

# Projet Quadricoptère

TUTEUR : M. LEBOULANGER

MATHIEU ESTEBAN

DAOUD ELHOR

OLIVIER MARTIN

LAURENT MARTIN

IUT DE VILLE D'AVRAY

## Table des matières

Introduction.....	2
Cahier des charges fonctionnel .....	3
Bête à cornes .....	3
Graphe des fonctions de services.....	4
Tableau de caractérisation des fonctions de services.....	5
Diagramme de FAST .....	7
Carte mentale.....	9
Diagramme de Gantt.....	11
Centrale inertielle.....	13
Carte d'interface avec la centrale inertielle .....	17
Capteur ultrason.....	25
Camera .....	26
Asservissements .....	33
Emetteur et récepteurs infrarouges.....	37
Besoins financiers et matériels.....	40
Conclusion .....	42
Bibliographie.....	43
Documentation.....	43
Annexe.....	44

## Introduction

Le projet quadricoptère est un projet qui consiste à créer et programmer un drone télécommandé. Il touche à plusieurs domaines qui sont l'électronique, l'automatique ou encore l'informatique industrielle. Ce projet a été commencé en 2011 par les étudiants en GEII de l'IUT de Ville d'Avray. Des étudiants de licence ce sont également joint au projet en 2014/2015, mais ils ont travaillé sur un microcontrôleur différent de celui utilisé lors des années précédentes. Le projet n'a cependant pas été continué durant l'année 2015/2016. Nous sommes donc deux groupes cette année travaillant sur ce projet, l'autre groupe allant créer un nouveau drone depuis le début, et notre groupe qui va partir du drone déjà existant afin de l'améliorer.

En effet, le drone n'était pas stable et possédait plusieurs défauts. Nos objectifs étaient donc d'améliorer la stabilité du drone en vol à l'aide d'une centrale inertielle qui a remplacé le gyroscope et l'accéléromètre présent précédemment (cette centrale nous a également servi à modifier l'inclinaison du drone en fonction de l'angle et non de la vitesse angulaire), d'utiliser un capteur barométrique pour avoir une information supplémentaire sur l'altitude, d'insérer une nacelle avec caméra sur le quadricoptère qui sera asservie en fonction de la position du quadricoptère, d'utiliser d'un GPS pour programmer les déplacements du quadricoptère, d'ajouter un dispositif de « tir » avec des LEDs et récepteurs infrarouges afin de pouvoir tirer sur le deuxième quadricoptère.

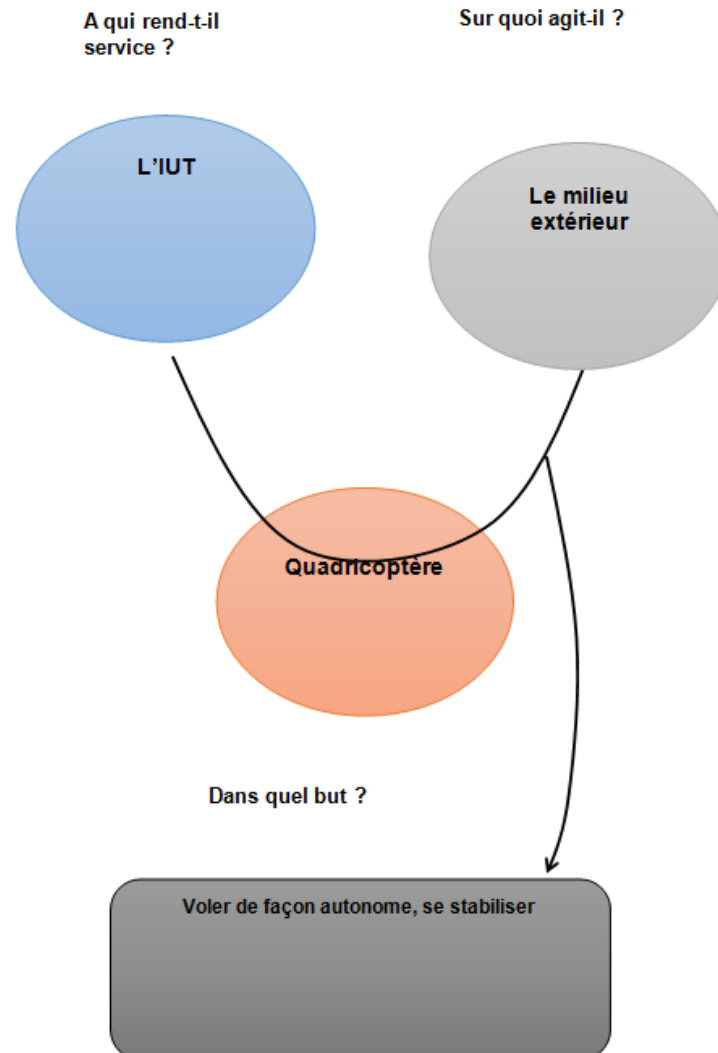
L'asservissement en angle a bien été effectué, mais le drone ne vole pas car le deuxième asservissement en vitesse angulaire ne fonctionne pas. Nous utilisons bien la centrale inertielle pour recevoir les différentes informations en angle et en vitesse angulaire. Le bras de la caméra a été monté et le programme qui va avec a été commencé mais il reste à fixer le bras sur le quadricoptère. Avec les émetteurs et récepteurs infrarouges nous avons réussi à obtenir une portée de 2,89 mètres ce qui est juste. Cependant nous n'avons pas utilisé de capteur barométrique ni de GPS.

## Cahier des charges fonctionnel

Dans cette partie, nous allons reprendre les points essentiels de l'analyse fonctionnelle.

### Bête à cornes

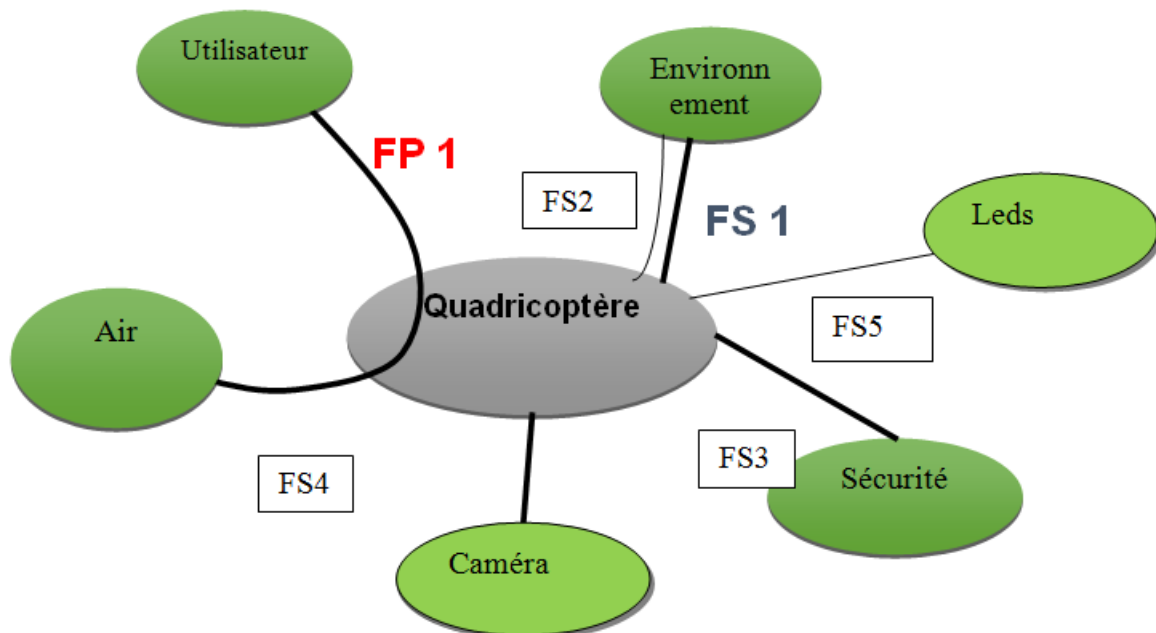
Voici notre "Bête à corne" :



Avec cette Bête à cornes que nous avons réalisé, nous avons d'abord déterminé que ce projet rendait en premier lieu service à l'IUT, qui en sera le principal utilisateur. Nous avons ensuite déterminé sur quoi agissait notre quadricoptère, ici le milieu extérieur. Enfin nous en sommes parvenu à déduire que son but était de voler de façon autonome et de se stabiliser, pour permettre à l'utilisateur de pouvoir le contrôler.

### Graphe des fonctions de services

Voici notre graphe des fonctions de services :



Nous avons établi les différentes fonctions en rapport avec notre quadricoptère. Nous avons ainsi trouvé une fonction principale et cinq fonctions de contraintes.

Notre fonction principale est de permettre à l'utilisateur de contrôler le quadricoptère aisément dans l'air, l'environnement, d'observer les paysages à distance. Notre première fonction de contrainte est de ne pas être trop encombrant. Notre deuxième fonction de contrainte est de respecter l'environnement qui l'entoure et de pouvoir résister à un certain vent. Notre troisième fonction est assez proche de la précédente mais se rapproche plus de la sécurité, car le quadricoptère ne doit pas nuire ni à l'utilisateur ni à son environnement. La quatrième fonction est de pouvoir explorer les alentours et repérer des cibles. Enfin la dernière fonction de contrainte est de pouvoir viser les récepteurs infrarouges d'un autre quadricoptère afin de réaliser un éventuel jeu de tir.

Voici pour résumer nos différentes fonctions :

- ❖ FP1 : Permettre à l'utilisateur d'observer les paysages à distance, transporter des objets dans les airs et pouvoir le contrôler facilement
- ❖ FS1 : Être peu encombrant
- ❖ FS2 : Respecter l'environnement et résister aux conditions climatiques
- ❖ FS3 : Ne pas nuire à l'utilisateur ni à son environnement
- ❖ FS4 : Explorer l'entourage et repérer les cibles
- ❖ FS5 : Viser les récepteurs infrarouges des autres quadricoptères

### Tableau de caractérisation des fonctions de services

Nous avons ensuite créé un tableau de caractérisation des fonctions de service que voici :

Fonction	Formulation	Critères	Niveau	Flexibilité
FP1	Voler et se diriger en vol, stabilisation	Vitesse maximum	~20km/h	F0
		Angle maximum	45°	F0
		Altitude maximale	15m	F0
		Poids		F0
		Diamètre		F0
FP2	Pilotage par onde radio	Fréquence	2,4GHz	F0

FS1	Affichage de l'état de la batterie sur la télécommande	Niveau batterie minimum	10%	F1
FS2	Mesurer l'altitude par rapport au sol	Distance maximum ultrason	6,45m	F0
FS3	Localiser le drone par GPS	Précision de la localisation	Au mètre près	F2
FS4	Prendre des photos et des vidéos de façon stable	Amplitude de déplacement de la caméra	180°	F2
FS5	Respecter l'environnement et résister aux conditions climatiques			F1
FS6	Viser les récepteurs infrarouges des autres quadricoptères	Distance infrarouge	10m	F3

**F0: Impératif - F1: Peu négociable - F2: Négociable - F3: Très négociable**

Dans ce tableau se trouve la deuxième fonction principale qui consiste à piloter le drone par ondes radios, mais elle a déjà été réalisée les années précédentes. Pour notre première fonction principale, tout est prioritaire. Nous avons donné des valeurs approximatives que nous essayerons d'atteindre

dans ce projet. Nous pensons d'abord avoir une vitesse maximale de 20 km/h, un angle d'inclinaison maximum de 45°, et une altitude de 15m pour l'instant.

Pour la fonction qui consiste à afficher l'état de la batterie sur la télécommande, nous avons choisi un seuil critique de 10% pour la batterie, avant par exemple de prévenir l'utilisateur.

Pour mesurer l'altitude par rapport au sol, nous allons utiliser un capteur ultrason qui mesurera jusqu'à 6,45m de haut. Pour les plus hautes altitudes, nous pensons utiliser un capteur barométrique.

Nous pensons aussi pouvoir localiser le drone par GPS, et nous espérons arriver à une précision au moins au mètre près.

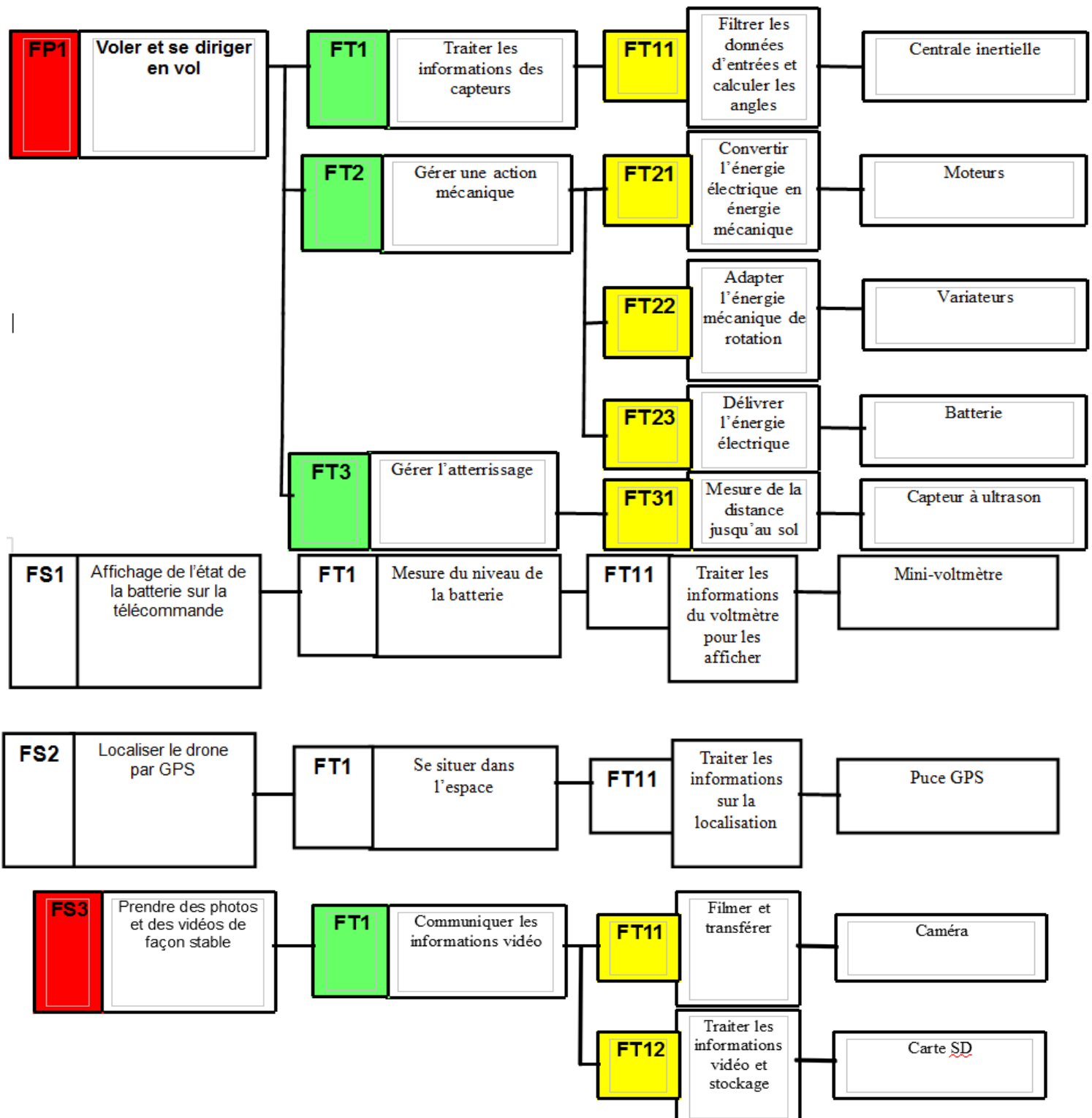
Pour prendre des photos et des vidéos de façon stable, nous utiliserons une caméra sur un support fixé au quadricoptère, et la caméra aura au moins une amplitude de déplacement de 180°.

Comme dit précédemment, le drone devra respecter l'environnement et résister à certains vents.

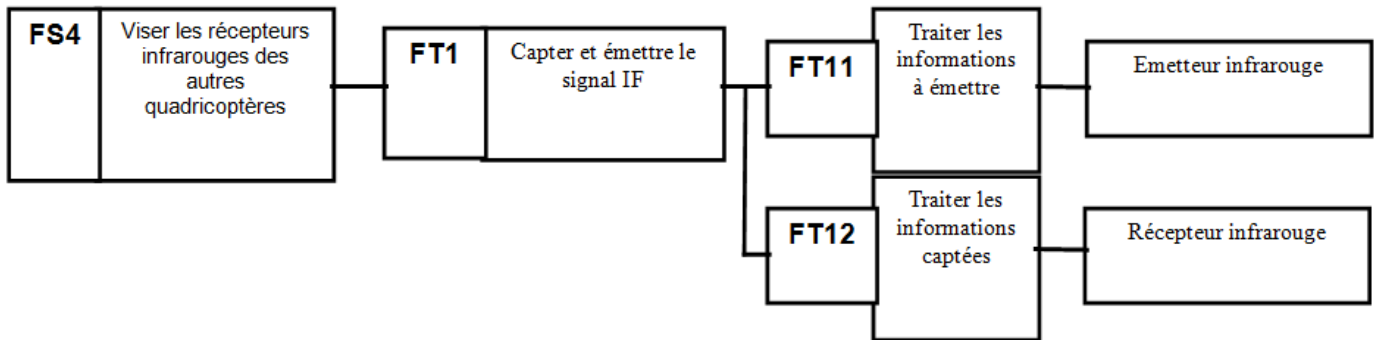
Enfin, pour le jeu de tir, le quadricoptère aura des récepteurs infrarouges et un émetteur, avec une portée d'au moins dix mètres.

Diagramme de FAST

Voici notre diagramme de FAST :

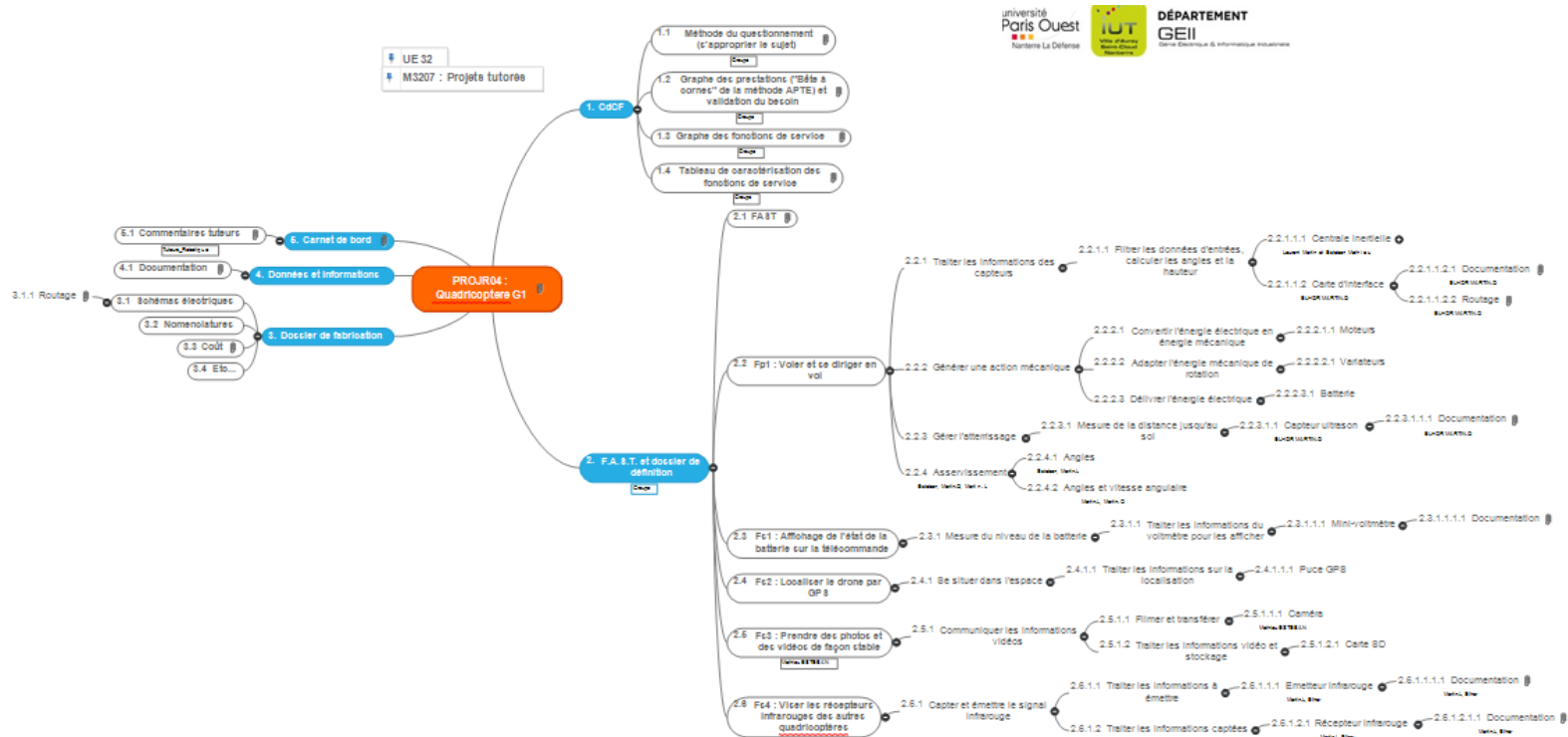






Ce diagramme de FAST nous montre les différentes fonctions de services ainsi que les différentes fonctions et solutions technologiques qui permettent de répondre à ces fonctions de service. Par exemple, nous voyons qu'il faut traiter les informations des capteurs pour voler et se diriger en vol, en filtrant les données d'entrée et en calculant les angles avec la centrale inertielle. Il en est de même pour l'affichage de la batterie. Il faut d'abord mesurer le niveau de la batterie ce qui n'est possible qu'en traitant les informations du voltmètre à l'aide d'un mini-voltmètre.

## Carte mentale



Sur cette carte mentale réalisée avec MindView, nous retrouvons tous les éléments importants pour l'avancement du projet. Nous retrouvons le diagramme de FAST avec les différentes fonctions, mais aussi la bête à cornes, le graphe des fonctions de services, le tableau de caractérisation des fonctions de services, les coûts dus aux composants, toutes les documentations mais aussi les fichiers de gravures.

Voici plus précisément les fonctions et les solutions envisagées avec les répartitions :



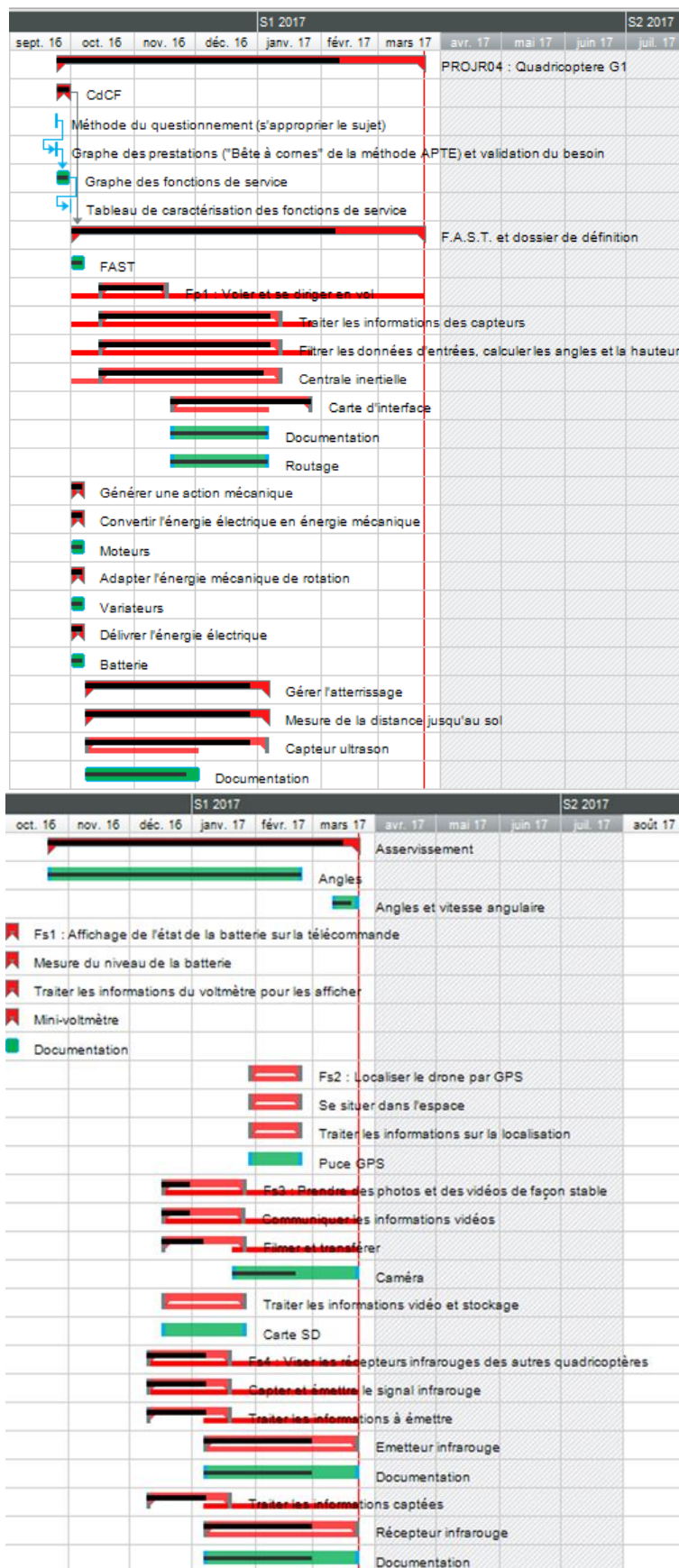
## Diagramme de Gantt

Voici notre diagramme de Gantt :

		Planifi...	Nom de tâche	Durée	Début	Fin	Prédé...	Progression	Priorité	Ressources
1			PROJR04 : <u>Quadricoptere G1</u>	22,5 jour...	23/09/2016	24/03/2017		77%	500	
2			1. CdCF	1,5 jours	23/09/2016	30/09/2016		100%	500	
3			1.1 Méthode du questionnement (s'approprier le sujet)	1 hr	23/09/2016	23/09/2016		100%	500	Groupe
4			1.2 Graphe des prestations ("Bête à cornes" de la méthode APTE) et validation du besoin	1 hr	23/09/2016	23/09/2016	3	100%	500	Groupe
5			1.3 Graphe des fonctions de service	2 hr	23/09/2016	30/09/2016	4	100%	500	Groupe
6			1.4 Tableau de caractérisation des fonctions de service	2 hr	30/09/2016	30/09/2016	5	100%	500	Groupe
7			2. F.A.S.T. et dossier de définition	21 jours?	30/09/2016	24/03/2017	2	75%	500	Groupe
8			2.1 FAST	4 hr	30/09/2016	07/10/2016		100%	500	
9			2.2 Fp1 : Voler et se diriger en vol	3,75 jours	14/10/2016	18/11/2016		94%	500	
10			2.2.1 Traiter les informations des capteurs	9,75 jours	14/10/2016	13/01/2017		94%	500	
11			2.2.1.1 Filtrer les données d'entrées, calculer les angles et la hauteur	9,75 jours	14/10/2016	13/01/2017		94%	500	
12			2.2.1.1.1 Centrale inertielle	9,75 jours	14/10/2016	13/01/2017		90%	500	Laurent Martin et Esteban M
17			2.2.1.1.2 Carte d'interface	9 jours	18/11/2016	27/01/2017		100%	500	ELHOR MARTIN.O
18			2.2.1.1.2.1 Documentation	5,75 jours	18/11/2016	06/01/2017		100%	500	ELHOR MARTIN.O
19			2.2.1.1.2.2 Routage	5,75 jours	18/11/2016	06/01/2017		100%	500	ELHOR MARTIN.O
20			2.2.2 Générer une action mécanique	1 jour?	30/09/2016	07/10/2016		100%	500	
21			2.2.2.1 Convertir l'énergie électrique en énergie mécanique	1 jour?	30/09/2016	07/10/2016		100%	500	
22			2.2.2.1.1 Moteurs	1 jour?	30/09/2016	07/10/2016		100%	500	
23			2.2.2.2 Adapter l'énergie mécanique de rotation	1 jour?	30/09/2016	07/10/2016		100%	500	
24			2.2.2.2.1 Variateurs	1 jour?	30/09/2016	07/10/2016		100%	500	
25			2.2.2.3 Délivrer l'énergie électrique	1 jour?	30/09/2016	07/10/2016		100%	500	
26			2.2.2.3.1 Batterie	1 jour?	30/09/2016	07/10/2016		100%	500	
27			2.2.3 Gérer l'atterrissage	9,75 jours	07/10/2016	06/01/2017		90%	500	
28			2.2.3.1 Mesure de la distance jusqu'au sol	9,75 jours	07/10/2016	06/01/2017		90%	500	
29			2.2.3.1.1 Capteur ultrason	9,75 jours	07/10/2016	06/01/2017		90%	500	ELHOR MARTIN.O
30			2.2.3.1.1.1 Documentation	7 jours	07/10/2016	02/12/2016		90%	500	ELHOR MARTIN.O
31			2.2.4 Asservissement	18,75 jo...	21/10/2016	24/03/2017		95%	500	Esteban, Martin.O, Martin.L
32			2.2.4.1 Angles	14,75 jours	21/10/2016	24/02/2017		100%	500	Esteban, Martin.L
33			2.2.4.2 Angles et vitesse angulaire	2,75 jours	10/03/2017	24/03/2017		70%	500	Martin.L, Martin.O
34			2.3 Fs1 : Affichage de l'état de la batterie sur la télécommande	1 jour?	30/09/2016	07/10/2016		0%	500	
35			2.3.1 Mesure du niveau de la batterie	1 jour?	30/09/2016	07/10/2016		0%	500	
36			2.3.1.1 Traiter les informations du voltmètre pour les afficher	1 jour?	30/09/2016	07/10/2016		0%	500	
37			2.3.1.1.1 Mini-voltmètre	1 jour?	30/09/2016	07/10/2016		0%	500	
38			2.3.1.1.1.1 Documentation	1 jour?	30/09/2016	07/10/2016		0%	500	
39			2.4 Fs2 : Localiser le drone par GPS	3,75 jours	28/01/2017	24/02/2017		0%	300	
40			2.4.1 Se situer dans l'espace	3,75 jours	28/01/2017	24/02/2017		0%	300	
41			2.4.1.1 Traiter les informations sur la localisation	3,75 jours	28/01/2017	24/02/2017		0%	300	
42			2.4.1.1.1 Puce GPS	3,75 jours	28/01/2017	24/02/2017		0%	300	
43			2.5 Fs3 : Prendre des photos et des vidéos de façon stable	5 jours	16/12/2016	27/01/2017		33%	300	Mathieu ESTEBAN
44			2.5.1 Communiquer les informations vidéos	4,25 jour...	16/12/2016	27/01/2017		33%	500	
45			2.5.1.1 Filmer et transférer	5 jours	16/12/2016	27/01/2017		50%	500	
46			2.5.1.1.1 Caméra	9,75 jours	20/01/2017	24/03/2017		50%	500	Mathieu ESTEBAN
47			2.5.1.2 Traiter les informations vidéo et stockage	5 jours	16/12/2016	27/01/2017		0%	500	
48			2.5.1.2.1 Carte SD	5 jours	16/12/2016	27/01/2017		0%	500	
49			2.6 Fs4 : Viser les récepteurs infrarouges des autres <u>quadricoptères</u>	4,75 jours	09/12/2016	20/01/2017		70%	500	
50			2.6.1 Capter et émettre le signal infrarouge	4,75 jours	09/12/2016	20/01/2017		70%	500	
51			2.6.1.1 Traiter les informations à émettre	4,75 jours	09/12/2016	20/01/2017		70%	500	
52			2.6.1.1.1 Emetteur infrarouge	11,75 jo...	06/01/2017	24/03/2017		70%	500	Martin.L, Elhor
53			2.6.1.1.1.1 Documentation	11,75 jours	06/01/2017	24/03/2017		70%	500	Martin.L, Elhor
54			2.6.1.2 Traiter les informations captées	4,75 jours	09/12/2016	20/01/2017		70%	500	
55			2.6.1.2.1 Récepteur infrarouge	11,75 jo...	06/01/2017	24/03/2017		70%	500	Martin.L, Elhor
56			2.6.1.2.1.1 Documentation	11,75 jours	06/01/2017	24/03/2017		70%	500	Martin.L, Elhor

Nous pouvons voir avec ce diagramme de Gantt que nous nous sommes répartis les tâches. Mathieu Esteban travaille sur la centrale inertielle et l'asservissement en angles avec Laurent Martin, puis il travaillera sur la caméra quand nous l'aurons à disposition. Laurent Martin travaille sur la centrale inertielle, puis sur les émetteurs et récepteurs infrarouges et sur les asservissements. Olivier Martin travaille sur le capteur ultrason, sur la carte d'interface avec la centrale inertielle et sur le double asservissement en angle et en vitesse angulaire. Enfin Daoud Elhor travaille sur la carte d'interface avec la centrale inertielle et sur les émetteurs et récepteurs infrarouges pour le jeu de tir.

Voici maintenant la répartition dans le temps des tâches :



## Centrale inertielle

La structure du message est de la forme suivante :

PREAMBLE	BID	MID	LEN	DATA	CHECKSUM
0xFA	0xFF	0x36	0x1E	DONNEE	0x0C

Notre travail a été de bien comprendre le programme de la centrale inertielle. Pour cela nous nous sommes aidés des compte rendus des anciennes années. A partir de là nous avons entrepris le travail de compréhension du programme pour mieux se l'approprier. Dans un premier temps nous nous sommes donc intéressé à la structure du message qui était envoyé de la centrale vers le microcontrôleur.

On a donc compris que le PREAMBLE, le BID, le MID, le LEN et le checksum sont des valeurs fixes qui nous indiquent que notre programme fonctionne. Ces trames doivent obligatoirement apparaître lors de la lecture de la trame. Si lors d'une lecture admettons que « MID » ne soit pas reçu on doit reprendre la lecture d'une nouvelle trame à 0. Cela permet d'assurer l'intégrité des informations reçues.

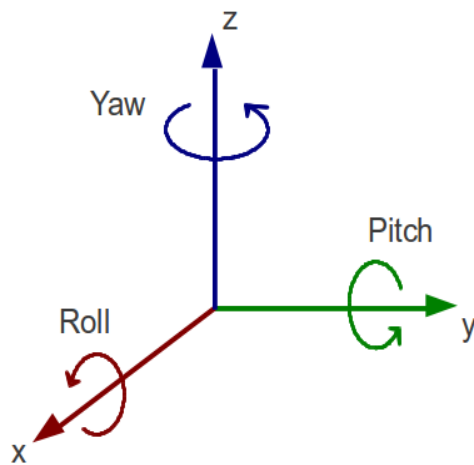
Nous allons donc détailler ici la schématisation d'une trame entière :



- 0xFA : Preamble
- 0xFF : BID (Bus Identifier)
- 0x36 : MID (Message Identifier) > MTData2
- 0x1E : All data Length
- 0x20 et 0x31 : Data ID > Angles
- 0x0C : Data length
- 0x40 et 0x31 : Data ID > Accélérations
- N : Numéro associé au « case » du programme.

L'orientation de notre quadricoptère est définie par les 3 angles Yaw/Pitch/Roll

**(Lacet/Tangage/Roulis) calculés par la centrale inertielle. Voici un schéma représentant ces 3 axes :**



Nous retrouvons donc dans cette trame complète nos valeurs fixes expliquées précédemment. Après « Len » nous avons un message, celui-ci correspond à l'accélération et se situe sur les cases 4 et 5. Nous recevons donc ici des valeurs de données aux case 4, 5 (accélération) puis 19 et 20 (Angle qui est l'information qui nous intéresse). Ces valeurs de données sont codées sur 2 octets. Voici un tableau récapitulant les valeurs de ces données :

	Valeur des données
Accélération	0x4031
Angle	0x2031
Vitesse angulaire	0x8021



Nous avons ensuite analysé les trames après les valeurs de données. Les valeurs dans les trames changent en fonctions des mesures faites par la centrale inertielle. La case 7 et 8 (seulement les 4 premiers bits de cet octet) correspondant à l'angle (ROLL) représentent la partie unitaire de ce que l'on reçoit de la centrale. Les 4 derniers bits de la case 8 correspondent à la partie décimale. Nous ne prendrons pas les deux octets suivant (case 9 et 10) car la précision est déjà assez bonne avec les octets déjà sélectionnés. Le programme fait la même chose ensuite pour les 2 premiers octets de PITCH et YAW. Dans notre programme les bits de la valeur de l'angle sont stockés dans un tableau `tab_result[]` qui pointe lui-même vers le tableau `angls[]` ou `vitesse_angulaire[]` :

```
static int*tab_result; //permet de faire pointer "tab_result"
vers le tableau "vitesse_angulaire[3]" ou "angls[3]" dans vars.t
```

```
case 5: //data ID (data reçues)
case 20: //data ID (data reçues)
    etat++;
    type_result = receiving | ((int)c & 0xff); // on masque
    switch (type_result) { //on test pour savoir dans le
        case 0x2031: //cas ou on reçoit l'angle
            tab_result = angls; // on fait pointer tab_r
            break;
        case 0x8021: //cas ou on reçoit la vitesse angul
            tab_result = vitesse_angulaire; // on fait p
            break;
    }
    break;
```

Une fois les valeurs de l'angle stockées, une fonction nommée « `centrale_read_angls()` » vient lire les angles et les associer en pointant vers la structure « `vars_t` ». Dans cette fonction nous avons multiplié les valeurs de l'angle X(ROLL), Y(PITCH) et Z(YAW) par 4 afin de décaler de 2 bits vers la gauche nos bits. Cela nous permet de rapprocher notre angle max près de 1000 pour comparer la consigne reçue par notre télécommande avec l'angle calculé par la centrale. A savoir que si on décale de 2 on obtient un angle max de 15 degrés (960 en décimal une fois décalé de 2). Dans le cas où l'on souhaite avoir un angle max plus grand (30 degrés par exemple) il faut décaler de 1 bits vers la gauche donc multiplier par 2 notre angle. Nous avons finalement lors de nos tests utilisé un angle d'inclinaison max de 30 degrés.

**Voici notre morceau de code qui stocke notre angle dans `AngleX/Y/Z` et notre vitesse angulaire dans `GyroX/Y/Z` :**

```
void centrale_read_gyro(struct vars_t *v) { //on récupère la vitesse angulaire
    // X axis
    v->GyroX = (vitesse_angulaire[0]);
    // Y axis
    v->GyroY = (vitesse_angulaire[1]);
    // Z axis
    v->GyroZ = (vitesse_angulaire[2]);
}

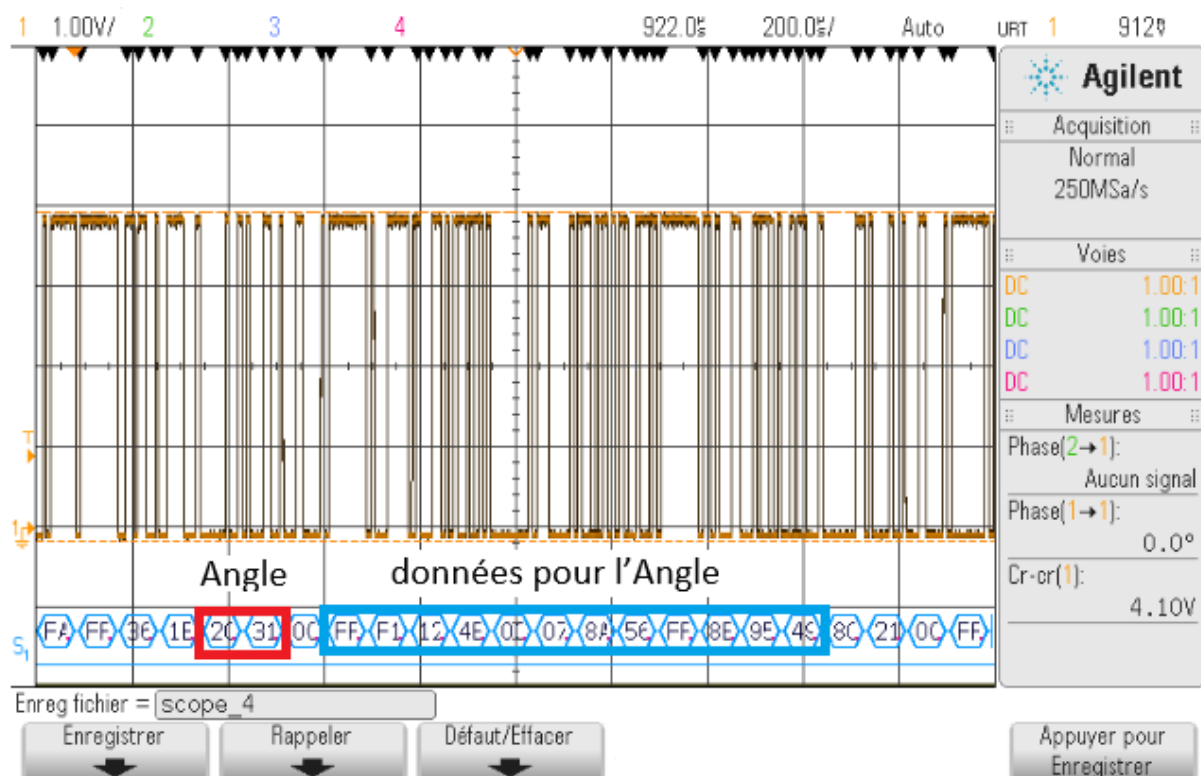
void centrale_read_angls(struct vars_t *v) { //on récupère l'Angle
    // X axis
    v->AngleX = -(angls[0])*2; //2 pour 30° //tangage//on multiplie par 2
    // pour décaler de 1 bit vers la gauche. Cela permet de rapprocher notre angle max de 1000 (valeur angle max telecommande)
    // pour comparer la consigne de la telecommande avec l'angle calculer par la centrale inertielle.
    // Y axis
    v->AngleY = -(angls[1])*2; //roulis
    // Z axis
    v->AngleZ = (angls[2])*2; //lacet
}
```



Cette fonction va permettre de réutiliser les angles dans « asservissement.c » pour comparer notre angle à la consigne de la télécommande. Nous avons dû modifier notre programme asservissement en modifiant RxInRoll et GyroX par RxInRoll\_Angle et AngleX (même chose pour le PITCH et le YAW) car nous n'utilisons plus le gyroscope. Une fois comparé, le programme « asservissement.c » stabilise le quadricoptère selon la consigne appliquée. Nous avons testé le programme une fois la carte d'interface de la centrale inertielle a été terminée.

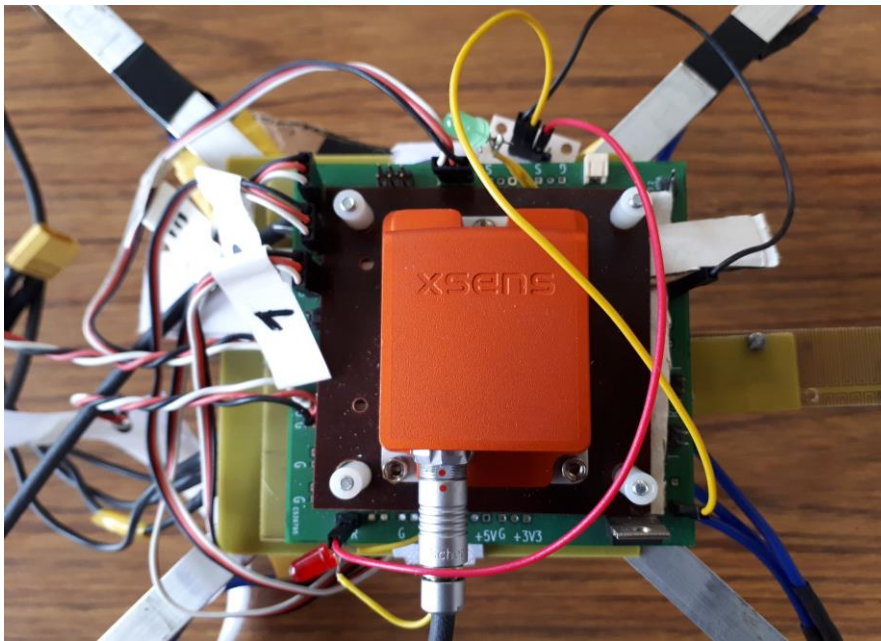
Lors de nos tests avec nos mains nous avons pu voir que le quadricoptère s'asservissait bien. Toutefois, lors du test de vol notre quadricoptère démarrait de travers. Nous nous sommes donc demandé si la centrale avait une position à plat qui renvoyait 45 degrés. Nous avons donc observé cela à l'oscilloscope et nous avons pu observer qu'à plat la centrale renvoyait bien aux alentours de 0 degrés. Nous nous sommes donc dirigés vers le réglage des coefficients mais cela n'a rien changé au problème d'instabilité au démarrage.

**Voici un screen qui permet d'observer la mesure de l'Angle sur l'oscilloscope :**



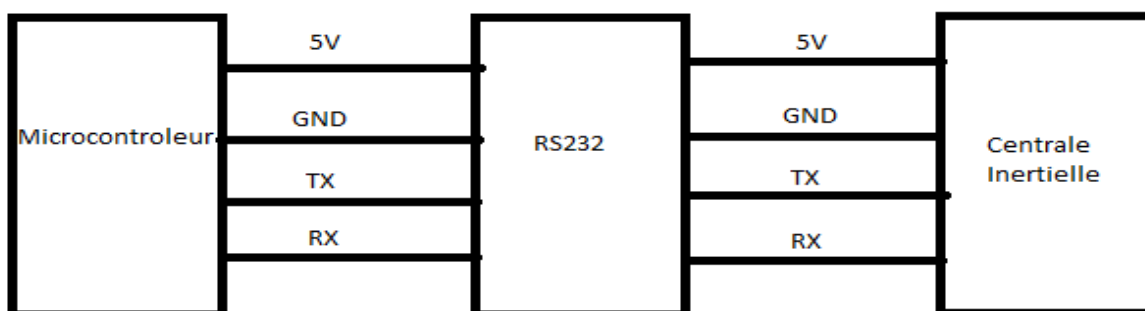
Nous avons également commenté tout le programme « central.c » afin que les prochaines années puissent se l'approprier plus rapidement que nous.

Voici une photo de la centrale fixée sur le quadricoptère :



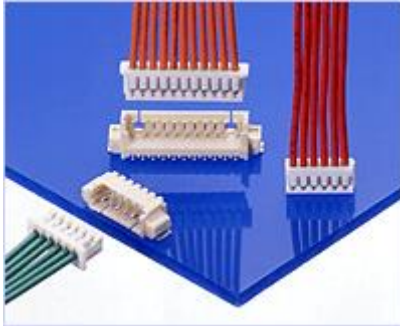
### Carte d'interface avec la centrale inertielle

La carte RS232 permet de prendre en charge l'interface entre la centrale inertielle et le capteur ultrason. En effet la centrale fournit des informations au microcontrôleur via une carte d'interface, voici un schéma de principe de l'interface :



La centrale nous a été fournie avec 2 câbles différents. En effet, il y en a un avec un connecteur USB afin de connecter la centrale inertielle à l'ordinateur pour programmer et transférer les programmes. Puis un autre câble avec un connecteur mâle 9 broches de type Picoblade.

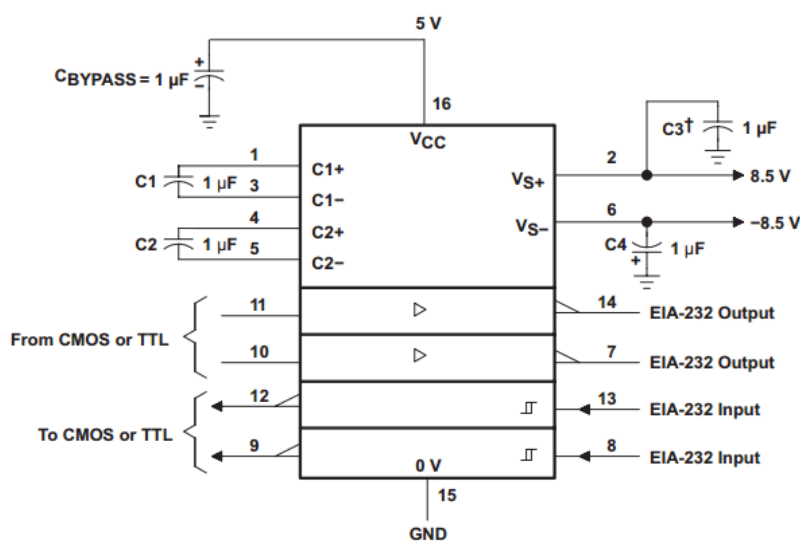
**Voici le connecteur 9 broches :**



Nous allons utiliser une carte RS232 composé d'un connecteur 4 broches qui sera relié au microcontrôleur du quadricoptère et un connecteur 9 broches pour alimenter la centrale et recueillir plus facilement les signaux Rx(Réception des données) et Tx(Transmission des données) à l'oscilloscope. Nous devons réaliser cette carte d'interface RS232 à l'aide d'un MAX232. Sachant que la centrale inertielle fonctionne avec des niveaux logiques en +/-12V. Pour être plus clair, le niveau « 0 » est formé par le - 12 V et le niveau « 1 » par le + 12 V. La carte RS232 va donc permettre d'adapter ces niveaux logiques entre la centrale inertielle et l'ATMEGA 1280 qui fonctionne en 0-5V.

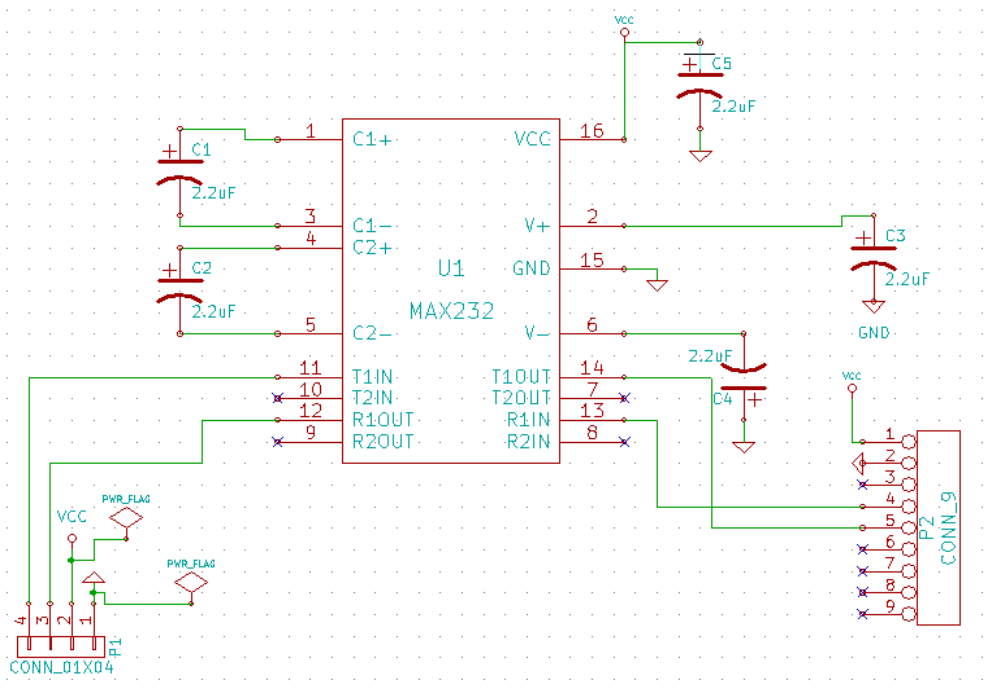
L'idée d'utiliser un MAX232 vient de nos prédécesseurs qui eux-mêmes ont dû remplacer la carte STK500 avec l'ATMEGA16 du début du projet. La carte STK 500 ne pouvant pas être transportée, elle a dû être ré-adapter.

**La datasheet du MAX232 nous donne le schéma du montage à réaliser :**



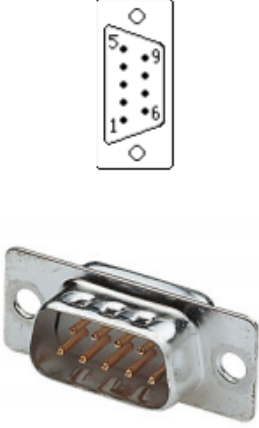
On retrouve 5 condensateurs polarisés de valeur 1  $\mu\text{F}$  ainsi que les bornes d'alimentations VCC et GND. Puis de chaque côté, on retrouve les entrées et sorties des signaux RX et TX sachant qu'on utilisera de chaque côté une entrée et une sortie.

Nous avons donc utilisé le logiciel Kicad, voici ci-dessous le schéma électrique de la carte :



Les 5 condensateurs CMS ont pour valeur 2.2  $\mu$ F car on fait en fonction de ce qu'il y a en stock à l'IUT. Cela ne change rien pour la suite. Ensuite on retrouve le connecteur 4 broches avec une broche pour l'alimentation VCC, une pour la masse GND et les broches pour la transmission et la réception des données.

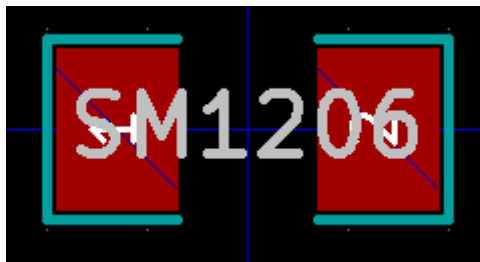
Pour le connecteur 9 broches, on retrouve sur une datasheet, les informations pour le branchement :

Broche	Signal	Définition	Sens du signal	Apparence
1	CD	Détection de porteuse	Entrée	
2	RXD	Réception	Entrée	
3	TXD	Emission	Sortie	
4	DTR	Terminal prêt à recevoir	Sortie	
5	GND	Masse	-	
6	DSR	Terminal prêt à émettre	Entrée	
7	RTS	Demande d'émission	Sortie	
8	CTS	Ok pour émettre	Entrée	
9	RI	Indicateur de sonnerie	Entrée	

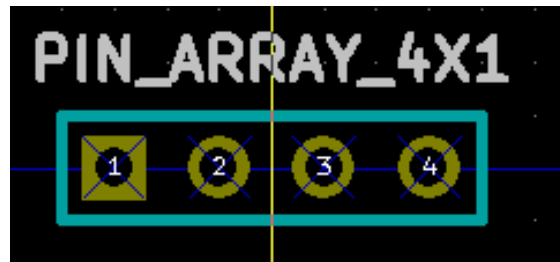
Ensuite, après avoir vérifié qu'il n'y a pas d'erreur sur le schéma, nous devons réaliser la Netlist :

1	C1 -	2.2uF : SM1206
2	C2 -	2.2uF : SM1206
3	C3 -	2.2uF : SM1206
4	C4 -	2.2uF : SM1206
5	C5 -	2.2uF : SM1206
6	P1 -	CONN_01X04 : PIN_ARRAY_4x1
7	P2 -	CONN_9 : connecteur_9broches_projet
8	P10 -	CONN_1 : 1pin
9	P11 -	CONN_1 : 1pin
10	P12 -	CONN_1 : 1pin
11	P13 -	CONN_1 : 1pin
12	U1 -	MAX232 : max232

Les références des 5 condensateurs ont été déduites à l'aide du technicien de l'IUT SM1206. Celle du connecteur 4 broches est dans la bibliothèque de Kicad aussi.



Longueur : 5.08 mm et largeur : 2.54mm

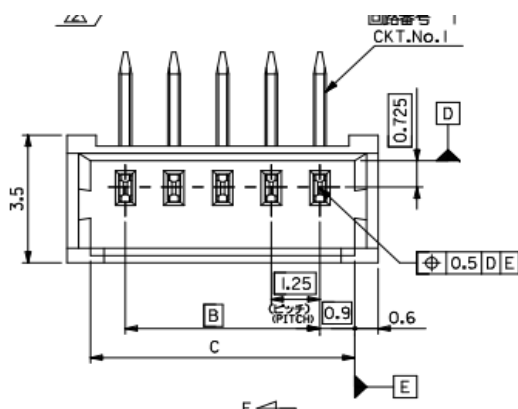


Longueur : 10.16mm et largeur : 2.54mm

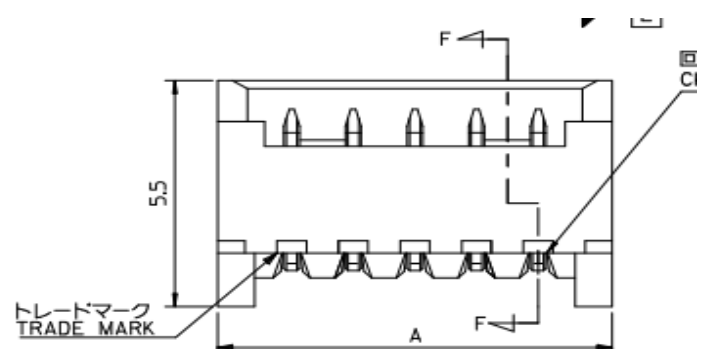
Nous devons réaliser les empreintes du connecteur 9 broches et du MAX232, pour ce faire nous analysons les datasheet de chaque composant.

- Empreinte du connecteur 9 broches :

- ❖ Vue de l'intérieur :



- Vue de dessus :



Etant donné que notre connecteur 9 broches a pour référence : MOLEX 53048-0910, les valeurs d'A, B et C sont : A = 13mm B = 10mm C = 11.8 mm

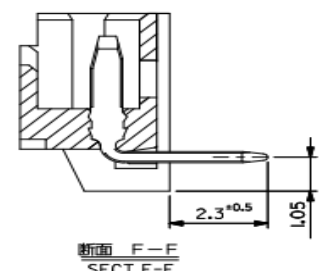
La longueur du composant est A = 13 mm, la hauteur est de 5.5 mm. La valeur B représente la distance entre le milieu du premier trou et le milieu du dernier trou. La valeur C représente la distance entre les 2 bords.

Voici ce qu'on a réalisé :

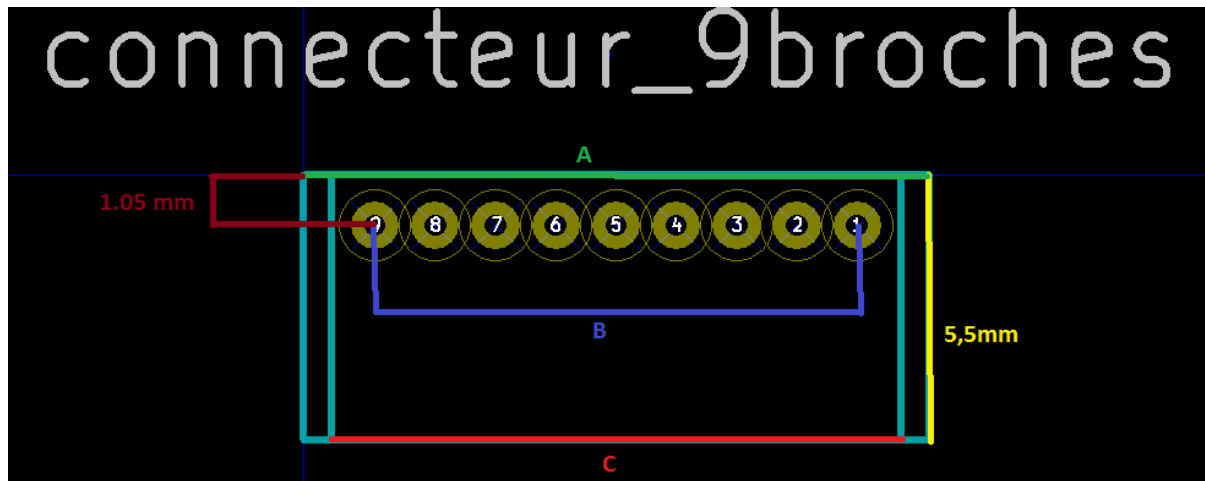


La dimension des trous de perçage est de 0.5 mm

19.3	17.5	20.5	53048-1510	15
18.05	16.25	19.25	-1410	14
16.8	15	18	-1310	13
15.55	13.75	16.75	-1210	12
14.3	12.5	15.5	-1110	11
13.05	11.25	14.25	-1010	10
11.8	10	13	-0910	9
10.55	8.75	11.75	-0810	8
9.3	7.5	10.5	-0710	7
8.05	6.25	9.25	-0610	6
6.8	5	8	-0510	5
5.55	3.75	6.75	-0410	4
4.3	2.5	5.5	-0310	3
3.05	1.25	4.25	53048-0210	2
C	B	A	ENG. NO.	極数
SCALE	DESIGN	MET		



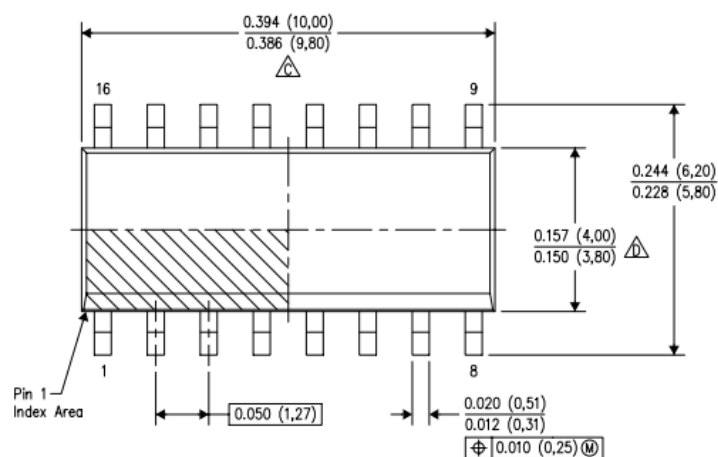
Voici les mesures correspondantes sur l’empreinte :



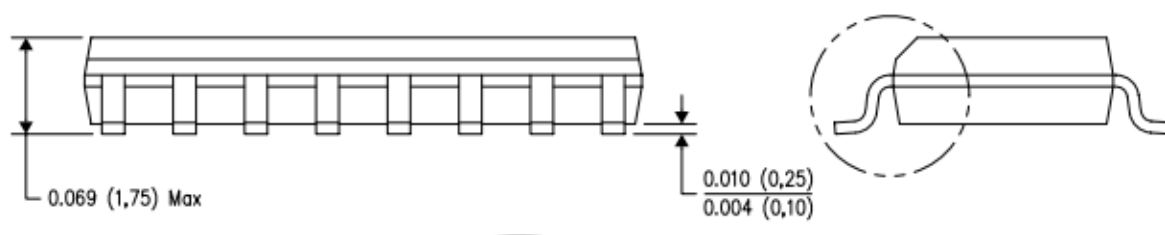
Nous n’oublions pas de rajouter une isolation de 0.2 mm lors du routage.

- Empreinte du MAX232 : Nous utilisons celui dont la référence est : D (R-PDSO-G16)

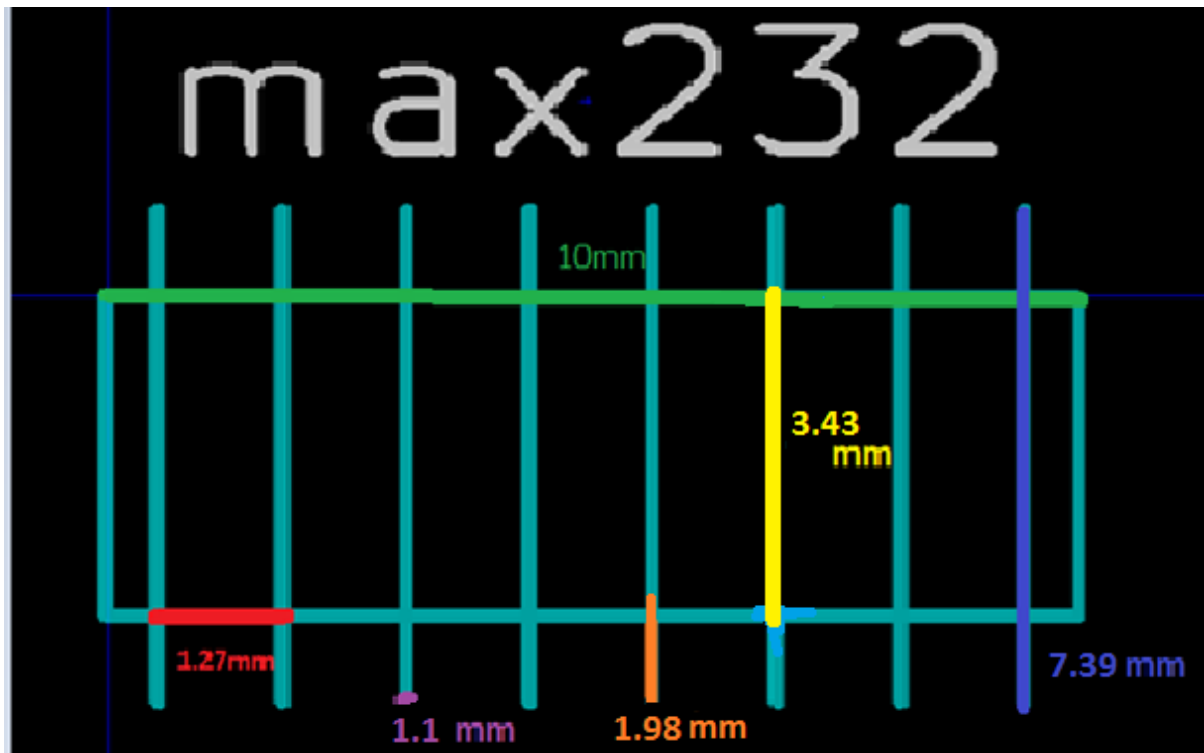
Vue du dessus :



Vue de profil :

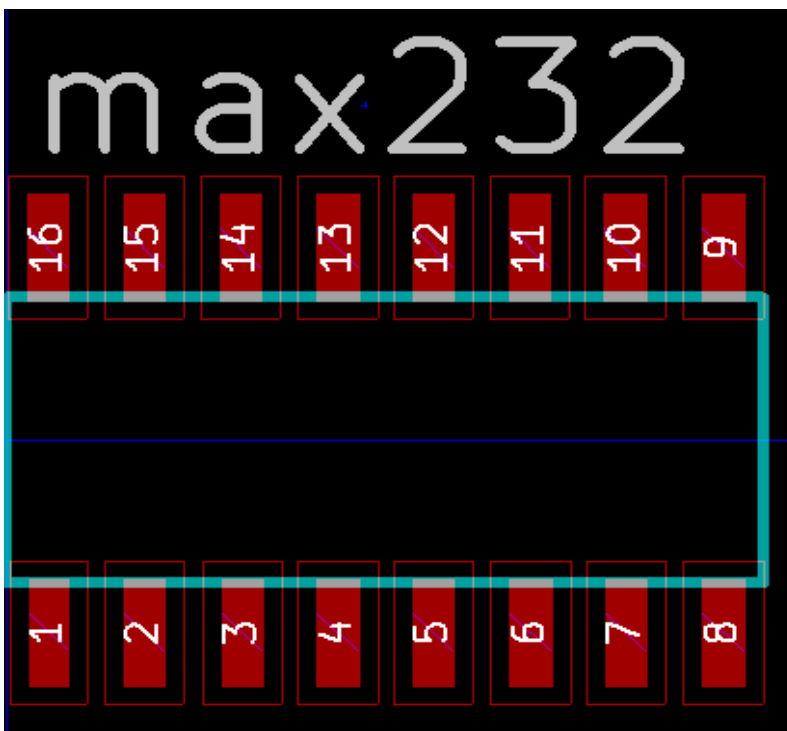


Voici le modèle de l'empreinte du MAX232 avec toutes les distances nécessaires :



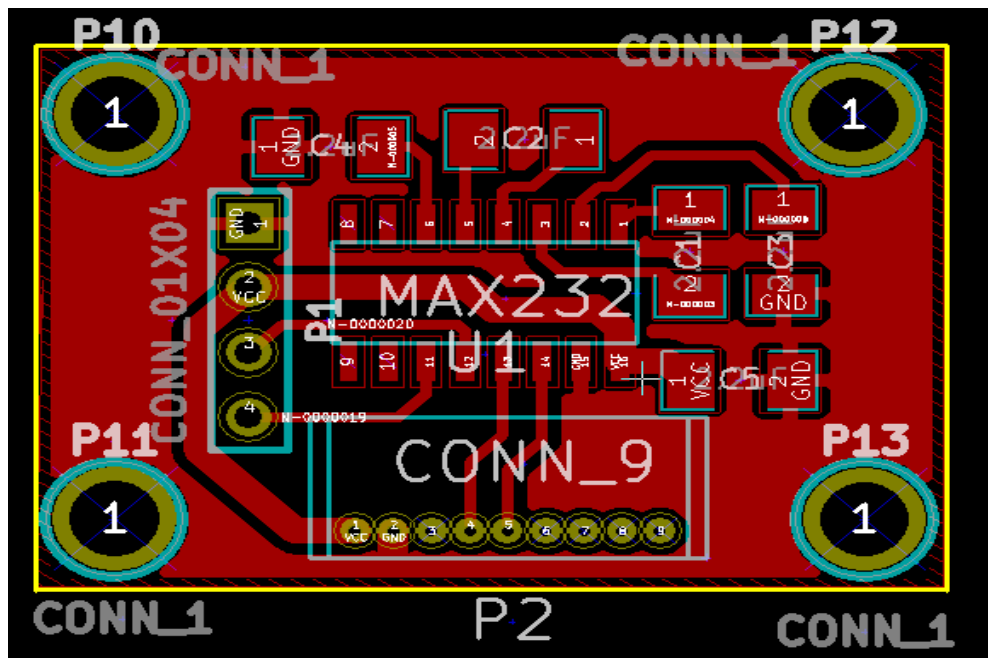
L'écart entre le milieu de chaque trou est de 1.27 mm, le diamètre de chaque broche est de 1.1 mm

Voici l'empreinte finale du MAX232 :





Voici la carte finale avec le plan de masse :



Voici ci-dessous les règles de conception, elles ont été choisies afin de faire réaliser la carte à l'iut.

Options Vias:		Valeurs Minimales Autorisées:	
Via par Défaut:		Largeur Min Piste (mm): 0,254	
<input checked="" type="radio"/> Via traversante <input type="radio"/> Via enterrée ou aveugle		Diamètre Min Via (mm): 0,889	
Micro Vias:		Perçage Min Via (mm): 0,500	
<input checked="" type="radio"/> Ne pas autoriser les micro vias <input type="radio"/> Autoriser les micro vias		Diamètre Min uVia (mm): 0,508	
		Perçage min uVia (mm): 0,127	

Classes d'Equipots:						
	Isolation	Epais. Piste	Diamètre Via	Perçage Via	Diamètre µVia	Perçage µVia
Default	0,240	0,400	1,000	0,500	0,508	0,127
Alimentation	0,240	1,000	1,000	0,500	0,508	0,127

Ajouter Enlever Vers le haut ^

Membres:

* (Any)	
Net	Classe
GND	Alimentation
VCC	Alimentation
	Default

* (Any)	
Net	Classe
GND	Alimentation
VCC	Alimentation
	Default

## Capteur ultrason

En ce qui concerne le capteur ultrason, nous avons d'abord voulu voir ce que renvoyait le capteur. Pour cela nous avons utilisé une liaison série. Nous avons connecté le capteur ultrason à une carte STK500, que nous avons reliée à l'ordinateur. Nous avons utilisé le logiciel PuTTY pour visualiser sur l'ordinateur les valeurs.

Dans le programme, nous avons tout mis en commentaire dans le main sauf `printf("Temps a l'etat haut =%lu \n", temps_etat_haut_US);`, car nous voulons juste regarder les valeurs renvoyées par le capteur. Il n'y a rien d'autre à faire car le capteur émet en permanence tant qu'il est alimenté.

**Sur PuTTY, nous avons réglé la vitesse à 115200 Baud comme indiqué sur le programme lors des années précédentes :**

```
/*TERMINAL:
115200 baud - Even Parity - 8bit of Data - 1 Stop Bit
```

Nous nous sommes alors rendu compte que cela ne fonctionnait pas, les caractères affichés à l'écran ne voulait rien dire. Nous avons fini par trouver que le problème venait de la fonction d'initialisation des interruptions. Il était en effet marqué que INT7 et INT6 (celle du capteur ultrason) étaient autorisées pour le front montant et descendant.

Or nous avons besoin du front descendant pour avoir le temps à l'état haut, qui est la valeur renvoyée par le capteur ultrason. **Il faut alors mettre d'après la documentation de l'ATmega1280 :**

**Table 15-3.** Interrupt Sense Control<sup>(1)</sup>

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Any logical change on INTn generates an interrupt request
1	0	The falling edge between two samples of INTn generates an interrupt request.
1	1	The rising edge between two samples of INTn generates an interrupt request.

**Soit dans le programme :**

```
EICRB = (1<<ISC61)|(1<<ISC71); // Falling Edge for INT7 and for INT6.
```

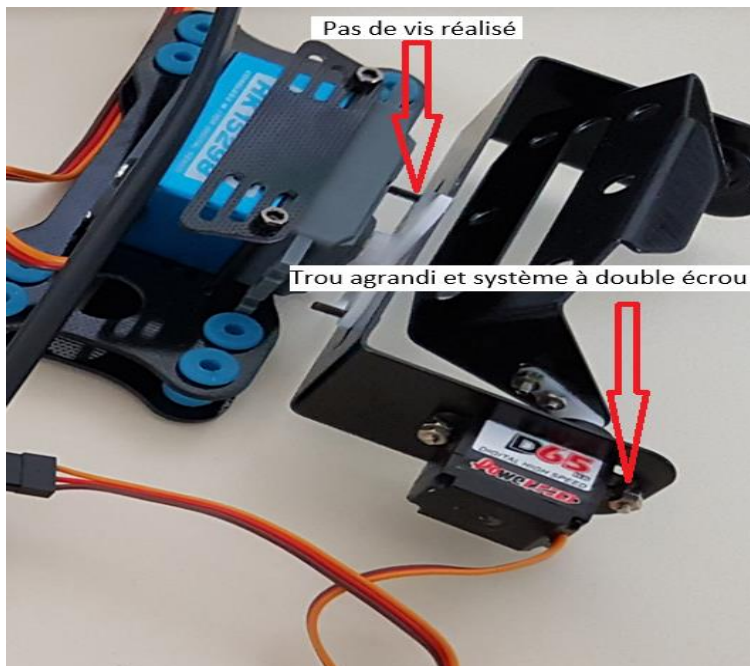
Nous avons pu alors voir que les valeurs étaient affichées sur l'écran. Cependant, malgré ces modifications cela ne fonctionnait toujours pas car la valeur affichée était toujours la même. Nous avons alors décidé d'observer les signaux renvoyés par le capteur à l'oscilloscope, et nous nous sommes rendu compte que quel que soit la distance à laquelle se trouvait l'obstacle du capteur, le signal de variait pas et le temps à l'état haut était toujours le même. Pour être sûr que c'était bien notre capteur ultrason qui était défaillant, nous avons emprunté le capteur du deuxième groupe quadricoptère et nous avons réalisé les mêmes manipulations : nous avons observé les signaux renvoyés par le capteur à l'oscilloscope. Mais cette fois, quand la distance variait, le temps à l'état haut variait aussi. Plus la distance était grande, plus le temps à l'état haut était grand. C'est donc notre capteur qui ne fonctionnait pas correctement. Pour la suite du projet, nous avons donc utilisé le capteur du deuxième groupe.

Nous avons alors pu visualiser les valeurs du temps à l'état haut sur l'ordinateur, et les valeurs affichées étaient cohérentes, le temps à l'état haut augmentait quand la distance augmentait et inversement.

## Camera

Dans un premier temps le travail a été de monter la fixation de caméra Gimbal qui a été reçue en pièce détachée. Le travail a été plutôt long car les trous de fixations des moteurs ne correspondaient pas toujours aux trous de notre Gimbal et il a également fallu faire le pas de vis sur le plastique qui reliait notre Gimbal à nos moteurs. Il a donc fallu étudier les différentes possibilités afin de pouvoir visser nos moteurs à la Gimbal sans trop endommager celle-ci. Nous avons donc décidé d'agrandir un des trous de notre Gimbal à l'aide d'une pince pour pouvoir y fixer notre petit servo moteur (DS65HB). En outre afin de garantir une bonne fixation de ce moteur nous avons opté pour un système à double écrou qui garantit une bonne fixation des moteurs. Ainsi les vis auront nettement moins de chance de se détacher en vol à cause des vibrations.

**Photo de la Gimbal une fois montée :**



Le travail qui suit a consisté à tester nos moteurs avec une carte STK500. Pour se faire nous avons créé un nouveau projet sur AvrStudio afin de tester les moteurs de notre caméra uniquement. Nous allons donc utiliser la PWM (**P**ulse **W**idth **M**odulation) que nous réaliserons à l'aide du Timer 1 en mode Fast PWM :

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Mode opératoire	TOP	Mise à jour d'OCR1x	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immédiate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immédiate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase et Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase et Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immédiate	MAX
13	1	1	0	1	Réservé	–	–	–
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

Le registre TCNT1 est délimité par une valeur min qui est 0 et max qui est ICR1. Il a également une valeur seuil qui est fixé par OCR1A/B. Le Timer 1 utilise OCR1A/B pour le comparer à TCNT1. La comparaison mettra à jour le drapeau OCF1A/B qui peut être employé pour produire une demande d'interruption. Le drapeau OCF1A/B est automatiquement mis à 0 quand l'interruption est exécutée. En cas d'égalité (TCNT1= OCR1A/B), le compteur TCNT1 a atteint la valeur contenue dans OCR1A/B alors une interruption est générée. Cette comparaison permet de gérer les signaux OC1A/B sur les broches PD4 et PD5 en sortie du « Waveform Generator ».

Nous avons donc dû ensuite sélectionner l'horloge qui devra être utilisé pour le Timer1 afin de pouvoir générer un signal de 50Hz. Les calculs suivant expliquent les valeurs utilisées pour le pré diviseur et ICR1 (registre du TOP).

Nous savons que la fréquence de notre microcontrôleur est de 3684000 Hz. Nous avons divisé cette valeur par un pré diviseur de 8 :

$$3684000 : 8 = 460500 \text{ Hz}$$

Nous voulions 50 Hz (20 ms) donc :

$$20 \cdot 10^{-3} \cdot 460500 = 9210$$

ICR1 aura comme valeur 9210 pour que nous ayons une largeur d'impulsion de 50 Hz.

Nous savons d'après la doc de notre servo moteur que pour avoir 0 degré d'inclinaison il faut générer un signal de 1,5 ms :

$$460500 \cdot 1,5 \cdot 10^{-3} = 690 \text{ qui est la valeur à appliquer à OCR1A pour obtenir 0 degré.}$$

### Choix du pré diviseur :

CS12	CS11	CS10	Description
0	0	0	Stop timer0
0	0	1	CK
0	1	0	CK/8
0	1	1	CK/64
1	0	0	CK/256
1	0	1	CK/1024
1	1	0	Pin T1 ↓ (PB1)
1	1	1	Pin T1 ↑ (PB1)

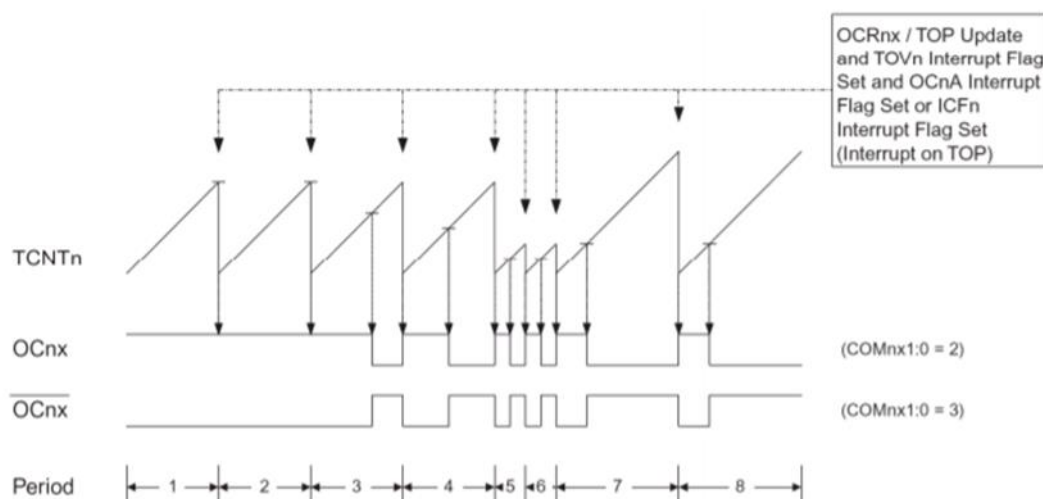
**Nous avons également sélectionné nos modes de comparaison pour nos 2 voies :**

COM1A1 COM1B1	COM1A0 COM1B0	Description
0	0	Opération normale du port. OC1A et OC1B sont déconnectés.
0	1	WGM13:0 = 15: Bascule OC1A quand il y a correspondance, OC1B est déconnectés. Pour les autres valeurs de WGM1 OC1A et OC1B sont déconnectés.
1	0	OC1A/OC1B = 0 quand il y a correspondance, Met OC0 à 1 au TOP
1	1	OC1A/OC1B = 1 quand il y a correspondance, Met OC0 à 0 au TOP

COM1A1, COM1A0 : Mode comparaison pour la voie A.

COM1B1, COM1B0 : Mode comparaison pour la voie B.

Le registre TCNT1 est le registre de comptage, il compte des fronts de l'horloge sélectionnée par TCR1A et TCR1B. Ce registre est comparé en permanence au registre OCR1A/B et le résultat peut générer des formes d'onde PWM sur les broches OC1A/B :



On peut constater que lorsqu'une interruption est générée (valeur seuil de OCR1A/B atteinte par TCNT1) alors le rapport cyclique OC1A/B varie. Cela permet d'obtenir nos temps d'impulsions souhaités pour contrôler nos moteurs.

Voici la première version du code réalisée :

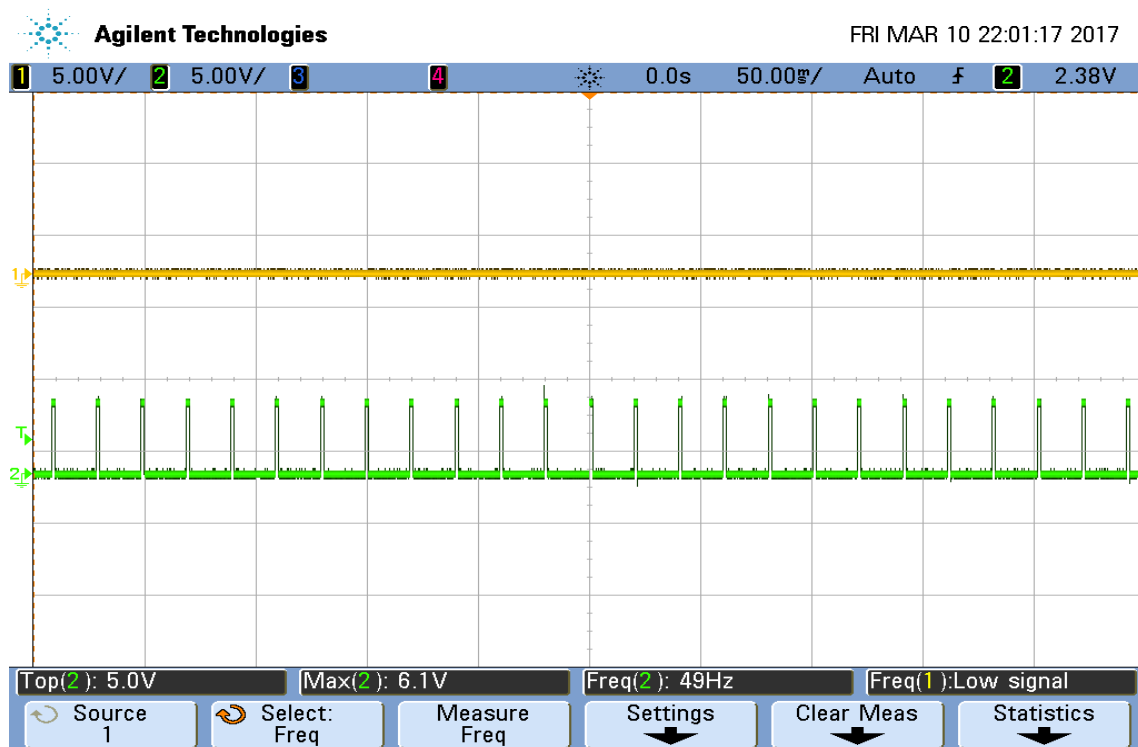
```

/*
 * camera.c
 *
 * Created: 07/03/2017 15:08:45
 * Author: Mathieu
 */
#include <avr/io.h>
#include <util/delay.h>
void Wait()
{
    uint16_t i;
    for(i=0;i<50;i++)
    {
        _delay_loop_2(0);
        _delay_loop_2(0);
    }
}
void main()
{
    //FOR TIMER1
    TCCR1A|=(1<<COM1A1) | (1<<COM1B1) | (1<<WGM11);          //NON Inverted
    PWM
    TCCR1B|=(1<<WGM13) | (1<<WGM12) | (1<<CS11) | (0<<CS10);    //PRESCALER=8
    MODE 14 (FAST PWM) (460500Hz timer 1) //PRESCALER=64 pour 16*10^6Hz
    ICR1=9210; //fPWM=50Hz, on règle la largeur d'impulsion //ICR1=4999
    pour 16*10^6Hz
    DDRD|=(1<<PD4) | (1<<PD5);    //PWM Pins as Output
    while(1)
    {
        OCR1A=460;    //-45 degree 1ms
        Wait();
        OCR1A=690;    //0 degree 1,5ms
        Wait();
        OCR1A=920;    //45 degree 2ms
        Wait();
        OCR1A=690;    //0 degree 1,5ms
    }
}

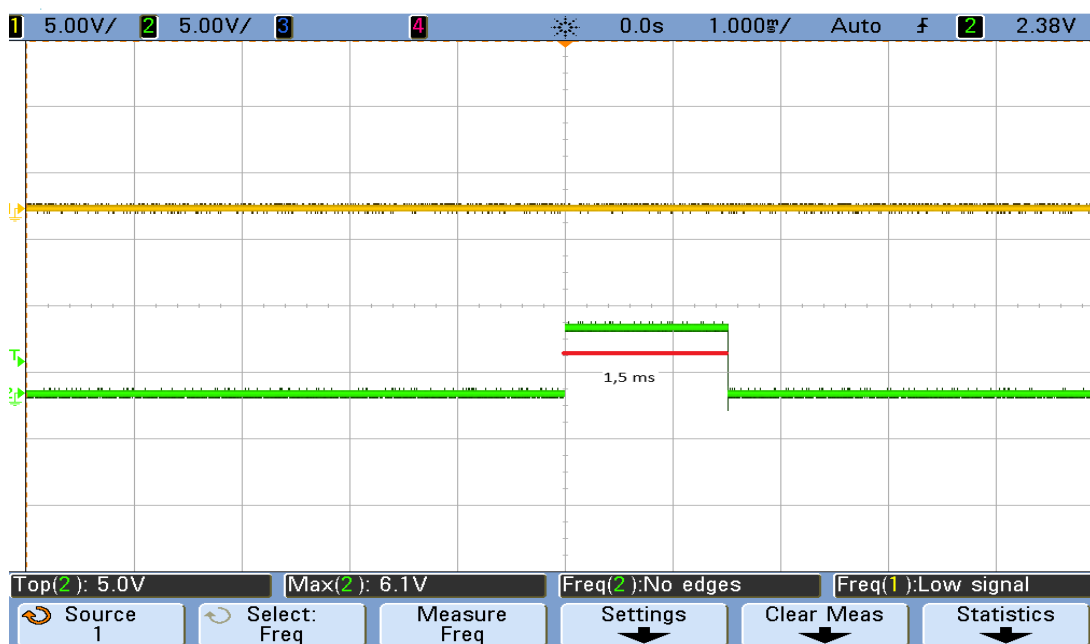
```

Nous avons donc observé à l'oscilloscope ce que nous obtenions en sortie de notre carte STK500 :

**Nous observons que notre signal généré en sortie est bien un signal d'environ 50 Hz :**

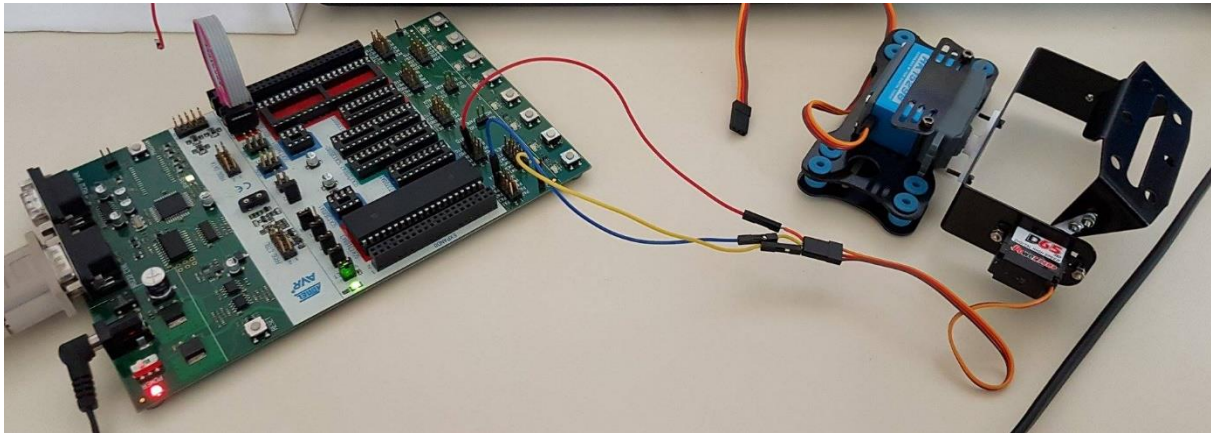


**Lorsque nous voulons générer 1,5 ms pour obtenir 0 degrés sur notre moteur nous obtenons bien 1,5 ms à l'oscilloscope :**





Nous avons donc relié notre caméra à notre carte STK500 en respectant bien les couleurs de fil. En partant du servo moteur, le fil marron est relié à la masse (GND), le rouge est relié à l'alimentation (VTG) et l'orange est le câble d'information qui est relié à PD5 :

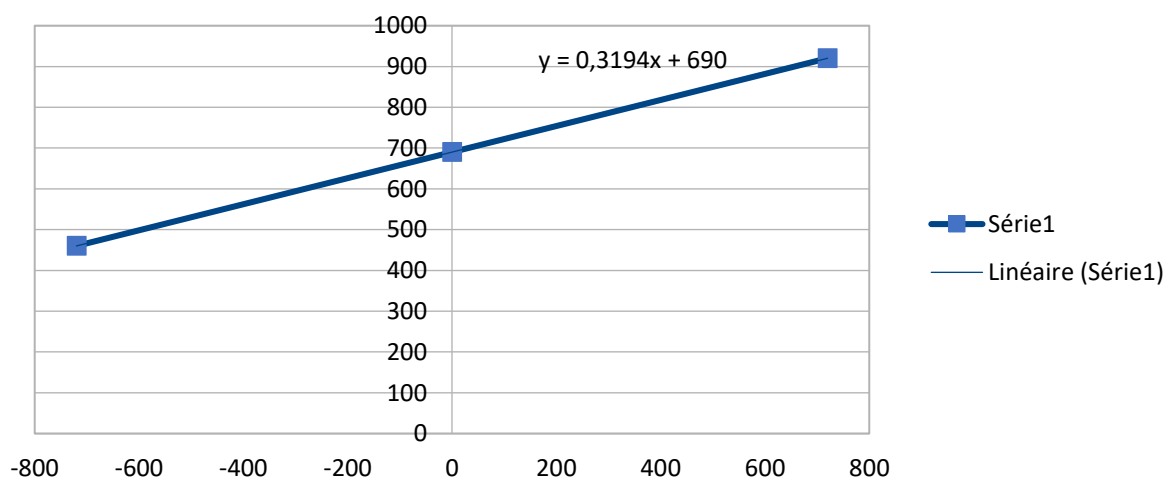


Notre servo moteur inclinait aux bons angles le socle de la caméra.

Nous devons maintenant adapter notre programme au programme de notre quadricoptère. Pour se faire nous devons trouver l'équation qui permettra de contrôler notre caméra en fonction de l'angle mesurée par la centrale :

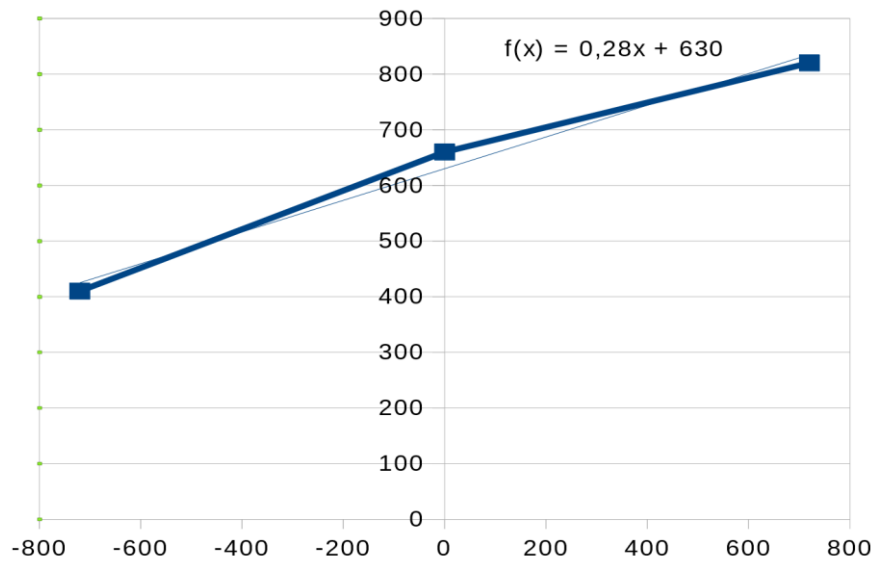
Degré	Base 16	OCR1X
-45	-720	460
0	0	690
45	720	920

Nous savons que notre centrale fonctionne en base 16. Nous avons donc multiplié notre angle souhaité en degré afin d'obtenir les bonnes valeurs qui seront récupérer dans notre équation. Nous avons donc tracé OCR1X en fonction de notre angle multiplié par 16. **Nous avons ensuite extrait l'équation de notre courbe :**





Nous avons ensuite réalisé les mêmes étapes pour notre second servo moteur (HK15298) et nous en avons déduit l'équation à l'aide de la courbe tracée :



A partir de ces informations nous avons réalisé le programme qui sera à rajouter dans le programme complet du projet quadricoptère en y spécifiant les ports de sortie (potentiellement autre que PD4 et PD5) disponibles sur la carte. **Notre programme est le suivant :**

```
#include <avr/io.h>
#include <util/delay.h>
void Wait()
{
    uint16_t i;
    for(i=0;i<50;i++)
    {
        _delay_loop_2(0);
        _delay_loop_2(0);
    }
}
void main()
{
    int AngleY;
    //FOR TIMER1
    TCCR1A|=(1<<COM1A1)|(1<<COM1B1)|(1<<WGM11); //NON Inverted PWM
    TCCR1B|=(1<<WGM13)|(1<<WGM12)|(1<<CS11)|(0<<CS10); //PRESCALER=8 MODE
14(FAST PWM)(460500Hz timer 1)//PRESCALER=64 pour 16*10^6Hz

    ICR1=9210; //fPWM=50Hz, on règle la largeur d'impulsion //ICR1=4999 pour
16*10^6Hz

    DDRD|=(1<<PD4)|(1<<PD5); //PWM Pins as Output //OC1A sur PD5 //OC1B sur
PD4

    while(1)
    {
        OCR1A=0.32*AngleY+690; //valeur de comparaison
        OCR1B=0.28*AngleY+630; //valeur de comparaison
    }
}
```

## Asservissements

Quand nous avons repris le projet, nous avons dû modifier une première fois légèrement l'asservissement pour qu'il soit réalisé en fonction de l'angle.

```
y = pid_Controller(v->RxInRoll_Angle, v->AngleX, &pidData_roll);
```

```
y = pid_Controller(v->RxInPitch_Angle, v->AngleY, &pidData_pitch);
```

```
y = pid_Controller(-v->RxInYaw_Angle, v->AngleZ, &pidData_yaw);
```

Dans les paramètres de la fonction qui calcule le PID, nous avons mis l'angle reçu de la télécommande (RxIn...\_Angle), ainsi que la valeur de l'angle renvoyée par la centrale inertielle (Angle...).

Nous avons alors réalisé plusieurs tests en tenant le quadricoptère à la main puis en extérieur afin d'essayer de régler **les coefficients suivant pour l'asservissement** :

```
//Grâce à de nombreux test sur le comportement du quadricoptère on trouve les coefficient suivant
#define K_ROLL_P    0.08    //Amélioration du coefficient P de roulis
#define K_ROLL_I    0.00
#define K_ROLL_D    0.00

#define K_PITCH_P    0.08    //Amélioration du coefficient P de tangage grâce à de nombreux tests
#define K_PITCH_I    0.00
#define K_PITCH_D    0.00

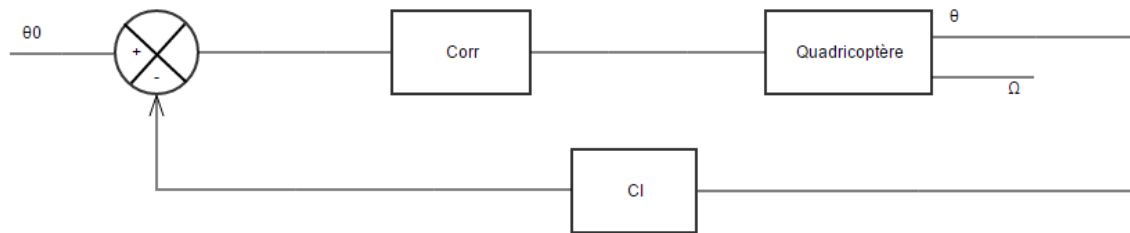
#define K_YAW_P      0.05
#define K_YAW_I      0.0
#define K_YAW_D      0.00
```

Le coefficient du Yaw est plus faible car il représente l'orientation en fonction du nord et du sud ce qui est moins important. Nous avons même parfois mis  $y=0$  pour le calcul du PID du Yaw pour faire nos tests afin de les faciliter.

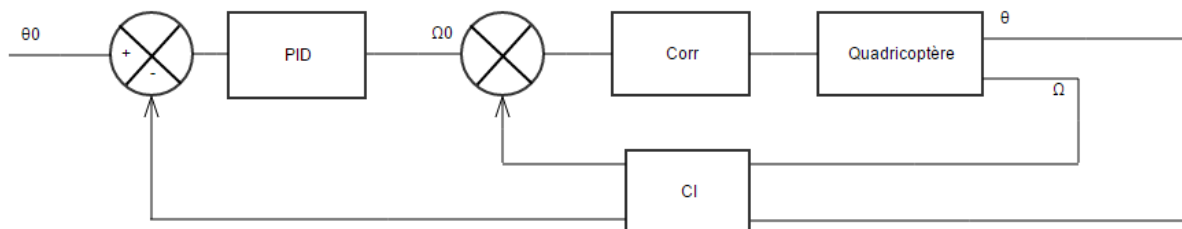
Malheureusement malgré tous nos tests nous n'avons jamais réussi à avoir un quadricoptère stable. En effet, nous avons une marge de phase qui dépasse  $-180^\circ$  ce qui rend le quadricoptère instable.

Nous avons donc repris un fichier simulink des années précédentes où ils réalisaient un double asservissement, en angle et en vitesse angulaire afin d'obtenir une marge de phase de  $-180^\circ$ .

Voici l'asservissement en angle que nous avons réalisé :



Voici maintenant le double asservissement que nous souhaitons réaliser :



Pour réaliser cet asservissement, nous avons donc dû d'abord décommenter la fonction qui permet de recevoir la vitesse angulaire de la centrale inertielle :

```
void centrale_read_gyro(struct vars_t *v) {    //lire gyro
    // X axis
    v->GyroX = (vitesse_angulaire[0]);
    // Y axis
    v->GyroY = (vitesse_angulaire[1]);
    // Z axis
    v->GyroZ = (vitesse_angulaire[2]);
}
```

Puis nous avons écrit une deuxième ligne permettant de calculer un deuxième PID pour chaque axe :

```
y = pid_Controller(v->RxInRoll_Angle, v->AngleX, &pidData_roll);
y = pid_Controller(y, v->GyroX, &pidData_roll2);

y = pid_Controller(v->RxInPitch_Angle, v->AngleY, &pidData_pitch);
y = pid_Controller(y, v->GyroY, &pidData_pitch2);

y = pid_Controller(-v->RxInYaw_Angle, v->AngleZ, &pidData_yaw);
y = pid_Controller(y, v->GyroZ, &pidData_yaw2);
```

On ne reçoit cette fois plus une commande de la télécommande mais plutôt ce qui avait été calculé à la ligne précédente, ici  $y$ . Ce deuxième asservissement se faisant avec la vitesse angulaire, il faut donc ajouter GyroX, GyroY et GyroZ qui reçoivent la vitesse angulaire de la centrale inertielle. Enfin on envoie cela dans une deuxième donnée pour le PID (ici "pidData\_roll2" pour le roll par exemple).

**Il a donc fallu définir de nouveaux coefficients pour la vitesse angulaire :**

```
//Pour la vitesse angulaire
#define K_ROLL_P2      1.0
#define K_ROLL_I2      0.00
#define K_ROLL_D2      0.00

#define K_PITCH_P2     1.0
#define K_PITCH_I2     0.00
#define K_PITCH_D2     0.00

#define K_YAW_P2       0.05
#define K_YAW_I2       0.0
#define K_YAW_D2       0.00
```

Nous avons réglé les coefficients P du roll et du pitch à 1 car nous souhaitons avoir une vitesse angulaire de 30°/seconde. Le P du yaw a été laissé à 0,05 car il n'est pas important pour nos tests.

**Il a également fallu définir des nouvelles structures pour les "pidData" :**

```
volatile unsigned char fin;
struct PID_DATA pidData_roll;
struct PID_DATA pidData_pitch;
struct PID_DATA pidData_yaw;

struct PID_DATA pidData_roll2;
struct PID_DATA pidData_pitch2;
struct PID_DATA pidData_yaw2;
```

**Enfin nous avons mis à jour les fonctions init et reset des pid des moteurs :**

```
void init_motors_pids() {
    pid_Init(K_ROLL_P * SCALING_FACTOR, K_ROLL_I * SCALING_FACTOR, K_ROLL_D * SCALING_FACTOR, &pidData_roll);
    pid_Init(K_PITCH_P * SCALING_FACTOR, K_PITCH_I * SCALING_FACTOR, K_PITCH_D * SCALING_FACTOR, &pidData_pitch);
    pid_Init(K_YAW_P * SCALING_FACTOR, K_YAW_I * SCALING_FACTOR, K_YAW_D * SCALING_FACTOR, &pidData_yaw);

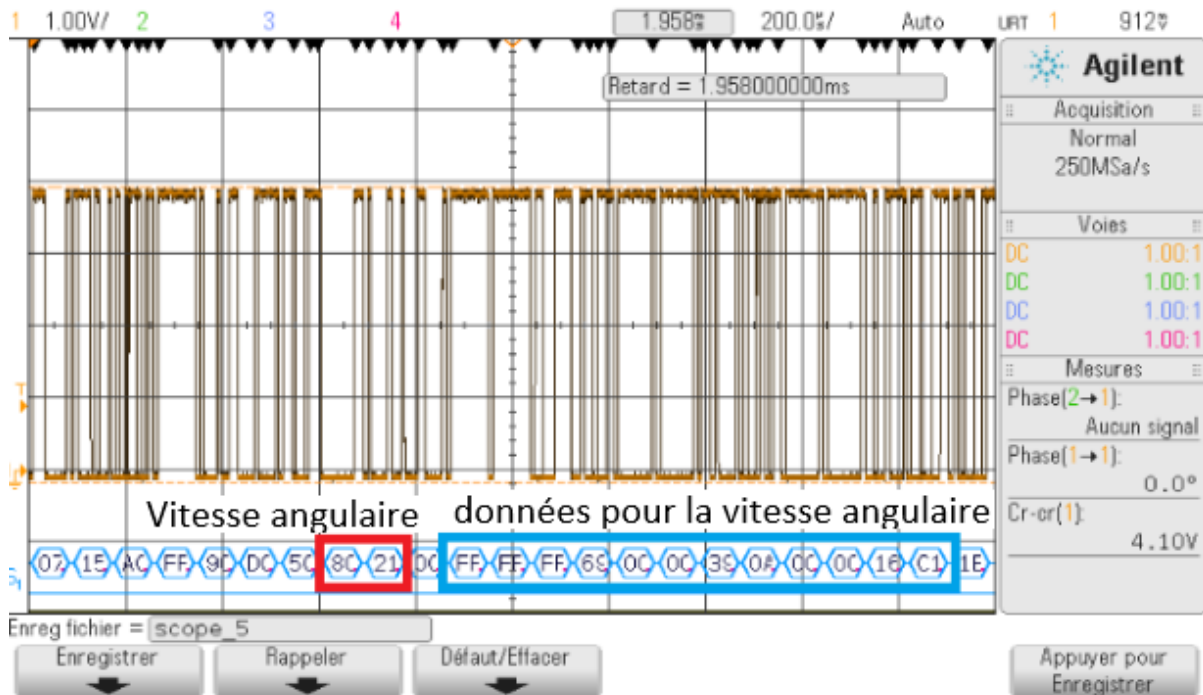
    pid_Init(K_ROLL_P2 * SCALING_FACTOR, K_ROLL_I2 * SCALING_FACTOR, K_ROLL_D2 * SCALING_FACTOR, &pidData_roll2);
    pid_Init(K_PITCH_P2 * SCALING_FACTOR, K_PITCH_I2 * SCALING_FACTOR, K_PITCH_D2 * SCALING_FACTOR, &pidData_pitch2);
    pid_Init(K_YAW_P2 * SCALING_FACTOR, K_YAW_I2 * SCALING_FACTOR, K_YAW_D2 * SCALING_FACTOR, &pidData_yaw2);
}

void reset_motors_pids() {
    pid_Reset_Integrator(&pidData_roll);
    pid_Reset_Integrator(&pidData_pitch);
    pid_Reset_Integrator(&pidData_yaw);

    pid_Reset_Integrator(&pidData_roll2);
    pid_Reset_Integrator(&pidData_pitch2);
    pid_Reset_Integrator(&pidData_yaw2);
}
```

Mais cela n'a pas résolu nos problèmes car lors de nos essais à la main, plus aucun asservissement n'était effectué. Nous pensons donc que la centrale inertielle n'est pas configurée pour renvoyer la vitesse angulaire. Nous n'avons pas eu le temps de vérifier cela à l'oscilloscope. Si c'est le cas, il faudrait utiliser le logiciel de Xsens pour la reconfigurer.

Toutefois un screen de plusieurs semaines indique que la centrale renvoie bien la vitesse angulaire :



Il faut toutefois quand même refaire les tests pour confirmer que c'est toujours le cas.

### Procédure de test du quadricoptère à la main

Afin de tester le programme sur le quadricoptère en le tenant à la main (par le dessous) sans la télécommande, il faut adapter le programme.

Voici ce qu'il faut ajouter :

```
v.RxChannels[0] = 200; //puissance
v.RxChannels[1] = 0; //roll
v.RxChannels[2] = 0; //pitch
v.RxChannels[3] = 0; //yaw

ConvertRxChannels(&v);

v.FlagArmed = 0;
v.AsserHaut = 0;
```

Ce morceau de programme s'ajoute dans le "do{}while(1)" du main. "RxChannels[0]" correspond à la puissance de moteur. "RxChannels[1]" correspond au Roll. "RxChannels[2]" correspond au Pitch. "RxChannels[3]" correspond au yaw. Ces quatre commandes simulent la télécommande, il faut donc ajouter après la ligne qui appelle la fonction "ConvertRxChannels(&v)". Ici nous avons mis une puissance de 200 et avons mis 0 dans Roll et Pitch afin que le quadricoptère s'asservisse en restant plat. La ligne "v.FlagArmed=0" permet de ne pas démarrer les moteurs. Il faut mettre 1 pour démarrer les moteurs puis remettre 0 si on veut arrêter le quadricoptère. Enfin il est important de mettre la ligne "v.AsserHaut=0" car sinon le quadricoptère va effectuer un asservissement en altitude et donc empêcher d'augmenter la puissance des moteurs.

## Emetteur et récepteurs infrarouges

Nous avons utilisé un émetteur et un récepteur pour nos tests. Les émetteurs infrarouges sont des TSAL6400 et les récepteurs infrarouges sont des TSOP34838.

Voici les différentes sorties pour notre récepteur TSOP34838:



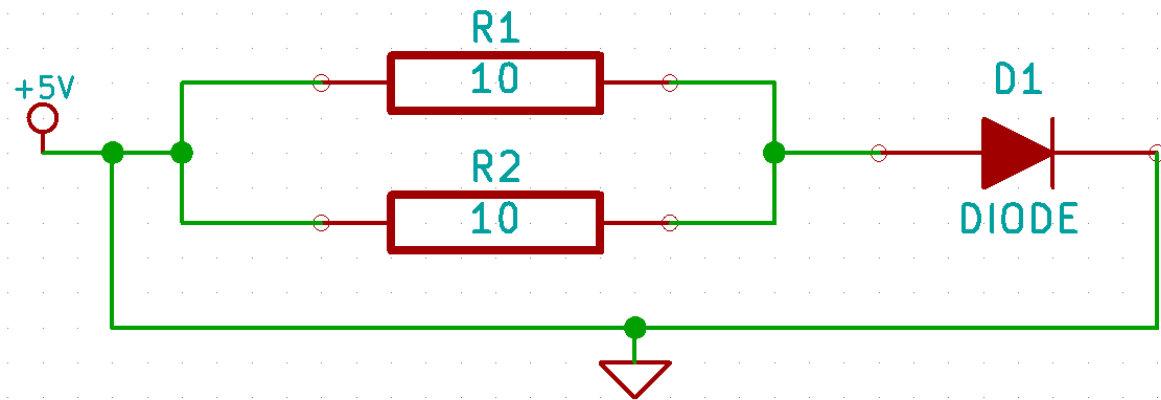
L'objectif était de mettre un seul émetteur mais plusieurs récepteurs sur le drone afin qu'il puisse être touché de tous les côtés.

Lors de nos premiers tests, nous n'avions qu'une portée de 1 mètre (ce qui est beaucoup trop faible vu la taille du quadricoptère) car nous ne prenions pas la bonne tension d'entrée ni les bonnes résistances. En effet nous n'avions pas pris en compte que le microcontrôleur du drone ne peut fournir qu'un voltage de 5V en signal carré.

Nous avons donc utilisé un générateur basse fréquence afin d'alimenter l'émetteur. Nous lui avons envoyé 5 volts en signal carré avec une fréquence de 38 kHz car le récepteur ne peut recevoir que cette fréquence d'après sa documentation technique.

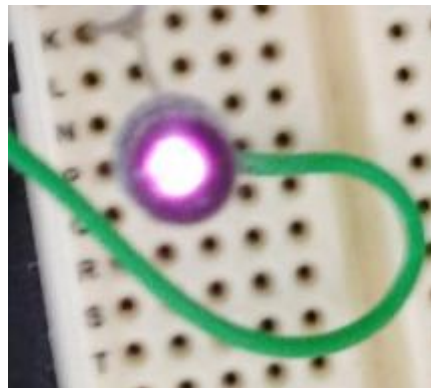
Pour choisir les résistances, nous avons fait plusieurs tests. Après avoir alimenté avec 5 volts, nous avons regardé la tension aux bornes de la résistance pour savoir ce qu'il y aurait dans la diode. Nous avons fait ces tests jusqu'à arriver à une résistance de 10 ohms. Le courant dans l'émetteur était bon mais nous n'avions qu'une portée de 2,40 mètres. Nous avons donc décidé de mettre en parallèle deux résistances de 10 ohms afin de faire une résistance de 5 ohms car  $R_{eq} = \frac{R1 \times R2}{R1 + R2} = \frac{10 \times 10}{10 + 10} = 5\Omega$ . Nous avons alors obtenu quasiment le courant limite que l'émetteur pouvait supporter. Cela nous a donné une portée de 2,89 mètres ce qui est juste.

Voici le schéma de câblage de l'émetteur :



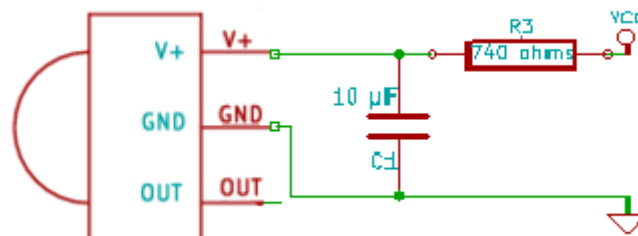
On peut vérifier que notre émetteur émet bien en utilisant l'appareil photo de notre téléphone portable. En effet celui-ci peut voir l'infrarouge contrairement à notre œil.

Voici ce que cela donne quand notre émetteur est en train d'émettre :

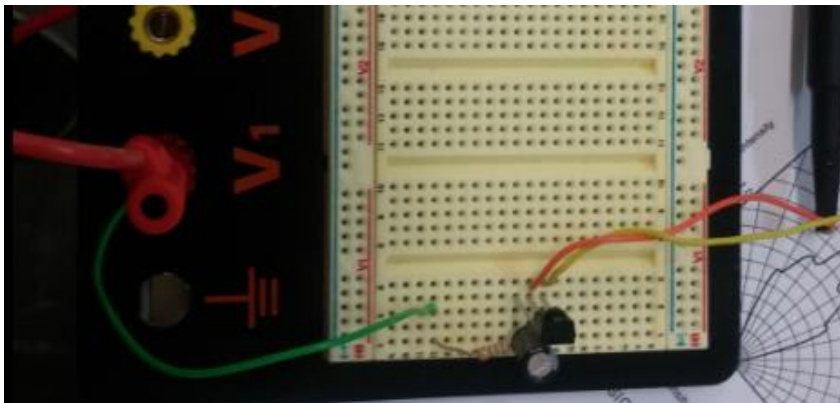


Concernant notre récepteur, nous avons suivi la documentation constructeur pour réaliser le câblage. Nous avons pris une résistance de 740 ohms et un condensateur de 10  $\mu\text{F}$ . Le tout est alimenté par une alimentation de 5 volts pour simuler le microcontrôleur.

Voici le schéma de câblage du récepteur :

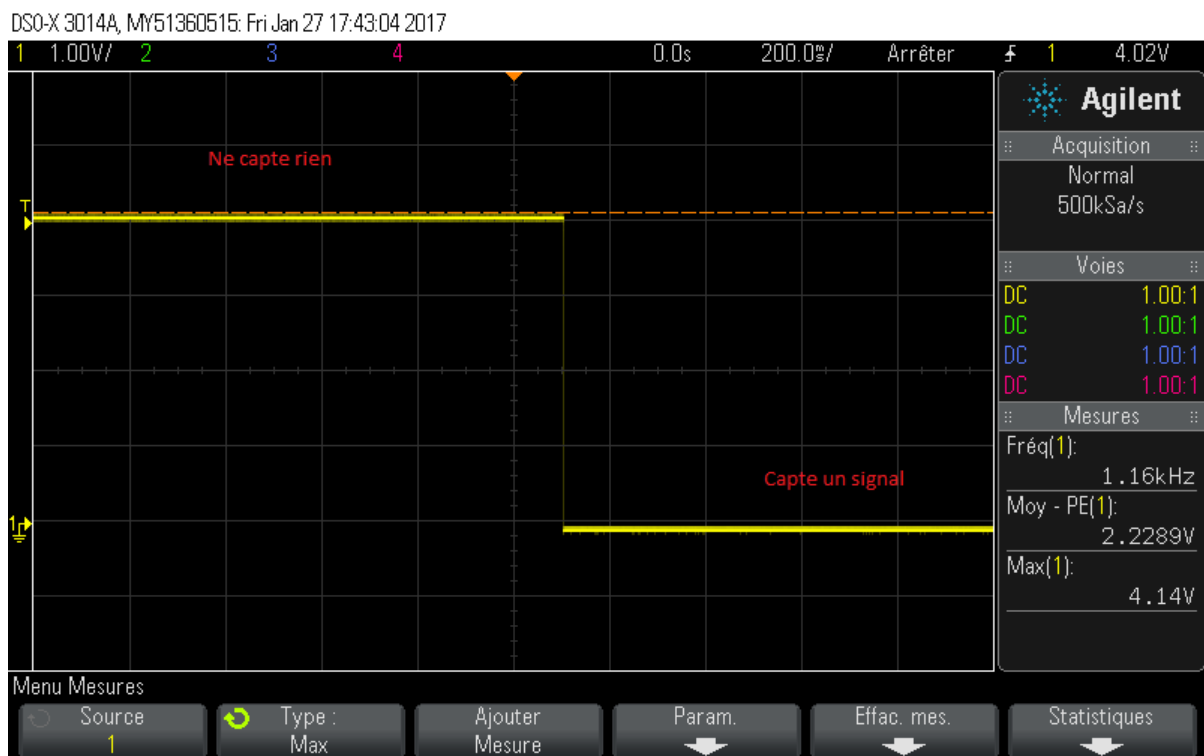


Voici une photo du câblage du récepteur :



Quand le récepteur ne capte rien, on observe à l'oscilloscope une tension qui n'est pas nulle, mais quand il capte le signal infrarouge le signal en sortie du récepteur est nul à l'oscilloscope.

Voici montrant le changement d'état quand le récepteur reçoit un signal :



Par la suite, nous avons essayé de déterminer jusqu'à quel angle d'inclinaison on pouvait émettre ou recevoir. Nous avons donc trouvé que l'on pouvait incliner de 30° d'un côté le récepteur, ou l'émetteur. Au-dessus de ces 30°, plus rien n'est capté par le récepteur.

Il faudrait donc six récepteurs pour couvrir la totalité d'un drone car  $\frac{360}{6} = 60$ . De plus il faudrait acheter des émetteurs avec une plus grande portée afin d'étendre la portée jusqu'au moins 5 mètres, sans que celle-ci ne soit non plus trop grande.



## Besoins financiers et matériels

Au cours de notre projet, nous avons apporté différentes modifications ou améliorations sur le quadricoptère. Nous avons donc eu besoin de nous procurer les composants qui nous ont permis de réaliser ces améliorations. Ces commandes devaient se faire prioritairement sur Farnell.

Voici un tableau récapitulant les différents composant que nous avons acheté chez Farnell :

<b>N°</b>	<b>Désignation</b>	<b>Code commande Farnell</b>	<b>Quantité</b>	<b>Prix unité HT</b>	<b>Montant HT</b>
1	Connecteur 4 contacts/1 rangée	1022231	1	0,32 €	0,32 €
2	Emetteur infrarouge	1652530	2	0,57 €	1,14 €
3	Récepteur infrarouge	4913139	2	1,49 €	2,98 €

<b>Total HT</b>	<b>4,43 €</b>
<b>Taux TVA</b>	<b>19,60%</b>
<b>Total TTC</b>	<b>5,30 €</b>

Le connecteur est utilisé pour réaliser la carte d'interface ainsi que les émetteurs et récepteurs infrarouges seront utilisés pour le dispositif de tir entre les drones.

Les composants nécessaires pour utiliser la caméra avec une nacelle ont été trouvés sur le site "[www.hobbyking.com](http://www.hobbyking.com)". Nous avons dû commander séparément par rapport aux composants de chez Farnell.

Voici un tableau récapitulant les différents composants que nous avons achetés chez Hobbyking :

N°	Désignation	Référence	Qté	Prix unité TTC	Montant TTC
1	ActionCam Inline Gimbal pour FPV et Multi-Rotor	450000070-0	1	8,63 €	8,63
2	HobbyKing <sup>TM</sup> HK15298 High Voltage Coreless numérique Servo MG / BB 15 kg / 0.11sec / 66g	HK15298	1	19,85 €	19,85
3	DS65HB Numérique Haute Vitesse Servo 6.5g / 1,5 kg / .07sec	DS65HB	1	5,83	5,83

<b>Total TTC</b>	<b>34,31</b>
------------------	--------------

$$\text{Prix Total TTC} = 5,30\text{€} + 34,31\text{€} = 39,61\text{€}$$

Le prix total toutes taxes comprises de nos commandes au milieu du projet s'élève à 39,61€.

Par ailleurs, nous n'avons pas eu besoin de commander le MAX232, ni le connecteur neufs broches, ni les condensateurs nécessaires pour réaliser la carte d'interface car il y en avait à l'IUT. Nous n'avons également pas pu commander le capteur barométrique car il était en rupture de stock. Concernant la caméra, nous n'avons également pas eu besoin d'en commander une car notre coach Mr Leboulanger en possédait déjà une. Pour la télécommande, il y a eu un problème lors de la commande et nous ne l'avons jamais reçu, nous avons donc aussi utilisé celle de notre coach. Enfin, nous n'avons pas eu besoin de commander la centrale inertielle car l'IUT l'avait déjà achetée pour les licences qui ont travaillé sur ce projet auparavant.

Après la première estimation des coûts, nous avons cassé plusieurs hélices et donc nous en avons commandé quatre nouvelles, ce qui a légèrement augmenté le budget de notre projet.

## Conclusion

Le projet quadricoptère est un projet intéressant car il permet de toucher à plusieurs domaines étudiés à l'IUT comme l'électronique, les asservissements ou l'informatique industrielle. Cependant le fait que celui-ci soit commencé depuis plusieurs années le rend plus difficile car il faut s'approprier ce que les autres ont fait, et surtout les programmes informatiques, ce qui n'est pas toujours évident. Ce projet permet aussi d'apprendre la gestion de projet à l'aide de logiciel comme Mindview.

Nous pensons que l'objectif principal est partiellement atteint mais s'approche à grand pas de la réussite. Le premier asservissement fonctionnait mais le drone était instable à cause d'un double intégrateur qui nous mettait une marge de phase en dessous de -180 degrés. Par conséquent, il a fallu modifier l'asservissement simple par un double asservissement qui ne fonctionne pas. Celui-ci sera sûrement un succès avec plus de tests.

De façon générale le projet peut toucher une grande partie des domaines étudiés en GEII, l'électronique, l'asservissement, l'informatique industrielle. De plus le projet est formateur notamment sur la gestion de projet.

Nous sommes satisfaits de notre travail, cependant, nous sommes un peu déçus car le drone ne volait pas au départ et il ne vole pas non plus à la fin du projet mais nous sommes confiants pour les prochains élèves qui s'occuperont de ce projet.

Nous tenions à remercier les personnes qui ont participé au projet tels que les techniciens de l'IUT, et plus particulièrement notre professeur tuteuré M.Leboulanger pour ses aides et ses explications durant le projet.

## Bibliographie

### Documentation

#### Microcontrôleur

Microcontrôleur ATmega1280

#### Capteurs

Capteur à ultrason MaxSonar-EZ4

Accéléromètre numérique ADXL345

Gyroscope numérique à triple axe ITG-3200

Capteur de pression numérique BMP085

#### Récepteur

Récepteur FrSky D4R-II

#### Emetteur et Récepteur IR

Emetteur TSAL6400

Récepteur TSOP34838

#### Caméra

Servo DS65HB

Servo HK15298

## Annexe

### Programme de la centrale :

```
// Récupération des données utiles (acc/gyr/angl)

#include <avr/io.h>
#include <avr/interrupt.h>
#include "vars.h"

/*
"volatile" signifie que l'on force le microcontrôleur à accepter "etat"
comme une variable, même si elle n'est pas appelée dans le programme.
*/

volatile char val_attendues[7] = {0xFA, 0xFF, 0x36, 0x1E, 0, 0, 0x0C};
//Ces valeurs précède les données et sont les octets qui précisent les types de
données.

int init_usart_centrale(void)
{
    UBRR0H = 0;
    UBRR0L = 8; // 115200 bauds
    /*UBRR (USART MSPIM Baud Rate Registers)

    RXIE permet de gérer les interruptions.
    Tant que le microcontrôleur ne reçoit pas de données, le programme n'est pas
    bloqué et peut donc executer d'autre consigne.
    Lorsqu'il reçoit une donnée, le registre RXCIE permet de la détecter et
    d'exécuter une variable.
    */
    UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
    UCSR0B = (1<<RXEN0)|(1<<RXCIE0);
    /* Mode Asynchrone (RS232), Données de 8 bits, Bit de parité désactivé pour
    ne pas rajouté de bit à la trame qui est déjà codée sur 8 bits
    */

    return(0);
}

ISR(USART0_RX_vect){ //interruption

    static int etat=0; //on initialise état à 0
    static int index_tab_result = 0; //place dans le tableau
    static int type_result; //permet de stocker les valeurs de données
    (accélération 0x4031, l'Angle 0x2031...)
    static int*tab_result; //permet de faire pointer "tab_result" vers le tableau
    "vitesse_angulaire[3]" ou "angls[3]" dans vars.h
    static int receiving; //sert à stocker les données qu'on reçoit par UDR0
    char c;
    c = UDR0; //notre variable c va recevoir en entrée les donnée de UDR0

    /*Les cas 0 à 6 sont les octets qui ne varient pas soit Preamble, Bid, Mid, Len,
    Checksum (cf cr 2015)*/
    switch(etat){
        case 0:
        case 1:
        case 2:
        case 3:
        case 6:
            if(c == val_attendues[etat]){ //on compare ce qu'on reçoit dans
            notre variable c à nos valeurs fixe, précisé dans "val_attendues[]"
```

```

        etat++; // on incrémente l'état pour
    }
    else{
        etat = 0;
    }
    break;

    /*
    */

    case 4: //data ID (data reçues)
    case 19: //data ID (data reçues)
        etat++;
        receiving=((int)c)<<8; // on décale ce qu'on reçoit de 8 bits
vers la gauche (ex : on reçoit 0x20 on le place 8 bits à gauche)
        break;
    case 5: //data ID (data reçues)
    case 20: //data ID (data reçues)
        etat++;
        type_result = receiving | ((int)c & 0xff); // on masque la donnée
reçue (ex : on reçoit 0x31 on fait un masque (&) avec 0xFF pour récupérer sa valeur
et un masque (|=ou) avec le 0x20 reçu précédemment pour faire 0x2031 et le stocker
dans type_result)
        switch (type_result) { //on test pour savoir dans lequel des
deux cas on est avec la valeur dans type_result
            case 0x2031: //cas ou on reçoit l'angle
                tab_result = angles; // on fait pointer
tab_result vers le tableau "angles"
                break;
            case 0x8021: //cas ou on reçoit la vitesse angulaire
                tab_result = vitesse_angulaire; // on fait
pointer tab_result vers le tableau "vitesse_angulaire"
                break;
        }
        break;

    case 7:
    case 11:
    case 15:
    case 22:
    case 26:
    case 30:
        etat++;
        receiving=((int)c)<<8; //partie entière qu'on décale de 8 bits
vers la droite pour laisser les 8 bits d'avant à la partie décimale
        break;
    case 8:
    case 12:
    case 16:
    case 23:
    case 27:
    case 31:
        etat++;
        tab_result[index_tab_result] = receiving|((int)c & 0xff);
//partie décimale et partie entière sockée dans "tab_result[]".
        index_tab_result++; //on incrémente "index_tab_result" pour
stocker l'angle en x, y, z, l'accélération x, y z...
        break;
    case 9:
    case 10:
    case 13:
    case 14:

```

```

        case 17:
        case 18:
        case 24:
        case 25:
        case 28:
        case 29:
        case 32:
            etat++; //on incrémente état lors de ces case car nous
n'utilisons pas les 2 derniers octets pour l'angles X, Y et Z (cf schématisation
d'une trame)
            break ;

        case 21:
            if (c == val_attendues[6] ) { // si on est à 0x0C alors on
incréméte état
                etat++;
                index_tab_result = 0; //et on remet index_tab_result à 0
car on va stocker les valeurs de l'accélération au lieu de l'angle donc on nettoir
le tableau avec les valeurs de l'angle
            }
            else {
                etat = 0; //sinon etat est remis à 0 et on reprend au
début
            }
            break;
        case 33:
            etat = 0; // etat vaut 0 car on est à la fin de notre trame
            index_tab_result = 0; //et on remet index_tab_result à 0
            break;

    } //

} //

void centrale_read_gyro(struct vars_t *v) { //on récupère la vitesse
angulaire
    // X axis
    v->GyroX = (vitesse_angulaire[0]);
    // Y axis
    v->GyroY = (vitesse_angulaire[1]);
    // Z axis
    v->GyroZ = (vitesse_angulaire[2]);
}

void centrale_read_angls(struct vars_t *v) { //on récupère l'Angle
    // X axis
    v->AngleY = -(angls[0])*2; //2 pour 30° //tangage//on multiplie par 2
// pour décaler de 1 bit vers la gauche.
Cela permet de rapprocher notre angle max de 1000 (valeur angle max telecommande)
// pour comparer la consigne de la
telecommande avec l'angle calculer par la centrale inertielle.
    // Y axis
    v->AngleX = -(angls[1])*2; //roulis
    // Z axis
    v->AngleZ = (angls[2])*2; //lacet
}

```

**Programme asservissement :**

```

/*
 * asservissement.c
 * Created: 27/11/2013
 */
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/sleep.h>

#include <util/delay.h>
#include <util/twi.h>

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>

#include "interruptions_init.h"
#include "timers_init.h"
#include "main.h"
#include "annexe.h"
#include "vars.h"

#include "asservissement.h"
#include "vars.h"
#include "pid.h"

//Voie 0, premier signal : Monter/descendre
//Voie 1, inclinaison latéral gauche/droite (roll)
//Voie 2, inclinaison avancer reculer
//Voie 3, yaw

void output_motors(struct vars_t *v){
    //Saturation en 0 % :
    if (v->MotorOut1<0) {v->MotorOut1=0;}
    if (v->MotorOut2<0) {v->MotorOut2=0;}
    if (v->MotorOut3<0) {v->MotorOut3=0;}
    if (v->MotorOut4<0) {v->MotorOut4=0;}

    //Saturation en 100 % :
    if (v->MotorOut1>1000) {v->MotorOut1=1000;}
    if (v->MotorOut2>1000) {v->MotorOut2=1000;}
    if (v->MotorOut3>1000) {v->MotorOut3=1000;}
    if (v->MotorOut4>1000) {v->MotorOut4=1000;}

    //Si les manches indiquent les moteurs à zeros, alors :
    if (v->FlagCollectiveZero==1){
        v->MotorOut1=0;
        v->MotorOut2=0;
        v->MotorOut3=0;
        v->MotorOut4=0;
    }

    //Scale :
    v->MotorOut1+=1000;
    v->MotorOut2+=1000;
    v->MotorOut3+=1000;
    v->MotorOut4+=1000;
}

```



```

        //printf("MotorOut1 = %d, MotorOut2 = %d, MotorOut3 = %d, MotorOut4 = %d\n",v->MotorOut1,v->MotorOut2,v->MotorOut3,v->MotorOut4);
        //Sortie finale :
        OCR4A = v->MotorOut1; //M1
        OCR4B = v->MotorOut2; //M2
        OCR4C = v->MotorOut3; //M3
        OCR5A = v->MotorOut4; //M4
    }

void init_motors_pids() {
    pid_Init(K_ROLL_P * SCALING_FACTOR, K_ROLL_I * SCALING_FACTOR , K_ROLL_D * SCALING_FACTOR , &pidData_roll);
    pid_Init(K_PITCH_P * SCALING_FACTOR, K_PITCH_I * SCALING_FACTOR , K_PITCH_D * SCALING_FACTOR , &pidData_pitch);
    pid_Init(K_YAW_P * SCALING_FACTOR, K_YAW_I * SCALING_FACTOR , K_YAW_D * SCALING_FACTOR , &pidData_yaw);

    pid_Init(K_ROLL_P2 * SCALING_FACTOR, K_ROLL_I2 * SCALING_FACTOR , K_ROLL_D2 * SCALING_FACTOR , &pidData_roll2);
    pid_Init(K_PITCH_P2 * SCALING_FACTOR, K_PITCH_I2 * SCALING_FACTOR , K_PITCH_D2 * SCALING_FACTOR , &pidData_pitch2);
    pid_Init(K_YAW_P2 * SCALING_FACTOR, K_YAW_I2 * SCALING_FACTOR , K_YAW_D2 * SCALING_FACTOR , &pidData_yaw2);
}

void reset_motors_pids() {
    pid_Reset_Integrator(&pidData_roll);
    pid_Reset_Integrator(&pidData_pitch);
    pid_Reset_Integrator(&pidData_yaw);

    pid_Reset_Integrator(&pidData_roll2);
    pid_Reset_Integrator(&pidData_pitch2);
    pid_Reset_Integrator(&pidData_yaw2);
}

void computedata_motors(struct vars_t *v){
    // Fonction pour calculer les valeurs à mettre sur les moteurs en fonction de la consigne
    int y;
    int z;
    int col;

    //saturation si le collectif est trop élevé
    col = v->computedCollective;

    if (col > MAX_MOTORS) {
        col = MAX_MOTORS;
    }
    // start mixing
    v->MotorOut1 = col;
    v->MotorOut2 = col;
    v->MotorOut3 = col;
    v->MotorOut4 = col;

    // Calculate roll command output
    //
    y = pid_Controller(v->RxInRoll_Angle, v->AngleX, &pidData_roll);
    y = pid_Controller(y, v->GyroX, &pidData_roll2);
    y = pid_Controller(v->RxInRoll_Angle, v->GyroX, &pidData_roll);

    //z = pid_Controller(v->y*(RxInRoll_Angle-, v->GyroX, &pidData_roll);

```

```

        // Add roll command output to motor 2 and 3
        v->MotorOut1 += y;
        v->MotorOut2 -= y;
        v->MotorOut3 -= y;
        v->MotorOut4 += y;

        // Calculate pitch command output

//      y = pid_Controller(v->RxInPitch_Angle, v->AngleY, &pidData_pitch);
//      y = pid_Controller(y, v->GyroY, &pidData_pitch2);
        y = pid_Controller(v->RxInPitch_Angle, v->GyroY, &pidData_pitch);

        // Add Pitch command output to motor 1 and 4
        v->MotorOut1 += y;
        v->MotorOut2 += y;
        v->MotorOut3 -= y;
        v->MotorOut4 -= y;

        // Calculate Yaw command output

        y = pid_Controller(-v->RxInYaw_Angle, v->AngleZ, &pidData_yaw);
        y = pid_Controller(y, v->GyroZ, &pidData_yaw2);
        y=0;

        // limit Yaw command to -YawLimit and YawLimit
        //if (y < -(v->YawLimit)){y = -(v->YawLimit);}
        //if (y > v->YawLimit){y = v->YawLimit;}

        // Add Yaw command output to motor 1, 2, 3 and 4
        v->MotorOut1 += y;
        v->MotorOut2 -= y;
        v->MotorOut3 += y;
        v->MotorOut4 -= y;

        if (v->MotorOut1 < LOWVALUE)
            v->MotorOut1 = LOWVALUE;
        if (v->MotorOut2 < LOWVALUE)
            v->MotorOut2 = LOWVALUE;
        if (v->MotorOut3 < LOWVALUE)
            v->MotorOut3 = LOWVALUE;
        if (v->MotorOut4 < LOWVALUE)
            v->MotorOut4 = LOWVALUE;
    }

```