



TIPE sur le Drone

Les enjeux de la réalisation d'un drone



Table des Matières

Table des Matières	2
Introduction.....	5
I. Étude théorique du déplacement d'un quadri-rotor	7
A. Le déplacement dans un repère	7
1. Principe de combinaison des vitesses.....	7
a. Rappels : vecteur position et vecteur vitesse.....	7
b. De l'addition de vecteurs au principe de combinaison des vitesses	7
2. Modèle matriciel de la rotation	8
a. Construction de la rotation vectorielle dans un espace vectoriel	8
i. Définition : espace vectoriel euclidien.....	8
ii. Définition : norme	8
iii. Définition : isométrie vectorielle.....	8
iv. Définition : rotation vectorielle.....	9
v. Définition : matrice de rotation.....	9
b. Construction de la matrice de rotation d'un vecteur dans le plan	9
c. Construction de la matrice de rotation d'un vecteur dans l'espace	9
B. Explication des 4 mouvements de base	10
1. Introduction à la géométrie de la rotation	10
a. Modèle géométrique des angles d'Euler	10
b. Modèle Lacet / Roulis / Tangage / Gaz	11
2. Déplacer le quadrirotor	12
a. Le mouvements de lacet et de gaz.....	12
i. Mouvement de lacet	12
ii. Mouvement de gaz	12
b. Mouvement de roulis et de tangage	12
Conclusion partielle.....	14
II. Construction du drone.....	15
A. Fiche Composant.....	15
B. Le châssis.....	18
C. Assemblage de la cage batterie :.....	19
D. Refroidissement des ESC et des moteurs :	19
E. Vibration et résonance :.....	21

F.	Connectivité :	21
G.	Circuit électronique :	21
H.	Interférences électromagnétiques.....	23
	Conclusion partielle	26
III.	Principes de fonctionnement de la télécommande.....	27
IV.	Fonctionnement du gyroscope.	35
A.	La configuration initiale.....	35
B.	Calibration.....	36
V.	Principe de fonctionnement d'un correcteur PID	37
A.	Introduction sur le correcteur PID	37
B.	Etalonnage des valeurs du gain du correcteur PID :.....	39
C.	Explication du code :	40
IV.	Assigner ou relever la tension d'un pin.....	43
A.	Manipuler les pins en utilisant la librairie Arduino.....	43
B.	Les ports registre	44
1.	Principe	44
2.	Les opérateurs	45
a.	Le Bitwise AND (&)	45
b.	Le bitwise OU ().....	45
c.	Le bitwise OU EXCLUSIF (^).....	45
3.	Temps d'exécution	46
C.	Utilisation des ports registres	48
1.	Les moteurs	49
VI.	Contrôle des moteurs avec la librairie servo.....	51
	Conclusion.....	52
	Conclusion.....	53
	Immenses Remerciements.....	54

Introduction

Les drones sont au cœur des préoccupations technologiques actuelles : la France a introduit une législation très stricte concernant leur utilisation, la poste australienne prévoit de les utiliser pour livrer des colis, l'armée américaine les utilise comme arme de guerre en opérations extérieures. Les drones illustrent l'avènement de la technique, de son essor, et de son introduction dans tous les domaines de la vie. Utilisés dans l'industrie du cinéma, dans l'agroalimentaire pour surveiller les champs, par R.F.F. et l'E.D.F. pour la surveillance des réseaux, simples à maîtriser et peu coûteux, ils sont en passe de révolutionner l'espace aérien. Alliés aux puissants algorithmes de Machine Learning, à l'augmentation des débits des réseaux de communication (3G, 4G, wifi mimo), ils tendent à devenir des outils inévitables, mais posent aussi de sérieuses questions sur le respect de la vie privée, pouvant tout aussi bien devenir les outils d'un État de plus en plus totalitaire et d'un Big Brother trop maternel.

De nos jours, acheter un drone est d'une facilité déconcertante, et le marché est en pleine expansion avec une demande toujours croissante : du simple jouet pour enfant à 10 € au drone professionnel à plusieurs milliers d'euro, la différence peut sembler abyssale, pourtant c'est bel et bien le même concept technologique qui est en jeu. Les multiples possibilités offertes par la modularité de ce concept en font un outil de choix dans tous les domaines où les mouvements peuvent être effectués amplement.

La réalisation d'un drone est un défi technologique et technique pour les ingénieurs. Chaque drone est un condensé de technologie, et, mis-à-part les jouets pour enfant, les drones embarquent des centrales inertielles de plus en plus complexes : GPS, magnétomètre, altimètre, baromètre, thermomètre, gyroscope et accéléromètre sont à la base de ce que nous voyons voler de plus en plus. Le quadrirotor que nous avons réalisé est classé dans la catégorie des systèmes volants les plus complexes à cause du nombre d'effets physiques ayant un lien avec sa dynamique.

En nous attelant à ce défi, nous avons cherché à déterminer quels étaient les enjeux de la réalisation d'un drone du point de vue d'un futur électronicien. Pour cela, nous avons d'abord étudié un modèle théorique simplifié du mouvement du drone, puis nous nous sommes attelés à sa réalisation matérielle, tout d'abord sur le hardware puis sur le software.

Le présent rapport de TIPE a pour but de présenter ces recherches et ce travail qui ont été effectués depuis septembre 2015 par Thierry LINCOLN, Romain VALAIX, Kévin TAN & Nicolas CHATIN DE CHASTAING, tous élèves de P1B à l'INSTITUT SUPERIEUR D'ÉLECTRONIQUE DE PARIS, encadrés par leur professeur de Physique M. ALI YAHIA, leur professeur de Mathématique M. DESLANDES, et le Responsable des Laboratoires, M. AFRIAT, qu'ils remercient chaleureusement pour leur aide et leur accompagnement durant l'année.

I. Étude théorique du déplacement d'un quadri-rotor

On muni l'espace d'un référentiel $(O; \vec{x}, \vec{y}, \vec{z})$ tels que $\vec{x} \cdot \vec{y} = \vec{y} \cdot \vec{z} = \vec{z} \cdot \vec{x} = 0$ et $\|\vec{x}\| = \|\vec{y}\| = \|\vec{z}\| = 1$ et $\vec{x} \vee \vec{y} = \vec{z}, \vec{y} \vee \vec{z} = \vec{x}, \vec{z} \vee \vec{x} = \vec{y}$, on note l'horloge du référentiel t . Notre étude se penchera tout d'abord sur les modes de représentation mathématiques du déplacement d'un système de points massiques dans un référentiel galiléen. Nous étudierons ensuite comment déplacer un quadrirotor.

A. Le déplacement dans un repère

1. Principe de combinaison des vitesses

a. Rappels : vecteur position et vecteur vitesse

Soit un point $M \begin{pmatrix} x \\ y \\ z \end{pmatrix}, (x, y, z) \in \mathbb{R}^3$, ses coordonnées dans la base du référentiel. On

définit le vecteur position par rapport au point O en $t [S]$ par : $\overrightarrow{OM}(t) = \begin{pmatrix} x_t \\ y_t \\ z_t \end{pmatrix}$ avec

(x_t, y_t, z_t) le triplet de fonctions donnant à l'instant $t [S]$ les coordonnées de M dans la base du référentiel. La position du mobile est supposée continue et dérivable. On définit le vecteur vitesse instantanée en $t [S]$ comme étant la dérivée du vecteur vitesse en t . On a alors la relation :

$$\vec{V}_M(t) = \frac{d\overrightarrow{OM}}{dt} = \begin{pmatrix} \frac{d(x_t)}{dt} \\ \frac{d(y_t)}{dt} \\ \frac{d(z_t)}{dt} \end{pmatrix} = \begin{pmatrix} v_{M,x} \\ v_{M,y} \\ v_{M,z} \end{pmatrix}.$$

b. De l'addition de vecteurs au principe de combinaison des vitesses

Rappel : addition de vecteurs : soient $\vec{i} \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$ et $\vec{j} \begin{pmatrix} x_j \\ y_j \\ z_j \end{pmatrix}$ deux vecteurs. Le vecteur $\vec{i} + \vec{j}$ est

défini par $\vec{i} + \vec{j} = \begin{pmatrix} x_i + x_j \\ y_i + y_j \\ z_i + z_j \end{pmatrix}$.

Nous allons évoquer ici la démonstration du principe de la combinaison des vitesses par une approche vectoriel. Cela aurait pu se faire par d'autres approches, cependant nous préférons la méthode vectorielle, car nous utiliserons des vecteurs dans la suite de l'exposé.

Pour $M \begin{pmatrix} x \\ y \\ z \end{pmatrix}$, on a, de manière évidente, d'après la définition de l'addition de vecteurs,

$$\overrightarrow{OM} = x \times \vec{x} + y \times \vec{y} + z \times \vec{z}$$

On a donc

$$\vec{V}_M(t) = \frac{d\overrightarrow{OM}}{dt} = \frac{d}{dt}(x \times \vec{x} + y \times \vec{y} + z \times \vec{z})$$

De plus, la dérivée d'une somme de fonction est la somme des dérivées des fonctions. Cette relation fonctionnelle est valable pour des vecteurs, cela a été démontré en cours de physique.

En notant $\vec{V}_{M,x}(t), \vec{V}_{M,y}(t), \vec{V}_{M,z}(t)$ définie comme étant respectivement la dérivée des vecteurs $x \times \vec{x}, y \times \vec{y}, z \times \vec{z}$

Il en résulte la relation finale :

$$\boxed{\vec{V}_M(t) = \vec{V}_{M,x}(t) + \vec{V}_{M,y}(t) + \vec{V}_{M,z}(t)}$$

2. Modèle matriciel de la rotation

a. Construction de la rotation vectorielle dans un espace vectoriel

Note : on ne traitera pas la construction de la rotation vectorielle dans un espace vectoriel d'un corps quelconque, on s'en tiendra à \mathbb{R} . Il faudrait alors traiter de la construction des valeurs absolue pour les corps quelconques.

i. Définition : espace vectoriel euclidien

Soit E , un \mathbb{R} espace vectoriel. E est dit espace vectoriel euclidien, si et seulement si :

$$\exists s: E \times E \rightarrow \mathbb{R} / s \text{ est bilinéaire, symétrique, positive, et } \forall x \in E, s((x, x)) = 0 \Rightarrow x = 0$$

On appelle s produit scalaire de E , noté communément « . » ou $\forall (x, y) \in E^2 \langle x|y \rangle$.

ii. Définition : norme

Soit \mathcal{N} une application de E dans \mathbb{R} . \mathcal{N} est appelée norme sur E si et seulement si :

- $\forall x \in E, \mathcal{N}(x) \geq 0$ (« positivité »)
- $\forall x \in E, \mathcal{N}(x) = 0 \Rightarrow x = 0_E$ (« séparation »)
- $\forall (\mu, x) \in \mathbb{R} \times E, \mathcal{N}(\mu x) = |\mu| \mathcal{N}(x)$ (« homogénéité »)
- $\forall (x, y) \in E^2, \mathcal{N}(x + y) \leq \mathcal{N}(x) + \mathcal{N}(y)$ (« inégalité triangulaire »)

Un espace vectoriel doté d'une norme est appelé espace vectoriel normé.

Nota bene : une norme n'est pas nécessairement une application linéaire. « La » norme n'est pas non plus nécessairement unique sur un espace vectoriel.

iii. Définition : isométrie vectorielle

Soit E un \mathbb{R} espace vectoriel normé euclidien. Soit \mathcal{N} une norme sur E . Soit f un endomorphisme de E . f est dite isométrie vectorielle si et seulement si $\forall x \in E, \mathcal{N}(f(x)) = \mathcal{N}(x)$

Vocabulaire : si le déterminant de la matrice représentative d'une isométrie vectorielle dans une base orthonormée est 1, celle-ci est dite positive, si le déterminant est égal à -1 l'isométrie est dite négative.

iv. Définition : rotation vectorielle

Soit E un \mathbb{R} espace vectoriel euclidien. Une rotation vectorielle est une isométrie vectorielle positive.

v. Définition : matrice de rotation

Soit f une rotation vectorielle de E un \mathbb{R} espace vectoriel euclidien. Soit B une base orthonormée de E . $\text{mat}_B(f)$ est appelée matrice de rotation.

b. Construction de la matrice de rotation d'un vecteur dans le plan

Pour une construction plus simpliste, nous admettons comme acquis le théorème complexe de la rotation v' d'un vecteur v par un angle θ . On note $z = x + iy$ & $z' = x' + iy'$ l'affixe respective de v et v' .

$$z' = e^{i\theta} z, \text{ soit } z' = (\cos \theta + i \sin \theta) z, \text{ soit } z' = x \times \cos \theta + ix \times \sin \theta + iy \times \cos \theta - y \times \sin \theta$$

$$\text{Soit : } x' + iy' = x \times \cos \theta - y \times \sin \theta + i(x \times \sin \theta + y \times \cos \theta)$$

Soit, par identification :

$$\begin{cases} x' = x \times \cos \theta - y \times \sin \theta \\ y' = x \times \sin \theta + y \times \cos \theta \end{cases}, \text{ on peut écrire le système sous forme matricielle :}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \times \cos \theta - y \times \sin \theta \\ x \times \sin \theta + y \times \cos \theta \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix}$$

On a donc :

$$\boxed{\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix}}$$

c. Construction de la matrice de rotation d'un vecteur dans l'espace

La construction des matrices de rotation sur 3 axes est assez complexe et demande un développement qui devrait faire l'objet d'un autre TIPE.

La construction part de la démonstration de l'existence d'une droite de l'espace, invariante par la rotation. Pour cela, on cherche un vecteur invariant. La rotation f étant une application linéaire, on a alors une invariance de rotation pour tout vecteur se trouvant dans la droite vectorielle engendrée par ledit vecteur trouvé invariant. Cette droite vectorielle est appelée axe de rotation.

Ensuite, on démontre que pour chaque vecteur de l'espace, la transformation par f se résume en une rotation d'un angle ϕ dans le plan orthogonal au vecteur unitaire de l'axe, tel que le vecteur à transformer appartiennent au plan. ϕ étant fixé et indépendant du vecteur à transformer. ϕ est appelé angle de la rotation.

Pour f une rotation quelconque, d'axe de vecteur unitaire $\vec{N} \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}$ et d'angle ϕ , on a pour matrice de rotation M :

$$M = (\cos \phi)I_3 + (1 - \cos \phi) \begin{pmatrix} n_x^2 & n_x n_y & n_x n_z \\ n_x n_y & n_y^2 & n_y n_z \\ n_x n_z & n_y n_z & n_z^2 \end{pmatrix} + (\sin \phi) \begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}$$

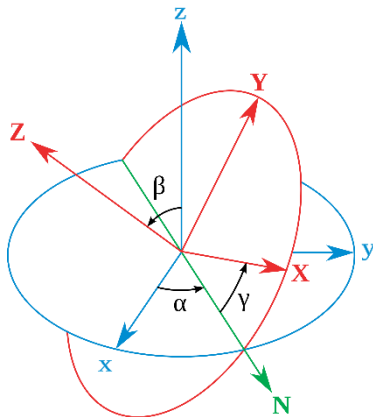
Nota bene : étant donné qu'une rotation est une application linéaire, il est démontrable que la rotation totale est la composée de trois rotations autour des 3 axes. La matrice de rotation totale est donc donnée par le produit des trois matrices de rotations.

B. Explication des 4 mouvements de base

1. Introduction à la géométrie de la rotation

a. Modèle géométrique des angles d'Euler

Euler a proposé un modèle géométrique pour décrire l'orientation d'un solide par rapport à un axe. Celui-ci est défini comme tel :



Soit un système de points mobiles supposé indéformable.

On fixe un repère fixe direct $(O; x, y, z)$ au centre de gravité du mobile.

On fixe trois points (X, Y, Z) sur le mobile. On définit alors le repère mobile par $(O'; X, Y, Z)$. O et O' étant ici confondus.

On définit :

- la ligne de nœud comme étant la droite de vecteur unitaire \vec{N} formée par l'intersection entre le plan $(O; x, y)$ et le plan $(O; X, Y)$ lorsque ceux-ci ne sont pas confondus.
- L'angle entre \vec{N} et \vec{x} noté α .
- L'angle entre \vec{Z} et \vec{z} noté β .
- L'angle entre \vec{N} et \vec{X} noté γ .

Pour définir la position d'un mobile (par exemple un drone) par rapport au repère fixe, on n'utilisera alors que 6 paramètres : les coordonnées de son centre de masse dans le repère $(O; x, y, z)$ et les trois angles d'Euler définis plus haut. Pour déterminer les angles d'Euler, on supposera que O et O' sont confondus.

Ce modèle constitue une généralisation du repérage sphérique qui n'utilise que 3 paramètres pour déterminer l'emplacement d'un point dans une sphère : sa distance à l'axe et 2 angles (longitude et latitude).

La matrice de rotation de ce modèle est assez simple :

$$M = \begin{pmatrix} \cos(\alpha) \cos(\gamma) - \sin(\alpha) \cos(\beta) \sin(\gamma) & -\cos(\alpha) \sin(\gamma) - \sin(\alpha) \cos(\beta) \cos(\gamma) & \sin(\alpha) \sin(\beta) \\ \sin(\alpha) \cos(\gamma) + \cos(\alpha) \cos(\beta) \sin(\gamma) & -\sin(\alpha) \sin(\gamma) + \cos(\alpha) \cos(\beta) \cos(\gamma) & -\cos(\alpha) \sin(\beta) \\ \sin(\beta) \sin(\gamma) & \sin(\beta) \cos(\gamma) & \cos(\beta) \end{pmatrix}$$

Nota bene : il est possible de la factoriser comme mentionné lors de la construction des matrices de rotation dans l'espace. On aurait alors le produit des trois matrices de rotation selon chaque angle.

Ce modèle nécessite la détermination d'une intersection entre deux plans qui n'est pas nécessairement une droite, notamment lorsque les deux plans sont confondus. De plus il est difficilement utilisable pour notre drone : les capteurs utilisent le référentiel terrestre et mesurent des angles différents de ceux détaillés ici. Cette représentation ne peut pas être en conséquence utilisée pour notre drone.

b. Modèle Lacet / Roulis / Tangage / Gaz

Comme nous l'avons dit, le modèle d'Euler n'est pas satisfaisant. Il a été nécessaire de développer un modèle qui se rapproche des notions de roulis et tangage présents dans la marine depuis les débuts de la navigation, auxquelles on rajoute les mouvements de lacet et de gaz développés pour l'aviation.

Soit un mobile orienté (disposant d'un avant et d'un arrière) sur lequel on fixe un repère orthonormé direct $(O; \vec{x}, \vec{y}, \vec{z})$. On impose que l'axe (O, \vec{x}) soit le sens avant du mobile et O soit confondu au centre de masse O' à l'état initial supposé stable. Le repère est supposé invariant. On fixe de même un repère $(O'; \vec{X}, \vec{Y}, \vec{Z})$ orthogonal direct.

On définit alors :

- L'angle de tangage comme étant l'angle formé par le vecteur \vec{x} et le projeté orthogonal de \vec{X} sur le plan $(O; \vec{x}, \vec{z})$. C'est-à-dire l'angle de rotation autour de l'axe $(O; \vec{y})$ que l'on appelle axe de tangage.
- L'angle de roulis comme étant l'angle formé par le vecteur \vec{y} et le projeté orthogonal de \vec{Y} sur le plan $(O; \vec{y}, \vec{z})$. C'est-à-dire l'angle de rotation autour de l'axe $(O; \vec{x})$ que l'on appelle axe de roulis.
- L'angle de lacet comme étant l'angle formé par le vecteur \vec{x} et le projeté orthogonal de \vec{X} sur la plan $(O; \vec{x}, \vec{y})$. C'est-à-dire l'angle de rotation autour de l'axe $(O; \vec{z})$ que l'on appelle axe de lacet.
- Les gaz comme étant la translation du point O vers le projeté orthogonal de O' sur (O, \vec{z})

On s'aperçoit que ce modèle est très proche de celui mis en place par Euler. Son avantage indéniable consiste en le fait que les projetés orthogonaux sont toujours définis, contrairement à la ligne de nœud.

2. Déplacer le quadrirotor

Sur le quadrirotor sont placés 4 moteurs au bout de deux axes formant une croix. On suppose que le centre de masse se trouve à la croisée des deux axes. On fixe sur le drone les deux repères désormais usuels tels que le centre de masse et les centres des repères soient confondus, et que les axes (O, \vec{x}) , (O, \vec{y}) , soient confondus à l'état initial supposé stable avec la direction avant du drone.

a. Le mouvements de lacet et de gaz

i. Mouvement de lacet

Comme nous l'avons vu, le mouvement de lacet consiste à réorienter l'avant du drone. Pour cela, nous utilisons une particularité du système {moteur-pale}. En effet, la rotation de la pale induit deux forces : la poussée \vec{P} orthogonale au plan de rotation de la pale, et la trainée \vec{T} qui est opposée à la vitesse tangentielle de rotation.

Nota bene : la valeur de la force de trainée dépend du carré de la vitesse de rotation et d'un coefficient variable selon les pales (incidence d'attaque, épaisseur, matériau, etc...). Physiquement, elle est une conséquence des frottements de la pale dans l'air.

C'est à cause de la force de trainée que les moteurs sont disposés dans des sens opposés deux à deux. En effet, pour les mêmes pales et moteur, les forces de trainée vont se compenser et annuler le mouvement de lacet naturel. Pour effectuer un mouvement de lacet, il suffit donc d'augmenter la vitesse de rotation des moteurs tournant dans le sens voulu et de diminuer d'un même facteur ceux tournant dans l'autre sens. La trainée totale induite par les moteurs du sens voulu sera supérieure à celle induite par les moteurs tournant dans l'autre sens et le drone se mettra à tourner sur l'axe de lacet.

ii. Mouvement de gaz

Le mouvement de gaz consiste à élever l'altitude du centre de masse du drone, c'est-à-dire effectuer une translation de O' sur l'axe de lacet. Les moteurs étant fixés orthogonalement les forces de poussée sont, à l'état stable initial, colinéaires à l'axe de lacet. Il suffit donc d'augmenter la vitesse de rotation des moteurs de manière identique pour effectuer un mouvement de gaz vers le haut, ou la diminuer pour effectuer un mouvement de gaz vers le bas.

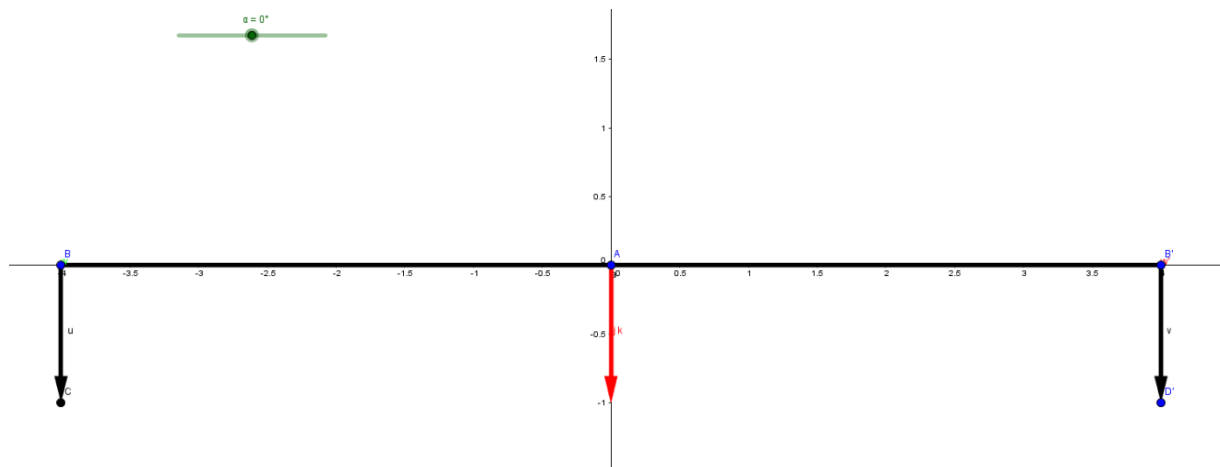
b. Mouvement de roulis et de tangage

Le mouvement de roulis et le mouvement de tangage sont sensiblement les mêmes, nous n'étudierons donc que le roulis.

La plus part des drones grand public font le choix de les placer sur les bissectrices de (O, \vec{x}, \vec{y}) dans le plan $(O; \vec{x}, \vec{y})$, généralement pour accorder un meilleur angle de vue aux éventuelles caméras embarquées. Nous avons fait le choix de placer chaque moteur sur un axe pour simplifier notre travail sur le drone. En effet, les moteurs étant placés sur les axes (O, \vec{x}) et (O, \vec{y}) il suffit de modifier la vitesse de rotation de seulement 2 moteurs pour déplacer le drone, ce qui simplifie le déplacement sur le plan $(O; \vec{x}, \vec{y})$.

On considère le plan $(O; \vec{x}, \vec{z})$ ainsi que les deux moteurs placés en B et B' tels que $\overrightarrow{OB} = -\overrightarrow{OB'}$ et B et B' appartiennent à l'axe (O, \vec{x}) .

La disposition des pales évoquée au-dessus imposent que la force de poussée induite par la rotation des pales soient orthogonale à l'axe (O, \vec{x}) et appartiennent au plan (O, \vec{x}, \vec{z}) . On se place donc dans ce plan.



État stable initial : la poussée étant orthogonale à l'axe et les deux moteurs tournant à la même vitesse, le système reste à l'horizontal. On suppose le système totalement isolé.

Le système étant isolé, d'après le PFD il ne se déplacera sur l'axe (O, \vec{y}) que s'il existe une force dont le projeté orthogonal sur l'axe n'est pas nul. Comme les moteurs sont fixés à l'axe, lui sera toujours orthogonale. Il suffit donc de déstabiliser le drone sur cet axe pour que la poussée admette une composante latérale et donc que le drone se déplace sur l'axe.



État déstabilisé : le drone n'est plus aligné avec l'axe, la poussée orthogonale à l'axe du drone admet donc une composante horizontale et verticale.

Pour déstabiliser le drone, on diminue d'un certain facteur la vitesse de rotation et l'on augmente la vitesse de rotation de l'autre moteur. Le moment de la poussée du deuxième moteur sera supérieur à celui du premier, le drone va donc se mettre à tourner autour de l'axe de (O, \vec{x}) , ce qui correspond à la rotation de roulis souhaitée.

Nota Bene : le système ne pourra pas aller jusqu'à une accélération infinie, les forces de frottement s'opposent très rapidement au mouvement : la surface de rotation des pales augmentent de beaucoup cette sensibilité aux rotations (les forces de frottement étant proportionnelles au carré de la surface soumise).

Pour augmenter la vitesse de déplacement sur l'axe, il suffit alors d'augmenter l'inclinaison du drone, ce qui aura pour effet d'augmenter la composante horizontale de la poussée. Toute fois ce système a des limites, car l'augmentation de l'inclinaison diminue la portance totale du drone (somme des composantes horizontales des poussées), jusqu'à être inférieure au poids et donc entrainer la diminution de l'altitude du drone.

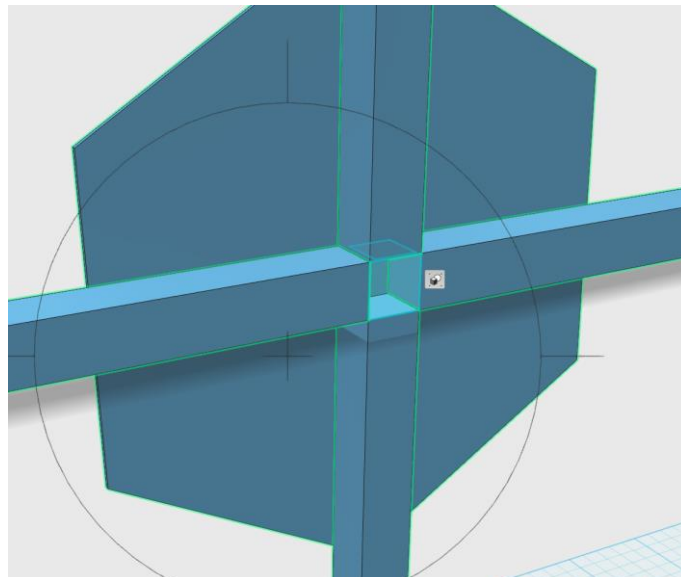
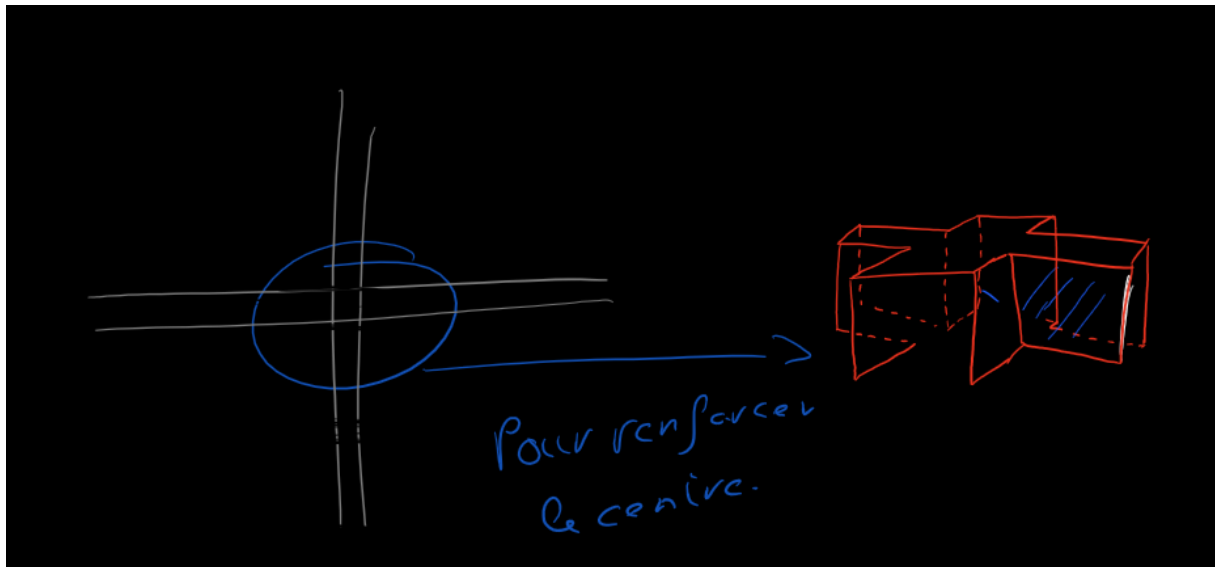
Conclusion partielle

Comme nous l'avons vu, la réalisation d'un quadrirotor nécessite le développement d'outils théoriques mathématiques et physiques pour décrire le mouvement, afin de, comme nous le verrons par la suite, asservir le drone.

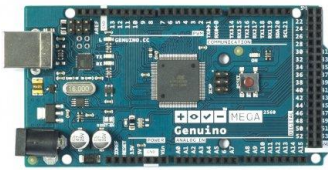
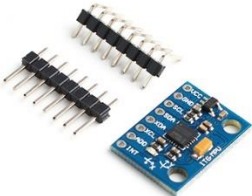


II. Construction du drone





Le réel enjeu de cette partie est de vous montrer à quel point le nombre de paramètres mécaniques et électroniques à prendre en compte est conséquent lorsque l'on conçoit un drone. Nous commencerons par la mécanique, puis nous verrons par les optimisations électroniques possibles.

La première chose à faire lorsque l'on se prépare à la construction d'un drone, c'est de choisir la taille du drone que l'on veut et donc son poids. Pour notre drone, le choix a été simple, nous voulions un drone suffisamment grand et lourd pour une stabilisation plus facile. En effet plus le drone est léger et petit plus celui-ci est difficile à stabiliser.

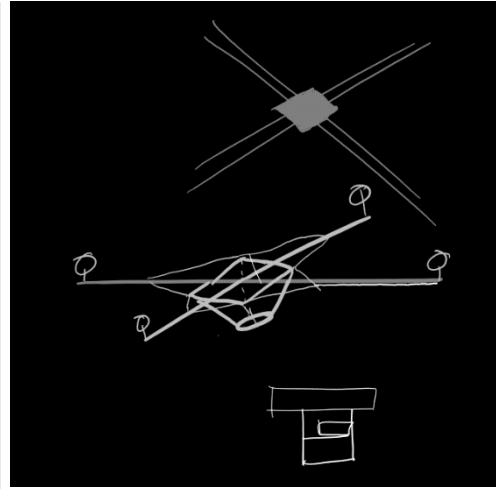
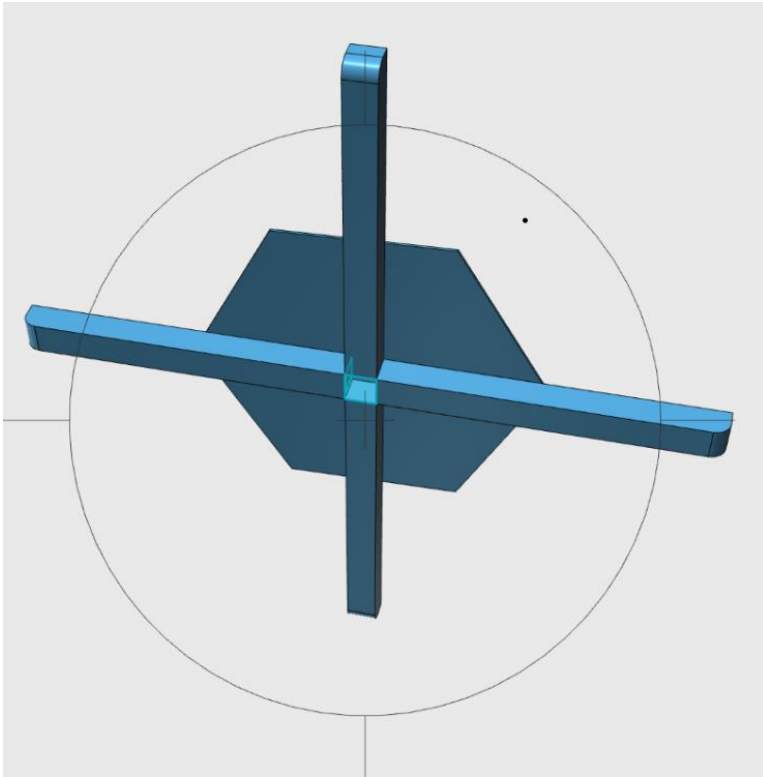


A. Fiche Composant.

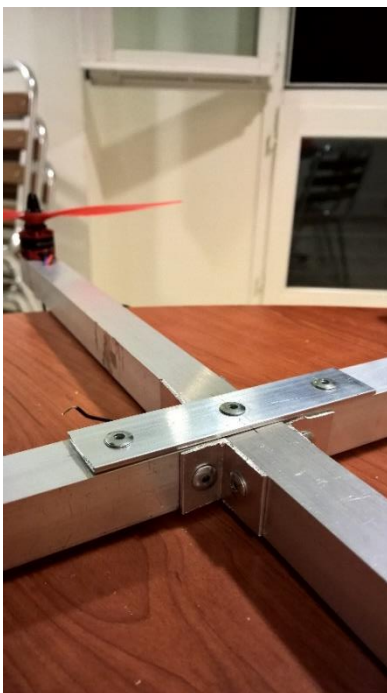
Produit	Image du Produit	Lien	Description
Genuino Méga. Original Arduino		https://store.arduino.cc/product/GBX00067	The Mega 2560 is a microcontroller board based on the ATmega2560 . It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button.
Kingduino MPU6050 3-Axis Gyroscope 3-Axis Accelerometer		http://www.hobbyking.com/hobbyking/store/_26860_Kingduino_MPU6050_3_Axis_Gyroscope_3_Axis_Accelerometer.html	Arduino MPU6050 3-Axis Gyroscope 3-Axis Accelerometer
Moteur + Turnigy D2830-11 1000kv Brushless Motor		http://www.hobbyking.com/hobbyking/store/_12921_Turnigy_D2830_11_1000kv_Brushless_Motor.html	Specs: Rpm/V: 1000kv Shaft: 3.17mm Voltage: 2S~4S (7.4v to 14.8v) Weight: 52g Watts: 210w Max Current: 21A ESC: 30A Suggested Prop: 8x4 (4S) ~ 10x7 (2S) Mounting Hole Bolt Circle: 16mm or 19mm
Contrôleur électronique de vitesse : Turnigy Multistar 30 Amp BLHeli Multi-rotor Brushless ESC 2-6S V2.0		http://www.hobbyking.com/hobbyking/store/_65157_Turnigy_Multistar_30_Amp_BLHeli_Multi-rotor_Brushless_ESC_2_6S_V2_0.html	Specs: Constant Input Voltage: 2-6 cells Lipoly Current: 30A Frequency: 20-500Hz BEC: Yes (Switching) [Remove middle wire to disable] BEC Output: 5V/4APWM: 8 KHz Max RPM: 240,000rpm for 2 Poles Brushless Motor PCB Size: 41mm x 24mm Discharge Plugs: Male 3.5mm Bullet Connector Motor Plugs: Female 3.5mm Bullet Connector Weight: 35g

Capteur Ultrasons		http://www.hobbyking.com/hobbyking/store/_31136_Ultrasonic_Module_HC_SR04_Kingduino.html	Power: 5V DC Quiescent Current : <2mA Effectual Angle: <15° Ranging Distance : 2cm – 500 cm/1" – 16ft Resolution : 0.3 cm LxWxH: 45 x 20 x 15mm Weight: 8.5g Includes: Ultrasonic Module 1 x Terminated ribbon cable 200mm
Telecommande+recepteur 2.4Ghz HobbyKing 2.4Ghz 6Ch Tx & Rx V2 (Mode 2)(THR OTTLE ON THE LEFT)		http://www.hobbyking.com/hobbyking/store/_9042_Hobby_King_2_4Ghz_6Ch_Tx_Rx_V2_Mode_2_.html	6-channel 2.4GHz transmitter with servo reversing. Easy to use control for basic models. Includes 6-channel receiver Trainer system option. This system must be programmed via PC cable. Note: Only HK-T6A receiver is compatible with HK-T6A transmitter. Only HK-T6Av2 receiver is compatible with the HK-T6Av2 transmitter. PRODUCT ID: HK-T6A-M2
Batteries:		http://www.hobbyking.com/hobbyking/store/_36195_ZIPPY_Compact_3700mAh_3S_25C_Lipo_Pack_EU_warehouse_.html	Spec. Capacity: 3700mAh Voltage: 3S1P / 3 Cell / 11.1V Discharge: 25C Constant / 35C Burst Weight: 264g (including wire, plug & case) Dimensions: 146x19x43mm Balance Plug: JST-XH Discharge Plug: HXT4mm
Pales / Helice		http://hobbyking.com/hobbyking/store/_25815_8045_SF_Props_2pc_CW_2_pc_CCW_Rotation.html	8045 SF Props 2pc CW 2 pc CCW Rotation - Complete with spacer rings. Length (Inch [X]) 8 Pitch (Inch [Y]) 4.5
Tube d'aluminium		http://www.castorama.fr/store/Tube-carre-alu-brut-20-x-20-mm-1-m-prod4950012.html;jsessionid=8AB7A5BCE00876C56E86F8002E43F15C.fo3atg2?navAction=jump&isSearchResult=true	

B. Le châssis



Pour ce projet nous avons choisi la construction la plus simple possible : la configuration en croix. Tout d'abord parce que la plupart des drones l'utilisent, mais aussi parce qu'elle nous paraît la plus simple à stabiliser. Voici le premier modèle réalisé.



Choix du matériau :

Plusieurs matériaux se présentaient à nous : le plastique, le bois, la fibre de carbone, etc... Finalement nous avons choisi l'aluminium à cause de sa légèreté et de sa rigidité.

Le châssis est assemblé de la manière suivante : une tige creuse d'un mètre d'aluminium est sectionnée en deux, ensuite un des deux morceaux de 50cm est lui aussi coupé en deux sections de 24cm chacune. Mise en une structure en croix, les pièces sont rivetées à 4 équerres d'aluminium sur le côté. Enfin pour solidifier le tout une plaque d'aluminium est rivetée sur le haut reliant les deux morceaux de 24cm. Le tout pèse précisément 261g ce qui peut paraître lourd, ce qui est cependant dans les normes des châssis trouvables dans le commerce.

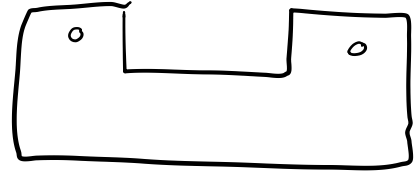
Un effort vers le bas est impossible, la plaque avec les trois rivets tient le tout bien en place. Cependant, on peut se poser des

questions quant à la solidité de la structure si l'effort est du bas vers le haut. Dans ce sens, le châssis en lui-même ne peut pas tenir un choc important, ce qui est évidemment problématique, car un drone tombe en général dans ce sens.

C. Assemblage de la cage batterie :

La cage batterie a été pensée tout d'abord pour accueillir la batterie, mais aussi le transmetteur radio et la carte Arduino. Elle est composée de plaques de plexi-glace coupées de la façon suivante (*cf. schéma*) :

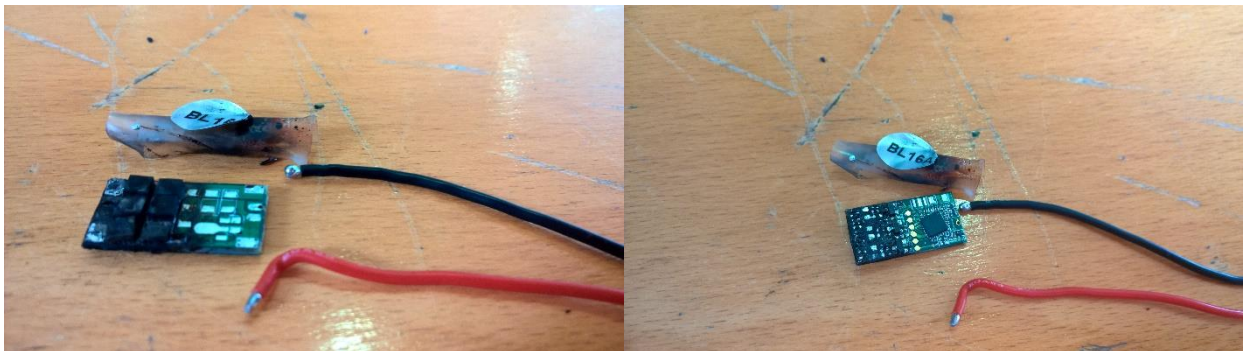
L'intérêt réside dans la manière d'accrocher la cage au châssis. Cette plaque y est boulonnée et permet de solidifier la structure si un effort est effectué du bas vers le haut.



Il est par ailleurs important de protéger la batterie d'un drone celle-ci peut en effet exploser. La plaque de plexi-glace a d'ailleurs épargné la mort d'un des membres du groupe : En effet lors de l'incendie d'un E.S.C., la flamme a atteint la cage de la batterie et a fondu le plexi-glace. Sans celle-ci la batterie aurait pris feu. L'explosion aurait pu être terrible.

Pourquoi avoir choisi le plexi-glace ? Bien que ce soit un matériau assez lourd, nous n'avions que cela à disposition.

D. Refroidissement des ESC et des moteurs :



Avant d'arriver à la meilleure configuration, nous avons quand même commis quelques erreurs.

Rapport d'erreur incendie des E.S.C. (*contrôleur électronique de vitesse*) :

Date : Février 2016 ; nous avons acheté des petits moteurs et des petits E.S.C.

En choisissant d'installer sur des petits moteurs de grosses pales, nous pensions que le seul problème qui pourrait arriver serait une usure prématurée des moteurs. Cependant

cela s'avéré être complètement faux. Les tests au laboratoire ne montraient pas en effet de signes de surchauffe, ou quelques autres problèmes avec cette configuration.

Trois facteurs peuvent expliquer l'absence de problème en laboratoire :

- L'E.S.C. était branché sur un amplificateur de puissance, qui limite le courant à 3A par sécurité.
- L'E.S.C. était commandé par un potentiomètre, et donc les variations de vitesse étaient moindres.
- Le temps d'utilisation était plus restreint

Le jour de la catastrophe :

- Les moteurs étaient équipés des pales 8x4.5 pouces.
- L'E.S.C. était branché à la batterie.

Après une minute avec puissance demandée au moteur maximale, l'ESC prit feu. La chaleur fut tellement importante que toutes les soudures de l'E.S.C. furent détruites. Les pales trop grandes produisaient une force contre-électromotrice trop importante. L'ESC envoyait alors au moteur plus de courant pour que celui-ci atteigne une vitesse de rotation maximale soit environ $2400 \times 12 \approx 28\,000$ *Rotation par minute*. Une accumulation de courant trop forte, supérieure à 16 Ampères s'est alors produite dans le circuit ce qui a causé l'incendie.

Les dégâts se sont étendus à d'autre composant, la surtension a flashé la mémoire des deux autres E.S.C. qui étaient branchés en parallèle sur le même circuit. Ces deux E.S.C. ne sont plus capables de délivrer du triphasé.

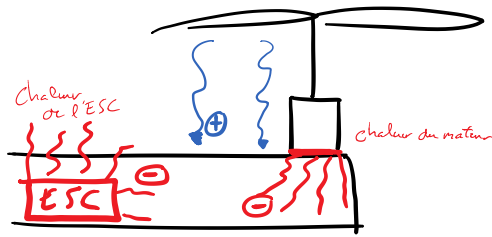
Suites à ces problèmes il fallait réagir rapidement. Fallait-il acheter de nouveaux E.S.C., en gardant les mêmes moteurs ? Fallait-il acheter de nouveaux moteurs et de nouveaux E.S.C. ? Un nouvel investissement était plus que nécessaire.

Aucune erreur ne devait plus être commise ; les nouveaux moteurs de 1000 KV seront suffisamment puissants pour accompagner des E.S.C. de 30 A. On a la relation suivante entre la tension d'alimentation aux bornes de l'E.S.C et le nombre de rotation par minutes du moteur : $V = \frac{rpm}{KV}$

Les nouveaux moteurs et E.S.C ont été montés sur le drone actuel, accompagnés de quelques ajustements thermaux.

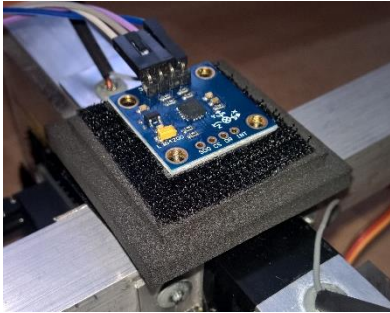


L'aluminium est un excellent conducteur thermique, nous avons donc décidé de coller la plaque réfrigérante des E.S.C. directement au châssis avec une pate thermique. En cas d'utilisation intensive les E.S.C. pourront évacuer rapidement leur chaleur dans l'aluminium pour minimiser les risques d'incendies. De même pour les moteurs.



De plus les pales en tournant produisent du vent qui lui-même refroidi la structure en aluminium. Le tout agit alors comme un grand radiateur.

E. Vibration et résonance :

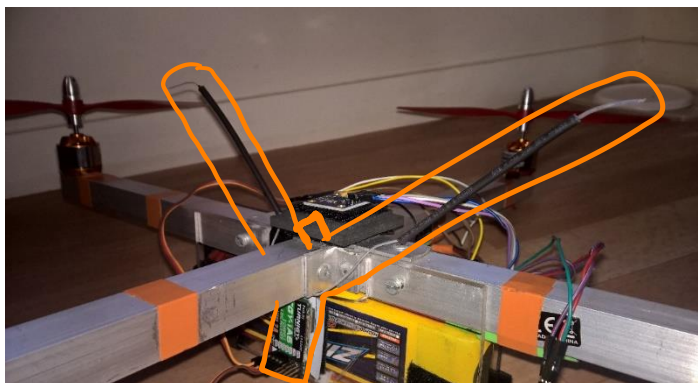


Le pire ennemi des drones est souvent les vibrations. En effet celles-ci peuvent provoquer de la résonance et perturber le gyroscope qui a son tour à cause du PID fait osciller le drone. Pour diminuer et optimiser ce problème nous avons recourt à une solution plutôt simple. Nous avons isolé le gyroscope du châssis de manière astucieuse : celui-ci est scratché sur une tour en mousse qui est elle-même attachée avec du velcro sur le châssis. Pour perturber le gyroscope, les vibrations doivent ainsi traverser une première couche de velcro, de la mousse et une seconde couche de velcro. Le gyroscope est ainsi quasiment isolé des vibrations et transmet une valeur plus correcte du mouvement sur les 3 axes.

Ce châssis a toutefois quelques failles : d'une part l'aluminium est un matériau qui a tendance à transmettre les vibrations, d'autre part la construction elle-même comporte certains défauts. En effet le fait d'avoir riveté deux pièces en aluminium laisse la place à des micros-vibrations.

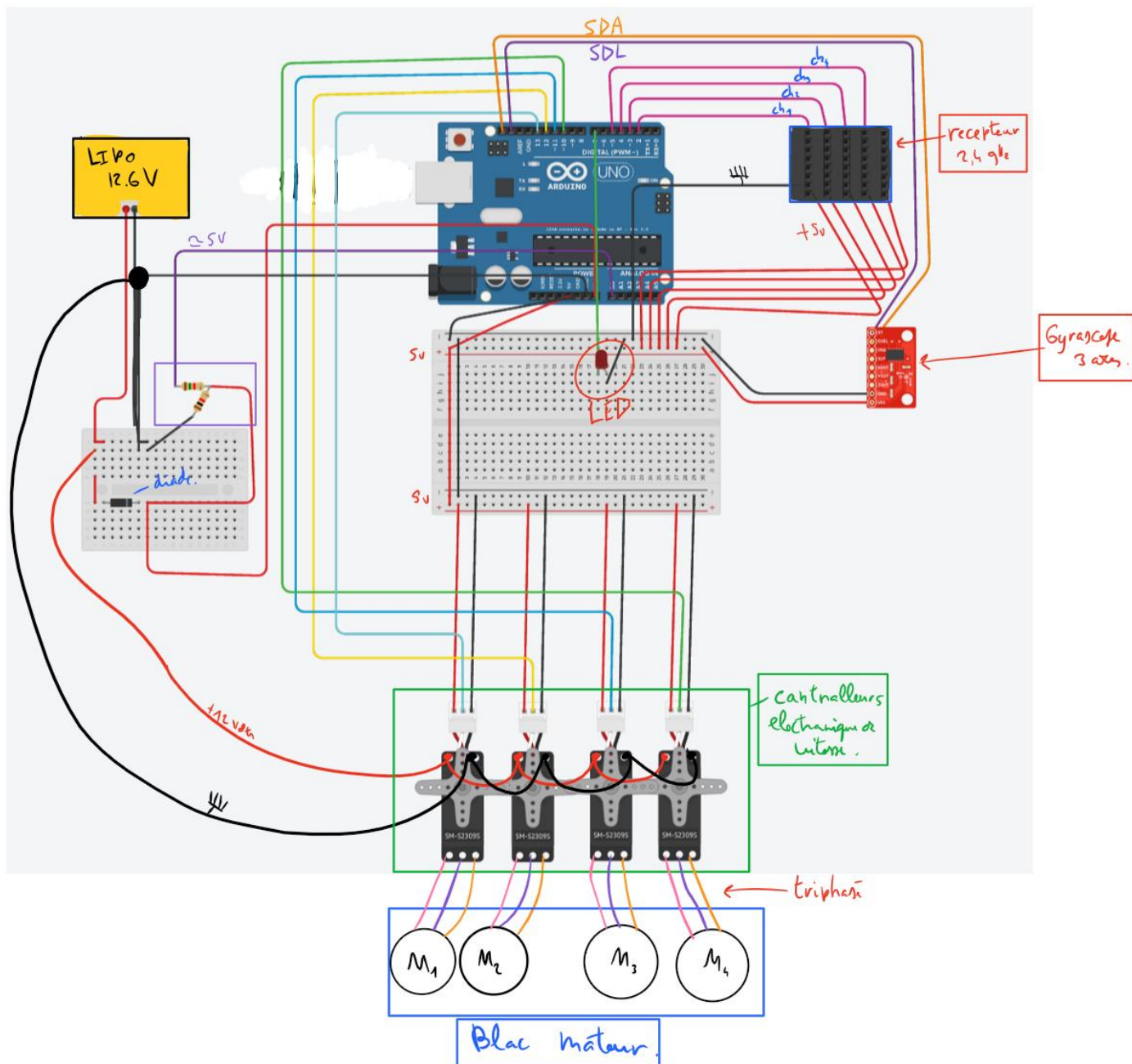
F. Connectivité :

Pour recevoir un signal correct de la télécommande, le récepteur doit être placé avec un angle de 90° entre les deux antennes :

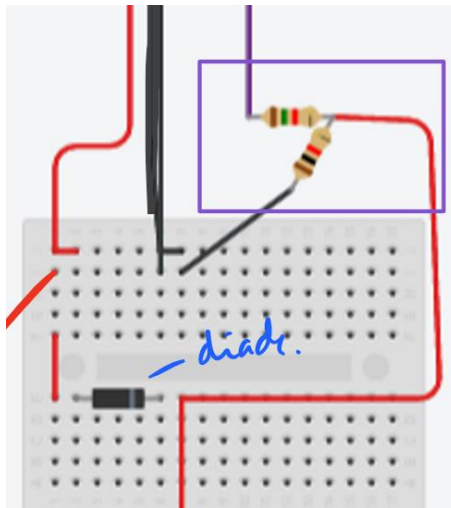


On notera passage que les antennes sont placées dans des tubes de gaine thermo-rétractable, et fixées sur des supports de fils de cuivre, pour pouvoir aplatir les antennes afin de transporter le drone.

G. Circuit électronique :



Les quatre contrôleurs électroniques de vitesse (*E.S.C.*) sont branchés en dérivation sur la batterie. Les moteurs sont directement reliés aux ESC. La carte Arduino est quant à elle alimentée en courant grâce au port *Vin* qui est directement branché sur un régulateur de tension 5 V. Les ports 5 V et la masse de l'Arduino alimentent quant à eux le gyroscope et le récepteur de la télécommande.



Une diode est branchée en dérivation sur la cathode de la batterie, puis celle-ci est relié à un simple diviseur de tension couplé à deux résistances l'une de 1,5 kΩ et l'autre de 1 kΩ.

Cela permet de lire la tension de la batterie. La carte Arduino ne peut lire sur ses ports analogiques qu'une tension de 0 à 5 V, ainsi brancher la batterie de 12V directement sur la carte celle-ci risquerait de la griller. On divise donc la tension par deux et l'on met les valeurs de résistance nécessaires pour ne jamais dépasser 5 V sur le pin de l'Arduino.

On multiplie ensuite la valeur de la tension pour avoir le voltage original. La diode empêche quant à elle le courant d'aller dans le sens inverse.

Plus la batterie est utilisée, plus la tension de la batterie diminue puisqu'elle s'épuise. Nous devons donc nous assurer que le pilote du drone soit informé de l'épuisement de celle-ci afin de limiter les risques d'accident.

Il faut tout d'abord calculer le voltage de la batterie. Celle-ci délivre une tension de 12.6V lorsqu'elle est pleine (La batterie est reliée au Pin A0). La valeur retournée par `analogRead()` est de 1023 pour 12.6V, nous devons donc faire le rapport $1260/1023 = 1.2317$ et multiplier ce ratio à la valeur d'`analogRead()` pour ramener la valeur de celle-ci à la bonne échelle. Il ne faut pas oublier que la diode fait perdre 0.7V aussi.

Le calcul de la tension de la batterie est donc :

```
tension_batterie= (analogRead(0) + 65) * 1.2317;
```

Le connecteur de la batterie étant relié au pin analogique 0 de l'Arduino.

Il faut maintenant prévenir le pilote lorsque la tension de la batterie est trop faible grâce à un buzzer :

```
if(tension_batterie < 1050 || tension_batterie > 600)digitalWrite(12, HIGH);
```

Pour faire fonctionner le buzzer, il suffit de lui envoyer une tension de 5 V. Ainsi, si la tension de la batterie est comprise entre 10,5 V et 9 V, c'est-à-dire lorsque la batterie est faible.

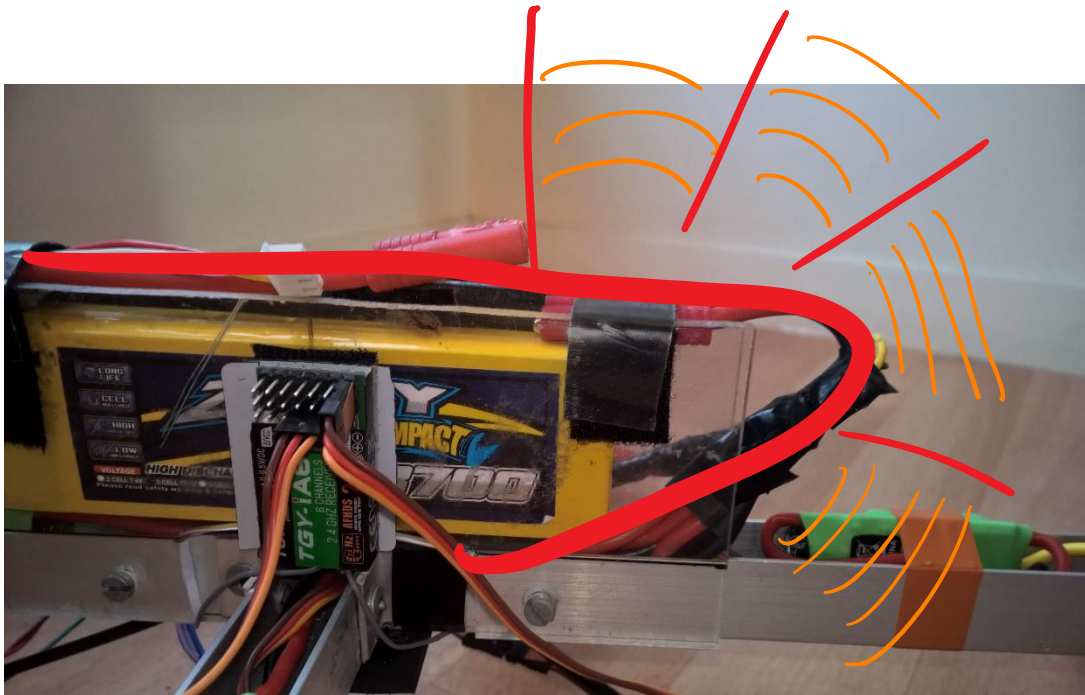
H. Interférences électromagnétiques

Un champ électromagnétique, peut être provoqué par un courant qui circule dans un fil. Plus le courant est fort, plus le champ électromagnétique est fort. On rappelle l'équation correspondante : $\vec{B} = \mu_o n I \vec{u}$

Les composants électroniques fonctionnent en général avec des courants de l'ordre du mA, il serait dommage de venir perturber ces faibles signaux. Cependant la consommation des 4 moteurs en pleine puissance simultanément fait circuler dans les fils

des courants de l'ordre de 20 A. Ces courants ont la capacité de générer des champs électromagnétiques importants. De plus, cet effet peut être plus important lorsque les fils qui transportent ces forts courants sont courbés. C'est un phénomène bien connu des ingénieurs du son. Lorsque l'on sonorise une pièce il est toujours déconseillé de courber des câbles qui transportent beaucoup de courant : des interférences électromagnétiques peuvent rayonner sur les appareils aux alentours et créer le fameux « bzzz » que l'on entend parfois.

Dans le cas du drone, les fils de la batterie sont configurés dans une position qui n'améliore guère la situation. En effet, l'on remarque que le fil central est très courbé, il se trouve d'ailleurs très proche des fils qui conduisent de précieux signaux, tels que ceux du gyroscope, des pulsations de la télécommande, et enfin des pulsations des moteurs. Tout ce méli-mélo conduit à bien des surprises : des temps de calculs irrégulier - le processeur ne tourne plus à la même vitesse -, les pulsations apparaissent parfois instables. Voir même pire, le niveau logique HIGH de la pulsation se retrouve à plus de 5V et le contrôleur électronique a du mal à comprendre l'information.



Pour régler ce problème, les concepteurs de circuits électroniques insolent tout câble contenant de l'information avec un tressage en fil qui fait office de bouclier. Les câbles conduisant de forts courants sont mis à l'écart du circuit électronique et ne sont surtout pas courbés.



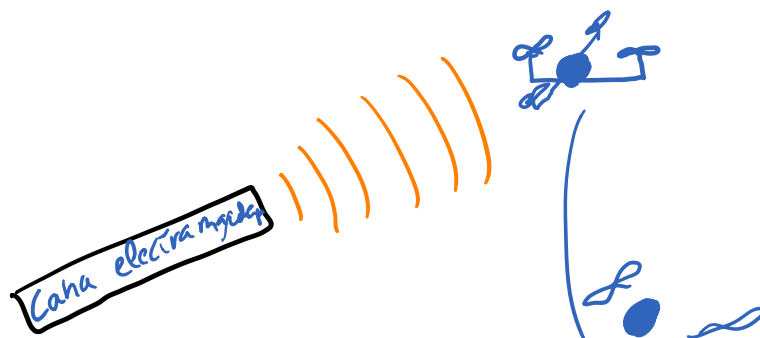
Les interférences électromagnétiques suscitent de très gros problèmes dans le monde de l'électronique. Même au niveau des avions de chasse ces interférences peuvent causer de sérieux problèmes de sécurité : l'avion peut s'écraser, et comme les missiles sont eux aussi commandés par un circuit électronique, ils peuvent être accidentellement détonés. C'est pour cela que ces appareils subissent des tests très sévères de résistances aux interférences.



Une vidéo qui discute de cela justement :

<https://www.youtube.com/watch?v=SmamWpTCuW8>

Avec les problèmes de sécurité qu'engendrent les drones aujourd'hui, n'est-il pas légitime de penser à utiliser des canons à interférences électromagnétiques à la place des aigles pour descendre les drones ?



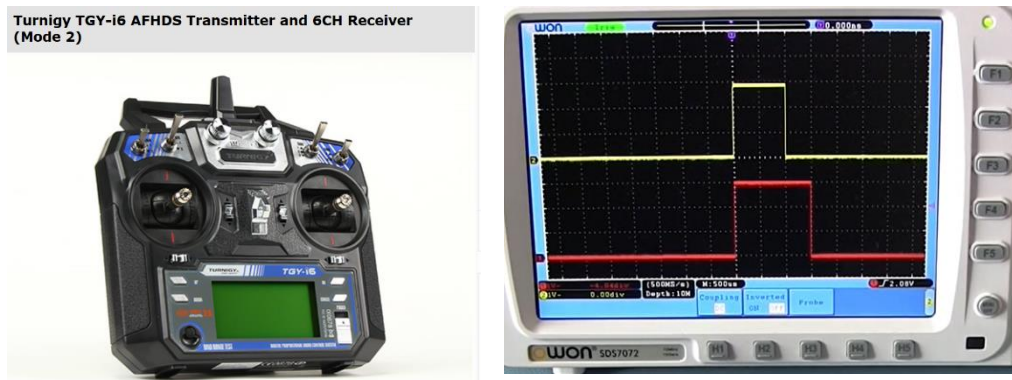
Conclusion : Comme vous l'avez vu dans cette partie construction, il y'a de réels enjeux lorsqu'on construit un drone. Car comme un avion, la partie matérielle n'est absolument pas à négliger au profil de la programmation. Le logiciel possède des attentes quand a la stabilité du système matériel. C'est pour cela qu'il est nécessaire de faire une construction impeccable.

Conclusion partielle

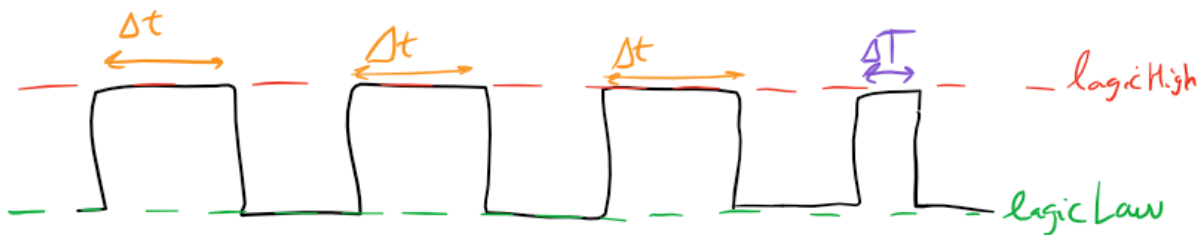
Comme vous l'avez vu dans cette partie construction, il-y-a de réels enjeux lorsqu'on construit un drone. Comme un avion, la partie matérielle n'est absolument pas à négliger au profil de la programmation. Le logiciel possède des attentes quand a la stabilité du système matériel. C'est pour cela qu'il est nécessaire de faire une construction parfaite.

III. Principes de fonctionnement de la télécommande

Un des enjeux de ce TIPE est la communication radio avec la télécommande, au niveau programmation. Il s'agit bel et bien la partie la plus dure du code. En effet, les temps de calculs sont limités, le code doit donc être le plus rapide possible.



La télécommande transmet les informations à l'aide d'une pulsation. La pulsation est plus ou moins longue d'après la position du stick sur la télécommande. Comme on peut le voir ici, la pulsation jaune est de $1000\mu s$ et la rouge de $1500\mu s$.



Taux de rafraîchissement du signal 62.5Hz. donc 16ms.

La fréquence des pulsations est de 62,5Hz. L'enjeu est alors de lire la valeur de la durée de cette pulsation.

Dans la première version de notre programme, nous avons utilisé une fonction appelée *pulseIn()* pour lire la pulsation de la télécommande. Cette fonction détecte le passage d'un signal *LOW(0V)* vers un signal *HIGH(5V)*. À partir de ce moment elle mesure le temps jusqu'à ce que le signal repasse au *LOW*.

L'utilisation de cette fonction était correcte pour un certain nombre de commande, cependant les limitations de celle-ci sont très vite apparues.


```
void receiver_signal()
{
  ch1 = pulseIn(pinch1, HIGH, 25000); // lit la longueur de pulsation de la telecommande
  ch2 = pulseIn(pinch2, HIGH, 25000);
  ch3 = pulseIn(pinch3, HIGH, 25000);
  ch4 = pulseIn(pinch4, HIGH, 25000);
}
```

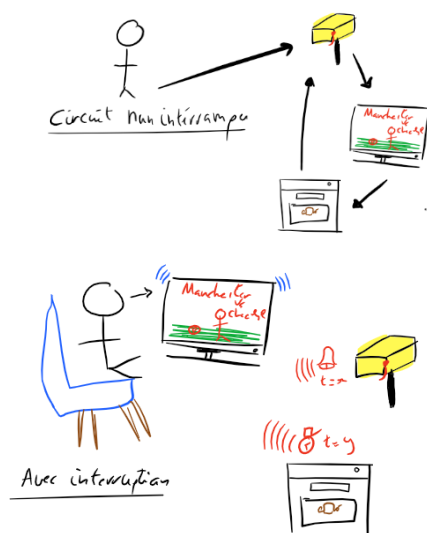
Tout d'abord comme stipulée sur le [site d'Arduino](http://www.arduino.cc) ; cette fonction, ne peut lire que le temps de passage de LOW a HIGH et donc si elle est appelée quand le signal est déjà sur HIGH, la fonction doit attendre que le signal revienne sur LOW pour ensuite mesurer le temps. Parfois cela, allié d'autre facteurs inconnus causaient d'énormes problèmes de calculs. Comme on peut le constater sur la vidéo suivante :

https://drive.google.com/file/d/0B7_JNnwiE-BqSmRVU0FhY012cGc/view?usp=sharing

Finalement ces limitations nous ont conduites à un changement radical dans la manière de programmer notre code. Une programmation en temps réel aura à être mise en place, une programmation qui prendra les pulsations de la télécommande comme prioritaire quand elles seront envoyées.

C'est ce que nous avons fait.

La carte Arduino est capable d'effectuer ce qu'on appelle le *PinChangeInterrupt*. À l'aide de la fonction *attachInterrupt*, certains pins (*fiches*) de l'Arduino se comportent comme des pins qui ont la capacité d'interrompre le programme en cours - lecture des valeurs du gyroscope ; calcul du PID - pour traiter en priorité les valeurs provenant de la télécommande. Pour comprendre ce système de manière simple considérons cet exemple :



Un homme regarde un match de football, mais il attend aussi une lettre du facteur, et il a mis un poulet à rôti au four. Pour ne rien manquer il fait des allers-retours entre le poulet, le facteur et le match.

On constate que ce circuit est très long et fait perdre beaucoup de temps et il se peut que le facteur passe pendant qu'il regarde le poulet rôti.

Considérons un nouveau circuit à présent comportant une horloge et une cloche. Dans cette situation l'homme peut regarder en permanence la télévision. Son sens auditif lui permettra d'interrompre son match pour aller voir le facteur, ou le poulet rôti.

On procèdera avec un raisonnement similaire avec le signal de la télécommande.

Sachant que pour qu'un drone fonctionne correctement ses ESC doivent être rafraichis tous les 4 *ms*. Le taux de rafraichissement de la télécommande étant de 16 *ms*,

L'interruption va se faire à chaque fois qu'il-y-aura un changement d'état au niveau de la télécommande. Ainsi toute les 16ms il-y-aura une interruption. Cela laisse à chaque fois 12ms au processeur pour effectuer les calculs pour les ESCs et le Gyroscope.

Voici le code :

```
attachInterrupt(digitalPinToInterrupt(pinch1),roulis_ch1 , CHANGE);
attachInterrupt(digitalPinToInterrupt(pinch2), tanguage_ch2, CHANGE);
attachInterrupt(digitalPinToInterrupt(pinch3), altitude_ch3, CHANGE);
attachInterrupt(digitalPinToInterrupt(pinch4), lacet_ch4, CHANGE);
```

La fonction a pour syntaxe *attachInterrupt (numéro de la fiche, fonction à appeler, ce qui cause l'interruption)*. La cause d'une interruption sera ici CHANGE. Donc tout passage de LOW a HIGH, ou l'inverse.

Dans notre code nous avons tout d'abord choisi de garder la fonction *pulseIn()*, en intégrant les interruption pour voir si cela était suffisant.

Ici la fonction *attachInterrupt* est liée à ces fonctions pour chaque canal :

```
void altitude_ch3()
{
  ch3 = pulseIn(pinch3, HIGH, 25000); //(le pin, La valeur de départ, temps maxi)
}

void roulis_ch1()
{
  ch1 = pulseIn(pinch1, HIGH, 25000);
}
void tanguage_ch2()
{
  ch2 = pulseIn(pinch2, HIGH, 25000);
}
void lacet_ch4()
{
  ch4 = pulseIn(pinch4, HIGH, 25000);
}
```

Cependant, on peut constater encore une fois que la télécommande ne répond pas à la vitesse qu'on aurait souhaitée. Cela est probablement lié au comportement même de la fonction *pulseIn()*. Il est nécessaire de trouver un nouveau moyen pour coder la lecture de la pulsation.

Voici un des moyens les plus simple et des plus efficace.


```

void altitude_ch3()
{
    if(mem_ch3 == 0 && digitalRead(pinch3)==HIGH){ //'1'==High , 0==Low si le booléen mem_ch est LOW
        mem_ch3=1 //et que le signal est HIGH on est sur la partie montante de la pulsation , on dit alors a mem_ch qu'on est HIGH
        temps_3=micros()// on prend ce temps de depart de high en considération
    }
    else if(mem_ch3 == 1 && digitalRead(pinch3)==Low){ // maintenant si le booléen est Haut et le signal est bas
        mem_ch3=0//on sait que on est sur la partie descendante de la pulsation. On dit donc ceci au booléen
        ch3=(micros()-temps_3) //On a donc parcouru toute la pulsation et donc on peut calculer sa durée.
    }
}

```

Il y'a deux variables dans cette fonction, *mem_ch3* qui est un booléen, et *temps_3* qui est une valeur a virgule flottante du temps. « && » correspond au *ET* logique.

Ce code attend de détecter un signal haut, dès qu'il en détecte un il prend le temps de départ de la pulsation, il change le booléen à HIGH, à ce moment la seconde condition n'est peut-être pas vérifiée (le signal est peut-être encore HIGH) , on sort donc de la fonction, par contre dès que le signal devient LOW, il-y-a une nouvelle interruption, cette même fonction est relancée, et, la première condition n'étant pas vraie contrairement à la seconde, on peut mesurer la différence entre les deux temps et connaître la taille de la pulsation. On notera que les variables à l'intérieur des fonctions d'interruptions sont dites volatiles.

```

volatile int ch1; //roulis
volatile int ch2; // tangage
volatile int ch3; // altitude
volatile int ch4; //lacet
volatile byte mem_ch1, mem_ch2, mem_ch3, mem_ch4;
volatile float temps_1, temps_2, temps_3, temps_4;

```

Avec ce code, on prévient juste la carte Arduino que ces variables peuvent et vont changer depuis plusieurs fonctions différentes ces variables sont donc « optimisées » en terme de

performance.

En programmation informatique, une variable volatile est une variable sur laquelle aucune optimisation de compilation n'est appliquée. Cela implique donc que les variables sont traitées nativement sans surcouche alourdissant le processus « d'assignation de la variable ». Nous n'allons pas nous attarder la dessus c'est beaucoup trop compliqué...

Lors de nos expériences, nous avons constaté que le signal n'est pas convenable : le nombre d'interruption est trop grand, ce qui cause un problème au niveau du processeur. En effet les pulsations que l'on envoie aux ESC n'est pas stable comme vous pouvez le voir sur la vidéo et la photo suivante.



Il est nécessaire encore une fois de ne pas passer par les bibliothèques d'interruption de la carte Arduino et d'utiliser les fonctions natives du processeur ATmega 2650. Car elles sont bien plus rapides. Pour cela nous avons consulté le [manuel](#) et nous avons découvert qu'il existe nativement un unique moyen de générer ce qu'on appelle un vecteur d'interruption. (C'est très similaire à la notion de fonction comme on le faisait précédemment).

15. External Interrupts

The External Interrupts are triggered by the INT7:0 pin or any of the PCINT23:0 pins. Observe that, if enabled, the interrupts will trigger even if the INT7:0 or PCINT23:0 pins are configured as outputs. This feature provides a way of generating a software interrupt.

The Pin change interrupt PC12 will trigger if any enabled PCINT23:16 pin toggles, Pin change interrupt PC11 if any enabled PCINT15:8 toggles and Pin change interrupts PC10 will trigger if any enabled PCINT7:0 pin toggles. PCMSK2, PCMSK1 and PCMSK0 Registers control which pins contribute to the pin change interrupts. Pin change interrupts on PCINT23:0 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than Idle mode.

Tout d'abord comme précisé dans le manuel, il est nécessaire de régler le Bit 0 ou PCIE0 à la valeur 1, tout changement ensuite sur les pins 7-0 de l'ATmega 2560 provoquera alors une interruption. Voir image ci-dessous :

15.2.5 PCICR – Pin Change Interrupt Control 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	

- **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 PCIE0 Interrupt Vector. PCINT7:0 pins are enabled individually by the PCMSK0 Register.

Digital pin 6 (PWM)	PH3 (OC4A)	15
Digital pin 7 (PWM)	PH4 (OC4B)	16
Digital pin 8 (PWM)	PH5 (OC4C)	17
Digital pin 9 (PWM)	PH6 (OC2B)	18
Digital pin 53 (SS)	PB0 (SS/PCINT0)	19
Digital pin 52 (SCK)	PB1 (SCK/PCINT1)	20
Digital pin 51 (MOSI)	PB2 (MOSI/PCINT2)	21
Digital pin 50 (MISO)	PB3 (MISO/PCINT3)	22
Digital pin 10 (PWM)	PB4 (OC2A/PCINT4)	23
Digital pin 11 (PWM)	PB5 (OC1A/PCINT5)	24

```

if (PINE & B00100000) {
  if (mem_ch3==0)
  mem_ch3=1;
  temps_3= temps_actuel;
}

```

Les pins PCint-0-7 correspondent sur l'Arduino méga au pins digital : 50,51,52,53.

Il est donc nécessaire de coder ceci, pour cela on doit écrire nativement évidemment, pas de fonction Arduino qui vont nous aider ici.

Il faut donc en premier régler le registre PCIE0 à 1 et ensuite il faut donc appeler le registre PCMSK0 pour les pins PCint entre 0 et 7. On n'utilise que 4 pins ici.

```

PCICR |= (1 << PCIE0); //Pour que le PinChangeInterruptRoutine sur les port PCIE0
PCMSK0 |= (1 << PCINT0); //Le pin PCINT0 (digital input 53) va crée l'interruption au changement d'état
PCMSK0 |= (1 << PCINT1); //le pin PCINT1 (digital input 52) va crée l'interruption au changement d'état
PCMSK0 |= (1 << PCINT2); //le pin PCINT2 (digital input 51) va crée l'interruption au changement d'état
PCMSK0 |= (1 << PCINT3); //le pin PCINT3 (digital input 50) va crée l'interruption au changement d'état

```

Le PCICR le pin « *Change Interrupt Communication Register* » doit alors contenir la fonction qui va lire les pulsations de la télécommande. En exécutant ce que l'on appelle un vecteur d'interruption.

Explication du nouveau code de lecture avec les vecteurs d'interruptions :

Les pins utilisés pour les différents canaux sont les pins digitaux 50 à 53, 52 pour le roulis, 51 pour le tangage, 53 pour la poussée et 50 pour le lacet. En regardant dans le mapping, ils font tous partie sur port registre B de PB3 à PB0.

Prenons l'exemple de la poussée sachant que le principe est le même pour les 3 autres canaux.

Pour récupérer le temps propre à la poussée, nous utilisons le code suivant :

Digital pin 53 (PWM)(RX1)	PB0 (SS/PCINT0)	19
Digital pin 52 (PWM)(SDA)	PB1 (SCK/PCINT1)	20
Digital pin 51 (PWM)(SCL)	PB2 (MOSI/PCINT2)	21
Digital pin 50	PB3 (MISO/PCINT3)	22

```

temps_actuel=micros()

////////// Pour le throttle (poussée)//////////
if(PINE & B00100000){ //Le pin de la poussée PB0 est-il HIGH?
    if (mem_ch3==0){ //La variable mémoire 3 est-elle à 0?
        mem_ch3=1; // Met la variable mémoire 3 à 1
        temps_3= temps_actuel;// Met temps_3 égale à temps_actuel
    }
}
else if(mem_ch3 == 1){ // Sinon si la variable mémoire 3 est à 1
    mem_ch3=0;//Met la variable mémoire 3 à 0
    altitude_in=(temps_actuel - temps_3); //Donne la pulsation de la poussé
}

```

Pour le premier if, nous testons si le pin de la poussée PB0 est HIGH, pour cela nous utilisons le bitwise &. PINB est, par exemple, de la forme 00001111 donc la poussée est HIGH. PINB & B00001111 va renvoyer 00000001 en binaire donc 1 en décimal, qui peut être considéré comme le booléen True (0 pour False), le 1^{er} if est donc vérifié.

Il va ensuite tester si la variable mémoire est à 0 (elle est au départ égal à 0 au lancement du programme), il va donc mettre cette variable mémoire à 1 et prendre le temps depuis le début du programme.

A la prochaine boucle, si le pin est toujours HIGH, le 1^{er} if sera vérifié mais pas le deuxième car nous avons déjà la variable mémoire égale à 1, enfin de compte le programme n'a rien fait dans cette boucle, et il continuera ainsi tant que le pin est à HIGH.

En effet le « else if » sinon si n'est pas testé car le 1^{er} est vérifié. Lorsque le pin sera à LOW, le 1^{er} if sera faux il va donc tester le else if qui sera vrai car nous avons mis au préalable la variable mémoire à 1. Il va donc prendre le temps actuel du programme moins le temps depuis lequel le pin est HIGH pour avoir la pulsation et remettre la variable mémoire à 0 pour la prochaine pulsation.

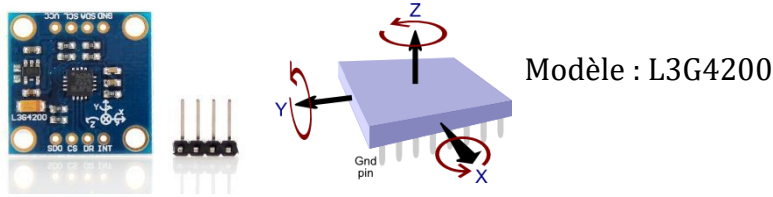
```

else if(mem_ch3 == 1){
    mem_ch3=0;
    altitude_in=(temps_actuel - temps_3);
}

```

Ce fonctionnement est répété 4 fois pour avoir les pulsations des 4 canaux indépendamment.

IV. Fonctionnement du gyroscope.



Contrairement à la télécommande qui fournit les données grâce à des pulsations, le gyroscope envoie directement les informations en binaire. Il communique avec la carte Arduino grâce au port SDA et SDL. Le gyroscope possède ainsi un cache (une petite mémoire où il stocke les informations). La réelle difficulté ici, c'est de savoir comment lire correctement ce cache pour obtenir les informations. Comme c'est toujours le même code que l'on a obtenu en ligne. On ne s'est pas trop intéressé à la compréhension en profondeur de celui-ci. Par contre, il convient évidemment de le comprendre en substance et de l'expliquer.

A. La configuration initiale

```
Wire.beginTransmission(105);           //Démarrer la communication avec le gyroscope
Wire.write(0x20);                       //On veut écrire sur le registre 1 (20 hex).
Wire.write(0x0F);                       //configurer le bit registre a 00001111 ( cela reveille le gyroscope )
Wire.endTransmission();                 //Termine la communication avec le gyroscope.

Wire.beginTransmission(105);           //Démarrer la communication avec le gyroscope
Wire.write(0x23);                       //On veut écrire sur le registre 4 (23 hex).
Wire.write(0x90);                       //Configure le bit registre a 10010000 (cela permet de mettre à jour le cache en permanence et configuration 500degrees par secondes )
Wire.endTransmission();                 //Termine la communication avec le gyroscope.

delay(250);
```

Il y a deux enjeux ici : trouver le registre d'écriture correct pour effectuer les commandes suivantes : réveiller le gyroscope et configurer le nombre de degrés par seconde que l'on veut obtenir. Ainsi systématiquement, nous initialisons la communication avec le gyroscope, nous écrivons sur le bit correspondant à la commande que nous voulons réaliser.

Une fois initialisé, le gyroscope peut transmettre les données. Ensuite comme la lecture des valeurs doit être effectuée de manière périodique, on met le code de lecture des informations dans une fonction que l'on appelle *gyro_signal()*.

```

void gyro_signal() {
1
2   Wire.beginTransmission(105);           //Commence la communication avec le gyroscope
3   Wire.write(168);                       //On commence la lecture au registre 28h, celui de la position du langage
4   Wire.endTransmission();               //On termine la communication
5   Wire.requestFrom(105, 6);             //On fait la requete des 6 octets du gyro (deux pour chaque axe)
6   while(Wire.available() < 6);          //On attend que les octets arrivent
7
8
9   lowByte = Wire.read();
10  highByte = Wire.read();
11  gyro_pitch=((highByte<<8)|lowByte);
12  gyro_pitch*=-1;
13  if(cal_int == 2000)gyro_pitch -= gyro_pitch_cal; |
14
15  lowByte = Wire.read();
16  highByte = Wire.read();
17  gyro_roll=((highByte<<8)|lowByte);
18  gyro_roll*=-1;
19  if(cal_int == 2000)gyro_roll += gyro_roll_cal; // + tout depend du offset g n ral de notre drone
20
21  lowByte = Wire.read();
22  highByte = Wire.read();
23  gyro_yaw=((highByte<<8)|lowByte);
24  gyro_yaw*=-1;
25  if(cal_int == 2000)gyro_yaw -= gyro_yaw_cal;
26  // Attention il faut tester pour inverser les axes on fait gyro_yaw*=-1
27
28 }

```

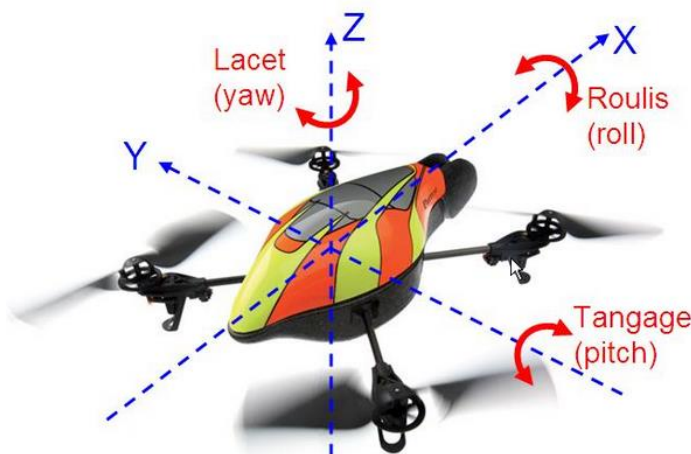
B. Calibration

Comme tout gyroscope qui se doit, il est n cessaire de calibrer celui-ci. En effet en position immobile, le gyroscope produit un petit bruit de l'ordre de 30 degr s par minutes, donc 0,5 degr  par seconde. Ce n'est  videmment pas n gligeable. Pour contourner le probl me, il suffit juste d'effectuer un processus de calibration, c'est- -dire de relever   peu pr s 2000 valeurs du gyroscope, en faire la somme et ensuite de diviser le tout par 2000. Ensuite, on va soustraire cette valeur aux valeurs du gyroscope.

```

//Calibration avant vol gyro
for (cal_int = 0; cal_int < 2000 ; cal_int++){           //Accumule 2000 valeurs pour la calibration
    if(cal_int % 15 == 0)digitalWrite(52, !digitalRead(52)); //Fait clignoter la LED pour indiquer la calibration.
    gyro_signal();                                         //Appelle la fonction gyroscope
    gyro_roll_cal +=gyro_roll;                             // Fait la somme sur 2000 valeurs sur tout les axes
    gyro_pitch_cal +=gyro_pitch;
    gyro_yaw_cal += gyro_yaw;
    //pour  viter une d calibration des ESC on envoi encore une pulsation de 1000us
    PORTH |= B01111000;
    delayMicroseconds(1000);
    PORTH &= B10000111;
    delay(3);
}
//Maintenant on a obtenu 2000 valeurs on peut faire la moyenne
gyro_roll_cal /= 2000;                                     //Divise le total par 2000
gyro_pitch_cal /= 2000;
gyro_yaw_cal /= 2000;

```



V. Principe de fonctionnement d'un correcteur PID

A. Introduction sur le correcteur PID

Le correcteur PID (Produit Intégrale Dérivée) a pour fonction de corriger les erreurs d'asservissement du drone. Sans lui, notre drone pourrait tourner sur lui-même ou se pencher dès que l'on veut le faire décoller. En effet, les 4 moteurs ne réagissent pas tous au même moment et rendent donc le drone instable sans le correcteur PID. L'objectif du correcteur est de maintenir les valeurs du gyroscope égales à celles du récepteur (qui traduit ce qu'on demande de faire au drone).

Comme son nom l'indique, ce correcteur renvoie une valeur qui correspond à la somme d'un produit, d'une intégrale, et d'une dérivée sur la différence entre la valeur demandée avec la télécommande et la valeur mesurée par le gyroscope pour chaque axe:

$$P_{output} = (val_{gyro} - val_{recepteur}) \times P_{gain}$$

$$I_{output} = I_{output_{pcdt}} + ((val_{gyro} - val_{recepteur}) \times I_{gain})$$

$$D_{output} = (val_{gyro} - val_{recepteur} - (val_{gyro_{pcdt}} - val_{recepteur_{pcdt}})) \times D_{gain}$$


$$PID_{output} = P_{output} + I_{output} + D_{output}$$

Le drone peut avoir trois mouvements différents : le roulis (roll), le tangage (pitch), et le lacet (yaw). Il y a donc 3 valeurs à calculer pour que le drone soit parfaitement stabilisé. Voici les rôles des différentes composantes de la valeur renvoyée par notre fonction PID:

Produit :

Il permet une première stabilisation du drone, mais il y aura des oscillations qui ne pourront être annulé par le produit seul. Augmenter le gain permet un meilleur temps de réponse.

Intégrale :

Etant la somme des erreurs, elle permet un redressement du drone rapide : par exemple, si le drone se penche sur la gauche violemment, l'intégrale va immédiatement prendre une très grande valeur, et donc corriger l'angle du drone rapidement, c'est un complément du produit, elle augmente la réactivité.

Dérivée :

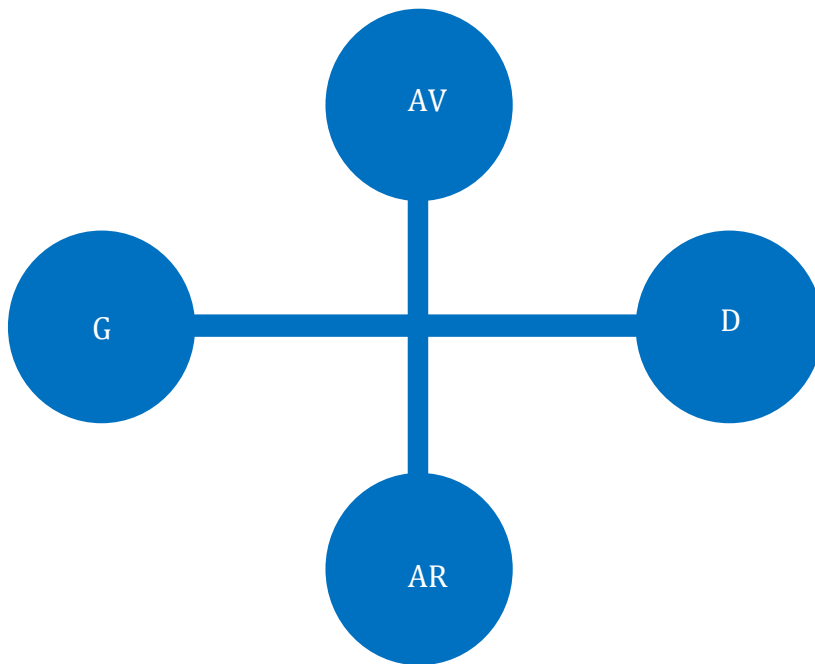
C'est la dernière valeur à calculer, elle permet d'amortir la compensation du mouvement non-voulu du drone, pour éviter une sur-correction. Plus la dérivée de la vitesse angulaire du drone est proche de 0, moins la dérivée entre en compte, et moins le mouvement de compensation du drone est rapide.

Ci-dessous, un graphe des fonctions produit (en vert), intégrale (en jaune), et dérivée (en bleu) avec des gains de 1 en fonction de l'erreur (en rouge).



Ici, on voit nettement que le produit compense exactement l'erreur, l'intégrale augmente la vitesse de compensation si l'erreur continue d'augmenter, et la dérivée intervient pour amortir la compensation en fin de correction.

Voici, avec un schéma du drone, les calculs à effectuer sur chaque moteur pour stabiliser le drone :



On a:

$$\begin{aligned}
 mot_{AV} &= throttle - pid_{output_{pitch}} + pid_{output_{yaw}} \\
 mot_D &= throttle - pid_{output_{roll}} - pid_{output_{yaw}} \\
 mot_{AR} &= throttle + pid_{output_{pitch}} + pid_{output_{yaw}} \\
 mot_G &= throttle + pid_{output_{roll}} - pid_{output_{yaw}}
 \end{aligned}$$

B. Etalonnage des valeurs du gain du correcteur PID :

Nous avons parlé plus haut d'une variable particulière qui intervient dans chaque équation : le gain. Nous allons voir dans cette partie comment déterminer ces valeurs :

- 1). Régler le gain du produit du lacet à 3 et celui de l'intégrale à 0.02, pour éviter que le drone tourne sur lui-même pendant le réglage du lacet et du tangage.
- 2). Régler le gain de la dérivée du tangage et du roulis à 3.
- 3). En augmentant la vitesse des pales, augmenter progressivement la valeur du gain de la dérivée jusqu'à obtenir une sur-réaction.
- 4). Baisser à nouveau la valeur jusqu'à ce que le drone réagisse de manière douce. Baisser cette valeur de 25% : c'est la valeur à garder pour ces gains.
- 5). Réglage du gain du produit du tangage et du roulis : à augmenter par pas de 0.2. A cette étape, il est possible de faire voler le drone.
- 6). Augmenter la valeur du gain sur le produit jusqu'à ce que le drone surcompense le mouvement, baisser cette valeur de 50% : c'est notre valeur pour ces gains.

7). Réglage des gains sur l'intégrale : augmenter la valeur par tranche de 0.01 jusqu'à ce que le drone oscille doucement. Baisser cette valeur de 50% : c'est notre valeur pour ces gains.

C. Explication du code :

Avant de calculer la correction à apporter, il faut convertir nos entrées (la commande et la valeur relevée par le gyroscope) en degrés par secondes.

Tout d'abord, la télécommande :

```
tel_pitch_dps = tel_yaw_dps = tel_roll_dps = 0;
if(roll_in > 1512)
{
    tel_roll_dps = (roll_in - 1504)/3.15;
}
else if(roll_in < 1496)
{
    tel_roll_dps = (roll_in - 1484)/3.15;
}
```

On commence par mettre la valeur à 0 (valeur par défaut). Les valeurs 1504 et 1484 (lignes 2 et 6) correspondent à la durée de la pulsation en ms envoyée par la télécommande, on considère qu'entre 1496 et 1512 la valeur envoyée est 0, pour éviter que le drone tourne alors que ce n'est pas demandé. Dans le premier if, on a un angle positif, qui correspond à un mouvement vers la droite (pour le roulis) on a une pulsation de 2004 ms envoyée au maximum donc le roulis maximum est :

$$tel_{roll_{dps}} = \frac{2004 - 1512}{3,15} = 156 \text{ degrés par seconde}$$

Ce qui est inférieur à 180 pour éviter que le correcteur soit trop nerveux. Pour augmenter la réactivité du correcteur, il faut diminuer la valeur du dénominateur, et l'augmenter pour ralentir encore la correction.

Nous avons la même chose en degrés négatifs pour un mouvement vers la gauche (second if). La valeur centrale étant 1504 ms, et l'écart de plus ou moins 500 ms, notre mesure en degrés par seconde est comprise entre -156 et +156.

La conversion pour le tangage et le lacet sont identiques, elles ne seront donc pas détaillées ici, tous les angles étant compris entre -156 et +156 degrés par seconde.

Ensuite vient la conversion des valeurs du gyroscope :

```
gyro_roll_ds = (gyro_roll_ds * 0.8) + ((gyro_roll / 57.14286) * 0.2);
gyro_pitch_ds = (gyro_pitch_ds * 0.8) + ((gyro_pitch / 57.14286) * 0.2);
gyro_yaw_ds = (gyro_yaw_ds * 0.8) + ((gyro_yaw / 57.14286) * 0.2);
```

Ici, c'est un peu plus compliqué : il faut tout d'abord trouver dans le manuel du gyroscope à quoi correspond une augmentation de 1 de la valeur envoyée par le gyroscope à la sensibilité du gyroscope correspondante. Ici, 1 correspond à 17,5 mdps/s (milli degrés par seconde). On a :

$$\frac{1}{17,5 \times 10^{-3}} = 57,14286 \text{ secondes par degré}$$

Enfin, il suffit de diviser notre valeur mesurée par celle-ci-dessus pour la convertir en degrés par seconde. Pour gagner en précision sur ces valeurs (le gyroscope est très sensible aux moindres perturbations), nous avons ajouté un filtre qui prends 80% de la valeur précédente en degrés par seconde, et y ajoute 20% de la valeur convertie qui vient d'être mesurée. Cela permet d'éviter des variations très importantes d'une mesure à l'autre, et donc améliorer la stabilité du drone.

Ensuite, le calcul de la correction peut être effectué :

```
//calcul des différentes erreurs:
roll_erreur = gyro_roll_ds - tel_roll_dps;
pitch_erreur = gyro_pitch_ds - tel_pitch_dps;
yaw_erreur = gyro_yaw_ds - tel_yaw_dps;

//calcul de la correction à apporter au roulis:
val_P_roll = roll_erreur*P_gain_roll;
val_I_roll = val_I_roll_pcdt + roll_erreur*I_gain_roll;
val_D_roll = (roll_erreur - roll_erreur_pcdt)*D_gain_roll;
/*
if(val_I_roll > max_roll)
{
    val_I_roll = max_roll;
}
else if(val_I_roll < max_roll * -1)
{
    val_I_roll = max_roll * -1;
}
*/
val_correction_roll = val_P_roll + val_I_roll + val_D_roll;
if (val_correction_roll > max_roll)
{
    val_correction_roll = max_roll;
}
else if(val_correction_roll < -1*max_roll)
{
    val_correction_roll = -1*max_roll;
}

val_I_roll_pcdt = val_I_roll;
roll_erreur_pcdt = roll_erreur;
```

On commence par calculer les erreurs (différence entre la mesure du gyroscope et la commande envoyée par la télécommande).

Ensuite, on calcule la correction apportée par les différentes parties : le produit, l'intégrale, et la dérivée. Le produit est simplement l'erreur mesurée multipliée par le gain. L'intégrale représente l'aire sous la courbe de l'erreur, c'est donc la valeur de l'erreur mesurée (similaire à celle du produit), à laquelle on ajoute l'erreur précédente : avec l'erreur de la correction sur l'axe des ordonnées et l'indice de la mesure sur l'axe des abscisses, on a bien entre 2 mesures :

$I = (I_{\text{précédent}} + I_{\text{actuelle}}) \times 1$ où 1 représente le nombre de mesures qui séparent nos 2 valeurs.

Enfin, la dérivée représente par définition la variation de l'erreur entre deux mesures (c'est un taux de variation), sans prendre en compte le gain, on a bien :

$$D = \frac{(val_{\text{actuelle}} - val_{\text{précédente}})}{1}$$

Ensuite, il faut vérifier que lorsqu'on somme les trois valeurs, la valeur de correction finale du PID est inférieure à notre seuil (ici fixé à ± 400 degrés par seconde). C'est le rôle du if final.

Pour finir, il faut sauvegarder la valeur actuelle de la dérivée et de l'intégrale, elles resserviront pour le prochain calcul de correction.

Enfin, il faut adapter la vitesse des moteurs :

```
mot_AV = throttle - val_correction_pitch + val_correction_yaw;
mot_D = throttle - val_correction_roll - val_correction_yaw;
mot_AR = throttle + val_correction_pitch + val_correction_yaw;
mot_G = throttle + val_correction_roll - val_correction_yaw;
```

C'est l'application directe de ce qui a été décrit plus haut.

Conclusion : Finalement le correcteur PID est l'élément clef qui permet de faire fonctionner l'ensemble du système. Sans lui le drone ne serait tout simplement pas pilotable.

IV. Assigner ou relever la tension d'un pin

Maintenant que nous avons la pulsation de la télécommande et le PID, nous avons donc les pulsations exactes qu'il faut fournir aux moteurs. L'enjeu ici est de fournir les pulsations propres à chaque moteur tout en ayant un temps d'exécution le plus faible possible. Pour envoyer une pulsation, il faut apprendre à manipuler la tension d'un pin.

A. Manipuler les pins en utilisant la librairie Arduino

La façon la plus commune pour assigner ou relever la tension d'un pin est d'utiliser la librairie de la carte Arduino. En utilisant la librairie arduino, il nous faut tout d'abord configurer le pin en question grâce à la fonction `pinMode()` avec pour paramètres : le numéro du pin en question et son mode, il faut configurer le pin en OUTPUT pour assigner une tension sortante, et pin en INPUT, et pour une relever tension entrante.

```
pinMode(pin,OUTPUT/INPUT)
```

Une fois le pin réglé en OUTPUT, il suffit d'utiliser la fonction `digitalWrite(pin,HIGH /LOW)` pour assigner une tension au pin, les pins de l'arduino ne pouvant émettre que des tensions de 5V, nous avons HIGH pour 5V et LOW pour 0V. Et pour relever la tension d'un pin, on utilise la fonction `digitalRead(pin)` qui retourne soit HIGH soit LOW. Nous pouvons donc utiliser `digitalWrite()` pour faire tourner les moteurs et `digitalRead()` pour les obtenir les signaux de la télécommande.

Exemple :

```
int pinMoteur = 6;           // Moteur connecté au pin 6

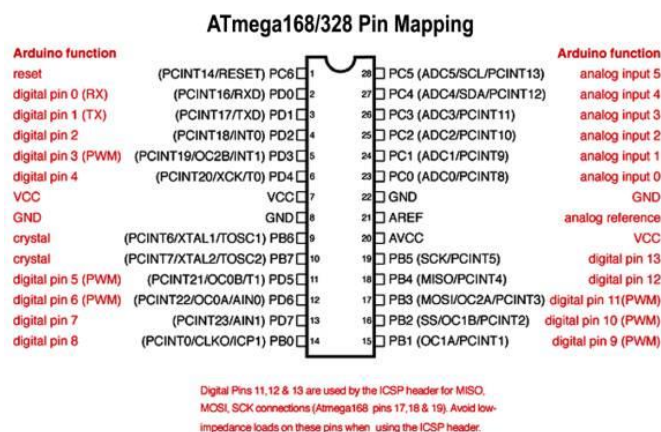
void setup()
{
  pinMode(pinMoteur, OUTPUT); // Met le pin en OUTPUT
}

void loop()
{
  digitalWrite(pinMoteur, HIGH); // Met le pinMoteur à HIGH
  delayMicroseconds(1000);       // On attend 1000microsecondes
  digitalWrite(pinMoteur, LOW);  // Met le pinMoteur à LOW
  // Permet d'envoyer une pulsation de 1000 microsecondes au moteur
}
```


L'inconvénient de cette technique est le temps d'exécution de ces fonctions. En effet nous devons envoyer une pulsation aux moteurs toutes les 4ms. Si le temps de calcul du PID, l'obtention des signaux de la télécommande et l'envoi des pulsations est supérieur à 4ms, nous n'enverrons pas suffisamment de pulsations aux moteurs. Il existe d'autres façons de manipuler les pins notamment en utilisant les ports registres

B. Les ports registre

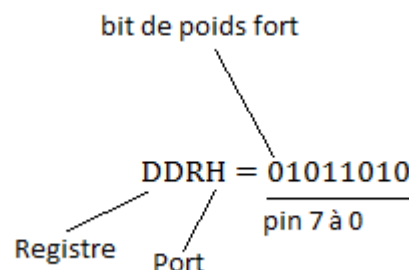
1. Principe



Les ports registre permettent de manipuler directement les pins de la carte Arduino sans passer par la librairie Arduino. Les cartes Arduino sont composées de plusieurs ports, 3 pour la UNO et de 11 pour la MEGA, chacun de ces ports est composé de 8 pins numérotés de 0 à 7. Chacun ces ports sont contrôlés par 3 registres : DDR pour la direction INPUT ou OUTPUT, PORT pour contrôler les pins OUTPUT, et PIN pour relever les valeurs des pin INPUT. Ce qui implique que les registres DDR et PORT sont des registres pour lire et écrire et que PIN est un registre uniquement pour lire.

Les registres DDR, PORT, et PIN sont des variables binaires composés de 8 bits pour 8pins, chaque bit représente 1 pin du port, le bit de poids fort (celui tout à gauche) signifie le PIN, le 7 du port puis plus nous allons à droite de celui-ci plus numéro du pin décroît.

Exemple :



Cela signifie que les pins du port H, 7,5,2,0 sont INPUT et 6,4,3,1 sont OUTPUT

2. Les opérateurs

Les bitwises sont des opérateurs que nous utilisons pour manipuler les variables en binaires, ils proviennent du C++, il en existe 3 principaux :

a. Le Bitwise AND (&)

Il est utilisé entre 2 valeurs, et agit sur chaque position des bits, si les 2 bits à la même position pour les 2 valeurs valent 1 alors le résultat du bit à cette position est 1 sinon c'est 0.

Exemple :

Valeur 1	0	1	0	1	1	1	0	0
Valeur 2	1	0	1	1	0	0	1	0
Résultat	0	0	0	1	0	0	0	0

b. Le bitwise OU (|)

Il est utilisé entre 2 valeurs, et agit sur chaque position des bits, si 1 des 2 bits à la même position pour les 2 valeurs vaut 1 alors le résultat du bit à cette position est 1 sinon c'est 0.

Exemple :

Valeur 1	0	1	0	1	1	1	0
Valeur 2	1	0	1	1	0	0	0
Résultat	1	1	1	1	1	1	0

c. Le bitwise OU EXCLUSIF (^)

Il est utilisé entre 2 valeurs, et agit sur chaque position des bits, si les 2 bits à la même position pour les 2 valeurs valent 1 alors le résultat du bit à cette position est 0, si 1 des 2 bits à la même position pour les 2 valeurs vaut 1, alors le résultat du bit à cette position est 1 sinon c'est 0.

Exemple :

Valeur 1	0	1	0	1	1	1	0
Valeur 2	1	0	1	1	0	0	0
Résultat	1	1	1	0	1	1	0

Nous pouvons donc utiliser ces opérateurs sur les registres DDR, PORT, PIN

3. Temps d'exécution

Nous avons vu 2 façons de manipuler les pins, nous allons maintenant comparer leur temps d'exécution. Nous avons fait 2 tests sur une Arduino 16Mhz et nous avons branché un des pins à un oscilloscope, le premier test pour mettre directement des variables au pin et le deuxième pour inverser sa valeur, avec des codes dans des boucles infinies pour avoir des sinusoïdes carrées nous avons donc pu avoir la période et la fréquence. Et ainsi déterminer le nombre de cycle nécessaire au processeur pour exécuter la commande. Le nombre de cycle est déterminé en divisant la période divisée par 2, car nous faisons dans chaque code 2 commandes, par la fréquence du processeur c'est à dire ici 16Mhz.

Le premier test pour mettre directement des variables au pin avec ces codes :

- digitalWrite(pin,LOW) ; digitalWrite(pin,HIGH)
- PORTB &= B00000000 ; PORTB |=B00000001.

Nous avons eu ces résultats :

Méthode	Frequence(kH)	Periode(nS)	½ Periode(Ns)	Cycles
digitalWrite()	142	7042	3521	56
PORT=	4000	250	125	2

Comme vous pouvez le voir, digitalWrite prend 56 cycles alors que PORT n'en prend que 2 il est donc 28 fois plus rapide.

Passons au deuxième test, pour inverser les valeurs avec ces codes :

- digitalWrite(pin, !digitalRead(pin)) ;
- PORTB^=B10000000 ;

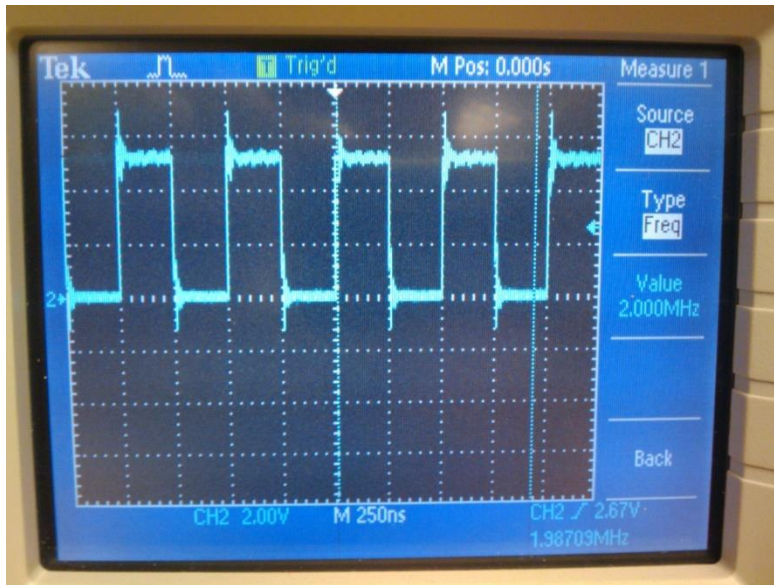
Nous avons eu ces résultats :

Méthode	Frequence(kH)	Periode(nS)	½ Periode(Ns)	Cycles
digitalWrite()	66	15151	7575	121
PORT^=	2636	379	189	3

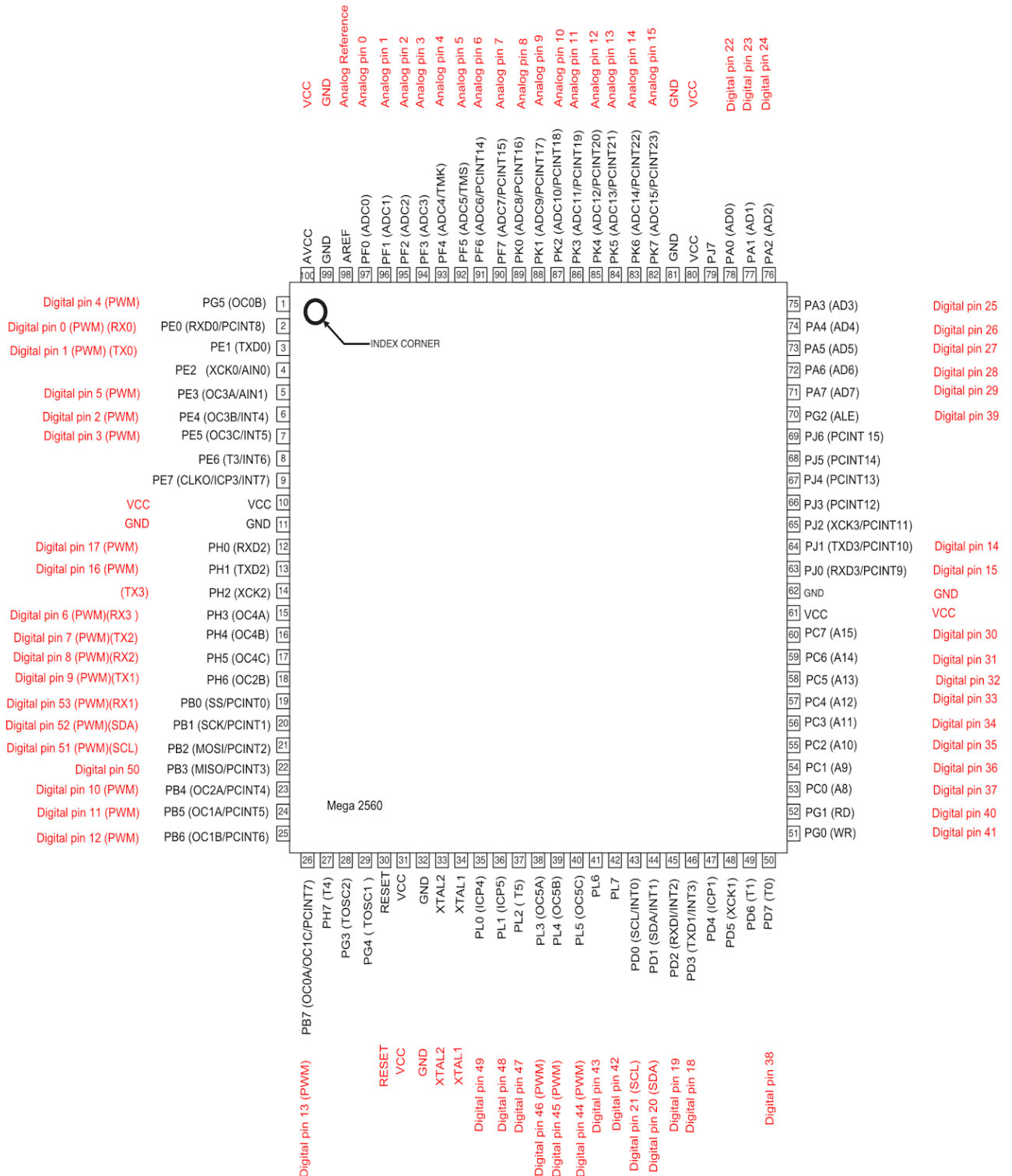
Cette fois-ci digitalWrite prend 121 cycles pour s'exécuter ce qui est énorme contre seulement 3.

Donc comme nous pouvons le voir, les fonctions Arduino prennent considérablement plus de temps pour s'exécuter que lorsqu'on manipule directement les ports. Cela est dû au fait que les fonctions Arduino font beaucoup de vérifications pour voir s'il y a des erreurs pour être sûr que le pin soit correctement configuré.

Donc les fonctions Arduino sont beaucoup plus lentes mais plus facile à manipuler. Pour notre drone, il nous est impossible de les utiliser car elles prennent trop de temps pour s'exécuter.







C. Utilisation des ports registres



Nous allons utiliser le mapping de l'Arduino atmega 2560 pour utiliser les ports.

1. Les moteurs

Digital pin 6 (PWM)(RX3)	PH3 (OC4A) 
Digital pin 7 (PWM)(TX2)	PH4 (OC4B) 
Digital pin 8 (PWM)(RX2)	PH5 (OC4C) 
Digital pin 9 (PWM)(TX1)	PH6 (OC2B) 

Pour les moteurs nous avons pris les digitaux pins 6, 7, 8 et 9, donc en regardant sur le mapping Atmega 2560 nous voyons que ces pins font partis du port H et sont numérotés de 3 à 6.

Les pins sont par défaut en INPUT, pour assigner ces pins en OUTPUT, nous avons utilisé :

DDRH |= B01111000

H pour le port H, l'opérateur « | » pour ne modifier que les pins du port H de 3 à 6 en OUTPUT car si on ne mettrait qu'un « = » on risque de modifier les autres pins.

Pour envoyer une pulsation de 1ms sur tous les moteurs en même temps :

```
PORTH |= B01111000;
delayMicroseconds(1000);
PORTH &= B10000111;
```

Tout d'abord il faut mettre les pins en HIGH en modifiant le registre comme ceci `PORTH |= B01111000`; l'opérateur « | » pour ne modifier que les pins du port H de 3 à 6 et laisser les autres pins à leur état initial.

Il faut ensuite attendre 1ms avant de remettre les pins en LOW donc nous utilisons la fonction `delayMicroseconds(1000)` de la librairie Arduino. Puis pour remettre les pins en LOW, `PORTH &= B10000111`; pour mettre forcément les pins 3 à 6 à 0 et laisser les autres pins à 1 s'ils le sont.

Il faut maintenant envoyer des pulsations quelconques entre 1ms et 2ms à chacun des moteurs pour pouvoir manipuler le drone.

Pour cela nous utilisons le code suivant :

```
while(micros() - loop_timer < 4000); //Crée une boucle infini qui attend 4ms
loop_timer = micros(); //Met le temps pour la prochaine boucle

PORTH |= B01111000; //Met les 4 moteurs pin 6,7,8,9 à HIGH.
timer_channel_1 = mot_AV + loop_timer; //Calcule le temps avant de mettre le moteur AV à LOW.
timer_channel_2 = mot_D + loop_timer; //Calcule le temps avant de mettre le moteur D à LOW..
timer_channel_3 = mot_AR + loop_timer; //Calcule le temps avant de mettre le moteur AR à LOW.
timer_channel_4 = mot_G + loop_timer; //Calcule le temps avant de mettre le moteur G à LOW.

while(PORTH >= 8){ //Reste dans cette boucle tant que les 4 moteurs ne sont
    esc_loop_timer = micros(); //Regarde le temps actuel
    if(timer_channel_1 <= esc_loop_timer)PORTH &= B11110111; //Met digital pin 6 à LOW si le temps est expiré
    if(timer_channel_2 <= esc_loop_timer)PORTH &= B11101111; //Met digital pin 7 à LOW si le temps est expiré
    if(timer_channel_3 <= esc_loop_timer)PORTH &= B11011111; //Met digital pin 8 à LOW si le temps est expiré
    if(timer_channel_4 <= esc_loop_timer)PORTH &= B10111111; //Met digital pin 9 à LOW si le temps est expiré
}
```


La fonction `micros()` retourne le temps depuis lequel l'Arduino a lancé le programme. Comme expliqué précédemment, nous devons envoyer aux moteurs une pulsation toutes les 4ms. `while(micros() - loop_timer < 4000);`

Pour cela on crée une boucle `while` mais qui ne fait rien, qui a pour condition de rester dans celle-ci tant que le temps depuis lequel nous avons commencé à lancer le programme (`micros()`) moins le temps à partir duquel nous avons commencé à lancer la pulsation précédente (`loop_timer`, à partir du début du programme) est inférieur à 4ms (pour la première pulsation `loop_timer = 0`).

```
loop_timer = micros();
```

Nous réaffectons ensuite à `loop_timer` avec `micros()` et nous envoyons une tension de 5V à tous les moteurs. Puis, nous calculons pour chaque moteur (`timer_channel_...`) le temps depuis le début du programme (`loop_timer` plus la pulsation a envoyé aux différents moteurs.

```
PORTH |= B01111000;
```

```
while(PORTH >= 8)
```

`PORT` comme expliqué précédemment est un nombre binaire, lorsque les 4 pins des moteurs sont HIGH, nous avons `PORT=01111000` en binaire donc 120 en décimal, à chaque fois qu'un moteur devient LOW un des 1 devient 0, donc le plus petit nombre décimal correspondant à `PORT` pour un seul moteur toujours HIGH est 8 (00001000). Nous utilisons donc la boucle `while(PORT >= 8)`, (ie Tant que tous les pins des moteurs ne sont pas à LOW), pour nous assurer que les pulsations ont bien été envoyées aux moteurs.

```
while(PORTH >= 8){
```

Dans cette boucle `while`, nous réaffectons à chaque fois la variable `esc_loop_timer` égale à `micros()`, et nous testons, par exemple pour le moteur 1, si le `timer_channel_1` donc si le temps depuis le début du programme (à l'instant où nous mettons les moteurs HIGH) plus la pulsation est inférieur au temps depuis le programme au début de la boucle `while`. Comme cette boucle `while` est infini, si un des moteurs est toujours HIGH, en l'occurrence pour le moteur 1 dans cet exemple, `esc_loop_timer` croît toujours alors que `timer_channel_1` est constante.

```
esc_loop_timer = micros();
```

```
if(timer_channel_1 <= esc_loop_timer) PORTH &= B11110111;
```

A partir d'un moment `esc_loop_timer` sera supérieur à `timer_channel_1` et nous changeons le port registre H à l'aide du bitwise `&` qui va changer uniquement le PH3 correspondant au pin du moteur 1 en LOW. Le bitwise `&` va mettre obligatoirement le PH3 en 0 (`1&0 = 0`) et garder les autres pins à leur état actuel (si c'est un 1 nous avons `1&1=1`, si c'est un 0, `0&1=0`).

En effet, le temps écoulé entre le déclenchement du HIGH et le du LOW est exactement la pulsation, car nous avons attendu que le temps du programme dépasse celui de la pulsation plus le temps du programme où nous avons mis le pin à HIGH. Nous utilisons ensuite le même principe pour les 4 moteurs dans la même boucle `while` qui va au bout d'un moment envoyé les pulsations propres à chaque moteur.

Pour envoyer la prochaine pulsation, nous attendons que le temps du programme dépasse celui à partir duquel nous avons commencé à émettre la pulsation précédente de 4ms.

VI. Contrôle des moteurs avec la librairie servo.

Nous avons trouvé une librairie, très simple à utiliser, pour envoyer des pulsations aux moteurs sans perdre de temps face aux ports registres : la librairie *Servo*.

Il faut donc inclure la librairie Servo dans notre code Arduino comme ceci :

```
#include <Servo.h>
```

La librairie Servo utilise la programmation orientée objet.

Un objet en programmation est un mélange de variable et de fonctions :

Par exemple un profil Facebook est un objet, il contient plusieurs variables : nom, prénom, date d'inscription, groupes auxquels vous appartenez etc... Mais il contient également des fonctions par exemple pour changer votre mot de passe, votre nom, votre prénom, votre date de naissance, souscrire à un groupe, etc...

Pour pouvoir créer votre profil Facebook donc votre objet, on appelle cela un constructeur qui va créer automatiquement votre profil en fonction des informations que vous avez entré (nom, prénom...).

Dans notre cas, chaque moteur va être un objet contenant plusieurs attributs et des fonctions pour les manipuler, son constructeur est Servo.

```
Servo myservoAV;  
Servo myservoAR;  
Servo myservoG;  
Servo myservoD;
```

Dans notre objet, nous avons plusieurs fonctions : celles que nous allons utiliser sont :

- Attach() // Pour renseigner le digital pin du moteur
- writeMicroseconds() // Pour envoyer une pulsation en microsecondes

Nous avons donc à mettre pour indiquer le numéro des pins des moteurs :

```
int pin_servoAV = 6;  
int pin_servoAR = 8;  
int pin_servoG = 9;  
int pin_servoD = 7;
```

Et nous devons indiquer à chaque objet ou moteur leur pin respectif en utilisant objet.attach(pin) :

```
myservoAV.attach(pin_servoAV);  
myservoAR.attach(pin_servoAR);  
myservoG.attach(pin_servoG);  
myservoD.attach(pin_servoD);
```

Puis ils suffit d'envoyer les pulsations aux moteurs
`objet.writeMicroseconds(temps_microsecondes)`

```
myservoAV.writeMicroseconds (mot_AV);  
myservoD.writeMicroseconds (mot_D);  
myservoAR.writeMicroseconds (mot_AR);  
myservoG.writeMicroseconds (mot_G);
```

Il n'est même pas nécessaire de faire des delays pour s'assurer d'envoyer une pulsation toutes les 4ms la librairie étant faite pour contrôler les moteurs, elle sait quand il faut envoyer les pulsations. Cette librairie n'est pas seulement simple à utiliser, elle supprime les pulsations erronées dû aux interférences, c'est-à-dire que si une pulsation est trop éloignée de la précédente, elle l'annule.

Conclusion

Nous avons vu différentes façons d'envoyer les pulsations aux moteurs, soit en manipulant directement les pins liés au différent moteur par la bibliothèque Arduino ou en utilisant les ports registres soit, tout simplement, en utilisant une librairie spécialisée pour les moteurs, la librairie Servo. L'enjeu ici était de trouver la façon la plus rapide, simple et efficace, la librairie Servo remporte ce concours haut la main.

Conclusion

Nous l'avons vu, les enjeux derrière la réalisation d'un drone pour un ingénieur en électronique sont très complexes. Il a été nécessaire de construire des outils mathématiques et physique pour décrire le mouvement du quadrirotor et asservir efficacement le drone : matrices de rotation, normes dans un espace vectoriel, espace vectoriel euclidien... Une fois nos conditions matérielles imposées après définition du cahier des charges, nous avons dû nous renseigner plus profondément sur notre matériel à disposition. La construction du drone a demandé beaucoup d'ingéniosité technique pour limiter les vibrations et garder une grande solidité au drone, limiter l'emplacement du matériel et garder un bon refroidissement et restreindre les interférences. La partie software a été des plus complexes : nombreuses étaient les limitations imposées par la partie technique (cadence du processeur de la carte Arduino Mega, temps de rafraichissement des ESC, etc..), et là encore il a fallu faire de nombreux calculs pour optimiser l'algorithme et la complexité de celui-ci en se rapprochant de plus en plus d'une programmation de bas niveau.

Nous nous sommes heurtés à de nombreux problèmes à chaque étape de la réalisation, sans doute beaucoup auraient pu être évités en ayant plus d'expérience. Ce que nous retenons cependant, c'est que le travail d'ingénieur en électronique que nous avons pu toucher du doigt à différents niveaux comprend de nombreuses réalités différentes et demande une certaine coordination au sein de l'équipe pour mener le projet à son terme, chacun selon ses capacités et ses préférences.

En conclusion, nous pouvons affirmer que les enjeux que nous avons rencontrés derrière la réalisation du drone ont été triples : tout d'abord humains à cause de la coordination pour le travail dans l'équipe et l'apprentissage au travail de groupe que cela a nécessité. Puis théoriques avec la compréhension du fonctionnement de ce que nous souhaitons réaliser, et enfin pratiques avec la réalisation et la confrontation avec le modèle qui paraît souvent très simple. Cependant, notre drone est très basique ; il reste encore de nombreuses étapes dans la vie d'un projet d'ingénierie : la conception d'un nouveau produit innovant, la réalisation du cahier des charges conséquent, les études préliminaires de marché, etc... qui rajoutent certainement autant d'enjeux, qu'ils soient économiques, politiques ou autres, et que nous n'avons pas encore rencontrés.

Immenses Remerciements

Thierry, Romain, Kevin & Nicolas tiennent encore à remercier vivement :

Monsieur ALI YAHIA,

Professeur de Physique à l'Isep.

Monsieur DESLANDES,

Professeur de Mathématiques à l'Isep.

Monsieur AFRIAT et ses assistants,

Responsable des laboratoires de l'Isep.

Pour leur soutien et leur aide précieuse dans ce TIPE.

Ainsi que tous ceux qui ont contribué de près ou de loin à la réalisation de ce TIPE.