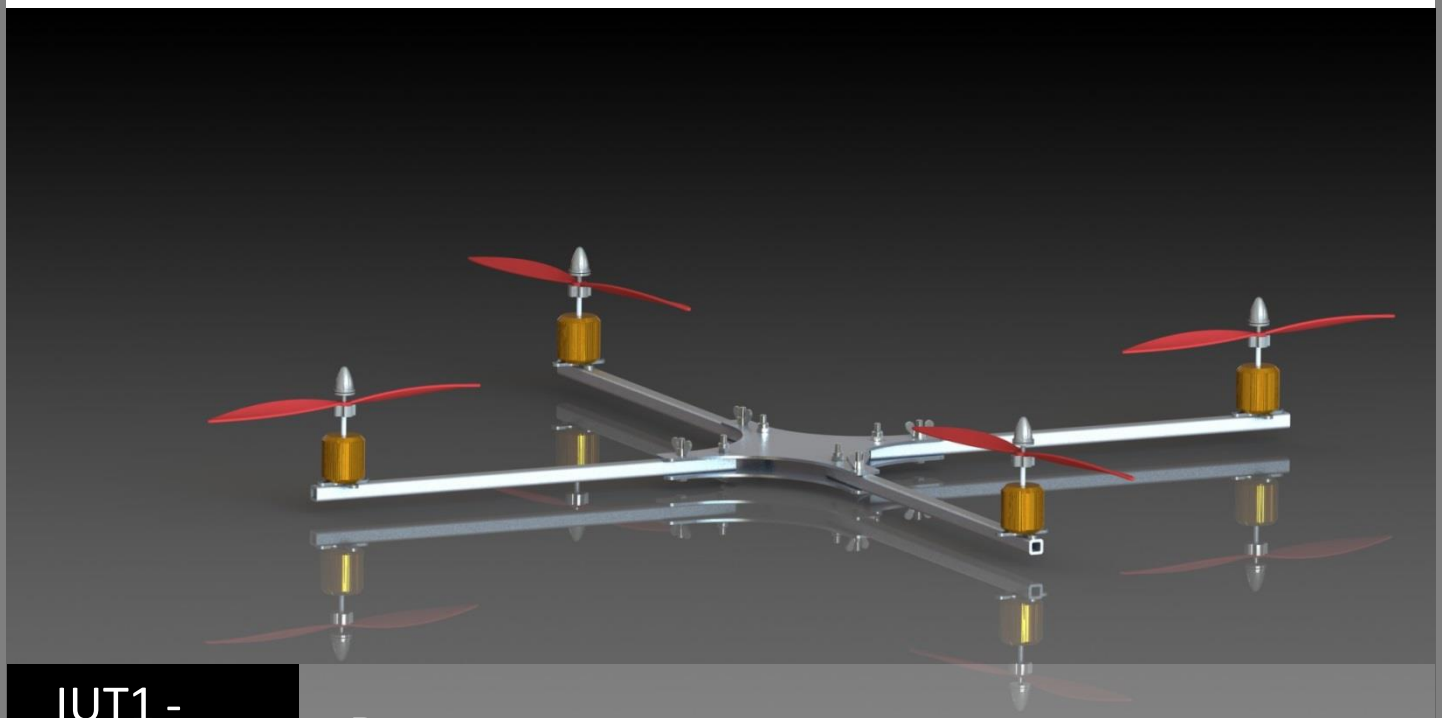
2013 - 2014



IUT1 - GRENOBLE

PROJETS TUTEURES

Réalisation d'un drone | Harold Ducept

Contenu

Introduction

Description générale

De nos jours, la présence de véhicules volant sans pilotes, (plus connus sous le nom de drones) se fait de plus en plus grande. Une des raisons à ce phénomène est le fait que certains fabricants tendent à rendre facile d'accès ce matériel qui, il y a quelques années encore coutait extrêmement cher. De plus, avec les technologies actuelles, le pilotage est rendu des plus aisés. Ainsi, du matériel qui était destiné à des professionnels aguerris est mis entre les mains de jeunes, voir d'enfants. Une autre des raisons qui peut pousser les gens à vouloir se procurer un drone est le fait de voir fleurir sur le web un grand nombre de vidéos prises depuis ces systèmes. En effet, celles-ci offrent des points de vue incomparable et offrent également une impression de liberté, et donnent le sentiment de pouvoir voler.

Cependant, tous les possesseurs de tels engins n'en font pas la même utilisation. Cela va du domaine du loisir avec la simple envie de piloter jusqu'au domaine militaire. Il faut donc que ces appareils soient adaptés à l'utilisateur final, ce qui explique la très grande variété de produits proposés et les différences de prix associées.

Environnement

La première des contraintes liées à l'environnement est la gravité. En effet, il me faudra une configuration capable de soulever le poids de l'engin et de l'élever à la hauteur désirée.

Le drone que je souhaite réaliser doit être capable de voler en extérieur. Il n'est pas particulièrement destiné aux vols d'intérieurs qui seraient certainement trop dangereux que ce soit pour les objets avoisinants que pour les personnes présentes ou bien pour le drone lui-même.

Pour pouvoir voler en extérieur, le drone devra être capable de réagir à son environnement. En effet, il faudra qu'il puisse se repérer, en particulier connaître son altitude et savoir s'il s'approche d'un obstacle. Tous ces paramètres requièrent le fait de prendre en compte que le système évoluera dans un milieu ouvert et non fermé, dans lequel nous aurions pu nous appuyer sur les murs et le sol pour nous repérer. Il faudra également prendre en compte un élément naturel : le vent. En effet, bien qu'il ne sera pas conçu pour voler par mauvais temps, une simple petite brise pourrait suffire à déstabiliser le système.

Un autre point important à prendre en compte concerne les obstacles qu'il pourrait y avoir concernant la transmission entre la partie commande et le drone lui-même. En effet, les transmissions pourront être perturbées par toute sorte de parasites. Ainsi, une communication ne sera pas aussi bonne en ville (présence de beaucoup de perturbations et de bâtiments) qu'en campagne au milieu d'un champ.

Contraintes

Les contraintes sur ce projet, outre celles liées à l'environnement, sont de 2 ordres : Celles liés au système lui-même, c'est-à-dire les limites que je souhaite pouvoir atteindre avec ; et celles liées au déroulement du projet.

Voici la liste des limites que je souhaite pouvoir atteindre :

- Autonomie : 15min minimum. En effet, un drone est extrêmement couteux en ressources. Les moteurs nécessaires à la propulsion demandent une puissance relativement grande alors que la capacité des batteries reste relativement limitée. Il est bien entendu possible d'augmenter cette capacité, mais pour cela, il faut une batterie plus importante, donc plus lourde, donc des moteurs plus puissants qui consomment d'avantages. On arrive donc rapidement dans des prix qui peuvent monter très haut.
- Portée du signal : 1 km grand minimum (1.5km serait correct). En effet, il est important que le pilote garde toujours un lien avec l'appareil. Il faut donc prévoir un système de transmission qui sera suffisamment puissant. Il faut donc également prévoir un système pour que le pilote puisse continuer à observer comment évolue le drone même lorsque celui-ci sera trop loin pour être à portée de vue.

Ces 2 contraintes, en plus d'être liées, en apportent d'autres. En effet, l'autonomie sera fonction du poids de l'appareil. C'est pourquoi j'ai choisis de ne pas dépasser les 2kg. La structure du drone pourra quant à elle jouer sur la qualité d'émission et de réception du signal ainsi que sur les vibrations du système. Celles-ci pourraient erroner les données provenant des capteurs.

Le 2ème type de contraintes est lié au déroulement du projet. En effet, étant donné l'importance du projet, il est primordial d'avoir une idée assez précise de comment atteindre l'objectif. Pour cela, 2 points importants sont à étudier :

- Le premier est au niveau financier. Pour un projet avec les spécifications données, il faut compter environ 400€. Ce qui représente une importante somme d'argent.
- Le second est au niveau du temps à y consacrer. Les 100 heures prévues pour les projets tuteurés ne seront en aucun cas suffisantes pour le terminer. Il faut donc être prêt à passer une partie de son temps personnel dessus.

Présentation des différents sous-ensembles et des différentes possibilités envisagées:

Choix de la forme du drone :

Il existe un certain nombre de type de drone. Chacun ayant ses particularités. Voici un bref aperçut des plus courant d'entre eux :

Drone de type avion :

Ces drones sont tout particulièrement faits pour faire de longues distances et des pointes de vitesse. Ils peuvent également servir à faire des figures. Ils peuvent être motorisé ou non. Si celui-ci ne dispose pas de motorisation, il s'agit alors d'un planeur.

Drone de type aile volante

Ce type de drone se rapproche beaucoup de l'avion sur le fait qu'il utilise également une surface de portance, sauf qu'à la différence de l'avion qui dispose d'une aile de chaque côté, l'aile volante, comme son nom l'indique n'en a qu'une seule qui représente l'ensemble de sa surface. Ils sont faits essentiellement pour la vitesse.

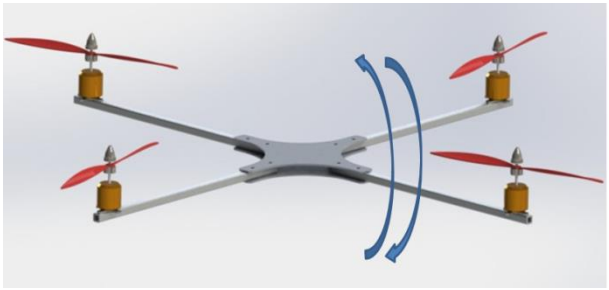
Drone de type hélicoptère/multicoptère :

Ceux-ci utilisent le même principe que l'hélicoptère. C'est-à-dire une voir plusieurs hélices qui tournent suffisamment vite pour permettre à l'appareil de s'élever. La particularité de ces appareils est qu'ils permettent de faire du surplace. Pour mon projet, j'ai choisis de réaliser un multicoptère, plus précisément un quadricoptère. J'ai choisi ce type d'appareils car la gestion des mouvements est plus simple et plus intuitive qu'un hélicoptère (cf. description des C.U.s). De plus, l'armature de celui-ci sera en aluminium tubulaire carré. J'ai choisis ce matériel pour sa légèreté et, le fait d'utiliser des tubes carrés augmente sa rigidité et me permettra de faire passer les câbles à l'intérieur de ceux-ci.

Gestion du tangage

Présentation :

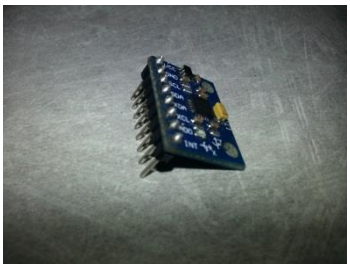
Le tangage est le mouvement d'inclinaison vers l'avant ou l'arrière de l'appareil. Sur un drone de type avion, le tangage permettra à celui-ci de se diriger vers le sol si il tangue vers l'avant ou de monter dans le cas contraire. Sur un drone de type quadricoptère, le tangage permet à celui-ci de se diriger vers l'avant ou l'arrière.



Solution envisagée :

Utilisation du gyroscope :

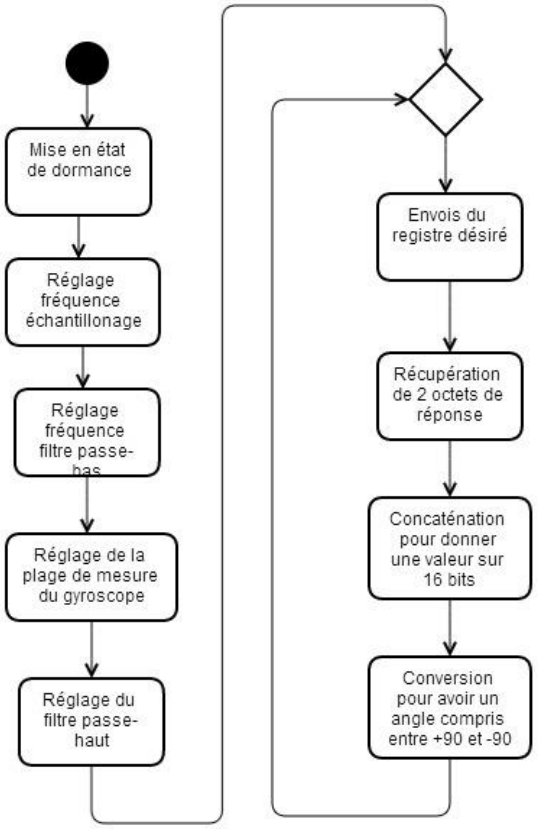
La solution qui m'est venu de suite est l'utilisation d'un gyroscope. Grâce à ce module, je peux connaître l'angle d'inclinaison du système aussi bien pour le roulis que pour le tangage.



Le choix du gyroscope s'est porté sur un MPU-6050 pour plusieurs raisons. La première étant son prix. En effet, celui-ci ne m'a coûté que 2.98€. Ce qui est très peu lorsque nous regardons les prix que proposent certains fabricants. La 2ème raison de ce choix fut sa connectivité. Etant donné que je souhaite utiliser le moins de ports possible sur le microcontrôleur pour en faciliter la configuration et réaliser une petite économie d'énergie, j'ai choisis de brancher tous mes périphériques sur un même bus I²C. Fonction que proposait ce capteur. De plus, ce capteur permet également de connaître l'accélération sur les 3 axes. Ce qui sera une information fort utile lorsque je devrais calculer la poussée que devront procurer les moteurs.

Fonctionnement détaillé de la solution :

N'ayant pas pu trouver d'informations intéressantes dans la documentation (pourtant très fournie) du MPU-6050, je me suis basé sur un exemple de code fourni sur internet. Les commentaires de celui-ci étant en chinois, je n'ai pas pu vraiment comprendre tout le sens de chaque commande. J'ai donc préféré utiliser les valeurs fournies dans ce code qui semblent fonctionner mais qui ne sont peut-être pas les meilleurs pour mon système. Voici ce que j'ai pu comprendre →



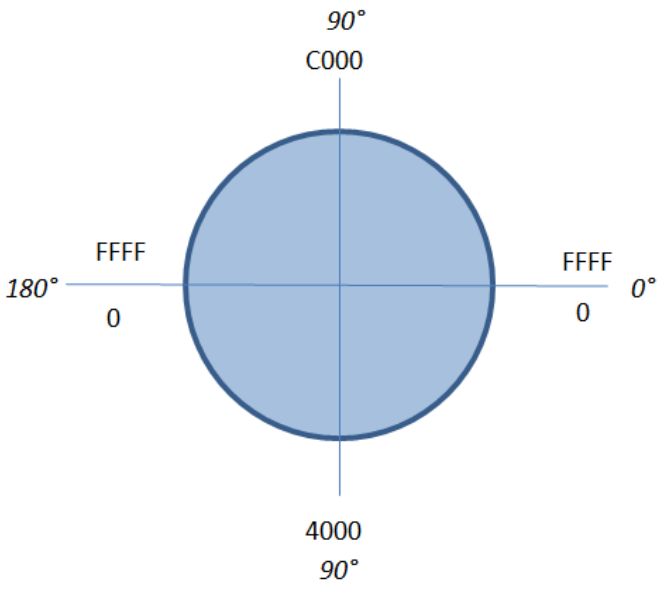
Pour connaître l'angle ou l'accélération sur un axe, il suffit de donner le registre correspondant au début de la boucle. Le contenu de la réponse sera alors la réponse désirée. Il y a donc 6 registres à interroger si nous souhaitons avoir l'angle et l'accélération sur les 3 axes.

L'inconvénient de ce système est qu'il ne détectera pas si l'appareil se retourne. Donc au lieu de se remettre en bonne position, celui-ci accélèrera jusqu'à toucher le sol. Je pense qu'il est possible de contrer ce problème mais je n'ai pas encore réalisé le morceau de code qui s'en chargera.

Conversion de l'angle brute en angle en degré :

Les valeurs envoyées par le gyroscope ne sont pas données directement en degré. Pour obtenir celles-ci comme tel, une conversion doit être faite.

En pratique, le gyroscope donne des valeurs codées sur 2 octets comprises entre 0x0 et 0xFFFF. Après plusieurs tests, voici ce que j'ai pu comprendre de comment interpréter ces données :



Pour convertir ces données en angle en degrés, voici l'algorithme que j'ai mis au point :

Si l'angle est compris entre 49151 (0xBFFF) et 65535 (0xFFFF) :

$$angle\ en\ degré = \frac{angle\ brute - 0xBFFF}{0xB6} - 0x5A$$

Sinon si, l'angle est compris entre 0 (0x0000) et 16384 (0x4000) :

$$angle\ en\ degré = \frac{angle\ brute}{0xB6}$$

Sinon :

$$angle\ en\ degré = 0$$

Sachant que 0x5A = 90 et que 0xB6 est le facteur diviseur à utiliser pour convertir une valeur comprise entre 0 et 0xFFFF à une valeur comprise entre 0 et 360 :

$$\frac{0xFFFF}{360} = 0xB6$$

Détail du code source associé :

```
static msg_t ThreadRoulisTangage(void *arg)
{
	//tableau stockage données reçues sur l'I²C
	uint8_t rxbuf[10] = {0};
	uint8_t txbuf[10] = {0};

	int tangage;
	int roulis;

	//Initialisation du gyroscope
	txbuf[0] = 0x6B;  //Registre
	txbuf[1] = 0x00;  //Donnée
	i2cAcquireBus(&I2CD1); //Acquisition bus I²C (éviter des collisions)
	//Mise en état de dormance
	i2cMasterTransmitTimeout(&I2CD1, 0x68, txbuf, 2, NULL, 0, 1000);
	i2cReleaseBus(&I2CD1);

	txbuf[0] = 0x19;  //Registre
	txbuf[1] = 0x07;  //Donnée
	i2cAcquireBus(&I2CD1);
	//Réglage de la fréquence d'échantillonage (125Hz)
	i2cMasterTransmitTimeout(&I2CD1, 0x68, txbuf, 2, NULL, 0, 1000);
	i2cReleaseBus(&I2CD1);
	txbuf[0] = 0x1A;  //Registre
	txbuf[1] = 0x06;  //Donnée
	i2cAcquireBus(&I2CD1);
	//Réglage de la fréquence du filtre passe-bas (5Hz)
	i2cMasterTransmitTimeout(&I2CD1, 0x68, txbuf, 2, NULL, 0, 1000);
	i2cReleaseBus(&I2CD1);

	txbuf[0] = 0x1B;  //Registre
	txbuf[1] = 0x18;  //Donnée
	i2cAcquireBus(&I2CD1);
	i2cMasterTransmitTimeout(&I2CD1, 0x68, txbuf, 2, NULL, 0, 1000);
	//Pas vraiment compris… Correspondrait à la plage de mesure de
	                                                    l'accéléromètre…
```

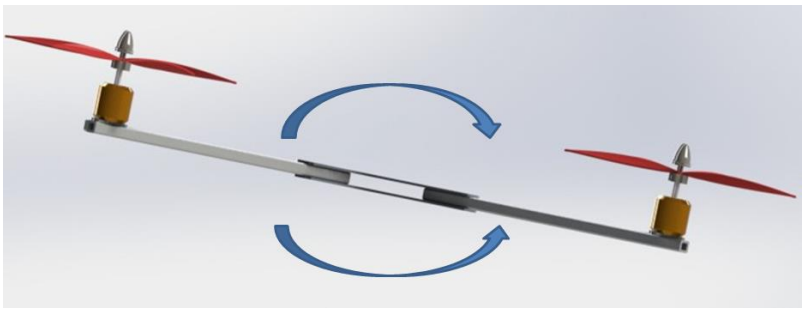
```
	i2cReleaseBus(&I2CD1);

	txbuf[0] = 0x1C;  //Registre
	txbuf[1] = 0x01;  //Donnée
	i2cAcquireBus(&I2CD1);
	//Pas vraiment compris… Gamme d'accéléromètre d'auto-test et mesure
	    la fréquence du filtre passe-haut, les valeurs typiques: 0x01 (non
	                                                        auto, 2G, 5 Hz)
	i2cMasterTransmitTimeout(&I2CD1, 0x68, txbuf, 2, NULL, 0, 1000);
	i2cReleaseBus(&I2CD1);

	while(TRUE)
	{
		//récupération du tangage
		txbuf[0] = 0x3B; //Registre de GYRO_XOUT_H
		i2cAcquireBus(&I2CD1);
		i2cMasterTransmitTimeout(&I2CD1, 0x68, txbuf, 1, rxbuf, 2,
	                                                              1000);
		i2cReleaseBus(&I2CD1);
		//Réunion de l'octet de poids faible avec l'octet de poids fort
		tangage = rxbuf[0]<<8;
		tangage = tangage | (unsigned int)rxbuf[1];
		//Conversion de l'angle
		if (tangage < 0xFFFF && tangage > 0xBFFF) //Si compris entre
	                                                    49151 et 65535…
		{
			tangage = (tangage - 0xBFFF)/0xB6-0x5A; //…On divise par
182 (rapport entre la valeur brute et l'angle en degré) et on soustrait 90
		}
		else if (tangage > 0x0000 && tangage < 0x4000) //Si compris
	                                                  entre 0 et 16384…
		{
			tangage = tangage/0xB6; //…On divise par 182
		}
		else // angle = 0
		{
			tangage = 0;
		}

		//récupération du roulis
		txbuf[0] = 0x3D; //Registre de GYRO_YOUT_Y
		i2cAcquireBus(&I2CD1);
		i2cMasterTransmitTimeout(&I2CD1, 0x68, txbuf, 1, rxbuf, 2,
	                                                              1000);
		i2cReleaseBus(&I2CD1);
		//Réunion de l'octet de poids faible et de l'octet de poids
	                                                              fort
		roulis = rxbuf[0]<<8;
		roulis = roulis | (unsigned int)rxbuf[1];
		//Conversion de l'angle
		if (roulis < 0xFFFF && roulis > 0xBFFF)
		{
			roulis = (roulis - 0xBFFF)/0xB6-0x5A;
		}
		else if (roulis > 0x0000 && roulis < 0x4000)
		{
			roulis = roulis/0xB6;
		}
		else // angle = 0
		{
			roulis = 0;
```

```
            }

            //Stockage dans des variables globales
            g_roulis = roulis;
            g_tangage = tangage;
            //Réinitialisation des valeurs
            tangage = 0;
            roulis = 0;
        }
}
```

Gestion du roulis

Présentation :

Le roulis est le mouvement d'inclinaison sur la droite ou la gauche du drone. C'est grâce au roulis que le drone pourra se diriger sur les côtés. Cependant, il faut faire attention à ne pas adopter un angle de roulis trop important au risque de voir l'appareil décrocher, ce qui résulterait en une perte de contrôle pour le pilote.

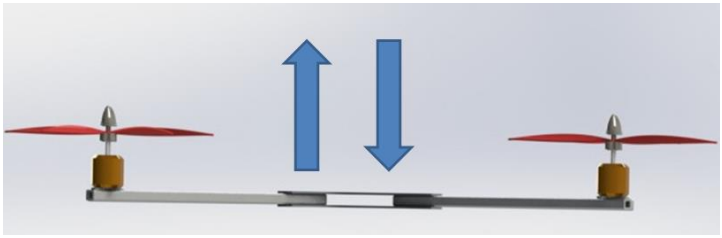


La solution utilisée pour répondre à ce problème est la même que celle pour le tangage. S'y référer pour le fonctionnement détaillé.

Gestion de l'altitude

Présentation :

L'altitude correspond la hauteur de vol de l'objet par rapport à un référentiel terrestre préétabli. Celui-ci peut être basé sur le niveau de la mer ou sur la distance réelle par rapport au sol ou bien encore la distance au sol à l'emplacement du décollage.

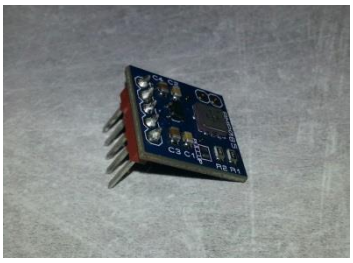


Solutions envisagées :

Utilisation de l'ultra son :

Une idée relativement intuitive pour répondre à ce problème, serait l'utilisation de capteurs à ultra-sons qui seraient disposés sous le drone, pointant en direction du sol. Cette solution à l'avantage d'être peu couteuse et m'aurait permis d'utiliser une technologie que je connaissais déjà. Cependant, l'utilisation des ultra-sons ne permet pas de monter à une altitude très élevée. En effet, la distance de détection de ces capteurs n'est que d'environ 2 mètres. Ce qui est très nettement inférieur à ce que je souhaite atteindre. J'ai donc finis par proscrire cette solution.

Utilisation d'un capteur de pression :



La deuxième solution que j'ai eue m'est venue en parcourant les sites dédiés au modélisme. En effet, lorsque les appareils sont destinés à monter relativement haut, il est alors conseillé de se baser sur la pression atmosphérique pour connaître l'altitude. Cette méthode est certes moins précise que les capteurs à ultra-sons (entre 15 et 40cm selon le capteur) mais permet d'atteindre des altitudes bien plus élevées. Dans le cas du capteur que j'ai choisis (le BMP085), l'altitude maximale est de 9km (ce qui est très nettement au-dessus de ce qui sera demandé au drone) et la précision est d'environ 25cm. Celui-ci utilise également le protocole I²C pour communiquer et j'ai pu trouver ce capteur pour 9.64€ sur internet.

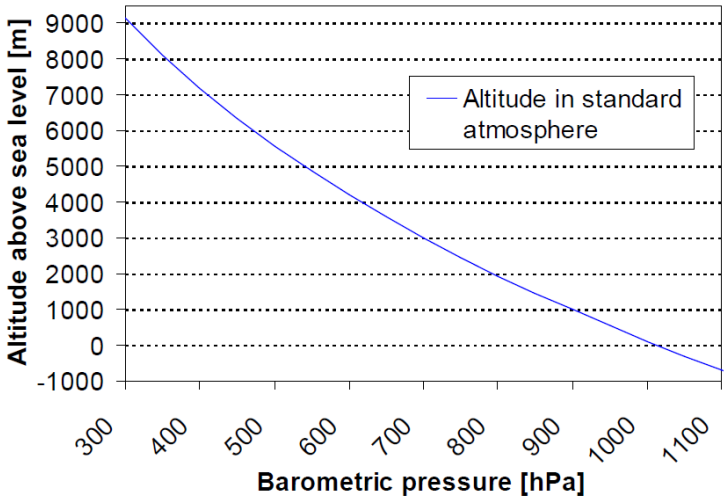
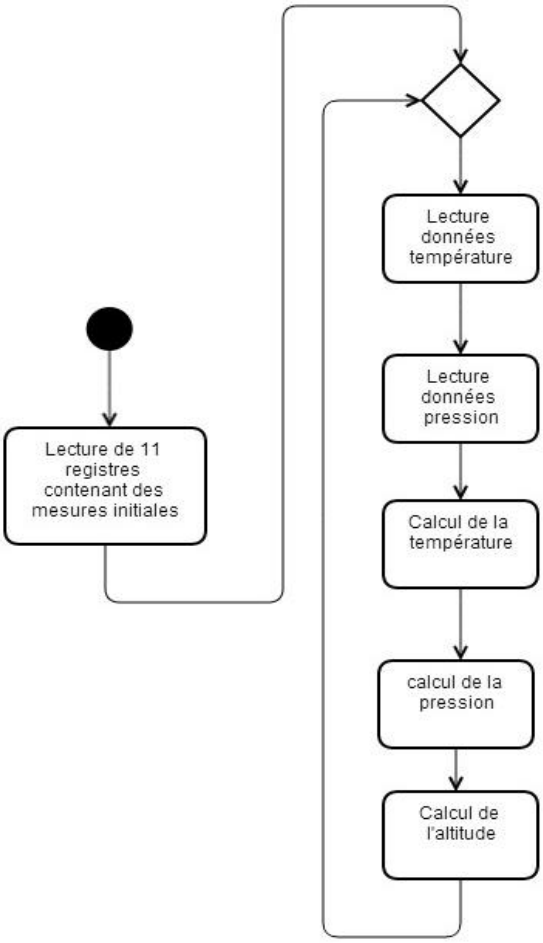


Figure 1: Rapport altitude / pression atmosphérique

Fonctionnement détaillé de la solution :

Le plus gros du travail avec cette solution, est la conversion de la température (également relevée par le capteur) et de la pression atmosphérique en une altitude. Cependant, tous les calculs sont bien expliqués dans la documentation de celui-ci. Voici un aperçu des différentes étapes de l'algorithme →



Une grosse partie de mon temps a été consacrée à la résolution d'un bug qui semblait de base être lié au bus I²C. En effet, au moment du relâchement de celui-ci, le programme plantait. Ce qui m'a dérouté est que le code lié à ce crash n'était exécuté que plus tard. Après plusieurs jours de recherche, j'ai finalement compris qu'il s'agissait d'un problème de mémoire. En effet, lors de la création de chaque tâche sur l'OS, il faut lui assigner une certaine quantité de mémoire. Celle-ci s'est avérée insuffisante, ce qui faisait planter le programme. Après avoir multiplié par 2 la quantité de mémoire allouée à la tâche de l'altitude, le programme tournait sans crasher.

Un autre problème qui m'a occupé pas mal de temps concernait l'exactitude des calculs. En effet, les valeurs retournées ne correspondaient pas aux valeurs réelles. Ici, le problème venait de quelques erreurs qui s'étaient glissées dans le code. En effet, étant donné que j'ai choisis de passer mes valeurs par des pointeurs de structures, il était plus compliqué de s'y retrouvé. Je suis finalement parvenu un premier soir à obtenir une valeur par rapport au niveau de la mer qui semblait bonne. Cependant, lorsque j'ai souhaité reprendre ce code quelques jours plus tard, celui-ci ne fonctionnait plus. J'ai certainement dû modifier quelques choses sans le remettre comme il fallait. Cette erreur ne serait pas arrivée si j'avais pensé à faire un commit de mon projet sur github.

Détail du code associé :

```
/**
*      Déclaration des variables et des types de donnés nécessaires
**/

#define OSS 3 //Oversampling settings (taux d'échantillonage du BMP085
compris entre 0 (faible taux mais peu énergivore) et 3 (8 échantillons
avant envois mais très énergivore))

typedef struct PressureVar PressureVar;
struct PressureVar
{
      long X1;
```

```
	long X2;
	long X3;

	long B3;
	unsigned long B4;
	long B5;
	long B6;
	unsigned long B7;
};

typedef struct BMP085_reg BMP085_reg;
struct BMP085_reg
{
	short ac1[2]; //Registre 0xAA
	short ac2[2]; //Registre 0xAC
	short ac3[2]; //Registre 0xAE
	unsigned short ac4[2]; //Registre 0xB0
	unsigned short ac5[2]; //Registre 0xB2
	unsigned short ac6[2]; //Registre 0xB4

	short b1[2]; //Registre 0xB6
	short b2[2]; //Registre 0xB8

	short mb[2]; //Registre 0xBA
	short mc[2]; //Registre 0xBC
	short md[2]; //Registre 0xBE

};

typedef struct long_BMP085_reg long_BMP085_reg;
struct long_BMP085_reg
{
	short ac1; //Registre 0xAA
	short ac2; //Registre 0xAC
	short ac3; //Registre 0xAE
	unsigned short ac4; //Registre 0xB0
	unsigned short ac5; //Registre 0xB2
	unsigned short ac6; //Registre 0xB4

	short b1; //Registre 0xB6
	short b2; //Registre 0xB8

	short mb; //Registre 0xBA
	short mc; //Registre 0xBC
	short md; //Registre 0xBE
};

/**
*	Lecture de la température comme définis dans la doc en
*	allant interroger les registres associés
**/
long readUncompensatedTemperature()
{
	uint8_t txbuf[2] = {0};
	uint8_t rawTemperature[2] = {0};
	txbuf[0] = 0xF4; //Registrer address //Calibration data ac1
	txbuf[1] = 0x2E; //Temperature
	i2cAcquireBus(&I2CD1);
	i2cMasterTransmitTimeout(&I2CD1, 0x77, txbuf, 2, NULL, 0, 1000);
	i2cReleaseBus(&I2CD1);
	chThdSleepMilliseconds(5);
```

```
	txbuf[0] = 0xF6;
	i2cAcquireBus(&I2CD1);
	i2cMasterTransmitTimeout(&I2CD1, 0x77, txbuf, 1, rawTemperature, 2, 100);
	i2cReleaseBus(&I2CD1);

	return (((uint16_t)rawTemperature[0]<<8)|((uint16_t)rawTemperature[1]));
}

/**
*	Lecture de la pression atmosphérique comme définis dans la doc
*	en allant interroger les registres associés
**/
long readUncompensatedPressure()
{
	uint8_t txbuf[2] = {0};
	uint8_t rawPressure[3] = {0};

	txbuf[0] = 0xF4; //Registrer address //Calibration data ac1
	txbuf[1] = 0x34+(OSS<<6); //Temperature
	i2cAcquireBus(&I2CD1);
	i2cMasterTransmitTimeout(&I2CD1, 0x77, txbuf, 2, NULL, 0, 100);
	i2cReleaseBus(&I2CD1);
	chThdSleepMilliseconds(26);

	txbuf[0] = 0xF6;
	i2cAcquireBus(&I2CD1);
	i2cMasterTransmitTimeout(&I2CD1, 0x77, txbuf, 1, rawPressure, 3, 100);
	i2cReleaseBus(&I2CD1);

	return((long)rawPressure[0]<<16)|
		((long)rawPressure[1]<<8)|
		(long)rawPressure[2])>>(8-OSS);
}

/**
*	Transcription de l'algorithme de calcul de la température
*   comme définis dans la doc
**/
long calculateTemperature(long_BMP085_reg long_registres, PressureVar
*variables, int *uncompensatedTemperature)
{
	variables->X1 = (*uncompensatedTemperature-
				long_registres.ac6)*long_registres.ac5>>15;
	variables->X2 = (long_registres.mc<<11)/(variables->
						X1+long_registres.md);
	variables->B5 = variables->X1+variables->X2;
	return (variables->B5+8)>>4;
}

/**
*	Transcription de l'algorithme de calcul de la pression
*	atmosphérique comme définis dans la doc
**/
long calculatePressure(long_BMP085_reg long_registres, PressureVar
*variables)
{
	int uncompensatedTemperature = 0;
	long uncompensatedPressure = 0;
	int temperature = 0;
	long pressure = 0;

	// récupération des valeurs brute de la température
	uncompensatedTemperature = readUncompensatedTemperature();
```

```
	// récupération des valeurs brute de la pression athmosphérique
	uncompensatedPressure = readUncompensatedPressure();
	// Calcul de la température (même si ce n'est pas le résultat qui
	// nous intéresse mais un calcul intermédiaire
	temperature = calculateTemperature(long_registres, variables,
	                                   &uncompensatedTemperature);

	variables->B6 = variables->B5-4000;
	variables->X1 = (long_registres.b2*(variables->B6*variables->
	                                    B6>>12))/pow(2,11);
	variables->X2 = long_registres.ac2*variables->B6/pow(2,11);
	variables->X3 = variables->X1+variables->X2;
	variables->B3 = ((((int32_t)long_registres.ac1*4+variables->
	                                    X3)<<OSS)+2)/4;
	variables->X1 = long_registres.ac3*variables->B6/pow(2,13);
	variables->X2 = (long_registres.b1*(variables->B6*variables->
	                                    B6/pow(2,12)))/pow(2,16);
	variables->X3 = ((variables->X1+variables->X2)+2)/pow(2,2);
	variables->B4 = long_registres.ac4*(unsigned long)(variables->
	                                    X3+32768)/pow(2,15);
	variables->B7 = ((unsigned long)uncompensatedPressure-variables->
	                                    B3)*(50000>>OSS);

	if(variables->B7 < 0x80000000)
	{
		pressure = (variables->B7*2)/variables->B4;
	}
	else
	{
		pressure = (variables->B7/variables->B4)*2;
	}
	variables->X1 = (pressure/pow(2,8))*(pressure/pow(2,8));
	variables->X1 = (variables->X1*3038)/pow(2,16);
	variables->X2 = (-7357*pressure)/pow(2,16);
	pressure = pressure+(variables->X1+variables->X2+3791)/pow(2,4);
	return pressure;
}

static WORKING_AREA(waAltitude, 128);
static msg_t ThreadAltitude(void *arg)
{
	uint8_t txbuf[10] = {0};

	BMP085_reg registres;
	long_BMP085_reg long_registres;
	PressureVar variablesPression;

	long pressure = 0;
	long altitude = 0;
	long pressure0 = 0;

	//Initialisation du BMP085 en récupérant les valeurs des registres
	                                    définis dans la doc
	//récupération de ac1
	txbuf[0] = 0xAA; //Calibration data ac1
	i2cAcquireBus(&I2CD1);
	i2cMasterTransmitTimeout(&I2CD1, 0x77, txbuf, 1,
	                         (uint8_t*)registres.ac1, 2, 1000);
	i2cReleaseBus(&I2CD1);
	long_registres.ac1 = (short)((registres.ac1[0]<<8)+registres.ac1[1]);
```

```
txbuf[0] = 0xAC; //Calibration data ac2
i2cAcquireBus(&I2CD1);
i2cMasterTransmitTimeout(&I2CD1, 0x77, txbuf, 1,
                                    (uint8_t*)registres.ac2, 2, 1000);
i2cReleaseBus(&I2CD1);
long_registres.ac2 = (registres.ac2[0]<<8)+registres.ac2[1];
//récupération de ac3
txbuf[0] = 0xAE; //Calibration data ac3
i2cAcquireBus(&I2CD1);
i2cMasterTransmitTimeout(&I2CD1, 0x77, txbuf, 1,
                                    (uint8_t*)registres.ac3, 2, 1000);
i2cReleaseBus(&I2CD1);
long_registres.ac3 = (registres.ac3[0]<<8)+registres.ac3[1];
//récupération de ac4
txbuf[0] = 0xB0; //Calibration data ac4
i2cAcquireBus(&I2CD1);
i2cMasterTransmitTimeout(&I2CD1, 0x77, txbuf, 1,
                                    (uint8_t*)registres.ac4, 2, 1000);
i2cReleaseBus(&I2CD1);
long_registres.ac4 = (registres.ac4[0]<<8)+registres.ac4[1];
//récupération de ac5
txbuf[0] = 0xB2; //Calibration data ac5
i2cAcquireBus(&I2CD1);
i2cMasterTransmitTimeout(&I2CD1, 0x77, txbuf, 1,
                                    (uint8_t*)registres.ac5, 2, 1000);
i2cReleaseBus(&I2CD1);
long_registres.ac5 = (registres.ac5[0]<<8)+registres.ac5[1];
//récupération de ac6
txbuf[0] = 0xB4; //Calibration data ac6
i2cAcquireBus(&I2CD1);
i2cMasterTransmitTimeout(&I2CD1, 0x77, txbuf, 1,
                                    (uint8_t*)registres.ac6, 2, 1000);
i2cReleaseBus(&I2CD1);
long_registres.ac6 = (registres.ac6[0]<<8)+registres.ac6[1];

//récupération de b1
txbuf[0] = 0xB6; //Calibration data b1
i2cAcquireBus(&I2CD1);
i2cMasterTransmitTimeout(&I2CD1, 0x77, txbuf, 1,
                                     (uint8_t*)registres.b1, 2, 1000);
i2cReleaseBus(&I2CD1);
long_registres.b1 = (registres.b1[0]<<8)+registres.b1[1];
//récupération de b2
txbuf[0] = 0xB8; //Calibration data b2
i2cAcquireBus(&I2CD1);
i2cMasterTransmitTimeout(&I2CD1, 0x77, txbuf, 1,
                                     (uint8_t*)registres.b2, 2, 1000);
i2cReleaseBus(&I2CD1);
long_registres.b2 = (registres.b2[0]<<8)+registres.b2[1];

//récupération de mb
txbuf[0] = 0xBA; //Calibration data mb
i2cAcquireBus(&I2CD1);
i2cMasterTransmitTimeout(&I2CD1, 0x77, txbuf, 1,
                                     (uint8_t*)registres.mb, 2, 1000);
i2cReleaseBus(&I2CD1);
long_registres.mb = (registres.mb[0]<<8)+registres.mb[1];
//récupération de mc
txbuf[0] = 0xBC; //Calibration data mc
i2cAcquireBus(&I2CD1);
```

```
	i2cMasterTransmitTimeout(&I2CD1, 0x77, txbuf, 1,
                                        (uint8_t*)registres.mc, 2, 1000);
	i2cReleaseBus(&I2CD1);
	long_registres.mc = (registres.mc[0]<<8)+registres.mc[1];
	//récupération de md
	txbuf[0] = 0xBE; //Calibration data md
	i2cAcquireBus(&I2CD1);
	i2cMasterTransmitTimeout(&I2CD1, 0x77, txbuf, 1,
                                        (uint8_t*)registres.md, 2, 1000);
	i2cReleaseBus(&I2CD1);
	long_registres.md = (registres.md[0]<<8)+registres.md[1];

	pressure0 = calculatePressure(long_registres, &variablesPression);
	while(TRUE)
	{
		//Calcul de la pression athmosphérique
		pressure = calculatePressure(long_registres,
                                                &variablesPression);
		//Calcul de l'altitude en fonction de la pression
		// athmosphérique
		altitude = 44330.75*(1-pow((double)pressure/pressure0,
                                                0.19029))*100;
		//Stockage de l'altitude dans une variable globale
		g_altitude = altitude;
	}
}
```

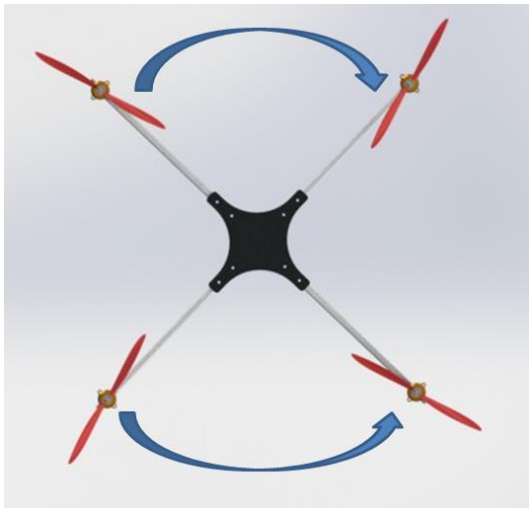
Conclusion du CU.

Après quelques heures passées dessus, je suis finalement parvenu à obtenir des valeurs qui semblent correspondre à la réalité. Certes celle-ci ne sont pas référencées par rapport au niveau de la mer mais par rapport au point de départ. Ce qui correspond au mode de fonctionnement dont j'aurais besoin. Cependant, les valeurs ne sont pas très précises. En effet une erreur de plus ou moins 1 mètre est à prendre en compte alors que la datasheet du capteur donne une précision de 25cm. Il est possible que cette erreur soit due à un problème d'initialisation des variables ou encore au fait que les tests aient tous été effectués en intérieur. Il serait donc judicieux de tester celui-ci en extérieur.

Gestion du lacet

Présentation :

En aéronautique, le lacet correspond à la rotation autour de l'axe de la verticale. Il est important que cette rotation soit maîtrisée. Dans le cas contraire, il serait compliqué pour le pilote de maîtriser la direction de l'engin.



Solutions envisagées :

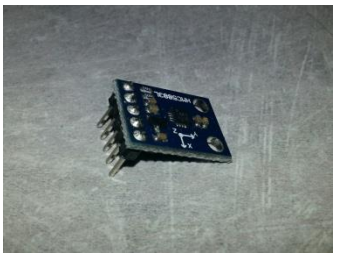
Utilisation du gyroscope :

La première solution envisagée fut d'utiliser l'axe correspondant qu'offrait le gyroscope choisit pour la solution concernant le tangage et le roulis. Cette solution me permettait de ne pas avoir à acheter de matériel supplémentaire et également de n'avoir que très peu de code à rajouter et donc une occupation système limitée. Cette solution permettait également de limiter la place occupée sur le PCB final. En effet, il est important de veiller à avoir un PCB minimal pour plusieurs raisons. La première est qu'un PCB plus large est systématiquement plus lourd. Et il faut garder à l'esprit que le poids est l'ennemi numéro 1 de tout bon matériel volant. Le deuxième avantage à avoir un PCB minimal est le prix qu'il coutera à réaliser. En effet, les fabricants calculent le prix de celui-ci en fonction de sa surface. Un PCB plus grand entrainera donc un coût plus important.

Cependant, pour le moment cette solution n'a pas été retenue car il m'a été impossible de récupérer les données sur l'axe correspondant. Je reviendrais peut-être sur cette solution plus tard, mais il m'a semblé important d'avancer sur des points plus complexes.

Utilisation de la boussole

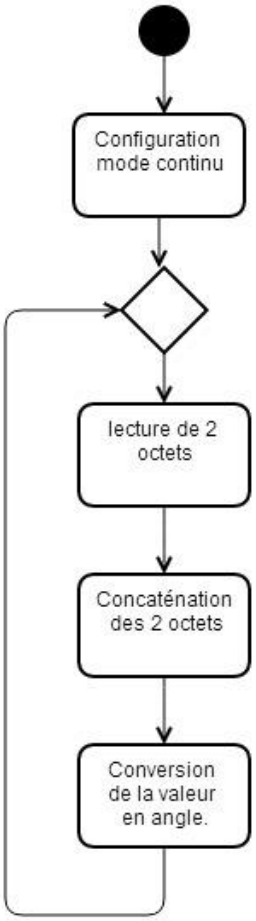
La solution que j'ai finalement adoptée est l'utilisation d'une boussole électronique. Avec cet élément, je connaîtrais en permanence mon angle en fonction du Nord. Il me suffira donc de me baser sur celui-ci. De plus, ayant déjà eu l'occasion de travailler un peu sur ce type de matériel, je savais que celui-ci pourrait répondre à mes attentes. Le capteur que j'ai choisis d'utiliser est le HMC5883L. J'ai pu trouver celui-ci pour environ 8€. De plus, celui-ci communique via le bus I²C, ce qui correspond à mes attentes.



Fonctionnement détaillé de la solution :

Le fonctionnement de cette boussole est extrêmement simple. Il me suffit de lire régulièrement sur le bus I²C 2 octets provenant de ce capteur qui a été préalablement placé en mode continu (envoi de données en permanence). Une fois la lecture terminée, je concatène les 2 valeurs pour former une variable sur 16 bits. Il me faut ensuite convertir cette donnée pour avoir un angle.

Pour le moment, certaines valeurs provenant de la boussole sont lues. Cependant celles-ci ne correspondent pas exactement à ce que j'espérais. Il faut donc que je continu à creuser de ce côté-ci.



Conclusion du CU

Le travail réalisé sur ce CU n'est pour l'instant qu'une ébauche. Celui-ci risque d'être grandement modifié voir même de disparaître si je parviens finalement à utiliser les données venant du gyroscope. Ce qui serait profitable d'un point de vue ressource utilisée et code développé.

Gestion de la communication

Présentation :

Pour permettre à l'utilisateur de piloter le drone, une communication sûre doit être établie. Celle-ci doit permettre un protocole permettant le contrôle de l'engin sur tous les axes et éviter au maximum les erreurs de communication. Il faut également que celle-ci soit fonctionnelle sur la distance demandée. C'est-à-dire 1.5km.

Solutions envisagées :

Radio fréquence:

La radio fréquence est le système de base utilisé dans le modélisme. Ce système à la particularité d'être très robuste et de permettre l'émission sur plusieurs canaux. Ce qui permet d'envoyer plusieurs informations à la fois.

Wi-Fi:

Le Wi-Fi est un mode de communication sans fil. On retrouve beaucoup celui-ci dans les box d'accès à internet. Celui-ci à une portée pouvant aller jusqu'à 150m. Ce qui est très nettement insuffisant dans mon cas. Je n'ai donc pas choisi ce module.

XBee Pro Série 2:

Le XBee Pro Série 2 est un petit module permettant d'avoir une liaison série avec un autre module du même type en utilisant le protocole zigBee. Ce module a été conçu pour être aussi simple à utiliser qu'une liaison filaire. De plus, sa portée est de 1.5km en champ libre. Il pourrait donc convenir à l'usage que j'ai à en faire. Cependant, ce produit à un coût. En effet, il faut compter environ 40€ par module. Sachant qu'il m'en faut 2 (un sur la partie embarquée et un sur la partie commande) et qu'il faut également un appareil permettant de les configurer (environ 10€), il faudrait que je compte environ 90€ uniquement pour la transmission.

Etat actuel du CU

Mon choix serait actuellement d'avantages axé sur les modules XBee mais de récentes recherches m'ont permises de m'apercevoir qu'un module de radio fréquence pourrait être également utilisable. Aucun choix n'est donc fixé pour le moment et la transmission reste filaire. Cependant, une partie du code est développée. En effet, si j'utilise une liaison XBee, le code ressemblera en grande partie à celui déjà développé. En voici un aperçu :

```
/**
*      Structure contenant les paramètres de la liaison série
**/
static SerialConfig uartCfg=
{
       57600,           /* Baudrate                    */
       0,               /* cr1 register values  */
       0,               /* cr2 register values  */
       0                /* cr3 register values  */
};

/**
*      Réception des données provant de la télécommande (PC)
**/
static WORKING_AREA(waComSnd, 128);
static msg_t ThreadComSnd(void *arg)
{
      char dataRead[50] = {0};

      while(TRUE)
      {
            //Réception des données
            sdRead(&SD2, (uint8_t*)dataRead, 50);
      }
}

/**
*      Envoi des valeurs des capteurs à la partie PC
**/
static WORKING_AREA(waComRcv, 128);
static msg_t ThreadComRcv(void *arg)
```

```
{
	// gestion de la communication (Xbee Pro Serie 2)

	//déclaration d'un string pouvant contenir toute la trame
	char dataWrite[]="T123R456L789A12345S";

	//Initialisation du module Xbee lorsque j'aurais celui-ci
	// ...

	// Initialisation de la liaison série
	palSetPadMode(GPIOA, 2, PAL_MODE_ALTERNATE(7));
	palSetPadMode(GPIOA, 3, PAL_MODE_ALTERNATE(7));
	sdStart(&SD2, &uartCfg);

	while(TRUE)
	{
		//Concaténation des valeurs dans la trame
		sprintf(dataWrite, "T%03dR%03dL789A%05dS", g_tangage, g_roulis,
		                                                    g_altitude);
		//Envois des données sur la liaison série
		sdWrite(&SD2, (uint8_t*)dataWrite, strlen(dataWrite));
	}
}
```

Choix des moteurs

Présentation :

Le choix des moteurs est à faire en fonction du poids qu'ils devront porter. J'ai pu réaliser ce calcul à l'aide d'une feuille Excel que j'ai pu trouver sur internet et qui permettait de donner une idée des moteurs, des hélices, des variateurs et de la batterie à choisir en fonction du poids et de l'autonomie de l'appareil (cf Annexe).

Solution envisagée :

Il en a donc résulté que le bon choix serait des moteurs brushless proposant une puissance de 1000kv (1000 tours par minute pour un Volt). Le voltage de ceux-ci est à choisir en fonction de la batterie utilisée. Mais il faudrait que je m'oriente vers 4 moteurs alimentés entre 7.4 et 14.8V et alimentés en 20 Ampères.

Mon choix final s'est porté sur des moteurs de marque NTM d'une puissance de 1100kv (1100 tours pour un volt) et de 252 watts à 14€ l'unité. Ceux-ci ne pèsent que 57 grammes chacun.

Choix des ESC

Présentation

Les ESC (pour Electronic Speed Control) permettent de faire varier la vitesse des moteurs. Ils reçoivent les informations en provenance du microcontrôleur et adaptent les 3 phases du moteur brushless connecté dessus pour en faire varier la vitesse. On les trouve aussi sous le nom de variateur ou contrôleur.

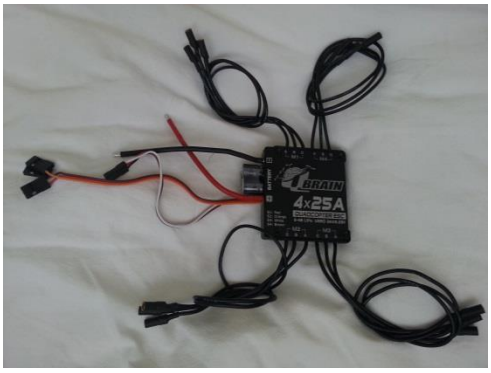
ESC piloté par PWM:

Le type de contrôleurs le plus courant est ceux piloté par une PWM. Ceux-ci ont pour avantage de ne pas être trop chère et ont un fonctionnement très simple à comprendre : Plus le rapport cyclique envoyé sur l'entrée de l'ESC sera élevé, plus la vitesse de rotation du moteur le sera également. La précision de la vitesse sera donc fonction de la précision du timer sur le microcontrôleur utilisé. L'inconvénient de cette solution est qu'il faut une sortie PWM pour chaque moteur sur le microcontrôleur. Il faut donc s'assurer que celui-ci soit compatible.

ESC piloté par I^2C :

Le $2^{ème}$ type de contrôleur concerne ceux connecté sur un bus I^2C. Dans ce cas, chaque variateur à une adresse et le microcontrôleur n'a qu'à envoyer la valeur que celui-ci doit prendre sur le bus. L'avantage de cette solution est que seul 2 fils sont nécessaires pour brancher tous les variateurs au microcontrôleur et ceux-ci permettent des vitesses de rotation plus élevées. Cependant, ces variateurs sont beaucoup plus chers que ceux pilotés par PWM et je ne suis pas sûr que ces quelques améliorations justifient la différence de prix.

Solution retenue :



Mon choix s'est finalement porté sur des variateurs piloté par une PWM. De plus, pour des raisons de coûts et de poids, je me suis orienté vers un produit de la marque QBrain qui permet de réunir les 4 variateurs dans un seul package. J'ai donc opté pour un système réunissant 4 variateurs 25 ampères et pouvant être alimenté par une batterie LiPo de 2 à 4 cellules. Cependant, d'après mes recherches sur internet, il semblerait que ce produit à tendance à beaucoup chauffer. Il faudra donc que je veille à le placer à un endroit bien ventiler sur le quadricoptère en espérant que celui-ci n'endommagera pas l'appareil. J'ai pu trouver celui-ci pour 24.23€.

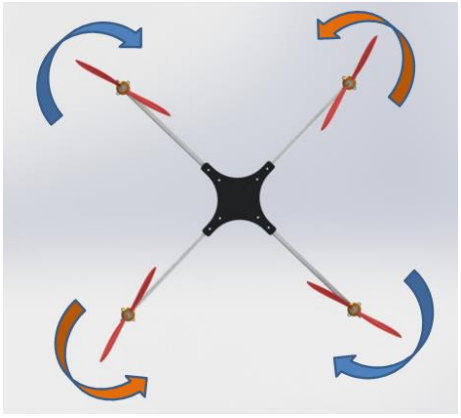
Choix des hélices

Présentation :

Les hélices seront fixées directement aux moteurs. C'est elles qui porteront le quadricoptère. Il faut donc qu'elles soient suffisamment grandes pour avoir une surface de portance assez élevée mais pas trop pour pouvoir être entrainées par les moteurs sans que ceux-ci ne forcent. Il faut également choisir un pas d'hélice adapté. En effet, un pas trop petit contraindra les moteurs à faire plus de tour pour que le système s'élève. Un pas trop grand risque de demander un couple trop élevé aux moteurs.

Solution envisagée

Le choix des hélices est à faire en liens avec le poids du quadricoptère et des moteurs choisis. Dans mon cas, d'après la même feuille de calcul qui m'a permis de déterminer le type de moteurs à utiliser, j'ai choisis des hélices de 10 pouces et un pas de 4.5 pouces à 2.30€ les 4.



Lors de l'achat des hélices, il est important de prendre 2 hélices avec un pas dans le sens horaire et 2 hélices avec un pas dans le sens trigonométrique. En effet, afin de gérer le lacet, les moteurs tournent 2 par 2 comme indiqué sur la figure à droite. Ainsi, si nous souhaitons que le quadricoptère tourne sur lui-même dans le sens horaire, il suffit de diminuer la vitesse des moteurs tournants dans le sens trigonométrique et augmenter celle des 2 autres.

Choix de la batterie

Le choix de la batterie doit se faire en lien avec les moteurs et le variateur choisis. En effet, celle-ci doit être suffisamment puissante pour alimenter au maximum de leur capacité les moteurs. Dans le cas contraire, une partie de cette puissance serait gâchée. Ce qui serait un manque à gagner aussi bien pour le poids que pour l'argent dépensé. Cette batterie devra également permettre l'alimentation du microcontrôleur et des différents périphériques. Il sera donc certainement nécessaire de faire un morceau de circuit électronique pour convertir la tension de sortie de la batterie en une tension admissible par celui-ci.

La batterie qui a été choisie est une batterie LiPo (Lithium Potassium). En effet, malgré l'instabilité de celles-ci, ce sont ces batteries qui offrent le meilleur ratio poids/puissance/prix. La capacité de cette batterie est de 3300mAh et dispose de 4 cellules. Celle-ci délivre une tension de 14.8V, ce qui est adapté aux moteurs et aux variateurs. Celle-ci m'a coûté 26.00€

J'ai également acheté un chargeur capable de recharger cette batterie sans l'endommager à 16.79€

Gestion de la vidéo

Présentation

La dernière étape de ce projet serait de mettre un système vidéo pour pouvoir piloter le drone en FPV. Le FPV (First Person View) est le fait de piloter le drone en se basant sur les images provenant d'une caméra embarquée. Ce système permet de piloter sur de plus longue distance, ce qui ne pouvait être fait au-par-avant. En effet un visuel est nécessaire pour se diriger et prévenir d'une éventuelle perte de contrôle du système.

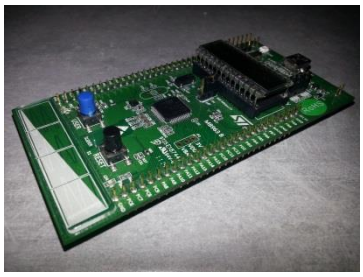
Solution envisagée

Concernant la vidéo, je pense utiliser une caméra de type GoPro ou un modèle équivalent mais moins cher. Concernant la transmission, ma première idée fut d'utiliser le même module que celui qui me permet de guider l'appareil. Cependant, il semblerait que la vitesse de celui-ci ne soit pas suffisante. J'ai donc décidé d'utiliser un module spécialement prévu à cet effet et qui permet d'atteindre 1km. Afin d'étendre sa portée à 1.5km, il me faudra changer l'antenne par une PinWheel qui de par sa forme permet d'émettre plus loin.

Figure 2: Antenne PinWheel

Choix du microcontrôleur

Pour réaliser ce projet, je me suis naturellement lancé sur un microcontrôleur STM32 de chez ST. Si j'ai choisis ce composant, c'est uniquement car j'ai déjà eu l'occasion de travailler dessus lors de projets durant mon DUT informatique. Sachant que celui-ci pourrait répondre à mes besoins (bus I^2C et fréquence en particulier), j'ai choisis d'acheter une plaque de démonstration vendue par ST incluant le microcontrôleur, un debugger et un certain nombre de périphériques pour 10€. De plus, cette plaque peut être installée sur une breadboard (plaque de prototypage), ce qui me facilite considérablement toute ma phase de tests.

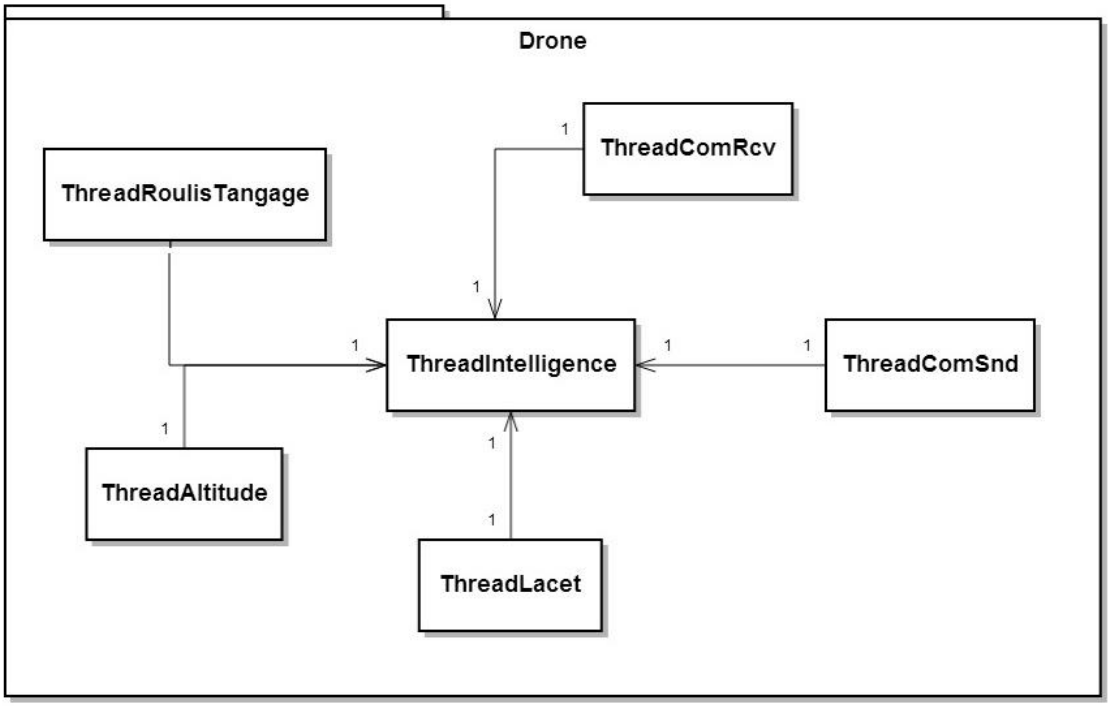


Choix du système d'exploitation

Il m'a semblé judicieux pour un projet de cette importance d'utiliser un système d'exploitation qui me permettrait d'avoir plusieurs tâches qui tourneraient en même temps. Mon premier choix ses basé sur FreeRTOS. En effet, j'ai eu l'occasion durant mon DUT Informatique de travailler sur cet OS. Cependant, j'ai dû abandonner cette voie car même si celui-ci existait pour mon microcontrôleur, il n'avait été porté qu'à l'aide de l'environnement de développement iAR qui est une suite payante et extrêmement chère. J'ai donc choisi de m'orienter vers un autre O.S.

Mon 2ème choix fut chibiOS. Cet O.S. est fourni sur un grand nombre de microcontrôleurs différents et qui plus est, sur la suite de développement que j'ai pris l'habitude d'utiliser durant mon DUT : Keil. Cet O.S. offre une bonne couche d'abstraction avec les différents périphériques de mon microcontrôleur. Cela me permet donc par exemple d'utiliser le bus I²C sans avoir à manipuler directement les registres de celui-ci.

Voici comment sont organisées les tâches à l'aide de ChibiOS :



La tache ThreadIntelligence permet le calcul de la poussée à mettre sur chaque moteur. Ces calculs se trouvent en réalité dans la boucle while du main. Peu de choses ont été faites jusqu'à présent sur cette partie si ce n'est un début de paramétrage des différentes PWMs nécessaires pour chaque moteur. Un problème sur lequel je bloque encore est le fait que la documentation du microcontrôleur indique 4 PWMs sur 4 timers différents disponibles. Cependant, l'OS que j'utilise ne me laisse en utiliser que 3. Je suppose donc qu'il doit en utiliser une pour son propre fonctionnement. Je vais donc voir s'il y a moyen de mettre toute les PWMs sur le même timer mais de jouer avec les channels. Les channels sont des outils qui permettent de fournir plusieurs PWM différentes avec un même timer simplement en jouant sur les prescalaires (coefficient diviseur du timer) et autres paramètres. Voici une ébauche du main actuel avec le lancement des différentes tâches, l'initialisation du bus I²C et de 3 PWMs sur des timers différents :

```
I2CDriver *i2cDriver1;
const I2CConfig *i2cConfig1;

/**
*      Structure configuration I²C
**/
static const I2CConfig i2ccfg1 =
{
      OPMODE_I2C,
```

```
        400000,     // Vitesse I²C
        FAST_DUTY_CYCLE_2
};
/**
*       Code à exécuter sur un front descendant de la PWM4
**/
static void pwmpcb4(PWMDriver *pwmp)
{
        (void)pwmp;
        palClearPad(GPIOB, GPIOB_PIN12); //Mettre GPIOB12 à 0
}
/**
*       Code à exécuter sur un front montant de la PWM4
**/
static void pwmc4cb(PWMDriver *pwmp)
{
        (void)pwmp;
        palSetPad(GPIOB, GPIOB_PIN12); //Mettre GPIOB12 à 1
}

/**
*       Structure configuration PWM4
**/
static PWMConfig pwmcfg4 = {
        10000,          /* 10kHz PWM clock frequency.   */
        10000,          /* PWM period 1S (in ticks).    */
        pwmpcb4,
        {
                {PWM_OUTPUT_ACTIVE_HIGH, pwmc4cb},
                {PWM_OUTPUT_ACTIVE_HIGH, NULL},
                {PWM_OUTPUT_DISABLED, NULL},
                {PWM_OUTPUT_DISABLED, NULL}
        },
        /* HW dependent part.*/
        0,
        0
};

int main(void)
{
        halInit();
        chSysInit();

        /*
        * Initialisation de l'I²C
        */
        //I²C 1
        palSetPadMode(GPIOB, 10, PAL_MODE_ALTERNATE(4) |
                                        PAL_STM32_OTYPE_OPENDRAIN);     /* SCL */
        palSetPadMode(GPIOB, 11, PAL_MODE_ALTERNATE(4) |
                                        PAL_STM32_OTYPE_OPENDRAIN);     /* SDA */

        i2cInit();
        i2cObjectInit(&I2CD1);
        i2cStart(&I2CD1, &i2ccfg1);

        pwmStart(&PWMD4, &pwmcfg4);
        palSetPadMode(GPIOB, GPIOB_PIN12, PAL_MODE_OUTPUT_PUSHPULL);
        //Pramétrage du rapport cyclique de la PWM
        //1000 sur le dernier paramètre veut dire rapport cyclique de 10%
        //car 1000/10000 = 10% (10000 = vitesse timer)
```

```
	pwmEnableChannel(&PWMD4, 0, PWM_PERCENTAGE_TO_WIDTH(&PWMD4, 1000));

	/*
	 * Creates the threads.
	 */
	chThdCreateStatic(waComRcv, sizeof(waComRcv), NORMALPRIO,
	                                                    ThreadComRcv, NULL);
	chThdCreateStatic(waComSnd, sizeof(waComSnd), NORMALPRIO,
	                                                    ThreadComSnd, NULL);
	chThdCreateStatic(waLacet, sizeof(waLacet), NORMALPRIO, ThreadLacet,
	                                                                  NULL);
	chThdCreateStatic(waRoulisTangage, sizeof(waRoulisTangage)+2000,
	                                NORMALPRIO, ThreadRoulisTangage, NULL);
	chThdCreateStatic(waAltitude, sizeof(waAltitude)+1000, NORMALPRIO,
	                                                  ThreadAltitude, NULL);

	while (TRUE) {
		//Calculs pour l'intelligence à faire ici
		chThdSleepMilliseconds(500); // Pause de 500ms
	}
}
```

Filtrage des données

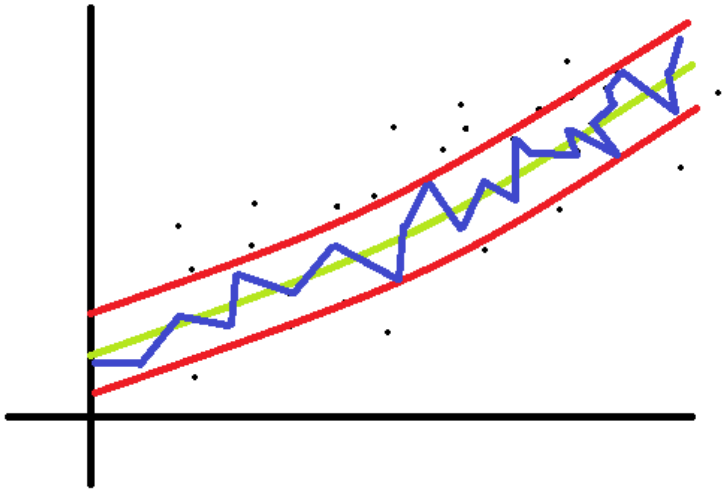
Les données provenant des capteurs peuvent difficilement être utilisées brutes. En effet, en fonction de la qualité du capteur, un bruit plus ou moins fort fait son apparition. Le bruit est le fait de recevoir une donnée se rapprochant de la valeur réelle mais contenant une erreur. Si ce bruit n'est pas pris en compte, le drone ne sera pas stable et deviendra plus compliqué à maîtriser. De plus, si un système vidéo est posé dessus, la capture ne sera pas aussi belle qu'elle l'aurait été avec l'utilisation de filtres.

Pour éliminer cette erreur, plusieurs filtres sont possibles. Les filtres sont des outils mathématiques plus ou moins complexes permettant au final d'avoir des valeurs se rapprochant d'avantages de l'état réel. Dans mon cas, j'ai commencé à regarder à 2 filtres différents qui sont « le filtre de deuxième ordre » et « le filtre de Kalman ».

Je ne ferais ici qu'une brève présentation de ces 2 filtres sans entrer dans la partie calcul n'ayant pas vraiment eut le temps de me pencher dessus.

Le filtre de deuxième ordre

Le filtre de deuxième ordre permet de supprimer les valeurs aberrantes. Il combine le filtre passe-haut qui ne laisse passer que les valeurs au-dessus d'une certaine limite et le filtre passe-bas qui élimine les valeurs en dessus d'une autre limite. Le signal qui sortira de ce filtre ne sera donc pas lisse mais il ne restera que des valeurs plus probables. Voici sous forme de schéma ce que nous aurions :



Les points noirs : les valeurs sorties du capteur

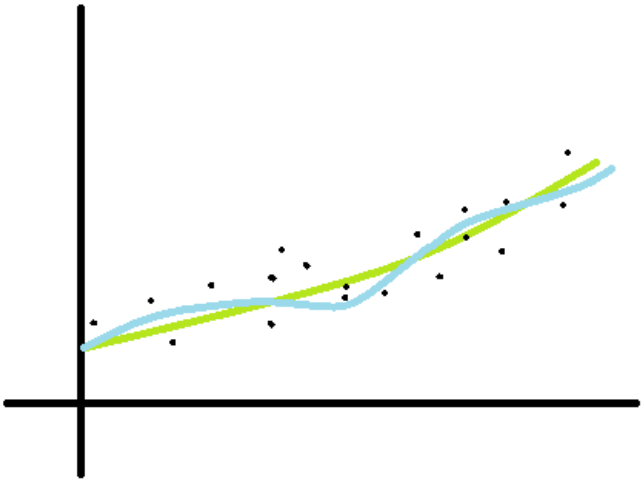
La courbe verte : l'état réel

Les courbes rouges : les limites haut et bas du filtre

La courbe bleue : Le signal qui sortira du filtre

Le filtre de Kalman

Le filtre de Kalman est un outil extrêmement puissant qui permet d'obtenir une estimation relativement précise de l'état réel du capteur en fonction de son signal bruité. Cependant, celui-ci est compliqué à mettre en place et nécessite pas mal de temps processeur. Pour le mettre en place, il est également important de connaître la courbe de gauss associée au bruit généré par les capteurs. Le signal qui sortira de ce filtre sera lisse est très proche du signal réel.



Les points noirs : les valeurs sorties du capteur

La courbe verte : l'état réel

La courbe bleue : Le signal qui sortira du filtre

Calcul de stabilisation

Une très grosse partie du travail consistera en la mise en place de l'ensemble des calculs qui permettront au drone de se stabiliser. Malgré les quelques recherches que j'ai commencé à faire et avec les documents fournis par Mr Gomez, je n'ai, pour l'instant, pas suffisamment avancé sur le sujet pour présenter quoi que ce soit.

Partie commande

Présentation

Concernant la partie commande, j'ai choisis d'utiliser mon ordinateur plutôt que d'utiliser les télécommandes traditionnelles. Mon choix s'est porté sur cette technologie au vue du prix de ces télécommandes. En effet, il faut compter minimum 150€ pour ce type de matériel et certainement plus pour atteindre les 1500 mètres. De plus, le but de ce projet était d'en réaliser la plus grande partie possible moi-même.

J'ai donc réalisé une interface graphique en C fonctionnant sur l'ordinateur à l'aide de la bibliothèque SDL. J'utilise également un port com. de mon ordinateur pour communiquer avec la maquette. Pour le moment, la transmission n'est que filaire étant donné que je ne me suis pas encore réellement décidé sur quel matériel je vais travailler pour la communication.

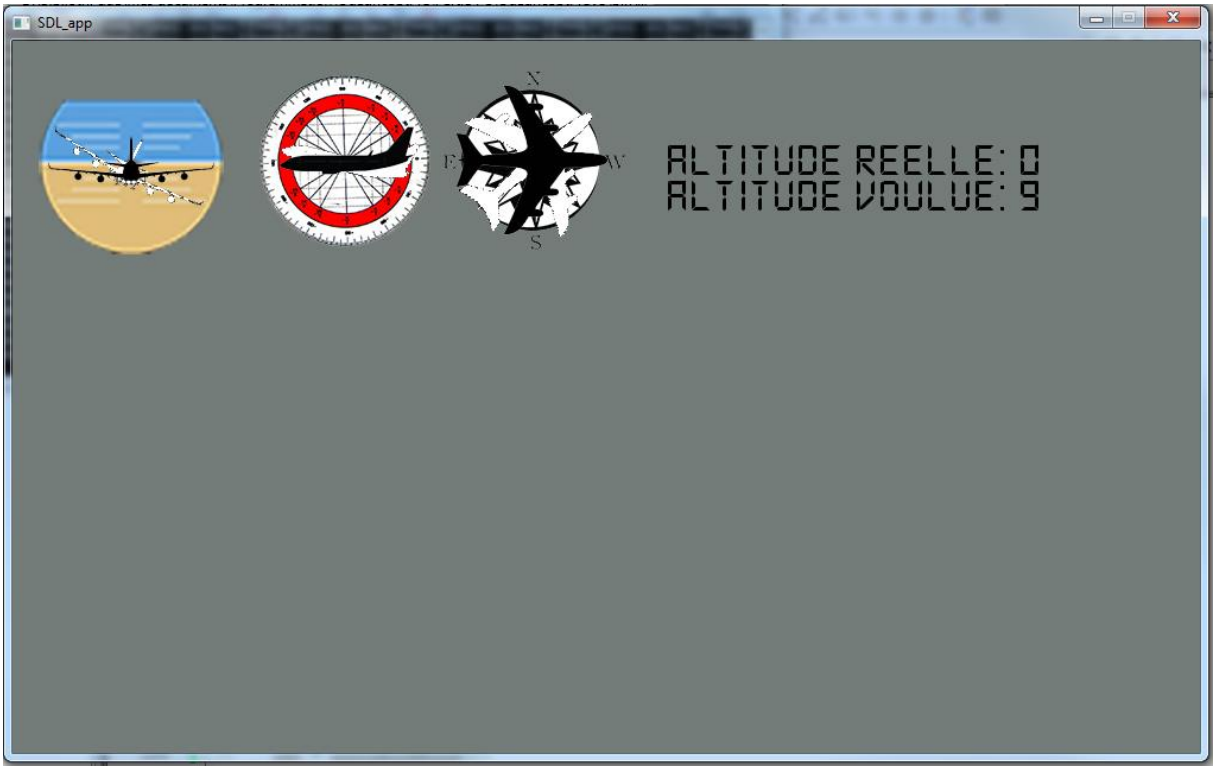


Figure 3: Etat actuel de l'interface graphique

Description de l'interface graphique :

De gauche à droite, nous avons les éléments suivants : Aperçut du roulis, aperçut du tangage, aperçut du lacet, altitude.

Pour chaque donnée, nous avons l'état réel de l'appareil (en noir sur les jauges) et l'état désiré (en blanc). J'ai choisis de faire la différence entre les 2 états pour 2 raisons : La première raison concerne le temps de réaction de l'appareil. Avec cette méthode, on a un aperçut pratique de si l'appareil a atteint ou non la position demandée. La 2ème raison prend son sens lorsque l'appareil est déplacé par un élément extérieur au système, par exemple le vent ou si celui-ci a touché le sol.

Pour le moment, pour commander l'appareil, l'utilisateur a besoin des touches Z, S, Q, D ainsi que de la souris. Voici la répartition des commandes :

- Z et S : faire monter et descendre le drone (commande de l'altitude)
- Q et D : faire tourner sur lui-même le drone (commande du lacet)
- Souris vers la gauche et vers la droite : Faire pencher le drone sur la droite ou la gauche (commande du roulis)
- Souris vers le haut et vers le bas : Faire penchez le drone vers l'avant ou vers l'arrière (commande du tangage)

Dans une version future, j'envisage de remplacer la souris par un joystick ou voir même remplacer tous les éléments de pilotage par un système de leap motion (contrôle à l'aide des gestes de la main).

Communication

La communication entre le système embarqué et la partie PC est faite par une liaison série. Sur cette liaison, j'ai choisi d'envoyer une trame qui me permet de stocker toute les données relatives à la position du quadricoptère. Voici comment la trame se forme :

Txxx	Rxxx	Lxxx	Axxxxx	S
Tangage	Roulis	Lacet	Altitude	Stop

Un exemple de trame pourrait donc être : T123R012L023A12345S

Dans cet exemple, nous aurions un tangage de 123° et un roulis de 12° par rapport à l'horizontal, un Lacet de 23° par rapport au Nord et une altitude de 12345 cm par rapport au point de départ.

Organisation du code source

Introduction

Afin de faciliter la rédaction du code source de l'application PC et avoir un programme modulaire, j'ai choisis de fractionner le programme en plusieurs sous-systèmes indépendants les uns des autres. Dès que j'ai commencé à rédiger celui-ci, j'ai souhaité que ces sous CU tournent en parallèle sans se soucier de la péremption des données. Je ne suis pas sûr qu'il s'agisse de la meilleure façon de s'y prendre mais cela me facilitais la tâche.

Les différentes sous-tâches de ce programme sont les suivantes :

- Réception
- Emission
- Affichage

Cependant, avant de lancer ces différentes tâches, une phase d'initialisation est nécessaire. C'est durant celle-ci que nous allons entre-autre nous connecter au port COM de l'ordinateur sur lequel a été branché la liaison série reliée au quadricoptère.

Explication du code :

```
//prototypes
int initialization();
void dataReceiving(void *data);
void dataSending(void *data);
void displaying(void *data);

enum //Etats de la machine à état
{
    ComPort_Open,
    ComPort_Config,
    ComPort_Poll,
    ComPort_Send,
    ComPort_End
};

typedef struct //Stucture pour contenir les évènements clavier/souris
{
    char key[SDLK_LAST];
    int mouseX, mouseY;
    int mouseXRel, mouseYRel;
    char mouseButtons[8];
    char quit;
} Input;

Input in;
int g_comport;
HANDLE g_hComm;

int g_angleTangageReel = 0;
int g_angleTangageSouris = 0;
int g_angleRoulisReel = 0;
int g_angleRoulisSouris = 0;
int g_angleLacetReel = 0;
int g_angleLacetSouris = 0;
int g_altitudeReel = 0;
int g_altitudeClavier = 0;

int main ( int argc, char** argv )
{
    //activation de la console avec la SDL
    freopen("CON","w", stderr);
    freopen("CON","w", stdout);

      //déclaration des threads
    pthread_t receivData;
    pthread_t sendData;
```

```
    pthread_t display;

    int ret = 0;
      //lancement de l'initialisation
    ret = initialization();

    if(ret == 0)
    {
        //Création de 3 threads qui vont gérer l'envois, la réception et
                                                                l'affichage:
        ret = pthread_create(&receivData, NULL, (void*)dataReceiving,
                                                                    NULL);
        if(ret)
        {
            printf("Erreur lors de la creation du thread de réception:
                                                                %d\n", ret);
        }

        ret = pthread_create(&sendData, NULL, (void*)dataSending, NULL);
        if(ret)
        {
            printf("Erreur lors de la creation du thread d'envois: %d\n",
                                                                    ret);
        }

        ret = pthread_create(&display, NULL, (void*)displaying, NULL);
        if(ret)
        {
            printf("Erreur lors de la creation du thread d'affichage:
                                                                %d\n", ret);
        }

        pthread_join(receivData, NULL);
        pthread_join(sendData, NULL);
        pthread_join(display, NULL);
    }
    return EXIT_SUCCESS;
}

int initialization()
{
    int retour = 0;
    int State = ComPort_Open;
    LPCWSTR port[20];
    char read[50] = {0};
    char tempRead[20] = {0};
    DWORD dwCommEvent;
    char chRead;
    DCB config;
    int i = 0;
    int j = 0;

    memset(&in, 0, sizeof(in)); //Initialisation du tableau pour la gestion
                                                        du clavier/souris

      //On demande à l'utilisateur sur quel port COM se connecter
    printf("Comport: ");
    scanf("%d", &g_comport);
    sprintf((char*)port, "\\\\.\\COM%d", g_comport);

      //machine à état pour l'initialisation
```

```
while(State != ComPort_End)
{
    switch(State)
    {
    case ComPort_Open:
              // On ouvre le port...
        g_hComm = CreateFile((LPCSTR)port, GENERIC_READ |
 GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING,
                                                 FILE_ATTRIBUTE_NORMAL, NULL);

        if(g_hComm == INVALID_HANDLE_VALUE)
        {
            printf("impossible de se co...%d\n", (int)GetLastError());
            CloseHandle(g_hComm);
            return -1;
        }
        else
        {
            printf("Connexion ... Ok\n");
            State = ComPort_Config;
        }
        break;

    case ComPort_Config:
              //On configure le port
        config.DCBlength = sizeof(DCB);
        if(GetCommState(g_hComm, &config) == 0)
        {
            printf("Erreur lors de la config!");
            return -1;
        }
        else
        {
            config.BaudRate = 57600;      // Current baud
            config.fBinary = TRUE;        // Binary mode; no EOF check
            config.fParity = TRUE;        // Enable parity checking
            config.fOutxCtsFlow = FALSE;  // No CTS output flow control
            config.fOutxDsrFlow = FALSE;  // No DSR output flow control
            config.fDtrControl = DTR_CONTROL_ENABLE;
            // DTR flow control type
            config.fDsrSensitivity = FALSE; // DSR sensitivity
            config.fTXContinueOnXoff = TRUE;// XOFF continues Tx
            config.fOutX = FALSE;       // No XON/XOFF out flow control
            config.fInX = FALSE;        // No XON/XOFF in flow control
            config.fErrorChar = FALSE;  // Disable error replacement
            config.fNull = FALSE;       // Disable null stripping
            config.fRtsControl = RTS_CONTROL_ENABLE;
            // RTS flow control
            config.fAbortOnError = FALSE;// Do not abort reads/writes
                                                                    on
            // error
            config.ByteSize = 8;          // Number of bits/byte, 4-8
            config.Parity = NOPARITY;     // 0-4=no,odd,even,mark,space
            config.StopBits = ONESTOPBIT; // 0,1,2 = 1, 1.5, 2

            if(!SetCommState(g_hComm, &config))
            {
                printf("erreur de config 2\n");
                return -1;
            }
            State = ComPort_Poll;
```

```
            }
            break;

        case ComPort_Poll:
            //On attend d'avoir des informations de quadri...
            GetCommState(g_hComm, &config);
            printf("baudrate = %d\n", config.BaudRate);

            if(!SetCommMask(g_hComm, EV_RXCHAR))
            {
                printf("F**k!");
            }

            if(WaitCommEvent(g_hComm, &dwCommEvent, NULL))
            {
                //Si on reçoit des info, on stock 50 caractères
                if(ReadFile(g_hComm, read, 50, &chRead, NULL))
                {
                    //On cherche le T qui marque le début de la trame
                    while(read[i]!='T')
                    {
                        i++;
                    }
                    //On récupère les 24 caractères suivant qui
                                                        représentent la trame
                    for (j = 0; j<24; j++)
                    {
                        tempRead[j] = read[i+j];
                    }
                    tempRead[19] = '\0'; //caractère de fin de chaine
                    //On récupère les valeurs de la chaine pour les stocker
                                                    dans des variables globales
                    sscanf(tempRead, "T%dR%dL%dA%dS", &g_angleTangageReel,
                    &g_angleRoulisReel, &g_angleLacetReel, &g_altitudeReel);
                }
                else
                {
                    printf("echec!");
                }
            }
            else
            {
                printf("Gros echec!");
            }

            State = ComPort_Send;
            break;

        case ComPort_Send:
            //On lui confirme qu'on a bien reçu les données.
                  // ToDo et voir si utile
            State = ComPort_End;
            break;
        }
    }
    printf("initialization ... Ok\n");
    return retour;
}
```

Réception des données

Le thread de réception des données scrute en permanence le port COM indiqué par l'utilisateur. Dès que des données passent sur celui-ci, il recherche une trame complète, en extrait les informations concernant les différents capteurs et les stocks dans des variables qui serviront aux autres threads. Voici le code expliqué :

```
void dataReceiving(void *data)
{
    DWORD dwCommEvent;
    char read[50] = {0};
    char chRead;
    char tempRead[20] = {0};
    int i = 0;
    int j = 0;

    while(!in.key[SDLK_ESCAPE] && !in.quit && g_run)
    {
        //On attend d'avoir des informations de quadri...
        if(!SetCommMask(g_hComm, EV_RXCHAR))
        {
                  //Si on ne parvient pas à se connecter
            printf("F**k");
        }
        if(WaitCommEvent(g_hComm, &dwCommEvent, NULL))
        {
                  //Si on reçoit des données
            if(ReadFile(g_hComm, read, 50, &chRead, NULL))
            {
                        //On en capture 50 caractrèes
                while(read[i]!='T')
                {
                              //On cherche le T pour connaitre le début de
                                                                    la trame
                    i++;
                }
                for (j = 0; j<24; j++)
                {
                              //On stocke les 24 caractères suivant pour
                                                          former la trame
                    tempRead[j] = read[i+j];
                }
                        //On récupère les valeurs de la trame pour les
                                          stocker dans des variables globales
                sscanf(tempRead, "T%dR%dL%dA%dS", &g_angleTangageReel,
                    &g_angleRoulisReel, &g_angleLacetReel, &g_altitudeReel);
            }
            else
            {
                printf("echec!");
            }
        }
        else
        {
            printf("Gros echec!");
        }
        i = 0;
    }
}
```

Envoi des données

Le thread d'envoi des données déduit des évènements clavier/souris les nouveaux angles à adopter pour le quadricoptère et les lui envoi. En effet, le tableau « in.key[] » est régulièrement mit à jour avec les évènements. Ainsi, ce thread n'a plus qu'à aller chercher dans celui-ci et calculer la nouvelle position. Les données sont ensuite envoyées sur la liaison série. Voici le code expliqué pour ce CU :

```
#define SENSITIVITY 0.5 //Facteur multiplicateur

/**
*     Structure contenant les évènements clavier et souris
**/
typedef struct
{
    char key[SDLK_LAST];
    int mouseX, mouseY;
    int mouseXRel, mouseYRel;
    char mouseButtons[8];
    char quit;
} Input;

//Déclaration globale de type Input
Input in;

void dataSending(void *data)
{
    int sendNOK = 1;
    int ret = 0;

    Uint32 debutBoucle = 0, finBoucle = 0, tempsEcoule = 0;
      Uint16 frameRate = 1000/200; //périodicité de l'envoi

    OVERLAPPED osWrite = {0};
    DWORD dwWritten;
    DWORD dwRes;
    BOOL fRes;

    //Gros while avec scrutation des touches/souris qui envois
    while(!in.key[SDLK_ESCAPE] && !in.quit && g_run)
    {
        debutBoucle = SDL_GetTicks();

        //envoi des valeurs
        char bufLettre = 0;
        char bufVal[3] = {0};
        char bufAltitudeReel[5] = {0};

        char bufComSnd[25] = {0};

            //Calcul du nouvel angle de roulis
        g_angleTangageSouris -= in.mouseXRel*SENSITIVITY;
        if(g_angleTangageSouris < 0)
        {
            g_angleTangageSouris = 360;
        }
        if(g_angleTangageSouris > 360)
        {
            g_angleTangageSouris = 0;
        }
```

```
g_angleRoulisSouris -= in.mouseYRel*SENSITIVITY;
if(g_angleRoulisSouris < 0)
{
    g_angleRoulisSouris = 360;
}
if(g_angleRoulisSouris > 360)
{
    g_angleRoulisSouris = 0;
}

in.mouseXRel = 0;
in.mouseYRel = 0;

    //calcul de la nouvelle altitude
if(in.key[SDLK_w])
{
    g_altitudeClavier+=1;
}
if(in.key[SDLK_s])
{
    g_altitudeClavier-=1;
}

    //calcul du nouvel angle de lacet
if(in.key[SDLK_a])
{
    g_angleLacetSouris+=1;
    if(g_angleLacetSouris >= 360)
    {
        g_angleLacetSouris = 0;
    }
}
if(in.key[SDLK_d])
{
    g_angleLacetSouris-=1;
    if(g_angleLacetSouris <= 0)
    {
        g_angleLacetSouris = 360;
    }
}
    //concaténation des angles dans la trame
sprintf(bufComSnd, "T%03dR%03dL%03dA%05dS",
                  g_angleTangageSouris%360, g_angleRoulisSouris%360,
                        g_angleLacetSouris%360, g_altitudeClavier);
                             //Concaténation de la trame à envoyer.
printf("%s\n", bufComSnd);

    //Envoi de la trame sur la liaison série
    if(!WriteFile(g_hComm, bufComSnd, 19, &dwWritten, &osWrite))
    {
          if(GetLastError()!=ERROR_IO_PENDING)
          {
                //WriteFile failed, but isn't delayed. Report error
                                                          and abort.
                fRes = FALSE;
          }
          else
          {
                //Write is pending
                dwRes = WaitForSingleObject(osWrite.hEvent,
```

```
                                                                  INFINITE);
                        switch(dwRes)
                        {
                        //OVERLAPPED structure's event has been signaled
                        case WAIT_OBJECT_0:
                              if(!GetOverlappedResult(g_hComm, &osWrite,
                                                      &dwWritten, FALSE))
                                    fRes = FALSE;
                              else
                              {
                                    fRes = TRUE;
                              }
                              break;
                        default:
                              fRes = FALSE;
                              break;
                        }
                  }
            }
            else
            {
                  fRes = TRUE;
            }

        finBoucle = SDL_GetTicks();
        tempsEcoule = finBoucle - debutBoucle;
        if(tempsEcoule<frameRate)
        {
            SDL_Delay(frameRate-tempsEcoule);
        }
    }
}
```

Affichage

Le thread affichage se charge de tout ce qui est communication avec l'utilisateur. Donc en plus d'imprimer à l'écran les valeurs de chaque angle du quadricoptère, il se charge des évènements clavier et souris. C'est donc lui qui va actualiser le tableau « in.key[] » utilisé lors de l'envoi des données. Voici le code expliqué pour cette partie :

```
#define HAUTEURFENETRE 600
#define LARGEURFENETRE 1000

char g_bufComRcv[20]= {0};

int g_angleTangageReel = 0;
int g_angleTangageSouris = 0;
int g_angleRoulisReel = 0;
int g_angleRoulisSouris = 0;
int g_angleLacetReel = 0;
int g_angleLacetSouris = 0;
int g_altitudeReel = 0;
int g_altitudeClavier = 0;

int g_run = 1;

/**
*     Fonction en charge de stocker dans un tableau les
*     touches du clavier et de la souris actives ainsi
*     que les mouvements de celle-ci
```

```
**/
void updateEvents()
{
    SDL_Event event;
    while(SDL_PollEvent(&event))
    {
        switch(event.type)
        {
            printf("dans le tableau\n");
        case SDL_KEYDOWN:
            in.key[event.key.keysym.sym]=1;
            break;
        case SDL_KEYUP:
            in.key[event.key.keysym.sym]=0;
            break;
        case SDL_MOUSEMOTION:
            in.mouseX=event.motion.x;
            in.mouseY=event.motion.y;
            in.mouseXRel=event.motion.xrel;
            in.mouseYRel=event.motion.yrel;
            break;
        case SDL_MOUSEBUTTONDOWN:
            in.mouseButtons[event.button.button] = 1;
            break;
        case SDL_MOUSEBUTTONUP:
            in.mouseButtons[event.button.button] = 0;
            break;
        case SDL_QUIT:
            in.quit = 1;
            break;
        default:
            break;
        }
    }
}

/**
*     Fonction permettant l'interaction avec l'utilisateur
**/
void displaying(void *data)
{
    SDL_Surface *ecran = NULL;
    Uint32 debutBoucle = 0, finBoucle = 0, tempsEcoule = 0;
    Uint16 frameRate = 1000/20;

      //Les surfaces correspondent aux images à afficher
    SDL_Surface *surfaceBGTangage = IMG_Load("res/TangageBG.png");
    SDL_Surface *surfaceAvTangage = IMG_Load("res/AvTangage.png");
    SDL_Surface *surfaceAvTangageSouris =
                                        IMG_Load("res/AvTangageSouris.png");
    SDL_Surface *surfaceBGLacet = IMG_Load("res/BoussoleBG.png");
    SDL_Surface *surfaceAvLacet = IMG_Load("res/AvBoussole.png");
    SDL_Surface *surfaceAvLacetSouris =
                                       IMG_Load("res/AvBoussoleSouris.png");
    SDL_Surface *surfaceBGRoulis = IMG_Load("res/RouliBG.png");
    SDL_Surface *surfaceAvRoulis = IMG_Load("res/AvRoulis.png");
    SDL_Surface *surfaceAvRoulisSouris =
                                         IMG_Load("res/AvRoulisSouris.png");

      //Les SDL_Rect correspondent aux emplacement des images
    SDL_Rect positionBGTangage;
```

```
    SDL_Rect positionAvTangage;
    SDL_Rect positionAvTangageSouris;
    SDL_Rect positionBGRoulis;
    SDL_Rect positionAvRoulis;
    SDL_Rect positionAvRoulisSouris;
    SDL_Rect positionBGLacet;
    SDL_Rect positionAvLacet;
    SDL_Rect positionAvLacetSouris;
    SDL_Rect positionAltitude;
    SDL_Rect positionAltitudeClavier;
    SDL_Rect positionSignal;
    SDL_Rect positionBatterie;

      //Initialisation de la position de chaque image
    positionBGTangage.x = 20;
    positionBGTangage.y = 20;

    positionAvTangage.x = 30;
    positionAvTangage.y = 30;

    positionAvTangageSouris.x = 30;
    positionAvTangageSouris.y = 30;

    positionBGRoulis.x = 200;
    positionBGRoulis.y = 20;

    positionAvRoulis.x = 210;
    positionAvRoulis.y = 30;

    positionAvRoulisSouris.x = 210;
    positionAvRoulisSouris.y = 30;

    positionBGLacet.x = 360;
    positionBGLacet.y = 20;

    positionAvLacet.x = 370;
    positionAvLacet.y = 30;

    positionAvLacetSouris.x = 370;
    positionAvLacetSouris.y = 30;

    positionAltitude.x = 550;
    positionAltitude.y = 80;

    positionAltitudeClavier.x = 550;
    positionAltitudeClavier.y = 110;

    positionBatterie.x = 30;
    positionBatterie.y = 90;

    positionSignal.x = 130;
    positionSignal.y = 90;

    // Initialisation de la SDL
    if(SDL_Init(SDL_INIT_VIDEO) < 0)
    {
        printf("Erreur lors de l'initialisation de la SDL : %s",
                                                    SDL_GetError());

        SDL_Quit();
        g_run = 0;
    }
```

```
    //Initialisation de SDL_ttf
    if(TTF_Init() == -1)
    {
        printf("Erreur lors de l'initialisation de SDL_ttf: %s\n",
                                                        TTF_GetError());
        exit(EXIT_FAILURE);
        g_run = 0;
    }
    // Création de la fenêtre
    ecran = SDL_SetVideoMode(LARGEURFENETRE, HAUTEURFENETRE, 32,
                                            SDL_HWSURFACE | SDL_DOUBLEBUF);

      //Chargement de la police d'écriture pour l'altitude
    TTF_Font *Font_g_altitude = NULL;
    Font_g_altitude = TTF_OpenFont("res/DS-DIGI.ttf", 40);
    SDL_Color couleurNoire = {0,0,0};
    SDL_Surface *surfaceAltitude;
    SDL_Surface *surfaceAltitudeClavier;

      //Masquage de la souris et celle-ci n'est plus limité aux bords de la
                                                                    fenêtre
    SDL_WM_GrabInput(SDL_GRAB_ON);
    SDL_ShowCursor(SDL_DISABLE);

    g_angleTangageSouris = g_angleTangageReel;
    g_angleRoulisSouris = g_angleRoulisReel;
    g_angleLacetSouris = g_angleLacetReel;
    g_altitudeClavier = g_altitudeReel;

    while (!in.key[SDLK_ESCAPE] && !in.quit && g_run)
    {
        debutBoucle = SDL_GetTicks();

            //actualisation des évènements clavier/souris
        updateEvents();
            //Remplissage du fond de la fenêtre
        SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 116, 124,
                                                                    121));
            //affichage des arrières plans des outils de mesure
        SDL_BlitSurface(surfaceBGTangage, NULL, ecran, &positionBGTangage);
        SDL_BlitSurface(surfaceBGRoulis, NULL, ecran, &positionBGRoulis);
        SDL_BlitSurface(surfaceBGLacet, NULL, ecran, &positionBGLacet);

            //Mise à jour de l'affichage du lacet désiré en fonction de son
                                                            nouvel angle
        SDL_Surface* rotatedAvLacetVoulue =
      rotozoomSurface(surfaceAvLacetSouris, g_angleLacetSouris%360, 1.0, 0);
        positionAvLacetSouris.x = 440-(rotatedAvLacetVoulue->w)/2;
        positionAvLacetSouris.y = 100-(rotatedAvLacetVoulue->w)/2;
        SDL_BlitSurface(rotatedAvLacetVoulue, NULL, ecran,
                                                &positionAvLacetSouris);
        SDL_FreeSurface(rotatedAvLacetVoulue);

            //Mise à jour de l'affichage du lacet réel en fonction de son
                                                            nouvel angle
        SDL_Surface* rotatedAvLacetReel = rotozoomSurface(surfaceAvLacet,
                                            g_angleLacetReel%360, 1.0, 0);
        positionAvLacet.x = 440-(rotatedAvLacetReel->w)/2;
        positionAvLacet.y = 100-(rotatedAvLacetReel->w)/2;
        SDL_BlitSurface(rotatedAvLacetReel, NULL, ecran, &positionAvLacet);
```

```
        SDL_FreeSurface(rotatedAvLacetReel);

            //Mise à jour de l'affichage du roulis désiré en fonction de
                                                              son nouvel angle
        SDL_Surface* rotatedAvRoulisVoulue =
  rotozoomSurface(surfaceAvRoulisSouris, g_angleRoulisSouris%360, 1.0, 0);
        positionAvRoulisSouris.x = 280-(rotatedAvRoulisVoulue->w)/2;
        positionAvRoulisSouris.y = 100-(rotatedAvRoulisVoulue->w)/2;
        SDL_BlitSurface(rotatedAvRoulisVoulue, NULL, ecran,
                                                  &positionAvRoulisSouris);
        SDL_FreeSurface(rotatedAvRoulisVoulue);

            //Mise à jour de l'affichage du roulis réel en fonction de son
                                                                  nouvel angle
        SDL_Surface* rotatedAvRoulisReel = rotozoomSurface(surfaceAvRoulis,
                                              g_angleRoulisReel%360, 1.0, 0);
        positionAvRoulis.x = 280-(rotatedAvRoulisReel->w)/2;
        positionAvRoulis.y = 100-(rotatedAvRoulisReel->w)/2;
        SDL_BlitSurface(rotatedAvRoulisReel, NULL, ecran,
                                                        &positionAvRoulis);
        SDL_FreeSurface(rotatedAvRoulisReel);

            //Mise à jour de l'affichage du tangage désiré en fonction de
                                                              son nouvel angle
        SDL_Surface *rotatedAvTangageVoulue =
rotozoomSurface(surfaceAvTangageSouris, g_angleTangageSouris%360, 1.0, 0);
        positionAvTangageSouris.x = 100-(rotatedAvTangageVoulue->w)/2;
        positionAvTangageSouris.y = 100-(rotatedAvTangageVoulue->w)/2;
        SDL_BlitSurface(rotatedAvTangageVoulue, NULL, ecran,
                                                   &positionAvTangageSouris);
        SDL_FreeSurface(rotatedAvTangageVoulue);

            //Mise à jour de l'affichage du tangage réel en fonction de son
                                                                  nouvel angle
        SDL_Surface *rotatedAvTangage = rotozoomSurface(surfaceAvTangage,
                                            g_angleTangageReel%360, 1.0, 0);
        positionAvTangage.x = 100-(rotatedAvTangage->w)/2;
        positionAvTangage.y = 100-(rotatedAvTangage->w)/2;
        SDL_BlitSurface(rotatedAvTangage, NULL, ecran, &positionAvTangage);
        SDL_FreeSurface(rotatedAvTangage);

        //Mise à jour de l'altitude réelle et voulue
        char chaineAltitudeReel[22] = {0};
        char chaineAltitudeClavier[22] = {0};
        char StringAltitudeClavier[18] = {'a', 'l', 't', 'i', 't', 'u',
                    'd', 'e', ' ', 'v', 'o', 'u', 'l', 'u', 'e', ':', ' '};
        char StringAltitudeReel[18] = {'A', 'l', 't', 'i', 't', 'u', 'd',
                        'e', ' ', 'r', 'e', 'e', 'l', 'l', 'e', ':', ' '};
        char bufAltitudeReel[4];
        char bufAltitudeClavier[4];

        strcat(chaineAltitudeReel, StringAltitudeReel);
        sprintf(bufAltitudeReel, "%d", g_altitudeReel);
        strcat(chaineAltitudeReel, bufAltitudeReel);
        surfaceAltitude = TTF_RenderText_Blended(Font_g_altitude,
                                        chaineAltitudeReel, couleurNoire);
        SDL_BlitSurface(surfaceAltitude, NULL, ecran, &positionAltitude);

        strcat(chaineAltitudeClavier, StringAltitudeClavier);
        sprintf(bufAltitudeClavier, "%d", g_altitudeClavier);
        strcat(chaineAltitudeClavier, bufAltitudeClavier);
```

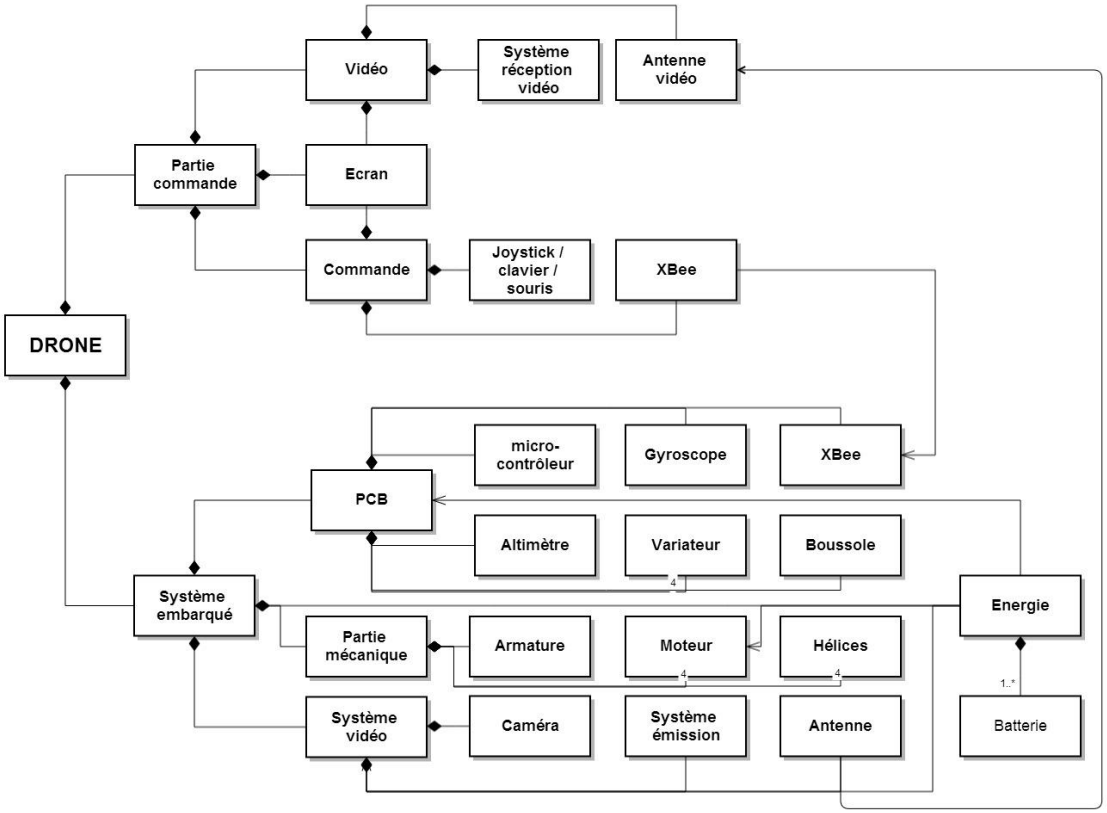
```
        surfaceAltitudeClavier = TTF_RenderText_Blended(Font_g_altitude,
                                        chaineAltitudeClavier, couleurNoire);
        SDL_BlitSurface(surfaceAltitudeClavier, NULL, ecran,
                                                &positionAltitudeClavier);

            //Mise à jour de l'écran
        SDL_Flip(ecran);
        finBoucle = SDL_GetTicks();
        tempsEcoule = finBoucle - debutBoucle;

        if(tempsEcoule<frameRate)
        {
            SDL_Delay(frameRate-tempsEcoule);
        }
    }

      //Libération de la police d'écriture et arrêt de la SDL
    TTF_CloseFont(Font_g_altitude);
    TTF_Quit();
    SDL_Quit();
}
```

Bilan du CU

Concernant le bilan de cette interface graphique, je pense la reprendre de 0 étant donné que le code est assez brouillon. De plus, je pense que le langage que j'ai utilisé n'est pas forcément le bon. En effet, j'ai décidé de faire celle-ci en C uniquement car il s'agissait du langage que je maîtrisais le mieux. Cependant, je pense la refaire en C++ pour améliorer l'allure général de l'interface et avoir un code plus fonctionnel. De plus, si j'intègre le leap motion comme j'aimerais le faire, le constructeur propose une librairie C++ et non C.

Résumé de l'organisation du drone

Comme on peut le voir sur le schéma ci-dessous, le drone est un grand ensemble de sous-systèmes. Ceux-ci sont donc sur deux plates-formes distinctes : la télécommande et le système embarqué :



Utilisation d'un système de versioning

Durant la première semaine de projets à l'IUT, il m'a paru utile d'utiliser un système de versioning. En effet, il m'est arrivé plusieurs fois de souhaiter revenir à un ancien code fonctionnant, cependant, ayant fait plusieurs modifications dessus, cela m'était bien souvent impossible simplement en effectuant des successions d'annulations.

Ma première solution, que j'utilisais alors depuis le début de ce projet, était de copier dans un répertoire réservé à cela, le projet ainsi que les sources. Je me retrouvais donc avec plusieurs versions de mon projet à différents stades et censées fonctionner. Cependant, au bout de quelques temps, il était difficile pour moi de m'y retrouver. En effet, étant donné que toutes les versions portaient le même nom à un numéro prêt, il était compliqué pour moi, de savoir qu'est-ce qu'il y avait de nouveau dans chaque version. Ayant bien tenté d'utiliser un fichier texte pour y décrire les nouveaux points, j'ai trouvé cela compliqué à tenir à jour alors qu'un système tout fait faisait cela bien mieux que moi : le versioning !

Pour utiliser le versioning, j'avais besoin d'un serveur SVN. Ne pouvant pas me permettre d'en faire un moi-même et en connaissant un fonctionnant à merveille, j'ai choisis de déposer mon projet sur github. Ce site permet à tout à chacun de créer des projets, et, à l'aide d'un client SVN (subversioning), s'y connecter pour mettre à jour ses sources. Après avoir créé mon projet sur ce site internet, il me fallait donc un client SVN. Ayant trouvé sur internet des cours permettant d'apprendre à se servir du client « git », j'ai choisis d'utiliser celui-ci. Git est un logiciel multiplateforme, utilisable en commande ou en graphique et me permettant de me connecter à mon répertoire sur github. Une fois celui-ci paramétré, j'ai pu envoyer une version de mon projet fonctionnant sur mon répertoire en ligne et je savais qu'à présent je pourrais à tout moment revenir en arrière. Au début (et toujours un petit peu aujourd'hui), l'utilisation de git fut relativement compliquée. En effet, ne connaissant pas par cœur les manipulations à effectuer, je me suis retrouvé quelques fois avec des effets non-escomptés. Cependant, je suis tout de même parvenu à m'en sortir à chaque fois.

Vous pouvez retrouver l'ensemble de mon projet à cette adresse :

https://github.com/SerialLooser/Quadrirotor

Conclusion

A la première moitié du projet

Cette première moitié de projet à, selon moi plutôt bien avancé. Surtout en ce qui concerne le gyroscope. Cependant, certains problèmes restent présents et pour le moment, je n'ai toujours pas la moindre idée de comment résoudre ceux-ci. Le principal étant le problème lié à l'I²C et les calculs de l'altimètre.

Je ne pense pas terminer ce projet dans les 100 heures imparties. Cependant, je commence à y voir plus claire sur comment parvenir à atteindre mon objectifs et les difficultés auxquelles je vais devoir faire face.

A la seconde moitié du projet

Cette seconde moitié du projet n'a malheureusement pas été aussi productive que la précédente. La première raison à cela fut le temps nécessaire au débogage du code lié à l'altimètre et la seconde, mon opération du genou qui m'a empêchée de venir durant une semaine et demie. Durant cette absence, j'en ai profité pour commander tout le matériel dont j'aurais besoin rapidement ainsi que la rédaction de ce compte rendu.

Je ne perds donc pas espoir concernant ce projet et pense bien le faire voler avant la fin de l'année scolaire. Certes celui-ci ne sera pas parfait mais cela sera suffisant pour me redonner un dernier coup de motivation pour aller plus loin.

Conclusion générale

Partant d'un projet personnel qui, de par la charge de travail qu'aurait demandé un autre projet à côté, était certainement voué à l'abandon, le fait de pouvoir poursuivre celui-ci durant ces 100 heures de projet m'a permis de me rassurer quant à la faisabilité de celui-ci.

Certes ce projet ne m'a pas permis de mettre en pratique les connaissances vues durant les cours de la licence pro, mais il m'a néanmoins permis de développer mes capacités de résolution de problèmes par moi-même et reprendre quelques points vus durant mon DUT Informatique. Il m'a également permis de découvrir de nombreuses notions telles que l'utilisation de l'OS ChibiOS, sa documentation et ses exemples.

De plus, je pense que ces projets nous permettent de voir ce que c'est que de travailler pendant des journées entières sur un système. Ceci pouvant éventuellement remettre en doute nos choix quant à la formation réalisée. En effet, si le fait de passer parfois plusieurs journées à la résolution de bugs nous dégoûte, il peut être opportun de penser à une reconversion étant donné que cela reste, malheureusement, la plus grande partie du travail d'un développeur. De mon point de vue personnel, celui-ci me confirme une nouvelle fois mon désir de travailler dans cette branche de l'informatique.

Annexe

Fichier Excel utilisée pour le dimensionnement des moteurs :

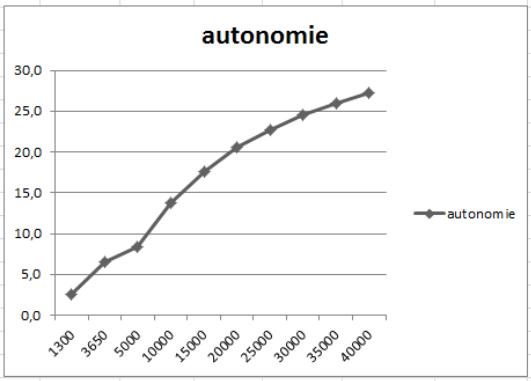
		Batterie mAh	1300	3650	5000	10000	15000	20000	25000	30000	35000	40000
		Poids (kg)	0,147	0,300	0,440	0,880	1,320	1,760	2,200	2,640	3,080	3,520
Engin	poid(k/g) sans batterie	1,7	1,85	2,00	2,14	2,58	3,02	3,46	3,90	4,34	4,78	5,22
	puissance/moteur (W)	85	92,35	100,00	107,00	129,00	151,00	173,00	195,00	217,00	239,00	261,00
Hélice	k (1,05<k<1,31)	1,2										
	L (pouces)	10										
	Pas (pouces)	4,5										
	rpm-maintien		7522	7724	7900	8408	8861	9272	9650	10000	10327	10635
Moteur	Tension batterie	14,8										
	kv min		508	522	534	568	599	627	652	676	698	719
	conso(r=80%) (A)	7,2	7,8	8,4	9,0	10,9	12,8	14,6	16,5	18,3	20,2	22,0
Autonomie	maintien(min)		2,5	6,5	8,3	13,8	17,6	20,5	22,8	24,6	26,0	27,2

Ce tableau permet de déterminer:
-le kv mini
-la consommation en maintien
-l'autonomie d'un quadricoptère en maintien

en fonction de :
la batterie, (capacité& nb cellules)
les pales
le poids à vide de l'appareil

Pour choisir son moteur, il faut choisir un kv> kv-mini qui permet aussi de prendre de l'altitude.

Il n'y a aucun coefficient de sécurité , c'est le minimum théorique donc prévoir des moteurs et contrôleurs un peu plus costaux.
Ne modifier que les cases bleues



L'autonomie calculée tiens seulement compte de la consommation des moteurs.

Pour toutes remarques guillaume.demond@imelavi.fr