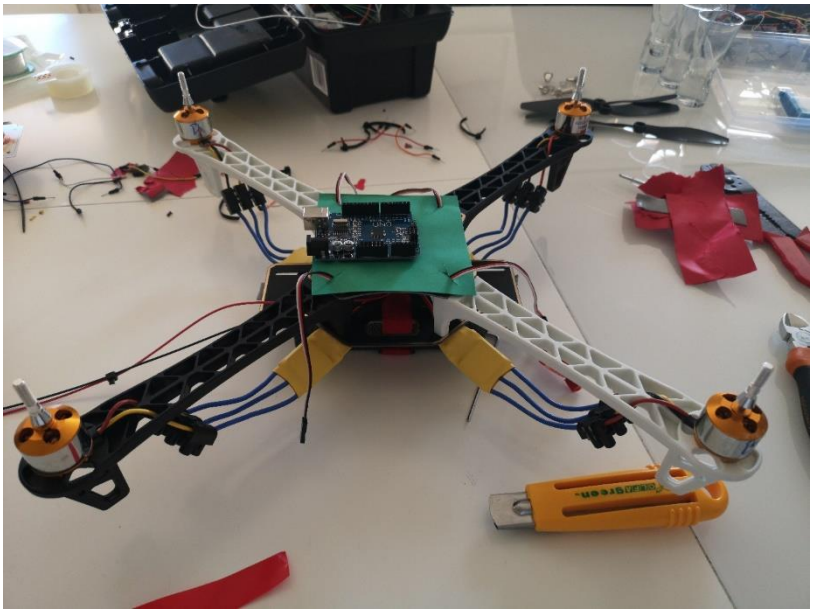
De plus les ESCs sont sur le dessus du drone, et ce n'est pas beau. Je vais donc les faires passer en dessous des bras.

<u>Commençons :</u>

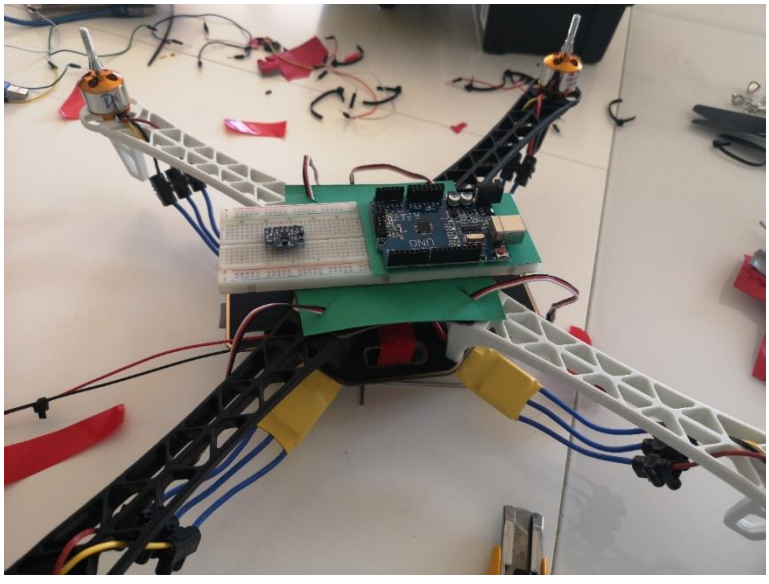
Je vais débuter en mettant la carte sur le dessus du drone, elle sera alors beaucoup plus accessible, et les câbles relient la masse et le + des ESCs iront au milieu du châssis.

Je vais aussi faire passer les câbles dans des trous du châssis pour plus de praticité.

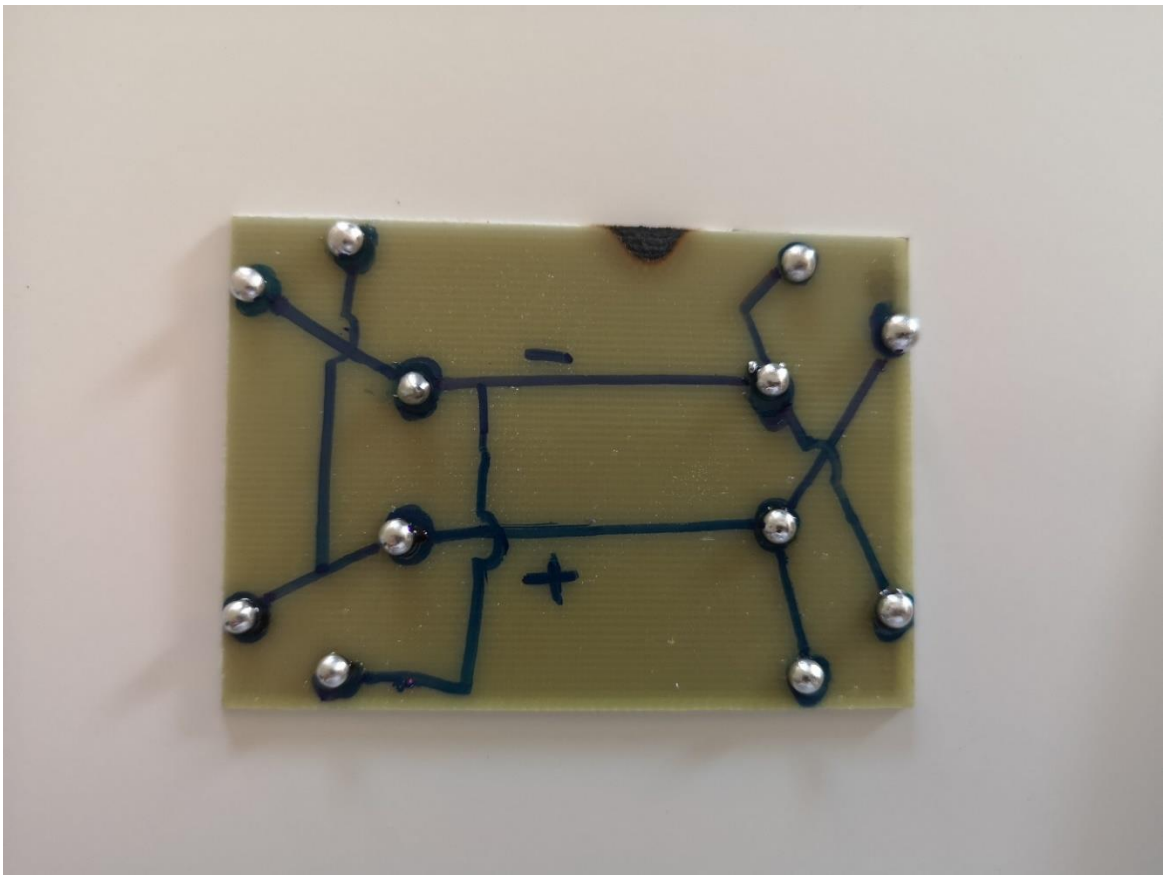
Et je mets du carton pour éviter les courts circuits dues aux parties métalliques du châssis.

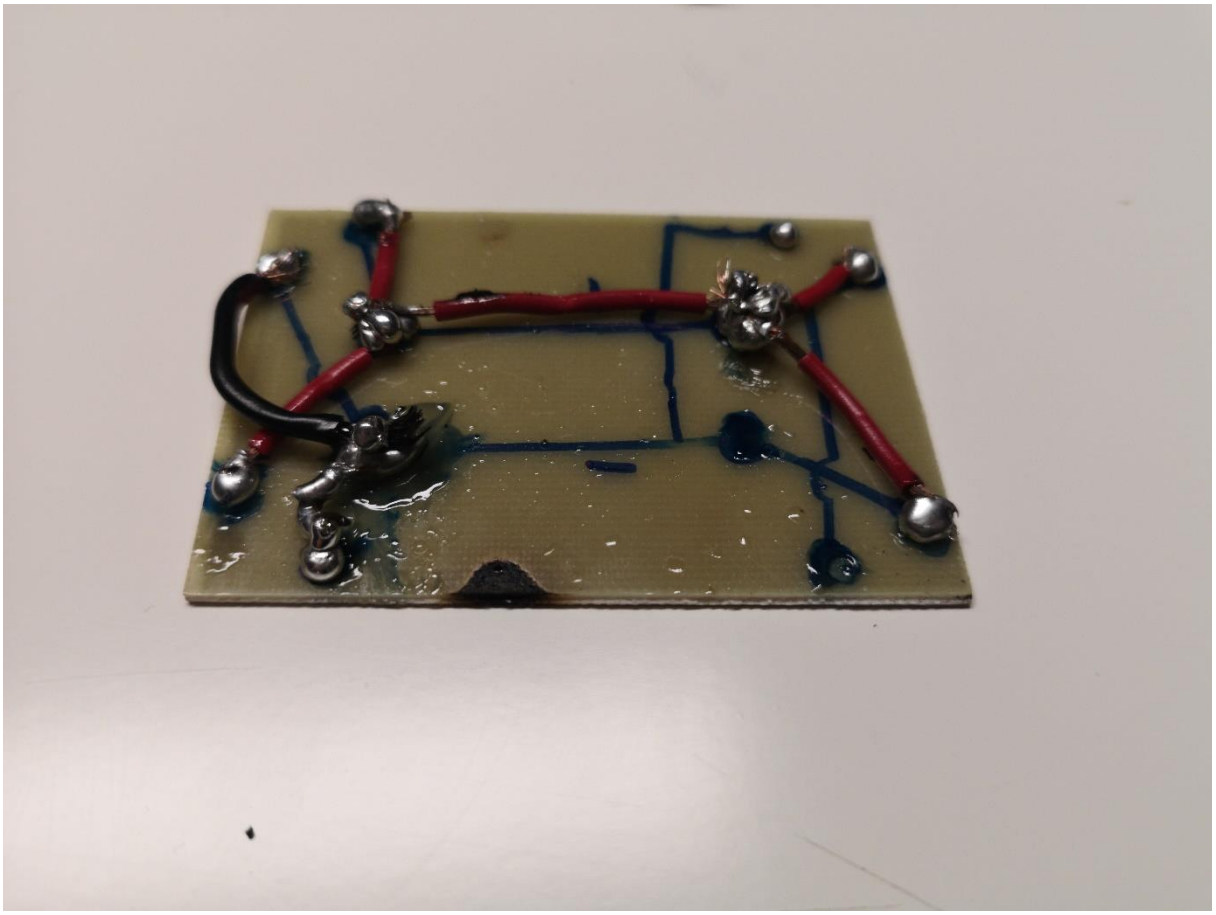


Finalement comme je ne sais pas si je peux découper la plaque qu'on l'on m'a confié, je vais scotcher la plaque puis l'Arduino dessus comme-ceci :



Cela étant fait, j'aimerai faire un petit circuit imprimé pour diviser le courant fournit par la batterie. Je vais donc « essayer » de souder une petite « PowerDistributionBoard » en suivant ce schéma au feutre :





Bon, alors comment dire, c'est extrêmement difficile de souder aussi petit avec mon matériel, donc je vais recommencer plus tard, les connecteurs rapides feront l'affaire pour le moment.

En revanche je dois construire un petit circuit imprimé à brancher sur le dessus de l'Arduino pour mettre proprement le MPU6050 et arrêter d'utiliser la planche en plastique fournit par la prépa. Ce genre de composant prend plusieurs jours voire semaines à arriver, donc en attendant je vais essayer de faire marcher le drone dans les conditions actuelles.

A cette fin de séance je me suis rendu compte après les recherches que le drone aurait des réactions atténuées lorsque la charge de la batterie diminuera, ce facteur doit être pris en compte lors de l'équilibrage.

Prochaine séance, j'annonce que le drone volera et se stabilisera !