Sur la page de garde doivent apparaître :
- Segment Vol,
- Nom prénom de la personne auteure du dossier personnel
- Même page de garde pour TOUS...
- Essayez de réduire certaines captures pour ne pas dépasser 30 pages...

DOSSIER PERSONNEL

- Il manque une conclusion (Bilan personnel...)

Projet InitCube 2019
SegmentVol

DEROZIER – LOREAL – MARYNUS - RIBET

Sommaire

1 . Introduction....3
1.1 Situation initiale....3
1.2 Étude de la charge utile....3
1.3 Répartition des tâches dans l'équipe....3
2. Ma mission dans ce projet....4
2.1 Explication de mon travail....4
2.2 Position dans l'architecture matérielle....5
2.3 Répartition dans le temps....7
2.4 Diagramme de classes - Métier....8
2.5 Diagramme des classes – Communication....9
2.6 Diagrammes de séquence....10
3. Logiciels et systèmes d'exploitations utilisés....12
4. Étude du capteur infrarouge de température TPA64....13
4.1 Choix de la caméra....13
4.2 Le fonctionnement du capteur....14
5. Codage des classes....22
5.1 Rédaction et réalisation des tests unitaires....22
....22
5.2 Rédaction et réalisation des tests d'intégration....23
6. Exécution du code de la Caméra IR....24
6.1 Organisation en matrice....24
6.2 Test des limites du capteur et affichage des erreurs....25
7. Fiches de tests unitaires....27
7.1 Test unitaire de Mesure....27
7.2 Test unitaire d'Instrument....29
7.3 Test unitaire de CameraIR....31
8. Fiche du test d'intégration....33
9. Journal de bord du projet....35

Dossier Personnel

1 . Introduction

1.1 Situation initiale

Le projet InitCube consiste à initier les étudiants au technologies utilisées dans l'étude spatiale. Notre partie du projet consiste à réaliser la station Vol qui est composée d'un ordinateur de bord surmonté d'une batterie pour rendre le système autonome. Le principe de l'étude étant de faire des mesures à l'aide de différents instruments, celui utilisé dans notre kit est une caméra infrarouge. Pour imiter au mieux les conditions réelles, les données récupérées concernant l'état du système et les mesures effectuées par la caméra doivent être transmises au segment Sol pour ensuite être analysées et pour que le segment Sol fasse des ajustements si nécessaire.

1.2 Étude de la charge utile

Le contexte de l'étude porte sur l'acquisition de données météorologiques et plus précisément sur la température au niveau des zones chaudes de la Terre. Pour cela nous avons choisi d'utiliser une caméra infrarouge afin de faire des mesures grâce à la radiation de ces zones. Une fois ces mesures effectuées, elles doivent être stockées avant d'être transmises à la station au sol.

1.3 Répartition des tâches dans l'équipe

Notre équipe est composée de quatre étudiants de BTS Systèmes Numériques option Informatique et Réseaux au lycée Victor Hugo de Colomiers.

DEROZIER Xavier : Etude de la batterie afin de collecter les données de charge tel que le niveau de charge, la tension et le courant traversant la batterie, la température pour prévenir d'une potentielle surchauffe, et savoir si la batterie est en charge ou non.

Un courant ne traverse pas la batterie. Il est fourni à ou par la batterie.

MARYNUS Lucas : Codage de la classe Ordinateur et récupération des données de stockage de l'ordinateur (mémoire USD et RAM). En parallèle, doit acquérir les valeurs de la température processeur et la température du système.

Il

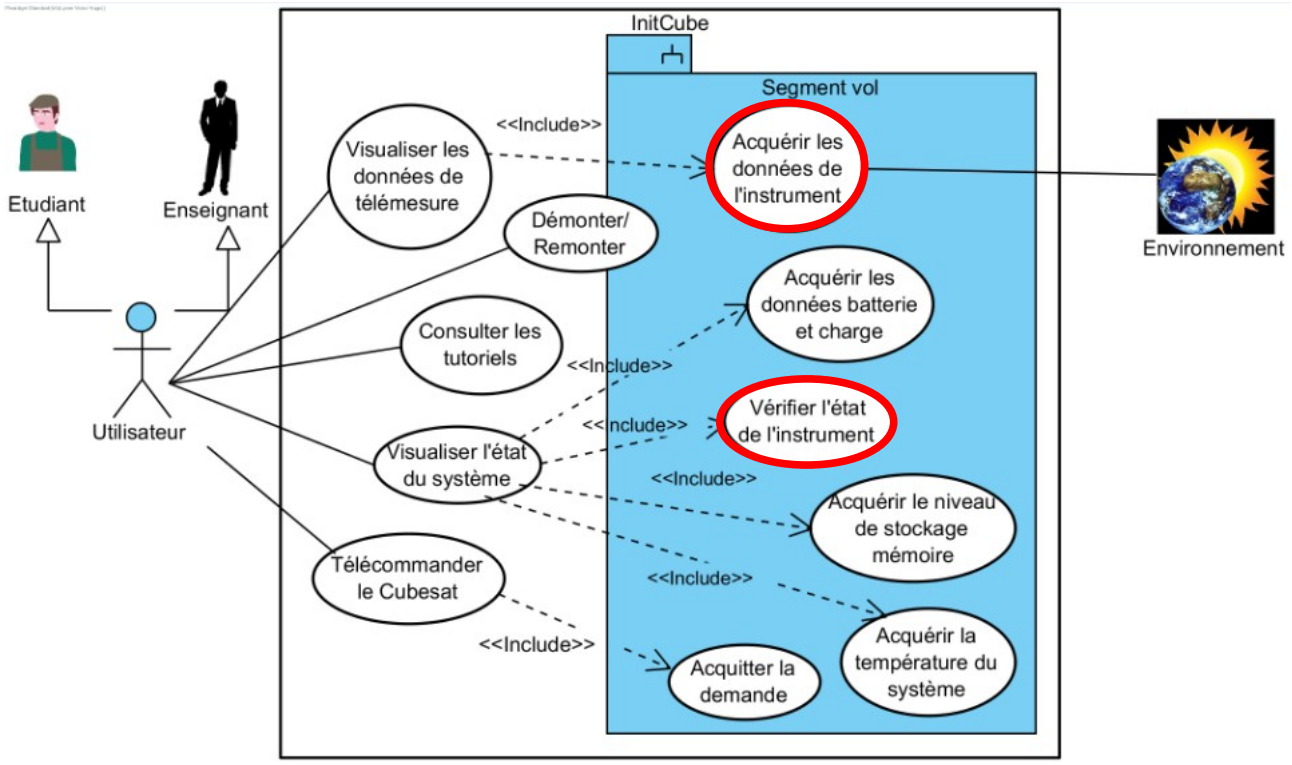
RIBET Jéremy : Mise en place d'un protocole pour la transmission des données et codage des classes SegmentSol, EmetteurRecepteur, Mission, Commande et Message.

LOREAL Gwendal : Je m'occupe de coder les classes Instrument, CameraIR, Status et Mesure pour la caméra infrarouge. Je dois récupérer soit la température moyenne calculée à partir des relevés de chaque pixel, soit la valeur de chaque pixel. Je dois aussi pouvoir récupérer l'état de l'instrument pour le configurer si nécessaire et obtenir la température de l'instrument.

2. Ma mission dans ce projet

2.1 Explication de mon travail

Diagramme des cas d'utilisation de notre projet :

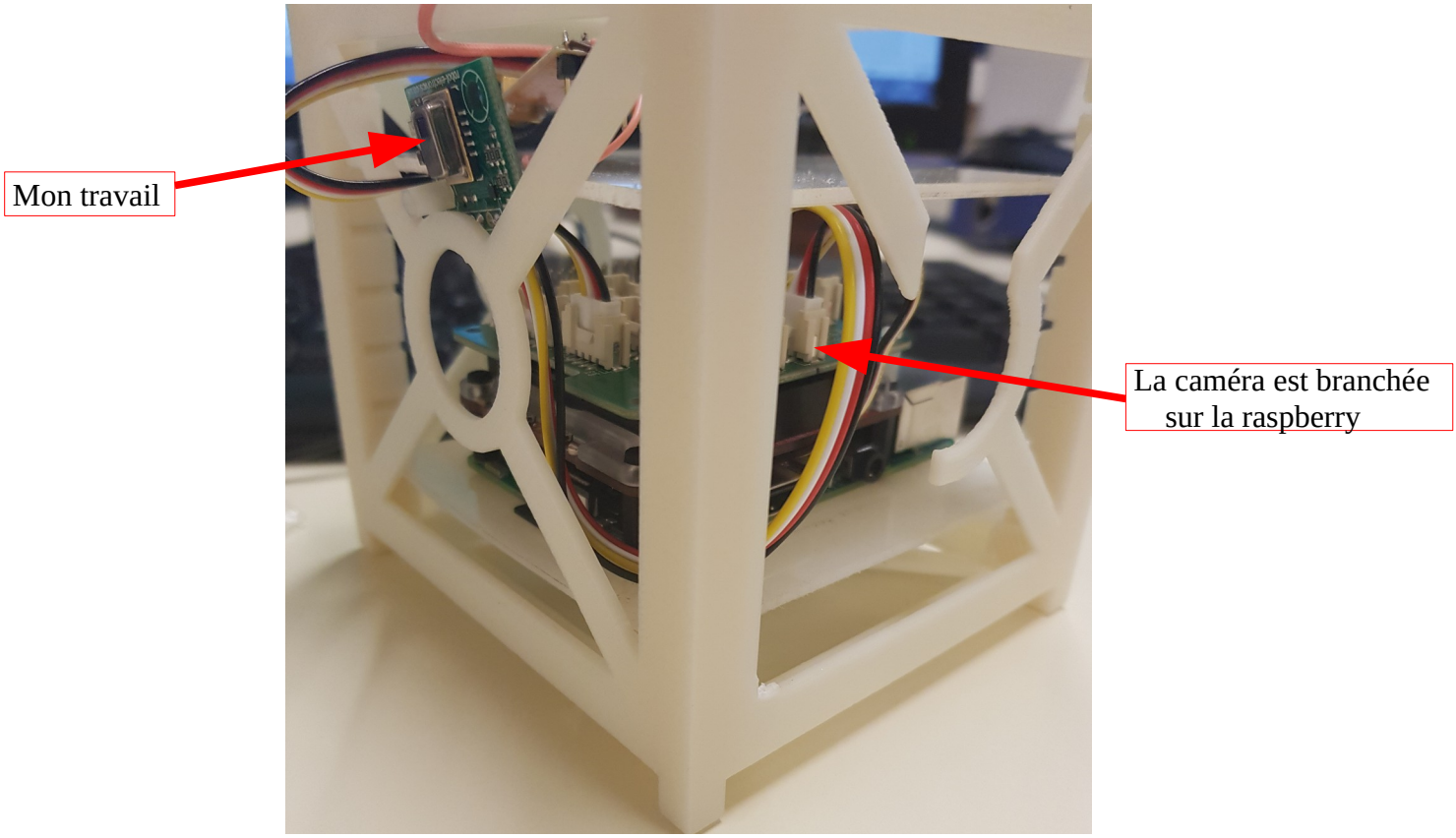


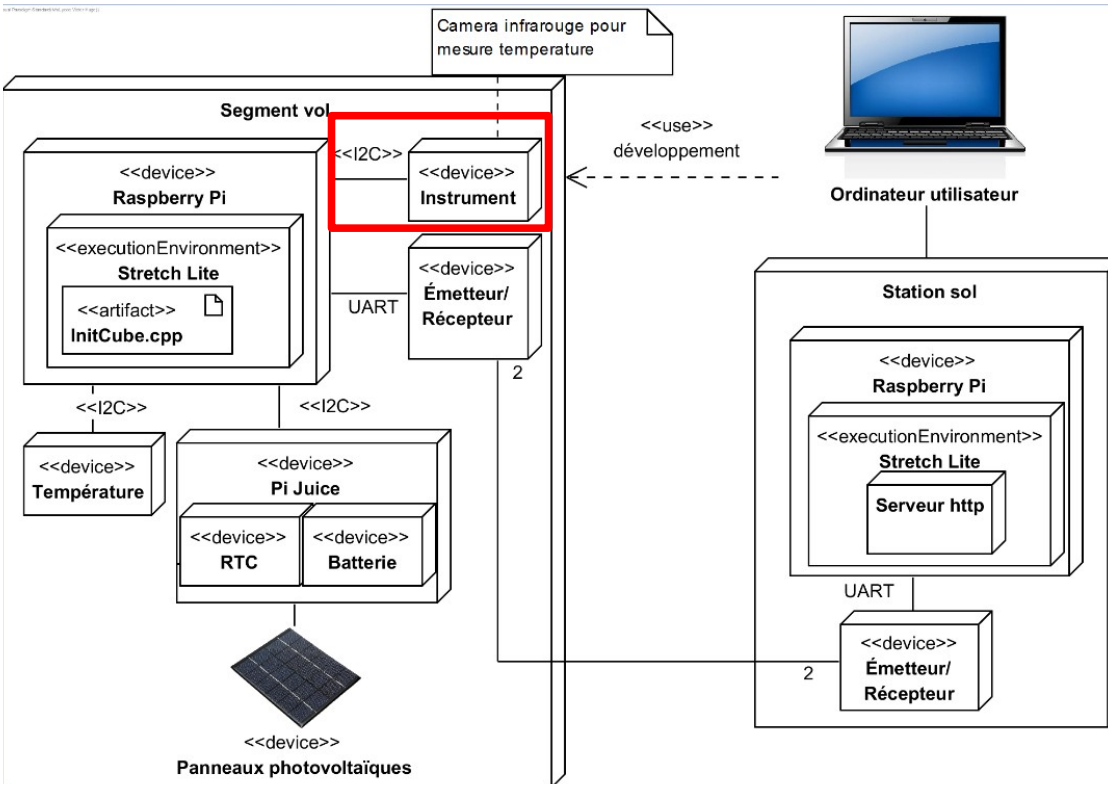
Dans le diagramme ci-dessus, nous pouvons voir à quel niveau je me situe dans le projet.

Le travail que je dois réaliser est l'acquisition des données collectées par l'instrument et la vérification de l'état de l'instrument. Dans notre cas, l'instrument est une caméra infrarouge. La caméra devra donc récupérer la température surfacique de ce qu'elle aura en face pour ensuite pouvoir transmettre les données au segment Sol.

De plus, nous aurons le contrôle sur le mode de fonctionnement du capteur car il devra pouvoir être configurable dans ses trois modes. Et le statut de l'instrument pourra être retourné si nécessaire.

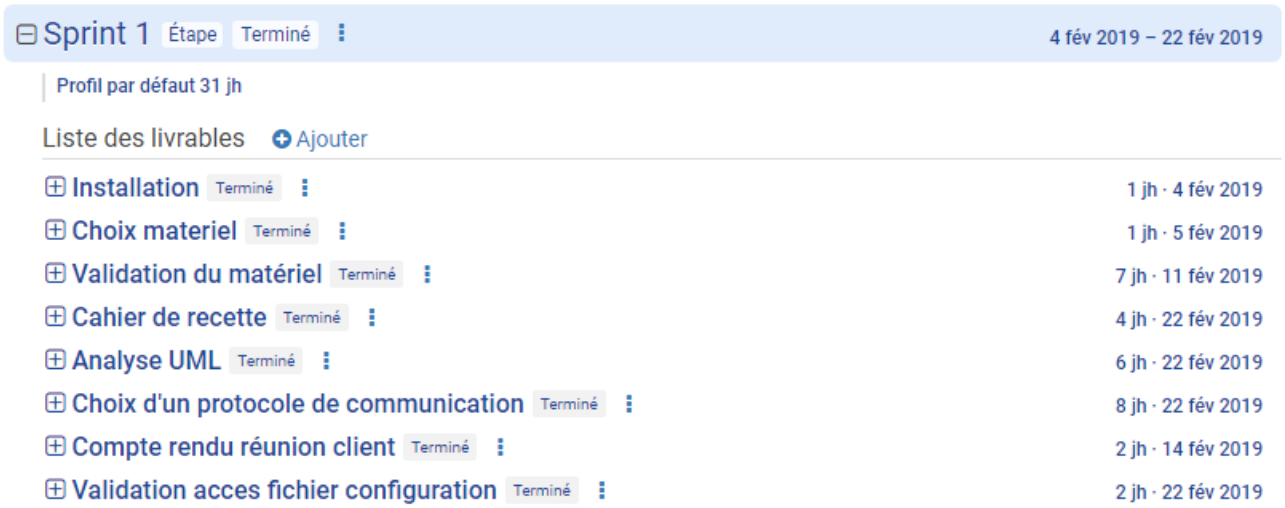
2.2 Position dans l'architecture matérielle





Le schéma et le diagramme précédent présente mon travail au sein du projet InitCube. Je dois donc réaliser différentes classes qui permettront de faire des mesures avec la caméra infrarouge et de les enregistrer afin qu'elles puissent être transmises au segment Sol. La caméra communique via une liaison I2C avec l'ordinateur de bord (la raspberry).

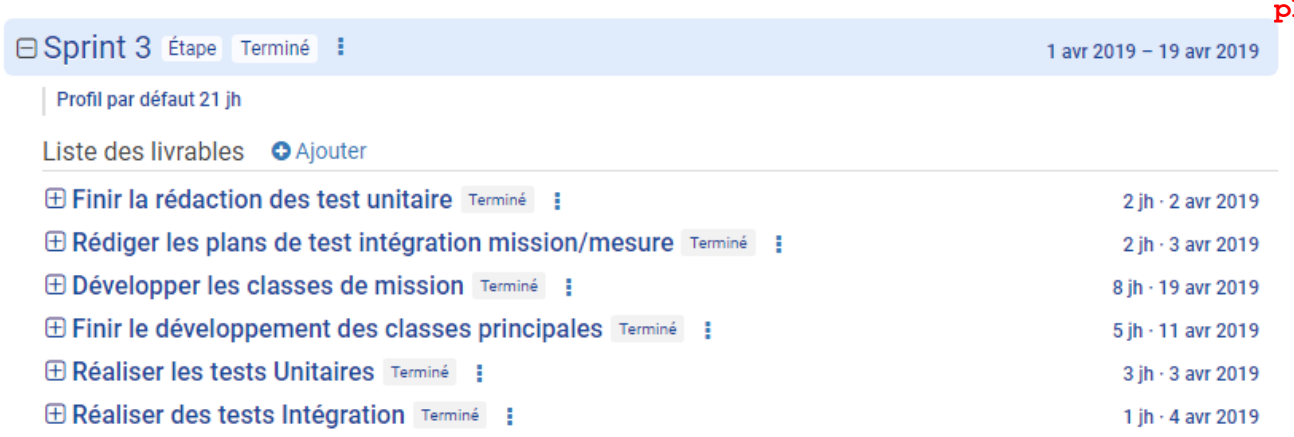
2.3 Répartition dans le temps



Lors du premier sprint, nous avons découvert le projet avec le cahier des charges, nous avons ensuite fait l'analyse UML et enfin nous avons fait la validation technologique des équipements choisis.

Correction des diagrammes, rédaction des tests unitaires des classes Instrument, Mesure, CameraIR, et Status. Et début du codage des mes classes.

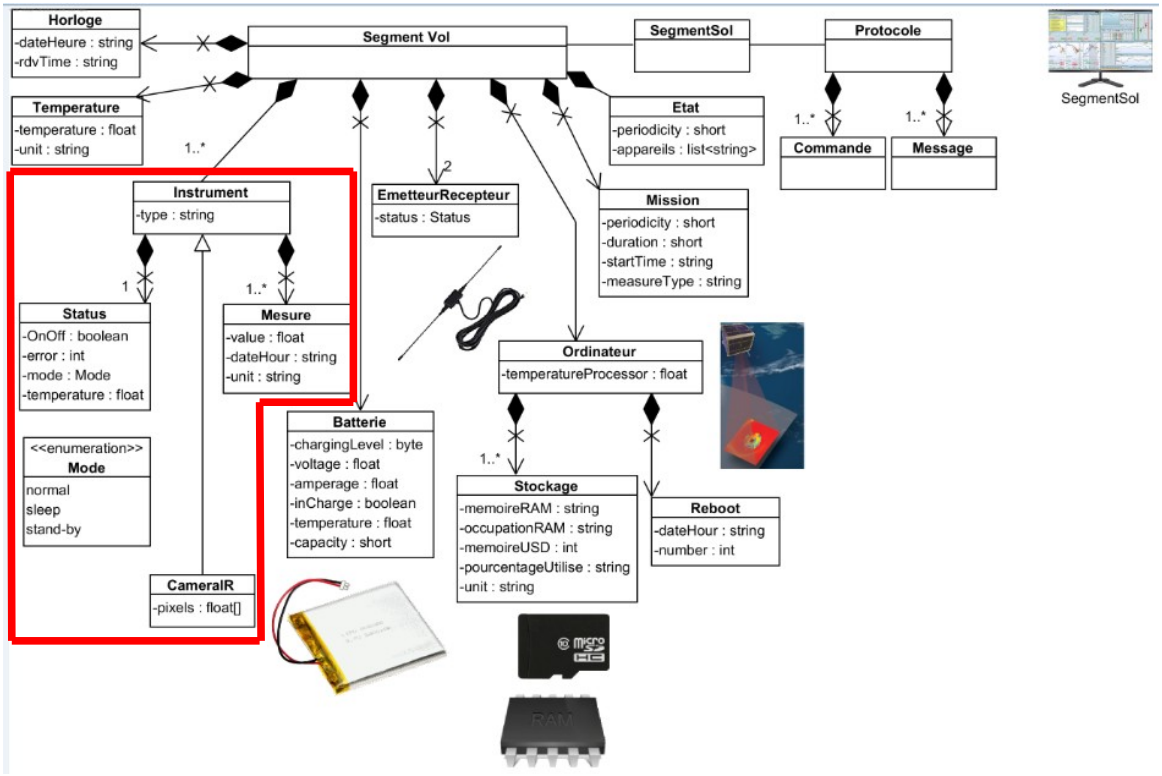
C'est mieux quand ce sont des phrases...



Continuation du développement des classes, rédaction des tests d'intégration et début de la réalisation des tests unitaires.

2.4 Diagramme de classes - Métier

Ma tâche principale est de coder les classes permettant l'acquisition des données de l'instrument. Pour cela je m'occupe donc des classes Instrument, Status, Mesure et CaméraIR.



Description des classes :

Classe CameraIR : Cette classe doit contenir le code permettant d'activer ou de désactiver l'instrument, de relever la valeur de chaque pixel de la caméra et de calculer la température moyenne, mais aussi de récupérer le mode de l'instrument pour savoir si on doit le changer ou non et enfin obtenir la température du capteur. Un horodatage sera effectué sur chaque mesure et on pourra aussi choisir le rayon de capture de la caméra infrarouge.

Confusion possible. Vous voulez parler de la thermistance qui mesure la température interne de la caméra?

Classe Status : Cette classe permet de configurer le mode de l'instrument (choix entre le mode NORMAL, SLEEP, STAND_BY) mais aussi de le retourner pour pouvoir le récupérer avec la classe CameraIR. La classe Status permet aussi de récupérer la température de l'instrument afin de surveiller une éventuelle surchauffe ou au contraire un température trop faible.

Attention! C'est CameraIR qui la récupère. Status ne fait que stocker. Elle est commune à tous les instruments.

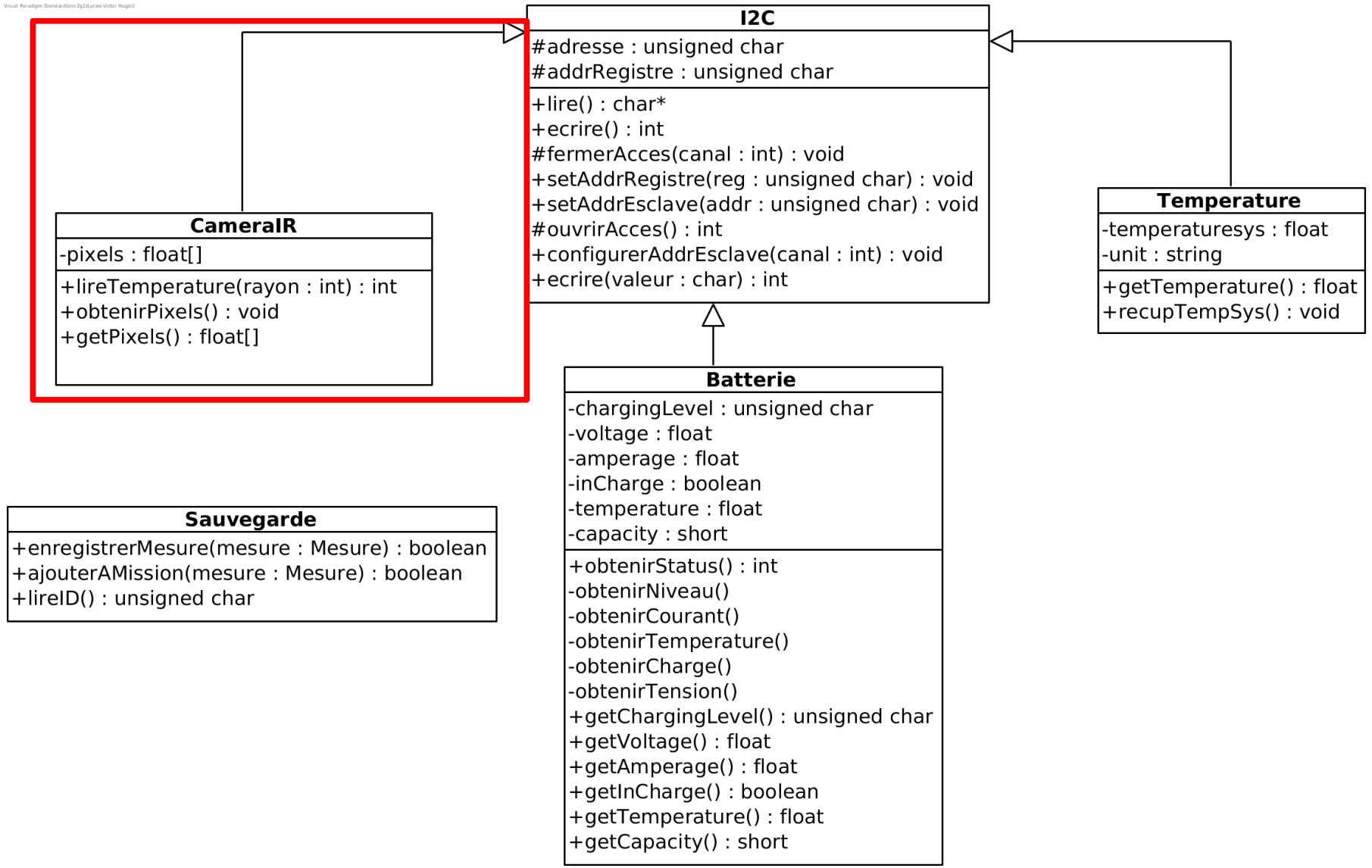
Classe Mesure : La classe Mesure sert à enregistrer les mesures faites par la caméra et permet d'assigner la date et l'heure à chaque mesure effectuée. L'unité choisie pour la mesure sera sélectionnée dans cette classe.

Non! C'est segment vol qui assigne, Mesure ne fait que stocker ou associer.

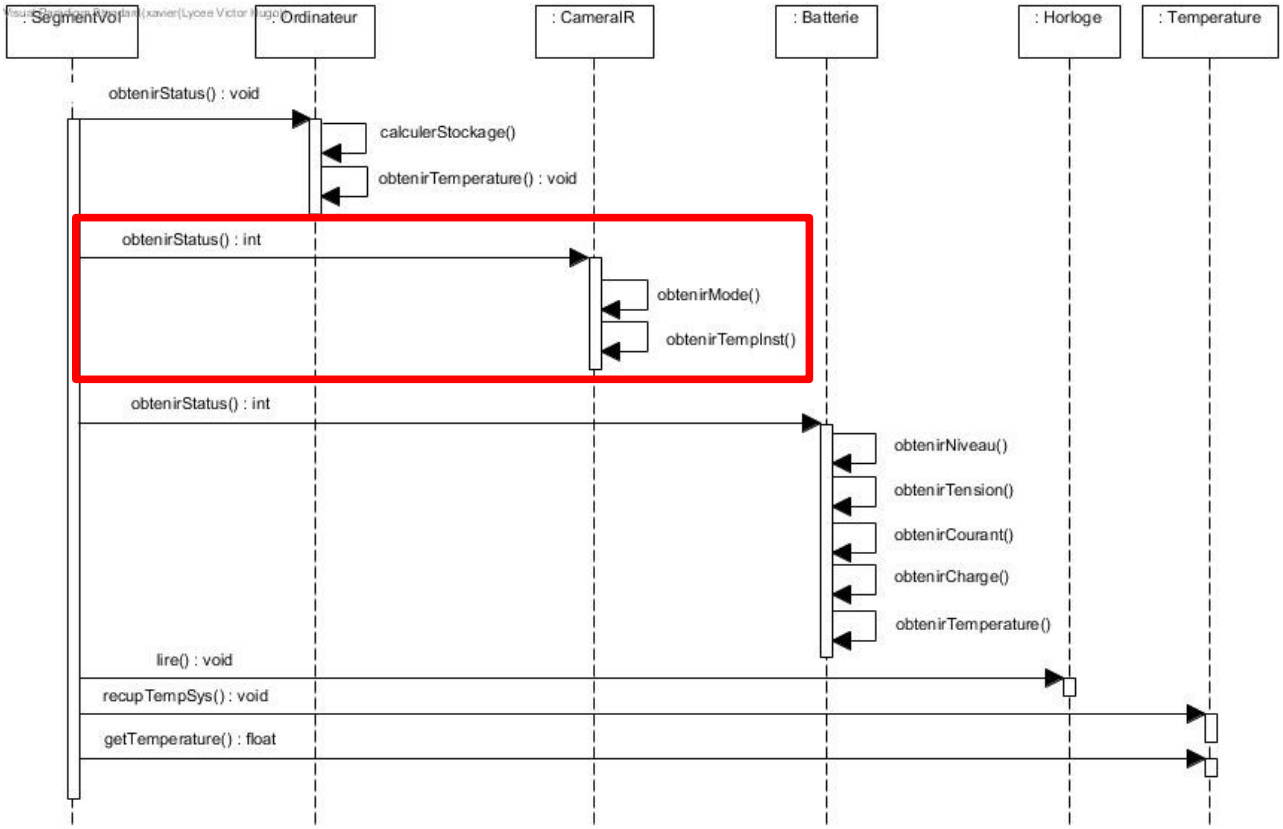
Classe Instrument : Cette classe permet d'activer et de désactiver l'instrument en appelant les méthodes du même nom. Elle permet aussi d'ajouter les mesures dans une liste qui sera ensuite récupérée pour la transmission. C'est cette classe qui permet de faire le lien avec les autres pour que toutes les classes puissent fonctionner ensemble.

Non! C'est nien votre CameraIR qui le fait. Cette classe est une classe générique dont héritera tous les instruments. Certaines méthodes sont codée dans Instrument (addMesure ou clearMesure maus les autres dépendent de l'instrument.

2.5 Diagramme des classes – Communication

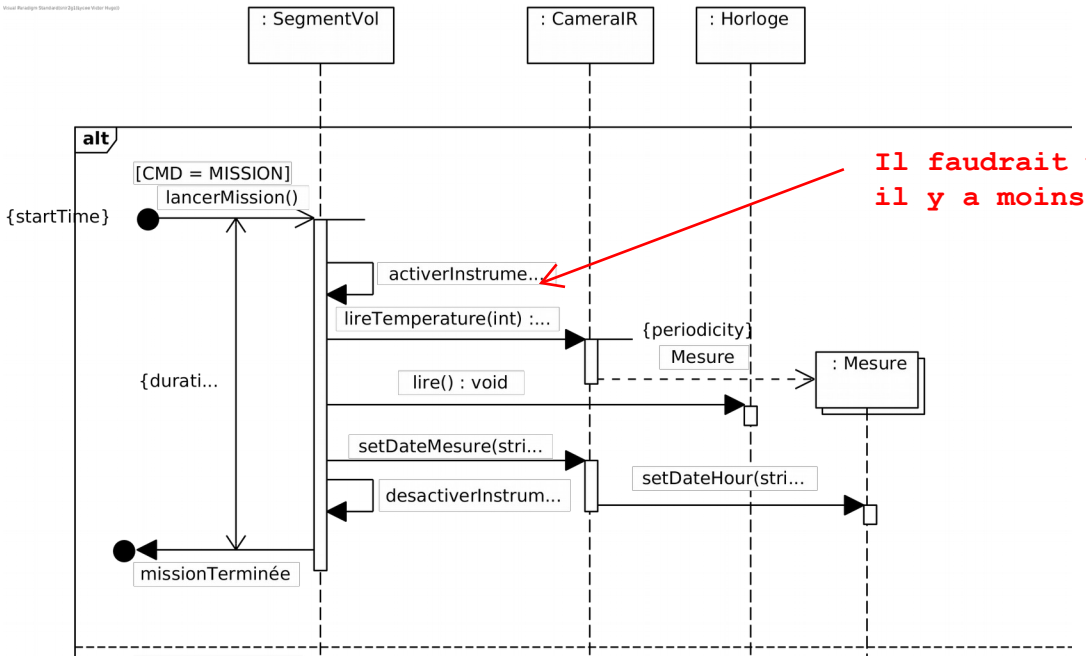


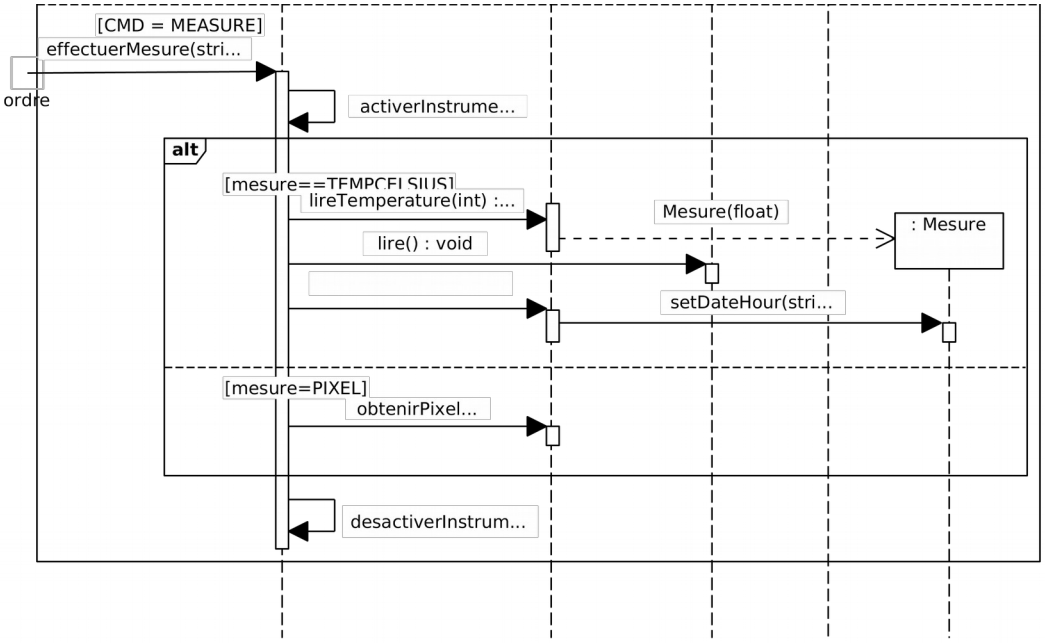
2.6 Diagrammes de séquence



Je m’occupe de récupérer le mode et la température de l’instrument.

Diagramme de la mission et de la mesure :





3. Logiciels et systèmes d'exploitations utilisés

- Raspian :

  Le SegmentVol utilisé dans le projet est une Raspberry Pi, nous avons installé**er** Raspbian qui est une distribution particulière de Debian. Cette version de Raspbian est lite c'est-à-dire qu'elle n'a pas d'interface graphique et donc consomme moins d'énergie.

NON! Ce n'est pas le segment vol qui est une Raspberry mais l'ordinateur de bord.

- Fedora 24 :

  Le ~~SegmentVol~~ étant une Raspberry Pi, nous devons nous connecter en ssh pour y téléverser les codes et les tester donc nous avons décidé d'utiliser Fedora pour le codage.

- Netbeans :

  Nous avons utilisé le logiciel Netbeans pour développer nos classes, réaliser nos tests unitaire et nos tests d'intégration.

Netbeans n'étant pas installé sur la Raspberry les tests avec l'I2C n'ont pu être "réalisés" sous Netbeans...

- Visual Paradigm :

  Nous nous sommes servis de Visual Paradigm pour créer et modifier les différents diagrammes.

- Time Performance :

  Ce logiciel nous a permis de nous coordonner sur notre travail et sur la gestion de la répartition des diverses tâches en fonction des échéances.

- Google Drive :

pas d'accent

  Le drive de Google nous a permis de partager nos fichiers et surtout d'y mettre nos codes pour que tous les membres du groupe puissent y avoir accès.

4. Étude du capteur infrarouge de température TPA64

4.1 Choix de la caméra

C'est normal le décalage?

Types

Tape and reel package : 1,000 pcs.

Product name	Number of pixel	Operating voltage	Amplification factor	Part number
Infrared array sensor Grid-EYE High performance type	64 (Vertical 8 × Horizontal 8 Matrix)	3.3 V.DC	High performance type High gain	AMG8833
			High performance type Low gain	AMG8834
		5.0 V.DC	High performance type High gain	AMG8853
			High performance type Low gain	AMG8854

Rating

Item	Performance	
	High gain	Low gain
Applied voltage	3.3 V.DC±0.3 V.DC or 5.0 V.DC±0.5 V.DC	
Temperature range of measuring object	0 °C to 80 °C +32 °F to +176 °F	−20 °C to 100 °C −4 °F to +212 °F
Operating temperature range	0 °C to 80 °C +32 °F to +176 °F	−20 °C to 80 °C −4 °F to +176 °F
Storage temperature range	−20 °C to 80 °C −4 °F to +176 °F	−20 °C to 80 °C −4 °F to +176 °F

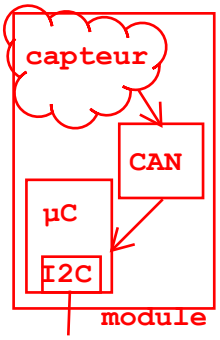
L'étude météorologique étant portée sur les zones chaudes tels que les déserts (Sahara, Gobi, etc.), le capteur n'a pas besoin de relever des température négatives. De plus, dans le cadre du projet InitCube, les jeunes ne seront pas amenés à faire de tels relevés (sauf pour tester les limites inférieures et supérieures de température).
Dans un soucis d'économie d'énergie et pour respecter au mieux les conditions réelles d'un CubeSat, le capteur devra consommer le moins possible donc on choisit le 3,3V.

Non! La consommation implique un courant et non une tension... A la rigueur, vous mettez ça :
Typical 4.5 mA (normal mode)
Typical 0.2 mA (sleep mode)
Typical 0.8 mA (stand-by mode)

Ce n'est pas qu'un capteur mais un module sinon vous aurez les mêmes remarques hors sujet qu'avec M. Gaboriaud.

4.2 Le fonctionnement du capteur

Pour comprendre comment coder le capteur il faut l'analyser pour en déduire son mode de fonctionnement. C'est le principe de la rétro-ingénierie. J'ai donc utilisé ce principe sur ~~on capteur~~ le module TPA64.



Plus communément appelé " Reverse engineering ", l'ingénierie inversée ou rétro-ingénierie représente l'étude d'un objet physique afin de découvrir la façon dont il fonctionne et dont il a été fabriqué.

Pour ma camera infrarouge (Capteur TPA64), j'ai utilisé un code en python trouvé sur internet pour ensuite étudier le capteur à l'aide d'un analyseur logique afin d'en déduire son mode de fonctionnement et pouvoir créer mon code en c++.

Après cela, j'ai pu configurer les différents registres de la caméra pour l'initialiser et demander une capture pour relever la température.

Explication du code python :

Le code en python utilisé (voir code en annexe), permet de récupérer la valeur de chaque pixel de la caméra et de stocker ces données dans un tableau. Cela nous permet de savoir comment monter notre code en c++ pour reprendre le même principe. Avec le code en c++ on récupère les données du capteur et on les enregistre dans une liste avant que ~~l'ordinateur de bord~~ ne la récupère.

la classe Segment vol

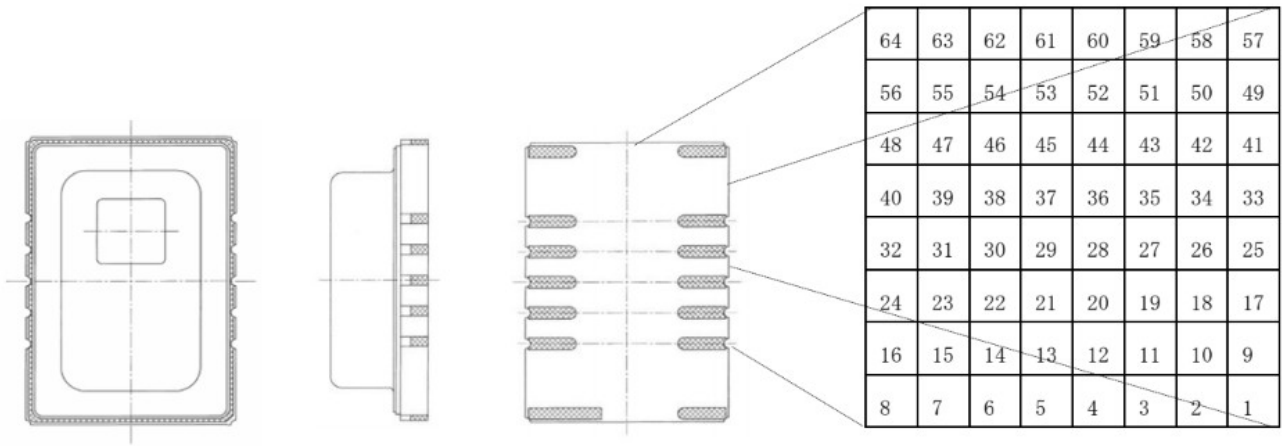
(En effet, l'ordinateur de bord exécute votre code...)

Organisation des pixels de la caméra :

6-7 Pixel Array & Viewing Field

(1)Pixel Array

Pixel Array from 1 to 64 is shown below.



Les pixels sont donc organisés de 1 à 64 en commençant en bas à gauche pour terminer en haut à droite lorsque l'on regarde la caméra de face.

Registres de la caméra :

Address	Register Name	Read/Write	Description	Initial value
0x00	Power control	R/W	Set operating mode (Normal, Sleep etc.)	0x00
0x01	Reset	W	Software Reset	0x00
0x02	Frame rate	R/W	Frame rate	0x00
0x03	INT control	R/W	Interrupt Function	0x00
0x04	Status	R	Interrupt Flag, low voltage Flag	0x00
0x05	Status clear	W	Interrupt Flag Clear	0x00
0x06			Reserved	
0x07	Average	R	Moving Average Output Mode	0x00
0x08	INT level-1	R/W	Interrupt upper value (Upper level)	0x00
0x09	INT level-2	R/W	Interrupt upper value (Upper level)	0x00
0x0A	INT level-3	R/W	Interrupt lower value (Lower level)	0x00
0x0B	INT level-4	R/W	Interrupt lower value (upper level)	0x00
0x0C	INT level-5	R/W	Interrupt hysteresis value (Lower level)	0x00
0x0D	INT level-6	R/W	Interrupt hysteresis value (Upper level)	0x00
0x0E	Thermistor-1	R	Thermistor Output Value (Lower level)	0x00
0x0F	Thermistor-2	R	Thermistor Output Value (Upper level)	0x00
0x10	INT-1	R	Pixel 1～8 Interrupt Result	0x00
0x11	INT-2	R	Pixel 9～16 Interrupt Result	0x00

Les registres sont tous à 0x00 par défaut mais ils sont configurables c'est ce que l'on fait pour changer le mode de la caméra ou encore récupérer sa température.

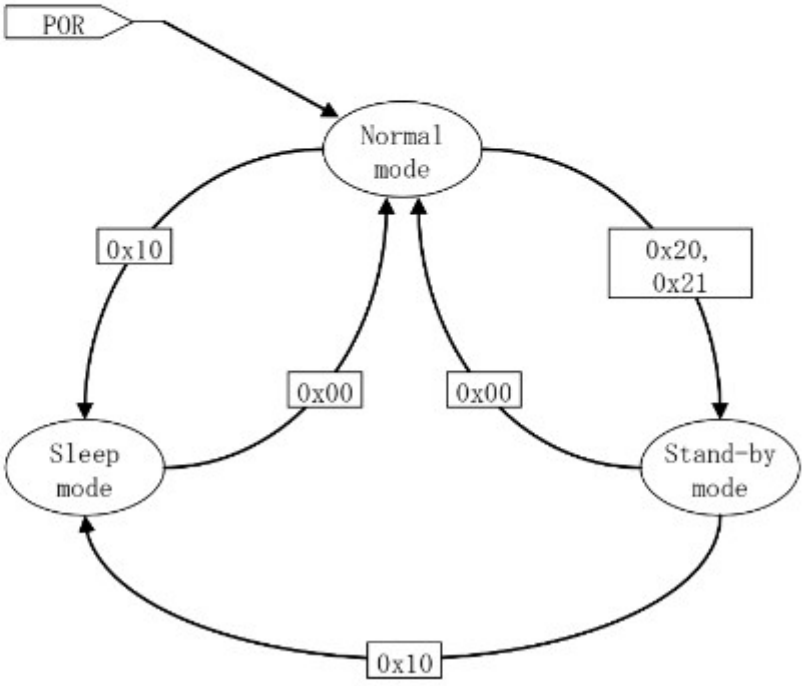
Address	Register Name	Read/Write	Description	Initial value
0x80	Pixel output	R	Pixel 1 Output Value (Lower Level)	0x00
0x81	Pixel output	R	Pixel 1 Output Value (Upper Level)	0x00
0x82	Pixel output	R	Pixel 2 Output Value (Lower Level)	0x00
0x83	Pixel output	R	Pixel 2 Output Value (Upper Level)	0x00
0x84	Pixel output	R	Pixel 3 Output Value (Lower Level)	0x00
0x85	Pixel output	R	Pixel 3 Output Value (Upper Level)	0x00
0x86	Pixel output	R	Pixel 4 Output Value (Lower Level)	0x00
0x87	Pixel output	R	Pixel 4 Output Value (Upper Level)	0x00
0x88	Pixel output	R	Pixel 5 Output Value (Lower Level)	0x00
0x89	Pixel output	R	Pixel 5 Output Value (Upper Level)	0x00
0x8A	Pixel output	R	Pixel 6 Output Value (Lower Level)	0x00
0x8B	Pixel output	R	Pixel 6 Output Value (Upper Level)	0x00
0x8C	Pixel output	R	Pixel 7 Output Value (Lower Level)	0x00
0x8D	Pixel output	R	Pixel 7 Output Value (Upper Level)	0x00
0x8E	Pixel output	R	Pixel 8 Output Value (Lower Level)	0x00
0x8F	Pixel output	R	Pixel 8 Output Value (Upper Level)	0x00
0x90	Pixel output	R	Pixel 9 Output Value (Lower Level)	0x00
0x91	Pixel output	R	Pixel 9 Output Value (Upper Level)	0x00
0x92	Pixel output	R	Pixel 10 Output Value (Lower Level)	0x00
0x93	Pixel output	R	Pixel 10 Output Value (Upper Level)	0x00
0x94	Pixel output	R	Pixel 11 Output Value (Lower Level)	0x00
0x95	Pixel output	R	Pixel 11 Output Value (Upper Level)	0x00

Chaque pixel de la caméra à son adresse à l'intérieur de cette dernière on peut donc écrire pour placer un pointeur sur l'adresse d'un pixel et ensuite faire une lecture de la valeur mesurée. Chaque pixel est codé sur deux octets, c'est pourquoi on fait une écriture / lecture toutes les deux adresses en hexadécimal. Soit par exemple l'adresse 0x80 pour le premier pixel puis 0x82, 0x84 etc.

Activation et désactivation de l'instrument :

Command	Operating mode
0x00	Normal mode
0x10	Sleep mode
0x20	Stand-by mode (60sec intermittence)
0x21	Stand-by mode (10sec intermittence)

【Trandition Diagram of Operating mode】

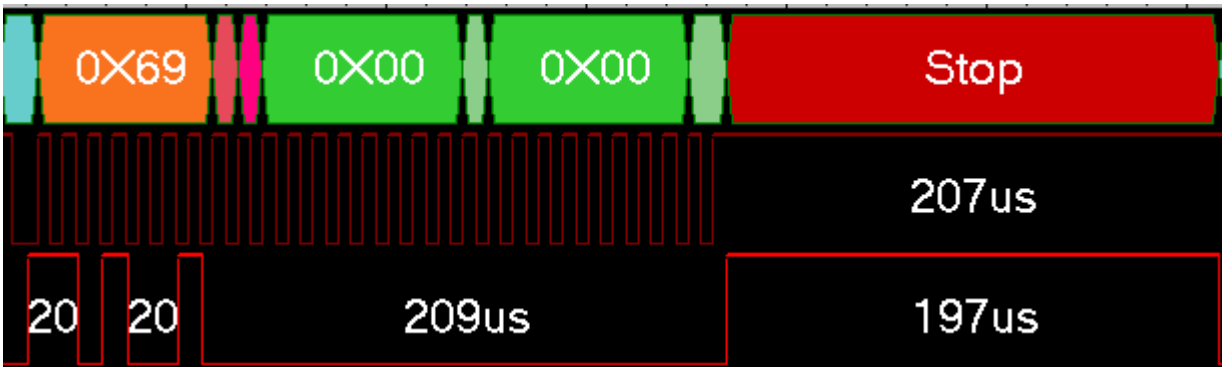


On a aussi le contrôle sur l'activation et désactivation de l'instrument. Des modes sont prédéfinis avec 0x00 pour le mode normal , 0x10 pour le mode sleep et 0x20 / 0x21 pour le mode stand_by suivant la durée nécessaire.

Rappeler :
Typical 4.5 mA (normal mode)
Typical 0.2 mA (sleep mode)
Typical 0.8 mA (stand-by mode)
Le mode sleep nous permettra de réduire la consommation et donc d'augmenter la durée batterie.

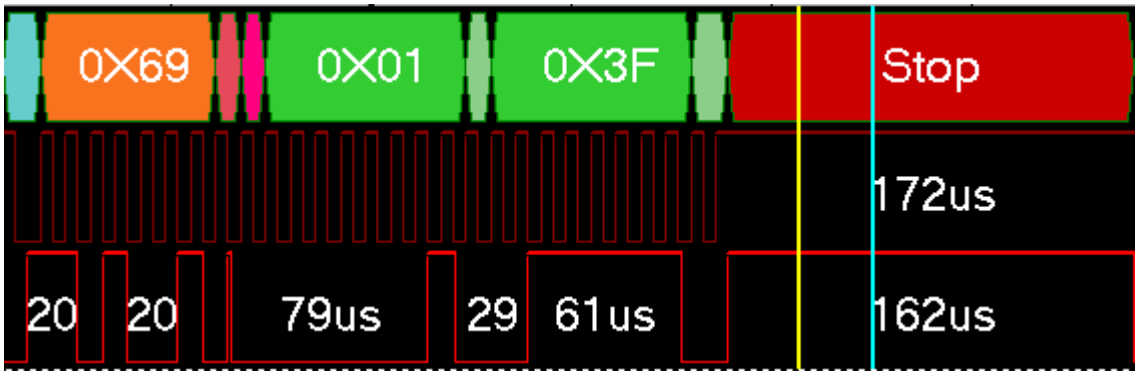
Capture configuration des registres :

Registre 1 :



Le registre 1 à l'adresse 0x00 est initialisé avec la valeur 0x00. Ce registre permet de configurer le mode de la caméra (0x00 = NORMAL, 0x10 = SLEEP, 0x20 et 0x21(selon la durée) = STAND_BY).

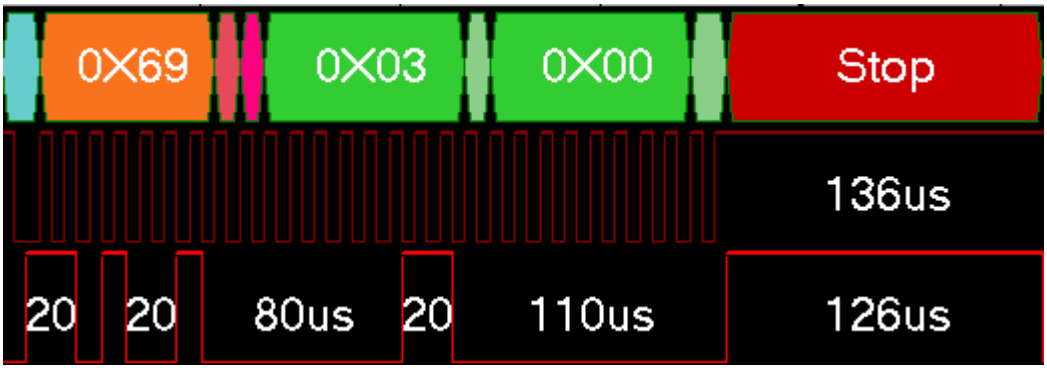
Registre 2 :



Le registre 2 à l'adresse 0x01 est initialisé avec la valeur 0x3F.

Cela correspond à une remise à 0 des registres de stockage des pixels de la camera.

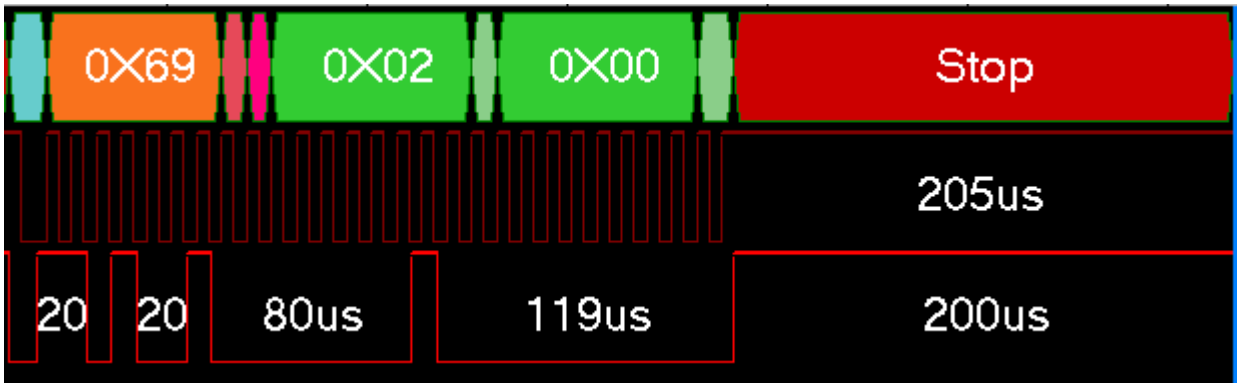
Registre 4 :



Le registre 4 à l'adresse 0x03 est initialisé avec la valeur 0x00.

Celui la n'est pas vraiment utile car on n'utilise pas les registres et la broche INT d'interruption...

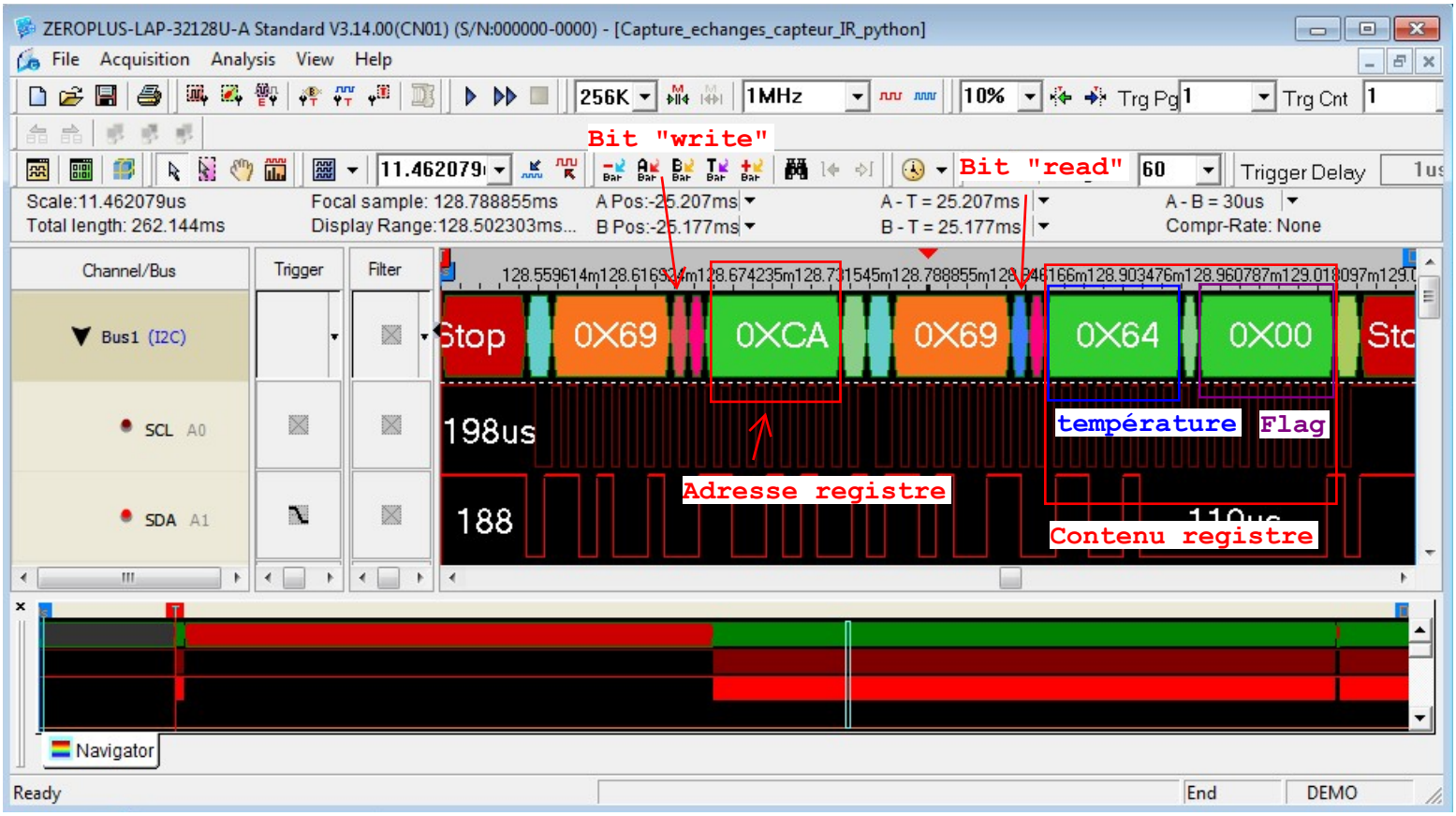
Registre 3 :



Le registre 3 à l'adresse 0x02 est initialisé avec la valeur 0x00.

Les données de température du pixel 1 × 64 × 0x80 × 0xFF sont renouvelées en une fois sans aucune instruction du maître externe.
La période du renouvellement dépend du nombre d'images par seconde configuré dans ce registre. Il s'appelle "Frame Rate Register".
Initialisé à 0x00, il correspond à une fréquence des captures de 10 images par secondes.
En le configurant à la valeur 0x01, on aurait 1 image par seconde.
Compte tenu de notre contexte d'utilisation, ce registre ne sera pas exploité. En effet, la camera est mise en mode "sleep" entre chaque relevé.

Test de la récupération des données lues par le capteur :

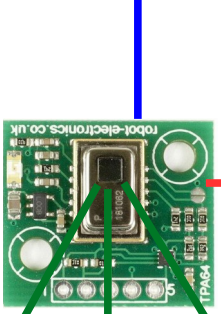


Après avoir étudié le capteur à l'analyseur logique on peut voir que son adresse sur la Raspberry est 0x69. Sur cette capture on voit qu'on écrit sur l'adresse 0x69 pour placer le pointeur sur l'adresse 0xCA (adresse du pixel 38) et on fait une lecture qui nous retourne les deux octets 0x64 et 0x00. Toutes les adresses des pixels sont consignées dans la documentation sur les registres de la caméra.

Pour conclure, le mode de fonctionnement du capteur est bien déterminé et les registres sont aux bonnes adresses. On peut maintenant coder les classes.

Schéma :

A l'intérieur du cube



Communication via liaison I2C

Capture la température surfacique ~~des nuages~~ du sol des régions chaudes...

Bein oui... On n'a que des températures positives...

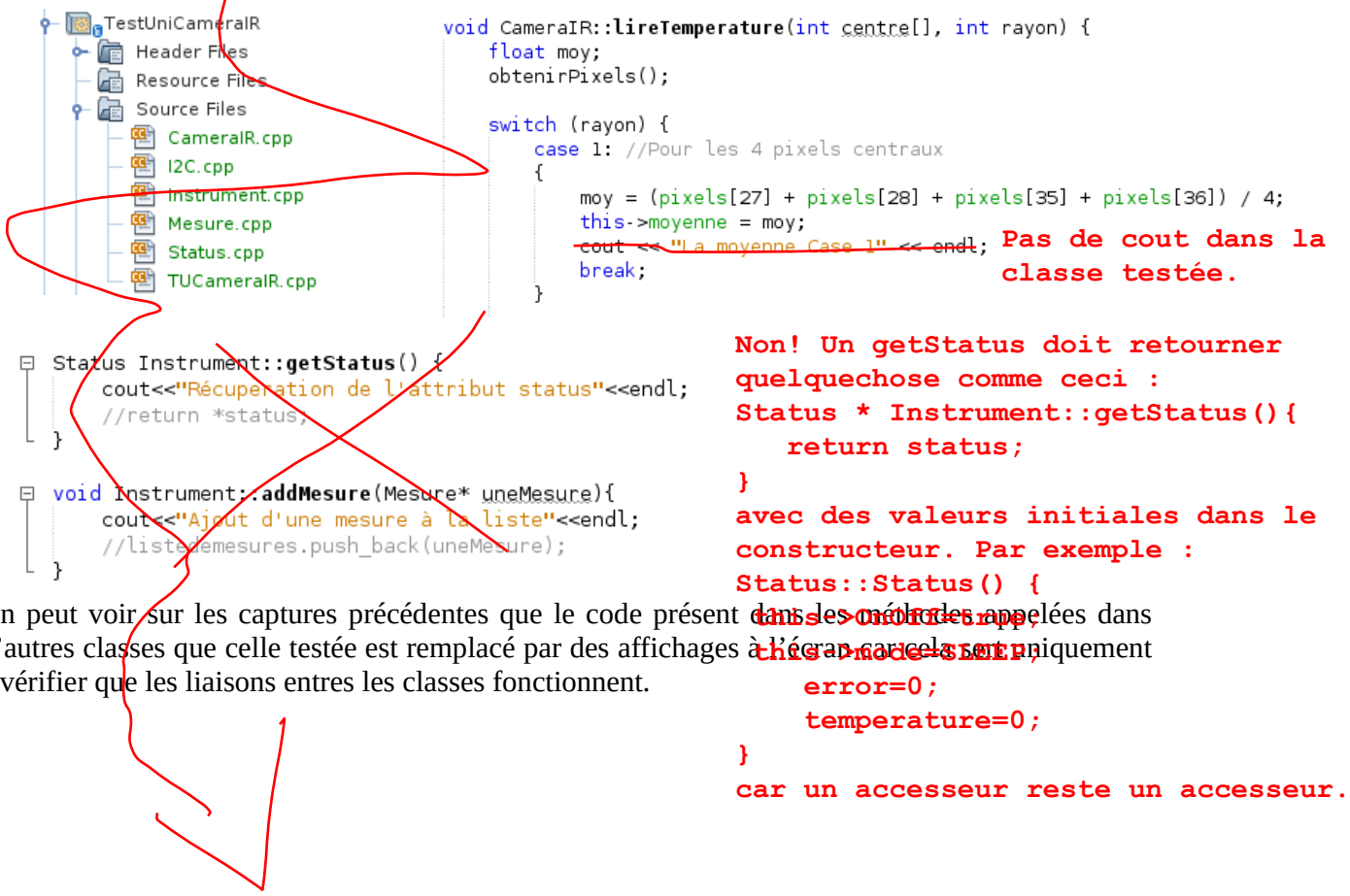
5. Codage des classes

5.1 Rédaction et réalisation des tests unitaires

Pour commencer, avant de coder mes différentes classes, j'ai rédigé les fiches de tests unitaires pour chacune d'entre-elles (voir à la fin du dossier). Ces fiches permettent de décrire un scénario particulier et de déterminer les différents traitements que la classe testée devra effectuer. Toutes les classes ne sont pas utilisées à chaque fois, seulement les classes nécessaires pour exécuter toutes les méthodes de la classe testée. Après avoir fini les fiches, je suis directement passé au codage.

Capture sur le logiciel Netbeans pour illustrer la notion de tests unitaires :

Test unitaire CameraIR :



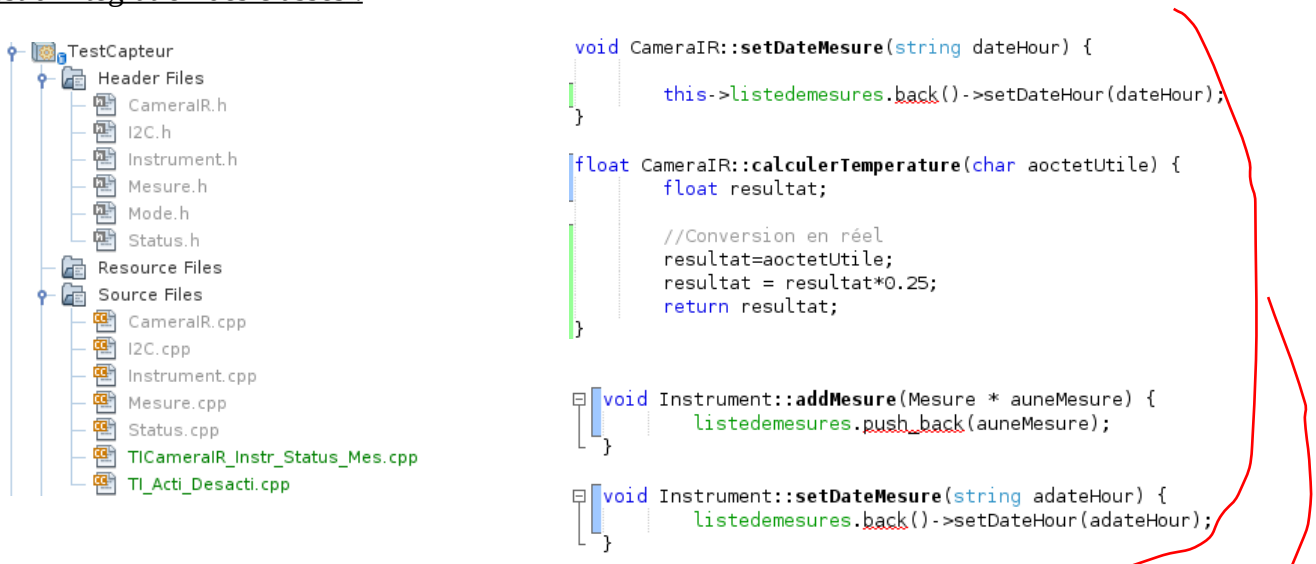
On peut voir sur les captures précédentes que le code présent dans les méthodes appelées dans d'autres classes que celle testée est remplacé par des affichages à l'écran car cela sert uniquement à vérifier que les liaisons entres les classes fonctionnent.

5.2 Rédaction et réalisation des tests d'intégration

Je n'ai eu qu'un test d'intégration à réaliser. Je devais m'occuper de l'intégration entre toutes mes classes. Pour cela j'ai rédigé une fiche de la même façon que pour mes tests unitaires (voir à la fin du dossier). Une fois la fiche réalisée j'ai récupéré toutes les codes de mes tests unitaires pour les mettre ensemble et en ajoutant certaines spécificités, j'ai pu réaliser mon test unitaire.

Capture sur le logiciel Netbeans pour illustrer la notion de tests d'intégration:

Test d'intégration des classes :



Lors du test d'intégration, le but est de vérifier que toutes les classes codées lors des tests unitaires fonctionnent si on les assemble. Pour cela il est parfois nécessaire de rajouter des méthodes qui permettront de lier toutes les classes entre-elles.

Ce code a peu d'intérêt,
Il vaut mieux montrer du code où
vous faites appel quex registres que vous citez
sur les captures à l'analyseur...

6. Exécution du code de la Caméra IR

6.1 Organisation en matrice

Pour commencer j'ai voulu vérifier que le code de la caméra était fonctionnel, j'ai donc récupéré la valeur de chaque pixels puis j'ai affiché une matrice.
Matrice affichée lors de l'exécution du code :

```
pi@raspberrypi:~/Projet $ ./NeoMatriceCapteur
      1       2       3       4       5       6       7       8
  1 17.750  17.500  17.000  17.000  17.000  17.000  17.750  17.500
  2 17.500  17.750  17.250  17.750  17.500  18.000  17.750  17.500
  3 17.000  18.000  17.500  17.500  17.750  17.750  17.750  18.000
  4 16.500  16.750  17.500  17.000  17.000  17.000  17.000  17.500
  5 16.750  16.750  18.000  17.000  17.750  17.250  18.000  17.750
  6 17.000  17.500  17.000  17.000  17.750  17.750  16.750  17.500
  7 17.000  17.250  17.250  17.250  17.500  16.500  17.250  16.750
  8 16.250  17.250  16.750  17.000  17.000  17.000  17.500  17.500
pi@raspberrypi:~/Projet $
```

Pour vérifier la fonctionnalité du capteur, j'ai testé avec un briquet pour avoir des températures chaudes et avec un freezer pour les températures froides.

```
pi@raspberrypi:~/Projet $ ./NeoMatriceCapteur
      1       2       3       4       5       6       7       8
  1 23.500  25.500  26.500  27.250  25.500  23.250  22.750  23.250
  2 24.500  25.750  33.750  43.500  30.500  25.250  25.000  25.250
  3 23.750  22.750  35.500  74.000  34.000  25.000  24.750  25.500
  4 22.250  22.000  25.250  50.250  37.000  25.250  25.750  26.750
  5 21.500  21.750  23.250  32.250  31.500  26.000  26.250  26.750
  6 21.500  20.750  21.750  25.500  28.750  27.250  25.250  26.750
  7 21.500  20.500  21.250  23.500  27.750  27.750  26.250  26.000
  8 21.250  20.750  20.500  21.500  25.250  26.500  25.750  26.000
pi@raspberrypi:~/Projet $
```

On voit sur la capture ci-dessus (celle avec le briquet) que la température au centre monte jusqu'à 74°C et l'extérieur oscille autour de 25°C qui correspond à la température des surfaces de la pièce.

```
pi@raspberrypi:~/Projet $ ./NeoMatriceCapteur
       1       2       3       4       5       6       7       8
  1  18.500  15.000  6.750  7.250  6.500  15.250  19.500  21.500
  2  17.500  13.750  5.750  5.750  6.250  14.500  20.750  22.000
  3  19.250  14.000  6.000  6.500  6.750  14.250  19.250  23.500
  4  17.500  13.750  6.250  6.000  6.500  12.500  18.500  22.250
  5  17.500  13.750  7.750  7.500  8.000  13.000  17.500  23.000
  6  17.500  13.500  8.000  8.250  9.000  14.000  17.500  19.250
  7  18.250  14.500  8.750  9.000  8.500  13.750  18.000  22.250
  8  17.000  15.750  8.500  6.000  6.750  15.000  18.000  22.750
pi@raspberrypi:~/Projet $
```

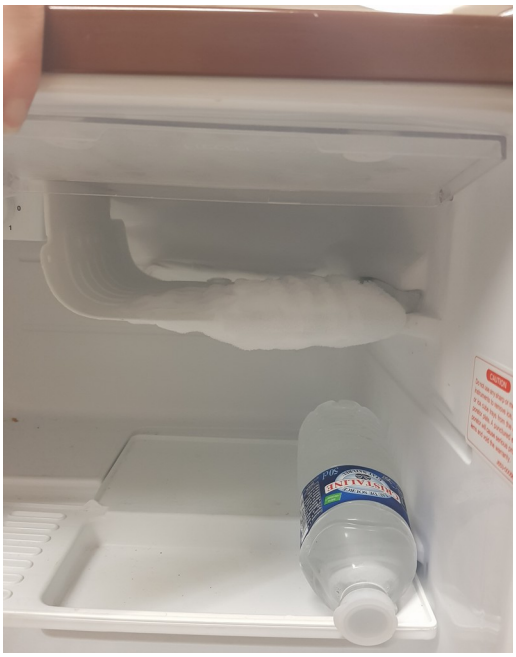
On voit au contraire sur la capture ci-dessus (effectuée avec une bouteille d'eau froide), que les pixels centraux sont cette fois autour de 7°C par rapport au reste de la pièce autour de 20°C.

7°C était la température dans le réfrigérateur.

6.2 Test des limites du capteur et affichage des erreurs

Températures négatives :

Pour tester la conformité de la documentation j'ai fais une capture dans un freezer pour voir l'erreur produite avec des températures négatives.



La matrice retournée est celle-ci :

```
pi@raspberrypi:~/Projet $ ./NeoMatriceCapteur
       1       2       3       4       5       6       7       8
  1  15 - 43.5  15 - 36.0  15 - 33.2  15 - 34.5  15 - 31.8  15 - 31.8  15 - 31.8  15 - 32.2
  2  15 - 37.2  15 - 34.5  15 - 33.5  15 - 34.2  15 - 34.0  15 - 35.0  15 - 36.0  15 - 34.8
  3  15 - 35.2  15 - 36.8  15 - 35.2  15 - 36.5  15 - 38.8  15 - 45.5  15 - 40.2  15 - 39.5
  4  15 - 37.8  15 - 37.2  15 - 39.5  15 - 45.8  15 - 56.5  15 - 61.8  15 - 61.2  15 - 47.2
  5  15 - 42.5  15 - 48.0  15 - 57.2  15 - 62.5  0 - 0.5  0 - 0.2  15 - 63.0  15 - 58.8
  6  15 - 56.0  15 - 62.5  0 - 1.0  0 - 0.0  0 - 0.8  0 - 0.0  15 - 61.8  15 - 63.2
  7  15 - 62.2  0 - 0.2  0 - 0.5  15 - 63.8  15 - 63.2  15 - 62.5  15 - 62.8  15 - 63.0
  8  15 - 63.5  0 - 0.0  15 - 62.5  15 - 63.0  15 - 62.5  15 - 63.8  0 - 0.2  0 - 0.2
pi@raspberrypi:~/Projet $
```

Pour cette capture, j'ai affiché l'octet de température et l'octet d'état (flag) de chaque pixel.

On voit donc que le capteur relève bien les températures supérieures ou égales à 0 mais affiche un code d'erreur égal à 15 lorsque la température est trop faible. Ce 15 correspond au quatre premiers bits de l'octet de poids fort qui sont mis à 1 soit 1111 = 15.

Capteur en mode SLEEP :

7. Fiches de tests unitaires

7.1 Test unitaire de Mesure

Ce TU comme celui de Status n'est pas très utile

Projet	Nom de l'élément testé	Type de l'élément testé	Version	Incrément
InitCube	Mesure	Classe	1	1

Description :	Le but est de tester la récupération de la mesure effectuée par la caméra et le stockage de cette mesure dans un tableau avant de le transmettre plus tard.

Description du test

Scénarios concernés	La classe Mesure récupère et stocke les données récupérées de la camera.
Description	La classe Mesure va récupérer les relevés de la camera et les stocker dans une variable avant de les transmettre à l'instrument. Elle va aussi attribuer la date et l'heure à la mesure ainsi que l'unité dans laquelle la mesure doit être exprimée.
Environnement nécessaire	Netbeans, RaspberryPI.
Situation initiale	Soit la mesure est planifiée par la mission, donc va s'exécuter toute seule, soit elle est commandée par le segment sol.
Classes de tests nécessaires	CameraIR, Instrument, I2C.
Nom du script	TUMesure.cpp

Auteur du test

Nom du testeur	Date	Conclusions	Validation
			Oui - Non

Description du script

N°	Traitement	Paramètres en entrée	Résultats attendus
1	Mesure()		Déclaration de l'unité de la mesure
2	Mesure(float valeur)		Enregistre la mesure dans une variable
3	setDateHour(string dateHour)		Attribue la date et l'heure pour chaque mesure
4	getUnite()		Retourne l'unité de la mesure choisie
5	getDateHour(string dateHour)		Retourne la date et l'heure de la mesure effectuée
6	getMesure()		Retourne la valeur de la mesure
7			
8			
9			

Problèmes identifies

N°	Traitement	Résultats obtenus	Gravité de l'erreur

7.2 Test unitaire d'Instrument

Projet	Nom de l'élément testé	Type de l'élément testé	Version	Incrément
InitCube	Instrument	Classe	1	1

Description :	Le but est de créer un instrument et de pourvoir ajouter des mesures dans une liste.

Description du test

Scénarios concernés	La caméra IR doit effectuer une mesure et la retourner à l'ordinateur de bord afin de l'envoyer ensuite au segment sol.
Description	La classe Instrument va appeler les différentes méthodes qui la compose afin de récupérer toutes les informations sur la camera IR dans notre cas.
Environnement nécessaire	Netbeans, RaspberryPI.
Situation initiale	L'instrument doit être activé pour lancer la récupération des mesures.
Classes de tests nécessaires	Mesure, Status, CameraIR
Nom du script	TUInstrument.cpp

Auteur du test

Nom du testeur	**Date**	**Conclusions**	**Validation**
			Oui - Non

Dossier Personnel

Description du script

N°	Traitement	Paramètres en entrée	Résultats attendus
1	Instrument()		Crée un nouveau status
2	getMesures()		Retourne la liste des mesures effecuées
3	addMesure(Mesure* uneMesure)		Ajoute une mesure à la liste de mesures
4	getStatus()		Retourne le status de l'instrument
5			
6			
7			
8			
9			

Problèmes identifies

N°	Traitement	Résultats obtenus	Gravité de l'erreur

7.3 Test unitaire de CameraIR

Projet	Nom de l'élément testé	Type de l'élément testé	Version	Incrément
InitCube	CameraIR (Infra Rouge)	Classe	1	1

Description :	Le but est de capturer la température surfacique du corps présent devant la caméra, on doit pouvoir récupérer la valeur de chaque pixel et calculer une température moyenne en fonction du rayon de capture choisi.

Description du test

Scénarios concernés	La caméra IR doit effectuer une mesure et stocker les valeurs pour les envoyer dans la classe Mesure par la suite.
Description	La Camera doit effectuer des mesures soit planifiées par une mission soit sur demande du segment sol, et stocker les valeurs pour les faire remonter par Mesure, puis Instrument.
Environnement nécessaire	Netbeans, RaspberryPI.
Situation initiale	La camera est désactivée et on doit donc s'y connecter pour lancer une mesure.
Classes de tests nécessaires	I2C
Nom du script	TUCameraIR.cpp

Auteur du test

Nom du testeur	**Date**	**Conclusions**	**Validation**
			Oui - Non

Description du script

N°	Traitement	Paramètres en entrée	Résultats attendus
1	CameraIR()		Permet de placer un pointeur sur l'adresse du capteur
2	obtenirPixels()		Récupère la valeur de chaque pixel
3	setDateMesure(string dateHour)		Ajoute la date et l'heure à la liste des mesures effectuées
4	calculerTemperature(string dateHour)		Convertit la valeur retournée en réel
5	lireTemperature(int rayon)	Rentrer le rayon de capture	Retourne la moyenne des pixels selon le rayon choisi
6	activer()	Activer l'instrument (O/N)	Activation ou non de l'instrument
7	desactiver()	Désactiver l'instrument (O/N)	Désactivation ou non de l'instrument
8	obtenirMode()		Retourne le mode actuel de fonctionnement du capteur
9	obtenirTempInst()		Retourne la température de l'instrument

Décrire comment on voit que cela fonctionne...

Problèmes identifies

N°	Traitement	Résultats obtenus	Gravité de l'erreur

8. Fiche du test d'intégration

Projet	Nom de l'élément testé	Type de l'élément testé	Version	Incrément
InitCube	Instrument / CameraIR / Status / Mesure	Classes	1	1

Description :	Après la réalisation des tests unitaires, j'ai associé toutes mes classes pour réaliser le test d'intégration

Description du test

Scénarios concernés	La classe Status contient les états de l'instrument. La caméra IR doit effectuer une mesure et la retourner à l'ordinateur de bord afin de l'envoyer ensuite au segment sol. La classe Mesure récupère et stocke les données récupérées de la camera. La caméra IR doit effectuer une mesure et stocker les valeurs pour les envoyer dans la classe Mesure par la suite.
Description	Nous allons tester la communication entre les différentes classes codées et observer si les résultats observés sont corrects.
Environnement nécessaire	Netbeans, RaspberryPI.
Situation initiale	L'ordinateur de bord demande d'effectuer une mesure.
Classes de tests nécessaires	Instrument, CameraIR, Status, Mesure.
Nom du script	

Auteur du test

Nom du testeur	**Date**	**Conclusions**	**Validation**
			Oui - Non

Dossier Personnel

Description du script

N°	Traitement	Paramètres en entrée	Résultats attendus
1	Lire la température	Choisir le rayon de capture pour la température	La caméra effectue une mesure puis on la récupère pour la transmettre
2	Configurer le status	Voulez-vous activer(ou désactiver) l'instrument ?	On récupère le status et la température de l'instrument
3			
4			
5			
6			
7			
8			
9			

Problèmes identifies

N°	Traitement	Résultats obtenus	Gravité de l'erreur

9. Journal de bord du projet

Semaine	Activité principale
1	Recherche d'information sur le capteur TPA64 et étude des liaisons I2C.
2	Recherche d'un code en python pour la caméra IR, installation des librairies nécessaires, analyse des échanges à l'analyseur logique et réunion avec notre client.
3	Analyse UML du système, en organisation du premier code de la caméra.
4	Préparation de la revue 1, puis amélioration du diaporama.
5	Début du codage en c++ des classes CameraIR, Mesure et Status, et de la rédaction des tests unitaires.
6	Écriture du makefile et des tests unitaires, test des classes CameraIR, Mesure et Status.
7	Écriture et réalisation du test d'intégration.
8	Finition de l'intégration et test sur la raspberry du projet.
9	Correction collégiale de la classe I2C, visite du CNES et rédaction de la documentation sur la rétro-ingénierie.
10	Rédaction d'un compte rendu de réunion sur la modification de la classe I2C et finition de la documentation sur la rétro-ingénierie.
11	Modification de mes classes pour être en accord avec la nouvelle classe I2C et intégration de mes classes sur la raspberry du projet.
12	Préparation de la documentation commune du projet et personnelle sur le capteur TPA64.
13	Finalisation du diaporama pour l'oral.
14	Entraînement pour le passage à l'oral et présentation de la démo.
15	Entraînement pour le passage à l'oral et présentation de la démo.