Projet BE : Réalisation et Contrôle à distance d'un Quadri-rotor

Lohan BUSO

Antoine Gay-Puig

Université Toulouse-III-Paul-Sabatier

2021-2022

Tuteur enseignant : PERISSE Thierry thierryperisse@free.fr

Sommaire

Introduction

Ce projet de Bureaux d'étude, s'inscrit dans le cadre de la formation L3 EEA REL. Il permet de finaliser notre cursus en nous faisant travailler sur un projet en totale autonomie.

L'objectif de ce projet est de réaliser la conception et programmation de la commande d'un quadri-rotor en passant par toutes les étapes de conception d'un drone.

Ce rapport présentera dans un premier temps le principe de fonctionnement d'un quadri-rotor, ses composants et chaque parties essentielles. Nous détaillerons ensuite chaque parties et les résultats obtenus avant de conclure sur les perspectives de projet.

I. Drones

I.1. Qu'est-ce qu'un Drone ?

Dès que l'on entend le mot « drone », on pense de suite à ce genre de chose :

Figure 1 : Drone du commerce

Cependant, un drone peut aussi ressembler à ça :

Figure 2 : Drone militaire

De manière générale un drone est un aéronef sans pilote à bord qui peut être télécommandé ou en pilotage automatique. Il peut avoir un usage civil ou militaire.

Ici, ce sont les quadri-rotor (ou quadricoptère) qui sont le sujet principal du projet, il s'agit d'une catégorie bien particulière de drones.

Les quadri-rotor sont des aéronefs à voile tournante comportant quatre rotors pour assurer sa portance. Ces derniers sont généralement placés aux extrémités d'une croix mais il existe d'autres formes de cadres que ceux en X comme ceux en H ou encore en T.

Pour le projet, un cadre en X a été utilisé, déjà présent dans le matériel du projet.

Il est nécessaire que deux hélices tournent dans un sens et les deux autres tournent dans l'autre sens.

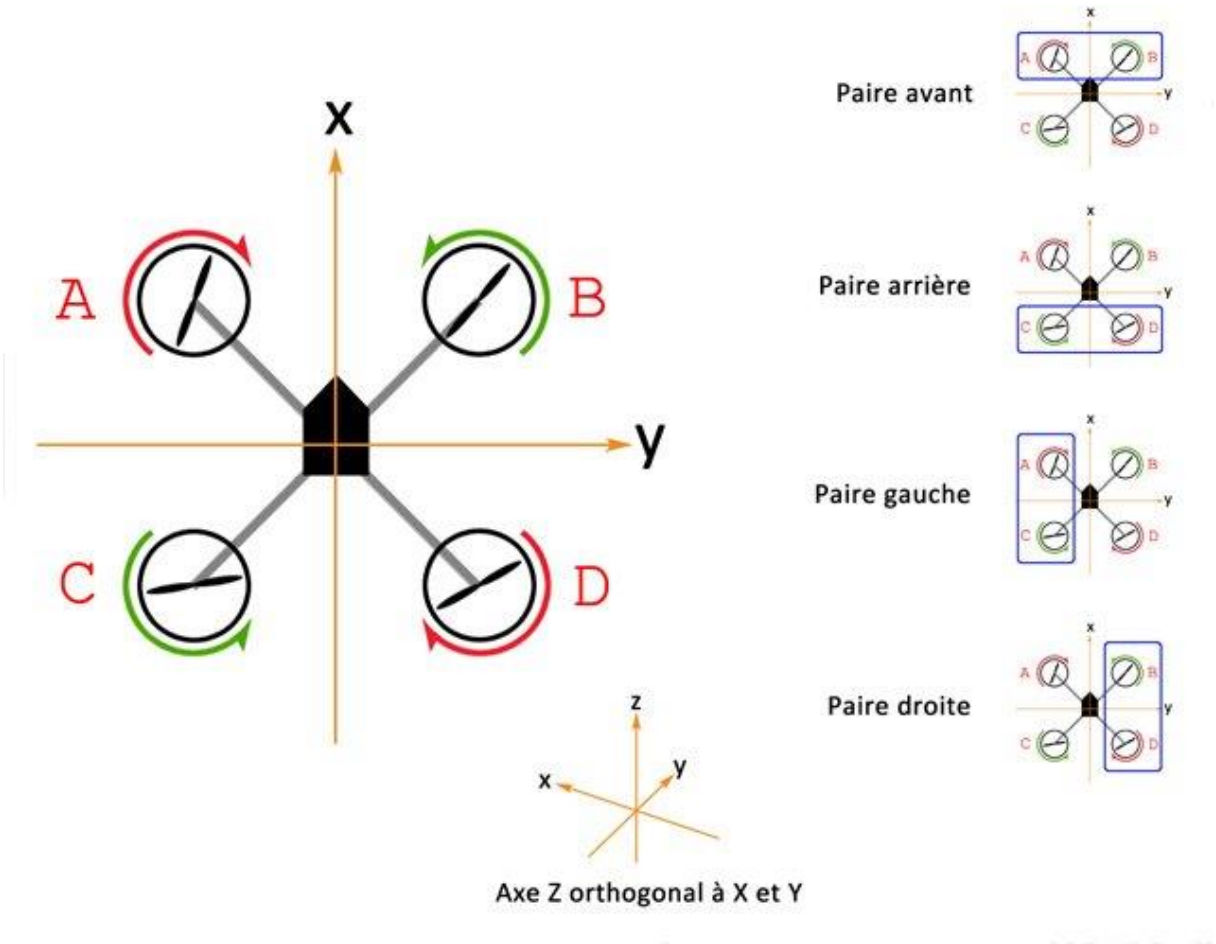

Figure 3 : Schéma du drone

D'après la troisième loi de Newton, toute masse soumise à une force (ou action) oppose à celle-ci une force qui lui égale et de sens opposé (encore appelée réaction). Les hélices du drone n'y échappent pas. En clair, une hélice qui tourne génère une force opposée à son sens de rotation.

Maintenant si toutes les hélices tournent dans le même sens, les forces de réaction se cumulent et le quadricoptère va se mettre à tourner sur lui-même.

En modifiant les sens de rotation de deux hélices et lorsque qu'elles tournent à la même vitesse, les forces de réaction se compensent et le drone garde son orientation.

I.2. Mécanique des déplacements

Pour s'orienter dans les airs le drone possède trois axes de mouvements :

- Axe X : axe de **roulis** (*Roll* en anglais)
- Axe Y : axe de **tangage** (*Pitch* en anglais)
- Axe Z : axe de **lacet** (*Yaw* en anglais)

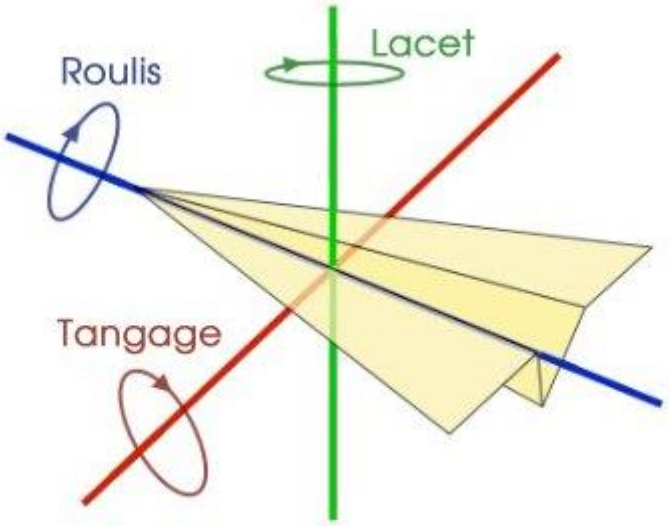

Figure 4 : Yaw, Pitch, Roll

Pour orienter le drone, il faut le faire s'incliner pour qu'il puisse se propulser dans une direction. En l'inclinant dans une direction cela va générer une force latérale qui va le déplacer dans la direction souhaitée.

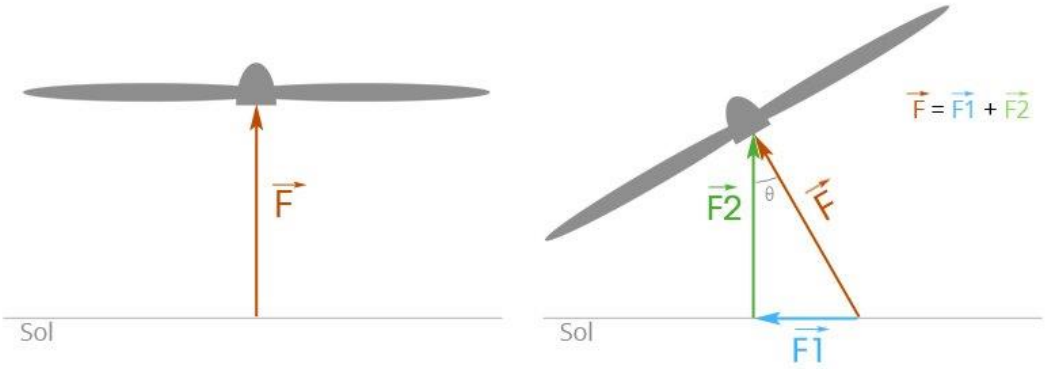

Figure 5 : Sommes des forces du drone

Donc, pour déplacer géographiquement le drone on doit jouer sur la vitesse de rotation des rotors pour qu'il s'incline dans une direction donnée.

Si l'on résume tous les mouvements possibles et en partant de la figure 3, on obtient le tableau suivant :

Déplacement		**Actions moteurs**
Monter	Translation sur l'axe de lacet	**Augmenter** la vitesse de rotation de tous les moteurs de la même manière
Descendre	Translation sur l'axe de lacet	**Diminuer** la vitesse de rotation de tous les moteurs de la même manière
Avancer	Rotation sur l'axe de tangage	**Diminuer** la vitesse de rotation des moteurs de la paire avant et **augmenter** la vitesse de rotation des moteurs de la paire arrière
Reculer	Rotation sur l'axe de tangage	**Augmenter** la vitesse de rotation des moteurs de la paire avant et **diminuer** la vitesse des moteurs de rotation de la paire arrière
Déplacement latéral gauche	Rotation sur l'axe de roulis	**Diminuer** la vitesse de rotation des moteurs de la paire gauche et **augmenter** la vitesse de rotation des moteurs de la paire de droite
Déplacement latéral droite	Rotation sur l'axe de roulis	**Augmenter** la vitesse de rotation des moteurs de la paire gauche et **diminuer** la vitesse de rotation des moteurs de la paire de droite
Rotation horaire	Rotation sur l'axe de lacet	**Diminuer** la vitesse de rotation des moteurs B/C et **augmenter** la vitesse de rotation des moteurs A/D (Voir figure 3)
Rotation anti-horaire	Rotation sur l'axe de lacet	**Augmenter** la vitesse de rotation des moteurs B/C et **diminuer** la vitesse de rotation des moteurs A/D (Voir figure 3)

Tableau 1 : Correspondance mouvements/propulsions

I.3. Boucle d'asservissement d'un drone

En automatique, un asservissement est une fonction d'un système dont l'objectif principal est d'atteindre une consigne et de s'y maintenir, peu importe les sollicitations extérieures. Pour cela il faut comparer la sortie et la consigne du système de manière à le corriger efficacement.

Ici, on récupère les informations d'un Gyroscope MPU6050 qui donne l'inclinaison et l'accélération du drone en temps réel qui sert à calculer la correction et la vitesse de rotation à appliquer sur les moteurs.

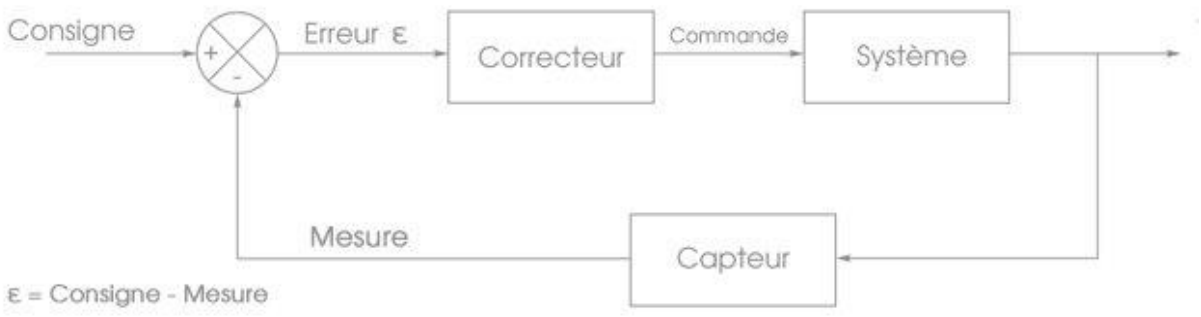

Figure 6 : Asservissement en boucle fermée

II. Anatomie d'un drone

Dans cette partie, les différentes pièces abordées dans le projet seront présentées successivement.

II.1. Moteurs brushless

Les moteurs brushless sont des machines électriques de la catégorie des machines synchrones qui fonctionnent en triphasés. Le rotor est constitué d'un ou plusieurs aimants permanents.

Ces moteurs, contrairement aux moteurs à courant continu, n'ont pas les mêmes inconvénients tels que les problèmes de commutation au niveau du collecteur, l'inertie, le refroidissement (les pertes joules sont principalement au stator donc plus faciles à évacuer), la puissance massique nettement plus grande, la durée de vie etc.

Ici les moteurs brushless étaient déjà fournis, il s'agit de moteurs de la marque Emax dont la référence est : MT2204-2300KV.

- Le premier nombre 2204 donne les dimensions du stator : 22mm de diamètre et 4mm de hauteur.
- La deuxième information 2300KV signifie 2300 tours/minute alimenté sous 1V.
- Ces moteurs fonctionnent sous 11.1V et absorbent un courant maximum de 11.5A.

Tableau 2 : moteur brushless

II.2. ESC

II.2.1. Analyse d'un ESC

Les ESC autrement dit Electronic Speed Controller sont des contrôleurs de vitesse qui régulent la vitesse de rotation de moteur.

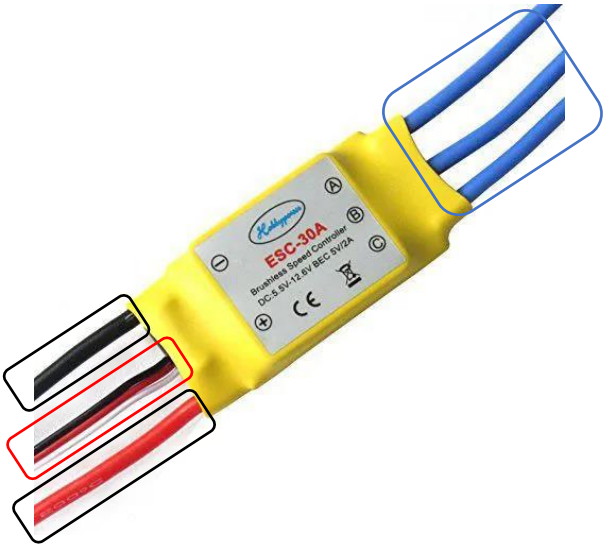

Figure 7 : ESC

Les ESC possèdent un connecteur de commande 3 broches, ici entouré en rouge, sur lequel on retrouve :

- La masse, ici le fil noir : à connecter avec la masse commune du système
- Le signal de commande, ici le fil blanc : c'est ici que l'on envoie le signal de commande de l'ESC
- L'alimentation (facultative), ici le fil rouge : elle fournit une tension +5V qui sert à alimenter d'autres composants comme par exemple une carte Arduino.

De ce même côté, il y a aussi deux gros fils, ici entouré en noir, ce sont les fils d'alimentation de l'ESC à brancher directement à l'alimentation du drone.

De l'autre côté, il y a trois fils de même couleur qui sont à brancher sur les 3 phases des moteurs, ici entourés en bleu.

II.2.2. Fonctionnement d'un ESC

Les ESC se commandent comme des servomoteurs c'est-à-dire qu'ils n'acceptent que des signaux PWM en entrée. Cependant pour les ESC c'est un signal PWM un peu particulier qu'il faut envoyer.

La particularité du signal est que les ESC ne prennent pas en compte le rapport cyclique dans le signal d'entrée mais seulement la largeur de l'impulsion à l'état haut, qui varie entre 1ms et 2ms.

Sur le schéma suivant, le signal varie entre 1ms et 2ms pour une période de 20ms (50hz).

De plus, puisque les ESC se commandent comme des servomoteurs, il faut leur donner un angle pour générer le bon PWM. Une durée de 1ms (durée minimale) correspond à un angle de -90° et une durée de 2ms (durée maximale) correspond à un angle de +90°, l'amplitude est donc de 180°.

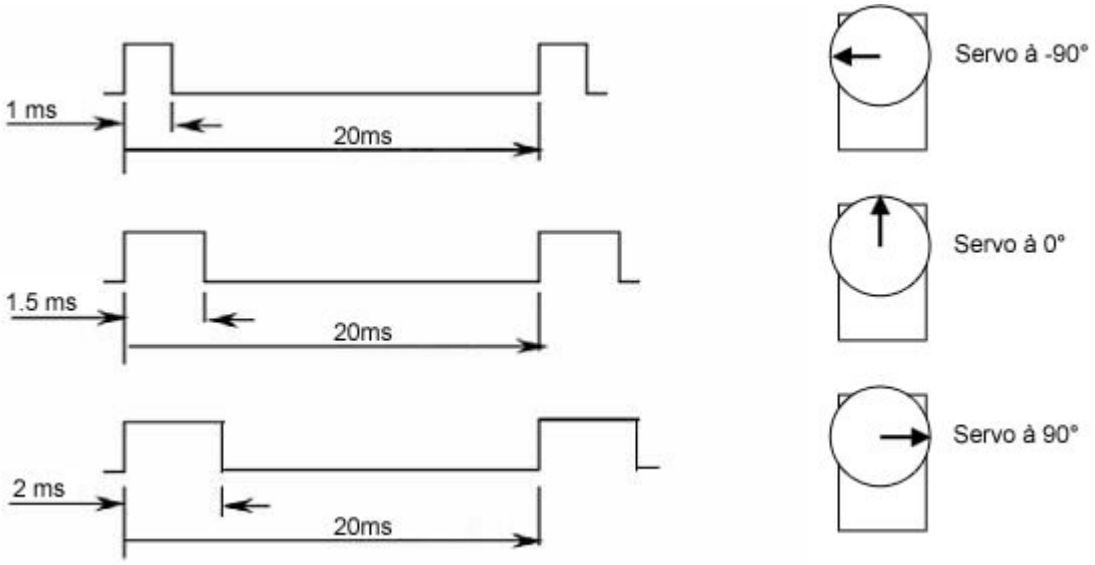

Figure 8 : Différents signaux PWM

Donc pour les ESC, un angle de 0° (largeur de 1ms) stoppera les moteurs et un angle de 180° (largeur de 2ms) les fera tourner à plein régime.

II.3. Capteurs

Le drone a besoin de capteur pour se stabiliser en vol et se déplacer géographiquement en l'air.

Ici c'est un gyroscope MPU6050 de la marque InvenSense qui remplit ce rôle. Un gyroscope est un capteur capable de mesurer l'inclinaison et l'accélération d'un objet sur lequel il est fixé.

Le gyroscope va mesurer en temps réel l'inclinaison du drone ainsi que son accélération sur les 3 axes de dimension à savoir, X, Y, Z.

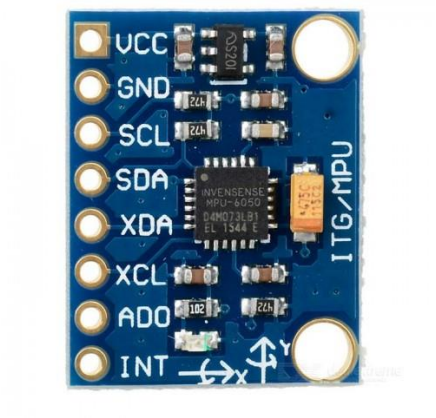

Figure 9 : Capteur MPU6050

En lisant la datasheet du capteur, on constate :

- La tension d'alimentation doit être entre 2.375V et 3.45V
- L'utilisation des interruptions
- La communication via le bus I^2C
- Une précision allant jusqu'à ±2000°/sec pour le gyroscope
- Une consommation nominale de 3.6mA pour le gyroscope
- Une précision allant jusqu'à ±16g pour l'accéléromètre
- Une consommation nominale de 500µA pour l'accéléromètre

II.4. Module de communication sans fil

Deux solutions ont été explorées pour commander le drone à distance, un module Bluetooth et une télécommande RF avec son récepteur, qui seront abordées successivement.

II.4.1. Module Bluetooth

Pour cette première solution, un module Bluetooth Arduino HC-04 a été sélectionné et une interface graphique communiquant avec une carte Arduino a été développée sur le site « Remote control Arduino – RemoteXY ».

Le traitement des données est réalisé par la bibliothèque "RemoteXY", qui traite et qui envoie les données sur l'application mobile capable d'afficher une interface. Le site web, quant à lui, génère un code à intégrer dans le programme et à stocker sur la carte Arduino.

Du côté téléphone, une phase d'initialisation a lieu permettant à l'application de se connecter au module Bluetooth. Il est ensuite possible d'interagir avec cette interface graphique qui évolue grâce aux données reçues.

La programmation du module se fait simplement par l'adressage de variables sur des ports de sortie que liera en serial la bibliothèque, avant de l'envoyer en UART sur le module Bluetooth.

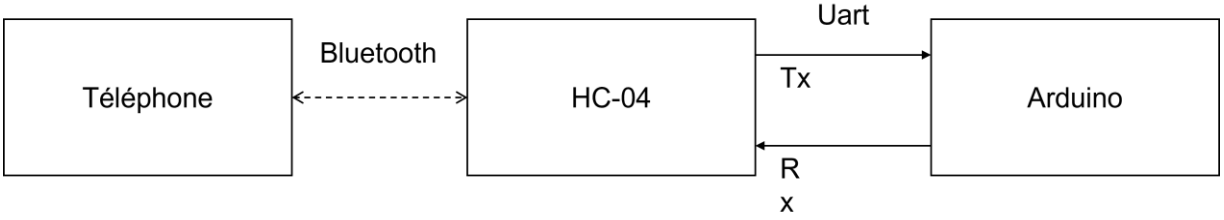

Figure 10 : Communication Bluetooth Arduino

La communication UART avec le module HC-04 est faite via deux fils de données :

- Le TX servant à la communication Module → Arduino
- Le RX servant à la communication Arduino → Module

II.4.2. Radio-télécommande

Pour cette seconde solution, une radio-télécommande Reely-HT-6 avec son récepteur ont été sélectionnés. Cette radio-télécommande possède 6 canaux de communication, 4 switch, un écran LCD et des boutons permettant de naviguer dans les menus pour configurer la télécommande.

De plus, elle possède 4 modes de fonctionnement pour personnaliser les commandes pour chaque utilisateurs, comme le montre cette figure :

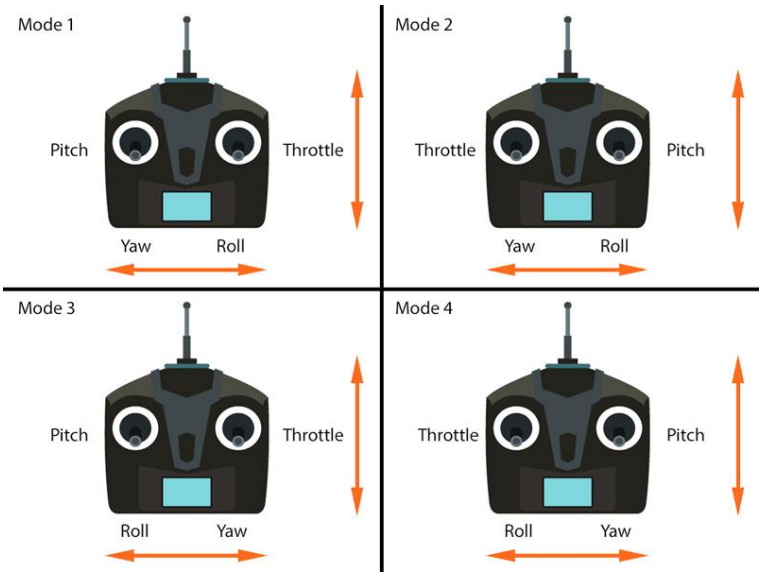

Figure 11 : Mode de configuration

Les switchs permettent de basculer d'une configuration préenregistrée à une autre.

La visualisation des commandes de chaque canal peut se faire sous la forme d'histogrammes dans un menu dédié, ce qui est pratique pour identifier l'association stick/canal.

Figure 12 : Visualisation des canaux

II.5. Ordinateur de bord

L'ordinateur de bord est la partie la plus importante du drone. Il va collecter les informations des capteurs, recevoir les instructions de vol via le module de communication, calculer la vitesse de rotation de chaque moteurs et enfin envoyer les consignes de vol aux ESC pour atteindre l'objectif de vol. De plus, il va aussi intégrer la boucle d'asservissement, vue précédemment [I.3], pour atteindre la stabilité en vol.

Tout ce ceci peut être résumé dans le schéma fonctionnel suivant :

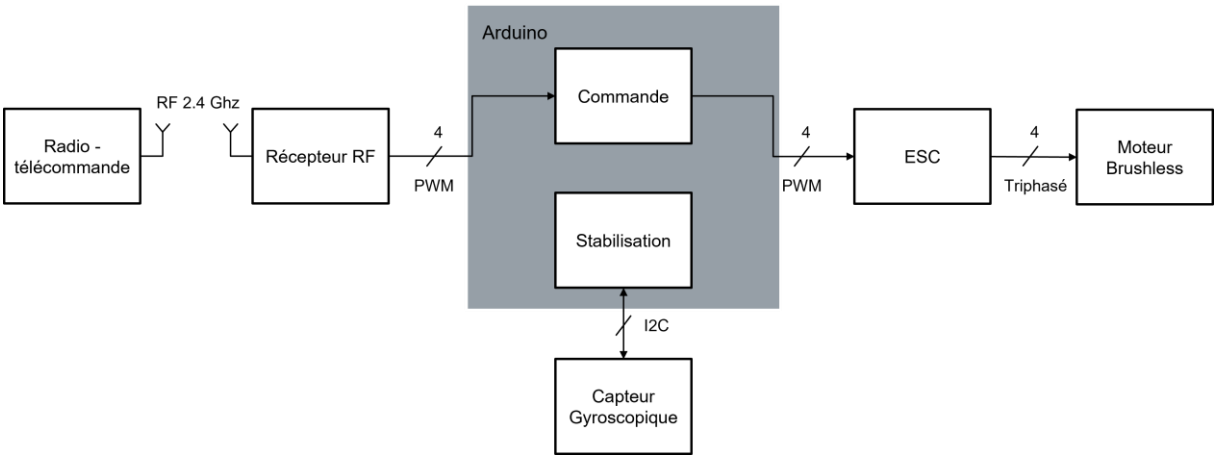

Figure 13 : Schéma fonctionnel du drone

Une carte Arduino Uno, a été sélectionnée pour servir d'ordinateur de bord, celle-ci va intégrer tout le code du drone.

III. Réalisation du drone

III.1. Utilisation et calibration des ESC

Après avoir appris le fonctionnement des ESC [II.2.2], il faut les calibrer pour pouvoir faire tourner les moteurs du drone à notre convenance.

Cette étape de calibration est très importante, il faut l'effectuer lors de chaque test et à chaque démarrage du drone.

Concrètement, la calibration n'a rien de compliqué : il s'agit d'envoyer aux ESC les extremums des commandes de gaz soit, le signal de commande « arrêt moteur » et le signal de commande « plein gaz ». Ainsi les ESC connaissent leur plage de fonctionnement et la limite minimale et maximale à ne pas dépasser.

III.1.1. Bibliothèque Servo de Arduino

Grâce à Arduino, il existe déjà une bibliothèque permettant de générer facilement des signaux PWM : la bibliothèque Servo.

Exemple d'utilisation de la bibliothèque servo :

```
#include <Servo.h>

// Déclaration d'un servo.
Servo myservo;

void setup()
{
  // On attache le servo au pin #9 de l'arduino.
  myservo.attach(9);

  // Et on écrit la valeur 90, ce qui correspond à un angle 0°.
  myservo.write(90);
}

void loop() {}
```

Figure 14 : Exemple utilisation de la bibliothèque Servo

La fonction qui permet d'envoyer la valeur souhaitée aux ESC est la fonction « write() », elle accepte en entrée une valeur comprise entre 0 et 180

Pour les ESC qui sont utilisés dans le drone, une valeur de 0 stoppe la rotation des moteurs alors qu'une valeur de 180 les fait tourner à plein régime.

III.1.2. Câblage et calibration des ESC

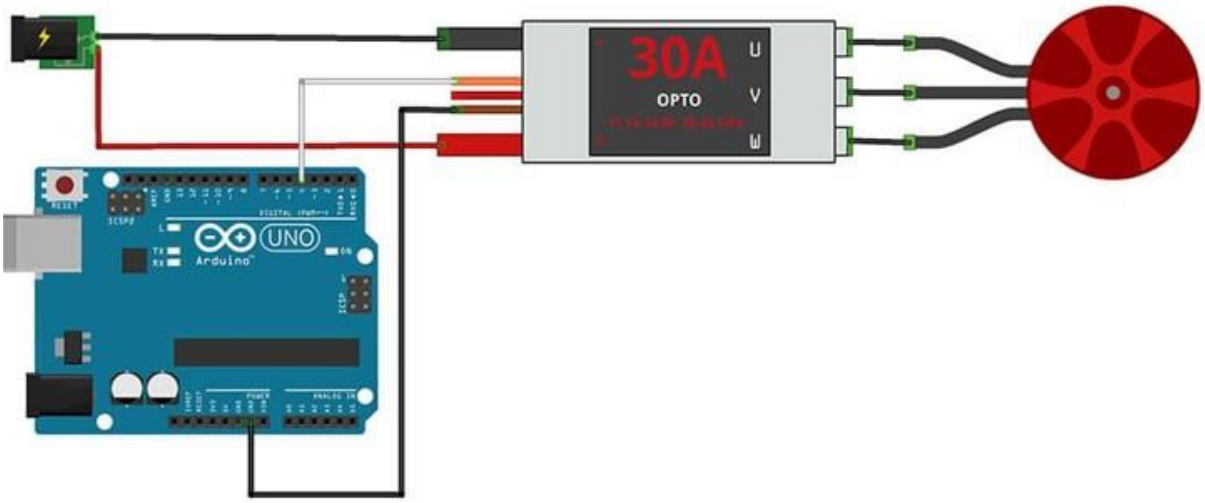

Figure 15 : Schéma câblage ESC

Pour calibrer les ESC plusieurs étapes sont nécessaires :

1. Sans que les ESC ne soient encore alimentés, connectez l'Arduino au PC grâce au câble USB. Avec Arduino software, envoyer la commande « plein gaz » (write(180)) aux ESC.

2. Alimenter les ESC, 3 bips « bip1 bip2 bip 3 » doivent se faire entendre signifiant que l'alimentation fonctionne.

3. Après 2 secondes, 2 bips « bip bip » se font entendre signifiant que la position « plein gaz » (valeur 180) a bien été enregistrée par les ESC.

4. Avec Arduino Software envoyer la commande « arrêt moteur » (write(0)) aux ESC.

5. Plusieurs bips sont émis correspondant au nombre de cellules de l'alimentation du drone.

Ceci fait, les ESC sont maintenant calibrés pour l'Arduino et prêts à l'emploi.

III.1.3. Photo des tests

Voici quelques photos des mesures du signal PWM des ESC réalisés à l'oscilloscope :

Figure 16 : Signal PWM avec état haut 1ms "arrêt moteur"

Figure 17 : Signal PWM avec état haut 1.5ms "mi-puissance"

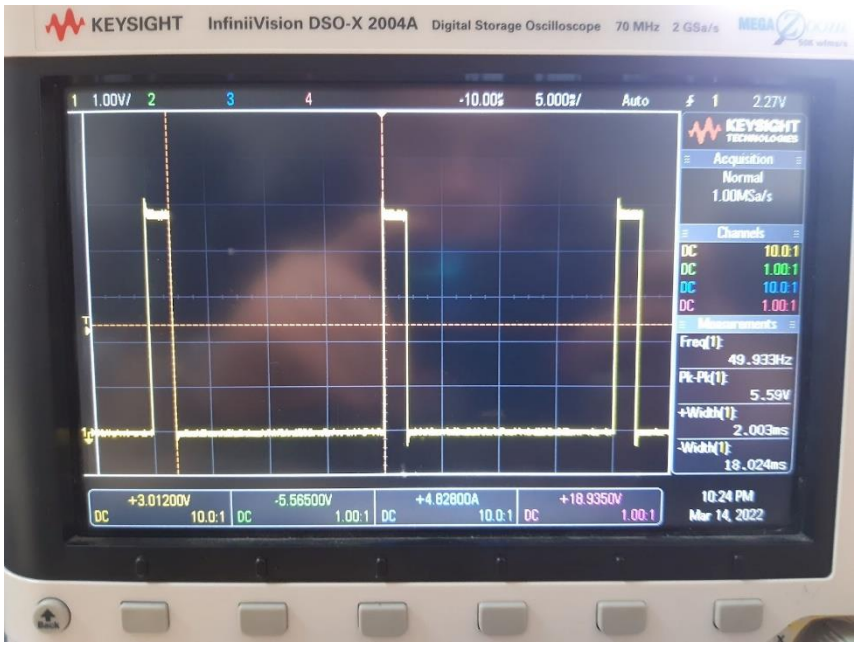

Figure 18 : Signal PWM avec état haut 2ms "pleins gaz

III.2. Calibration/exploitation du capteur gyroscopique MPU6050

III.2.1. Câblage

Le capteur MPU6050 se sert du protocole de communication I^2C dont la liaison est effectuée par l'intermédiaire de deux voies :

- SDA (Serial Data Line) : voie de données bidirectionnelle
- SCL (Serial Clock Line) : voie d'horloge de synchronisation bidirectionnelle

Grâce à Arduino, il existe une bibliothèque dédiée à ce protocole : Wire. Dans ceci, on apprend que sur l'Arduino Uno ce sont les entrées analogiques A4 et A5 qui coïncident avec les voies SDA et SCL.

Le MPU6050 a besoin d'une tension d'alimentation de 3.5V. Il faut donc relier Vcc à la bronche 3.5V de l'Arduino Uno et le GND sur le GND de l'Arduino.

Ce qui donne le schéma de câblage suivant :

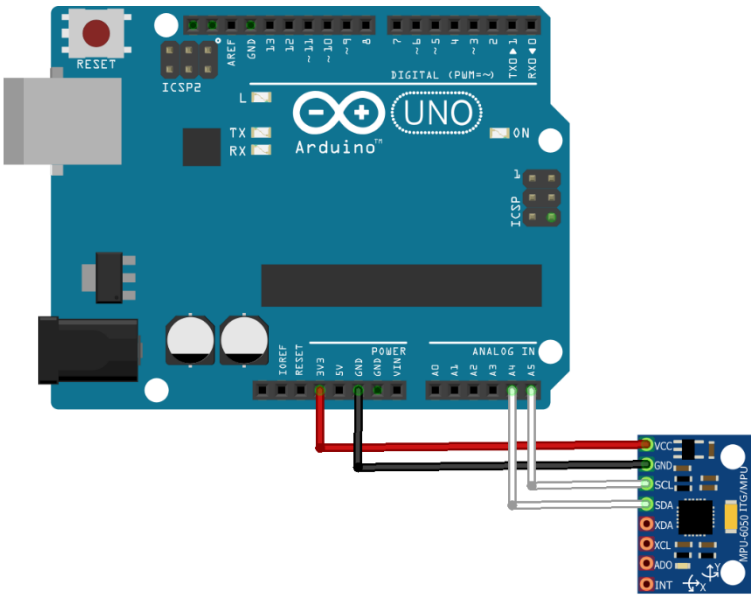

Figure 19 : Câblage du capteur MPU6050 avec Arduino Uno

III.2.2. Configuration du capteur MPU6050

III.2.2.1. Sensibilité du gyroscope

Le MPU6050 dispose de son propre processeur pour calculer et mesurer l'angle d'inclinaison et l'accélération du drone. En cherchant dans la datasheet des registres, on peut voir à la page 14 la description du registre 27 « GYRO_CONFIG ». Le gyroscope peut être utilisé dans différentes plages de sensibilité en fonction de la valeur de FS_SEL :

FS_SEL	Full Scale Range
0	± 250 °/s
1	± 500 °/s
2	± 1000 °/s
3	± 2000 °/s

Figure 20 : Sensibilité du gyroscope

Une plage de ±500°/sec a été sélectionnée car elle est amplement suffisante, ce qui donne à FS_SEL la valeur de 1.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-	-	-

Figure 21 : Octet du registre GYRO_CONFIG

FS_SEL est codé sur les bits 3 et 4, correspondant à « 01 » en binaire. Les bits 0, 1 et 2 sont réservés, on laisse à 0. Les bits 5, 6 et 7 servent à faire un autotest des X, Y et Z, on les laisse à 0.

Au final la valeur de l'octet est « 00001000 » en binaire et « 0x08 » en hexadécimal.

Enfin l'adresse du registre est « 0x1B » en hexadécimal.

```
1 Wire.beginTransmission(0x68); // Start communication with MPU
2 Wire.write(0x1B);             // Request the GYRO_CONFIG register
3 Wire.write(0x08);             // Apply the desired configuration to the register : +-500°/s
4 Wire.endTransmission();       // End the transmission
```

Figure 22 : Configuration registre GYRO_CONFIG

III.2.2.2. Sensibilité de l'accéléromètre

La méthode de configuration de la sensibilité de l'accéléromètre est la même que celle utilisée pour la sensibilité du gyroscope : le registre ACCEL_CONFIG permet de configurer cette plage de sensibilité.

AFS_SEL	Full Scale Range
0	± 2*g*
1	± 4*g*
2	± 8*g*
3	± 16*g*

Figure 23 : Sensibilités de l'accéléromètre

Une plage de ±8g est suffisante, ce qui donne une valeur à AFS_SEL de 2.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-		

Figure 24 : Octet du registre ACCEL_CONFIG

AFS_SEL est codé sur les bits 3 et 4, correspondant à « 10 » en binaire. Les bits 0, 1 et 2 sont réservés, on les laisse donc à 0. Les bits 5, 6 et 7 servent à effectuer un autotest des axes X, Y et Z, on les laisse à 0.

Au final la valeur de l'octet est « 00010000 » en binaire et « 0x10 » en hexadécimal.

Enfin l'adresse du registre ACCEL_CONFIG est « 0x1C » en hexadécimal.

```
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
```

Figure 25 : Configuration registre ACCEL_CONFIG

III.2.2.3. Fréquence d'horloge

La page 40 de la datasheet correspond au registre « Power management » permettant de choisir la source d'horloge que le capteur va utiliser.

CLKSEL	Clock Source
0	Internal 8MHz oscillator
1	PLL with X axis gyroscope reference
2	PLL with Y axis gyroscope reference
3	PLL with Z axis gyroscope reference
4	PLL with external 32.768kHz reference
5	PLL with external 19.2MHz reference
6	Reserved
7	Stops the clock and keeps the timing generator in reset

Figure 26 : Source horloge

Une horloge interne de 8Mhz a été sélectionnée, soit un CLKSEL = 0.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE _RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

Figure 27 : Octet registre Power Management

Comme précédemment vu, on touche uniquement aux bits qui doivent être modifiés, c'est-à-dire ici les bits 0, 1 et 2. La valeur « 0 » en binaire s'écrit alors « 000 ».

Les autres bits ne servent pas, donc on ne les touche pas. Au final, on obtient une valeur pour le registre de « 00000000 » en binaire, soit « 0x00 » en hexadécimal.

Enfin, l'adresse du registre est « 0x6B » en hexadécimal.

```
1 Wire.beginTransmission(0x68); // Start communication with MPU
2 Wire.write(0x6B);             // Request the PWR_MGMT_1 register
3 Wire.write(0x00);             // Apply the desired configuration to the register
4 Wire.endTransmission();       // End the transmission
```

Figure 28 : Configuration registre Power Management

III.2.2.4. Filtre passe-bas

Un dernier registre « CONFIG » va permettre de configurer la fréquence de coupure du filtre passe-bas du gyroscope et de l'accéléromètre au travers du paramètre DLPF_CFG.

DLPF_CFG	Accelerometer (F_s = 1kHz)		Gyroscope		
	Bandwidth (Hz)	Delay (ms)	Bandwidth (Hz)	Delay (ms)	Fs (kHz)
0	260	0	256	0.98	8
1	184	2.0	188	1.9	1
2	94	3.0	98	2.8	1
3	44	4.9	42	4.8	1
4	21	8.5	20	8.3	1
5	10	13.8	10	13.4	1
6	5	19.0	5	18.6	1
7	RESERVED		RESERVED		8

Figure 29 : Coupure filtre passe-bas

Une fréquence d'environ ~43Hz a été sélectionnée, ce qui donne au paramètre DLPF_CFG une valeur de 3.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1A	26	-	-	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		

Figure 30 : Octet registre CONFIG

DLPF_CFG est codé sur les bits 0, 1 et 2. La valeur « 3 » en binaire s'écrit « 011 ». On laisse comme toujours les autres bits à 0, tout ceci donne une valeur pour l'octet de « 00000011 » en binaire, soit « 0x03 » en hexadécimal.

Enfin, l'adresse du registre est « 0x1A » en hexadécimal.

```
1 Wire.beginTransmission(0x68); // Start communication with MPU
2 Wire.write(0x1A);             // Request the CONFIG register
3 Wire.write(0x03);             // Apply the desired configuration to the register : DLPF about 43Hz
4 Wire.endTransmission();       // End the transmission
```

Figure 31 : Configuration registre CONFIG

III.2.2.5. Configuration complète

Toutes ces configurations de registres ont été regroupées dans un seul sous-programme pour faciliter son utilisation et pour avoir un code propre.

III.2.3. Lire les données brutes du MPU6050

Le MPU6050 est câblé et configuré, il faut maintenant lire les données brutes du capteur. Depuis la datasheet des registres, on peut lire que les registres 59 à 72 stockent les valeurs de sortie de l'accéléromètre et du gyroscope et qu'ils commencent à l'adresse « 0x3B ».

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT[7:0]							
41	65	TEMP_OUT_H	R	TEMP_OUT[15:8]							
42	66	TEMP_OUT_L	R	TEMP_OUT[7:0]							
43	67	GYRO_XOUT_H	R	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT_L	R	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT_H	R	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT_L	R	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT[7:0]							

Figure 32 : Registres de sorties

Pour récupérer les valeurs brutes du capteur, on va simplement requêter le contenu de ces 14 registres, en commençant à l'adresse « 0x3B ».

```
#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
}
```

Figure 33 : Valeurs brutes MPU6050

III.2.4. Mesure angulaire avec le gyroscope

III.2.4.1. Conversion des données brutes

Pour pouvoir interpréter les données brutes du MPU6050 il faut les convertir. On a paramétré la sensibilité du gyroscope FS_SEL=1 pour ±500°/sec. On voit que dans la datasheet que la valeur brute retournée est « 65.5 » lorsque la vitesse angulaire est de 1°/sec.

FS_SEL	Full Scale Range	LSB Sensitivity
0	± 250 °/s	131 LSB/°/s
1	± 500 °/s	65.5 LSB/°/s
2	± 1000 °/s	32.8 LSB/°/s
3	± 2000 °/s	16.4 LSB/°/s

Figure 34 : Conversion valeurs brutes

Pour obtenir la valeur réelle il suffit de diviser la valeur brute par 65.5.

Exemple :

Si le drone fait deux tours complets en 1 minute, cela veut dire qu'il fait 720° en 60s, soit 12°/s en moyenne. La valeur en sortie du gyroscope pour cette vitesse angulaire est de 786. Si on divise cette valeur brute par 65.5 :

$$\frac{720}{65.5} = 12°/s$$

La moyenne de 12°/s est bien retrouvée.

Maintenant si le drone tourne à une vitesse maximum angulaire de 500°/s (ce qui est la vitesse maximale que l'on a configurée). En sortie, la valeur brute retournée est de :

$$500 * 65.5 = 32750$$

Cette valeur coïncide pratiquement à la valeur maximum d'un « integer » (32767), donc on pourra stocker les valeurs brutes du gyroscope dans des variables de type « integer ».

On se retrouve au final, après la conversion des valeurs brutes, une vitesse angulaire. Il suffit maintenant de faire une seconde conversion pour obtenir la position angulaire du drone en temps réel.

III.2.4.2. Conversion en degré

Pour obtenir la position angulaire à partir de la vitesse angulaire, il faut « intégrer ». Intégrer, c'est le fait de sommer des échantillons à intervalles réguliers. La vitesse angulaire est la dérivée de la position angulaire par rapport au temps. Il est donc normal qu'en intégrant la vitesse on retrouve la position angulaire.

Maintenant pour avoir un asservissement efficace et avoir une mesure de la position angulaire assez régulière, on prend une fréquence de 250Hz. Cette période de 4ms permet d'intégrer toutes les 4ms et donc d'avoir une mesure de la position angulaire toutes les 4ms.

En langage mathématique, ça se traduit comme ceci :

$$\sum_{i=1}^{n} \frac{raw_i}{SSF * 250}$$

- raw_i = valeurs brutes du gyroscope
- SSF = sensibilité du gyroscope (65.5)
- 250 = fréquence des mesures du gyroscope

III.2.4.3. Dérive du gyroscope

Après avoir obtenu la position angulaire du capteur pour le drone, il faut corriger un autre problème récurrent du gyroscope que l'on appelle la « dérive ».

Lorsque le capteur ne bouge pas et après quelques instants , les valeurs mesurées dérivent et s'éloignent de leur valeur réelle. C'est un phénomène courant et connu des gyroscopes. Les vibration des moteurs vont amplifier ce phénomène et on se retrouve vite avec des valeurs incohérentes avec la réalité.

C'est ici qu'intervient l'accéléromètre qui permet de corriger ce problème.

III.2.5. Mesure angulaires avec l'accéléromètre

Un accéléromètre est un capteur qui mesure l'accélération subite et l'exprime en g. Sur terre 1g représente l'accélération gravitationnelle de la terre soit environ 9,8m.s^{-2}.

Donc au sol à l'arrêt, le capteur mesure une accélération de 1g.

L'accéléromètre mesure l'accélération subite sur chaque axe. Le vecteur gravité, ici en vert, de la masse jaune est la somme des deux vecteurs X et Y :

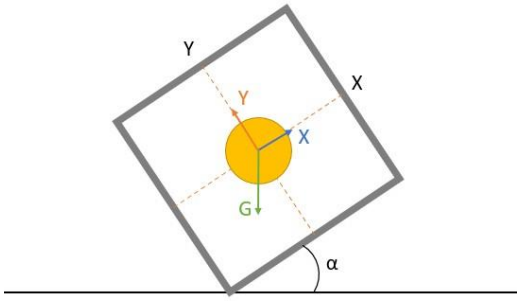

Figure 35 : Analogie fonctionnement accéléromètre

On trouve un triangle rectangle dont l'angle adjacent n'est autre que l'angle « α » que l'on cherche.

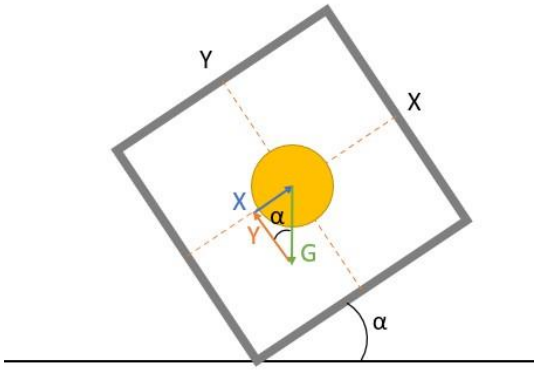

Figure 36 : Calcul angle α

On utilise le théorème de Pythagore pour calculer la norme du vecteur gravité :

$$G = \sqrt{X^2 + Y^2}$$

A partir de ce calcul , on calcule l'angle adjacent :

$$\alpha = \text{asin}\left(\frac{X}{G}\right)$$

On arrive donc à récupérer la position angulaire du capteur de façon précise grâce à l'accélération. Cependant à cause des vibrations, les mesures deviennent instables et inutilisables. Il va donc falloir renforcer les mesures en couplant les mesures du gyroscope et de l'accéléromètre.

III.2.6. Renforcement des mesures

Pour faire un récapitulatif :

- Le gyroscope est moins sensible aux vibrations que l'accéléromètre
- Les mesures du gyroscope dérivent dans le temps
- Les vibrations rendent l'accéléromètre inutilisable

Il faut prendre le meilleur de chaque instrument pour avoir des mesures exploitables.

III.2.6.1. Mesures exploitables

Pour rendre les mesures utilisables, il faut compenser la dérive du gyroscope, pour cela on utilise un peu des mesures des deux instruments :

$$Angle\ filtré = \alpha * (Gyroscope\ Angle) + (1 - \alpha) * (Accelerometer\ Angle)$$

Où α est le facteur de correction qui dépend de l'intervalle de mesure.*

```
1 pitch_angle = pitch_angle_gyro * 0.9996 + pitch_angle_acc * 0.0004;
2 roll_angle = roll_angle_gyro * 0.9996 + roll_angle_acc * 0.0004;
```

Figure 37 : Correction des mesures

On prend 99.96% de la mesure du gyroscope et 0.04% de la mesure de l'accéléromètre.

III.2.7. Conclusion mesure position angulaire

Après toutes ces étapes, le drone, grâce au capteur MPU6050, est capable de mesurer son inclinaison avec une fréquence de rafraîchissement de 250Hz.

III.3. Utilisation de la radio-télécommande avec l'Arduino

Les signaux de sortie de la radio-télécommande et de son récepteur RF sont, comme pour les ESC, des signaux de type « Servo control », donc des PWM. On ne va pas connecter directement les sorties de récepteur RF sur les ESC pour piloter indépendamment les moteurs, puisque c'est le travail de l'ordinateur de bord ici l'Arduino Uno.

Il va donc falloir décoder l'information de chaque canal du récepteur RF pour la rendre exploitable par l'ordinateur de bord.

III.3.1. Interruptions

Pour décoder le signal de sortie du récepteur RF, on utilise une fonctionnalité courante disponible sur les microcontrôleurs : les interruptions.

L'interruption est une fonctionnalité qui met en pause le programme principal pour exécuter une autre tâche (qu'on nomme aussi routine d'interruption). Lorsque cette interruption est terminée, le programme principal reprend là où il s'était arrêté.

Il existe deux types d'interruptions :

- Les interruptions provenant de périphériques externes (reset, bouton poussoir, clavier...)
- Les interruptions provenant de périphériques internes (timer, passage par zero d'un compteur interne…)

Dans le cas présent, ce sont les routines d'interruption externes que l'on va utiliser via le récepteur RF de la radio-télécommande.

La routine d'interruption doit être la plus courte possible. En effet, si sa durée d'exécution dépasse la période du signal d'interruption alors le programme principal ne sera jamais exécuté.

Il existe une fonction pour utiliser simplement les interruptions de l'Arduino, c'est la fonction « attachInterrupt » :

```
1  volatile byte state = LOW;
2
3  void setup() {
4    pinMode(2, INPUT_PULLUP);
5    attachInterrupt(digitalPinToInterrupt(2), mySubRoutine, CHANGE);
6  }
7
8  void mySubRoutine() {
9    state != state;
10 }
```

Figure 38 : Exemple routine d'interruption

Le problème avec cette fonction c'est que dans le cas de l'Arduino Uno, elle ne permet de faire que deux interruptions, sur les broches 2 et 3.

Board	Digital Pins Usable for Interrupts
Uno, Nano, Mini, other 328-based	2, 3
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21
Micro, Leonardo, other 32u4-based	0, 1, 2, 3, 7
Zero	All digital pins, except 4
MKR1000 Rev.1	0, 1, 4, 5, 6, 7, 8, 9, A1, A2
Due	All digital pins
101	All digital pins (Only pins 2, 5, 7, 8, 10, 11, 12, 13 work with CHANGE)

*Tableau 3 : *Digital Pins With Interrupts*

Cependant 4 voies à décoder sont nécessaires pour le drone. Néanmoins il est possible d'utiliser plus de 2 interruptions en modifiant les registres d'interruption de l'Arduino Uno sur la datasheet de « l'ATmega328P ».

III.3.2. Registres d'interruption

Les broches 8, 9, 10 et 11 ont été sélectionnées pour décoder les quatre voies du récepteur RF.

Sur le pin mapping de l'ATmega38P, les interruptions liées à ces broches sont les suivantes :

- Broche #8 : PCINT0
- Broche #9 : PCINT1
- Broche #10 : PCINT2
- Broche #11 : PCINT3

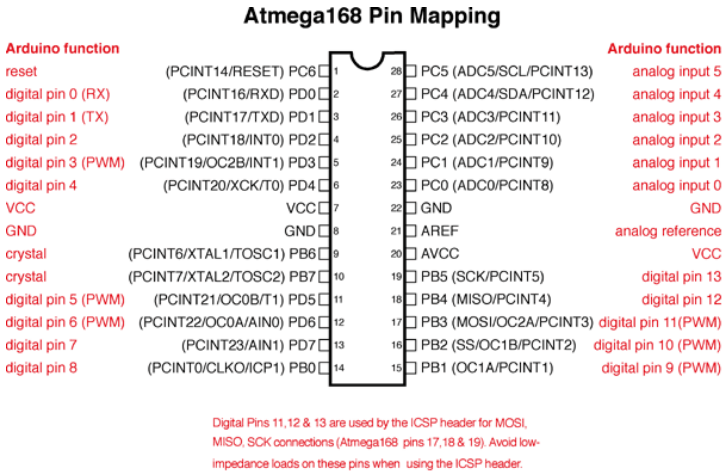

Figure 39 : Pin mapping Atmega168

Maintenant que l'on connait les noms des bronches que l'on doit configurer pour pourvoir utiliser les interruptions dessus, il faut aller dans la datasheet de l'ATmega, et plus précisément dans la section sur les « External interrupts ». Ces interruptions sont gérées par le registre PCMSK0 (Pin Change Mask Register 0).

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 40 : Registre PCMSK0

- **Bit 7..0 – PCINT7..0: Pin Change Enable Mask 7..0**

Each PCINT7..0 bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is set and the PCIE0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

Figure 41 : Utilisation du registre PCMSK0

En lisant ceci (fig41), on comprend qu'il faut faire deux choses :

- Activer le bit PCEI0 dans le registre PCICR
- Activer les bits interruptions dans le registre PCMSK0

Explication du fonctionnement du registre PCICR (pin Change Interrupt Register) :

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 42 : Registre PCICR

- **Bit 0 - PCIE0: Pin Change Interrupt Enable 0**

When the PCIE0 bit is set (one) and the I-bit in the status register (SREG) is set (one), pin change interrupt 0 is enabled. Any change on any enabled PCINT7..0 pin will cause an interrupt. The corresponding interrupt of pin change interrupt request is executed from the PCI0 interrupt vector. PCINT7..0 pins are enabled individually by the PCMSK0 register.

Figure 43 : Utilisation du registre PCICR

On compile tout ce que l'on vient de voir dans la datasheet dans un programme Arduino et cela donne ceci :

```
// Activation du bit PCIE0 dans le registrer PCICR
PCICR  |= (1 << PCIE0);

// Activation des interruptions que l'on souhaite utiliser
PCMSK0 |= (1 << PCINT0); // Activation de l'interruption PCTINT0 (broche #8)
PCMSK0 |= (1 << PCINT1); // Activation de l'interruption PCTINT1 (broche #9)
PCMSK0 |= (1 << PCINT2); // Activation de l'interruption PCTINT2 (broche #10)
PCMSK0 |= (1 << PCINT3); // Activation de l'interruption PCTINT3 (broche #11)
```

Figure 44 : Code Arduino configuration des registre

Maintenant chaque changement d'état des broches 8, 9 , 10 et 11 déclenchera une interruption.

Pour créer une routine d'interruption il faut utiliser l'instruction « ISR() » (interrupt Sub Routine) :

```
void setup() {
    PCICR  |= (1 << PCIE0);
    PCMSK0 |= (1 << PCINT0);
    PCMSK0 |= (1 << PCINT1);
    PCMSK0 |= (1 << PCINT2);
    PCMSK0 |= (1 << PCINT3);
}

ISR(PCINT0_vect) {
  // Nos instructions pour la routine d'interruption
}
```

Figure 45 : Interrupt Sub Routine

On peut enfin faire ce que l'on souhaite dans la routine d'interruption, en n'oubliant pas que le temps d'exécution de la routine doit être le plus petit possible.

III.3.3. Implémentation

III.3.3.1. Algorithme

Comme vu précédemment, c'est la durée de l'impulsion du signal PWM qui représente l'information. Le but de notre algorithme est de mesurer la durée des impulsions de chaque voie du récepteur RF. On sait que chaque signal a une durée comprise entre 1ms et 2 ms.

Pour calculer cette durée, un timer a été utilisé. On le déclenche sur chaque front montant et on l'arrête au prochain front descendant. Si on procède ainsi, on mesure bien la durée de l'impulsion.

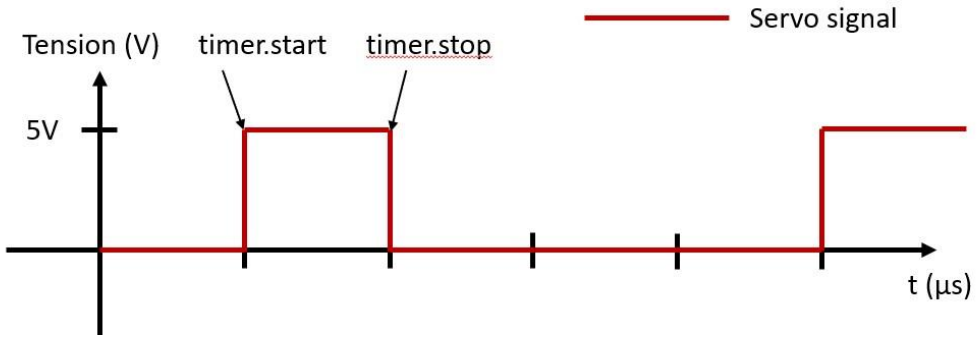

Figure 46 : Algorithme

Pour start/stop le timer, on utilise les interruptions qui sont déclenchées sur chaque front montant et descendant. Pour que le temps d'exécution de la routine d'interruption soit le plus court, on manipule directement les ports d'E/S de la carte Arduino Uno.

III.3.3.2. Exemple du principe sur une voie

Voici un exemple avec une seule voie (la bronche 8) pour bien comprendre.

Quand on se trouve dans la routine d'interruption cela signifie qu'il y a eu un changement d'état de la broche. Il faut donc déterminer s'il s'agit d'un front montant ou d'un front descendant :

- Si état = HAUT et que état précédent = BAS, alors il s'agit d'un front montant
- Sinon, si état = BAS et que état précédent = HAUT, alors il s'agit d'un front descendant

```
if (PINB & B00000001) {
    if (previous_state == LOW) {
       // Il s'agit d'un front montant
    }
} else if(previous_state == HIGH) {
    // Il s'agit d'un front descendant
}
```

Figure 47 : Détection front montant ou descendant

Il faut sauvegarder l'état actuel dans « previous_state » pour la prochaine interruption. Après si on a un front montant, on démarre le timer. Sinon dans le cas d'un front descendant, on stoppe le timer et on sauvegarde la durée mesurée :

```
current_time = micros();

if (PINB & B00000001) {
    if (previous_state == LOW) {
        previous_state = HIGH; // On sauvegarde l'état actuel
        timer = current_time;  // Et on démarre le timer
    }
} else if(previous_state == HIGH) {
    previous_state = LOW;                 // On sauvegarde l'état actuel
    pulse_duration = current_time - timer; // On calcul et on sauvegarde la durée mesurée
}
```

Figure 48 : Calcul durée impulsion

Pour finir il suffit d'appliquer la même méthode pour les autres broches et ainsi pourvoir connaitre les durées à l'état haut de chaque bronches et canaux du récepteur RF.

III.3.4. Câblage et test

Pour le câblage, il faut relier la broche signal de chaque canal à l'Arduino en respectant le câblage suivant :

- Canal 1 → Broche 8
- Canal 2 → Broche 9
- Canal 3 → Broche 10
- Canal 4 → Broche 11

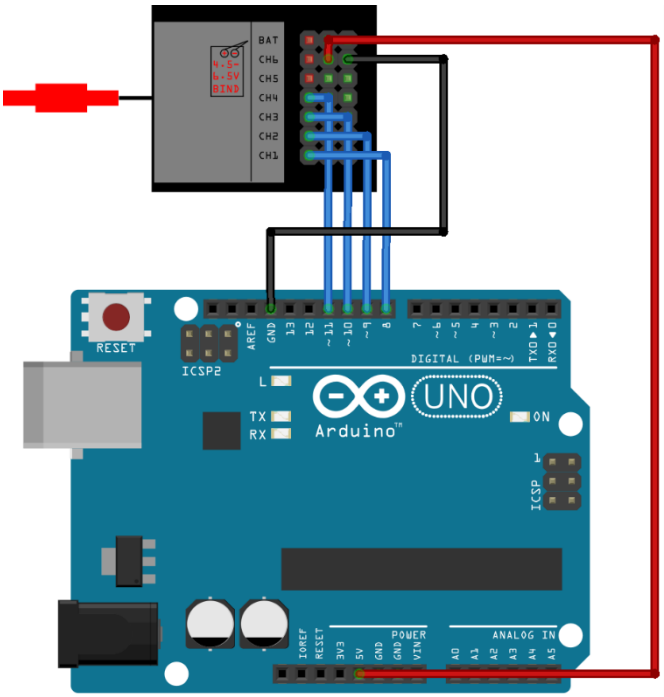

Figure 49 : Câblage Arduino recepteur RF

Lors de tests réalisés avec la radio-télécommande et son récepteur RF, il a été observé que les impulsions varient entre 1ms et 2ms sur les quatre voies des PWM.

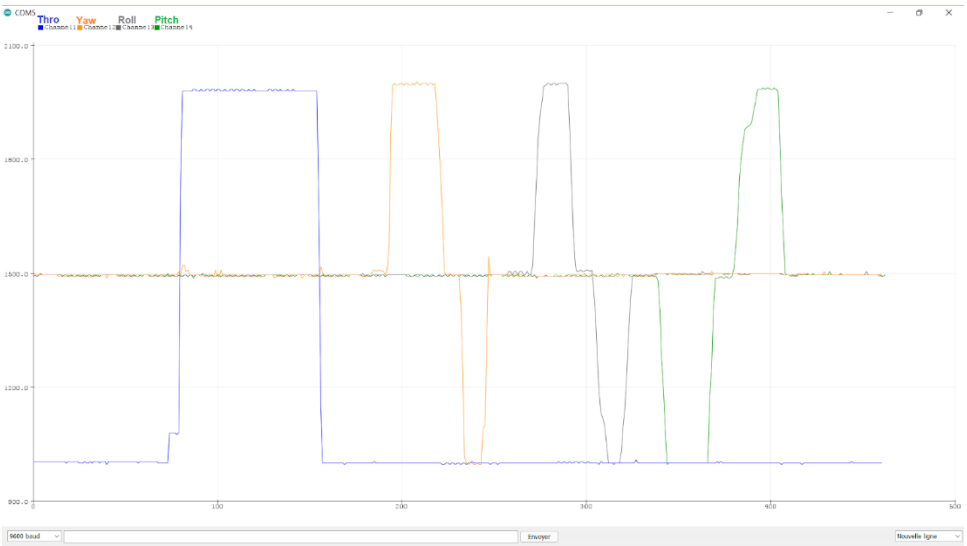

Figure 50 : Résultats des tests

III.4. Asservissement PID

Comme précédemment vu au paragraphe I.3, le drone a besoin d'une boucle d'asservissement pour pouvoir se maintenir en équilibre en l'air. Il fait donc appel à un correcteur pour corriger les erreurs d'inclinaison du drone et ajuster la vitesse de rotation des moteurs.

III.4.1. Régulateur PID

Le régulateur (ou correcteur) PID est un système de contrôle permettant d'améliorer les performances d'un asservissement, son rôle est d'appliquer une correction au système en fonction de l'erreur mesurée.

Au niveau mathématique, un régulateur PID s'écrit de la façon suivante dans le domaine temporel :

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

PID est l'acronyme de Proportionnel, Intégrale, Dérivé. Ce sont les trois paramètres du régulateur :

- Action Proportionnelle : l'erreur est multipliée par un gain K_p
- Action intégrale : l'erreur est intégrée et multipliée par un gain K_i
- Action Dérivée :l'erreur est dérivée et multipliée par un gain K_d

Si l'on rajoute le régulateur PID dans le précédent schéma vu en I.3, voici ce que l'on obtient :

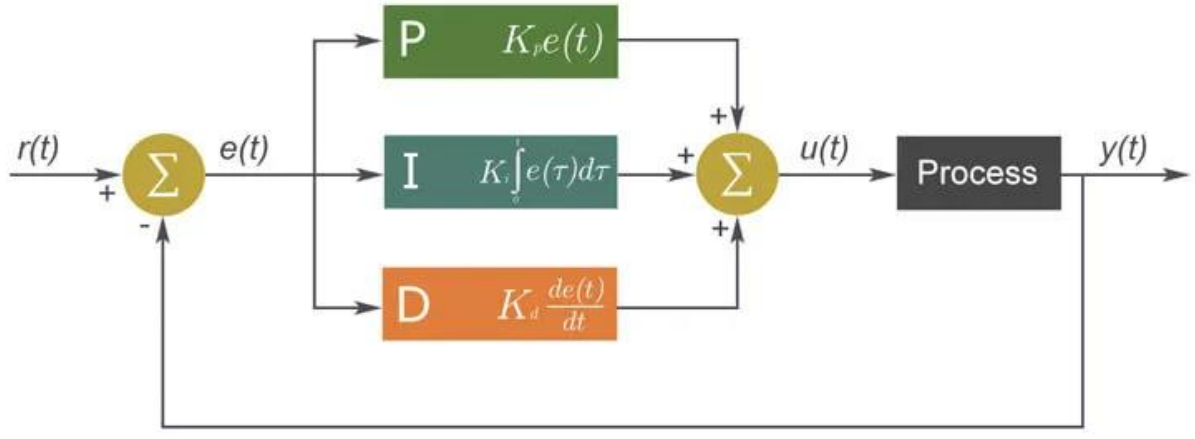

Figure 51 : Schéma bloc d'un régulateur PID

III.4.1.1. Action Proportionnelle

L'action proportionnelle, comme son nom l'indique, applique un coefficient K_P à l'erreur.

$$K_p * e(t)$$

Plus on augmente le gain et plus le système réagit vite mais il gagne en instabilité et perd en précision et donc en performance. Si l'on augmente trop le gain on se rapproche du point d'instabilité et lorsque celui-ci est dépassé le système devient instable.

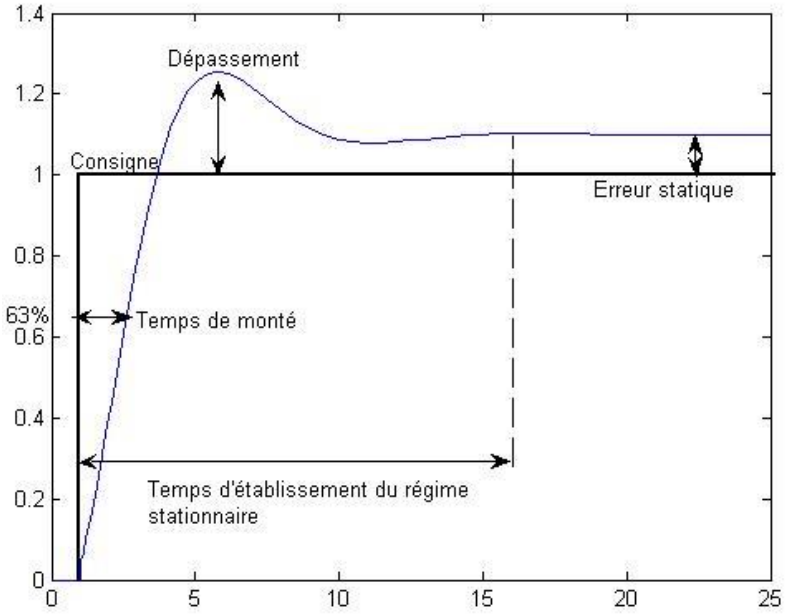

Figure 52 : Réponse d'un système à une consigne donnée

Précision	Stabilité	Rapidité
↓	↓	↑

Tableau 4 : Résumé de l'action proportionnelle

III.4.1.2. Action Intégrale

L'action intégrale consiste à sommer dans le temps les erreurs mesurées et d'appliquer un gain K_i à l'ensemble :

$$K_i \int_0^t e(t)dt$$

Ce paramètre permet de réduire l'erreur statique et donc d'améliorer la précision du système au détriment de la stabilité et de la rapidité.

Précision	Stabilité	Rapidité
↑	↓	↓

Tableau 5 : Résumé de l'action intégrale

III.4.1.3. Action Dérivée

L'action dérivée consiste à calculer la différence entre l'erreur courante et l'erreur à t-1 et à appliquer un gain K_d au résultat :

$$K_d \frac{de(t)}{dt}$$

Cette composante augmente la stabilité du système mais diminue sa rapidité.

Précision	Stabilité	Rapidité
-	↑	↓

Tableau 6 : Résumé de l'action dérivée

III.4.1.4. Action PID

Le correcteur PID est donc un mélange de ces trois actions. Si l'on fait varier correctement les trois coefficients, on peut alors corriger parfaitement l'erreur.

III.4.2. Codage Arduino du régulateur PID

Le code Arduino a beaucoup de choses à réaliser pour faire voler le drone. Voici un récapitulatif des étapes :

1. Lecture du capteurs MPU6050
2. Calcul des angles et des vitesses angulaires
3. Lecture du récepteur RF
4. Calcul des consignes
5. Calcul des erreurs
6. Calcul des sorties : régulateur PID
7. Génération des signaux de sortie

Les étapes 1, 2 et 3 ont déjà été réalisées précédemment.

III.4.2.1. Calcul des consignes

Pour que le régulateur puisse utiliser les signaux reçus, il faut les convertir dans une dimension qu'il puisse exploiter. Voici une fonction permettant cela :

```
//float angle = angle mesuré en degré sur l'axe X Y Z
//int channel_pulse = Durée en us de l'impulsion reçue pour l'axe (comprise entre 1000us et 2000us)

float calculateSetPoint(float angle, int channel_pulse) {
    float level_adjust = angle * 15;
    float set_point    = 0;

    if (channel_pulse > 1508) {
        set_point = channel_pulse - 1508;
    } else if (channel_pulse <  1492) {
        set_point = channel_pulse - 1492;
    }

    set_point -= level_adjust;
    set_point /= 3;

    return set_point;
}
```

Figure 53 : Calcul consigne

Cette fonction permet d'avoir une consigne exprimée en °/sec. Il faut appliquer ce calcul à chacun des trois axes.

La fonction ci-dessus peut être exprimée par la formule suivante :

$$set_{point} = \frac{pulse - 1508 - angle * 15}{3}$$

La valeur 15 est un coefficient servant à limiter l'inclinaison du drone. Avec une valeur 15, on obtient une inclinaison maximale de ±32.5°

III.4.2.2. Calcul des erreurs

Calcul des erreurs dont le PID a besoin :

```
void calculateErrors() {
    // Calcul des erreurs courantes
    errors[YAW]   = angular_motions[YAW]   - pid_set_points[YAW];
    errors[PITCH] = angular_motions[PITCH] - pid_set_points[PITCH];
    errors[ROLL]  = angular_motions[ROLL]  - pid_set_points[ROLL];

    // Calcul des sommes d'erreurs : composante intégrale
    error_sum[YAW]   += errors[YAW];
    error_sum[PITCH] += errors[PITCH];
    error_sum[ROLL]  += errors[ROLL];

    // on s'assure que ∫e.Ki ne dépasse pas 400
    error_sum[YAW]   = minMax(error_sum[YAW],   -400/Ki[YAW],   400/Ki[YAW]);
    error_sum[PITCH] = minMax(error_sum[PITCH], -400/Ki[PITCH], 400/Ki[PITCH]);
    error_sum[ROLL]  = minMax(error_sum[ROLL],  -400/Ki[ROLL],  400/Ki[ROLL]);

    // Calcul du delta erreur : composante dérivée
    delta_err[YAW]   = errors[YAW]   - previous_error[YAW];
    delta_err[PITCH] = errors[PITCH] - previous_error[PITCH];
    delta_err[ROLL]  = errors[ROLL]  - previous_error[ROLL];

    // L'erreur courante devient l'erreur précédente pour le prochain tour de boucle
    previous_error[YAW]   = errors[YAW];
    previous_error[PITCH] = errors[PITCH];
    previous_error[ROLL]  = errors[ROLL];
}
```

Figure 54 : Calcul des erreurs

En premier, on calcule d'abord l'erreur courant pour l'action proportionnelle. Ensuite on fait la somme de ces erreurs pour l'action intégrale. Enfin on calcule l'erreur de l'action dérivée grâce à l'erreur courante précédente.

III.4.2.3. Calcul des sorties : Régulateur PID

Le rôle du régulateur PID va être de calculer la durée de l'impulsion a envoyer à chaque ESC pour faire varier la vitesse de rotation des moteurs. Cette durée d'impulsion doit être comprise entre 1ms et 2ms.

Sur Arduino cela donne ceci :

```
// PID = e.Kp + ∫e.Ki + Δe.Kd
yaw_pid   = (errors[YAW]   * Kp[YAW])   + (error_sum[YAW]   * Ki[YAW])   + (delta_err[YAW]   * Kd[YAW]);
pitch_pid = (errors[PITCH] * Kp[PITCH]) + (error_sum[PITCH] * Ki[PITCH]) + (delta_err[PITCH] * Kd[PITCH]);
roll_pid  = (errors[ROLL]  * Kp[ROLL])  + (error_sum[ROLL]  * Ki[ROLL])  + (delta_err[ROLL]  * Kd[ROLL]);
```

Figure 55 : Calcul des sorties

Le régulateur PID a été appliqué sur les trois axes de dimensions X, Y et Z. On obtient en sortie la correction sur les trois axes.

Pour le calcul il est nécessaire de reprendre les figures 3 et 4 vues précédemment.

Si le drone penche vers la droite sur l'axe de roulis à +15°/s, son erreur par rapport à la consigne sera donc de :

$$e(t) = +15 - 0 = +15$$

Avec une erreur et des coefficients de gain Kp, Ki et Kd positifs, on déduit que :

$$roll_{pid} > 0$$

Pour équilibrer le drone, il faut ralentir les moteurs A et C et accélérer les moteurs B et D. Pour le moteur C par exemple, il faut donc lui soustraire « $roll_{pid}$ » pour diminuer la vitesse de rotation du moteur.

Roll (mesure)	Consigne	Erreur = mesure – consigne	Roll_pid	Correction à appliquer	Opération à appliquer
+15°/s	0°/s	+15°/s	>0	A & C ↓ B & D ↑	A -= roll_pid B += roll_pid C -= roll_pid D += roll_pid

Tableau 7 : Correction à appliquer axe Roll

Même logique pour l'axe Pitch (tangage) :

Pitch (mesure)	Consigne	Erreur = mesure – consigne	Pitch_pid	Correction à appliquer	Opération à appliquer
+15°/s	0°/s	+15°/s	>0	A & B ↓ C & D ↑	A -= pitch_pid B -= pitch_pid C += pitch_pid D += pitch _pid

Tableau 8 : Correction à appliquer axe Pitch

Et pour l'axe Yaw (lacet) :

Yaw (mesure)	Consigne	Erreur = mesure – consigne	Yaw_pid	Correction à appliquer	Opération à appliquer
+15°/s	0°/s	+15°/s	>0	A & D ↑ C & B ↓	A += yaw_pid B -= yaw _pid C -= yaw _pid D += yaw _pid

Tableau 9 : Correction à appliquer axe Yaw

Au final on regroupe toutes ces corrections dans quelques lignes de codes, et cela donne ceci :

```
pulse_length_esc1 = throttle - roll_pid - pitch_pid + yaw_pid;
pulse_length_esc2 = throttle + roll_pid - pitch_pid - yaw_pid;
pulse_length_esc3 = throttle - roll_pid + pitch_pid - yaw_pid;
pulse_length_esc4 = throttle + roll_pid + pitch_pid + yaw_pid;
```

Tableau 10 : Calcul durée impulsion axe Roll Pitch Yaw

III.4.2.4. Génération des signaux de sortie des ESC

On va manipuler les ports d'entrée/sortie de la carte Arduino pour des raisons de performances. L'algorithme permettant de générer les signaux de sortie pour chaque ESC est simple :

1. On passe toutes les sorties à l'état haut en même temps.
2. On attend que la durée correspondante à chaque impulsion pour chaque ESC soit écoulée pour repasser la sortie concernée à l'état bas.

En code Arduino cela donne ceci :

```
void applyMotorSpeed() {
    // Fs = 250Hz : on envoie les impulsions toutes les 4000µs
    while ((now = micros()) - loop_timer < 4000);

    // Update loop timer
    loop_timer = now;

    PORTD |= B11110000;

    // On boucle tant que toutes les sorties ne sont pas retournées à l'état LOW
    while (PORTD >= 16) {
        now = micros();
        difference = now - loop_timer;

        if (difference >= pulse_length_esc1) PORTD &= B11101111; // Passe la broche #4 à LOW
        if (difference >= pulse_length_esc2) PORTD &= B11011111; // Passe la broche #5 à LOW
        if (difference >= pulse_length_esc3) PORTD &= B10111111; // Passe la broche #6 à LOW
        if (difference >= pulse_length_esc4) PORTD &= B01111111; // Passe la broche #7 à LOW
    }
}
```

Figure 56 : Génération des signaux de sortie

III.5. Code final

Le code final du projet regroupe toutes les fonctions et tous les sous-programmes vus au cours de ce rapport.

IV. Bilan

Le travail réalisé nous a permis de nous impliquer techniquement dans de nombreux aspects jamais traités en cours, tout en nous offrant des applications pour de nombreuses notions étudiées cette année.

Malgré les problèmes rencontrés nous avons pu mettre en œuvre de nombreuses solutions. Nous n'avons pas manqué de mettre ces dernières en relation avec nos compétences techniques. Nous avons aussi pu nous organiser en se partageant les tâches et planifiant les étapes de la conception tout en faisant en sorte de suivre au mieux le cahier des charges. Nous avons également créé des petits projets, chacun répondant à un problème ou apportant une solution technique pour progresser.

Avançant de tâches en tâches, il a été possible de progresser rapidement tout en développant des solutions qui pouvaient rendre obsolète d'autres solutions mises en œuvre précédemment, comme ce fut le cas avec l'utilisation du Bluetooth remplacé par une radio-télécommande bien plus adaptée car offrant plus de commandes et une portée bien meilleure.

Cependant malgré notre implication et nos efforts pour les résoudre, certains problèmes ne peuvent être contournés et se sont avérés critique, dont un en particulier qui nous a bloqué tout au long du projet.

En effet nous n'avons pas pu mener à bien le projet dû à des problèmes sur les moteurs que nous n'avons pas pu résoudre. Ces derniers n'arrivaient vraisemblablement pas synchroniser leur champ statorique et leur champ rotorique, empêchant le drone de démarrer de façon convenable. Nous n'avons donc pas pu tester toute la partie asservissement avec le régulateur PID.

Enfin, il semble important de dire que ce projet s'est révélé enrichissant pour nous. Il nous a permis de nous rapprocher un peu plus des qualités pour devenir ingénieur, nous préparant à trouver par nous-même des solutions à un problème donné, planifier une conception, travailler en équipe, respecter un planning et appliquer un cahier des charges.