PROJET INTERDISCIPLINAIRE

Quadricoptère

Logiciel de vol

DIETRICH Lucas

Lycée Jean-Baptiste Schwilgué - Sélestat

Terminale S – Option SI

Session 2013-2014

D3 engineering

Dussourd | Dietrich | Dantz

www.kestrel.fr

REMERCIEMENTS

Nous souhaitons remercier les personnes qui nous ont aidés à concrétiser notre projet tout au long de l'année :

- nos professeurs de Sciences de l'Ingénieur, M. Didier COUCHEVELLOU (professeur de Génie Electrique), M. Romuald FOULON, M. François RAGUSAF et M. Remi FREIXINOS (professeurs de Génie Mécanique), pour leurs disponibilités et leurs conseils,

- M. Bertrand SPITZ, enseignant à l'Ecole Nationale de l'Aviation Civile (ENAC), pour ses conseils et son aide concernant le filtre complémentaire,

- Mme Céline KAVALIAUSKAS, notre professeur de mathématiques, pour ses précisions concernant les matrices de rotation,

- le lycée Jean-Baptiste Schwilgué pour l'achat du matériel nécessaire au projet,

- les relations extérieures au lycée qui nous ont apportés soutien et conseils.

TABLE DES MATIERES

INTRODUCTION

THEME : ROBOTIQUE

SOUS-THEME : Quadricoptère

PROBLEMATIQUE :

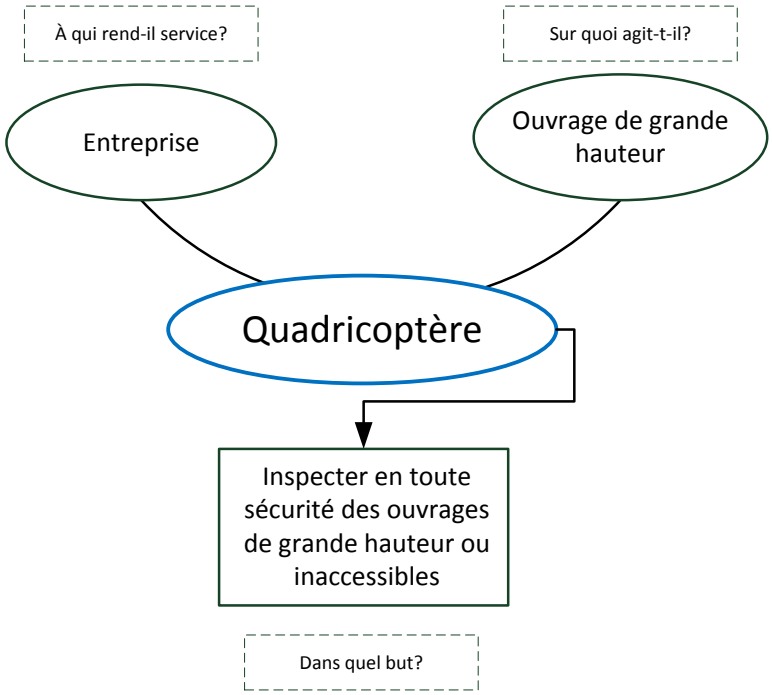
Comment inspecter en toute sécurité un ouvrage de grande hauteur ?

1. Présentation du projet

Dès la fin du mois d'août, notre équipe a choisi de concevoir un quadricoptère dans le cadre du projet de terminale S. Nous souhaitions **concevoir nous-mêmes l'intégralité** d'un objet technique en mettant en œuvre des **compétences diverses** : électronique, mécanique, informatique, physique et mathématiques. C'est un projet ambitieux, choisi en raison de sa complexité.

2. Identification du besoin

Certains édifices doivent être inspectés pour détecter d'éventuels dangers ou évènements anormaux. Cependant, certains d'entre eux sont très difficiles d'accès, voire inaccessibles (ponts, immeubles, barrages), et nécessitent de mettre en place des équipements lourds et coûteux (grue, échafaudages, etc…). Parfois, l'inspection nécessite la cessation de l'activité dans la zone pour des questions de sécurité ou d'accessibilité.



Un quadricoptère permet d'inspecter la plupart des grands ouvrages en toute sécurité pour les usagers, sans empêcher l'activité et sans mettre en œuvre des équipements coûteux. De plus, il évite l'intervention humaine directe et dangereuse sur l'ouvrage lorsqu'il ne s'agit que d'une inspection.

3. Cahier des charges

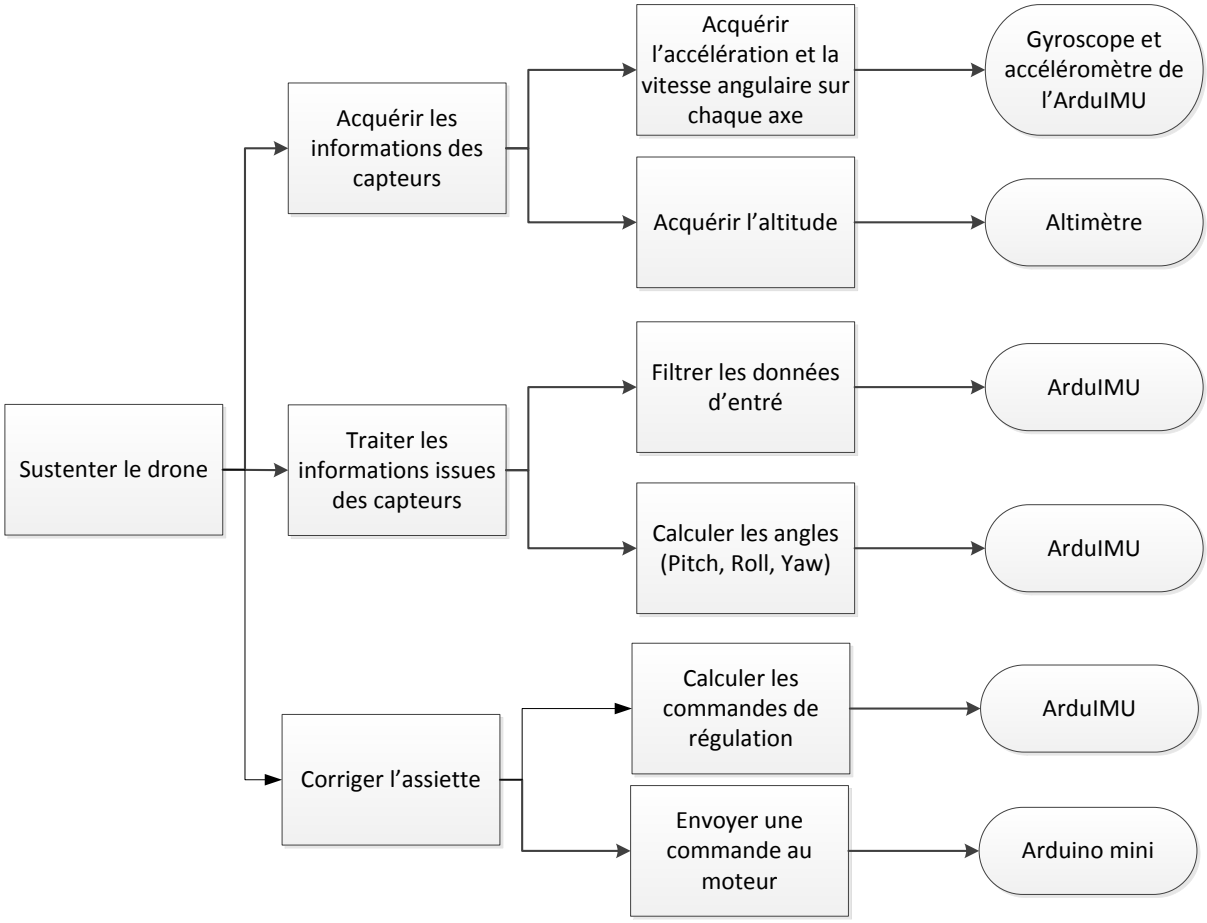
COÛT MAXIMAL	300€			
SOLUTIONS	Utilisation de cartes standards	Altitude maximale : 50m	Autonomie minimale de vol : 5min	Portée minimale en champ libre : 500m

4. Tâche à effectuer

Ma tâche consiste à mettre en place un système de contrôle qui permette de sustenter le quadricoptère dans les airs. Il faut tout d'abord concevoir **le système électronique,** incluant les capteurs qui permettent d'acquérir les informations, les microcontrôleurs pour le traitement des données et les interfaces de communication. Il est nécessaire de mettre en place **un modèle mathématique**, concevoir **l'algorithme de contrôle** (algorithme de régulation) et enfin permettre l'envoi des commandes calculées aux moteurs afin de stabiliser l'appareil. De plus, le système doit pouvoir communiquer avec le sol, être sécuritaire et permettre un développement continuel, pratique et rapide.

5. Diagramme FAST

Voici le diagramme FAST de ma partie dans le projet :



Vous trouverez à cette adresse (https://github.com/Kestrel67/quadcopter) le logiciel de contrôle en C++ hébergé par GitHub. Chaque annotation du type « réf : bibliothèque / fichier » fait référence à un fichier d'une librairie de ce logiciel. Elles permettent d'illustrer par du code chaque partie du document.

I. MATERIEL : CHOIX ET MISE EN PLACE DES COMPOSANTS

1. Microcontrôleurs Arduino

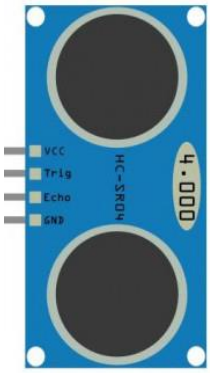
Le microcontrôleur ArduIMU conçu et vendu par Sparkfun Electronics est un dérivé des cartes Arduino, programmable en C++ par le logiciel Arduino. Il possède un gyroscope 3 axes, un accéléromètre 3 axes et également un magnétomètre 3 axes. Le processeur est un ATmega328P 8bits cadencé à 16MHz avec 32Ko de mémoire programmable. Il est également possible d'y ajouter un GPS. Son coût est approximativement de 60€.

Il a été choisi pour sa facilité de mise en œuvre, c'est-à-dire que tous les capteurs sont sur la même carte. Le processeur est suffisamment puissant pour effectuer l'acquisition, le traitement et le calcul des commandes moteurs ainsi que la communication avec le sol.

Un Arduino Mini (ATMega328P 8bits 16MHz) a été choisi comme contrôleur annexe pour générer les 4 signaux MLI destinés à chaque ESC. Il est peu cher et coûte environ 12€.

2. Altimètre *(réf. : ALN3D : Sensors.cpp : Ligne 61 →fin)*

L'altimètre à ultrason choisi est le HC-SR04 à 5€. Il a une portée maximale de 4m et une précision de 3mm sur une surface plane réfléchissante, l'angle maximal avec la surface doit être de 15°. Un signal à 1L (1 logique) d'une durée minimale de 10µs sur l'entrée « Trig » du composant permet de déclencher une mesure. Il envoie un signal ultrason à une fréquence de 40kHz et mémorise la durée nécessaire à l'onde pour revenir. Ainsi, il envoie sur la sortie « Echo » un signal à 1L dont la durée est celle du voyage de l'onde ultrason.



La durée de ce signal est mesurée par le timer 16bit (timer1) du processeur en mode compteur (« counter »). Ce timer est cadencé à la fréquence de l'horloge principale divisé 8 fois c'est-à-dire f = 16MHz / 8 = 2MHz. Ce timer1 incrémente un compteur de une unité toutes les 0.5µs tant que le signal « Echo » est à l'état haut. Précision de distance théorique : 85µm (<< 3mm donc suffisant). La distance avec l'obstacle peut être calculée ainsi :

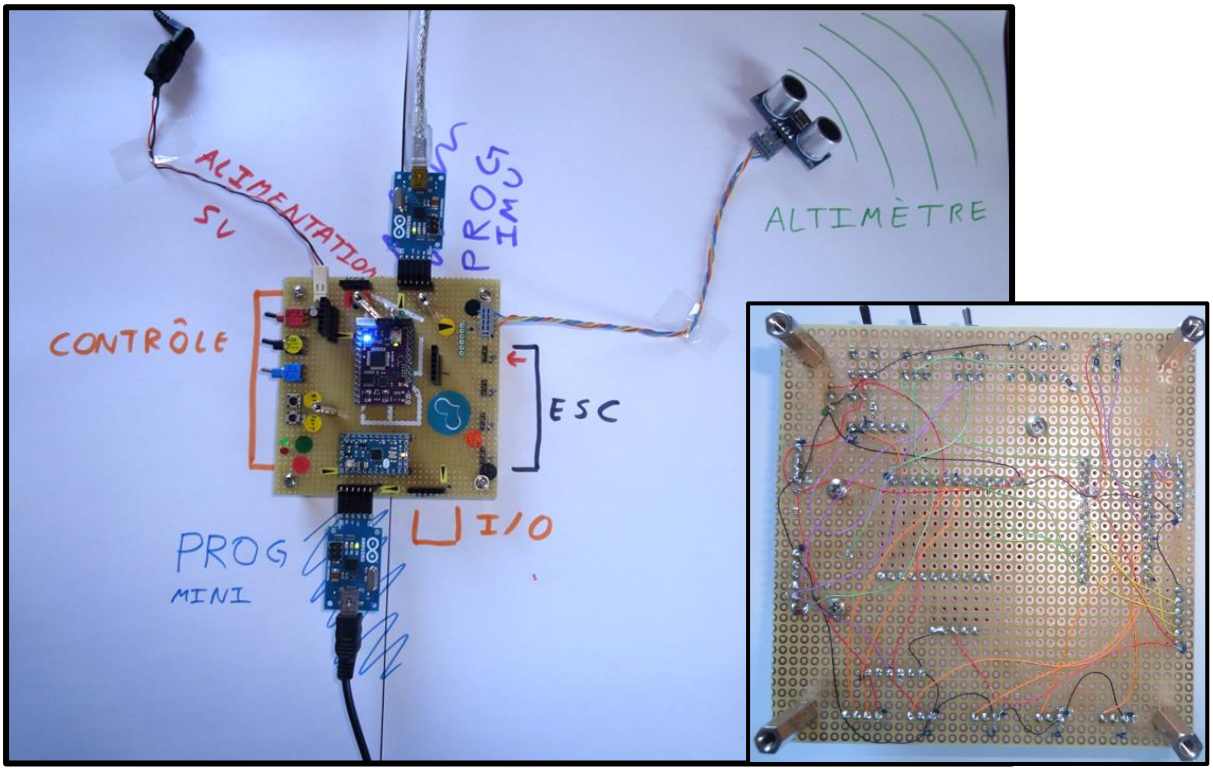
$$D = \frac{C * V}{2 * f}$$

Ou C est la valeur du compteur à la fin du signal, f la fréquence du timer1 (en Hz) et V la vitesse du son dans l'air (340m.s^-2). Il est nécessaire de diviser le temps par deux, car l'onde a fait un aller-retour.

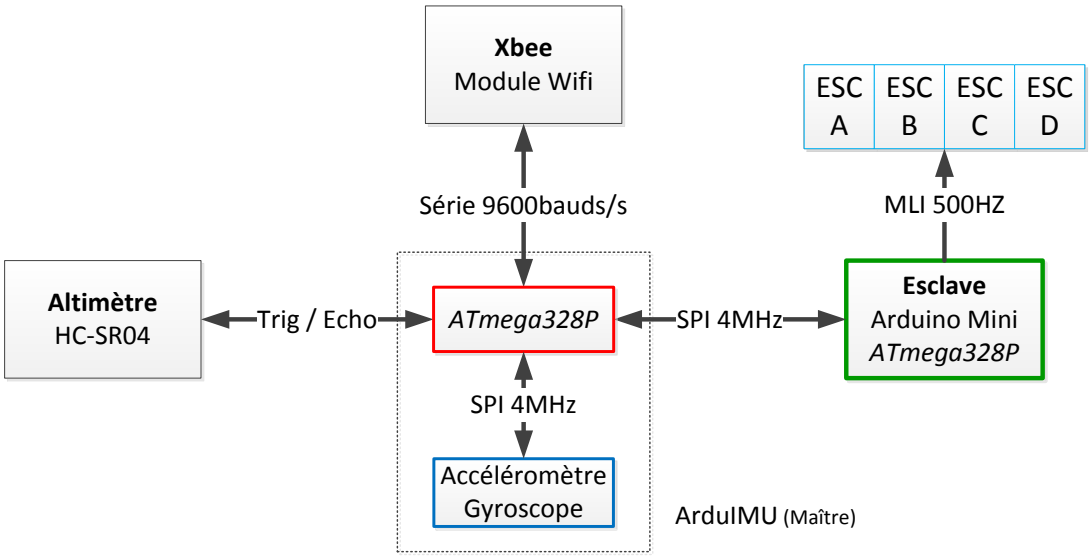
Ce composant a été choisi pour sa bonne portée et sa précision, l'altitude du drone sera contrôlée « à vue » pour une altitude supérieure à 3-4m. L'acquisition de l'altitude sera exécutée à une fréquence de 2Hz.

3. Circuit électrique

Un circuit électrique a été conçu pour y placer les deux microcontrôleurs, ainsi que les connexions, des entrées/sorties complémentaires, des diodes électroluminescentes et des sélecteurs. Un connecteur est prévu pour l'altimètre et quatre autres sont disponibles pour les ESC. L'ESC D a le rôle de fournir le courant à la carte pendant le vol, cependant un emplacement a été prévu pour une alimentation externe. Cette carte permet la programmation de chaque CPU. Des fixations ont été prévues pour y placer le shield et le module Xbee utilisés pour la communication. Ci-dessous deux photographies de la carte : (Vous trouverez le schéma électrique de la carte dans l'annexe)



Voici un schéma représentant l'organisation des composants sur la carte, avec le type de communication utilisé entre eux:



II. LOGICIEL

1. Le gestionnaire d'exécution

a) Tâches à effectuer

Le logiciel à concevoir doit exécuter des tâches à des fréquences différentes et variables. Les principales tâches figurent dans le tableau ci-dessous. Les fréquences ont été obtenues par expérimentation.

Fonction	Fréquence
Acquisition des données du gyroscope et de l'accéléromètre	100Hz
Acquisition des données du magnétomètre	10Hz
Acquisition de l'altitude	2Hz
Calculs cinématiques afin de déterminer l'assiette et l'orientation du drone dans l'espace	100Hz
Correction des angles de tangage et de roulis (PID)	25Hz
Correction de l'angle de lacet (PID)	10Hz
Correction de l'altitude (PID)	2Hz
Vérifier la disponibilité d'une commande sur la liaison de communication série	40Hz (Variable)
Envoyer les données de vol via la liaison série	5Hz (Variable)
Effectuer une analyse sur l'état du système	10Hz (Variable)

Les fonctions les plus lourdes en temps d'exécution et en fréquence sont surlignées en bleu.

b) Un gestionnaire traditionnel inadapté

Une méthode traditionnelle utilisée pour la programmation des petits microcontrôleurs serait de construire une boucle principale qui, pour chaque fonction, vérifierait si elle doit être exécutée ou non par rapport à sa dernière invocation. La boucle doit donc effectuer la double tâche de gérer la fréquence et l'exécution des fonctions. De plus, l'ajout d'une fonction nécessite d'insérer une condition dans la boucle principale et donc de l'alourdir, ce qui rend ce système peu flexible.

c) Un système évènementiel *(réf. : ALN3D : System.cpp, Timers.cpp)*

Le principe d'un tel système est d'exécuter la fonction associée à un évènement reçu. Chaque évènement est associé à une fonction (appelée callback). Les évènements sont par exemple générés par des timers logiciels à des fréquences variées, des interruptions ou par n'importe quelle autre fonction. Les évènements à traiter sont ajoutés à une file d'attente (FIFO : First in First Out). Le seul rôle de la boucle principale est de prélever l'évènement en tête de file et d'exécuter la callback associée.

Du fait que la boucle principale n'a qu'une action à effectuer, elle est très simple (annexe page 21 *system_loop()*). De plus, il est facile d'ajouter une fonction à exécuter à une fréquence déterminée. Il faut tout d'abord créer un évènement et lui associer la fonction (annexe page 20 catégorie EVENTS), et enfin ajouter un timer logiciel et définir la fréquence à laquelle exécuter la fonction (annexe page 20 catégorie TIMERS). Un système de la sorte est plus souple et permet un développement plus agréable et rapide.

d) Analyse des performances de la gestion des tâches *(réf.: ALN3D : Analyser.cpp, Manager.cpp)*

Une fonction d'analyse du logiciel a été mise en place. Elle permet de connaitre la fréquence de la boucle principale, le temps pris par le processeur pour exécuter les fonctions, le nombre de dépassements de file (overflow) et également le nombre de commandes reçues et envoyées. Cet analyseur est utile pour ajuster les différentes fréquences des fonctions afin d'optimiser leur utilisation.

2. Communication

a) Liaison SPI *(réf. : ALN3D : Communication.cpp : Lignes 1→93, ArduinoProLibrary : Communication.cpp)*

La communication entre l'ArduIMU et l'Arduino Mini est assurée par une liaison synchrone SPI à 4MHz où l'ArduIMU est le maître. Très rapide, elle permet d'échanger rapidement beaucoup de données sur de courtes distances. L'Arduino Mini (esclave) se comporte comme un registre, l'ArduIMU écrit ou lit à l'une des 32 adresses disponibles. S'il veut modifier la vitesse d'un moteur, il devra écrire la valeur du rapport cyclique à l'adresse réservée au moteur. L'écriture et la lecture consistent à envoyer 2 octets. Le protocole de communication utilisé est le suivant :

ADR	A4	A3	A2	A1	A0	R/$\overline{W}$	B7	$\overline{VAL}$	B6	B5	B4	B3	B2	B1	B0
Octet de l'adresse								Octet du paramètre							

ADR/$\overline{VAL}$: indique à l'esclave qu'il s'agit d'une adresse ou d'un paramètre

A0 →A4 : ces 5 bits forment l'adresse de l'octet à écrire ou à lire.

R/$\overline{W}$: ce bit correspond à l'action d'écriture ou de lecture

B0 → B7 : ces **8** bits forment l'octet de la valeur à écrire dans le registre.

Une interruption est générée chez l'esclave lorsqu'une requête a lieu, ainsi le traitement de cette requête (écriture ou lecture) est immédiatement effectué.

b) Liaison série *(réf. : ALN3D : Communication.cpp : Lignes 93→fin)*

La liaison série entre le quadricoptère et la station au sol est assurée par deux modules radio de type Xbee. La station envoie des commandes au quadricoptère, et ce dernier lui retourne les données de vol comme les angles, les commandes des moteurs et d'autres informations sur l'état du système.

Le protocole d'envoi des commandes est le même que celui utilisé pour la SPI, sauf que l'adresse est plus longue de 1 bit et que le bit de lecture/écriture est absent. L'adresse correspond à la commande et le paramètre correspond à la valeur optionnelle.

c) Contrôle des moteurs *(réf. : ArduinoProLibrary : Motors_ESC.cpp)*

Le contrôle de la poussée des moteurs s'effectue via un ESC. Il suffit de générer un signal rectangulaire dont le rapport cyclique est « proportionnel » à la poussée du moteur. La fréquence de ce signal est d'environ 500Hz. L'Arduino Mini génère 4 signaux de ce type (un pour chaque moteur) à l'aide de 2 timer physiques : timer0 et timer2.

3. Sécurité

Un quadricoptère est un engin dangereux, de nombreux accidents ont déjà eu lieu. La chute d'un drone de quelques kilogrammes est l'un des risques, mais plus basiquement les hélices sont tranchantes. Ces moteurs brushless sont puissants et tournent à une vitesse de 200tours/sec, malgré des hélices en plastique léger et souple, il faut être très vigilant.

Quelques précautions ont été prises en période de développement :

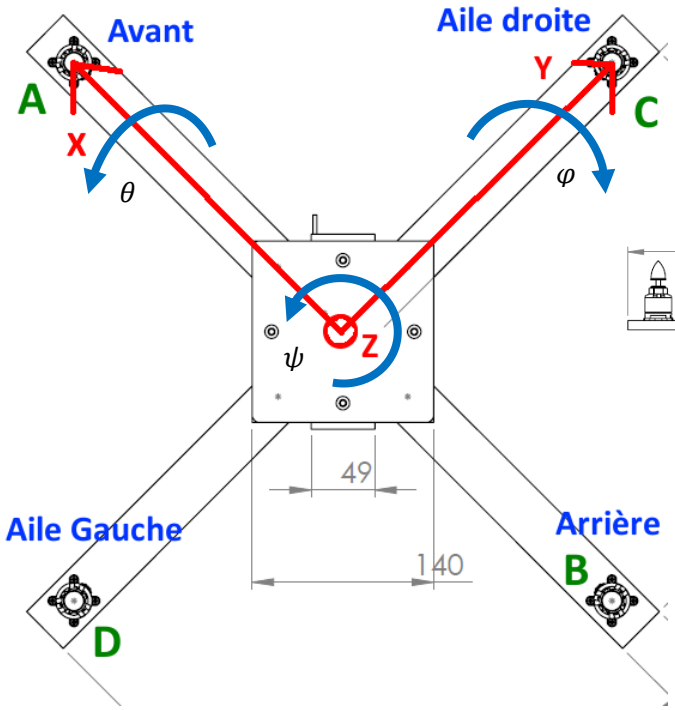
- lorsqu'un des angles de tangage ou de roulis est supérieur à 25°, on coupe les moteurs,
- lorsqu'une commande d'arrêt d'urgence est reçue via la liaison série, on coupe également les moteurs.

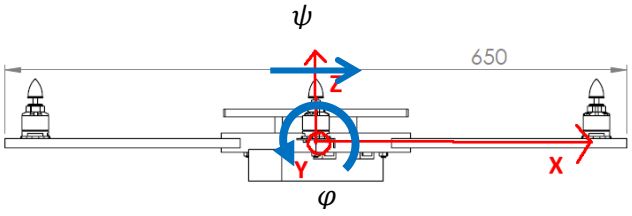
III. DYNAMIQUE

1. Conventions et notations

Le quadricoptère est constitué de 2 branches principales, qui se croisent perpendiculairement en leur milieu. Aux 4 opposés se trouvent 4 moteurs brushless. Le repère orthonormé du drone $R_Q: (o, \vec{x}, \vec{y}, \vec{z})$ est le repère de travail. Nous allons également définir le repère orthonormé terrestre par $R_T: (O, \vec{X}, \vec{Y}, \vec{Z})$, ou $(\vec{X}, \vec{Y})$ est la plan représentant la surface terrestre.

Pour un quadricoptère, engin symétrique, il est peut-être inapproprié de parler de tangage et de roulis, mais nous mettons en place cette convention pour simplifier l'étude. L'angle de tangage $(\boldsymbol{\varphi})$ est l'angle $(\vec{x}; \widehat{\vec{U}})$ où $\vec{U}$ est la projection du vecteur $\vec{x}$ dans le plan $(\vec{X}, \vec{Y})$. L'angle de roulis $(\boldsymbol{\theta})$ est l'angle $(\vec{y}; \widehat{\vec{V}})$ où $\vec{V}$ est la projection du vecteur $\vec{y}$ dans le plan $(\vec{X}, \vec{Y})$. L'angle de lacet $(\boldsymbol{\psi})$ est l'angle $(\vec{x}; \widehat{\overrightarrow{W}})$ où $\overrightarrow{W}$ est la projection du vecteur $\vec{x}$ dans le plan $(\vec{X}, \vec{Z})$.





Repère R_Q lié au drone, avec les angles de tangage, de roulis et de lacet.

2. Calcul des angles *(réf. : ALN3D : AP_IMU.cpp)*

Le vecteur d'accélération de pesanteur $\vec{g}\begin{pmatrix}0\\0\\-1\end{pmatrix}$ dans R_T est orthogonal au plan $(\vec{X}, \vec{Y})$ et est un bon référentiel pour calculer les angles de tangage et de roulis. Nous allons donc créer le vecteur $\vec{G}$, qui représentera $\vec{g}$ dans le repère R_Q ; il est colinéaire à celui-ci.

$$\|\vec{G}\| = 1, \tan\varphi = \frac{x}{z} \text{ et } \tan\theta = \frac{y}{z}$$

Par conséquent, nous pouvons reconstruire $\vec{G}$ avec les angles de tangage et de roulis : Soit $\boldsymbol{I} = \sqrt{1 + \tan^2\varphi + \tan^2\theta}$ (eq.16) :

$$\vec{G}\begin{vmatrix} x = -\tan\varphi\,\boldsymbol{I}^{-1} \\ y = -\tan\theta\,\boldsymbol{I}^{-1} \\ z = -\boldsymbol{I}^{-1} \end{vmatrix}$$

Le vecteur $\vec{G}$ est assimilé à une matrice colonne d'ordre 3: $G = \begin{bmatrix} G_x \\ G_y \\ G_z \end{bmatrix}$

Le but global est d'estimer dynamiquement et le plus précisément possible le vecteur $\vec{G}$ afin de calculer les angles de roulis et de tangage.

a) Accéléromètre

L'accéléromètre mesure l'accélération subite par le drone (m.s^-2), et donc l'accélération de pesanteur ($\vec{g}$) lorsque le quadricoptère est immobile. Il est théoriquement inadapté pour estimer $\vec{G}$ en dynamique. Outre le fait que n'importe quelle forte accélération fausse totalement la mesure des angles, il donne cependant des coordonnées moyennes correctes ce vecteur pour des accélérations faibles. Cependant, il est extrêmement sensible aux vibrations. Ces valeurs sont donc filtrées sur chaque axe à l'aide de la fonction de lissage exponentiel (Filtre-passe-bas) suivante ; cette fonction supprime les hautes fréquences (les vibrations) et lisse globalement le signal (eq.11) :

$$S_t = \eta * S_{t-1} + (1 - \eta) * X_t$$

S_t est la valeur de l'accélération lissée, η est le coefficient de lissage, ici égal à 0.90, et X_t est la valeur de l'accélération. Vous trouverez en **annexe (1)** un graphique montrant l'efficacité d'un tel filtre.

b) Gyroscope *(réf. : ALN3D : AP_RotationMatrix.cpp)*

Le gyroscope permet de mesurer les vitesses de rotation ω_θ, ω_φ et ω_ψ (rad.s^-1) respectivement autour de ses 3 axes $\vec{x}, \vec{y}$ et $\vec{z}$. En intégrant ces vitesses de rotation par le temps, on obtient les angles θ, φ et ψ (rad) effectués respectivement autour des axes $\vec{x}, \vec{y}$ et $\vec{z}$ pendant l'intervalle de temps dt :

$$\theta(t) = \int_0^t \omega_\theta(\tau)d\tau \qquad \varphi(t) = \int_0^t \omega_\varphi(\tau)d\tau \qquad \psi(t) = \int_0^t \omega_\psi(\tau)d\tau$$

Plus l'intervalle de temps est court ($dt \to 0$), plus la rotation globale calculée sera précise. C'est pourquoi nos fonctions de mesure, d'intégration et de calcul des angles seront à 100Hz.

Les matrices de rotation permettent de modéliser les 3 rotations successives, la matrice représentant la rotation globale du drone par rapport au sol est $(eq.2a)$:

$$R_{\varphi\theta\psi} = R_\varphi * R_\theta * R_\psi$$

$$R_\varphi = \begin{bmatrix} \cos\varphi & 0 & -\sin\varphi \\ 0 & 1 & 0 \\ \sin\varphi & 0 & \cos\varphi \end{bmatrix} \qquad R_\theta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \qquad R_\psi = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ainsi, la matrice représentant la rotation du sol par rapport au drone est $(eq.2b)$:

$$R_{\varphi\theta\psi}{}^{-1} = \boldsymbol{R_{\varphi\theta\psi}{}^{T}} = (R_\varphi * R_\theta * R_\psi)^T = R_\psi{}^T * R_\theta{}^T * R_\varphi{}^T$$

À chaque itération, nous appliquons la nouvelle matrice de rotation au vecteur $\vec{G}$ (initialisé grâce à l'accéléromètre) pour connaître ces nouvelles coordonnées :

$$G' = R_{\varphi\theta\psi}{}^T * G$$

Le gyroscope mesure très précisément les rotations rapides. Mais la principale contrainte est qu'il ne mesure pas exactement 0 rad.s^-1 lorsque l'IMU redevient immobile après une rotation, ceci est l'effet de dérive (« drift ») qu'il faut corriger. De plus, cette erreur est amplifiée par l'intégration. Par conséquent, le gyroscope ne pourra être utilisé que pour déterminer les angles à court terme (quelques secondes). Vous trouverez en **annexe (2)** un graphique illustrant la problématique.

c) Filtre complémentaire

Nous utilisons donc un filtre spécial pour combiner le gyroscope et l'accéléromètre. Le principe d'un **filtre complémentaire** est simplement d'utiliser à court terme les angles issus du gyroscope et de les corriger à long terme à l'aide de ceux issus de l'accéléromètre (eq.14) (voir **annexe (3)**):

$$\alpha(t) = \eta * \alpha_{Gyro}(t) + (1-\eta) * \alpha_{Accel}(t)$$

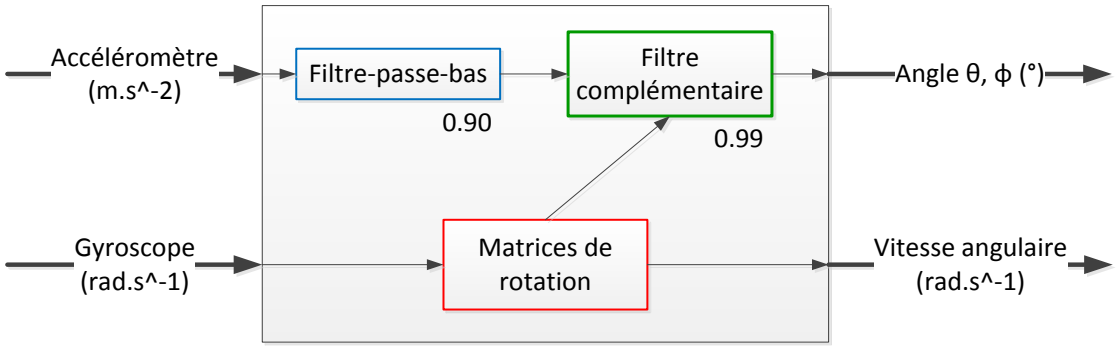
Dans notre cas $\eta = 0.99$, ceci implique que l'angle calculé est en majeur partie issu du gyroscope, cependant l'accéléromètre a une influence, certes faible sur l'angle calculé.

Ainsi, les deux angles calculés seront utilisés pour mettre à jour le vecteur $\vec{G}$ et permettre au gyroscope de calculer les nouvelles rotations en fonction de celui-ci.

Une amélioration apportée est d'ignorer les données probablement fausses de l'accéléromètre et de se fier uniquement au gyroscope lorsqu'une rotation très rapide est mesurée ou lorsque l'accélération dépasse une certaine valeur.

Synthèse

Voici le schéma récapitulant la procédure pour estimer les angles.



IV. ASSERVISSEMENT

1. Influence des moteurs

D'après la figure du drone figurant en **(III.1)** nous pouvons voir que les moteurs A et B influent sur le tangage de l'appareil et symétriquement les moteurs C et D sur le roulis. En augmentant la vitesse de A et en diminuant celle de B, le nez de l'appareil va s'incliner vers l'arrière.

Les moteurs A et B tournent dans le sens horaire et les moteurs C et D tournent dans le sens trigonométrique. Cette solution technique a été retenue pour minimiser l'effet de lacet dû aux moteurs. Une hélice tournant dans le sens horaire produit une force colinéaire à l'axe de rotation qui soulèvera l'appareil et une force tangentielle qui sera exercée sur l'air. D'après la troisième loi de Newton, l'air va exercer une force sur l'hélice qui va entrainer l'ensemble (moteur et hélice → drone) dans la rotation inverse : sens trigonométrique.

2. Régulateur proportionnel intégral dérivé (PID) *(réf. : ALN3D : AP_PID.cpp)*

L'asservissement consiste à stabiliser et sustenter le quadricoptère en modifiant la poussée de chaque moteur afin de se rapprocher de la position de consigne. Pour cela, nous allons utiliser un régulateur automatique de type PID (proportionnel, intégrale, dérivé). Ce régulateur tente de minimiser l'erreur $\varepsilon(t)$ le plus rapidement possible :

L'erreur $\varepsilon(t) = S(t) - \alpha(t)$ avec $S(t)$ valeur de consigne et $\alpha(t)$ la valeur d'entrée.

$\varsigma(t)$ correspond à la valeur de commande calculée par le régulateur pour minimiser l'erreur.

L'équation de régulation est la suivante (eq.17) :

$$\varsigma(t) = K_p * \varepsilon(t) + K_i * \int_0^t \varepsilon(\tau)\mathrm{d}\tau + K_d * \frac{\mathrm{d}\varepsilon(\tau)}{\mathrm{d}\tau}$$

Une version améliorée permet d'éviter le dépassement de la composante dérivée lorsque l'on modifie la consigne. Pour cela on suppose la valeur de consigne constante ($\forall t, S(t) = c$) et on obtient une dérivé de mesure (eq.18):

$$\varsigma(t) = K_p * \varepsilon(t) + K_i * \int_0^t \varepsilon(\tau)\mathrm{d}\tau - K_d * \frac{\mathrm{d}\alpha(\tau)}{\mathrm{d}\tau}$$

Les constantes K_p, K_i et K_d sont respectivement les coefficients proportionnel, intégral et dérivée. Ils déterminent quelle(s) doit (doivent) être la ou les composantes dominantes pour la régulation. Ces coefficients doivent être déterminés expérimentalement et dépendent fortement des paramètres physiques du drone (poussée des moteurs, poids, envergure, etc…) et du type de vol (stabilité maximale, acrobatie, réactivité, etc…).

3. Grandeurs physiques régulées *(réf. : ALN3D : AP_Motors.cpp, lignes 24→31)*

Un asservissement est réalisé sur le tangage, le roulis, le lacet et l'altitude.

ς_T : PID d'altitude ς_φ : PID de tangage ς_θ : PID de roulis ς_ψ : PID de lacet

Les valeurs de commande $\varsigma_A, \varsigma_B, \varsigma_C$ et ς_D à appliquer respectivement aux moteurs A, B, C et D sont :

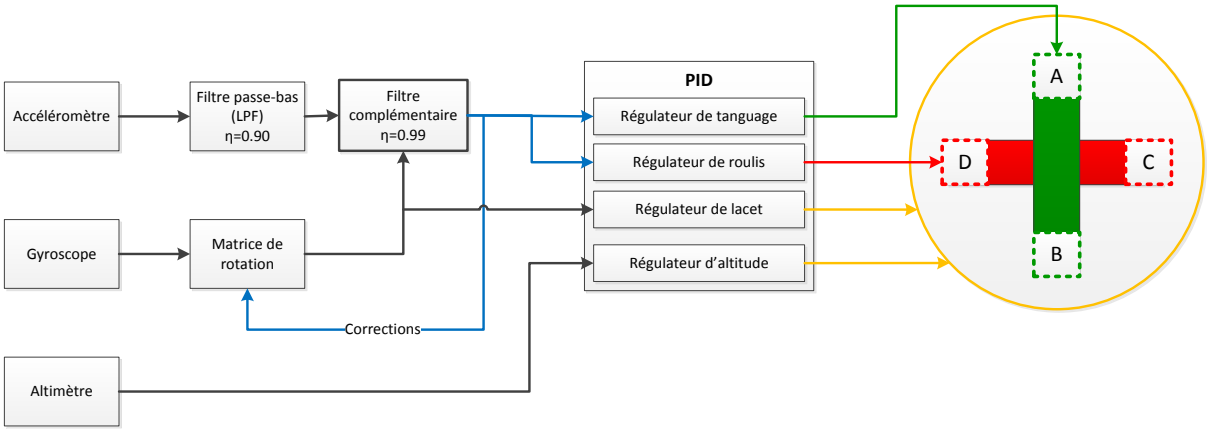
$$\varsigma_A = \varsigma_T + \varsigma_\varphi + \varsigma_\psi$$

$$\varsigma_B = \varsigma_T - \varsigma_\varphi + \varsigma_\psi$$

$$\varsigma_C = \varsigma_T + \varsigma_\theta - \varsigma_\psi$$

$$\varsigma_D = \varsigma_T - \varsigma_\theta - \varsigma_\psi$$

Voici un schéma résumant les 2 dernières parties de cette étude:



CONCLUSION

A l'issue des PPE, j'estime que le système mis en place pour sustenter l'appareil est abouti. Les tests de stabilisation effectués sur 1 axe montrent l'efficacité du modèle mathématique et des solutions technologiques employés. Le quadricoptère conserve de façon stable une position, et atteint rapidement et de manière précise une position de consigne.

La centrale inertielle choisie pour un coût total de 80€ est suffisamment performante pour effectuer toutes les mesures, calculs et corrections nécessaires. Le microcontrôleur ATmega328P cadencé à 16MHz permettrait même d'effectuer les calculs à une fréquence 2 fois plus élevée. Le filtre complémentaire est l'algorithme avec le meilleur rapport performances/complexité. Cependant, une amélioration possible serait de le remplacer par un Kalman. Ce filtre est idéalement utilisé pour combiner des grandeurs physiques, telles que l'accélération et la vitesse angulaire, mais il est mathématiquement complexe (hors du programme de terminale), lourd à coder et à exécuter.

Ainsi, nous pourrions envisager un décollage ainsi qu'un vol autonome d'une durée supérieure à 5 minutes, cependant beaucoup de tests sont encore nécessaires pour y parvenir. Les connaissances requises en mathématiques et en physique sont un frein majeur à notre projet, ce qui nous oblige à passer un temps considérable à comprendre les notions qui nous font défaut.

L'objectif que nous nous étions fixé a cependant été atteint : être capable de concevoir un drone de façon totalement autonome sans l'utilisation d'aucun logiciel existant. Ce projet a été très bénéfique pour nous, car il nous a permis de comprendre et d'appliquer des concepts nouveaux. De plus, la réalisation du prototype et les tests effectués marquent la concrétisation d'un travail collectif et l'achèvement de notre projet.

GLOSSAIRE

ESC : Electronic Speed Control est un circuit électronique qui permet de modifier la vitesse d'un moteur électrique, ici de type brushless.

MLI : Modulation de Largeur d'Impulsion (en anglais PWM : Pulse Width Modulation) est un procédé qui consiste à modifier le rapport cyclique d'un signal rectangulaire afin de modifier la vitesse d'un moteur par exemple.

Shield : élément de type socle conçu ici pour se poser sur un Arduino et proposer de nouvelles fonctionnalités. Shield Xbee pour relier chaque pin de l'arduino à celles du module Xbee.

Xbee : Catégorie de module radio.

SPI : Serial Peripheral Interface, Protocole de communication synchrone bidirectionnel (full-duplex) de type maître/esclaves.

ATmega328P : Microcontrôleur développé par la société ATmel. CPU (Cœur): 8bits de type AVR (fréquence maximale : 20Mhz) flash (mémoire programmable): 32ko, RAM : 2ko. Il est utilisé dans de nombreux modèles de cartes Arduino.

IMU : Inertial Measurement Unit, une centrale à inertie est un instrument permettant d'intégrer le mouvement d'un mobile (accélération, vitesse angulaire) afin d'estimer sa position et son orientation (cap, tangage, roulis, etc…).

PID : Un régulateur de type Proportionnel Intégral Dérivé permet d'effectuer une régulation d'une grandeur physique en boucle fermée.

Timer : une « horloge » est une fonction d'un processeur qui permet de mesurer ou générer un signal d'une certaine durée avec une précision qui dépend de sa fréquence.

ANNEXES

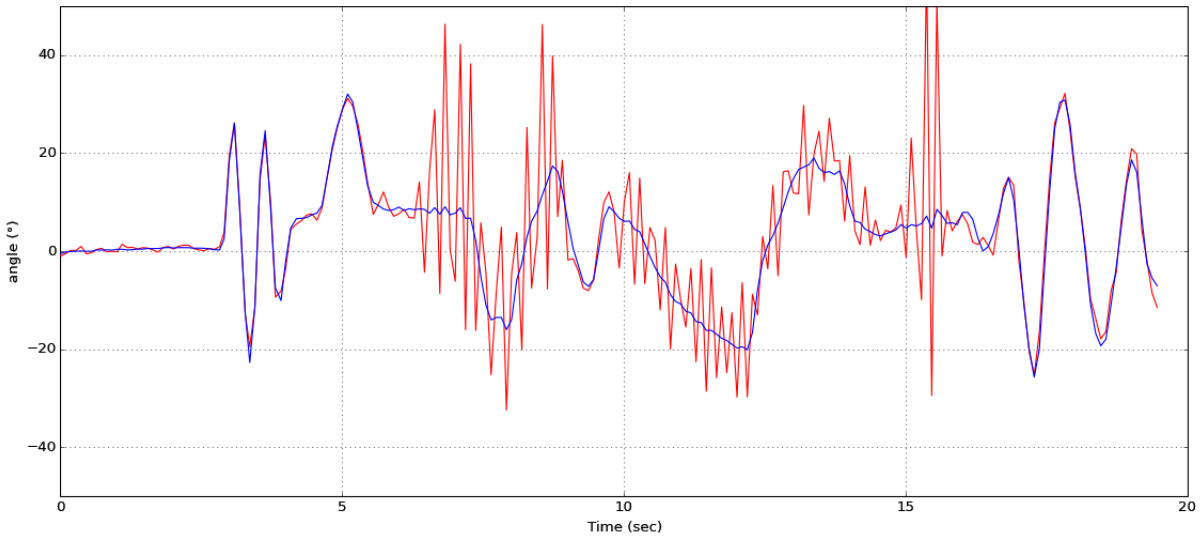
1. Graphiques

Ces graphiques ont été réalisés à l'aide de python et des packages **matplotlib** et **numpy**.

a) Filtre passe-bas et accéléromètres

Ce graphique montre l'angle de roll calculé d'une part avec l'accélération non filtré (en **rouge**) et d'autre part avec l'accélération filtré (en **bleu**). Nous constatons :

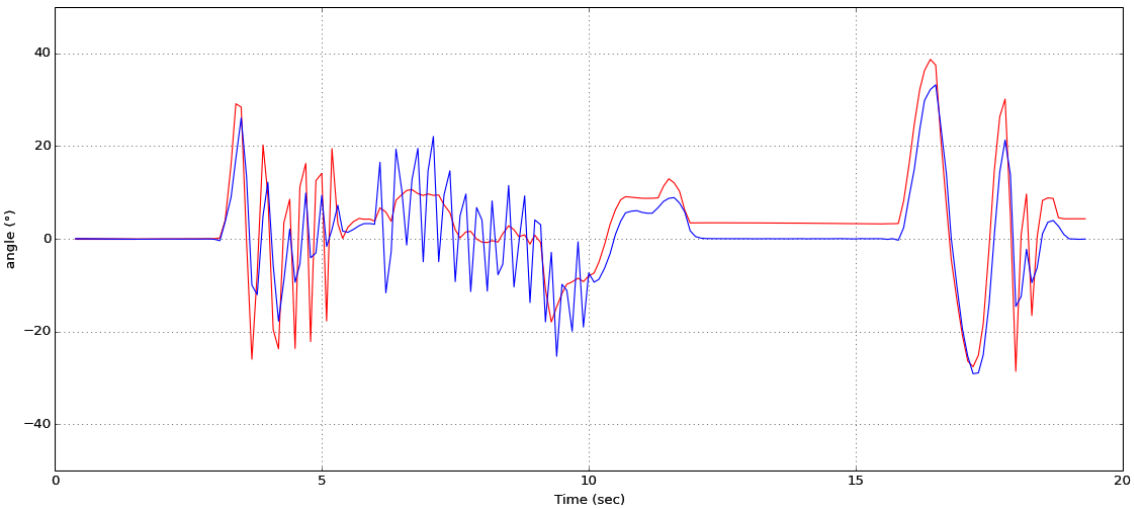
- Le respect des rotations : secondes 0 à 6 et 16 à 20.
- L'atténuation des hautes fréquences (vibrations) : secondes 6 à 16
- Le respect de la valeur moyenne du signal : secondes 10 à 15



b) Accéléromètres et gyroscopes

Ici, la courbe **bleue** représente l'angle calculé à partir de l'accéléromètre (non filtrés) et la courbe **rouge** représente l'angle calculé à partir du gyroscope. Nous constatons :

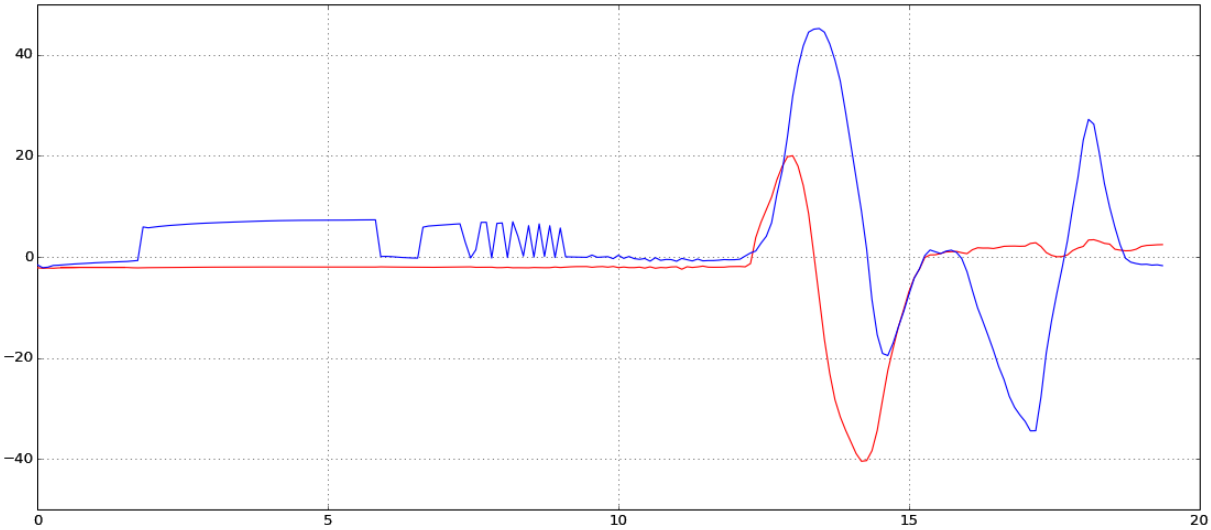
- La meilleure précision du gyroscope sur les rotations : secondes 3 à 6
- L'insensibilité du gyroscope aux vibrations et aux déplacements latéraux : secondes 6 à 10
- L'effet de « drift » du gyroscope par rapport à l'angle estimé à l'aide de l'accéléromètre : secondes 10 à 20



c) Filtre complémentaire

Le filtre complémentaire combine les angles issus du gyroscope et de l'accéléromètre afin d'obtenir des angles théoriques les plus proches de la réalité. La courbe en **rouge** représente l'angle de tangage et la courbe **bleue** représente l'angle de roulis. Nous constatons :

- La sensibilité aux rotations rapides : secondes 6 à 9
- L'insensibilité aux vibrations et déplacements latéraux : secondes 9 à 12
- La précision des angles calculés : secondes 12 à 20



2. Vidéos, animations, photographies et documents

Vous trouverez des ressources supplémentaires sur le site du projet : http://kestrel.fr/quadcopter

3. Logiciel

Voici le programme principal (*Main.h*) du logiciel de contrôle. Tous les autres programmes sont disponibles à l'adresse suivante : https://github.com/Kestrel67/quadcopter

```
void system_setup(bool conf, bool synchronization, bool calibrate, bool analyser, bool manager)
{
	/*********** OUTPUTS ***********/
	pinMode(EMBEDED_LED_RED, OUTPUT);
	pinMode(EMBEDED_LED_BLUE, OUTPUT);

	digitalWrite(EMBEDED_LED_RED, HIGH);

	/*********** COM ***********/
	SPIMaster Master = SPIMaster();		// SPI : Master class
	Serial.begin(SERIAL_BAUDRATE);		// Serial
	AP_init(&Master);			// SPI : Slave

	/*********** COMPONENTS ***********/
	MPU6000_Init(&Master);	// accel, gyro
	HMC5883L_Init();	// compass

	/*********** EVENTS ***********/
	//		EVENT			CALLBACK
	register_event(EVENT_DYNAMIC,		_proc_dynamic_calculation);
	register_event(EVENT_HMC5883L,		HMC5883L_Read);
	register_event(EVENT_PID_ROLL_PITCH,	_proc_dynamic_angles_PID);
	register_event(EVENT_PID_ALTITUDE,	callback_pid_altitude);
	register_event(EVENT_SERIAL_DATA_IN,	callback_ser_data_in);
	register_event(EVENT_SERIAL_DATA_OUT,	_proc_com_out);
	register_event(EVENT_LED_POSITION,	update_position_leds);

	/*********** TIMERS ***********/
	//		EVENT			FREQUENCY
	add_timer(EVENT_DYNAMIC,		FREQUENCY_DYNAMIC);
	add_timer(EVENT_PID_ROLL_PITCH,		FREQUENCY_PID_ROLL_PITCH);
	add_timer(EVENT_SERIAL_DATA_OUT,	FREQUENCY_SERIAL_DATA_OUT);
	add_timer(EVENT_LED_POSITION,		FREQUENCY_LED_POSITION);
	add_timer(EVENT_HMC5883L,		FREQUENCY_GET_HMC5883L);

	/*********** ALTIMETER ***********/
	initialize_hc_sr04(PWM0,			FREQUENCY_SAMPLE_HC_SR04, EVENT_PID_ALTITUDE);

	/*********** COM IN ***********/
	set_serial_observer(FREQUENCY_SERIAL_OBSERVER, EVENT_SERIAL_DATA_IN);

	/*********** ANALYSER + MANAGER ***********/
	if (analyser)
	{
		set_CPU_Analyser(FREQUENCY_ANALYSER);

		if (manager)	set_CPU_Manager(FREQUENCY_MANAGER);
	}

	/*********** PID ***********/
	PID_Init();
	PID_Manual();
```

```
	/*********** IMU ***********/
	IMU_Init(calibrate);

	/*********** SPI ***********/
	if (synchronization)
		AP_synchronization();

	AP_write(APL_REG_BLK, 0);
	ApplyMotorsThrottle();

	/*********** AUTHOR ***********/
	if (conf)
		ser_display_IMU_conf();

	digitalWrite(EMBEDED_LED_RED, LOW);

	/*********** TIMERS ***********/
	// MSTimer2 : 1000Hz
	initialize_timers(FREQUENCY_SEQUENCER_TIMER);
}


void system_loop()
{
	for(;;)
	{
		Event_t e = event_dequeue();

		if (e != EVENT_NULL)
			execute_callback(e);
	}
}
```

BIBLIOGRAPHIE

Accéléromètre, Gyroscope:

http://www.instructables.com/id/Accelerometer-Gyro-Tutorial/

https://code.google.com/p/ardu-imu/

https://github.com/TKJElectronics

http://theboredengineers.com/2012/09/the-quadcopter-get-its-orientation-from-sensors/

http://theboredengineers.com/category/diy/quadri/

http://theccontinuum.com/2012/09/24/arduino-imu-pitch-roll-from-accelerometer/

Matrice de rotation :

http://en.wikipedia.org/wiki/Rotation_matrix

http://en.wikipedia.org/wiki/Euler_angles

Complementary Filter, Kalman filter:

http://www.pieter-jan.com/node/11

PID:

http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/

http://www.ferdinandpiette.com/blog/2012/04/asservissement-en-vitesse-dun-moteur-avec-arduino/

Arduino:

http://www.atmel.com/Images/doc8161.pdf

http://playground.arduino.cc/Code/Timer1

http://sphinx.mythic-beasts.com/~markt/ATmega-timers.html

https://www.inkling.com/read/arduino-cookbook-michael-margolis-2nd/chapter-18/recipe-18-7