

DRONE D'IMAGERIE ET DE TÉLÉMÉTRIE



Académie Aix-Marseille

Lycée Alphonse Benoît
Cours Victor Hugo,
84800 L'Isle-sur-la-Sorgue

GRIMOUD Yohan
MEDEIROS Vincent
THO Yann
VILLIELM Emmanuel

SOMMAIRE

Présentation générale.....	1
Introduction	1
Cahier des charges.....	2
Specification.....	3
I – EC : Mesurer (Géolocalisation, altitude et distance)	5
Présentation personnelle.....	5
Travail réalisé.....	6
Programme test sur Rpi2	6
Schéma Fritzing du câblage sur la Rpi2	7
Capture d'une trame.....	7
Suite du projet	8
II – EC : Incrustation vidéo	9
Présentation personnelle.....	9
Travail réalisé.....	10
Gantt	10
Bon de commande pour les composants de la carte d'incrustation personnelle	11
Tests sous Arduino et schéma de câblage	12
Carte d'incrustation (v1) – Schéma ISIS.....	13
Carte d'incrustation (v1) – Schéma ARES	14
Commande d'un émetteur et d'un récepteur 5.8GHz	15
Test du programme d'incrustation sous Raspberry Pi	16
Carte d'incrustation (v2) – Schéma ISIS.....	17
Carte d'incrustation (v2) – Schéma ARES	18
Tests du Quartz de la carte d'incrustation de Sparkfun et d'un oscillateur.....	19
Carte d'incrustation (v3) – Schéma ISIS.....	20
Carte d'incrustation (v3) – Schéma ARES	21
Ajout de caractères spéciaux dans la mémoire EEPROM de l'incrustateur MAX7456..	22
Tests de l'émetteur et du récepteur 5.8GHz	23
IBD de ma carte d'incrustation personnelle	24
Simulation de la fabrication de la carte d'incrustation.....	25
Test de la carte d'incrustation Sparkfun avec les caractères modifiés	26
Partie Physique.....	27
Analyse d'une trame SPI.....	27

Conclusion.....	28
Annexes	29
Schéma ISIS carte d'incrustation personnelle v1	29
Schéma ARES carte d'incrustation personnelle v1 (top / bottom)	30
Schéma ISIS carte d'incrustation personnelle v2.....	31
Schéma ARES carte d'incrustation personnelle v2 (top / bottom)	32
Schéma ISIS carte d'incrustation personnelle v3.....	33
Schéma ARES carte d'incrustation personnelle v3 (top / bottom)	34
Tests du Quartz de la carte d'incrustation de Sparkfun et d'un oscillateur.....	35
Ajout de caractères spéciaux dans la mémoire EEPROM de l'incrustateur MAX7456 ..	35
III – IR : Développement du logiciel EDD	36
Présentation personnelle.....	36
Travail réalisé.....	37
Vue d'ensemble du logiciel EDD.....	37
Conception du multithread	38
Mise en place de la zone de mémoire partagée	39
Sauvegarde des mesures dans un fichier local	41
Communication entre processus.....	43
Gestion de la caméra	43
Amélioration supplémentaire	45
Cross-Compilation	48
Intégration des parties EDD	49
Simulation du logiciel EDD	49
Optimisation du système EDD envisageable	53
Analyse physique I2C	54
Définition du bus I2C	54
Caractéristique.....	54
Analyse personnelle	55
Conclusion.....	57
Annexes	58
Diagramme de Classe « Class_EDD » :	58
Diagramme de séquence : « SD_Thread »	60
Diagramme de séquence : « SD_C_Save »	61
Diagramme de séquence : « SD_FinMission »	62
IV – IR : Historisation et supervision.....	63

Présentation personnelle.....	63
Travail réalisé.....	66
Présentation de la base de données (BDD)	66
Gestion de la base de données.....	68
Présentation du programme de la station au sol (GCS)	69
Communication avec la base de données.....	72
Communication avec le drone (EDD).....	73
Méthodes utilisées.....	74
Communication avec l'entreprise à distance.....	76
Interface utilisateur.....	77
Analyse physique	78
Transmission via le module 433MHz	78
Conclusion.....	79
Annexes.....	80
Diagramme des exigences du système	80

PRÉSENTATION GÉNÉRALE

Introduction

Il est maintenant devenu courant d'utiliser un drone pour effectuer des prises de vue ou des séquences filmées de maisons ou de paysages.

Un besoin supplémentaire apparaît dans l'industrie, celui de pouvoir effectuer des mesures afin de surveiller, détecter, maintenir des ouvrages hauts et difficiles d'accès de manière préventive.

Les architectes-maitres d'œuvre ont besoin lors des phases de construction ou de maintenance d'observer les murs et toitures des bâtiments, vérifier les montages d' huisseries, de faîtères, rechercher des entrées d'eau etc...



L'accès à ces zones n'est pas toujours possible pour les grands ouvrages. L'utilisation d'un drone d'observation équipé d'une transmission vidéo, ainsi que d'une batterie de capteurs des grandeurs physiques est alors une solution économique et efficace.

En cas de problème détecté, sa localisation dans les trois dimensions et sa mémorisation permettent une intervention de maintenance précise et rapide.

Ce système nécessite deux personnes à la station au sol (GCS). Le pilote s'occupe du pilotage à vue du drone. Un technicien (co-pilote) à ses côtés supervise la prise d'images, la télémétrie et la communication avec l'entreprise.

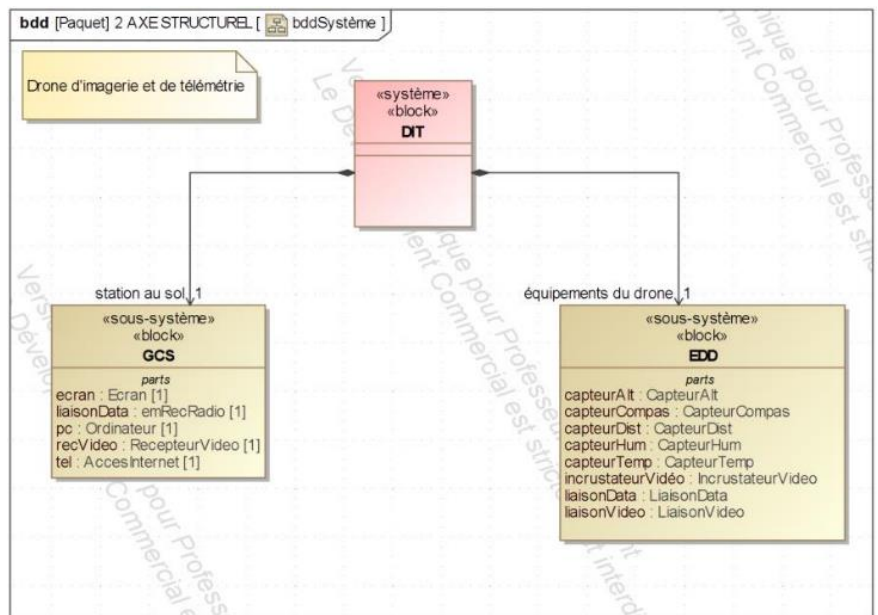
Cahier des charges

Le système comprend deux parties :

- Les équipements du drone (EDD)



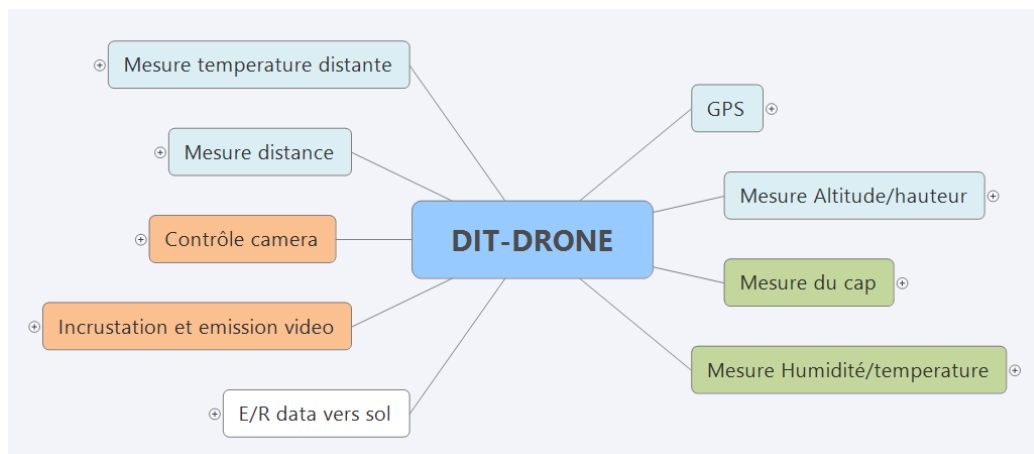
- La station de contrôle au sol (GCS)



Un vol dure entre 15 et 30 minutes.

La mesure de température sans contact ainsi que l'humidité des structures amènent le pilote à positionner le drone très près des obstacles. En raison de la taille de certains ouvrages, une mesure de distance d'objet est indispensable.

Le drone sera équipé d'un capteur GPS, d'un altimètre, d'un capteur d'humidité, de température distante, de capteur de distance d'obstacle.



Le co-pilote (technicien) envoie au drone des ordres pour prendre des photos ou des séquences vidéo qui seront stockées dans la carte mémoire de l'appareil photo.

Sur un ordre de la GCS, le drone effectue des mesures, les transfère à la station au sol, les sauve également dans sa mémoire en cas de difficultés de communication.

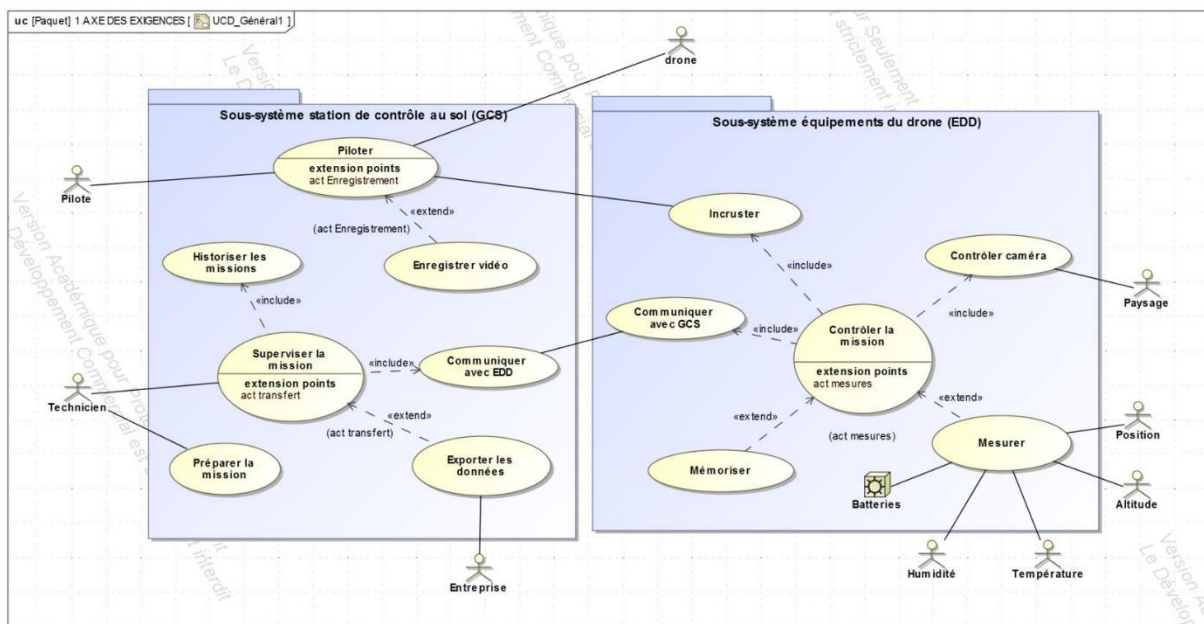
Le drone émet l'image de la caméra vers un écran de la station au sol avec la possibilité d'incruster des paramètres (texte, mesures, etc.) au choix du pilote.

3



Une liaison Internet permet de transmettre les données vers le site de l'entreprise.

Les besoins du système sont représentés par le diagramme des cas d'utilisation suivant :



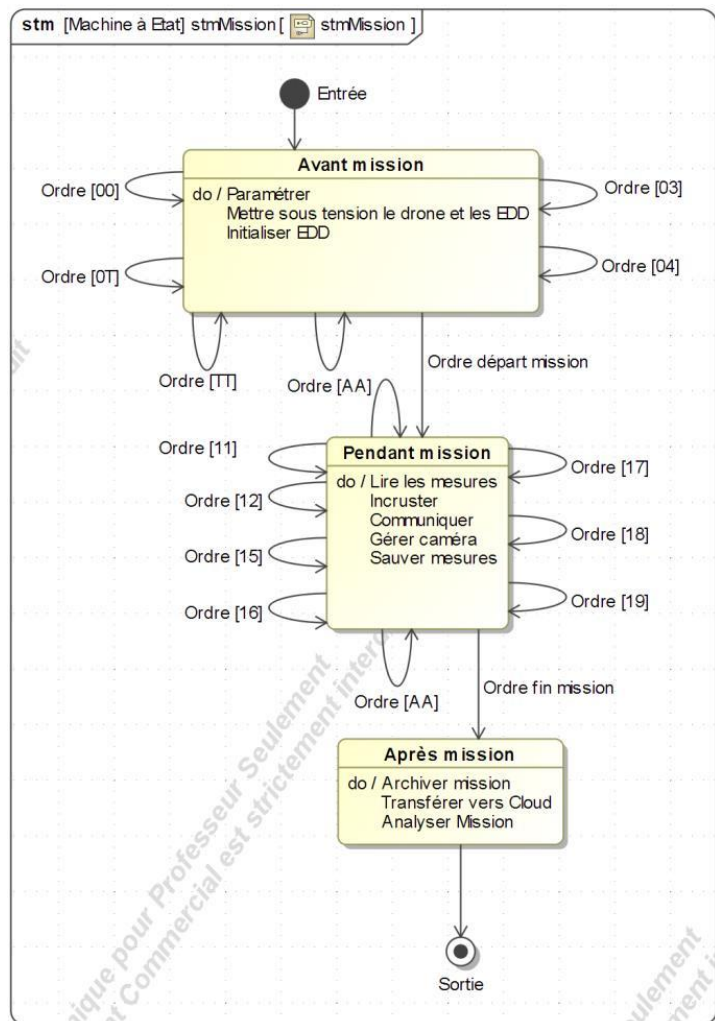
Durant son fonctionnement, le système peut se trouver dans un des trois états suivants :

- Préparation : état « avant mission »
- Mission de vol : état « pendant mission »
- Récupération et exportation des données de mission : état « après mission »

Pour chaque état, un protocole de communication est établi.

Les fonctions (ordres) de ce protocole sont accessibles selon l'état du système.

Le diagramme d'états ci-contre permet d'identifier les ordres disponibles en fonction de l'état du système.



I – EC : MESURER (GEOLOCALISATION, ALTITUDE ET DISTANCE)

Présentation personnelle

Étudiant 3	Liste des fonctions assurées par l'étudiant	Installation : Driver de bus Mise en œuvre : Choix des capteurs Configuration : Initialiser les capteurs Réalisation : Carte fille de test intégrant les différents capteurs, classes C++ de gestion des bus et des capteurs. Documentation : Elaboration partielle du document « en cas d'anomalie de fonctionnement »
EC	EDD : UC Mesurer (Géolocalisation, altitude et distance) IR	

Je suis dans l'équipe 1 du projet, elle est composée de Vincent Medeiros (EC), Emmanuel Williém (IR) et Yann Tho (IR).

Ma partie du projet est : UC Mesurer (Géolocalisation, altitude et distance).

Pour réaliser ma partie du projet il faut que je mette en marche un capteur de distance, un capteur d'altitude ainsi qu'un capteur de géolocalisation pour que l'étudiant 4 puisse stocker en mémoire les informations des capteurs et les afficher sur un écran en collaboration avec le travail de l'étudiant 1.

Voici le capteur de distance que j'ai à ma disposition.



C'est un capteur à ultrason qui peut être alimenté en 3.3V - 5V ce qui convient avec la Raspberry Pi. Il peut mesurer une distance assez précisément de 2 cm à environ 300 cm, il coûte entre 10 et 15 € à l'unité.

Travail réalisé

Programme test sur Rpi2

Mr HORTOLLAND a trouvé un programme source sous Arduino, il fallait donc appliquer des changements à ce programme pour qu'il soit compatible avec Rpi2.

Voilà le programme test qui fonctionne sous Rpi2 pour le capteur de distance Ultrasonic Ranger v1.0 qui permet d'afficher la distance entre le capteur et une surface.

```
#include <stdlib.h>
#include <wiringPi.h>

#define TRUE 1
#define SIG 5

void setup() {
    wiringPiSetup();
    pinMode(SIG, OUTPUT);

    //SIG pin must start LOW
    digitalWrite(SIG, LOW);
    delay(30);
}

int getCM() {
    //Send trig pulse
    digitalWrite(SIG, HIGH);
    delayMicroseconds(20);
    digitalWrite(SIG, LOW);

    pinMode(SIG, INPUT);

    //Wait for echo start
    while(digitalRead(SIG) == LOW);

    //Wait for echo end
    long startTime = micros();
    while(digitalRead(SIG) == HIGH);
    long travelTime = micros() - startTime;

    //Get distance in cm
    int distance = travelTime / 58,8;

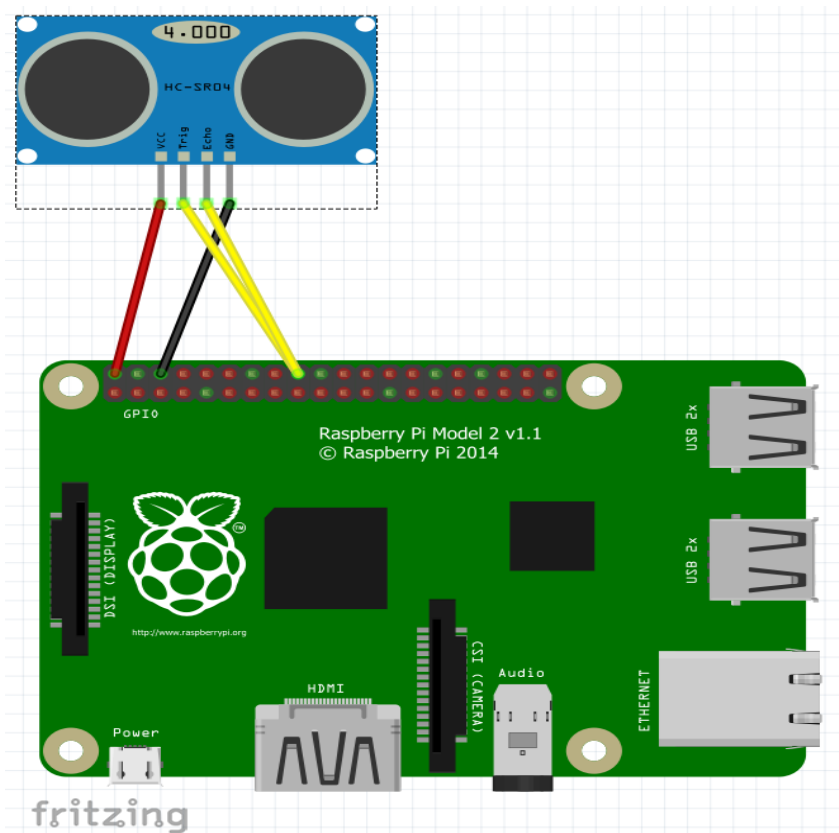
    return distance;
}

int main(void) {
    setup();

    while(printf("Distance: %dcm\n", getCM()));

    return 0;
}
```

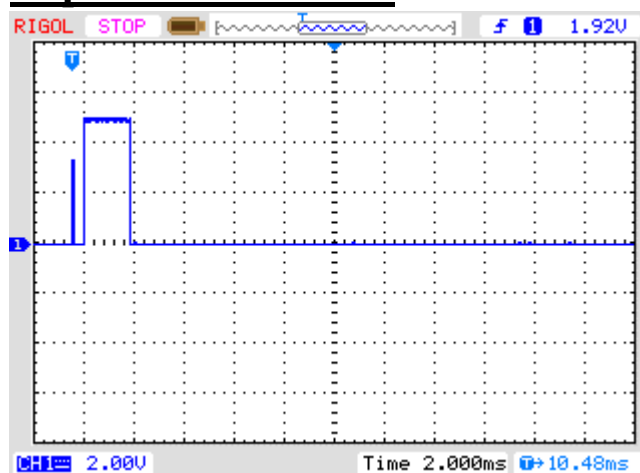
Schéma Fritzing du câblage sur la Rpi2



Ce schéma représente les connexions entre le capteur de distance et la carte, ici le capteur n'est pas le bon mais celui qui s'en rapproche le plus car on ne trouve pas de schéma Fritzing sur le capteur que j'ai à disposition et je n'ai pas trouvé de moyen de changer le nom des broches sur le logiciel.

La liaison des masses est représentée par le fil noir, l'alimentation par le rouge et la liaison de l'envoi du programme par les fils jaunes.

Capture d'une trame



On peut voir ci-contre la réception d'une trame sur un oscilloscope. À l'aide de cette trame on peut calculer la distance qu'il y avait entre le capteur et la surface exploitée en utilisant la formule :

$$V = d/t$$

$$d'où d = V \cdot t$$

On sait que la vitesse du son est de 340 m/s. Ici le temps est de 2 ms.

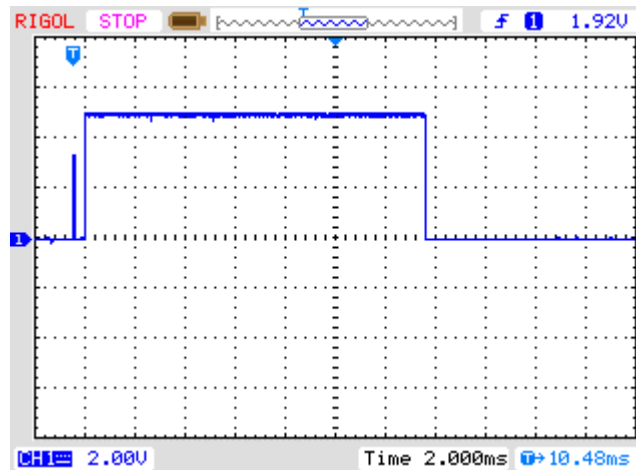
On calcul : $d = (2 \cdot 10^{-3}) \times 340 = 0,68\text{m}$

On doit diviser le résultat par 2 car l'onde fait un aller – retour (2 fois la distance)

$0,68\text{m}/2 = 0,34\text{m}$

Sur cette trame on sait que le capteur était à une distance de 0,34 m d'une surface.

Un autre exemple :



Ici le temps est de 12,8 ms

On calcul : $d = (12,8 \cdot 10^{-3}) \times 340 = 4,35\text{m}$

$4,35\text{m}/2 = 2,175\text{m}$.

Suite du projet

Pour les semaines qui suivent il faut que j'adapte le programme du capteur de distance avec une nouvelle librairie (bcm2835).

Il faut que je complète le diagramme de bloc des capteurs et que je m'occupe des 2 autres capteurs que je dois mettre en œuvre.

II – EC : INCRUSTATION VIDEO

Présentation personnelle

Je fais partie de l'équipe 1 du projet composée de 4 étudiants (2 EC et 2 IR).

Mon rôle dans le projet est de faire afficher, à l'aide d'un incrustateur, les informations relatives aux différents capteurs, disposés sur le drone, sur un écran de contrôle mis à disposition de l'équipe au sol.

Cette partie du projet sera donc liée avec le travail de Yohan Grimoud (capteurs d'altitude, de distance et géo localisation), Thomas Bonnefille (capteur de température et d'humidité), et Yann Tho (stockage des données).

L'incrustation des données recueillies par les capteurs se fera à l'aide d'une carte comportant une puce d'affichage à l'écran MAX7456. Cette carte n'étant plus en vente, il faudra en réaliser une nouvelle lors de la deuxième partie du projet mais remplir un bon de commande en début de projet pour ne pas avoir de contraintes au niveau du temps de livraison.

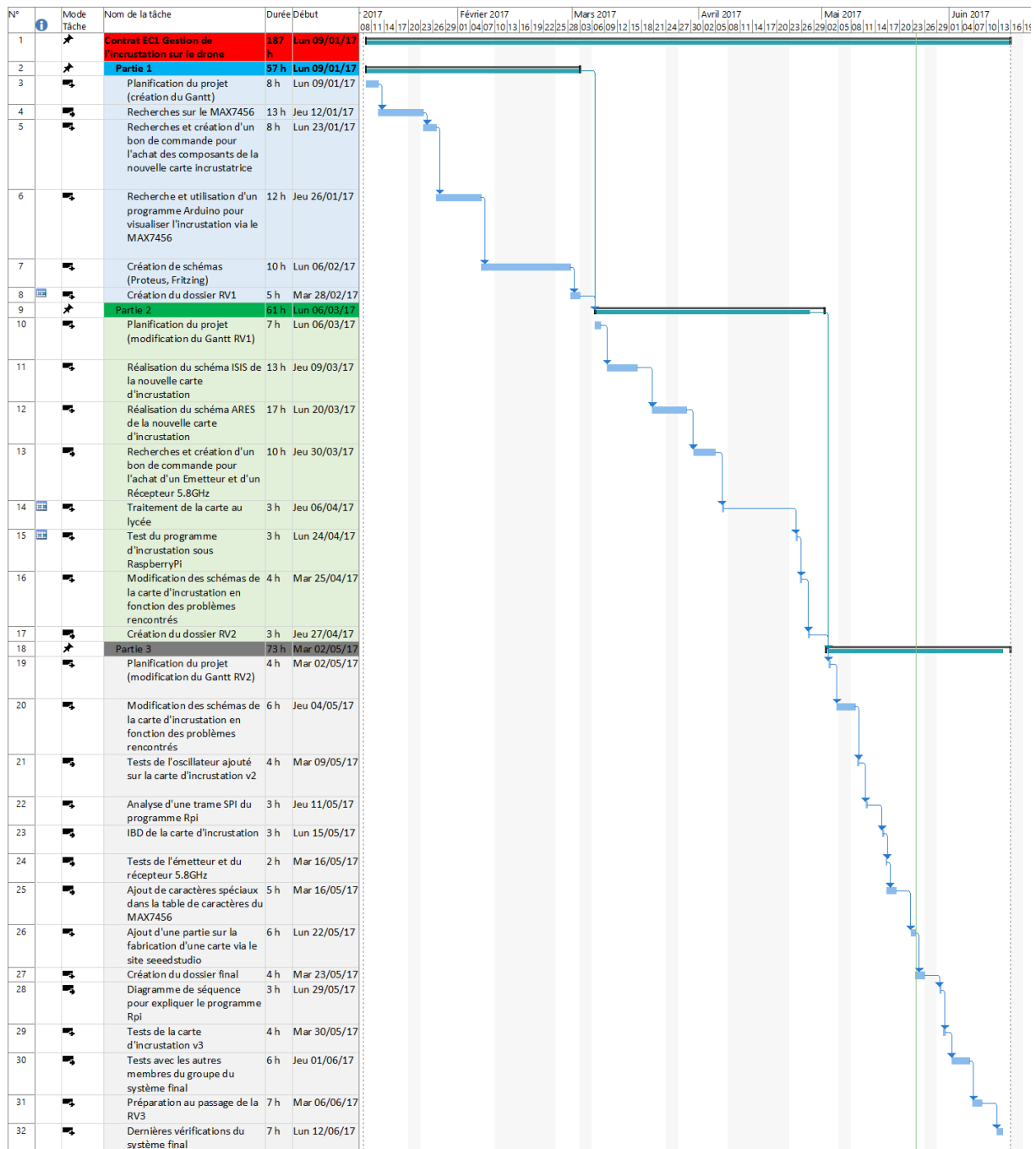
La programmation au niveau de cette carte se fera à l'aide d'une Raspberry Pi qui communiquera en SPI avec la carte d'incrustation disposant du MAX7456.

Étudiant 1	Liste des fonctions assurées par l'étudiant :	
EC	EDD : UC Incruster	<p><i>Installation</i> : Driver du bus.</p> <p><i>Mise en œuvre</i> : Communication avec l'incrustateur vidéo.</p> <p><i>Configuration</i> : Mettre à jour la table des caractères de l'incrustateur pour intégrer les caractères nécessaires aux unités des mesures à incruster.</p> <p><i>Réalisation</i> : Carte fille de test intégrant l'incrustateur (en liaison étroite avec l'étudiant 3), Développement d'une classe C++ de gestion du bus et de l'incrustateur.</p> <p><i>Documentation</i> : Elaboration partielle du document « en cas d'anomalie de fonctionnement »</p>

Travail réalisé

Gantt

Voici le planning des tâches que j'ai accomplies lors de ce projet, il a été complété et mis à jour au cours des 3 parties du projet :



Bon de commande pour les composants de la carte d'incrustation personnelle

Le début du projet a consisté pour ma part à la répartition des tâches via la création d'un Gantt, lequel allait être mis à jour et complété au cours du projet, j'ai ensuite effectué des recherches sur la carte d'incrustation de chez Sparkfun comportant le MAX7456. Ces recherches ont abouti en un bon de commande comportant tous les composants nécessaires à la création d'une carte d'incrustation personnelle qui sera liée au travail de l'étudiant 8 EC : Thomas Bonnefille.

Voici la liste des composants présents dans le bon de commande :

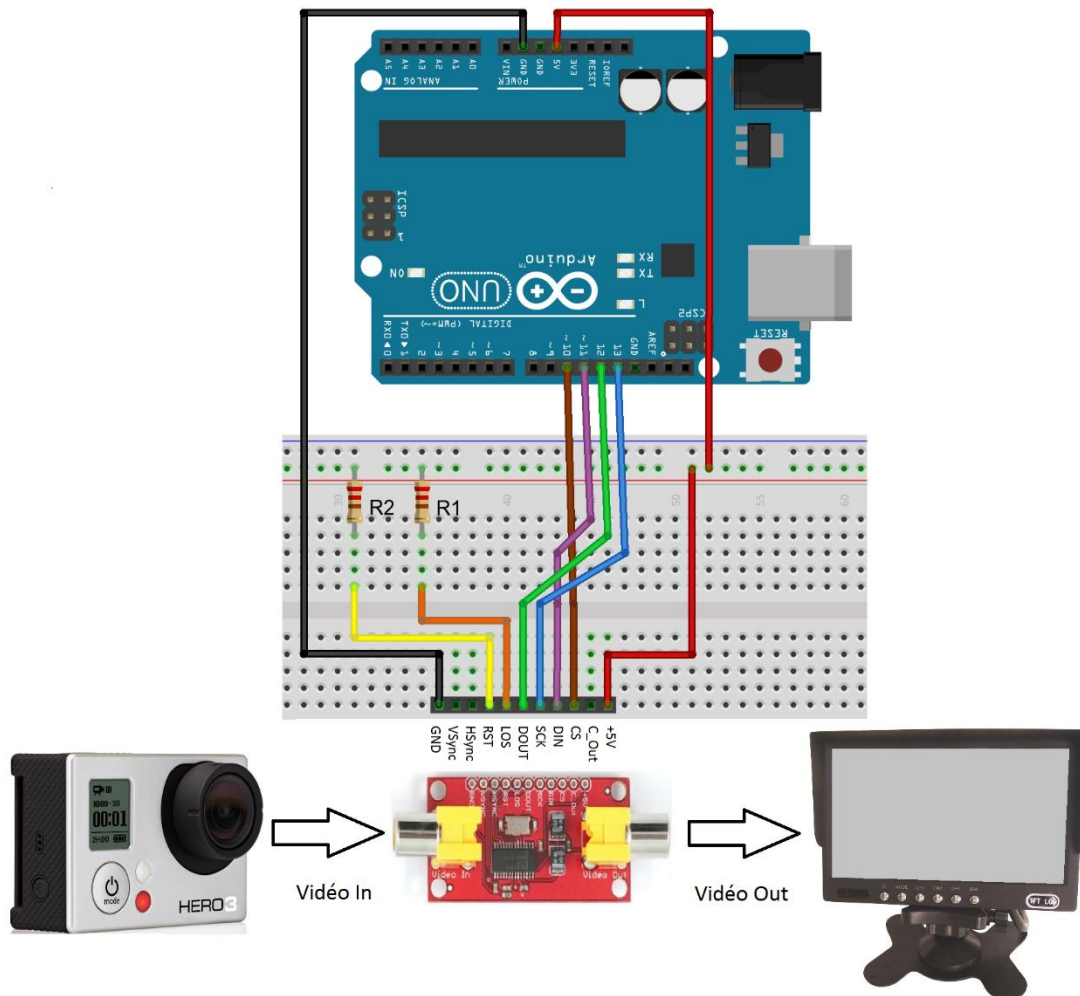
Désignation	réf. fournisseur	Qté	PUHT	PTHT
Incrustateur MAX7456	732-6970	2	17,91	35,82
Quartz CMS 27MHz	693-8917	2	0,8	1,6
Condensateur CMS 22pF	698-2942	5	0,346	1,73
Condensateur CMS 100nF	500-2376	10	0,298	2,98
Résistance CMS 75Ω	721-9321	5	0,056	0,28
Prise RCA Femelle	868-834	1	10,91	10,91
Condensateur polarisé 47uF	648-0597	10	0,278	2,78
Transistor 2N7002 CMS 3 broches	753-3134	100	0,047	4,7
Résistance 10kΩ CMS boîtier 1206	740-9110	50	0,02	1

Le coût de ce bon de commande est de 61.80€ HT, soit 74.16€ TTC.

Tests sous Arduino et schéma de câblage

Suite à cela, j'ai fait des recherches concernant la programmation du MAX7456, avec l'aide de M. Hortolland nous avons opté pour un programme Arduino simple, disponible sur le site f5mna.free.fr, que j'ai ensuite mis en œuvre avec la carte d'incrustation de chez Sparkfun.

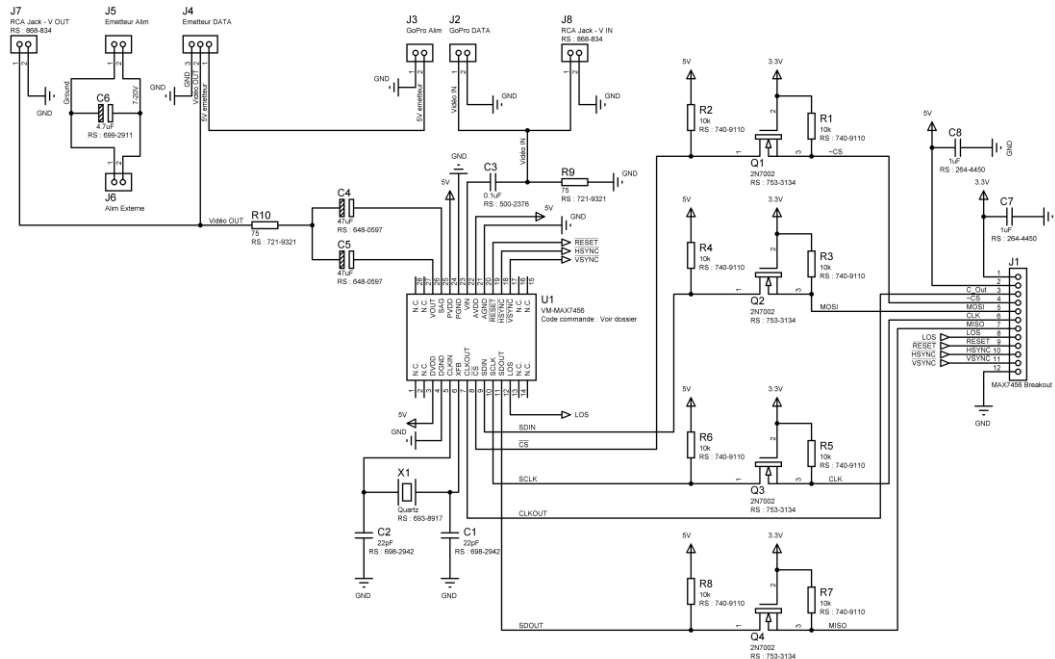
Je me suis ensuite attelé à la réalisation d'un schéma de prototypage rapide avec le logiciel Fritzing pour retranscrire les câblages des tests réalisés.



La fin de la première partie du projet a été consacrée à la création du dossier et du diaporama pour la première Revue de projet.

Carte d'incrustation (v1) – Schéma ISIS

Le début de la 2^{ème} partie du projet a consisté pour ma part à la répartition des tâches via la modification du Gantt utilisé lors de la 1^{ère} revue de projet, je me suis ensuite attelé à la réalisation de la nouvelle carte d'incrustation, j'ai donc réalisé un schéma structurel ISIS de cette future carte :



	TITLE Projet Drone Carte Incrustatrice	DATE: 06/04/17
		PAGE: 1/1

A noter qu'une version plus lisible de ce schéma sera disponible à la page 29.

Ce schéma ISIS a dû tenir compte des différentes spécifications de cette nouvelle carte :

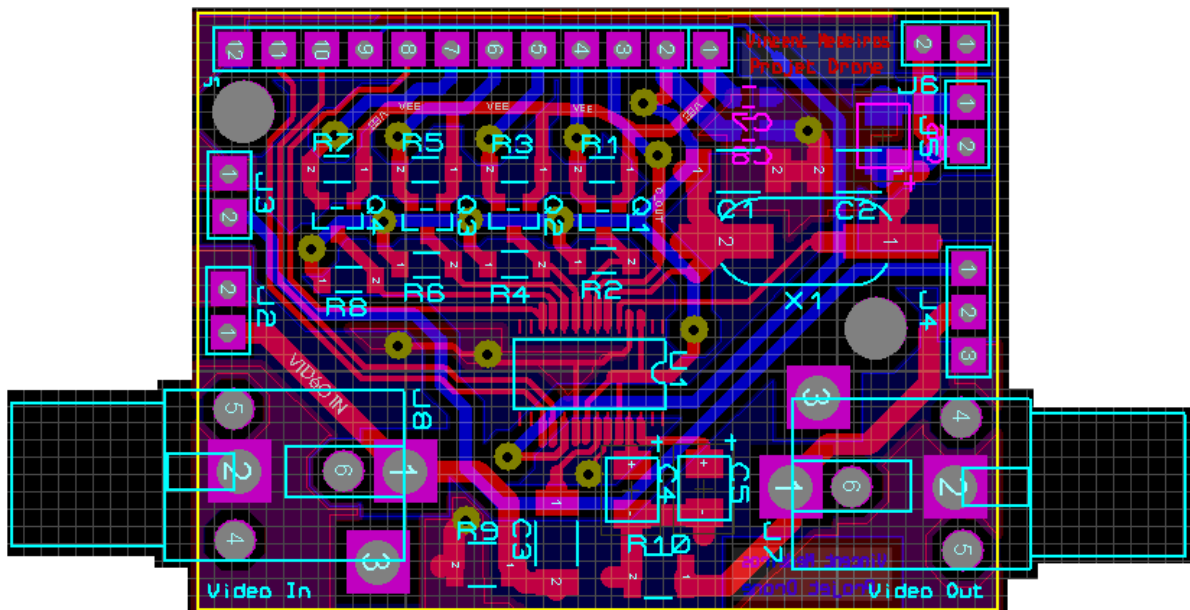
- Fonctionnement sous Arduino et Raspberry Pi ;
- Garder le même connecteur 11 broches (+ ou -) ;
- Respecter les mêmes dimensions (+ ou -) ;
- Ajouter aux connecteurs RCA Jack un moyen de se connecter à la GoProHero3+ et l'émetteur en soudant les fils ;
- Permettre l'alimentation de l'émetteur et de la GoProHero3+ directement sur la carte (Relier la batterie à la carte qui sera reliée à l'émetteur et à la GoProHero3+).

Carte d'incrustation (v1) – Schéma ARES

Dans la continuité du schéma ISIS, j'ai réalisé le schéma ARES qui nous a permis de lancer la création de la carte, il a fallu rechercher les empreintes des différents composants utilisés sur le schéma ISIS pour pouvoir obtenir les bonnes dimensions des composants.

Il a été convenu que ma carte serait connectée au-dessus de la carte de l'étudiant 8 EC : Thomas Bonnefille. Sa carte disposera d'un connecteur 12 broches femelles reliées au GPIO de la Raspberry Pi pour assurer l'alimentation et la communication SPI nécessaire à ma carte.

Voici le typon réalisé sous ARES de la nouvelle carte d'incrustation :



A noter qu'une version « top » et « bottom » de ce schéma sera disponible à la page 30.

La création de cette carte a été réalisée par M. Hortolland, au lycée, avec le matériel disponible.

Commande d'un émetteur et d'un récepteur 5.8GHz

La suite fut la recherche et la création d'un bon de commande pour l'achat d'un émetteur et d'un récepteur 5,8GHz capable de se câbler facilement sur la carte d'incrustation.

L'émetteur servant à envoyer les données de l'incrustation depuis le drone vers la station au sol munie d'un récepteur servant à recueillir ces informations et à les afficher sur l'écran de contrôle.

Ces recherches ont abouti à la rédaction d'un bon de commande, comportant un émetteur et un récepteur compatibles. La commande a été réalisée sur le site : <http://www.studiosport.fr/>

Voici un tableau comparatif des différents choix d'émetteurs et de récepteurs 5.8GHz disponibles :

Tableau comparatifs des différents Émetteurs et Récepteurs 5,8GHz
Émetteurs

Choix	Site internet	Nom du produit	Réf. Produit	Prix	Nb° canaux	Alimentation	P émission	Dimensions	Poids	Antenne fournie ?	Distance maxi	Connectique Carte -> Émetteur
E1		Émetteur 5.8GHz 200mW 40 canaux - sortie 5V	AR0030861	35,00 €	40	6 - 28V	200mW	56mm * 27mm * 16mm	16g	Oui		Fils dénudés vers émetteur
E2		Émetteur Raceband 25mW 5.8 GHz ImmersionRC	TX_5G8_25_RB	29,00 €	15	7 - 25V	25mW			Oui		Fils dénudés vers émetteur
E3	http://www.studiosport.fr	Émetteur AV 5.8GHz 25mW SGV - sortie 5V - 32 canaux	FLY-SGV-TX-58025-5V-32CH	45,00 €	32	6 - 16V	25mW	45mm * 31mm * 15mm	19g	Oui	200-300m	
E4		Émetteur Mini 25mW 5.8GHz 40CH Raceband	FX799T	19,00 €	40	7 - 20V	25mW	31mm * 22mm * 10mm	7,5g	Oui		Fils dénudés vers émetteur
E5		Émetteur Mini Raceband 40CH 200mW 5.8GHz	AR0030450	35,00 €	40	7 - 25V	200mW	29mm * 20mm * 8mm	14g	Oui		Fils dénudés vers émetteur
E6		Émetteur AV 5.8GHz 1200mW SGV	TX-5812-1200mW-SGV-TX5	79,00 €	8	6,5 - 15V	1200mW	68mm * 35,2mm * 27,5mm	42g	Oui	600-1000m	Fils dénudés vers émetteur

Récepteurs

Choix	Site internet	Nom du produit	Réf. Produit	Prix	Nb° canaux	Alimentation	Dimensions	Poids	Antenne fournie ?	Distance maxi	Connectique Récepteur -> Écran
R1		Récepteur 5.8GHz miniature - 32 canaux	RX-SGV-MINIATURE-RC306	29,00 €	32	4,5 - 16V	55mm * 40mm * 15mm		Oui		Jack vers RCA Audio/Video femelle
R2	http://www.studiosport.fr	Récepteur 5.8GHz miniature	RC305-MINI-SGV	19,90 €	8	Secteur 220V 8 - 12V	60mm * 53mm * 12mm	100g	Oui		Jack vers RCA Audio/Video femelle
R3		Récepteur vidéo 5.8GHz 32 canaux	SKY-32S	59,90 €	32	Secteur 220V 12V		80g	Oui		Jack vers RCA Audio/Video femelle

Le choix de l'émetteur s'est porté sur le choix E4 disponible sur le site [studiosport.fr](http://www.studiosport.fr).

Cet émetteur est compatible avec les restrictions de notre projet, il est abordable, disponible dans un délai raisonnable, et ne dispose pas d'une structure (poids et dimensions) handicapant le drone.

De plus, le choix de cet émetteur s'est fait dans le respect d'une loi en vigueur concernant les émetteurs 5,8GHz :

"La réglementation française impose une puissance max de 25mW pour l'émission d'un signal analogique dans la bande de fréquence des 5,8 GHz"

Le choix du récepteur s'est effectué avec les mêmes critères, bien que la structure de celui-ci ne pose aucun problème du fait de son positionnement au sol.

Le récepteur choisi pour notre projet est notre choix R2, il est abordable et peut directement être branché au secteur via l'adaptateur fourni.

Le choix de l'émetteur a dû se faire dans le respect de la réglementation française qui impose une puissance maximale de 25mW pour l'émission d'un signal analogique dans la bande de fréquence des 5.8GHz. Le coût de cette commande est de 38.90€.

L'émetteur et le récepteur choisis pour notre projet sont compatibles, ils utiliseront la bande B Boscam des fréquences 5.8GHz.

(Lien vers réglementation française : <https://fpv-spirit.com/reglementation-francaise-drone-fpv/>)

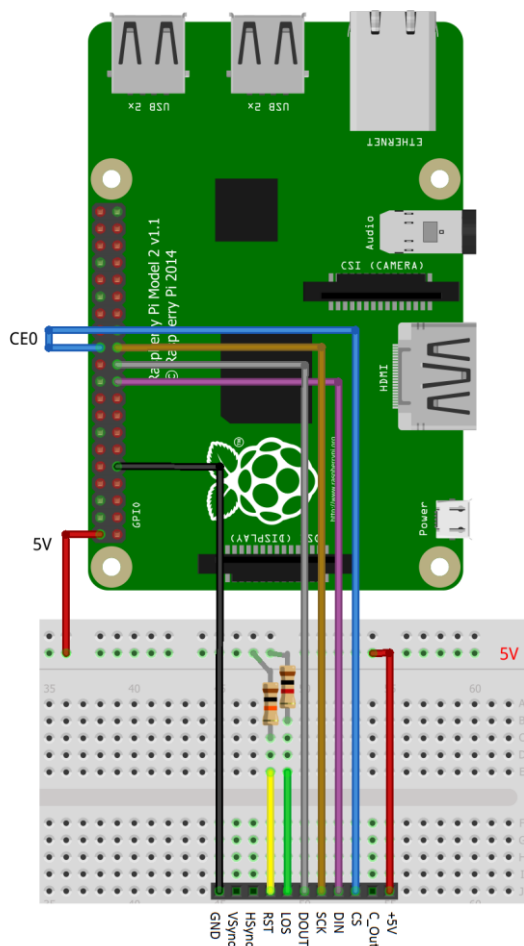
Les dernières lignes de cette page indiquent la réglementation sur la puissance d'émission vidéo).

Test du programme d'incrustation sous Raspberry Pi

Mon travail à ensuite consisté en des tests d'un programme d'incrustation, semblable à celui testé lors de la 1^{ère} partie du projet sous Arduino mais converti par M. Hortolland pour fonctionner sous Raspberry Pi grâce notamment à la librairie *bcm2835* pour la liaison SPI. Ces tests ont décelé des problèmes sur la carte d'incrustation personnelle, mais ont aussi permis d'éliminer des éléments censés rendre la carte plus polyvalente qui au final n'étaient pas nécessaires.

Ces tests ont donc permis de déceler des problèmes, avec notamment des éléments à ajouter ou à supprimer, ce qui a donc conduit à la modification des schémas ISIS et ARES de la carte d'incrustation personnelle pour concevoir une deuxième version de la carte au lycée, avant la version finale sous-traitée par un site spécialisé.

Voici le schéma de prototypage rapide réalisé sous Fritzing de ces tests :



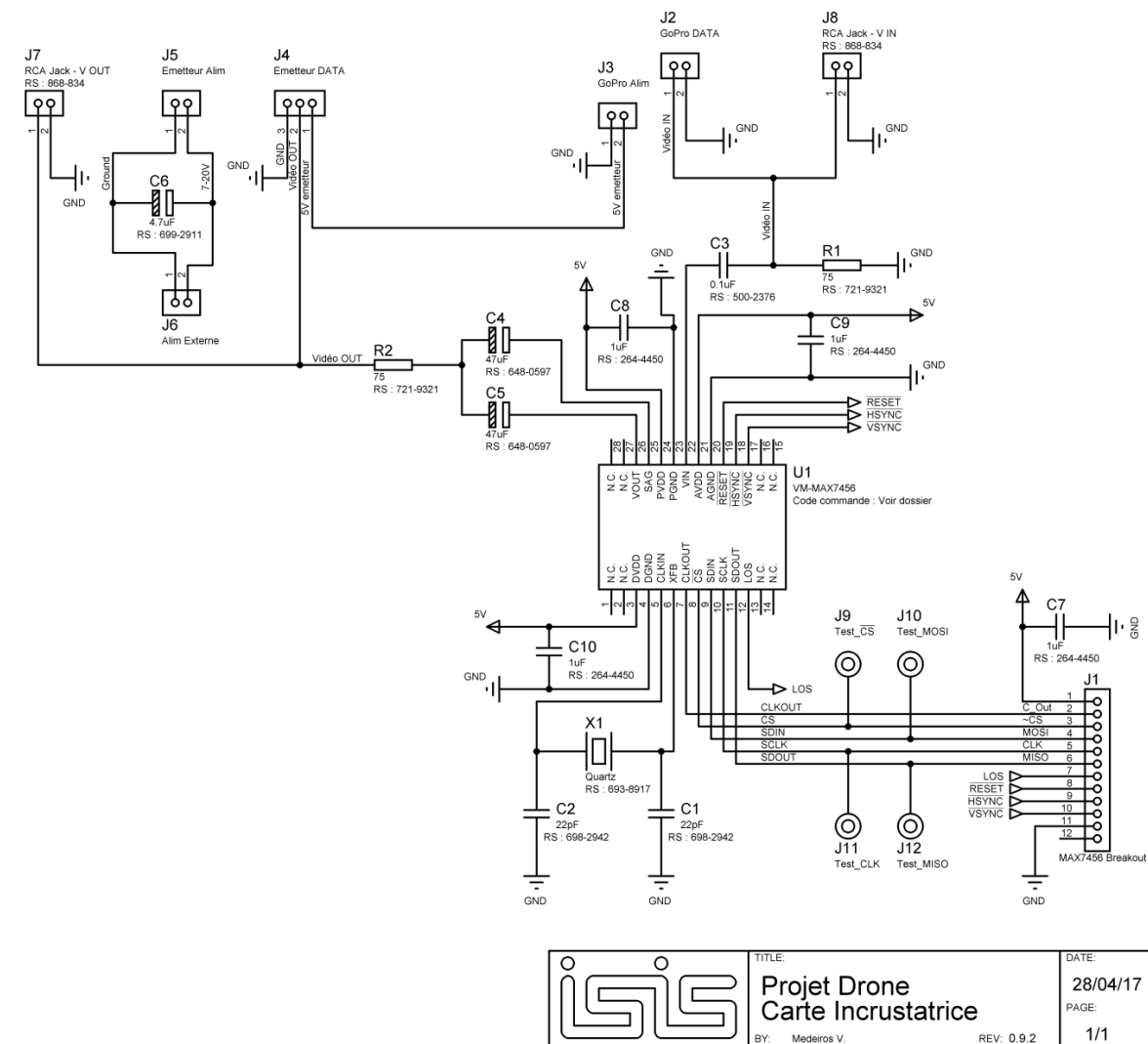
Carte d'incrustation (v2) – Schéma ISIS

Les tests réalisés ont donc permis de déceler des problèmes sur cette nouvelle carte d'incrustation.

Il semble y avoir un problème avec le Quartz X1 (réf. Schéma ISIS), les levels shifters disposés sur la carte pour améliorer la communication SPI semblent inutiles ce qui rend par conséquent inutile la broche 3.3V du connecteur J1 (réf. Schéma ISIS).

Enfin, il faudra ajouter 3 condensateurs de 0.1uF entre certaines broches du MAX7456 U1 (réf. Schéma ISIS) DVDD & DGND, PVDD & PGND, et AVDD & AGND comme indiqué sur la documentation du composant. Il faudra aussi ajouter 4 points test à la liaison SPI : ~CS, MOSI, MISO et CLK arrivant par le connecteur J1 (réf. Schéma ISIS).

Voici le schéma ISIS de la carte d'incrustation (v2) :



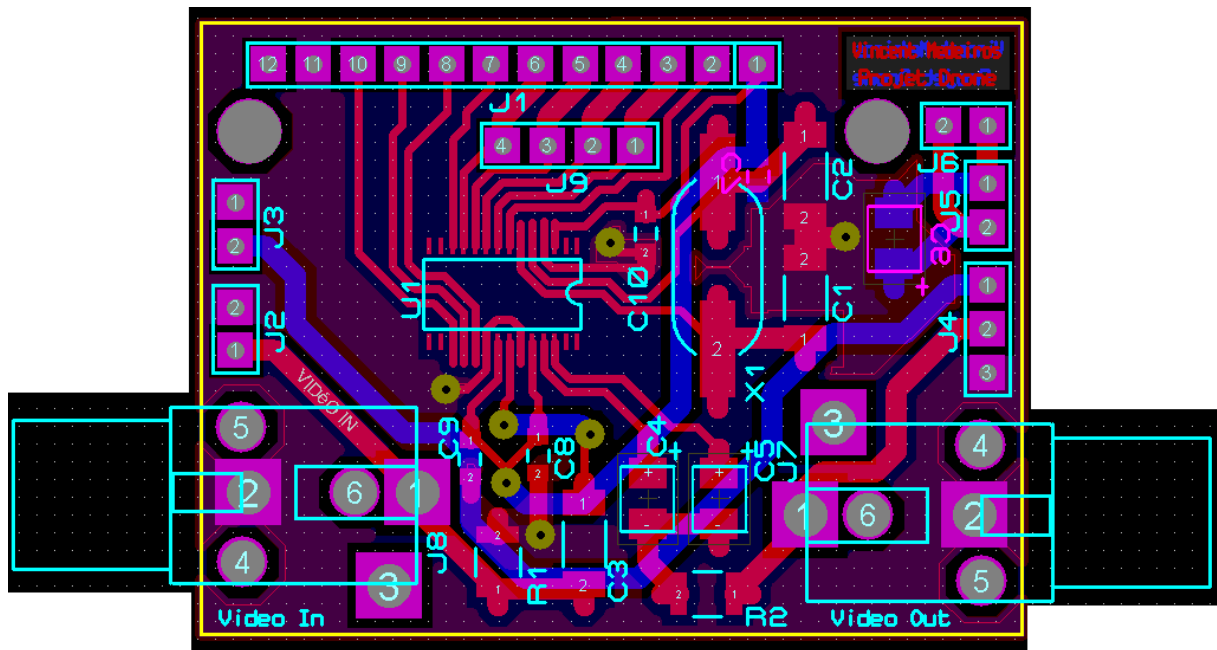
A noter qu'une version plus lisible de ce schéma sera disponible à la page 31.

Carte d'incrustation (v2) – Schéma ARES

Pour conclure la seconde partie du projet, j'ai rédigé une deuxième version du dossier, et commencé le typon (sous Proteus ARES) de la seconde version de ma carte d'incrustation, qui sera normalement prête en début de 3^{ème} partie du projet.

Le schéma ARES de la carte d'incrustation (v2) a donc été réalisé en début de troisième partie du projet.

Voici le schéma ARES de la carte d'incrustation (v2) :



A noter qu'une version « top » et « bottom » de ce schéma sera disponible à la page 32.

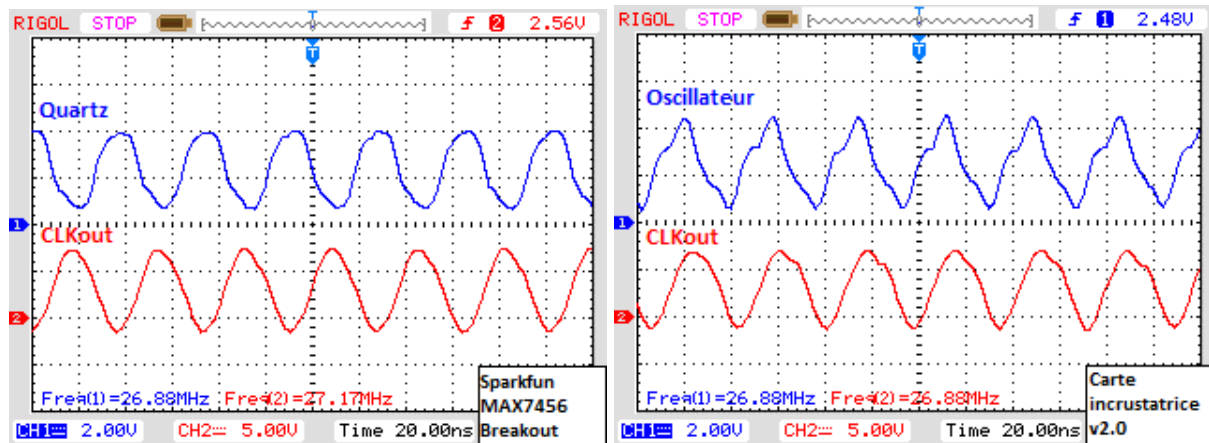
Tests du Quartz de la carte d'incrustation de Sparkfun et d'un oscillateur

Lors de cette dernière partie du projet, j'ai réalisé des tests sur le Quartz de ma carte d'incrustation (v2), et sur le Quartz de la carte d'incrustation de Sparkfun. Ces tests ont permis de voir que le Quartz n'oscille pas à la bonne fréquence, comparée à celle attendue (27MHz) et obtenue sur la carte de Sparkfun.

M. Hortolland m'a donc renseigné sur un oscillateur 25MHz capable d'assurer le rôle du Quartz.

A la réception de cet oscillateur, M. Hortolland l'a soudé à la carte d'incrustation (v2) puis j'ai réalisé des tests sur celui-ci, tout en prenant le Quartz fonctionnel de la carte d'incrustation de Sparkfun comme référence.

Voici les chronogrammes obtenus sur l'oscilloscope utilisé dans ces tests :



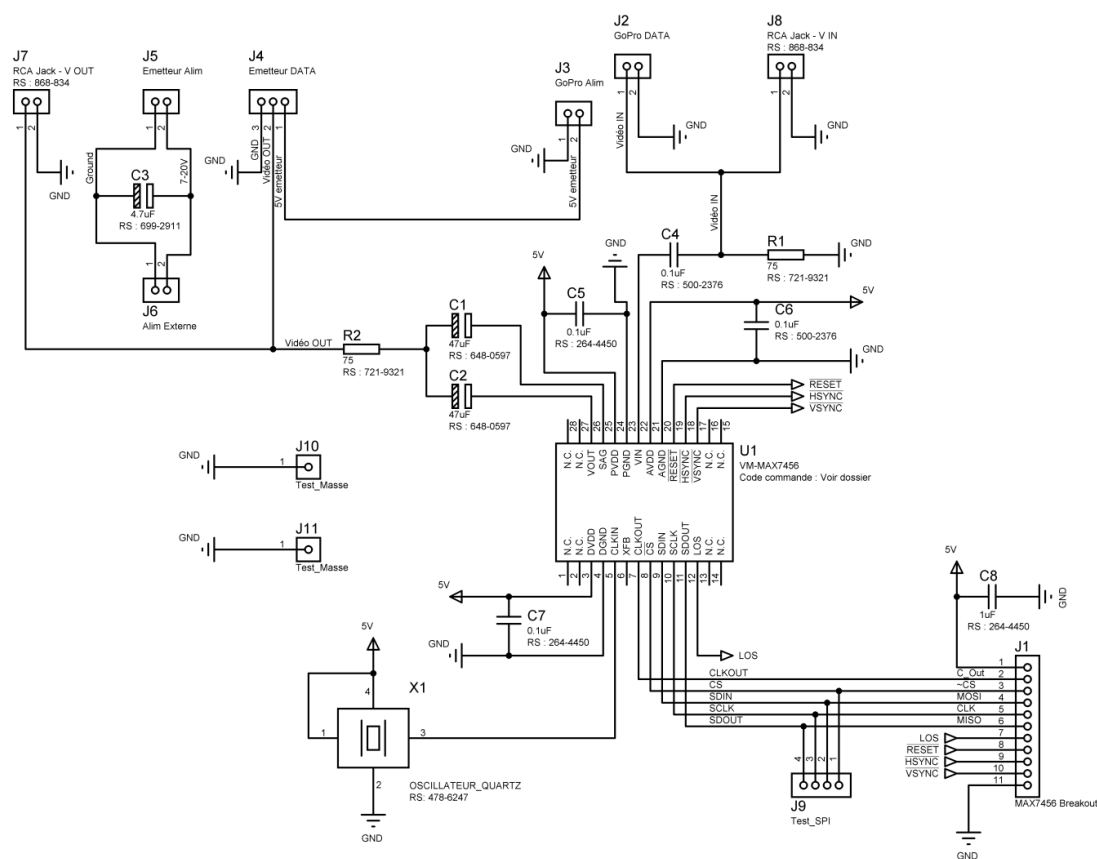
A noter qu'une photo relatant des tests est disponible à la page 35.

Carte d'incrustation (v3) – Schéma ISIS

Les tests concluants effectués sur l'oscillateur 27MHz m'ont conduit à réaliser une version finale de ma carte d'incrustation personnelle.

Celle-ci sera donc équipée d'un oscillateur à la place du Quartz et de ses deux condensateurs initialement disposés sur ma carte. Il a aussi fallu réaliser un symbole ISIS et une empreinte ARES pour un connecteur 11 broches (à l'aide de l'ancien connecteur 12 broches).

Voici le schéma ISIS de la carte d'incrustation (v3) :



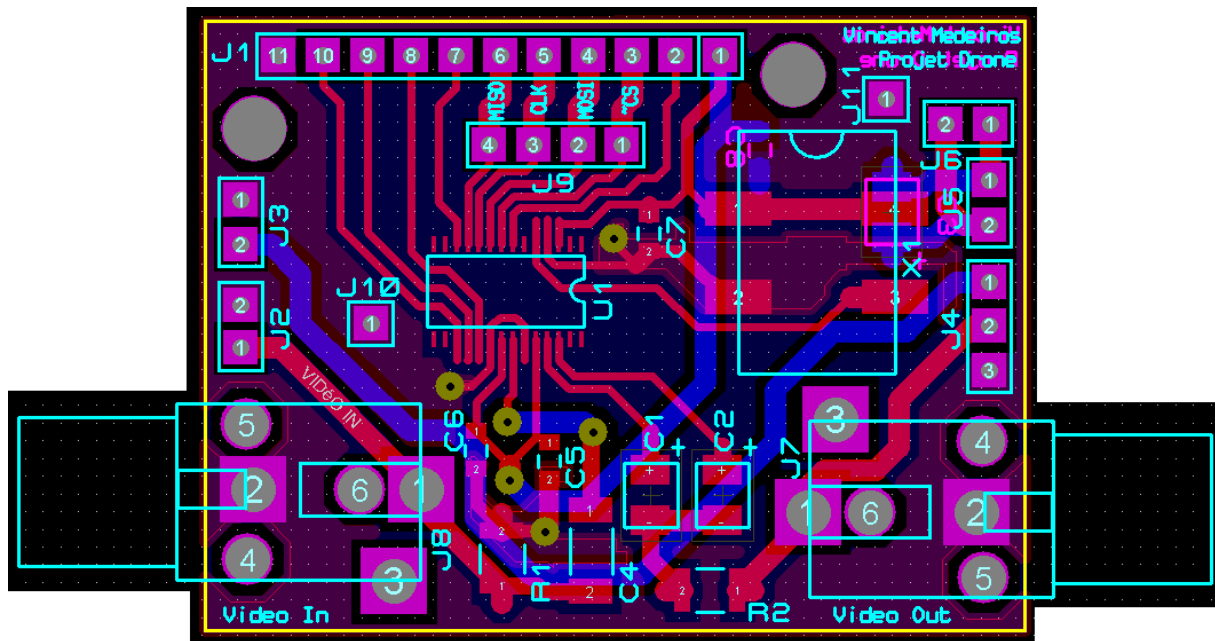
	TITLE:	DATE:
	Projet Drone Carte Incrustatrice	24/05/17
BY: Medeiros V.	REV: 1.0	PAGE: 1/1

A noter qu'une version plus lisible de ce schéma sera disponible à la page 33.

Carte d'incrustation (v3) – Schéma ARES

Dans la continuité du schéma ISIS, j'ai réalisé le schéma ARES qui nous a permis de lancer la création de la carte sur le site *seedstudio.com*.

Voici le schéma ARES de la carte d'incrustation (v3) :



A noter qu'une version « top » et « bottom » de ce schéma sera disponible à la page 34.

Ajout de caractères spéciaux dans la mémoire EEPROM de l'incrustateur MAX7456

Ma partie du projet comprend l'ajout de caractères affichables non présents dans la mémoire du MAX7456. Une procédure permet de modifier la liste des caractères affichables par le MAX7456, par des caractères voulus, dans notre cas le symbole de la température °C et pour l'humidité %

La réalisation de cette procédure s'est faite via le site f5mna.free.fr/Arduiexpert.htm sur lequel sont indiquées les étapes à suivre pour la modification de la liste des caractères affichables par le MAX7456.

Ce procédé nous demande de mettre en œuvre la carte d'incrustation de Sparkfun, le logiciel Tera Term et une carte Arduino, reliée à la carte d'incrustation, sur laquelle sera envoyé un programme pour modifier l'EEPROM du MAX7456.

Voici le tableau des caractères affichables par le MAX7456 avant et après la procédure :

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	1	2	3	4	5	6	7	8	9	0	A	B	C	D	E	F
1	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
2	U	W	X	Y	Z	a	b	c	d	e	f	g	h	i	j	k
3	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	(
4	)	.	?	:	;	,	'	/	""	-	<	>	@	アイウ		
5	エ	オ	カ	キ	ク	ケ	コ	サ	シ	ス	セ	ソ	ダ	チ	ツ	テ
6	ト	ナ	ニ	ヌ	ネ	ノ	ハ	ヒ	フ	ヘ	ホ	マ	ミ	ム	メ	モ
7	ヤ	ユ	ヨ	ラ	リ	ル	レ	ロ	ワ	ン	ビ	ブ	ボ	ビ	ボ	グ
8	ズ	ダ	デ	ド	ヤ	ユ	ョ	ッ	あ	い	う	え	お	が	き	く
9	げ	こ	さ	し	す	せ	そ	た	ち	つ	て	と	な	に	ぬ	ね
A	の	は	ひ	ふ	へ	ほ	ま	み	む	め	も	や	ゆ	よ	ら	り
B	る	れ	ろ	わ	を	ん	が	こ	だ	づ	て	ど	や	ゆ	よ	っ
C	再	生	早	巻	戻	年	月	日	火	水	木	金	土	主	副	声
D	音	色	濃	淡	開	始	終	了	時	刻	確	認	計	押	停	止
E	入	力	出	質	操	作	方	法	使	用	曜	量	■	□		
F	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■

A noter qu'une photo relatant de la procédure suivie est disponible à la page 35.

La procédure suivie nous a permis d'ajouter des caractères (encadrés en rouge) dans la mémoire du MAX7456 à la place de caractère inutiles pour notre projet.

Tests de l'émetteur et du récepteur 5.8GHz

Lors de cette dernière partie du projet, nous avons reçu l'émetteur et le récepteur acheté lors de la seconde partie du projet. Ceux-ci ont donc été testés pour vérifier qu'ils étaient bien compatibles avec notre projet. Pour ces tests nous avons utilisé la carte d'incrustation de Sparkfun reliée en entrée à la GoPro Hero3+ et en sortie à l'émetteur lui alimenté par une alimentation 10V (il possède une plage d'alimentation entre 7 et 20V), cet émetteur a été réglé à une fréquence de 5.705GHz ainsi que l'émetteur relié lui à l'écran de contrôle.

Voici les photos des tests réalisés :

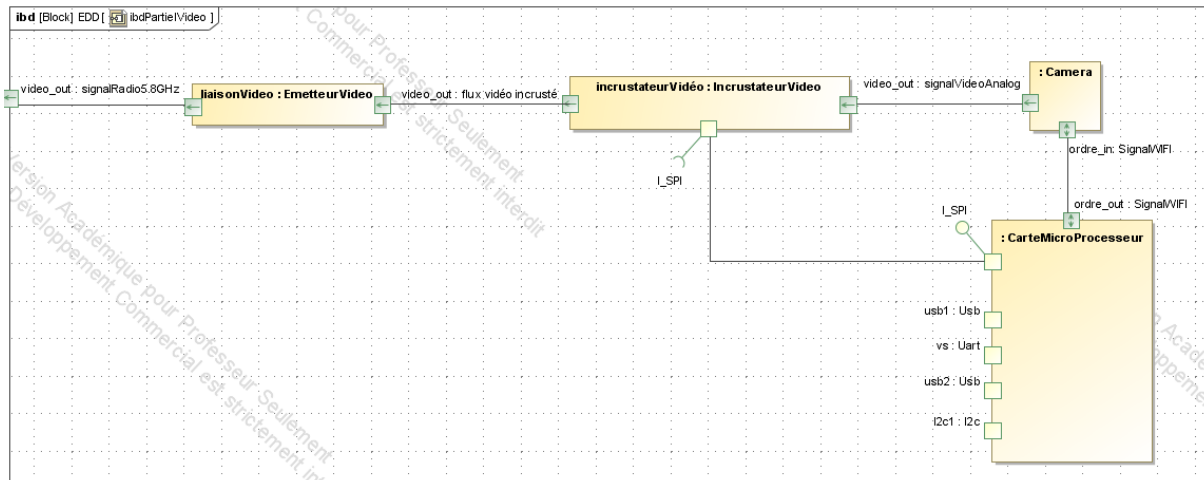


IBD de ma carte d'incrustation personnelle

Suite aux tests réalisés sur l'émetteur et le récepteur je me suis attelé à la réalisation d'un IBD de ma carte d'incrustation personnelle avec l'aide de M. Antoine.

Cet IBD a été réalisé en tenant compte des différents diagrammes déjà réalisés pour le projet.

Voici l'IBD de la carte d'incrustation :



Simulation de la fabrication de la carte d'incrustation

Nous avons pu réaliser une simulation de la fabrication de ma carte d'incrustation (v3) sur le site seedstudio.com

Cette simulation nous permet de vérifier la conformité de notre carte, mais aussi les différents coûts auxquels s'attendre lors de la commande de cartes.

Plusieurs options de fabrication peuvent être choisies, il est aussi possible de faire fabriquer sa carte et de faire souder ses composants (en les indiquant lors de la commande) mais cela entraîne des coûts supplémentaires.

Voici quelques captures d'écran relatant des possibilités offertes par le site seedstudio.com :

The screenshot displays the SeedStudio PCB configuration interface. The main configuration area includes options for Material (FR-4 TG130), Layer (2 layers), PCB Dimension (35.8 mm * 47.8 mm), PCB Qty (5), Different design(s) (1), PCB Thickness (1.6 mm), PCB Color (Blue), and Surface Finish (HASL). A right-hand panel shows the 'PCB Orders' summary with a total cost of €4.37 and a weight of 0.03kg. Below the configuration area, the 'Layers' section lists various CAD files for drilling, mechanical, solder resist, silk screen, and copper. The bottom part of the image shows a preview of the PCB layout with components labeled J1 through J11, C1 through C7, R1, R2, and X1, along with 'Video In' and 'Video Out' ports.

PCB Orders €4.37

Material	FR-4 TG130
Layer	2 layers
PCB Dimension	35.8mm * 47.8mm
PCB Qty.	5
Different design(s)	1
PCB Thickness	1.6mm
PCB Color	Blue
Surface Finish	HASL
Min Solder Mask Dam	0.4mm†
Copper Weight	1oz.
Min Hole Size	0.3mm
Min Tracking / Spacing	6/6 mil
Blind Vias	no
Half-cut / Castellated Holes	no
Impedance Control	no
Production time:	5 working days

Sub-Total €4.37
Weight 0.03kg

Layers

EasyEDA - Free PCB Editor

- ✓ Carte_Incrustatrice_Medeiros_v 1.1 - CAD/CAM Drill.TXT
- ✓ Carte_Incrustatrice_Medeiros_v 1.1 - CAD/CAM Mechanical 1.TXT
- ✓ Carte_Incrustatrice_Medeiros_v 1.1 - CAD/CAM Top Solder Resist.TXT
- ✓ Carte_Incrustatrice_Medeiros_v 1.1 - CAD/CAM Top Silk Screen.TXT
- ✓ Carte_Incrustatrice_Medeiros_v 1.1 - CAD/CAM Top Copper.TXT

Overall Top Bottom theme Surface Finish Reassign Layers Need help? Old GerberViewer

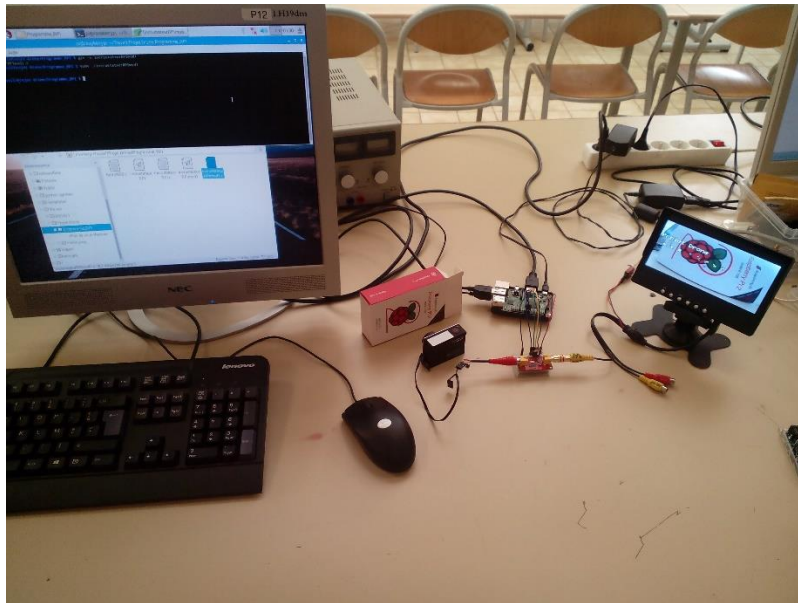
Video In Video Out Vincent Medeiros Projet Drone

Test de la carte d'incrustation Sparkfun avec les caractères modifiés

L'une des dernières tâches de cette troisième partie du projet a été de tester d'incruster à l'écran des caractères ajoutés plus-tôt via la procédure sous Arduino.

Il a donc fallu modifier le programme utilisé (celui transféré d'Arduino vers la Raspberry Pi) pour y inclure des lignes indispensables à l'appel de ces nouveaux caractères.

Voici des photos illustrant le test :



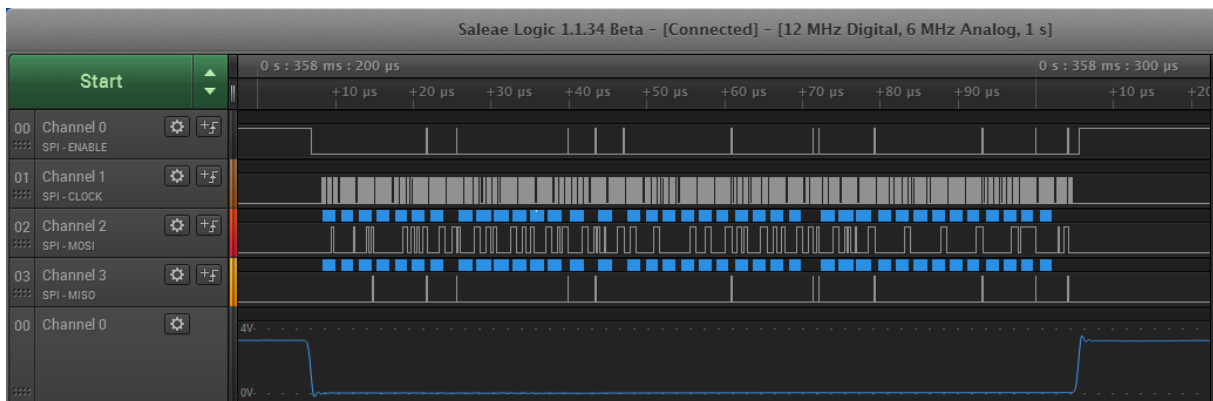
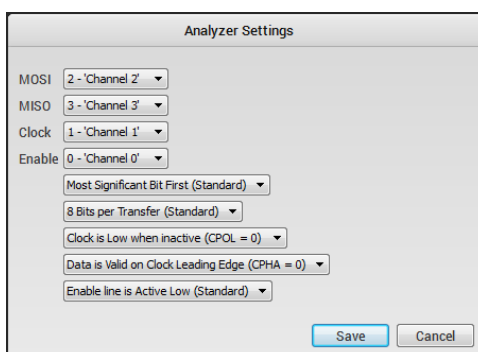
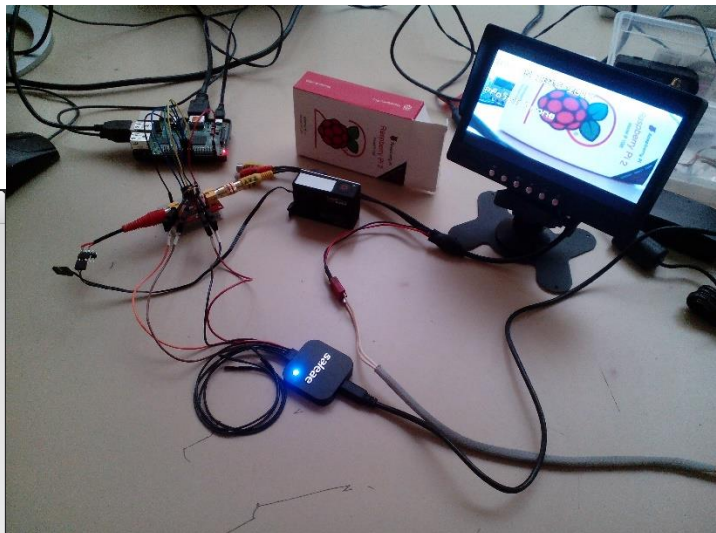
Partie Physique

Analyse d'une trame SPI

Au cours de la troisième partie du projet, j'ai réalisé avec l'aide de M. Haye, professeur de physique du groupe EC, une capture de la communication SPI entre la carte d'incrustation de Sparkfun et la Raspberry Pi.

Cette capture a été réalisée grâce à l'analyseur logique Saleae.

Voici une capture de la trame SPI sur le logiciel Logic et une capture de la configuration utilisée pour l'analyse logique :



Conclusion

Finalisation du projet :

Ma mission dans le projet a été accomplie en grande partie, il reste encore quelques petites parties à finaliser.

L'une d'entre elles est l'utilisation de la carte d'incrustation (v3) pour s'assurer de son bon fonctionnement, elle devrait être reçue la semaine suivant la rédaction de ce dossier, il faudra ensuite souder les composants et réaliser des tests avec le programme d'incrustation utilisé plus tôt dans le projet.

Il faudra ensuite incruster sur l'écran de contrôle les données des capteurs disposés sur la carte de l'étudiant 8 EC : Thomas Bonnefille.

La fin du projet comprendra aussi la création d'un diagramme de séquence pour expliquer le programme d'incrustation.

Problèmes rencontrés :

Ma partie du projet a rencontré plusieurs problèmes, principalement lors de la création de la nouvelle carte d'incrustation. Celle-ci a été réalisée avec comme modèle la carte d'incrustation de Sparkfun, fonctionnelle, mais je n'ai pas été capable de concevoir une réplique fonctionnelle de celle-ci. Les nombreux problèmes rencontrés lors de la réalisation de celle-ci en attestent. La version finale (v3) sous-traitée par *seedstudio.com* peut fonctionner comme être défectueuse, seuls les tests réalisés sur celle-ci lors de sa réception pourront définir de son bon état de fonctionnement. J'ai aussi rencontré des problèmes lors de l'installation des logiciels MagicDraw et Project (Gantt) sur mon ordinateur personnel.

Compétences acquises :

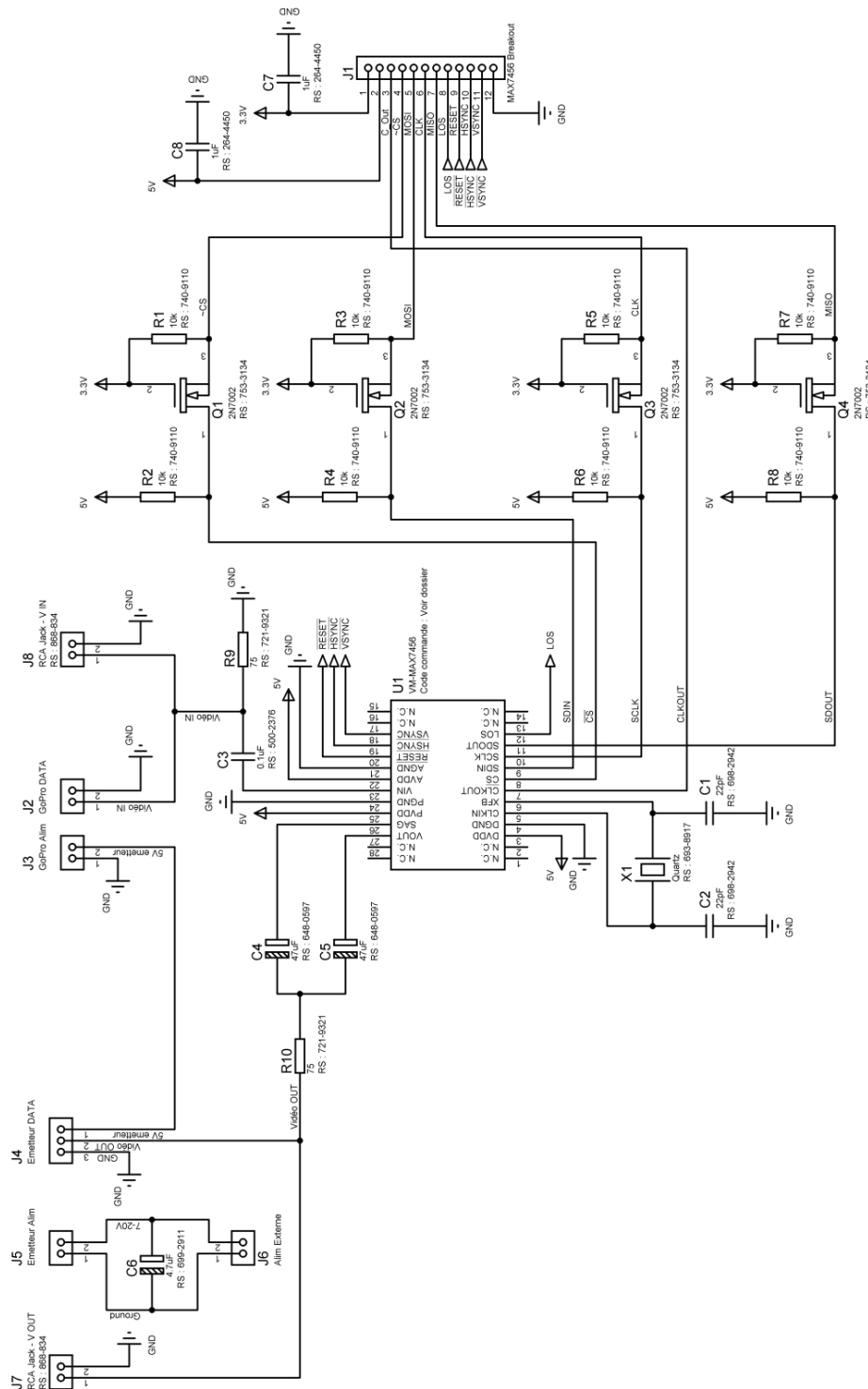
Ce projet a été très enrichissant pour moi, il m'a permis de perfectionner l'apprentissage reçu dans différents domaines : au niveau de la réalisation de schémas structurels sur ISIS, de typons sur ARES, de l'utilisation et du paramétrage d'une Raspberry Pi...

Le travail en groupe a aussi été une des compétences acquises lors de ce projet, il m'a permis de découvrir certains camarades de classe dotés de compétences variées, qui une fois mises en commun, ont permis l'avancement du projet.

Ce projet m'a donc été bénéfique, sur le plan intellectuel mais aussi sur le plan humain.

Annexes

Schéma ISIS carte d'incrustation personnelle v1



	TITLE: Projet Drone Carte Incrustatrice	
	BY: Medeiros V.	REV: 0.6
	DATE: 06/04/17	PAGE: 1/1

Schéma ARES carte d'incrustation personnelle v1 (top / bottom)

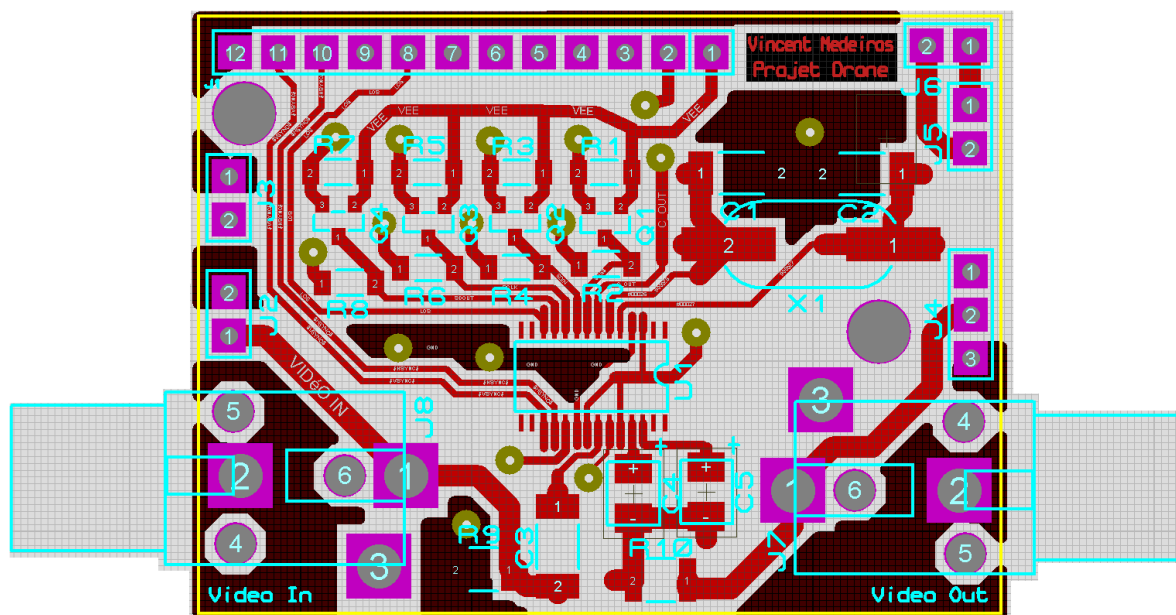
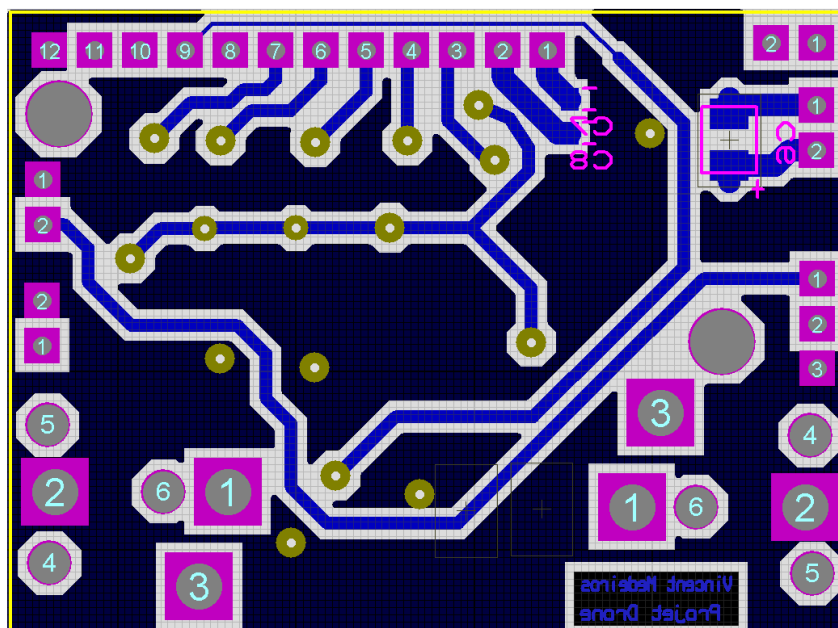
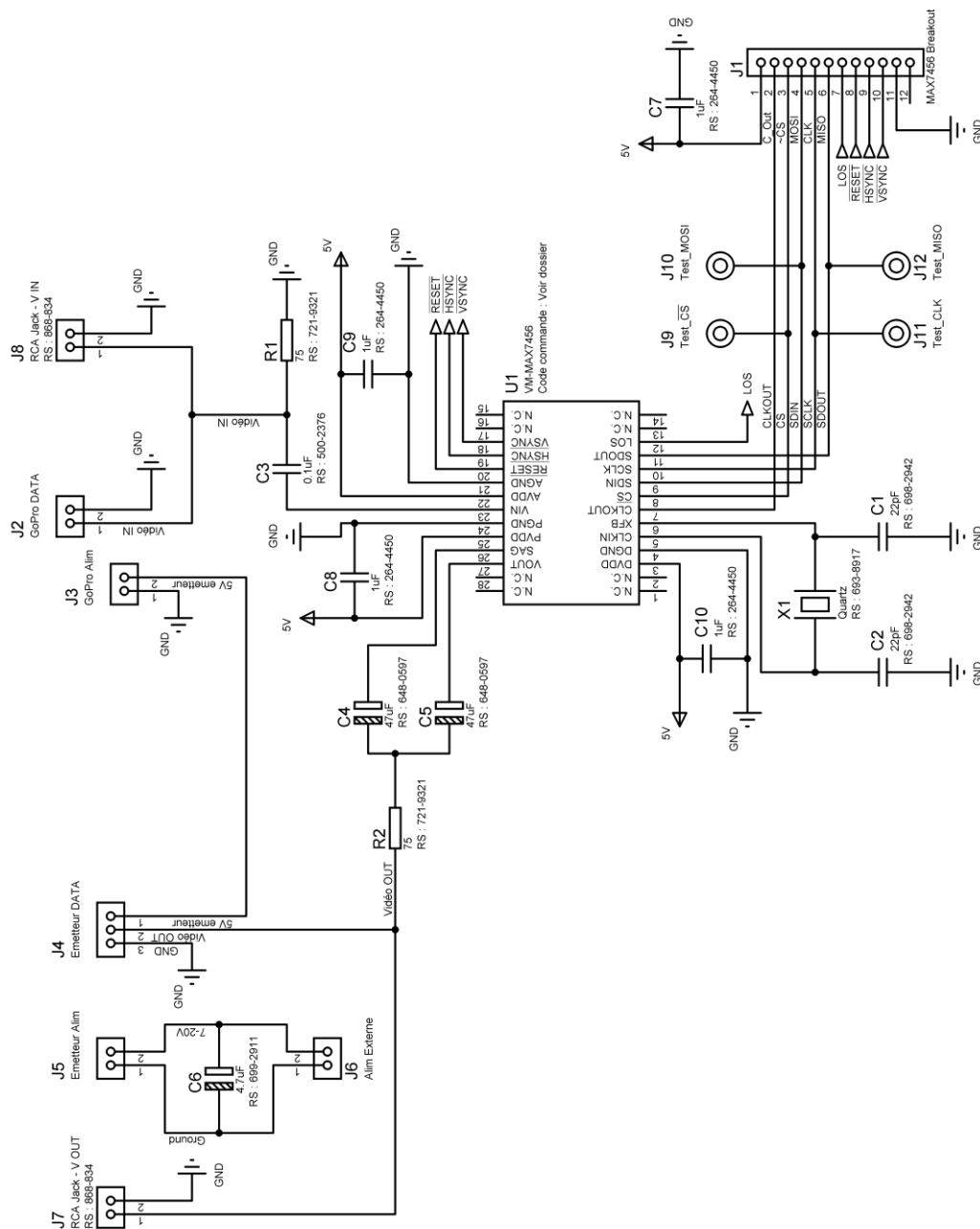


Schéma ISIS carte d'incrustation personnelle v2



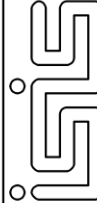
	TITLE:	Projet Drone Carte Incrustatrice	
	BY: Medeiros V.	REV. 0.9.2	PAGE 1/1

Schéma ARES carte d'incrustation personnelle v2 (top / bottom)

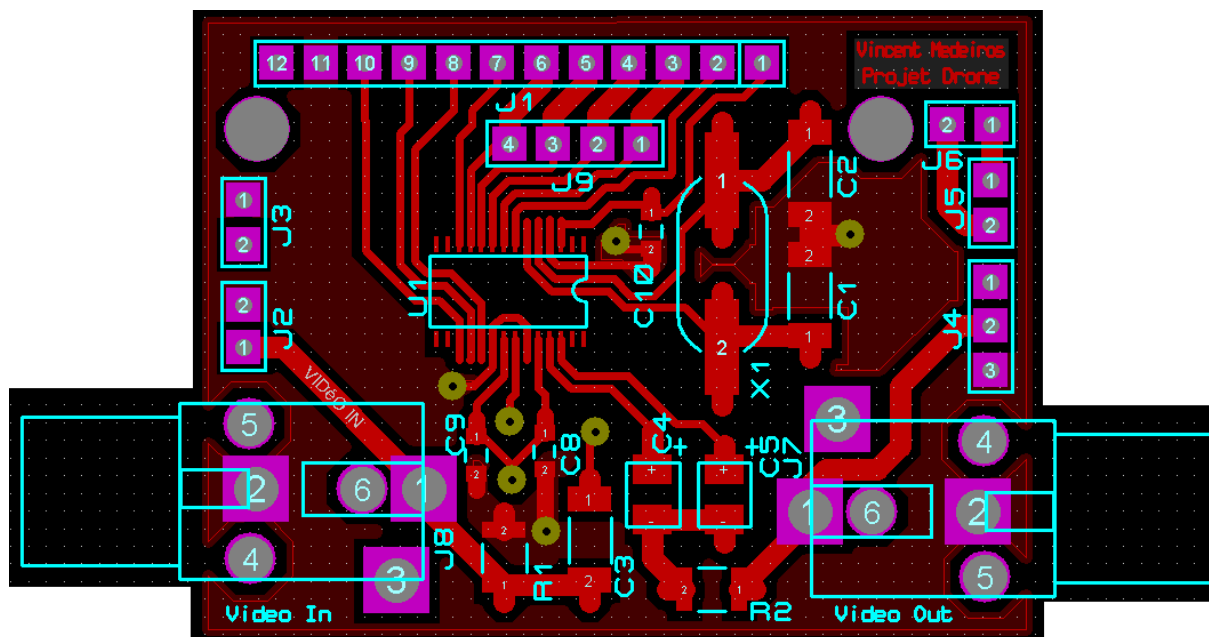
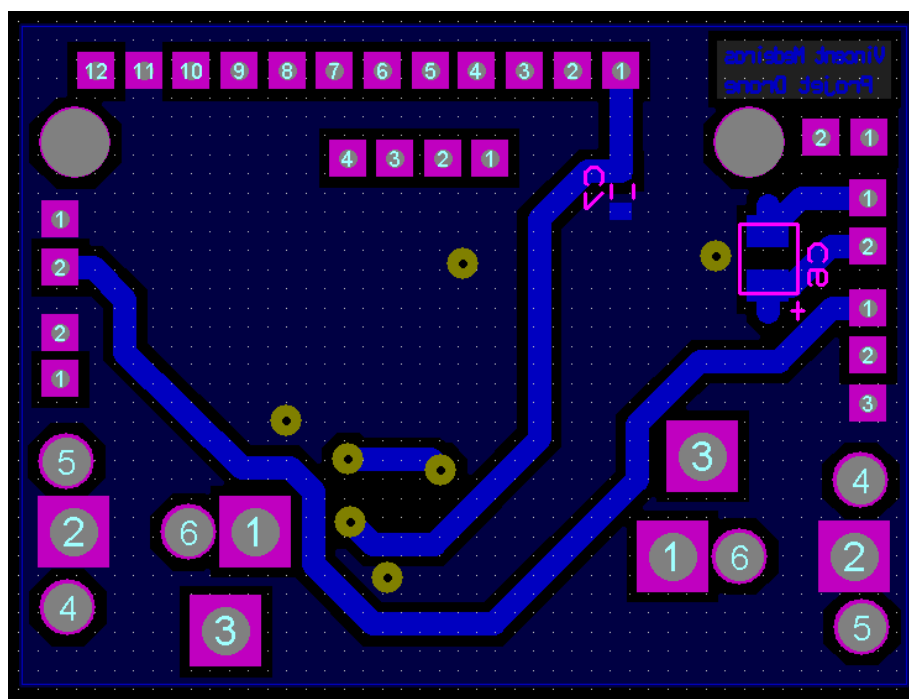
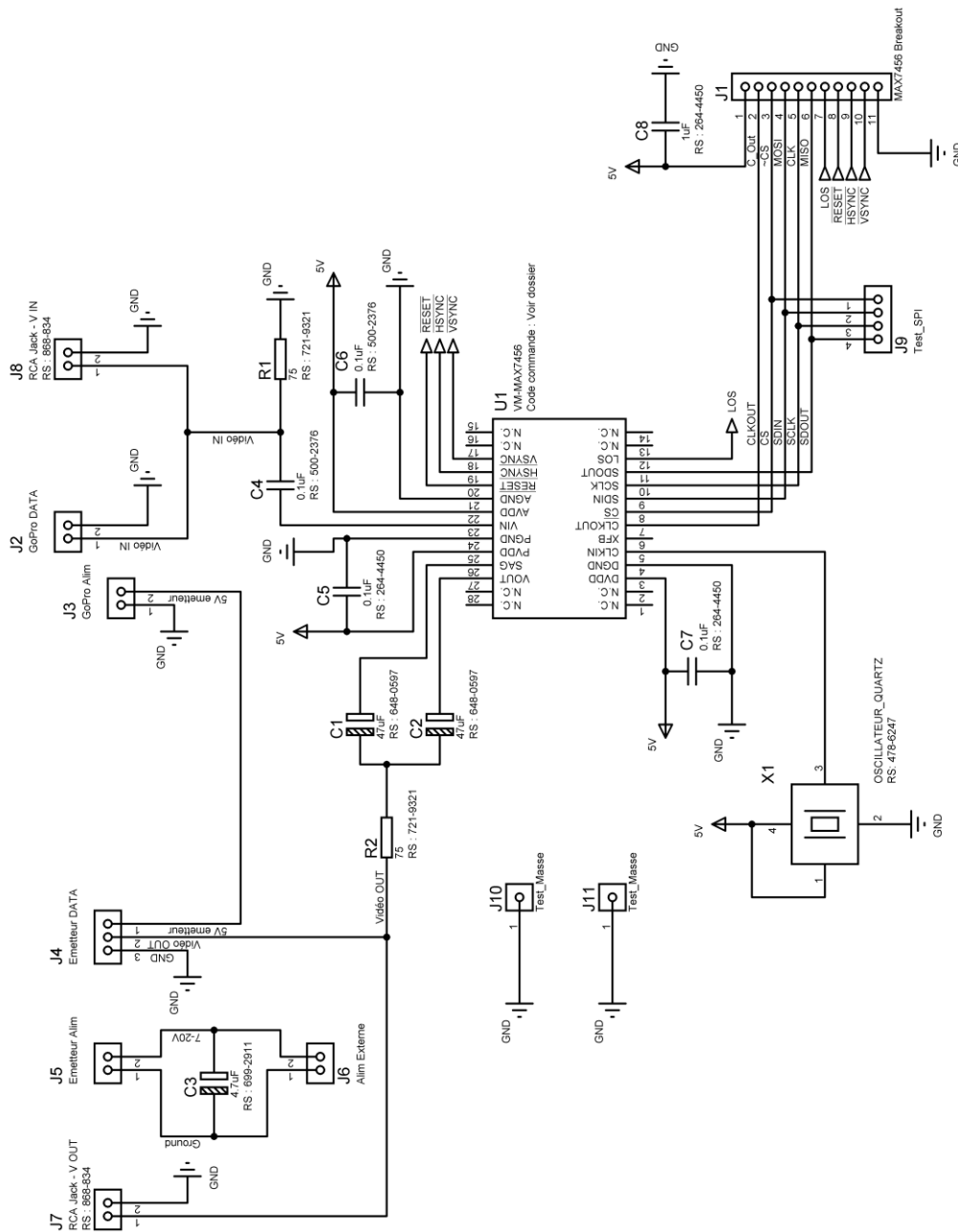
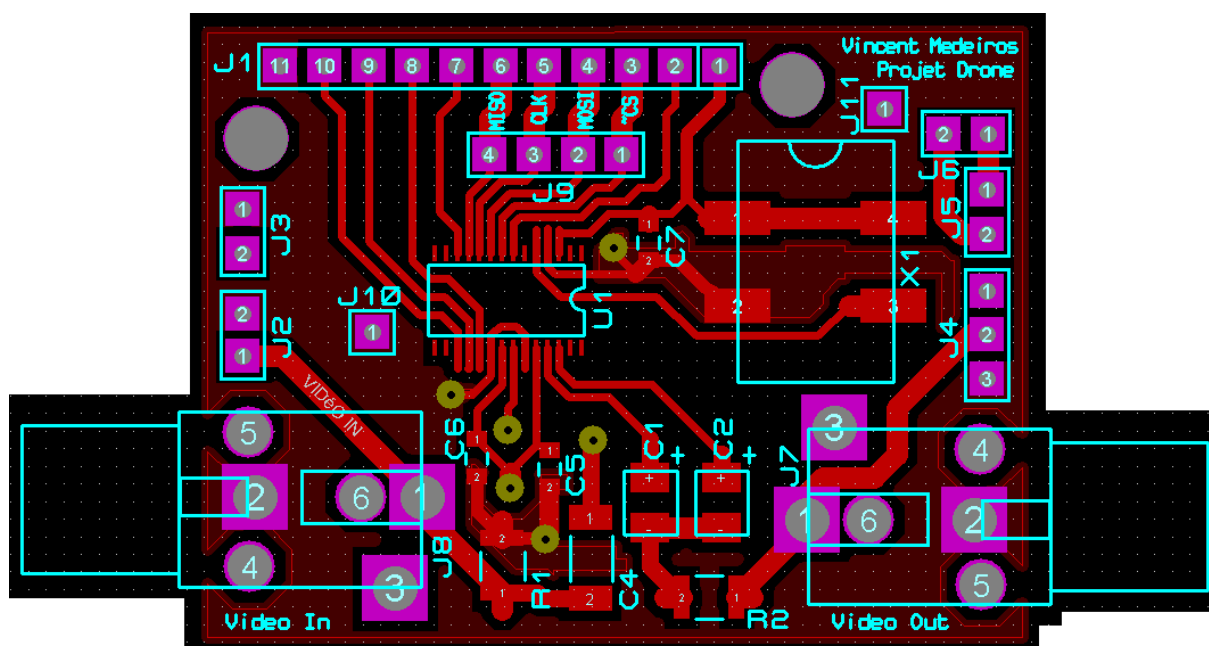
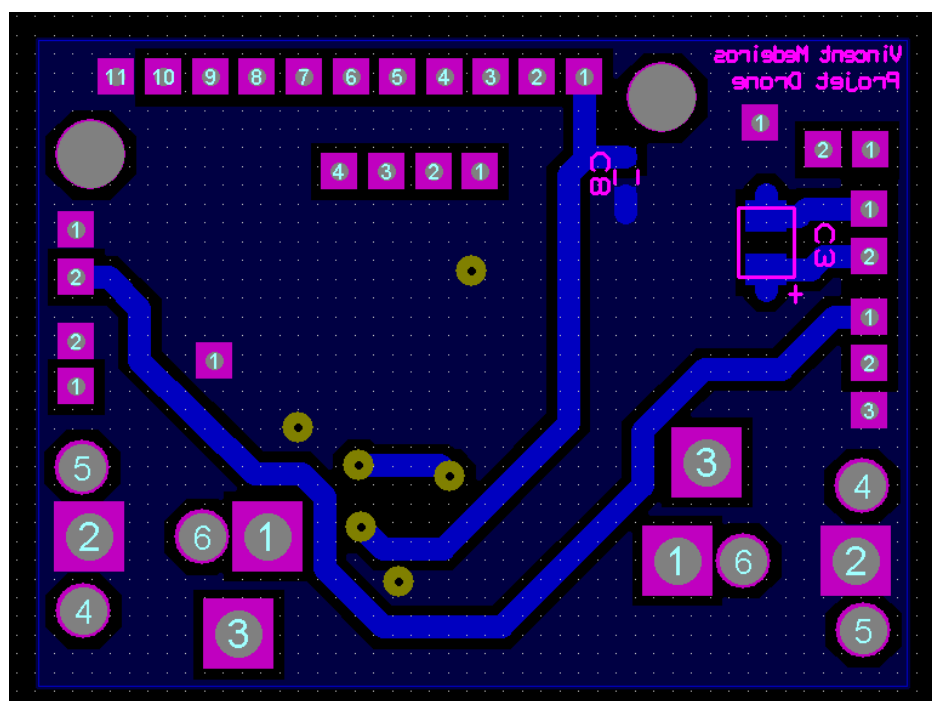


Schéma ISIS carte d'incrustation personnelle v3

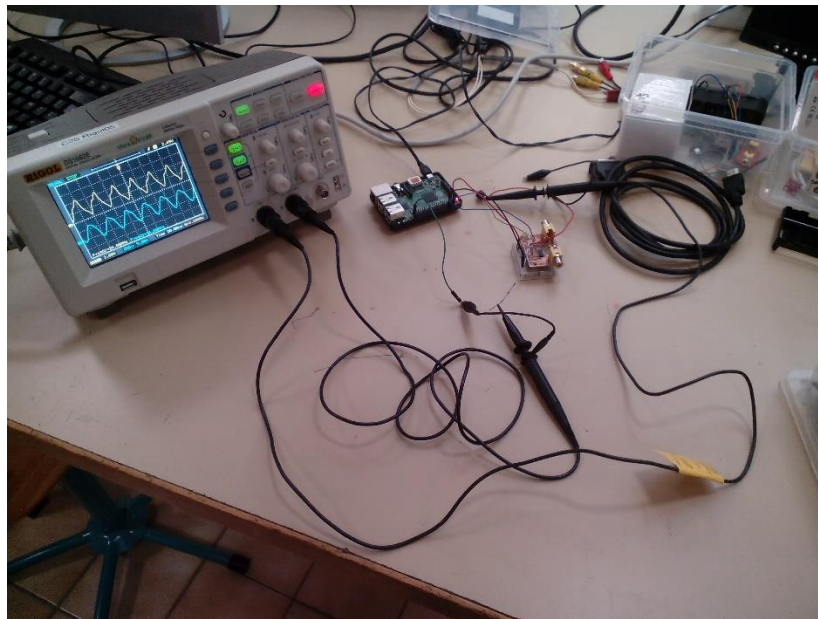


	TITLE:	Projet Drone Carte Incrustatrice	DATE:	24/05/17
	BY:	Medeiros V.	PAGE:	1/1
		REV:	1.0	

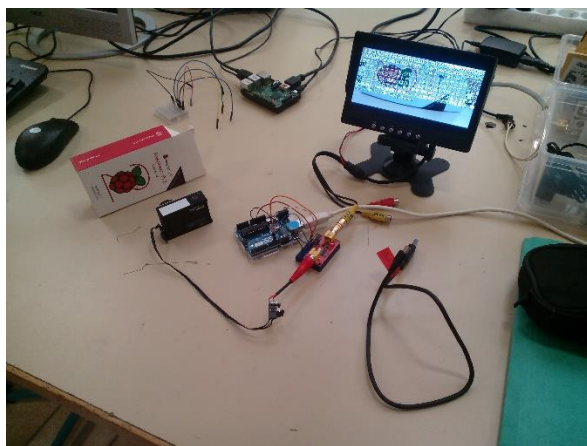
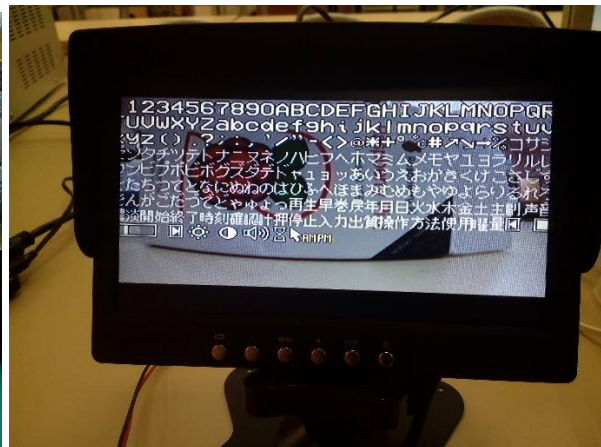
Schéma ARES carte d'incrustation personnelle v3 (top / bottom)



Tests du Quartz de la carte d'incrustation de Sparkfun et d'un oscillateur



Ajout de caractères spéciaux dans la mémoire EEPROM de l'incrustateur MAX7456

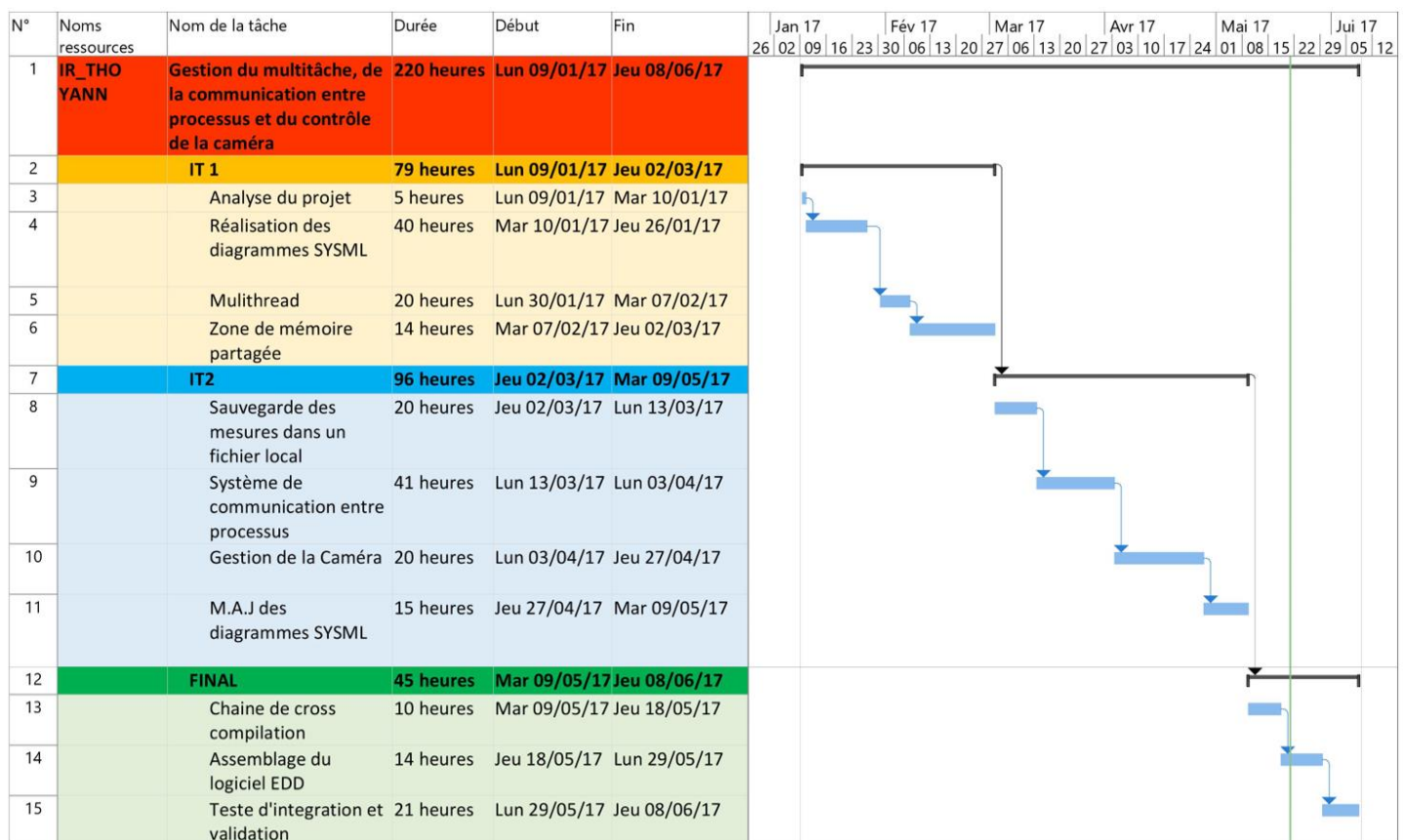


III – IR : DEVELOPPEMENT DU LOGICIEL EDD

Présentation personnelle

Pour ma part, je m'occupe de la partie logiciel du drone (EDD). Mon travail consiste à développer une partie du logiciel EDD en C++ à l'aide de l'environnement de développement QT sur une carte Raspberry Pi 3. J'ai pour but de mettre en œuvre une gestion multitâche (réaliser du multithread, principalement pour le contrôle des capteurs), une zone de mémoire partagée accessible par tous les threads, une sauvegarde des mesures dans un fichier local, un système de communication entre processus, de mettre en place la gestion de la caméra ainsi que d'assembler toutes les parties du logiciel EDD. Je suis en collaboration avec un IR, qui lui s'occupe de la communication avec GCS et trois EC, dont un s'occupe de l'incrustation vidéo de la caméra et puis les deux derniers, de la réalisation de classe C++ des capteurs.

Afin d'optimiser mon travail, j'ai réalisé un diagramme de GANTT « D_GANTT_Tho », sur une période du 09/01/17 au 08/06/17.



Ce diagramme me permet de visualiser les différentes tâches à réaliser, ainsi que la durée qui peut leur être accordée. Cela me permet ainsi de visualiser l'avancement de mon travail dans le temps et de pouvoir m'organiser en conséquence. J'ai donc découpé mon temps de travail en trois parties pour chaque itération. Pour la première itération, je consacre mon temps à l'analyse du cahier des charges ainsi qu'à la réalisation des diagrammes SYSML. Ensuite je passe à la réalisation de la gestion du multithread et puis pour finir avec la mise en place de la zone mémoire commune et accessible par tous les threads (processus lancé en arrière-plan).

Pour la deuxième itération, je commence par réaliser la sauvegarde des mesures dans un fichier local. Puis je développe le système de communication entre processus et pour terminer avec la conception de la gestion de la caméra. Pour la phase finale, je m'occupe de la cross compilation, ensuite d'assembler toutes les parties du logiciel EDD et puis finir par les tests d'intégrations et de validations du logiciel EDD.

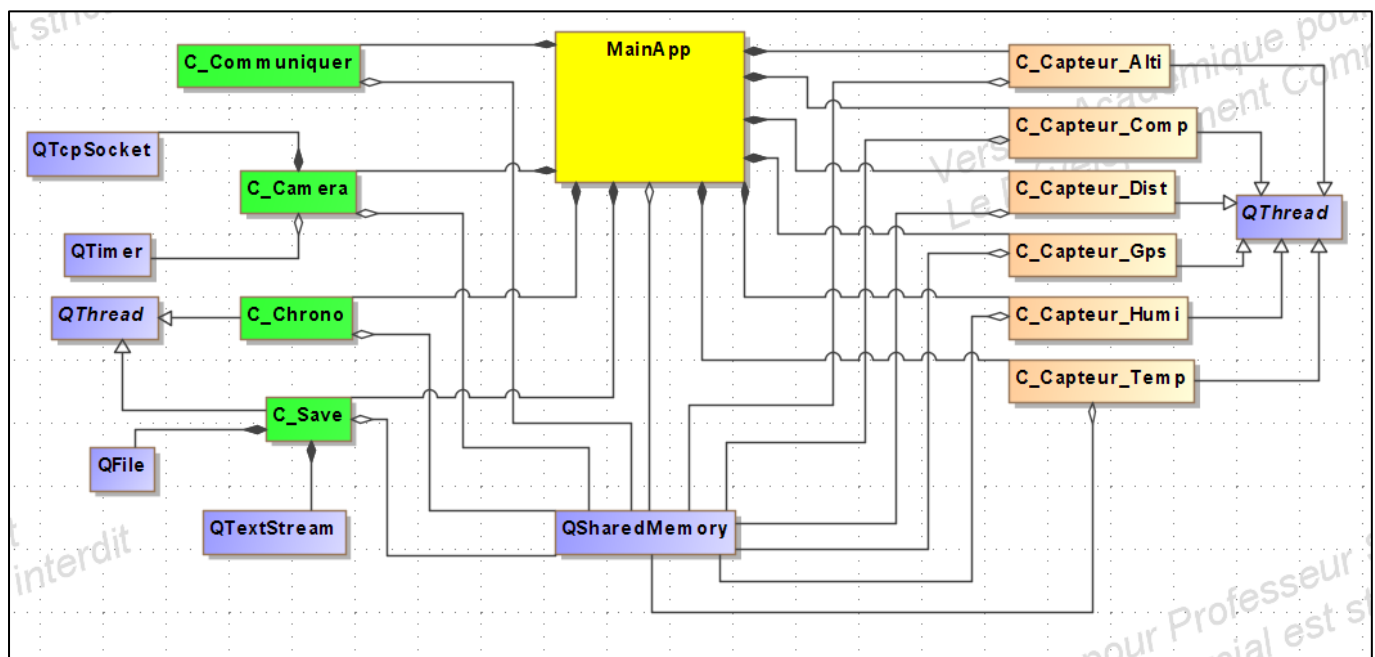
Travail réalisé

Vue d'ensemble du logiciel EDD

Jusqu'à présent, j'ai pu concevoir 20 versions du logiciel EDD. J'ai mis au point un diagramme de classe (schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que les différentes relations entre celles-ci) « Class_EDD » dans le but de pouvoir schématiser la composition du logiciel.

Diagramme de classe synthétisé du logiciel EDD : « Class_EDD »

Voir annexe p58-59 : « Class_EDD »



Le logiciel EDD est composé de 11 classes. Ci-dessus, représenté en jaune, la classe « MainApp » correspond à la classe principale. Cette classe permet d'initialiser le programme, de créer la mémoire partagée, d'instancier les objets des classes présente dans le logiciel EDD et ainsi que d'instancier des threads. Ensuite, les classes beiges représente les classes « C_Capteur », elles permettent de prendre le contrôle sur les capteurs (récupèrent les mesures prélevées par les capteurs). Puis les classes vertes, sont les classes ni principales et ni « C_Capteur ». Et pour terminer, les classes représentées en bleu correspondent aux classes propres à l'environnement de développement QT.

Les classes « C_Chrono », « C_Save » et les classes « C_Capteur » sont utilisées en tant que thread.

Conception du multithread

Le logiciel EDD est équipé de divers processus qui fonctionnent en simultané. La conception du multithread est nécessaire dans le but de pouvoir lancer plusieurs processus en simultanés permettant de faire des prélèvements de mesures ainsi que de sauvegarder les données.

Pour qu'un thread puisse utiliser une classe, la classe doit hériter des méthodes de la classe « QThread ».

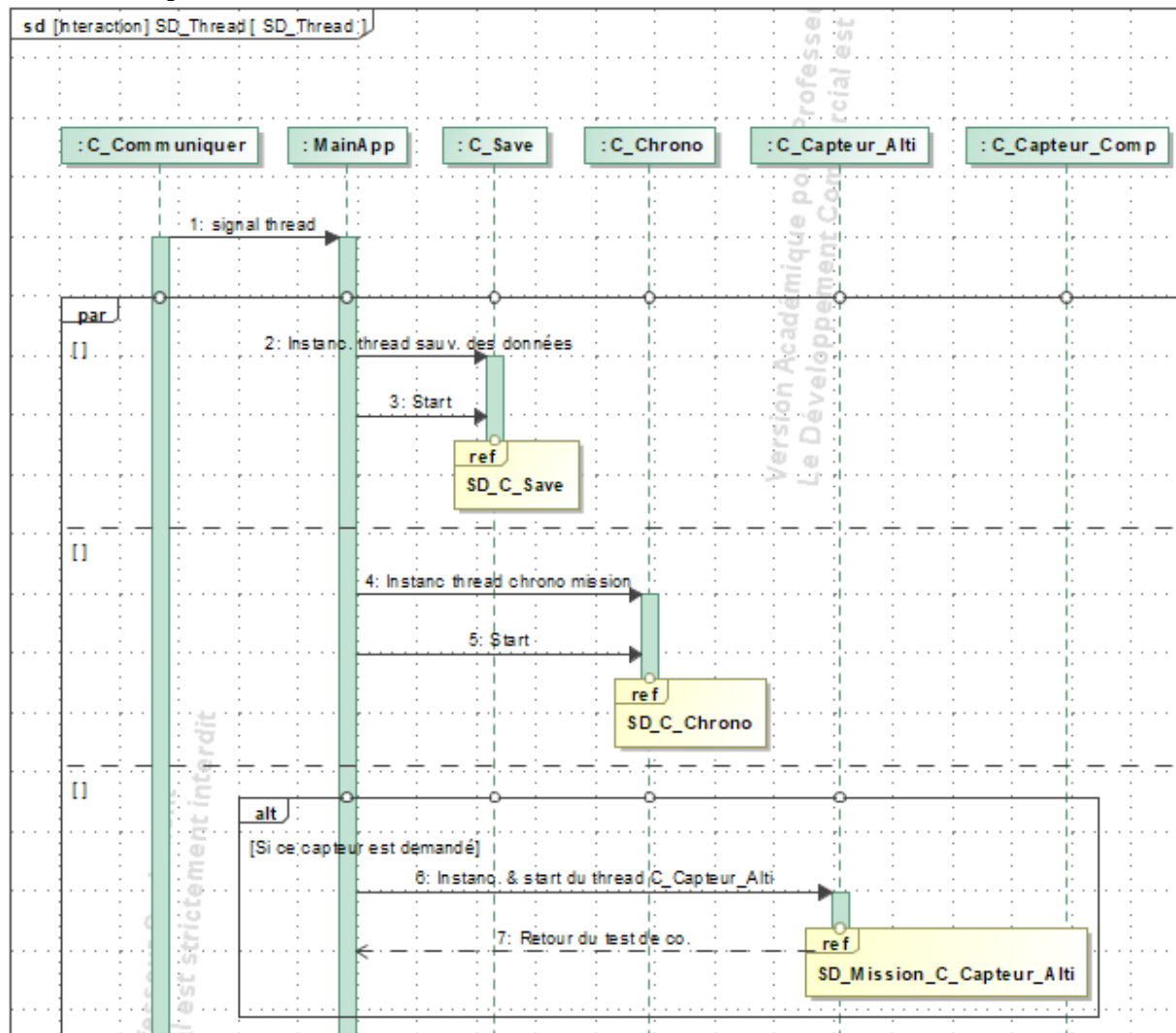
Exemple : La classe « c_save.h »

```
class C_Save : public QThread
{
```

Elle hérite de cette dernière dans le but de pouvoir utiliser une fonction virtuelle « run() ». Cette fonction permet de lancer un thread.

J'ai créé un diagramme de séquence (représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique) pour pouvoir visualiser de manière séquentielle le scénario du développement d'un thread : « SD_Thread »

Voir annexe p60 : « SD_Thread »



Avant de lancer un thread, je commence par instancier l'objet de la classe du thread de manière dynamique, puis je le lance.

Instanciation de l'objet dans la classe « mainapp.h » :

```
C_Save *sauver;
```

Puis dans « mainapp.cpp » :

```
sauver = new C_Save(this, nbCapteur);
```

ATTENTION !

L'instanciation des objets « C_Capteur » se fait par une liste de thread à l'aide de la classe QList dans la classe « mainapp.h » :

```
QList<QThread*> capteurs;
```

Puis dans « mainapp.cpp » :

```
capteurs.append(new C_Capteur_Alti(this, data[i].numCap));
```

Lancement du thread « mainapp.cpp » :

```
sauver->start();
```

Je lance le thread à l'aide de la fonction « start() », cette fonction va agir sur la fonction « run() » du thread concerné.

Une fois un thread lancé, le thread tourne en boucle dans la fonction « run() » jusqu'à qu'on lui demande de s'arrêter. De manière à que l'on puisse arrêter un thread, j'ai édité une fonction « stop() » dans la classe d'un thread.

```
void stop();
```

Cette fonction permet de sortir de la boucle dans la fonction « run() », et par conséquent le thread se termine.

Mise en place de la zone de mémoire partagée

Une zone de mémoire partagée est le concept le plus approprié au projet. Cette zone de mémoire partagée permet d'avoir une zone de mémoire commune accessible par tous processus du logiciel EDD, ce qui permet de faciliter les échanges de données entre processus.

Pour réaliser cette zone de mémoire partagée, j'utilise une structure « T_Mes » qui représente l'espace mémoire à partager :

```
typedef struct
{
    // Capteurs
    int numCap;
    char idCap[AFFMAX];
    char nomCap[AFFMAX];
    char typeCap[AFFMAX];
    char posL[AFFMAX];
    char posC[AFFMAX];
    char unite[AFFMAX];
    char symbole[AFFMAX];
    char mesure[AFFMAX];
    // Elements
    int freqAcqui;
    char nomMission[AFFMAX];
    char chrono[AFFMAX];
    char date[AFFMAX];
    char heure[AFFMAX];
} T_Memo;
```

La structure est composée de deux parties, une réservée pour les données des capteurs et l'autre pour les données de configurations.

Pour pouvoir exploiter cette zone de mémoire, j'instancie cette zone de mémoire en l'instanciant à partir de la classe QT « QSharedMemory », et puis je réalise la zone de mémoire partagée.

Instanciation de l'objet de la classe QT « QSharedMemory » dans la classe « mainapp.h » :

```
QSharedMemory *mshm;
```

Et puis dans « mainapp.cpp » :

```
mshm = new QSharedMemory(KEY, this);
```

Je crée la zone de mémoire partagée :

```
mshm->attach();
if (!mshm->isAttached())
{
    int res = mshm->create(NBMAXCAP*sizeof(T_Memo));
    if (res == false)    qDebug(mshm->errorString().toString().c_str());
} // if
```

Lors du démarrage du logiciel, j'initialise cette zone de mémoire partagée :

```
// Initialisation de la mémoire partagée
T_Memo *init = (T_Memo *)mshm->data();
for (i = 0; i < NBMAXCAP; i++)
{
    mshm->lock();
    // Capteurs
    init[i].numCap = 404; // Pourquoi 404? parceque demander aux suisses !
    strcpy(init[i].idCap, "");
    strcpy(init[i].nomCap, "");
    strcpy(init[i].typeCap, "");
    strcpy(init[i].posL, "");
    strcpy(init[i].posC, "");
    strcpy(init[i].symbole, "");
    strcpy(init[i].unite, "");
    strcpy(init[i].mesure, "");
    // Elements
    init[i].freqAcqui = 1000;
    strcpy(init[i].nomMission, "");
    strcpy(init[i].chrono, "");
    strcpy(init[i].date, "");
    strcpy(init[i].heure, "");
    mshm->unlock();
}
```

Afin de pouvoir lire et écrire dans cette zone de mémoire partagée, j'utilise la fonction « lock() » dans le but de bloquer l'accès à la zone de mémoire partagée aux autres processus car cela peut engendrer des problèmes d'écriture et de lecture :

```
mshm->lock();
```

Une fois bloqué, j'écris ou je lis l'information, puis j'utilise la fonction « unlock() » pour rendre l'accès possibles aux autres processus :

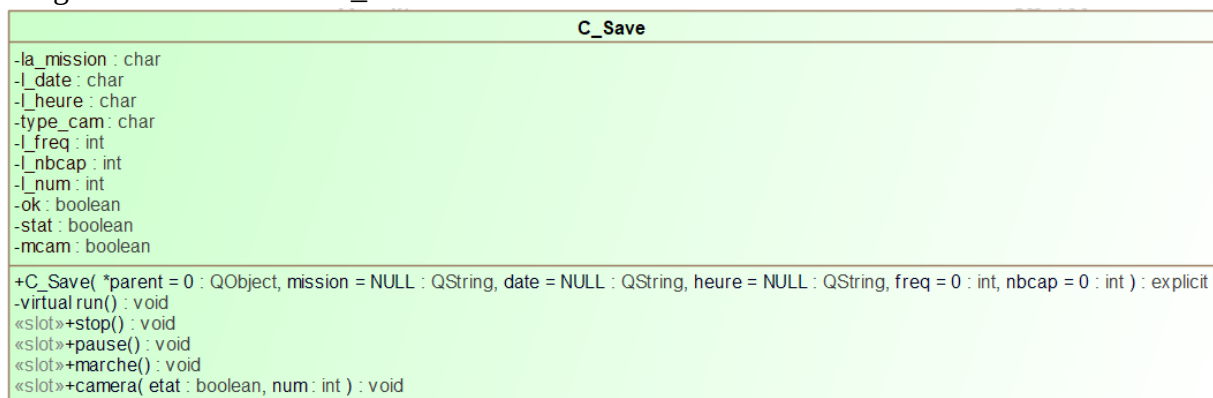
```
mshm->unlock();
```

Sauvegarde des mesures dans un fichier local

Un système de sauvegarde est très important, ce système est utilisé dans le but de garder une trace sûre des mesures prélevées car les mesures envoyées vers GCS pourront être mal transmises dû à des perturbations.

Pour gérer ce problème, j'ai conçu un système de sauvegarde de données à l'aide d'une classe « [C_Save](#) ». Cette classe est lancée en fond de tâche par un thread de façon à ce que la sauvegarde des données soit faite en continue et en simultanée avec le prélèvement des mesures, de plus cette classe enregistre à la fois les données de configuration. Les données sont sauvegardées dans deux fichiers .csv (Comma-separated values, fichier texte éditable) différent, dont un fichier pour les données des capteurs ainsi que de celui de la caméra (numéros de photo et vidéo) et l'autre pour les paramètres de l'avant mission et la durée de la mission.

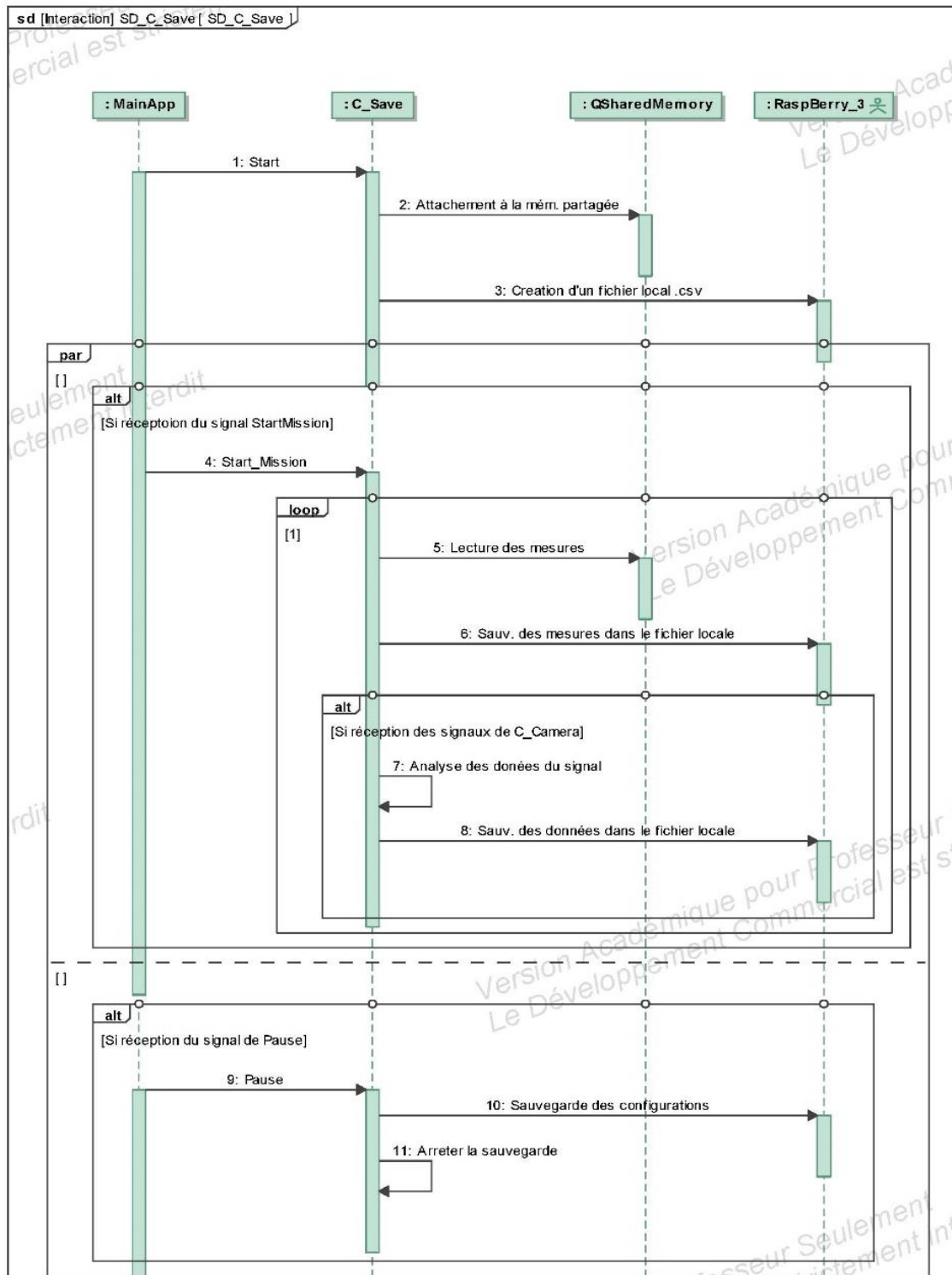
Diagramme de classe : « C_Save »



Ce diagramme représente la composition de ma classe de sauvegarde « C_Save ».

J'ai développé par la suite un diagramme de séquence qui illustre le scénario du fonctionnement de la classe « C_Save » depuis son lancement jusqu'à la fin de celui-ci : « SD_C_Save »

Voir annexe p61 : « SD_C_Save »



Communication entre processus

Le logiciel EDD est composé de plusieurs processus, il est donc primordial de mettre en place un système de communication entre processus.

Pour pouvoir faire communiquer les processus entre eux, j'utilise des émissions de signaux à l'aide des méthodes de l'environnement QT :

« c_capteur...h » : méthode du signal

```
signals:
    void sigAnalyse(bool, QString);
```

« c_capteur...h » : envoi du signal

```
emit sigAnalyse(stat, idCap);
```

Le signal est envoyé vers tous les processus (il réagit comme un « hub ») et il est intercepté par la classe mère « MainApp » par la fonction « connect() », puis cette fonction exécute une fonction particulière dans la classe actuelle ou dans une autre classe.

```
connect(capteurs.at(i), SIGNAL(sigAnalyse(bool, QString)), com, SLOT(capDefaux(bool,QString)));
```

Réception du signal
de l'objet souhaitée

Exécution d'une fonction
de la classe souhaitée

Ces signaux peuvent transmettre leur(s) paramètre(s) à la fonction souhaitée.

```
connect(capteurs.at(i), SIGNAL(sigAnalyse(bool, QString)), com, SLOT(capDefaux(bool,QString)));
```

Les paramètres sont transmis du signal vers la fonction

Gestion de la caméra

Parmi les équipements du drone, une caméra GoPro héro 3+ en fait partie. Le développement d'une classe pour la GoPro permettra d'en prendre le contrôle.

La GoPro héro 3 + est équipé d'un système de communication wifi. La carte Raspberry communique avec elle par wifi à travers le protocole de communication http. Pour prendre le contrôle de la caméra, j'ai développé une classe « C_Caméra » qui permet d'envoyer des requêtes http vers la caméra.

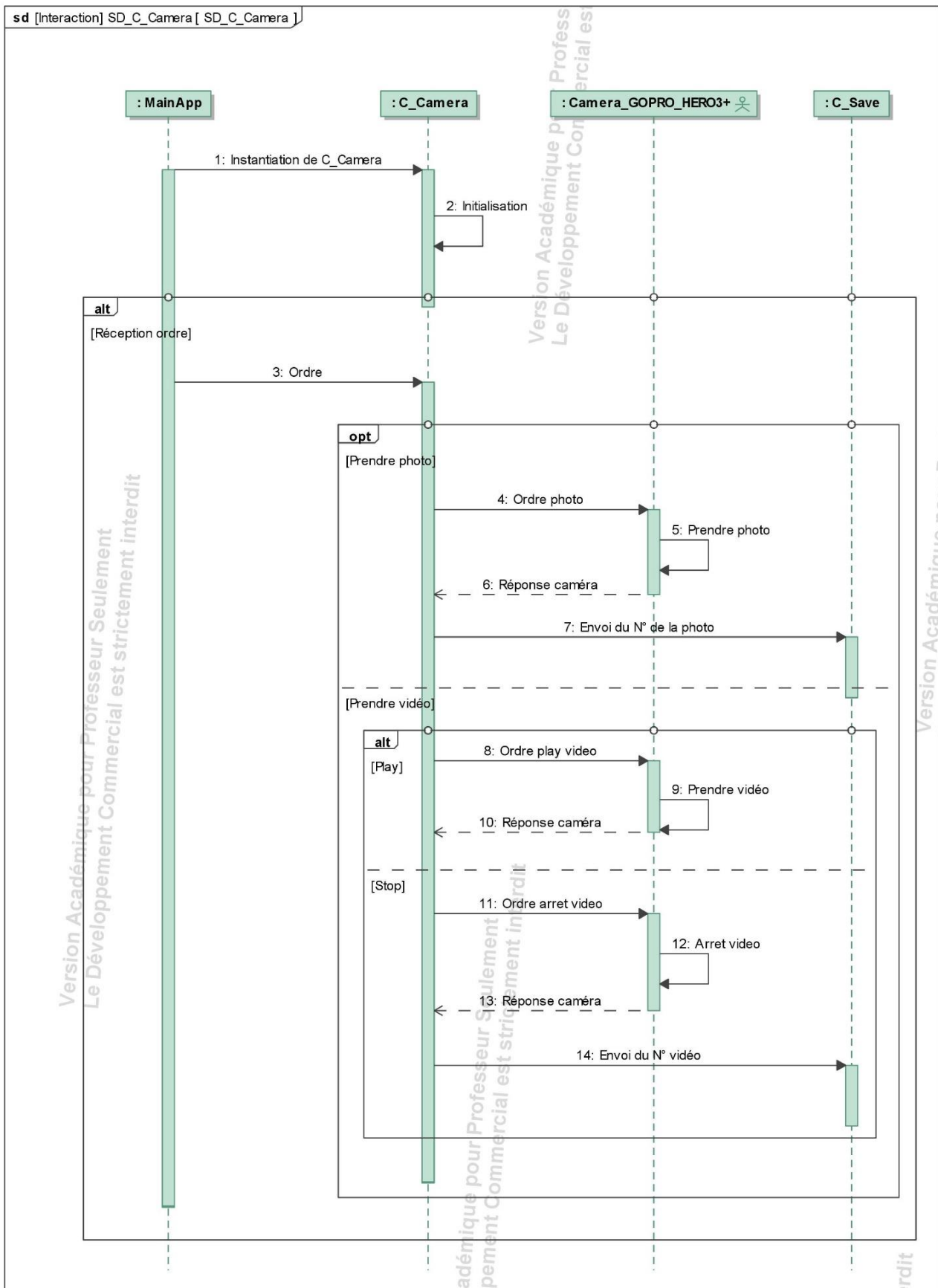
```
#define CAM_DEMARRER "GET /bacpac/PW?t=goprohero&p=%01 HTTP/1.1\r\n\r\n"
#define CAM_ARRETER "GET /bacpac/PW?t=goprohero&p=%00 HTTP/1.1\r\n\r\n"
#define CAM_MODE_PHOTO "GET /camera/CM?t=goprohero&p=%01 HTTP/1.1\r\n\r\n"
#define CAM_MODE_VIDEO "GET /camera/CM?t=goprohero&p=%00 HTTP/1.1\r\n\r\n"
#define CAM_ORDRE_PLAY "GET /bacpac/SH?t=goprohero&p=%01 HTTP/1.1\r\n\r\n"
#define CAM_ORDRE_STOP "GET /bacpac/SH?t=goprohero&p=%00 HTTP/1.1\r\n\r\n"
```

REQUETE : « GET /bacpac/PW?t=goprohero&p=%01 HTTP/1.1\r\n\r\n »

Une fois avoir pris un cliché Photo ou Vidéo, le numéro du cliché est automatiquement envoyé à l'aide d'un signal vers le thread de sauvegarde de données :

```
emit cam(true, numPh);
```

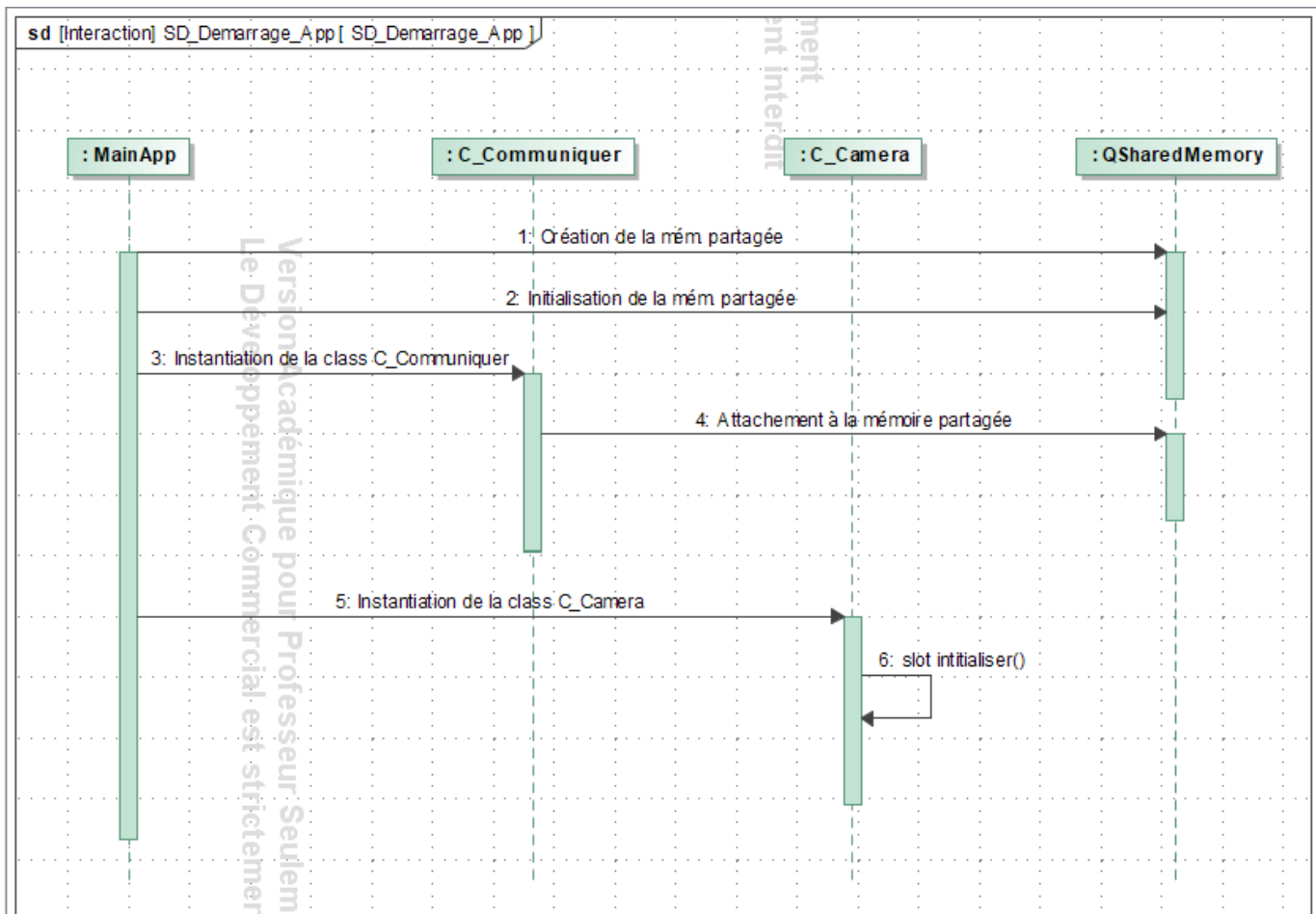
J'ai développé par la suite un diagramme de séquence qui illustre le scénario du fonctionnement de la classe « C_Camera » depuis son lancement jusqu'à la fin de celui-ci : « SD_C_Camera »



Amélioration supplémentaire

Durant la deuxième itération, j'ai consacré une partie de mon temps de travail à faire évoluer le logiciel EDD, je lui ai donc apporté certaine optimisation.

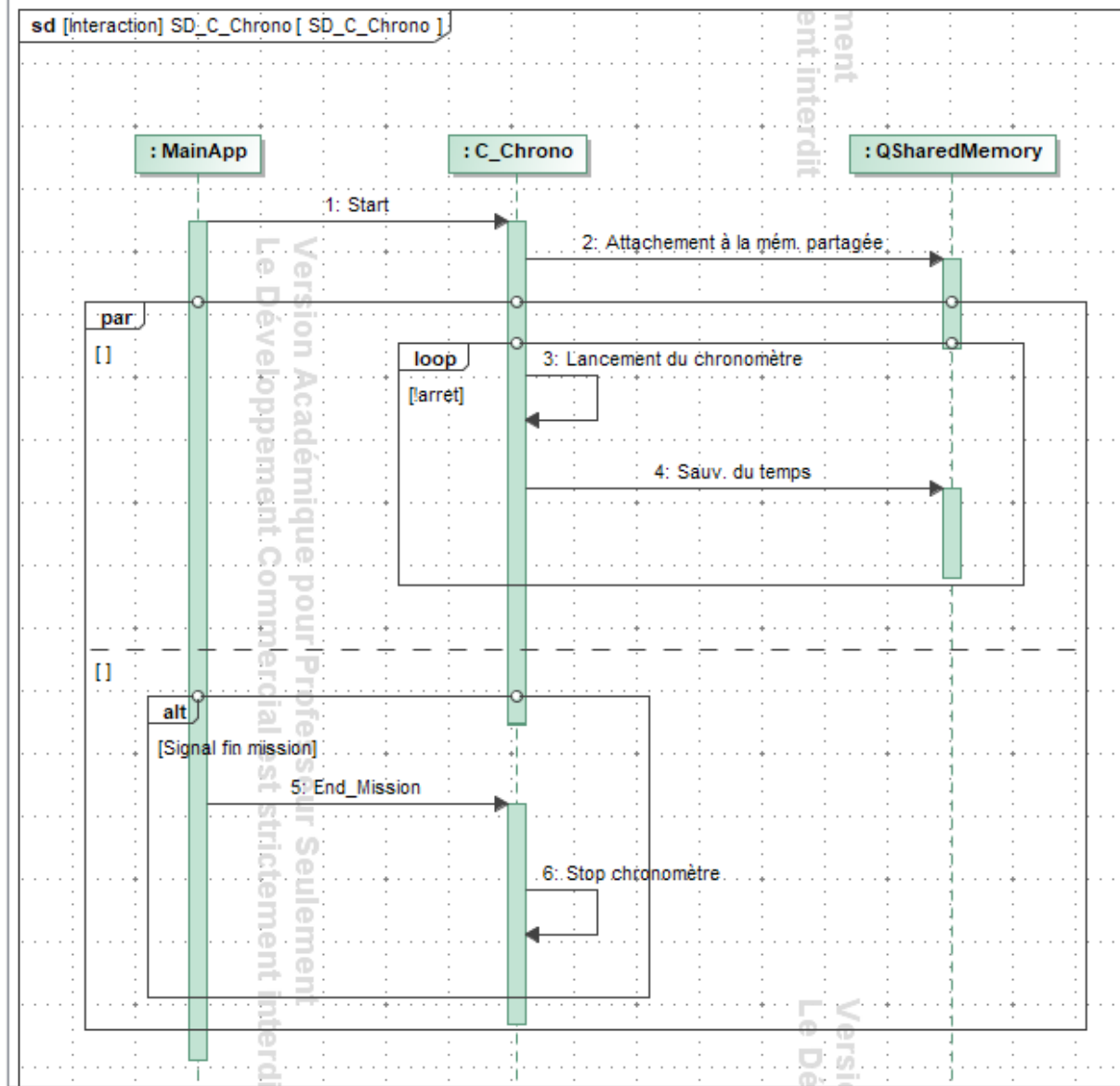
Pour commencer, lors du démarrage du logiciel, le logiciel est capable d'activer la caméra (de la mettre en mode opérationnel). Ci-dessous représente le diagramme de séquence du scénario lors du démarrage : « SD_Démarrage_App »



Lors de l'initialisation de la classe caméra, le système de gestion de la caméra active la caméra.

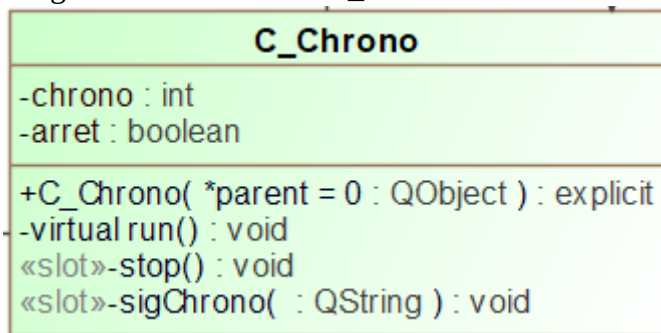
Ensuite, j'ai développé un système de chronométrage qui va permettre de déterminer le temps d'une mission mais aussi d'indiquer à quelle période une mesure, photo ou vidéo a été faite. Il s'agit d'un thread qui va utiliser une classe que j'ai conçue : la classe « C_Chrono ». Le thread agit comme cela :

Diagramme de séquence : « SD_C_Chrono »



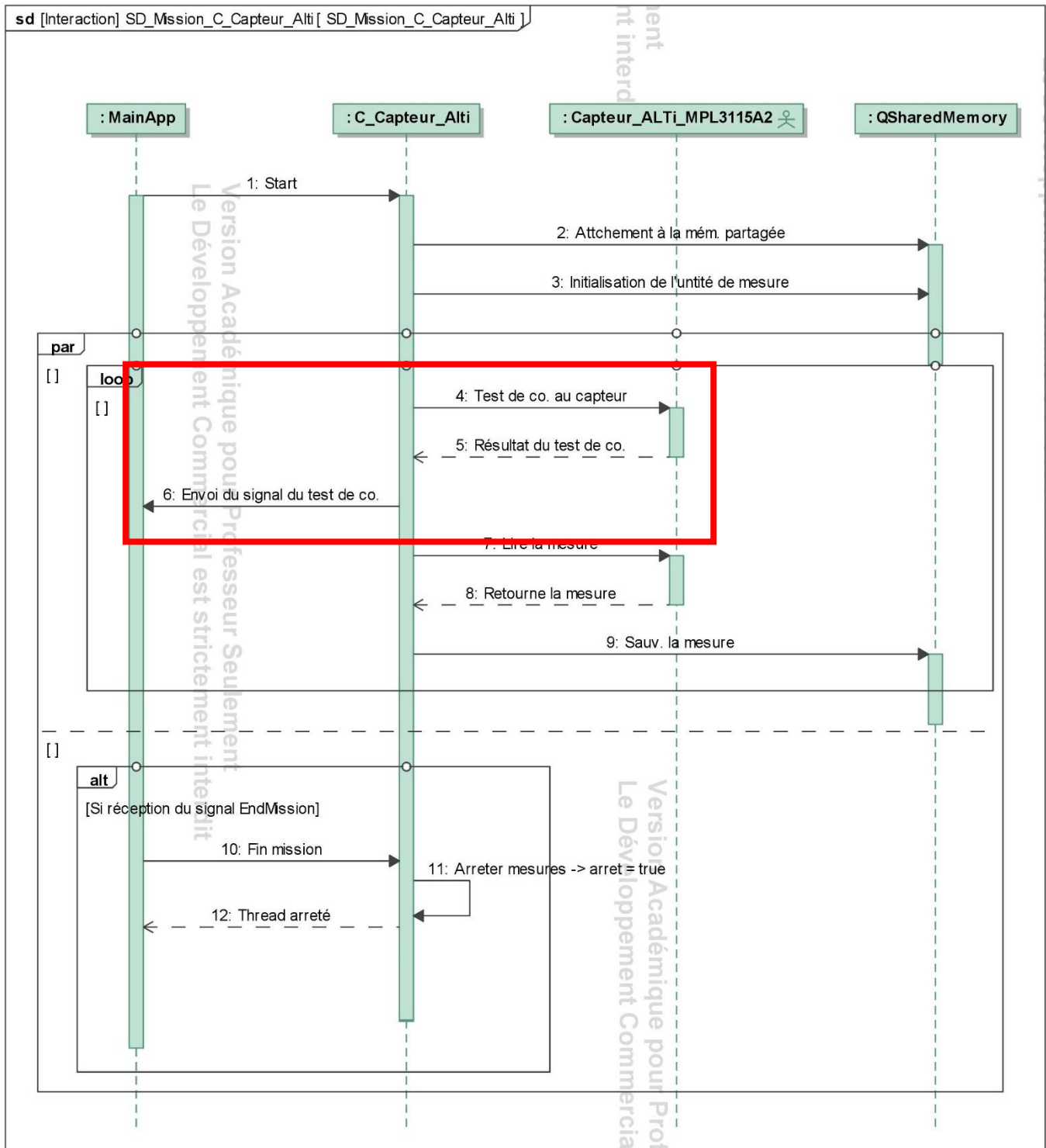
Le système de chronométrage est représenté comme ceci :

Diagramme de classe : « C_Chrono »



Après le système de chronométrage, j'ai optimisé la surveillance des capteurs, c'est-à-dire, j'ai mis au point un système de surveillance des capteurs. Pour cela j'ai développé dans chaque classes « C_Capteur » un moyen de test de connexion vers leur capteur.

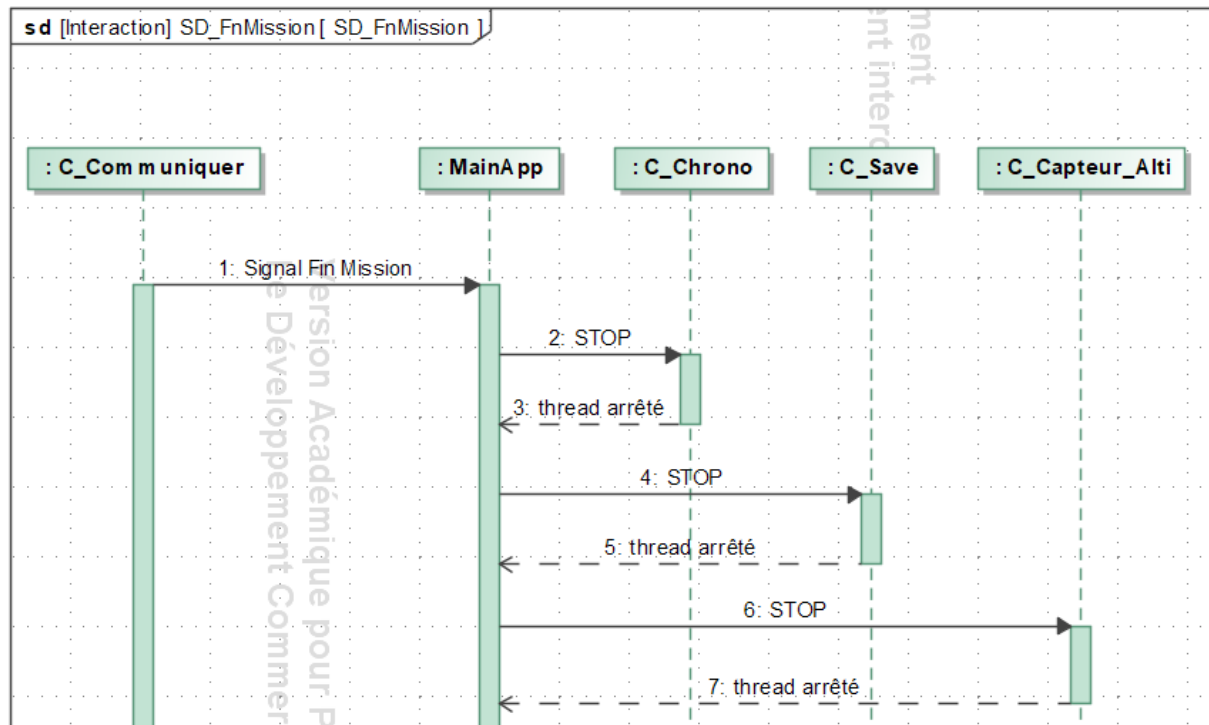
Digramme de séquence, scénario d'un thread « C_Capteur » : « SD_C_Alti »



L'encadré en rouge représente le système de test de connexion au capteur. Ce système envoie en permanence l'état du capteur à GCS. Si le capteur n'est plus reconnu durant le fonctionnement du thread, les valeurs enregistrées sont représentées par « XXX ».

Pour finir, lors de la réception du signal de fin mission, le logiciel met en arrêt tous les threads lancés : scénarii de fin mission « SD_FinMission »

Voir annexe p62 : « SD_FinMission »



Ci-dessus, ce diagramme de séquence est un fragment du diagramme de séquence « SD_FinMission ».

Cross-Compilation

La cross-compilation est une méthode de compilation très efficace, elle permet de développer un fichier exécutable sur une machine ciblée dans le but de gagner du temps lors de la compilation d'un programme.

Pour cela, j'ai effectué une cross-compilation entre un pc linux et une carte Raspberry pi 3.

Pour effectuer l'installation de cette cross-compilation, j'ai suivi un tuto sur internet :

« <http://wiki.qt.io/RaspberryPi2EGLFS> ».

Après plusieurs tests, j'ai pu établir la cross-compilation Linux → Raspberry. Seul le fichier exécutable s'installe sur la carte Raspberry.

Intégration des parties EDD

Durant la phase finale, j'ai pour rôle d'assembler toutes les parties du logiciel EDD. Jusqu'à présent, je n'ai pu que réunir le programme d'un EC, un programme contrôlant le capteur température et d'humidité car mes coéquipiers n'ont pas pu terminer leur programme en C++.

Simulation du logiciel EDD

Etant donné que la collaboration avec mon collègue de travail au sujet de la partie communication avec GCS n'a pas aboutie, j'ai développé une interface de simulation afin de pouvoir faire fonctionner le logiciel EDD. Il s'agit d'une interface de simulation qui gère la partie communication, plus précisément de la communication avec GCS (système réceptionnant les ordres émis par GSC, puis générant des signaux précis selon l'ordre reçu).

Cette simulation est divisée en trois parties, la première partie, pour les actions de l'avant mission, puis la deuxième partie, pour les actions du pendant mission et la dernière partie, pour les actions de fin mission.

Première partie :

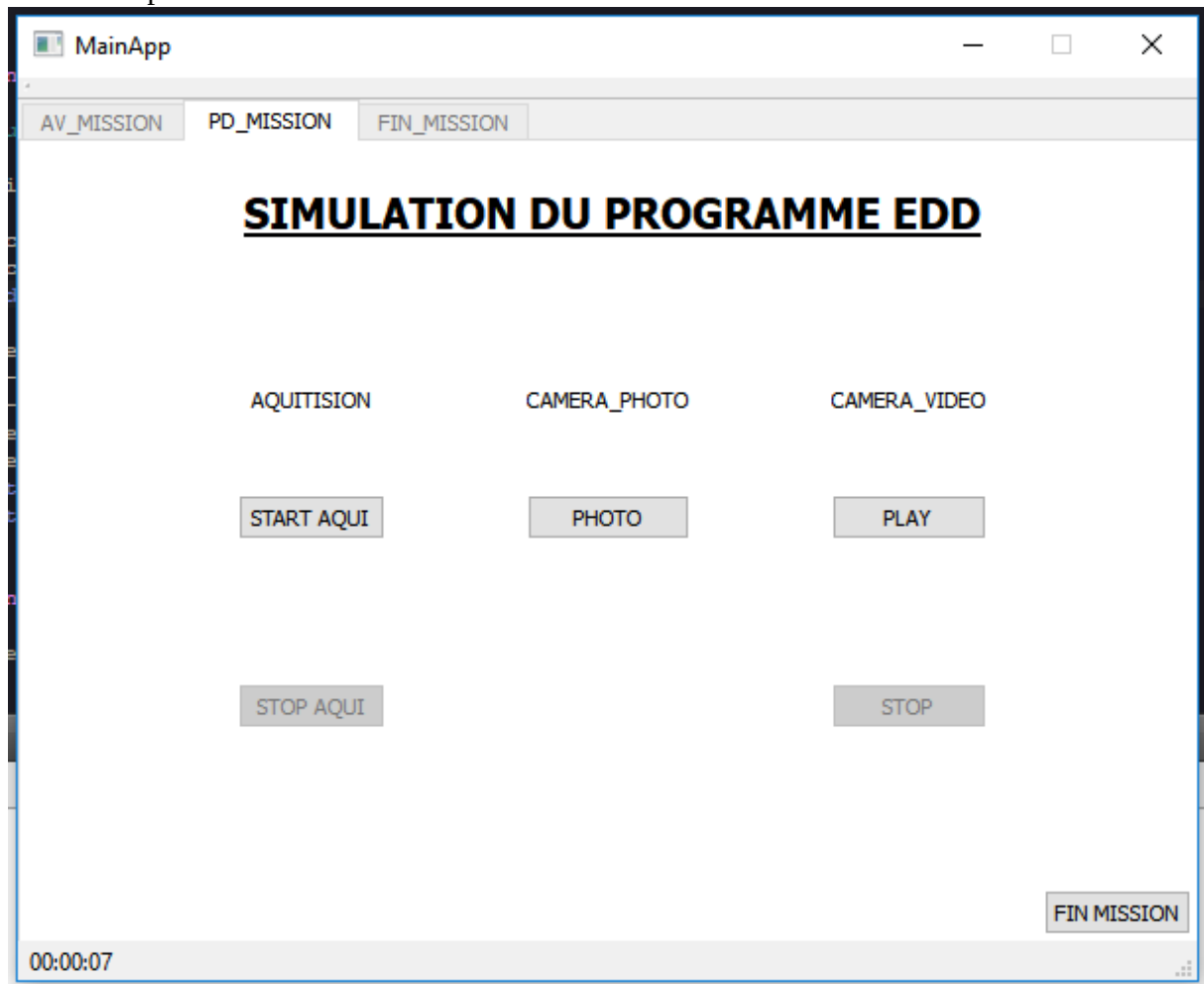
The screenshot shows a software window titled 'MainApp' with three tabs: 'AV_MISSION' (selected), 'PD_MISSION', and 'FIN_MISSION'. The main content area is titled 'PARAMETRE AV_MISSION' in bold, underlined text. Below the title, there are several input fields and controls:

- 'Nom Mission*': A text input field.
- 'Envoi des mesures à GCS*': Two radio buttons labeled 'Oui' and 'Non'.
- 'Fréquence d'acquisition (sec)*': Three radio buttons labeled '0.5', '1', '1.5', and '2'.
- 'Capteurs*': A list box containing a list of sensors and their IDs, including 'CAPTEUR_ALTI', 'CAPTEUR_COMP', 'CAPTEUR_DIST', 'CAPTEUR_GPS', 'CAPTEUR_HUMI', and 'CAPTEUR_TEMP'.
- 'Capteurs sélectionnés': An empty rectangular box for selecting sensors.
- 'Envoyer': A button to send the data.
- 'DEPART MISSION': A button to start the mission.

There is also a scroll bar at the bottom of the list box and a double arrow '==>' pointing from the list box to the 'Capteurs sélectionnés' box.

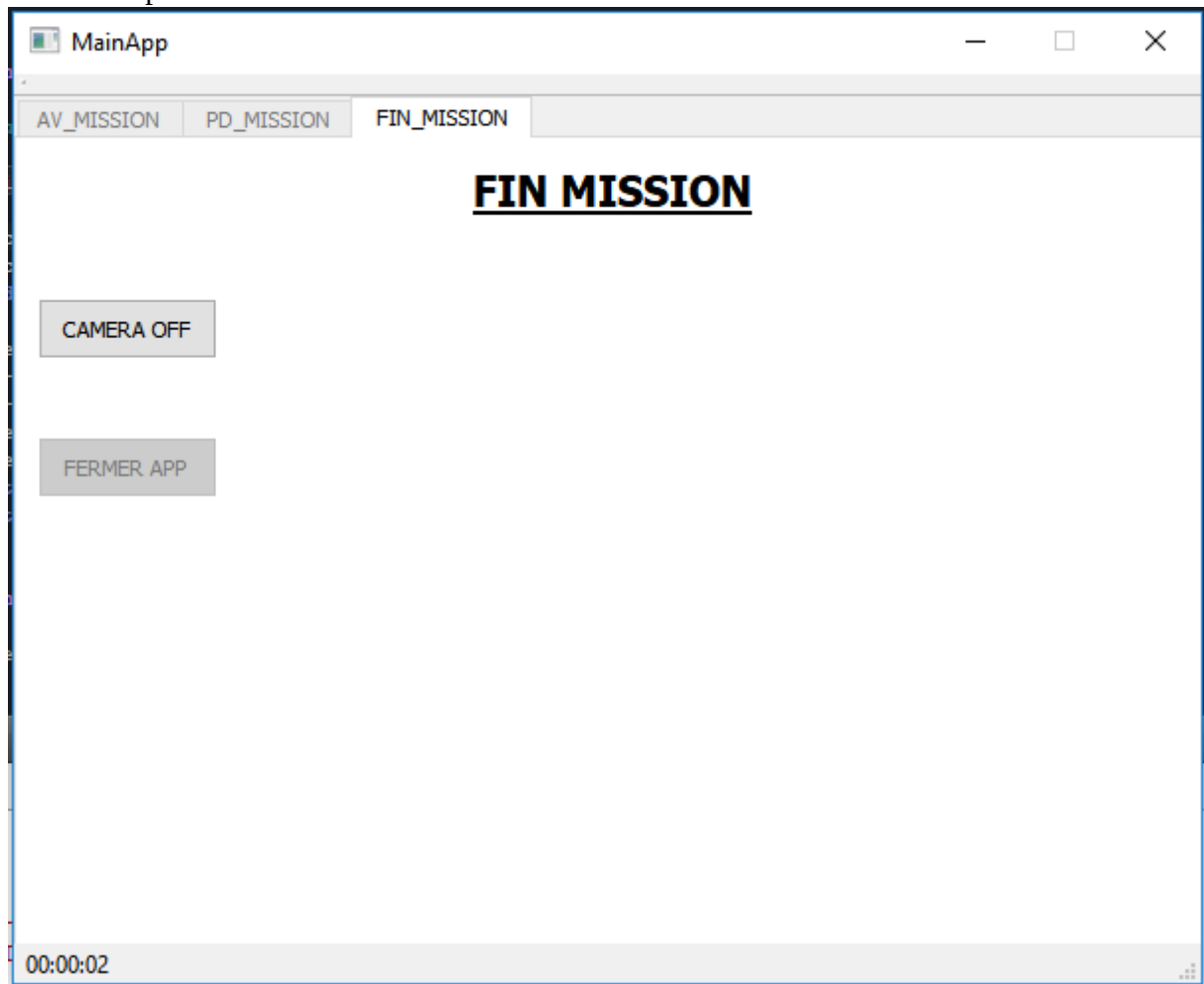
Cette interface permet d'envoyer à EDD les paramètres de la mission durant la phase de l'avant mission.

Deuxième partie :



Cette interface permet de simuler l'envoi des ordres GCS à EDD durant la phase du pendant mission.

Troisième partie :



L'interface ci-dessus représente la phase de fin mission. Elle permet de mettre la caméra en mode veille ainsi que de quitter le logiciel EDD.

Optimisation du système EDD envisageable

Le système EDD est constamment en évolution. Après avoir réalisé toutes les demandes du cahier des charges, des optimisations sont envisageables.

Un point important peut-être amélioré ou voir même développé, le système de supervision sur les capteurs. Pour le moment, ce système est actif et opérationnel mais il reste un système que je trouve « primitif ».

Une autre option peut-être développé, un système de contrôle sur les batteries du drone, car ce sont-elles qui alimentent tout le système EDD.

Un système de bras mécanique équipé de capteurs permettant de prélever les mesures avec une précision maximale est aussi envisageable. Ce qui sous-entend, le développement d'un nouveau système permettant de prendre le contrôle du bras mécanique, et de récupérer les données des capteurs présent sur le bras métallique.

Analyse physique I2C

Pour ma partie physique du projet, je me suis attribué la tâche d'analyser un échange de trame sur le bus de communication I2C entre la carte Raspberry et un capteur.

Définition du bus I2C

Le bus I2C (Inter Integrated Circuit) fait partie des bus série : 2 fils pour transmettre les données. Il a été développé au début des années 1980 par Philips pour minimiser et faciliter les liaisons entre les circuits intégrés numériques de ses produits (Téléviseurs, éléments HiFi, magnétoscopes, ...).

Caractéristique

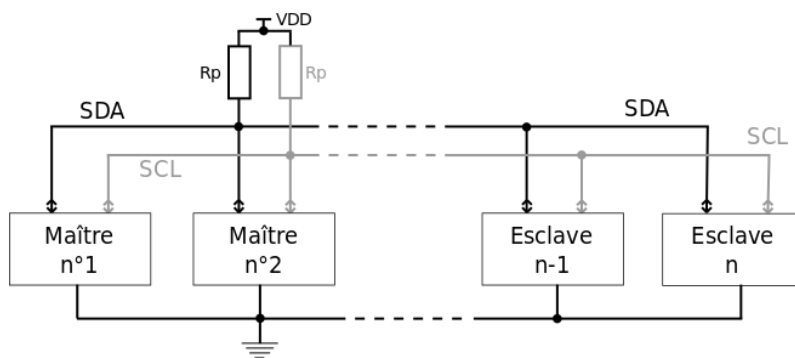
I2C est un bus série synchrone bidirectionnel half-duplex, où plusieurs équipements, maîtres où esclaves, peuvent être connectés au bus.

Les échanges ont toujours lieu entre un seul maître et un (ou plusieurs) esclave(s), toujours à l'initiative du maître.

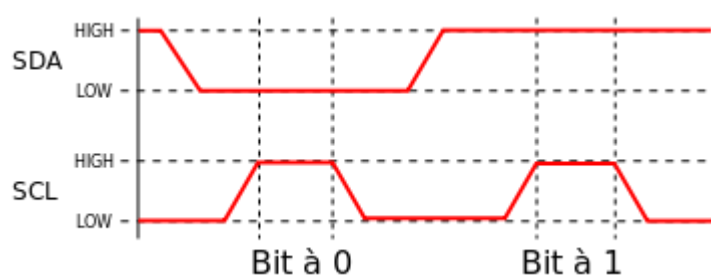
La connexion est réalisée par l'intermédiaire de deux lignes :

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

Il ne faut également pas oublier la masse qui doit être commune aux équipements.



Le niveau (« HIGH » ou « LOW ») de la ligne SDA doit être maintenu stable pendant le niveau « HIGH » sur la ligne SCL pour la lecture du bit.

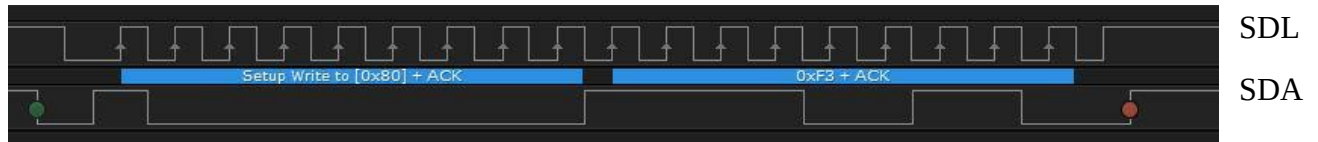


Dans notre cas, l'état logique 1 correspond à une tension de 3.3 V et pour l'état logique 0 à une tension de 0 V.

Analyse personnelle

Parmi les équipements du drone, le capteur de température SI7021 communique avec la carte Raspberry à l'aide du bus I2C. J'ai donc analysé les échanges de données avec le logiciel « SALEAE ».

Je récupère dans un premier temps ceci :



Ci-dessus, à partir du point vert au point rouge, correspond à la requête envoyée par le maître (Raspberry) vers l'esclave (capteur de température), cette requête permet de récupérer la température.

La requête est composée de cette manière suivante :

- 2 bits de stop
- 1 bit de repos
- 8 bits de données A
- 1 bit de repos
- 8 bits de données B
- 2 bits de stop



Données A :

$(1000\ 0000)_2 \rightarrow (80)_{16}$

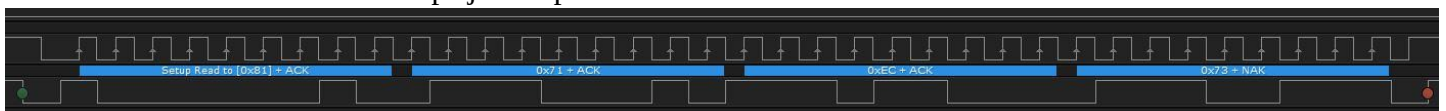
Les données A correspondent à l'adresse de l'esclave 0x80.

Données B :

$(1111\ 0011)_2 \rightarrow (F3)_{16}$

Les données B correspondent à l'ordre demandé 0xF3 (demande de la température).

Puis dans un second temps je récupère ceci :



La requête si dessus correspond à la réponse du capteur. Elle est composée de la manière suivante :

- 2 bits de stop
- 1 bit de repos
- 8 bits de données A
- 1 bit de repos
- 8 bits de données B
- 1 bit de repos
- 8 bits de données C

- 1 bit de repos
- 8 bits de données D
- 2 bits de stop

Les données A :

Adresse destinée au maître.

Les données B et C :

Ces données corresponde aux données de la température.

Les données D :

Réponse de l'ordre demandé.

Conclusion

J'ai pu atteindre les objectifs que je me suis donnés, le logiciel EDD que j'ai développé est conforme aux charges du cahier des charges. J'ai pu apporter certaine optimisation au logiciel non demandé dans le cahier des charges, ainsi que d'apporter des optimisations envisageables.

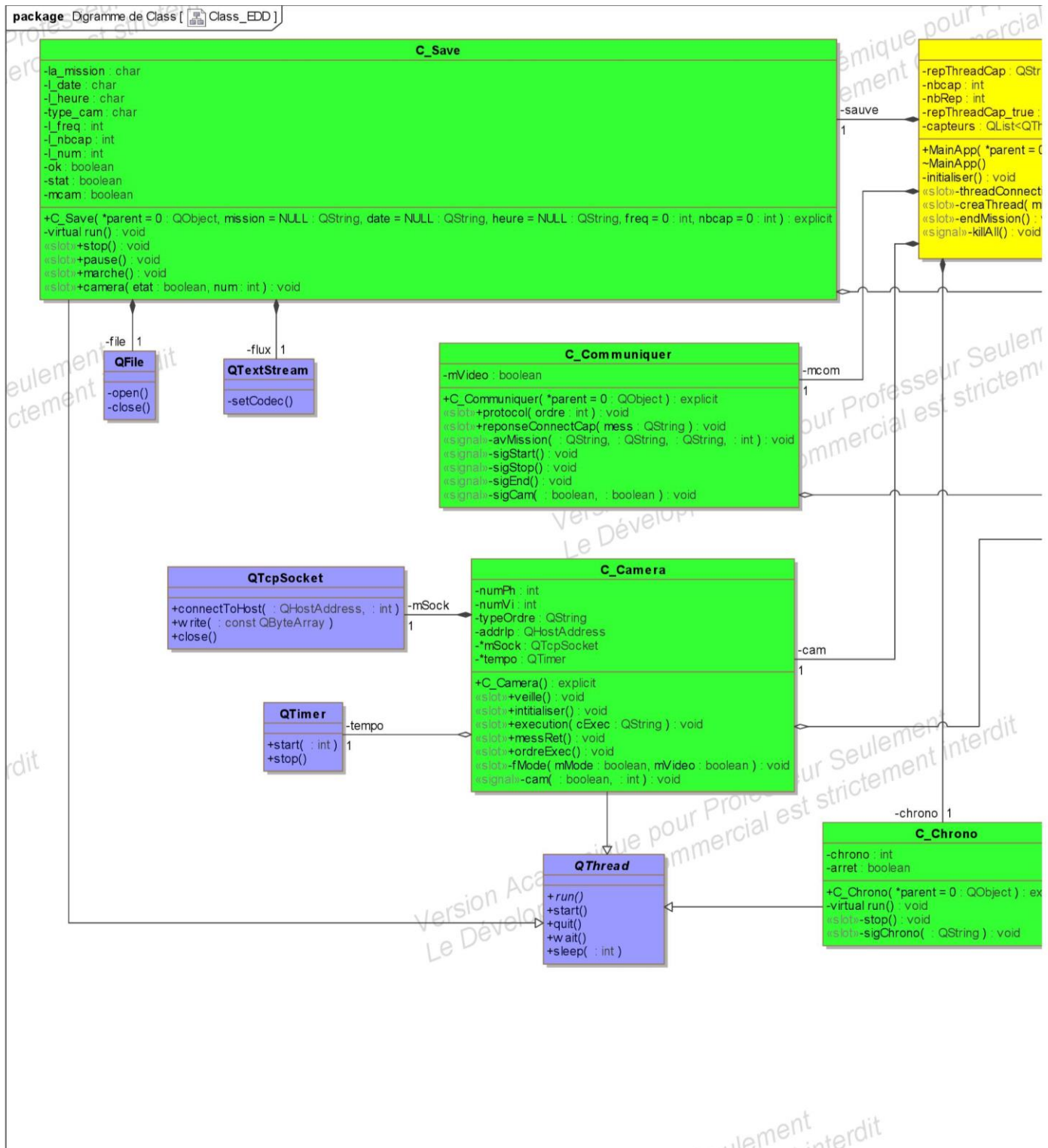
Par contre je n'ai pas pu intégrer la partie communication avec GCS géré par mon collègue IR, de même pour la partie incrustation vidéo de la caméra et ainsi que certain développement des classes « C_Capteur » développé par mes collègues EC (Capteur de distance, capteur de géolocalisation, capteur de position). Les parties manquantes sont développées en C, c'est pourquoi je n'ai pas pu intégrer leurs parties au logiciel, car en effet le logiciel est développé en C++.

Durant tout le projet, j'ai travaillé en collaboration avec l'équipe EDD mais aussi avec l'équipe GCS. Afin de mettre au point le bon fonctionnement du logiciel EDD et GCS, j'ai élaboré des réunions de groupe de projet. Etant donné que l'on m'a donné le titre de chef d'équipe 1, j'ai aidé et conseillé mes collègues de travail tout au long du projet.

Ce projet m'a apporté énormément de connaissance sur le développement en C++ de l'environnement QT. Ainsi que sur le rôle de chef de projet.

Annexes

Diagramme de Classe « Class EDD » :



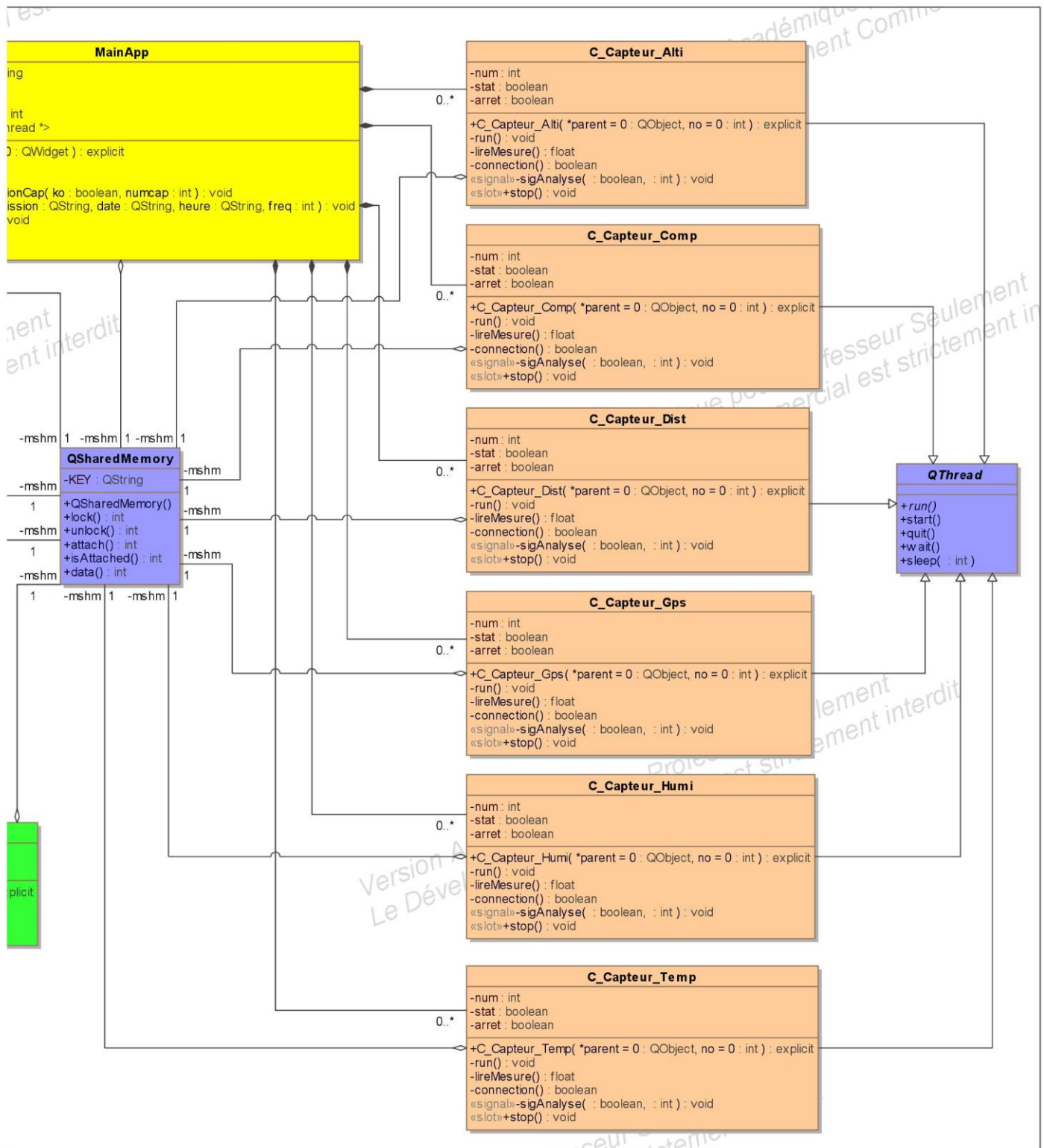


Diagramme de séquence : « SD Thread »

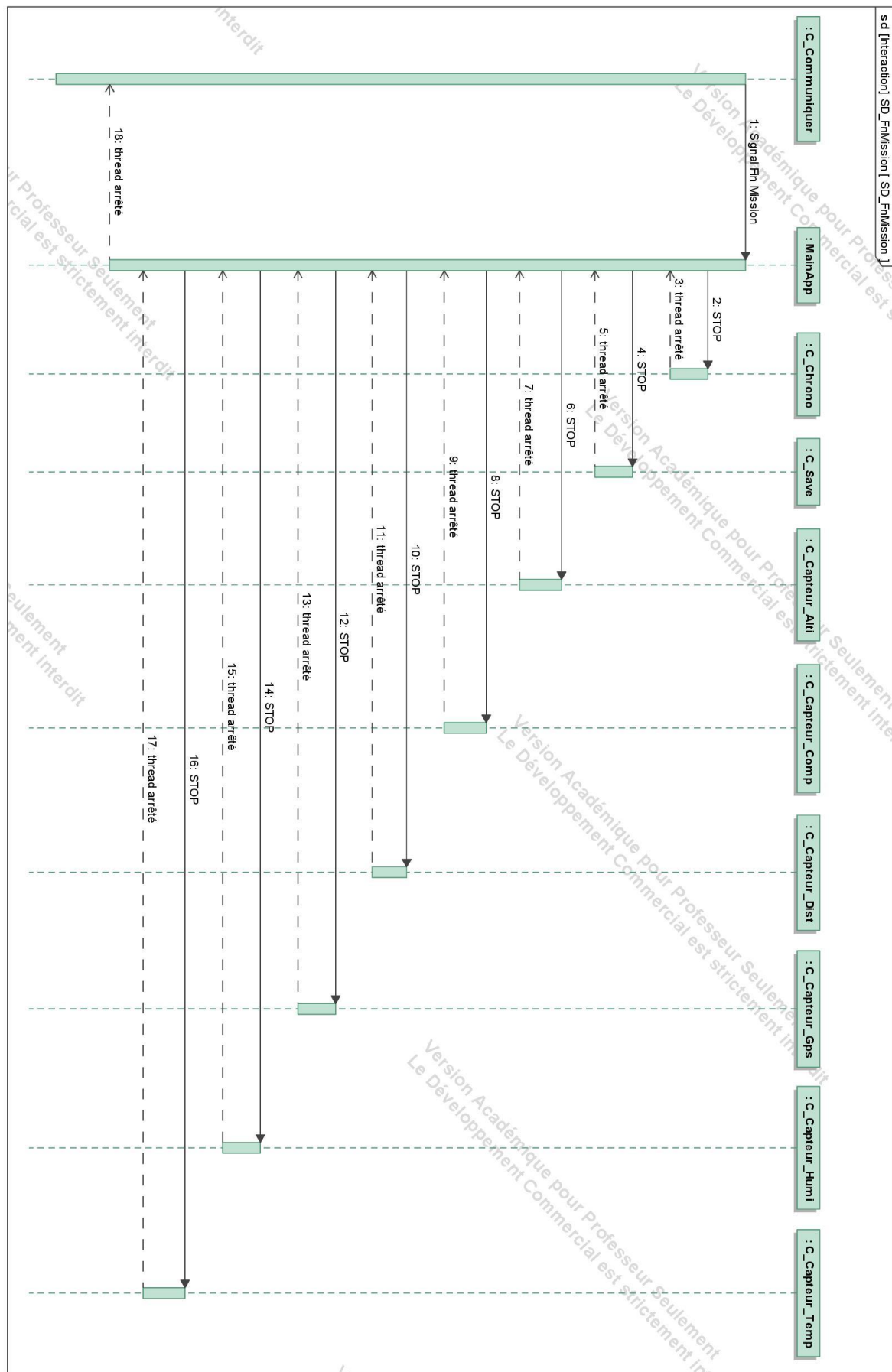


Diagramme de séquence : « SD C Save »

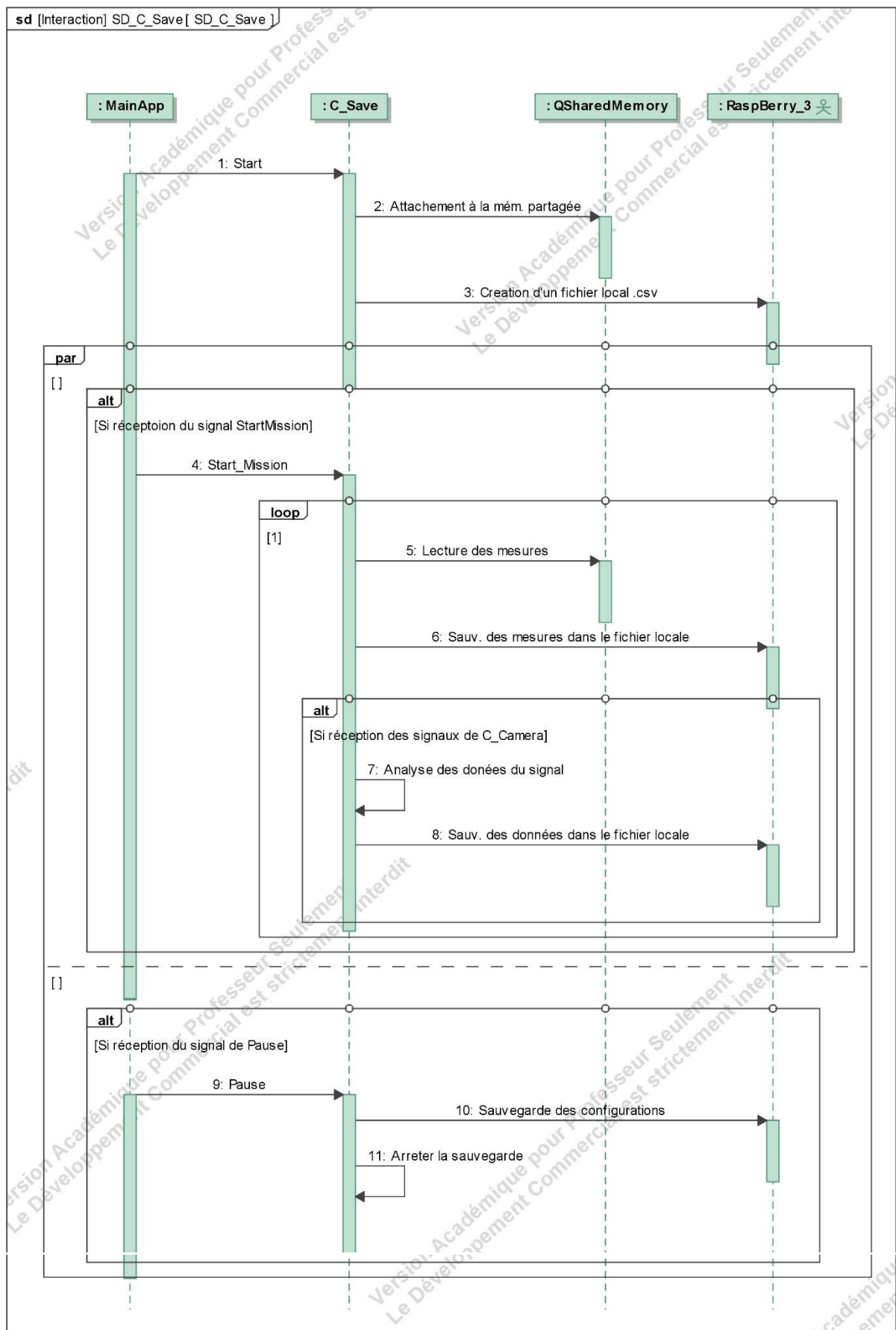
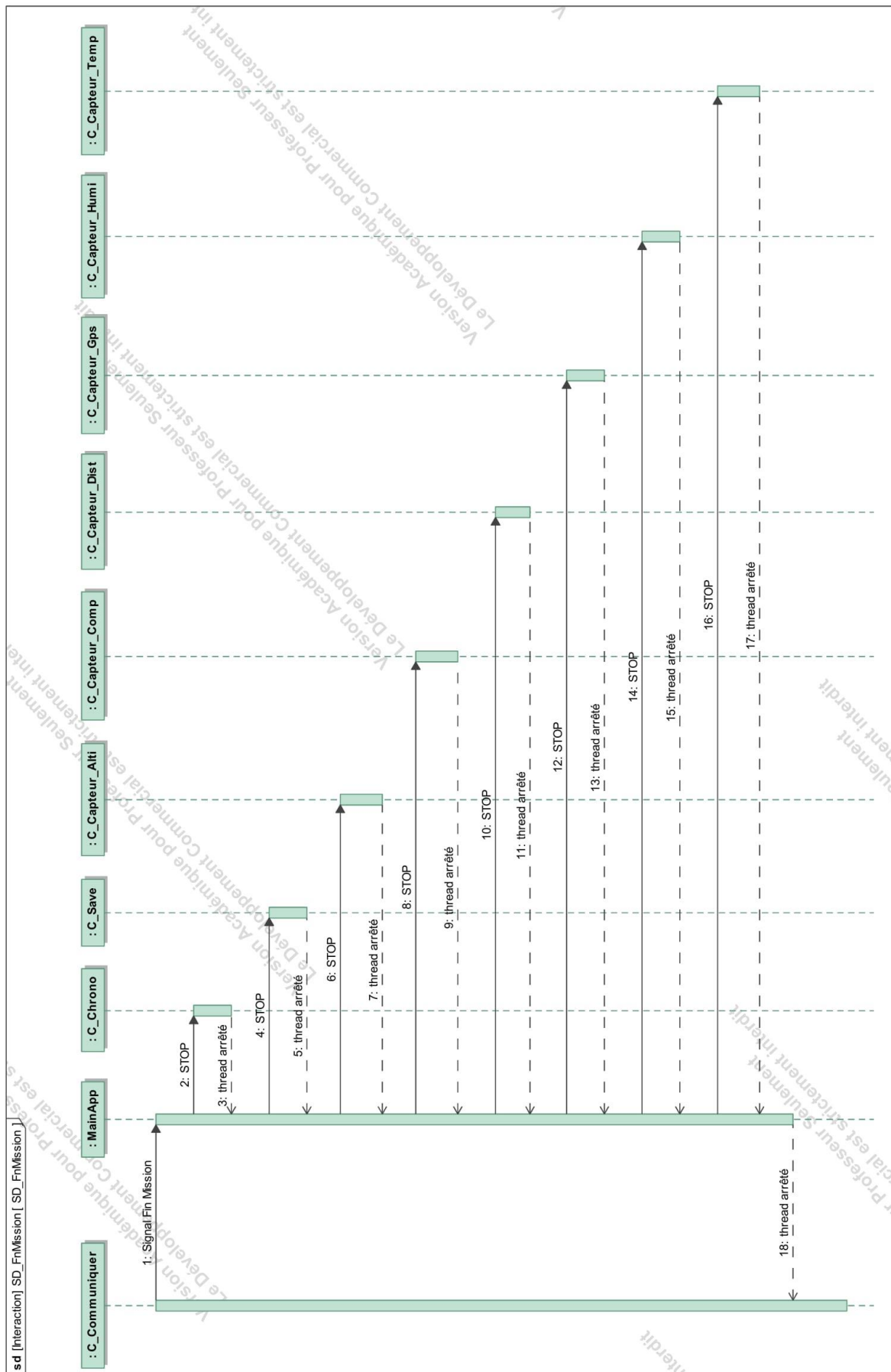


Diagramme de séquence : « SD_FinMission »

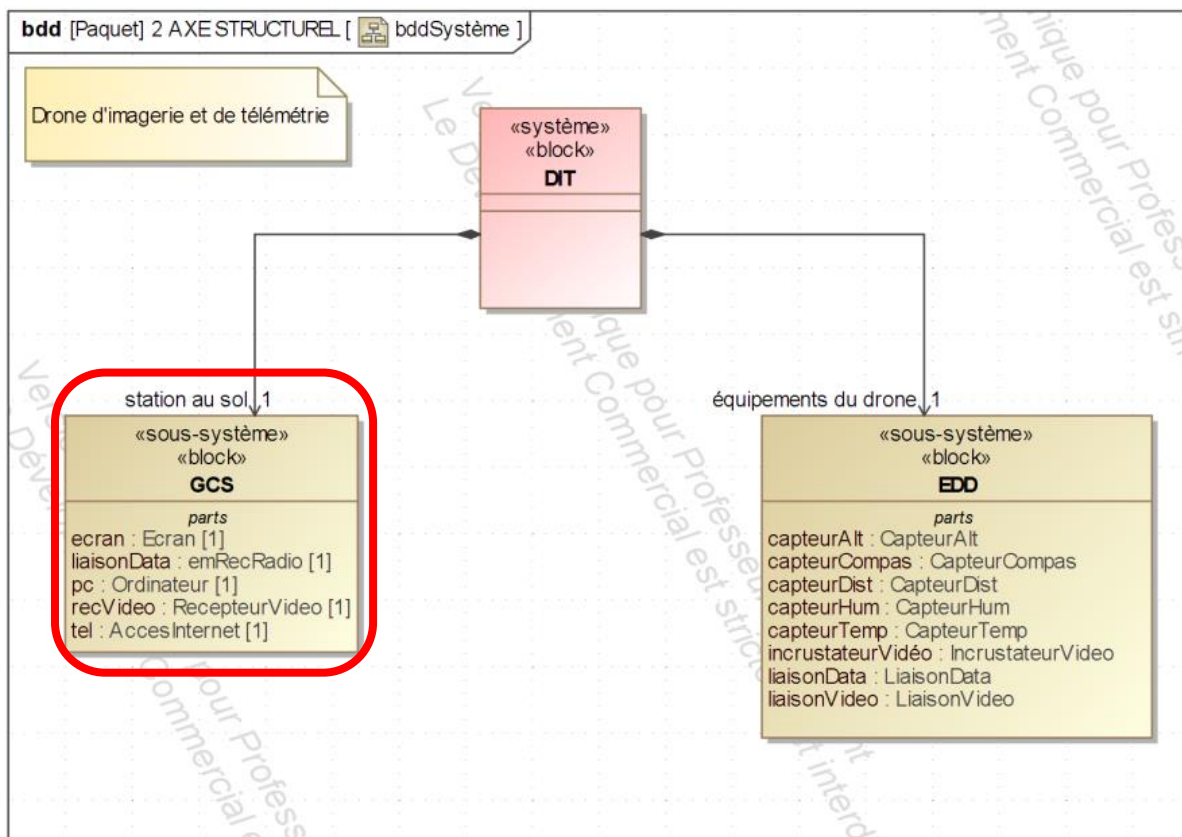
IV – IR : HISTORISATION ET SUPERVISION

Présentation personnelle

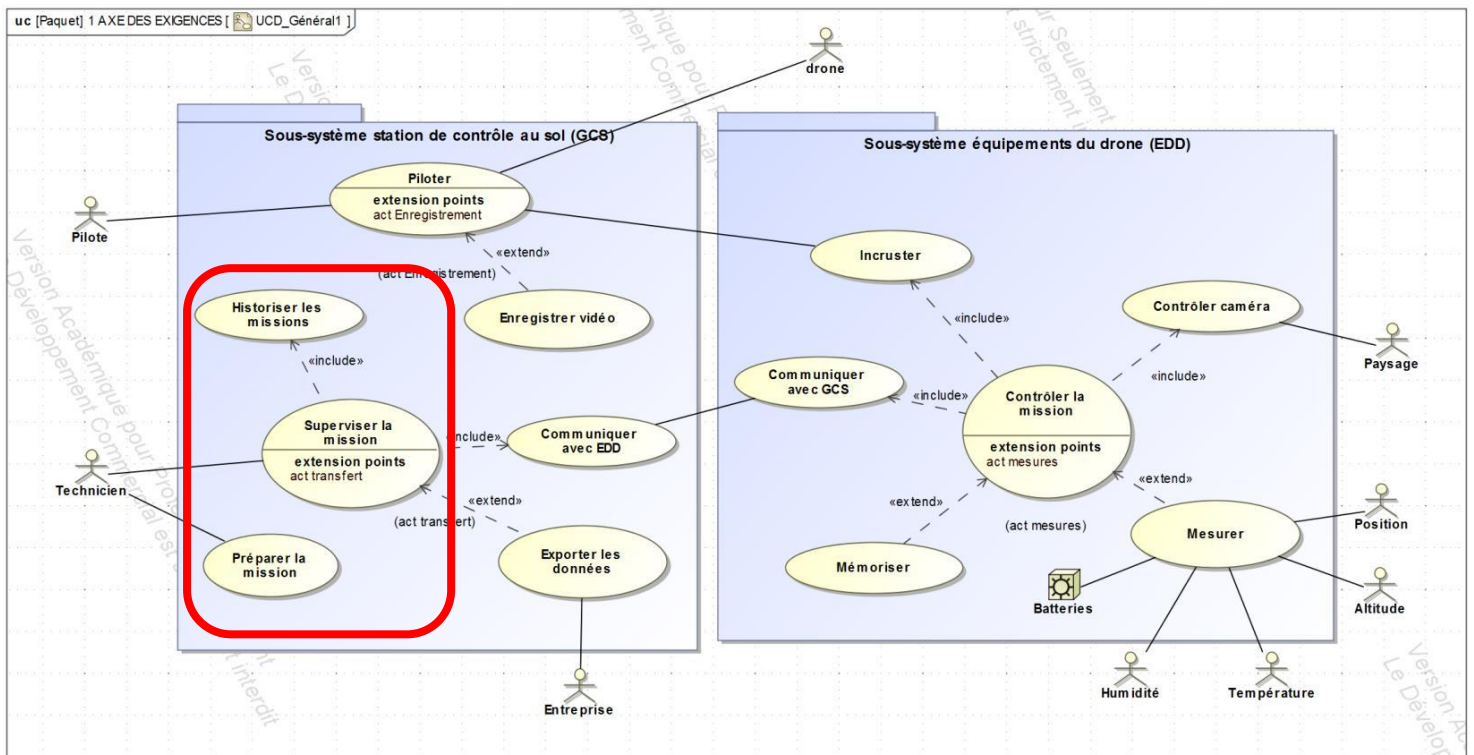
Dans le cadre de ma formation en BTS SN IR au lycée Alphonse Benoît, je réalise durant ma deuxième année de BTS, un projet industriel. Ceci me permet de mettre en pratique des connaissances théoriques apprises durant ma formation.

Ce projet technique, en collaboration avec quatre autres étudiants de la section IR (COLUS Julia, HOUTTEMANE Nicolas, MAUBERNARD Kevin et THO Yann) et trois étudiants de la section EC (BONNEFILLE Thomas, GRIMOUD Yohan et MEDEIROS Vincent), consiste en la surveillance de l'état d'édifices et d'ouvrages. Le principe du projet sera d'utiliser un drone afin d'effectuer différentes mesures telles la température ou l'angle et de filmer pour ensuite pouvoir rapidement faire les réparations nécessaires.

En ce qui me concerne, j'ai été chargé de rendre possible l'historisation des mesures et des missions, grâce à une base de données (IHM), et de préparer, superviser et terminer les missions depuis la station au sol (voir diagramme ci-dessous).

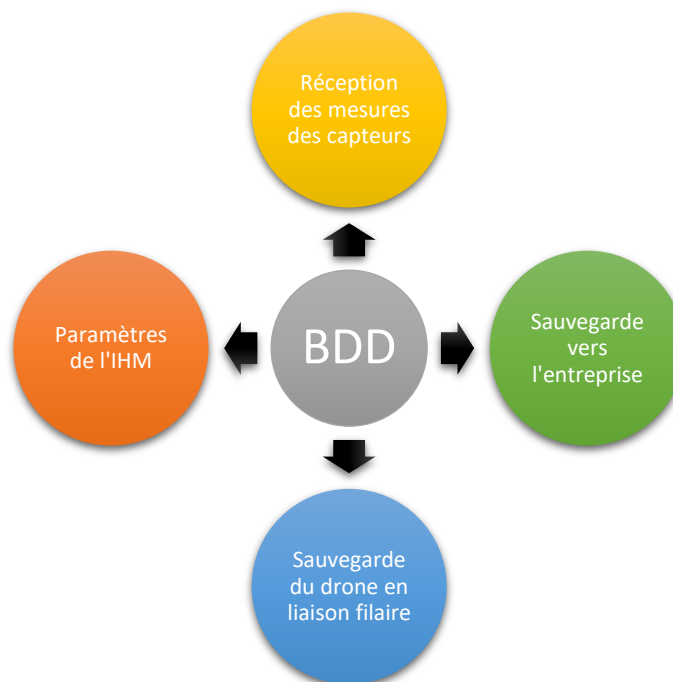


Ma partie concerne les blocs encadrés en rouge dans le diagramme de cas suivant :



Je dois donc mettre en œuvre une base de données qui permettra de gérer l'historisation des missions, des mesures, mais aussi coder une application pour contrôler et superviser l'acquisition des mesures, notamment.

Voici un schéma simple qui explique les différents rôles de la base de données :



Au début du projet, j'ai élaboré une planification personnelle prévisionnelle des tâches à accomplir pour mener à bien le projet :

	Nom de la tâche ▼	Durée ▼	Début ▼	Fin ▼	Pr ▼
1	▲ Projet	31,71 jours	Lun 09/01/17	Lun 29/05/17	
2	▲ Analyse du projet, et planification prévisionnelle	4,43 jours	Lun 09/01/17	Lun 23/01/17	
3	Analyse du cahier des charges et choix de groupe	15 hr	Lun 09/01/17	Jeu 12/01/17	
4	Installation de tous les logiciels, VM...	12 hr	Lun 16/01/17	Jeu 19/01/17	3
5	Création du planing prévisionnel	4 hr	Jeu 19/01/17	Lun 23/01/17	4
6	▲ Itération 1 : BDD et classe TMP de gestion	8,43 jours	Lun 23/01/17	Jeu 02/03/17	5
7	Finalisation du diagramme de planification	7 hr	Lun 23/01/17	Mar 24/01/17	
8	Diagramme conceptuel de la BDD	14 hr	Jeu 26/01/17	Mar 31/01/17	7
9	Création de la BDD	23 hr	Mar 31/01/17	Jeu 09/02/17	8
10	Création classe de gestion basique de la BDD	22 hr	Sam 25/02/17	Jeu 02/03/17	9
11	Revue 1	9 hr	Lun 06/03/17	Lun 06/03/17	10
12	▲ Itération 2 :	13,14 jours	Mar 07/03/17	Jeu 04/05/17	11
13	IHM : préparation et supervision mission	30 hr	Mar 07/03/17	Lun 20/03/17	
14	Mise en oeuvre de l'historisation des missions	67 hr	Mar 21/03/17	Jeu 04/05/17	13
15	Revue 2	1 hr	Mar 09/05/17	Mar 09/05/17	14
16	▲ Itération 3 :	3,29 jours	Jeu 11/05/17	Mar 23/05/17	15
17	Réalisation du dossier, codage et tests unitaires	32 hr	Jeu 11/05/17	Mar 23/05/17	
18	Revue 3	8 hr	Ven 26/05/17	Lun 29/05/17	17

Dans cette planification n'ont pas été comptées les réunions, car non prévues, ni les problèmes rencontrés, notamment des machines virtuelles, malgré que ça ait eu de l'influence sur le temps de travail.

Travail réalisé

Présentation de la base de données (BDD)

Une des tâches de ma partie est d'assurer l'historisation des missions. Cela comprend, l'enregistrement de chaque mission, avec sa configuration, et de toutes les mesures faites. Cela permet d'analyser les données après la mission, ou de comparer à des missions antérieures.

Les données à stocker peuvent être représentée sous forme de différents tableaux, selon le type de données.

Exemples :

- Enregistrements des missions :

ID mission	date	heure début	durée	nom
1	2017/02/28	16:40	00:50	mission001
2	2017/05/23	11:02	00:58	mission TE
3	2017/05/18	17:59	01:33	mission003
4	2017/12/01	09:15	01:12	004
5	2018/01/20	14:33	00:39	TEST
6	2018/06/27	08:00	00:10	mission84

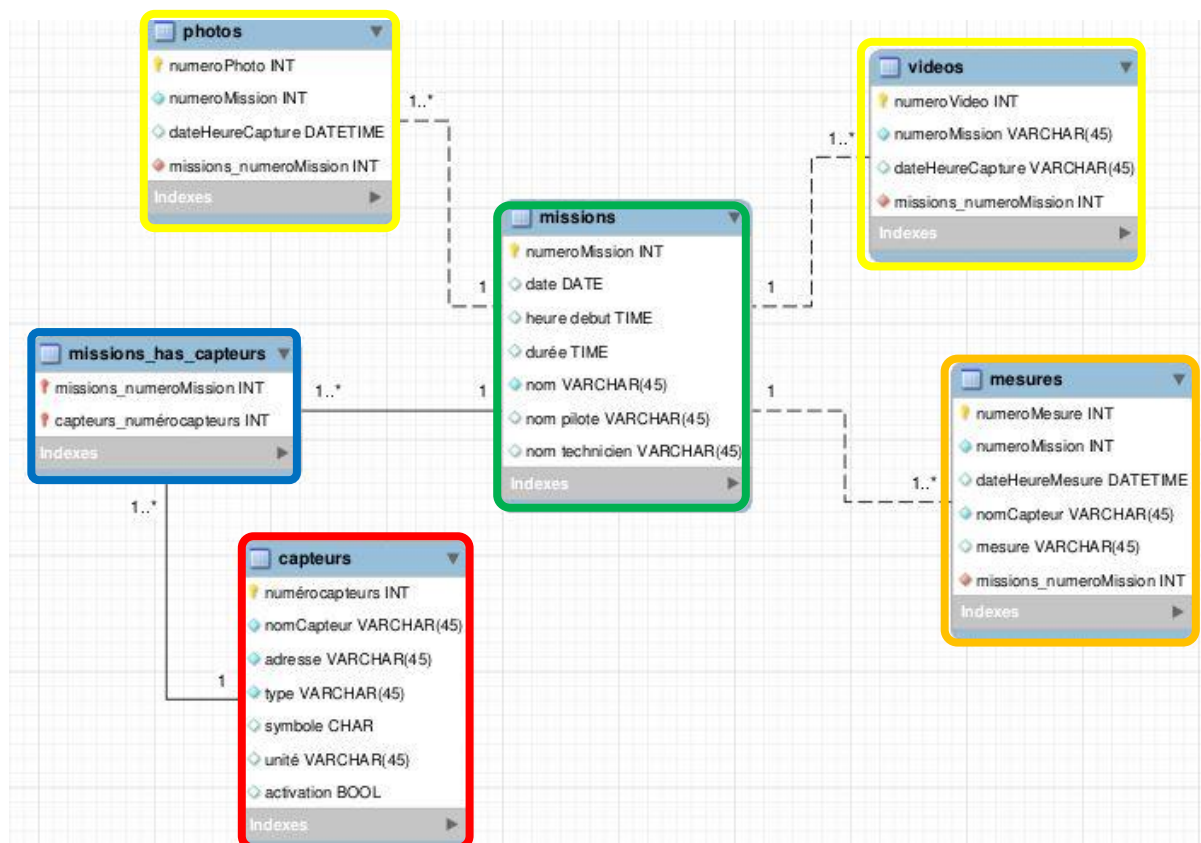
- Enregistrement des mesures :

ID mesure	ID mission	date heure	nom capteur	mesure
1	1	2017/02/28 16:50	captalt1	182
2	1	2017/02/28 16:50	capttemp2	27
3	2	2017/05/23 11:30	captdist1	13
4	3	2017/05/18 18:50	captalt1	231
5	3	2017/05/18 18:50	captalt2	253
6	3	2017/05/18 18:51	captdist3	7

- Enregistrement des capteurs :

ID capteur	nom	type	symbole
1	captalt1	altitude	m
2	captalt2	altitude	m
3	capttemp1	température	°C
4	capttemp2	température	K
5	captdist1	distance	m
6	captdist3	distance	m

Voici un modèle de conception de base de données modélisé avec MySQL Workbench :



Voici comment j'ai décidé de modéliser la base de données de la station au sol :

La table « **missions** » contient un enregistrement de chaque mission, avec son numéro, qui sera auto-incrémenté par le système de gestion (MySQL), la date, l'heure de début et la durée, et le nom du pilote ainsi que du technicien (facultatif).

La table « **mesures** » contiendra un enregistrement de chaque mesure (une par capteur). Ce sera donc la table qui contiendra le plus de données. Les colonnes de cette table sont, le numéro de la mesure auto-incrémenté, le numéro de mission correspondant, la date et l'heure de la mesure, le nom du capteur associé, et la mesure.

La table « **capteurs** » répertoriera chaque capteur pouvant être utilisés sur le drone. Ils comporteront le nom donné par le fabricant, l'adresse I2C ou GPIO sur le drone, le type de capteur (température, altitude...) et le symbole et l'unité de ses mesures. Une colonne « activation » sera utilisée pour gérer l'utilisation des capteurs sur le drone lors de chaque mission.

Une table de **jointure** a été ajoutée automatiquement lors de la création de ce modèle entre la table « **missions** » et la table « **capteurs** », car une mission utilisera un ou plusieurs capteurs, et un capteur sera utilisé par une ou plusieurs missions.

Les tables « **photos** » et « **videos** » serviront à répertorier chacune des photos et vidéos prises par la caméra GoPro, avec un numéro identificateur donné par le système du drone, un horodatage et le numéro de mission correspondant.

Gestion de la base de données

Nous avons décidé de coder en C++ avec QT sous Linux (OpenSuse), j'ai donc décidé d'utiliser MySql pour créer et utiliser la base de données.

Pour la créer à partir du modèle de conception, je l'ai exportée en fichier .Sql, et l'ai importée grâce à phpMyAdmin que j'utilise pour l'administrer.

Au vu des différentes utilisations de la base de données, j'ai décidé de créer plusieurs utilisateurs à cet effet :

nom	type	accès	rôle
root	admin	accès total	administration
GCSTransfert	utilisateur	r	Récupérer les données à transférer à l'entreprise.
GCS	utilisateur	rw	Récupérer les données et enregistrer les données.

Avec :

- r : lecture seule
- rw : lecture, modification et écriture

Seul l'utilisateur « root » a accès aux autres bases présentes sur le serveur.

C'est le service apache2 qui sert à rendre accessible la base de données en local (directement sur le PC), mais aussi dans le réseau local.

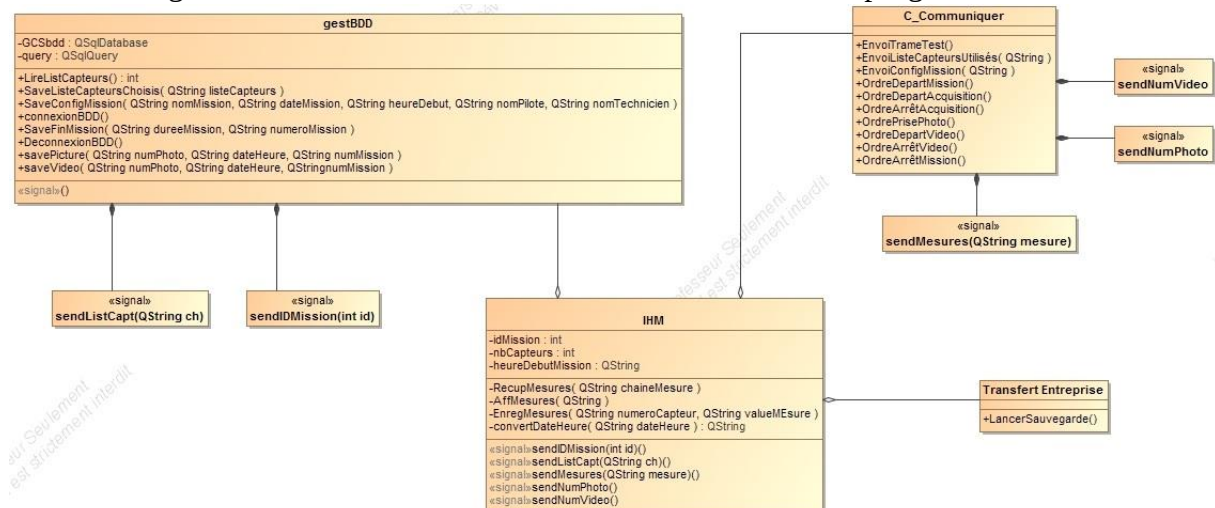
Présentation du programme de la station au sol (GCS)

Ma deuxième tâche pour ce projet, est de coder une interface homme/machine (IHM) permettant de préparer et superviser les missions, mais aussi en arrière-plan de communiquer avec la base de données afin de permettre l'historisation des mesures.

J'ai programmé cette interface en C++, sous QT (sous Linux). Il s'agit donc de programmation objet.

Le programme ayant plusieurs fonctions différentes, il fonctionnera avec une classe principale (IHM), une classe de gestion de la base de données (gestBDD), une classe gérant la communication avec le module (C_Communication), et une classe pour exporter les données en fin de mission (C_Transfert).

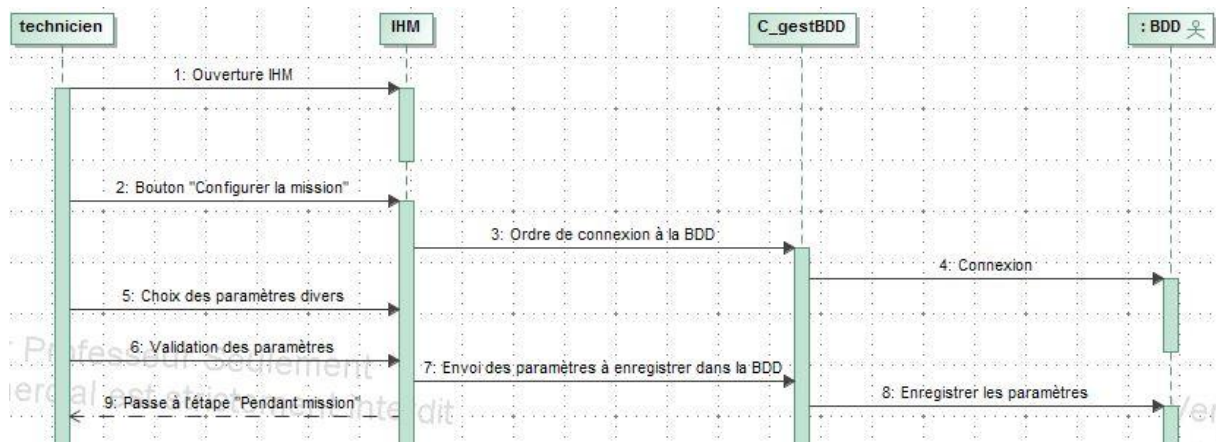
Voici un diagramme de classes montrant les différentes classes du programme :



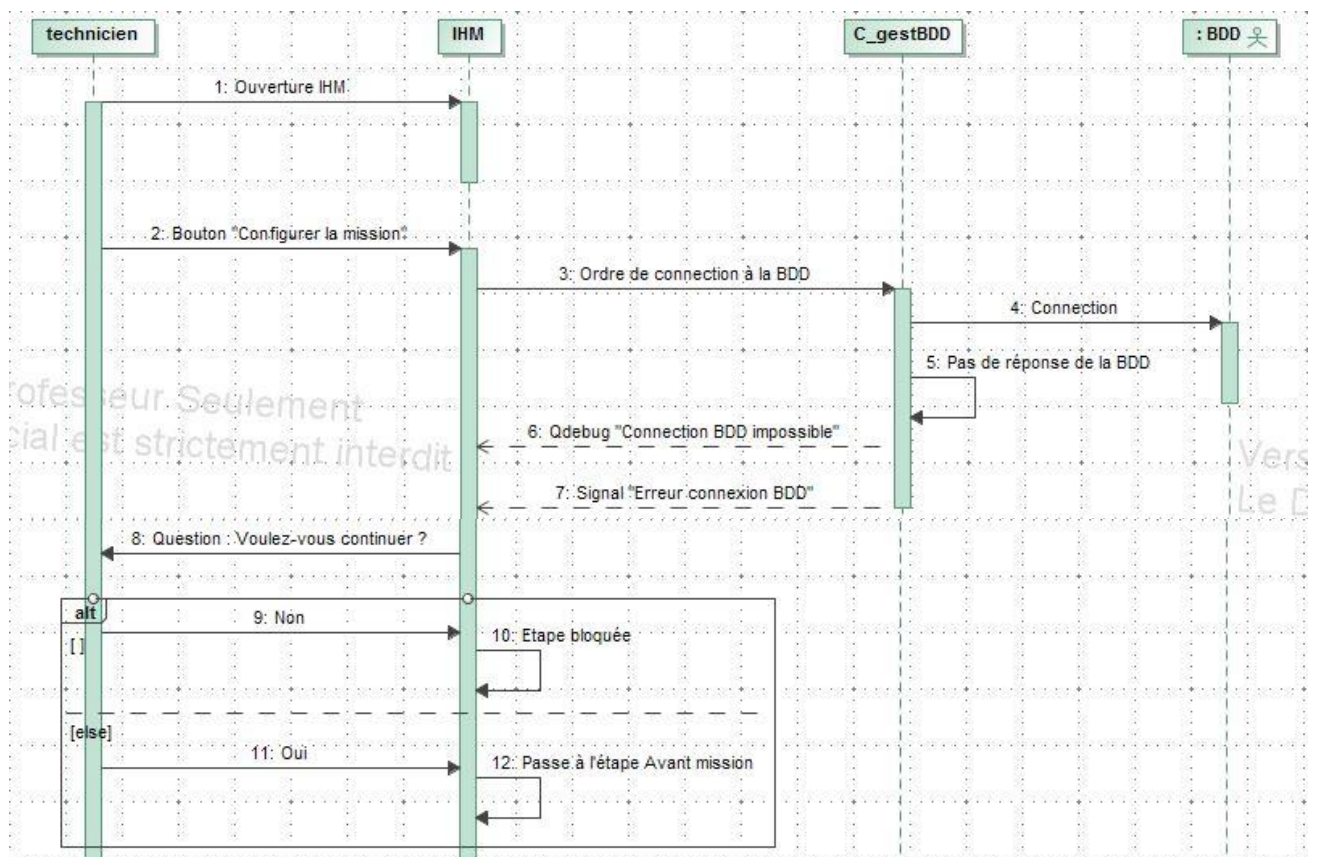
Toutes les méthodes présentes dans les classes « gestBDD », « C_Commiquer » et « C_Transfert » sont appelées par la classe principale « IHM ». On voit aussi que les données dont la classe « IHM » peut demander sont fournies via des signaux.

L'envoi des mesures en temps réel de la classe « C_Commiquer » à la classe principale se fera aussi via un signal « sendMesures() ». Le slot lié à ce signal est le slot « RecupMesures() » qui va extraire les mesures, et appeler les méthodes « AffMesures() » et « EnregMesures() » afin de les afficher sur l'interface, et de les enregistrer dans la base de données.

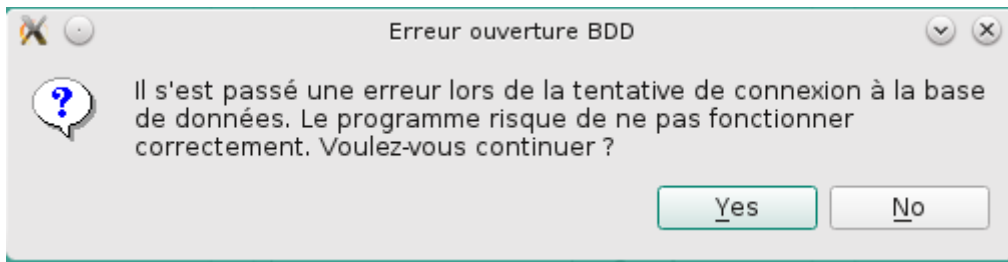
Voici un diagramme de séquence illustrant les échanges entre le technicien gérant l'application, l'IHM, la classe « gestBDD » et la base de données :



Et voici un diagramme de séquence d'erreur lors de la connexion à la base de données avant le début d'une mission :



La question posée à l'utilisateur est concrétisée par une boîte de dialogue qui bloque l'accès au programme tant que l'utilisateur n'a pas répondu :



Cette boîte de dialogue est utilisée grâce à la classe QMessageBox avec la méthode statique question().

Communication avec la base de données

La classe principale (IHM) va appeler des méthodes de l'objet créé à partir de la classe « gestBDD », qui va ensuite émettre les requêtes SQL en local correspondantes.

Les méthodes de « gestBDD » utilisées par l'IHM sont :

- Connexion à la BDD

Cette méthode sert à se connecter à la base de données avec les identifiants utilisateur grâce à la librairie QSqlDatabase de QT

Les identifiants et paramètres sont entrés en « dur » dans le code, et donc non configurable par l'utilisateur.

- Lecture de la liste des capteurs

Récupération des noms des capteurs pour l'affichage sur l'interface afin de permettre l'activation par le technicien. Elle contient une simple requête Sql « SELECT » qui liste les noms des capteurs dans la table « capteurs », et elle la renvoie par l'intermédiaire du signal « sendListCapt ».

- Enregistrement de la configuration de la mission

Envoi des différents paramètres choisis par le technicien pour la mission, notamment le nom de mission (obligatoire), l'heure et la date (automatique par défaut), la liste des capteurs utilisés (tous activés par défaut, le temps d'actualisation des mesures (1 seconde par défaut) et les noms des pilotes et technicien (facultatif).

Un nouvel enregistrement est créé dans la table « missions » grâce à la commande « INSERT ».

- Activation des capteurs

Modification d'une valeur dans l'enregistrement de chaque capteur (avec 0 : non utilisé et 1 : utilisé).

Une boucle modifie une par une les valeurs en écrasant les premières, selon la demande avec la commande Sql « UPDATE ».

La méthode reçoit en plus de la liste des capteurs utilisés, le nombre de capteurs enregistrés dans la base de données, et c'est ainsi qu'une boucle « while » peut fonctionner.

- Enregistrement des photos et vidéos

Envoi de l'horodatage, et du numéro de chaque photo et vidéo correspondant à leur enregistrement sur la carte SD du la caméra.

L'IHM envoie l'heure de prise du cliché ou de début de la vidéo, et le numéro de mission, et après réception du numéro par le drone, envoie celui-ci.

- Enregistrement des mesures

Sauvegarde de chaque mesure, une par une, dans la table « mesures ». Chaque mesure correspond à un capteur, donc pour chaque mission, il y aura X mesures enregistrées simultanément suivant le nombre X de capteurs utilisés. Cela engendre le fait que cette table aura un grand nombre de mesures.

Configuration de la liste des capteurs

Pour l'instant, la liste des capteurs enregistrés dans la BDD, et donc que l'équipe chargée de la mission peut utiliser, n'est prévue d'être configurée que en utilisant phpMyAdmin manuellement, au lieu d'utiliser l'IHM avec, soit l'utilisateur « GCS », soit l'utilisateur « root ».

Communication avec le drone (EDD)

Communications sans-fil nécessaires

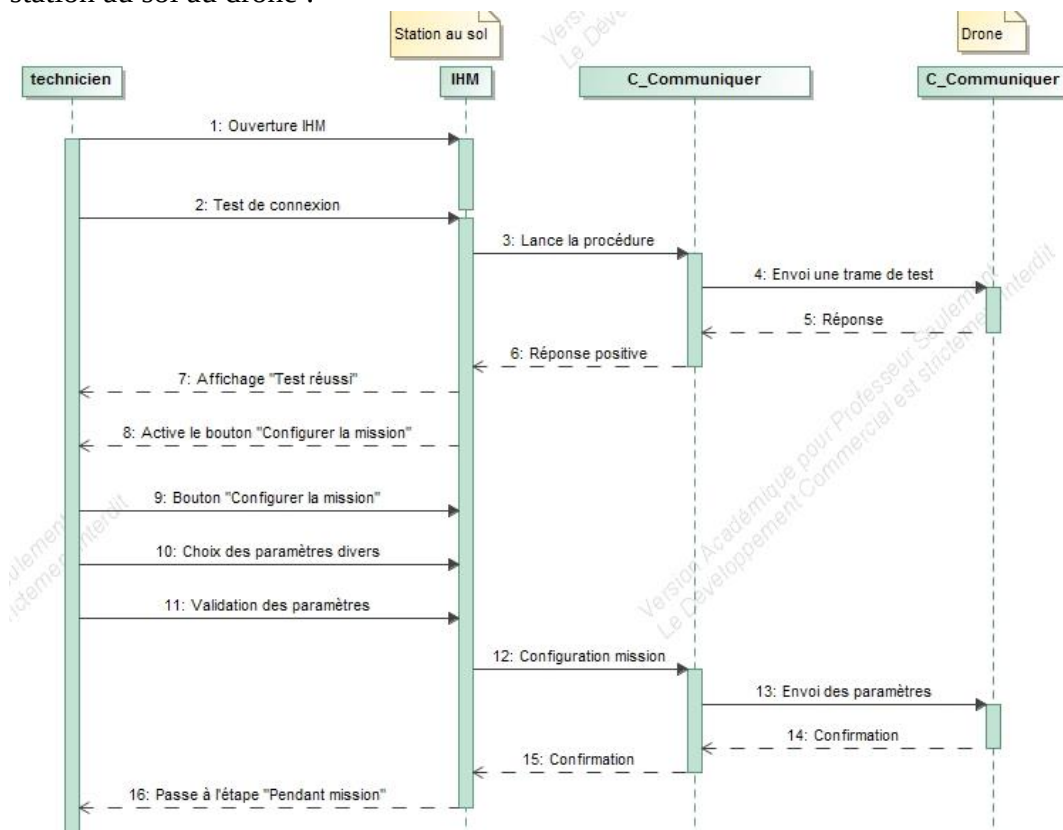
Les différentes communications depuis la station au sol vers le drone sont :

- Connexion via les modules émetteurs/récepteurs 433MHz
- Trame de test de connexion suivant le protocole
- Envoi des paramètres de la mission choisis par le technicien
- Ordre de début de la mission
- Communications diverses pendant la mission (départ de l'acquisition des mesures, arrêt, prise de photos, de vidéos, modification des paramètres d'incrutation vidéo)
- Ordre de fin de mission

Et les communications depuis le drone vers le sol sont :

- Réponse de connexion sans-fil
- Réponse au test de connexion data
- Validation de la réception des paramètres de la mission avec la liste des capteurs disponibles et fonctionnels sur le drone
- Envoi des mesures toutes les X secondes (temps défini avant la mission), envoi des numéros de photos et vidéos prises

Voici un diagramme de séquence décrivant les échanges effectués lors de la connexion de la station au sol au drone :



Méthodes utilisées

Pour permettre la communication entre la classe principale et la classe « C_Comuniquer », nous avons décidé avec Kevin MAUBERNARD, que la classe « IHM » appellerait des méthodes de la classe distante, et que les réponses ou informations asynchrones que doit recevoir l'IHM seraient envoyée via un signal.

Les signaux envoyés pas la classe « C_Communiqueur » sont :

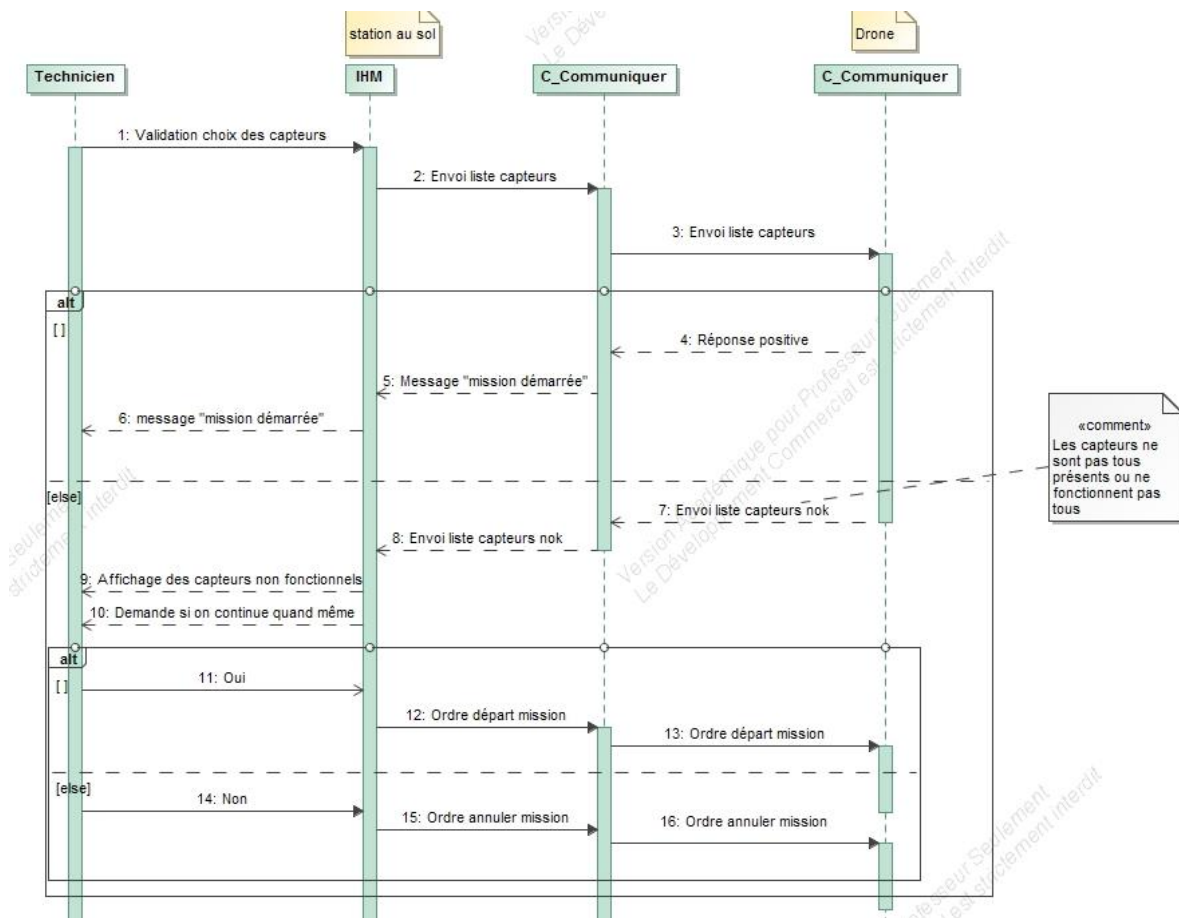
- « sendDataTest » : envoi de la confirmation de réception de la trame de test
- « sendListCaptOk » : envoi le liste des capteurs disponibles et fonctionnels sur le drone
- « sendNumPhoto » : envoi le numéro de la photo, donné par le drone, qui vient d'être prise
- « sendNumVideo » : envoi le numéro de la vidéo, donné par le drone, qui vient d'être démarrée
- « sendMesures » : envoi des mesures en temps réel reçue du drone

Les signaux sont liés à des slots qui vont récupérer les données :

- « ValidationTestData » : active le bouton pour passer à l'étape « Avant Mission », ou affiche un message « Connexion avec le drone impossible »
- « ValidationListCaps » : affiche une confirmation de la disponibilité des capteurs, ou affiche la liste de ceux non disponibles et propose de continuer ou d'annuler la mission

- « enrNumPhoto » : envoi le numéro de la photo prise, ainsi que son horodatage et son numéro de mission à la classe chargée de l'enregistrer dans la BDD
- « enNumVideo » : envoi les mêmes informations que pour les photos, mais ajoute la durée de la vidéo prise
- « RecupMesures » : extrait les mesures reçues, puis les envoie à deux autres méthodes, l'une chargée de les afficher « AffMesures », et l'autre de les envoyer à la base de données « EnrMesures »

Voici un diagramme de séquence illustrant les échanges lors de la réception par l'IHM de la liste des capteurs disponibles :



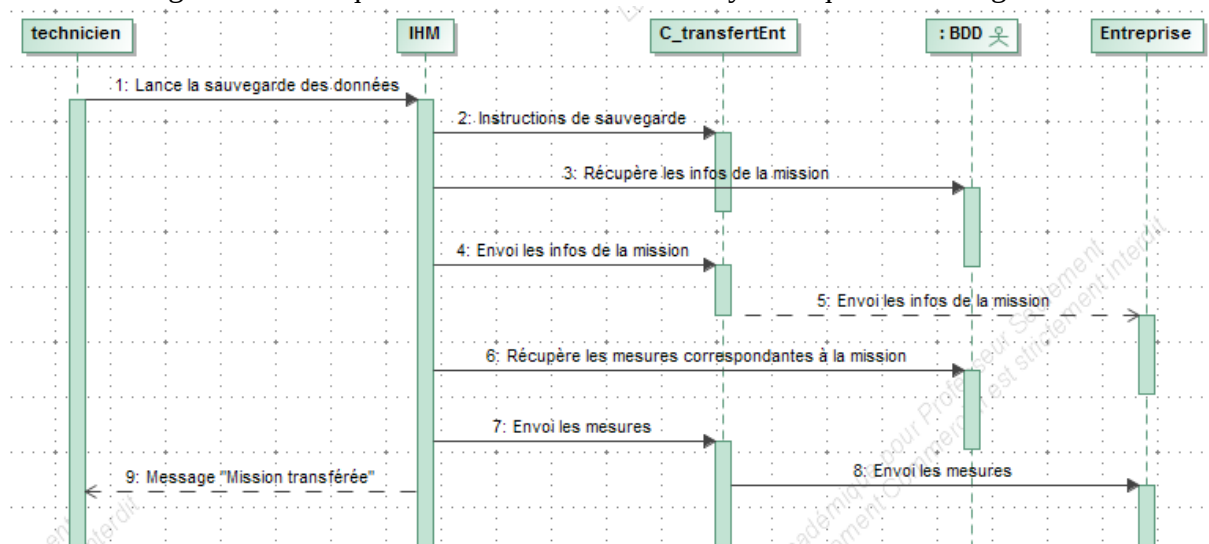
Communication avec l'entreprise à distance

L'équipe en charge de la mission doit pouvoir exporter les données (mesures et informations de la mission) vers l'entreprise qui se chargera de l'enregistrer dans sa base de données en local.

Pour cela, l'IHM doit se connecter, grâce à une liaison internet (4G) au serveur de l'entreprise.

C'est lors de l'étape « Après mission », que l'IHM va appeler les méthodes de la classe « C_Transfert » pour se connecter, récupérer les données et les envoyer.

Voici un diagramme de séquence illustrant de manière synthétique cet échange :



Interface utilisateur

L'interface homme/machine, c'est le « visuel » de l'application, la partie qui va avoir des interactions avec le technicien (afficher un message, cliquer sur un bouton...).

Le principe sous QT, c'est de lier un signal émis par un élément de l'interface (un bouton par exemple), qui est déclenché par une action de l'utilisateur (un clic par exemple), à un slot, une méthode présente dans la classe principale.

L'idée pour cette interface, était de se baser sur trois étapes de la mission :

- L'étape **Avant Mission**

Le technicien saisi le nom de la mission, sélectionne les capteurs qu'il veut utiliser, choisi le temps d'actualisation des mesures, vérifie l'heure et la date, et éventuellement renseigne son nom ainsi que celui du pilote.

- L'étape **Pendant Mission**

Le technicien peut démarrer et arrêter les mesures, modifier les paramètres d'incrémentations vidéo et prendre des photos ou des vidéos.

- L'étape **Après Mission**

Le technicien peut se connecter à l'entreprise à distance et y exporter les données de la mission. Il peut aussi récupérer les données sauvegardées dans le drone pour les enregistrer dans la base de données.

Il y a malgré tout, une petite étape avant la première, qui permet de configurer le module 433MHz branché sur le PC, et de se connecter au drone ainsi qu'à la base de données, mais tout cela se fait pratiquement automatiquement.

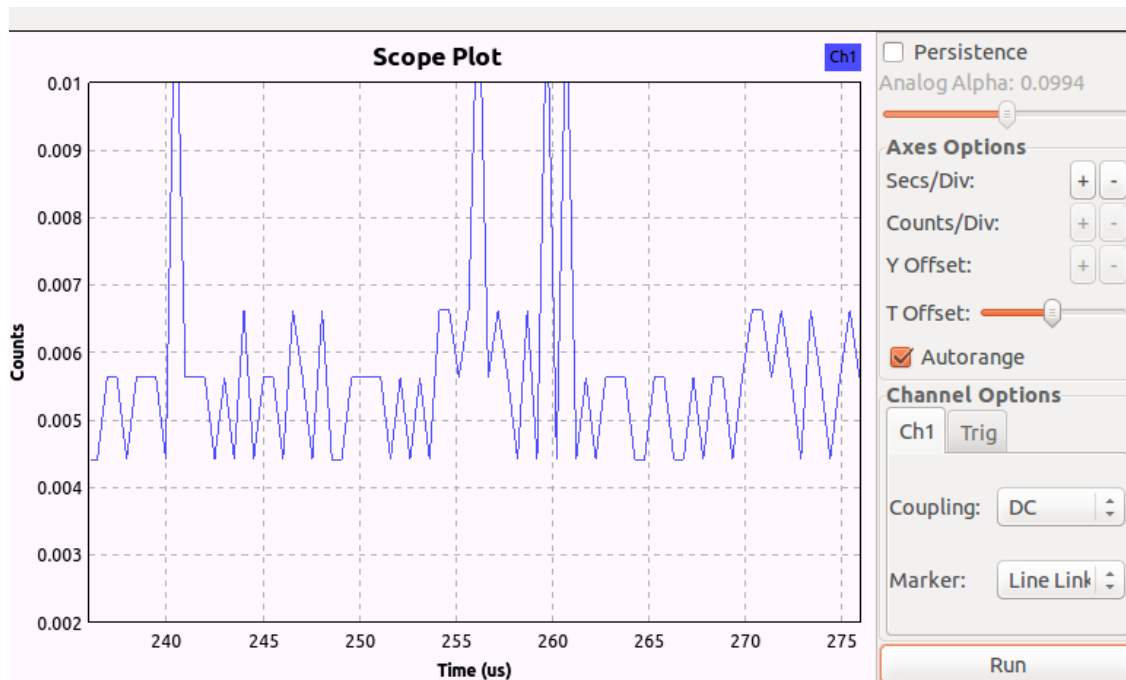
J'ai donc décidé de créer trois zones sur l'interface, grâce aux widget « groupBox ». Cela me permet aussi de les activer/désactiver lors d'un passage d'une étape à l'autre afin d'éviter une mauvaise manipulation de l'utilisateur.

Analyse physique

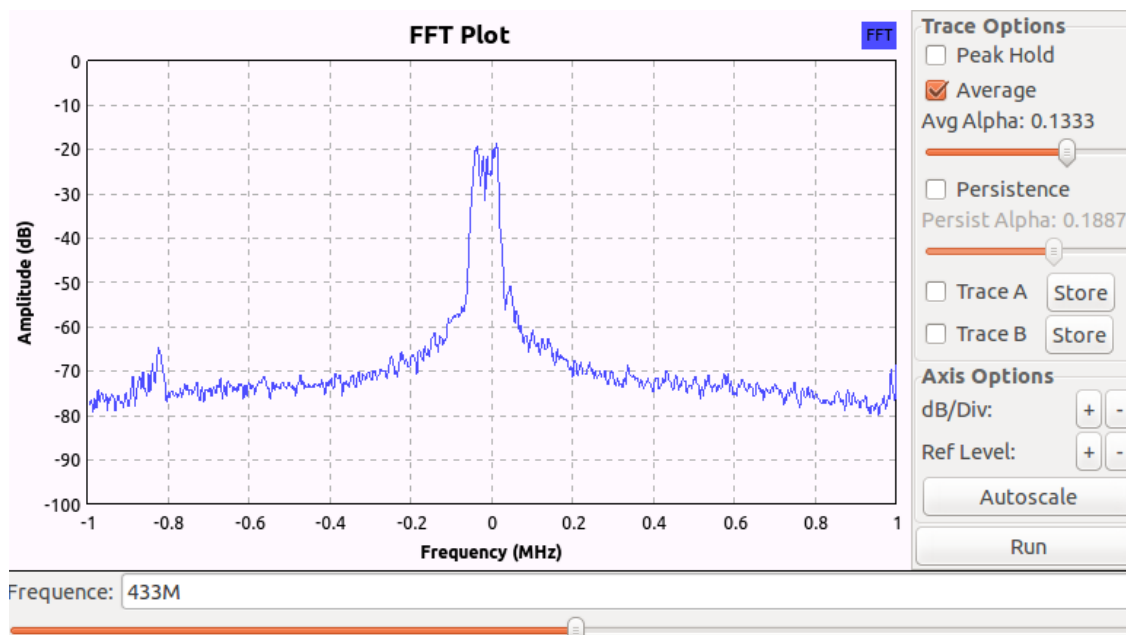
Transmission via le module 433MHz

Lors de notre étude des modules qui fonctionnent en 433,92MHz, nous avons envoyé une trame de test, et visualisée celle-ci avec un oscilloscope.

Voici un graphique où l'on peut observer l'émission de trames via le module :



Voici le spectre de fréquence correspondant :



Conclusion

Pour ma part j'ai démarré le projet un peu en retard, j'ai eu du mal à bien le comprendre dès le début.

Mais après la première revue, et le premier cycle de développement, j'ai vraiment avancé ma partie.

J'ai aussi commencé un peu tard à communiquer avec ceux avec qui mon code devait communiquer, pour la communication avec le drone et le transfert vers l'entreprise.

Cependant, j'ai beaucoup appris en développement de projet en groupe, avec différentes revues de projet permettant de mettre à jour notre avancement, et les réunions tous ensemble ou avec seulement quelques-uns.

Nous avons eu du mal à préparer notre étude de la partie physique avec Kevin MAUBERNARD et Nicolas HOUTTEMANE, car notre partie consistait uniquement à coder sans nous pencher directement sur le côté physique, moi-même étant responsable d'une interface logicielle et non matérielle.

Les étapes de ma partie qui sont finies sont celles-ci :

- La base de données
- Les différents utilisateurs de la base de données
- La préparation de la mission (IHM)

Et les parties en cours de développement ou de réflexion sont celles-ci :

- La supervision de mission (mesures, caméra, incrustation vidéo)
- L'historisation des mesures
- Le transfert vers l'entreprise

Pour ce dernier cycle de développement, je prévois donc de finir la partie supervision de mission, avec la collaboration de la classe « C_Communique », l'enregistrement des mesures dans la base de données au fur et à mesure, et de communiquer avec la classe « C_Transfert » afin de pouvoir transférer les données à distance.

ANNEXES

Diagramme des exigences du système

