MSE | MASTER OF SCIENCE IN ENGINEERING

Master of Science HES-SO in engineering

Av. de Provence 6

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Master of Science HES-SO in Engineering

Orientation : Technologies de l'information et de la communication

Plate-forme d'assistance au télépilotage

Fait par

Christian MULLER

Sous la direction du

Prof. David Grünenwald

HE-ARC

Mandant : Cyrille Henry

Lieu, HES-SO//Master, 6 juin 2014

Accepté par la HES-SO//Master (Suisse, Lausanne) sur proposition de Cyrille Henry.

Prof. David Grünenwald	Mandant	Prof. Fariba Moghaddam Bützberger
Conseiller	Cyrille Henry	Responsable de la filière MSE

1 TABLE DES MATIÈRES

2 ABRÉVIATIONS

OSD	On Sceen Display
RPi	Raspberry Pi
IMU	Inertial Mesurement Unit – Centrale inertielle (accéléromètre, gyroscope et magnétomètre)
Ips, fps	Image par seconde, frame per second

3 RÉSUMÉ

Ce travail consiste à mettre en place sur une Raspberry Pi (sur Linux) une application permettant d'afficher à l'écran, par-dessus le flux vidéo de la caméra, un calque avec des informations de navigation (sous forme d'affichage tête haute). Cette assistance permet de faciliter le pilotage à distance, plus particulièrement lorsque l'appareil n'est pas en vue ou que les conditions visuelles sont mauvaises. Une caméra, une centrale inertielle et un GPS ont été reliés à la RPi. L'objectif a été séparé en trois tâches distinctes : l'affichage de la vidéo, la récupération des données des capteurs avec le calcul des données de navigation ainsi que l'affichage des informations sous forme graphique. Chacune de ces tâches a ensuite été analysée pour en déterminer les facteurs critiques et les priorités à donner dans sa réalisation. Différents outils et langages ont été étudiés afin de vérifier s'ils permettaient de la réaliser. La comparaison a alors permis de choisir les bons moyens pour réaliser l'objectif. La principale difficulté se situe au niveau des performances de la plate-forme, plus particulièrement de son CPU. L'accent est également mis sur la simplicité d'utilisation des outils afin que le résultat puisse être modifié facilement.

La première des tâches est réalisée à l'aide de la librairie native raspivid. Le programme enregistre le flux vidéo en HD (720p) au format h264 et affiche ce même flux sur la sortie vidéo. Un module Python vient d'interfacer sur la librairie native afin de faciliter l'utilisation.

La seconde tâche, la récupération des valeurs et le calcul des données de navigation complémentaires, est réalisée à l'aide de Python. L'IMU est interrogée à une fréquence de 25 Hz alors que le GPS l'est à 1 Hz. Les données de l'IMU sont alors traitées à la volée pour en faire ressortir les valeurs utiles, comme l'accélération sur chaque axe. Les informations complémentaires sont calculées immédiatement et on détermine ainsi l'état du drone (roulis, tangage, lacet). Les données du GPS, qui sont composées de données absolues et n'arrivent qu'une fois par seconde, sont modulées en fonction des données relatives de l'IMU. L'ensemble de ces données est ensuite enregistré dans un fichier CSV.

La troisième tâche, l'affichage des données de navigation à l'écran, est effectuée par un programme en Qt, à l'aide de QML. La spécification de la vue est décrite dans un fichier QML qui permet de dessiner des composants représentatifs des informations de navigation. Cette vue est alimentée par un contrôleur en Javascript, aidé de composants Qt en C++ qui permettent la lecture du fichier CSV afin d'en extraire les données de navigation.

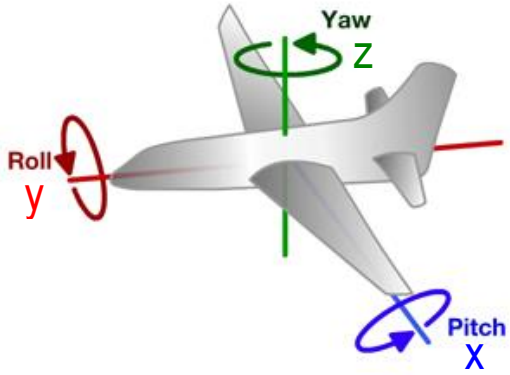
L'exécution simultanée de ces trois tâches permet de réaliser l'objectif global.

Key Words: Drone, Pilotage, IMU, GPS, OpenGL ES, Qt5, QML, Raspberry Pi, ARM, Linux

4 INDICATIONS DE LECTURE

4.1 Axes

Les lettres x, y et z sont utilisées pour représenter les axes relatifs au drone (selon le schéma ci-dessous) alors que les lettres X, Y et Z sont utilisées pour représenter les axes absolus, X étant l'axe est-ouest, Y étant l'axe nord-sud et Z l'axe vertical.



5 INTRODUCTION

L'utilisation d'avions de modélisme ou d'autre type de drones devient de plus en plus courante, que ce soit dans le privé, pour des loisirs, ou dans le monde professionnel. Les récentes améliorations techniques rendent leur pilotage toujours plus accessible tout en réduisant fortement leur cout. Dans les applications, on retrouve fréquemment une utilisation à grande distance, où le drone sort du champ visuel du pilote. Afin de pouvoir piloter dans de telles conditions, l'image d'une vidéo embarquée n'est parfois pas suffisante pour diriger efficacement le drone. Afin de résoudre ce problème, on ajoute par-dessus l'image des informations qui représentent l'état du drone, tels qu'une ligne d'horizon, des informations concernant la vitesse et la direction.

Il existe diverses solutions qui répondent à ce besoin. La plupart se présentent sous la forme d'une carte dédiée avec un traitement dédié (par un microcontrôleur). Ces appareils ne peuvent alors pas être modifiés car ils sont vendus en produit finis et ne peuvent pas être reprogrammés. Il manque aujourd'hui une solution low cost, ouverte et flexible qui permet de configurer l'affichage.

5.1 Objectif du projet

L'objectif du travail est de mettre en place une plate-forme de capture et de diffusion vidéo avec l'ajout d'informations spatiales à la volée, ceci dans le but de l'installer sur un avion de modélisme pour le pilotage à distance.

Partie 1 - Conception

6 CAHIER DES CHARGES

6.1 Spécifications matérielles

La Raspberry PI est utilisée comme plate-forme d'acquisition et de traitement des informations. Deux périphériques de géolocalisation sont ajoutés, un contrôleur GPS et une IMU. Une caméra externe est ajoutée pour capturer un flux vidéo.

6.2 Spécifications techniques

L'implémentation se fait sur Linux, avec la distribution Raspbian.[1]

A) Traitement vidéo

La Raspberry PI est chargée de récupérer le flux vidéo de la caméra, de l'enregistrer sur la mémoire, d'ajouter des informations en overlay sur l'image et de retransmettre le flux vidéo avec l'overlay d'informations sur la sortie vidéo analogique de la Raspberry. Le flux avec l'OSD est également enregistré, si possible.

La récupération du flux vidéo se fait à l'aide de la librairie Raspivid. Un pré-traitement de l'image est alors effectué pour corriger une éventuelle déformation de l'image de la caméra due à la lentille grand-angle.

Une librairie OpenGL ES est développée pour dessiner du texte et des formes basiques sur le flux vidéo. Cette librairie permet de dessiner une surcouche à l'image regroupant des informations essentielles de navigation.

Pourront être dessinés à l'aide de la librairie :

- texte (taille / pos / couleur)
- ligne (longueur / centre / couleur)
- rectangle plein (longueur / largeur / position / rotation / couleur)

Image d'illustration de ce à quoi pourrait ressembler le HUD :

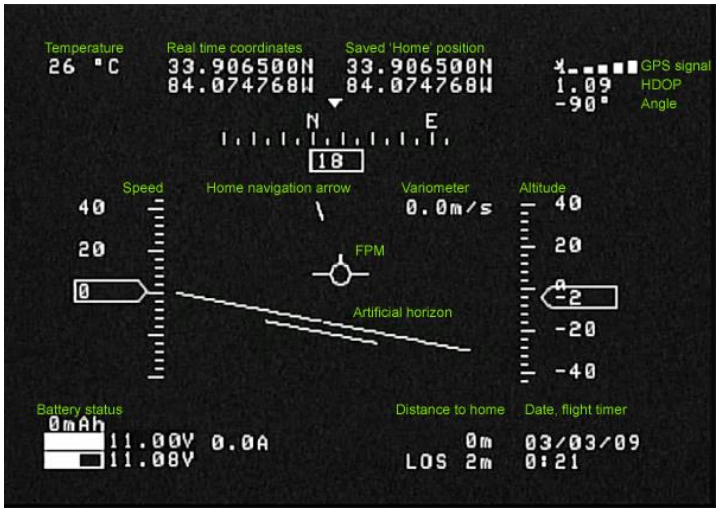


Image 1[2]

[1] Errata : spécifié implicitement mais pas présent dans le cahier des charges original

[2] Source : http://www.rc-eagleeye.cz/

B) Traitement des données du contrôleur GPS

Les données du contrôleur GPS sont récupérées par la Raspberry PI via le port série. Une librairie permet d'extraire ou de calculer diverses informations des données GPS, notamment les données suivantes :

- altitude
- vitesse au sol
- direction
- direction et distance du point de départ
- vitesse de montée
- date et heure
- coordonnées GPS
- qualité du signal

L'historique des positions du GPS est stocké en mémoire dans un fichier GPX (ou équivalent) afin de pouvoir retracer le parcours après le vol.

C) Traitement des données inertielles

Les données de l'IMU sont récupérées par la Raspberry PI par I2C. Les données sont synthétisées et traitées afin d'en extraire les informations utiles pour le pilotage d'un drone.

Données récupérées :

- magnétomètre
- accéléromètre
- gyroscope

Données calculées :

- orientation absolue
- ligne d'horizon

D) Unification

Les données de localisation géographique et spatiales sont combinées et affichées sur le flux vidéo à la volée, de manière à ce que la sortie vidéo, diffusée par RF, permette de piloter un drone qui n'est pas à portée de vue.

7 PLATE-FORME

Afin de mieux cerner les difficultés et spécificités de la réalisation du cahier des charges, il convient d'étudier les caractéristiques du matériel.

La plate-forme utilisée est un Raspberry Pi (modèle B). C'est une mono carte low-cost basée sur une architecture ARM. En voici les principaux composants[3] :

SoC	**Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, et 1 port USB)**
CPU	700 MHz ARM1176JZF-S core (ARM11)
GPU	Broadcom VideoCore IV,OpenGL ES 2.0,OpenVG 1080p30 H.264 high-profile encode/decode
RAM	512 MiB

Le CPU utilisé est assez peu performant comparé à ce qu'on peut trouver actuellement dans les téléphones par exemple. De loin insuffisant pour du traitement graphique.

Le GPU est lui cependant bien plus performant, équivalent à celui d'une Xbox (1ère génération), avec une puissance de traitement d'environ 24GFLOPS. C'est donc le GPU qui est la principale source de calcul sur notre plate-forme, suffisant pour des rendus 3D. Cet élément jouera un rôle clé dans la conception du programme d'affichage des informations.

7.1 Caméra

La caméra est placée à l'avant du dispositif et permet d'avoir un flux vidéo HD à plus de 25 ips. Nous utilisons la caméra officielle de la RPi.

7.2 Centrale inertielle

La centrale inertielle est composée de plusieurs capteurs. Ces derniers nous permettent de connaitre l'état relatif du drone. Le module utilisé est un 10DOF IMU GY-88[4] (Barometer, Gyroscope, Accelerometer, Magnetometer). Il est connecté à la RPi par un bus I2C et alimenté par la RPi en 3V3.

Ce module contient trois composants qui sont tous connectés au même bus I2C :

- MPU-6050 (6-Axis Angular Rate Sensor & Accelerometer)
- HMC5883L (3-Axis Digital Compass)
- BMP085 (Barometric Pressure Sensor)

7.3 Module GPS

L'*Ultimate GPS Breakout*[5], comporte un contrôleur GPS et une antenne céramique, ce qui le rend utilisable sans antenne externe. On peut toutefois y connecter une antenne alternative pour une meilleure réception.

Le module se connecte à la RPi par une ligne série. Il est alimenté en 5V par la RPi.

[3] Source : eLinux

[4] Référence : play-zone.ch

[5] http://www.adafruit.com/products/746

8 SÉPARATION DES TÂCHES

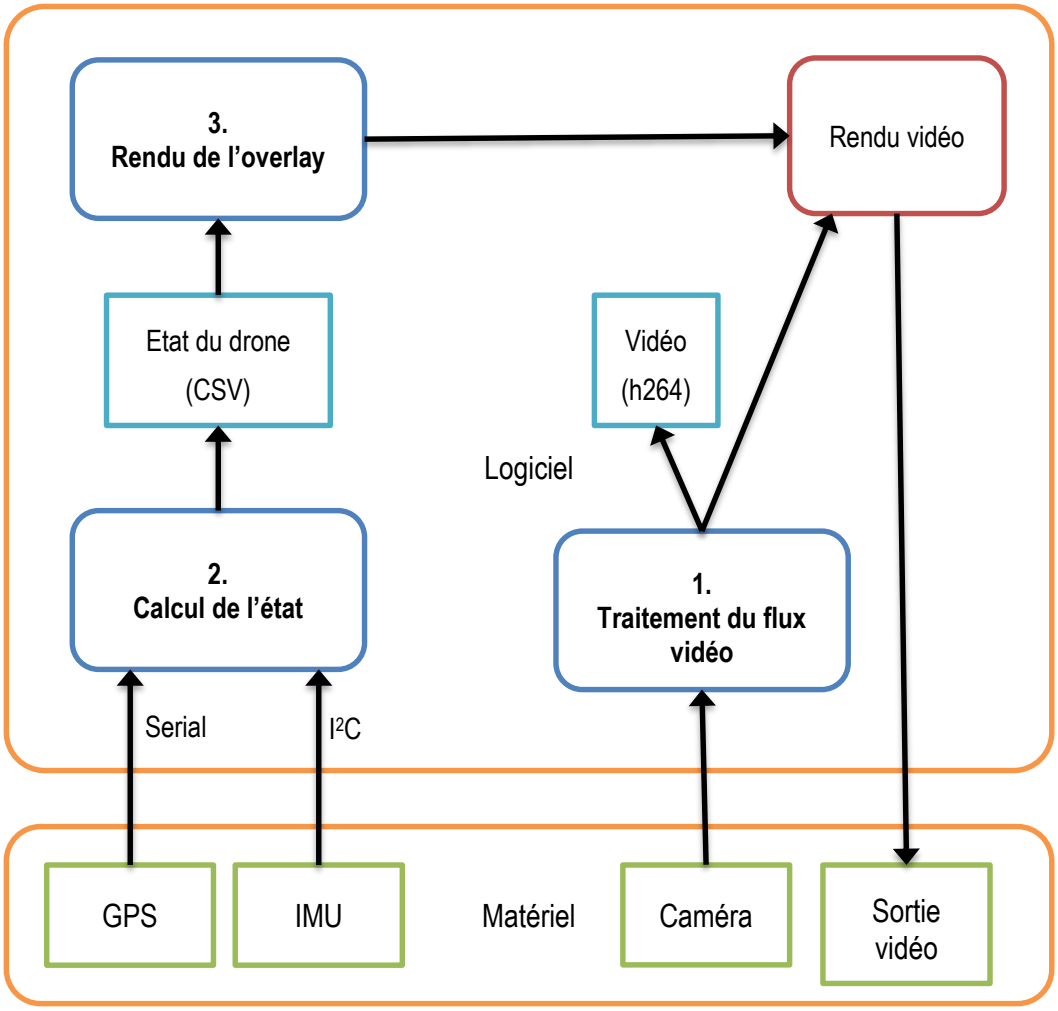
Les opérations que doit réaliser la plate-forme peuvent être séparées en trois tâches distinctes :

1. Enregistrement et diffusion de l'image de la caméra sur la sortie vidéo
2. Récupération des données des capteurs et calcul les informations manquantes
3. Calcul et rendu d'un calque (overlay) avec les informations de navigation et diffusion sur la sortie vidéo

Chacune de ces tâches à un objectif défini qui est indépendant de celui des deux autres tâches :

1. Informer (en direct ou en différé) l'utilisateur de ce que « voit » le drone
2. Déterminer l'état du drone (vitesse, position, …)
3. Mettre en forme les informations d'état pour qu'elles soient facilement lisibles et les transmettre au pilote

La combinaison de ces trois éléments nous donne le résultat demandé par le cahier des charges. La séparation de ces trois tâches en programmes distincts va permettre de cerner plus facilement leurs priorités et leurs facteurs critiques, en plus de faciliter la modularité et la maintenance. La mise en place d'une interface standardisée (fichier CSV) entre deux tâches permet de facilement remplacer un des programmes. On peut également choisir de manière indépendante pour chacun des programmes les meilleurs outils et le meilleur langage pour l'implémentation.



9 ENREGISTREMENT ET DIFFUSION DU FLUX VIDÉO

Le premier programme effectue la diffusion du flux de la caméra sur la sortie vidéo de la RPi. L'objectif principal est de simplement diffuser l'image en 1280x720 sur la sortie vidéo, à une fréquence de 25 images par seconde. Un second objectif est d'enregistrer le flux vidéo sur la mémoire flash de la RPi.

9.1 Format du flux vidéo

La caméra officielle de la RPi utilise un format de vidéo spécifique. Ce flux peut être directement interprété par les programmes et librairies officiels afin de diffuser l'image à l'écran ou de l'encoder pour sauvegarder la vidéo dans un fichier.

Pour utiliser le flux vidéo avec une autre application, par exemple avec le framework Qt, il est nécessaire d'utiliser une interface de lecture standardisée. La plus connue est le driver V4L[6] (vidéo for Linux) dont il existe depuis peu une version officielle pour la caméra de la RPi. Cependant, celui-ci ne fait état que de rares références et documentation.

9.2 Encodage de la vidéo pour l'enregistrement

L'encodage en h264[7] est supporté par le programme officiel. Cette norme, très répandue à l'heure actuelle permet d'encoder efficacement des vidéos et est supporté par notre GPU.

9.3 Facteurs critiques

- **Performances de capture** : le logiciel permettant l'enregistrement doit offrir des performances suffisantes pour enregistrer des images en HD (1280x720) à un débit minimal de 30 images par seconde.
- **Capacité d'encodage** : le logiciel doit permettre l'enregistrement d'une vidéo de la qualité citée sur le stockage de la RPi, et donc de compresser suffisamment la vidéo pour qu'elle prenne moins de place. Le stockage de la RPi se faisant sur la carte SD, dont on trouve aisément des capacités de 32Go à prix abordable, il faut qu'une heure de vidéo ne dépasse pas 10 Go. On peut ainsi imaginer stocker pas loin de 3 sessions de 1h de vidéo.
- **Temps de réponse** : le temps de traitement du flux vidéo doit être suffisamment court, ceci afin que l'utilisateur ne ressente pas de délai à l'utilisation.

9.4 Comparaison des outils

A) C

Raspbian fourni deux programme, raspivid et raspistill, permettant de prendre des vidéos, respectivement des photos à l'aide de la caméra. Les sources de ces programmes sont disponibles sur github.

Intégration : la librairie officielle est installée par défaut, et disponible dans un package. L'installation est simple et rapide.

[6] http://fr.wikipedia.org/wiki/Video4Linux
[7] http://fr.wikipedia.org/wiki/H.264

Encodage : une vidéo de full HD de 60 secondes (encodage H264, 10Mbits/s) utilise environ 120 Mo d'espace. La vidéo est donc volumineuse, mais on pourrait quand même en stocker plusieurs heures sur notre SD.

Performances : le programme raspivid est utilisé avec les paramètres suivant :

```
$ raspivid -p -f -e -t 0 -o video.h264 -w 1280 -h 720
```

Il s'agit d'une capture vidéo avec encodage en H264 avec une résolution HD, le contenu est affiché sur la sortie HDMI et enregistré sur la flash en même temps. Visuellement, le flux est parfaitement fluide, la résolution est bonne et il n'y a aucune saccade en direct comme sur l'enregistrement.

Voici l'utilisation CPU de la RPi pour cette capture. La mesure a été faite à l'aide du programme *sar* sur une durée de 30 secondes (une mesure par seconde, soit *sar 1 30*).

```
21:35:08  CPU %user %nice %system %iowait %steal %idle
Moyenne : all 2.42  0.00  10.05   7.53    0.00   80.01
```

La consommation de CPU est non négligeable, mais reste toutefois raisonnable. La mesure des autres opérations permettra de savoir si ce résultat est acceptable. Dans le cas contraire, l'enregistrement sur la mémoire pourrait être supprimé en cas de visualisation en direct et donc l'encodage également, l'émission du flux étant analogique.

Mesure sans enregistrement en mémoire :

```
$ raspivid -p -f -t 0 -w 1280 -h 720

21:35:08  CPU %user %nice %system %iowait %steal %idle
Moyenne : all 0.98  0.00  2.07    0.00    0.00   96.95
```

Sans l'enregistrement, on constate de très bonnes performances, ce qui est tout à fait logique, étant donné qu'il n'y a pas de calcul, seulement un traitement permettant de formater les données pour l'affichage. Au niveau des performances, cette solution est donc envisageable.

Utilisation : La librairie est complète, mais offre un faible niveau d'abstraction. Le programme raspivid, qui offre un nombre important de fonctionnalités (au niveau des paramètres), fait plus de 1500 lignes. On peut également imaginer l'utilisation de la librairie uniquement au travers du programme fourni et ses paramètres. Les possibilités sont alors restreintes à ce qui est proposé par le programme, mais devient très facile à utiliser.

Évaluation

- Intégration : 6
- Encodage : 5
- Performances de capture : 5
- Utilisation : 4

B) Python

Le module Picamera (Dave Hughes, BSD) est très complet et il y a un grand nombre d'exemples bien documentés. La documentation cite des performances de 30 ips à une résolution de 1024x768.

Intégration : le module est disponible sous forme de package dans les dépôts officiels sur Raspbian, l'installation se résume donc à :

```
$ sudo apt-get install python-picamera
```

C'est un binder sur la librairie native en C cité ci-dessous.

Encodage : le module supporte l'encodage en H264 avec une résolution de 1280x720. 37Mo pour une vidéo de 1 minute.

Performances : enregistrement et streaming de vidéo H264. Le test a été effectué de la même manière que pour la librairie C native.

```
21:35:08  CPU  %user %nice %system %iowait %steal  %idle
Moyenne : all  7.85  0.00  5.87    5.36    0.00    80.91
```

On a approximativement les mêmes performances qu'en C, ce qui est logique compte tenu du fait que c'est un binder, on s'attend seulement à un très léger overhead de Python.

Utilisation : le module est simple d'utilisation et permet une plus grande flexibilité. Par exemple, le code pour effectuer la capture précédente se résume en quelques lignes :

```
import picamera
with picamera.PiCamera() as camera:
    camera.resolution = (1280, 720)
    camera.framerate = 30
    camera.start_preview()
    camera.start_recording('video.h264')
    camera.wait_recording(120)
    camera.stop_recording()
    camera.stop_preview()
```

Le module permet également de faire de nombreuses autre choses. On ainsi la possibilité d'enregistrer une image toutes les n secondes pour faire un timelapse, séparer la vidéo en plusieurs fichiers d'une durée spécifique.

Évaluation

- Intégration : 6
- Encodage : 6
- Performances : 5
- Utilisation : 6

9.5 Choix

Le module Python offre les mêmes performances que la librairie native. Il n'y a donc pas de raison de se passer de la simplicité d'utilisation et des possibilités qu'offre le module picamera.

10 CALCUL ET ENREGISTREMENT DES DONNÉES DE NAVIGATION

10.1 Récupération des données des capteurs

A) IMU

L'IMU est connectée à la RPi par un bus I2C. Pour récupérer les données, il nous suffit d'interroger les registres des différents capteurs à travers ce bus I2C.

La fréquence de rafraichissement des capteurs de l'IMU est de l'ordre du kHz. Nous n'avons cependant pas besoin de données aussi fréquemment. Dans notre cas, ces données sont à destination de l'utilisateur. Certaines seront affichées numériquement alors que d'autres seront représentées graphiquement. Nous avons donc deux conditions :

- Les données doivent être mises à jour suffisamment lentement afin d'être perceptibles
- Les données doivent être mises à jour suffisamment rapidement afin que leur représentation graphique soit fluide

La première limitation sera gérée par l'affichage, celle-ci étant variable en fonction de la méthode de visualisation choisie pour l'information. La seconde nous impose une fréquence supérieure à 25 Hz[8]. Notre plate-forme ayant des capacités limitées, il est préférable de minimiser les opérations au strict minimum. Les informations seront donc relevées à 25Hz.

Les données fournies par les capteurs sont les suivantes :

Accélération sur 3 axes

Ces données nous proviennent de l'accéléromètre. Celles-ci doivent être séparées en deux composantes : l'accélération de mouvement linéaire et l'accélération normale. L'interprétation de l'accélération normale nous permet de connaitre l'orientation du capteur. Il est cependant nécessaire d'avoir d'autres informations afin de distinguer l'accélération de mouvement linéaire de l'accélération normale dans un cas ou le capteur peut être soumis à des variations de vitesse et d'inclinaison simultanément.

Rotation sur 3 axes

Cette information, fournie par les gyroscopes, nous permet de connaître les mouvements de l'IMU sur ces trois axes. Les gyroscopes sont cependant affectés de dérives cumulatives, importantes sur des durées plus longues. On peut donc les utiliser pour déterminer des rotations rapides, mais pas des positions sur le long terme.

Direction du champ magnétique sur 3 axes

La direction du champ magnétique nous donne une information sur la rotation du capteur sur l'axe Z. Associée à la rotation de x et y, elle permet de connaitre la rotation sur z, la direction du drone. Il convient d'ajouter une compensation, appelée « déviation du compas »[9]. De plus, si on veut indiquer le nord géographique, il faut également ajouter la déclinaison magnétique du lieu d'utilisation[10].

[8] Bien que la notion de fluidité soit très variable en fonction de l'application, il est couramment admis qu'une image est fluide dès 25 images par secondes. Ref.

[9] http://fr.wikipedia.org/wiki/Champ_magnétique_terrestre#La_boussole

[10] http://fr.wikipedia.org/wiki/Déclinaison_magnétique

Le champ magnétique est facilement perturbé par des facteurs extérieurs tels que les appareils électroniques ou certains métaux. Il est donc nécessaire de compenser les valeurs du magnétomètre avec les perturbations mesurées de l'environnement. Ceci est particulièrement valable dans notre cas, où le capteur est monté sur un drone.

Température et pression atmosphérique

La température n'as pas un très grand intérêt, mais peut toujours être affichée. La pression atmosphérique est utile pour connaitre la différence d'altitude par rapport à une autre mesure. Elle peut également donner l'altitude mais le calcul nécessite beaucoup de paramètres et donne un résultat peu précis.

B) GPS

Le GPS a une fréquence de rafraichissement paramétrable allant de 1 à 10 Hz. Les données apportées par le GPS sont des données absolues qui peuvent être très précises pour notre utilisation si la réception est bonne. Le problème vient de son faible taux de rafraichissement, et d'une imprécision qui croit avec le taux de rafraichissement. C'est donc un bon moyen d'avoir des valeurs de référence à moduler grâce à d'autres données.

10.2 Calcul des informations complémentaires

L'IMU nous fournit certaines données avec un haut taux de rafraîchissement, mais elles ne nous permettent pas de déterminer certaines informations de manière précise, telle que la vitesse[11] ou l'altitude. Le GPS nous fournit des données absolues précises, mais à un taux de rafraichissement inférieur à celui désiré par l'affichage. Afin de déterminer ces informations complémentaires précisément et avec un bon taux de rafraichissement, il faut combiner les données du GPS et de l'IMU afin de tirer avantage de chacun d'entre eux.

Cela se traduit par une série de calculs, quelques dérivées dans sa forme la plus simple, mais dont on peut améliorer la précision à l'aide d'outils plus spécifiques, tels que des filtres de Kalman.

10.3 Facteurs critiques

Le temps de réponse ne doit pas être trop élevé, sinon il entrainera d'une part un déphasage entre la vidéo et les informations affichées à l'écran et d'autre part une réduction du temps de réponse du pilote du drone.

10.4 Comparaison des langages et des outils

Les capteurs que nous utilisons ont déjà des librairies qui permettent de les paramétrer et de les lire. Il n'y a pas de raison de ne pas réutiliser ces travaux.

A) C et C++

Des exemples de drivers pour accéder aux différents capteurs ont été trouvés, mais ils sont généralement destiné à une utilisation sur Arduino et nécessiteraient d'être adaptés pour la RPi. Ces drivers ne permettent que la lecture des capteurs avec un traitement pour les dimensionner correctement (unités habituelles), il n'y a pas de calcul de position complémentaire.

[11] L'inefficacité de l'accéléromètre pour déterminer la vitesse est expliquée plus en détail dans la réalisation

B) Python

Une librairie Python, proposée par Bitify[12], permet de lire les trois capteurs de notre IMU. Elle implémente également le calcul de l'état du drone à partir de plusieurs capteurs pour des IMU similaires à la nôtre. Une adaptation pour le GY88 serait très facile. Au niveau des performances, un test a été effectué avec un démonstrateur de lecture et de visualisation de la position de l'IMU[13]. Celui-ci affiche une bonne réactivité avec une faible utilisation CPU.

10.5 Choix

Python nous offre une solution qui demande peu d'adaptations. Le langage est flexible et facile à appréhender pour une personne extérieure qui désire modifier le code. De nombreux modules, faciles à installer, permettent d'étendre le langage pour diverses tâches, comme l'utilisation d'outils mathématiques. Les outils en C/C++ seraient à créer entièrement, excepté la base de quelques pilotes.

Le test de performance en annexe permet de se rendre compte des piètre performances de Python par rapport à un langage compilé. Dans notre cas, cela ne devrait pas poser de problème, la démo montrant de bonnes performances avec une faible utilisation du CPU. Toutefois, on pourrait se retrouver limité au niveau des performances en cas d'ajout de fonctionnalités avancées qui ne sont pas prévues dans le cadre de ce travail.

Pour la lecture du GPS, on peut utiliser le très répandu gpsd[14], qui offre une librairie en C/C++ tout comme en Python. Il n'y a donc pas tant de préférences de ce côté.

Python répond à toutes nos attentes pour la récupération des données et, dans un souci d'unité pour ce programme, c'est le langage qui sera également utilisé pour la lecture du GPS.

[12] https://github.com/bitify/raspi

[13] http://blog.bitify.co.uk/2013/11/reading-data-from-mpu-6050-on-raspberry.html

[14] http://www.catb.org/gpsd/

11 AFFICHAGE DES INFORMATIONS DE NAVIGATION

11.1 Adaptation aux performances de la plate-forme

Comme cité précédemment, les performances du CPU sont trop faibles pour effectuer un rendu en HD de nos informations de navigation. Cependant, les performances du GPU sont parfaitement en adéquation avec cette tâche. Pour exploiter la capacité du GPU, les libraires OpenGL ES 2 sont disponibles sur la RPi.

Il faut toutefois garder à l'esprit que le GPU ne peut se charger que du rendu, et non du calcul du placement des éléments. Le CPU se charge donc de faire les calculs permettant de déterminer où doivent être dessinés les éléments (en calculant l'emplacement des sommets) et le GPU les dessine (en calculant l'emplacement des pixels nécessaires à relier les sommets). Dans notre cas, nous avons plusieurs centaines de triangles dont il faudra calculer l'emplacement, ce qui pourra utiliser intensivement le CPU selon la qualité de l'implémentation et le langage utilisé.

11.2 Possibilités d'affichage

Pour obtenir un affichage correspondant au cahier des charges, il faut pouvoir dessiner différents éléments, tels que des lignes, des rectangles ou du texte, et pouvoir déplacer et adapter ces éléments selon l'état du drone. OpenGL ne nous offre cependant que la possibilité de dessiner des triangles, ce qui permet effectivement d'obtenir l'ensemble des éléments cités, mais de manière fastidieuse. Le développement d'une librairie offre une abstraction permettant de dessiner tous les éléments nécessaire. Ce besoin n'étant pas extraordinaire, on peut trouver diverses librairies permettant de faire cela dans différents langages.

11.3 Facteurs critiques

Le point critique se trouve ici au niveau des performances. Il faut garantir un rendu fluide et sans délai malgré les faibles performances du CPU. Cela se traduira notamment par l'exploitation obligatoire du GPU pour le rendu.

Il ne faut pas non plus négliger la flexibilité de l'interface. Le but du projet est de créer un framework facile à utiliser et à modifier, dans un temps assez faible. Développer une librairie à partir de rien, uniquement en utilisant OpenGL semble difficilement réalisable durant le temps imparti.

11.4 Comparaison des langages et des outils

A) Pi3d

Ce module Python permet de créer des formes et du texte en utilisant OpenGL pour le rendu. Le module est rapidement mis en place et les démos sont fluides. Pour évaluer les performances, un prototype dessinant le bon nombre de formes et de textes est créé et permet de se rendre compte que le programme est trop gourmand. Un profilage a été fait pour d'identifier les opérations les plus lourdes, qui s'avèrent être au cœur du fonctionnement du module. Le détail de cette analyse est en Annexe B. Une modification du module pour corriger l'ensemble des défaillances n'est pas envisageable. Pi3d ne sera pas retenu.

Évaluation

- Utilisation : 6
- Performances : 3

B) C/C++

Le langage natif nous assure de meilleures performances, mais il n'existe pas de librairie de dessin 2D correspondant à nos besoins. Une évaluation des possibilités d'OpenGL est faite en créant un programme dessinant quelques formes. On se rend rapidement compte que l'implémentation est fastidieuse et sera difficile à faire évoluer. C'est une solution envisageable, mais uniquement si on ne trouve rien de mieux.

Évaluation

- Utilisation : 3
- Performances : 6

C) Qt5 et QML

Qt5 permet l'utilisation d'OpenGL ES 2 pour le rendu graphique. De plus, à l'aide de QML, on peut facilement décrire des interfaces graphiques très flexibles qui seront interprétées par Qt et envoyées à OpenGL pour le rendu. Grâce à cela, on associe à la fois un moyen simple pour définir des éléments graphiques et les positionner, avec QML, et un moteur de rendu exploitant les capacités du GPU, avec Qt.

QML est un langage de plus en plus répandu, émergeant dans le domaine mobile et web. Il permet de créer bien plus d'éléments que ce dont nous avons besoin, en offrant par exemple la possibilité d'ajouter des images ou du contenu audio. Cela laisse imaginer de bonnes perspectives d'évolution. Bien que QML soit encore méconnu de la plupart des développeurs, il est facile à appréhender et une modification d'un code existant est généralement triviale. Une évaluation des performances, disponible en Annexe C a montré d'excellents résultats.

L'inconvénient est qu'il nécessite la librairie Qt5, mais celle-ci est disponible en package sur raspbian. Autre inconvénient, il est nécessaire de cross-compiler le programme, ou d'installer les outils de compilation sur la RPi, mais cela ne concerne que la partie C++, le QML n'étant pas compilé. Dans notre cas, c'est dans le QML que se tient toute la partie que l'on désire généralement personnaliser.

Evaluation

- Utilisation : 5
- Performances : 6

11.5 Choix

De toute évidence, QML est le plus adapté à nos besoins, offrant à la fois une solution flexible avec un grand nombre de fonctionnalités d'un côté, tout en restant performant et facile à utiliser de l'autre.

Partie 2 – Réalisation

12 MISE EN PLACE DE L'ENVIRONNEMENT

12.1 Installation de Qt5

Qt5 est disponible sous forme de packages mis à disposition par un tiers. Pour les installer, il faut ajouter le dépôt[15] des packages :

```
$ deb http://twolife.be/raspbian/ wheezy main backports
$ deb-src http://twolife.be/raspbian/ wheezy main backports

$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-key
2578B775
```

Il faut ensuite installer les packages liés à Qt5 avec :

```
$ sudo apt-get install libqt5core5a libqt5opengl5 libqt5quick5
libqt5qml5 libgles2-mesa libegl1-mesa
```

12.2 Installation de gpsd

Gpsd est disponible sous forme de packages également, dans les dépôts officiels :

```
$ sudo apt-get install gpds python-gps
```

Pour utiliser le module Python gpsd, il faut exécuter le daemon gpds en écoute sur notre port série relié au GPS :

```
sudo gpsd /dev/ttyAMA0 -F /var/run/gpsd.sock
```

Pour tester le GPS, on peut utiliser gpsmon qui va se connecter à gpsd et afficher les informations de géolocalisation :

```
$ gpsmon
```

Le module python nous permet également de récupérer les données du GPS en se connectant sur gpsd.

12.3 Accès à l'IMU

L'accès à l'IMU se fait en i2c. Il est pour cela nécessaire d'avoir les drivers i2c, qui ne sont pas chargés par défaut. Pour changer cela, modifier le fichier suivant et commenter la ligne qui blacklist l'i2c :

```
$ sudo vi /etc/modprobe.d/raspi-blacklist.conf

#blacklist spi-bcm2708
#blacklist i2c-bcm2708
```

[15] http://twolife.be/raspbian/

Redémarrer la RPi pour charger le driver. Nous accédons alors au bus à l'aide du programme i2c-tools, qui peut être installé avec :

```
$ sudo apt-get install i2c-tools
```

On peut tester la présence des capteurs sur le bus à l'aide de :

```
$ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- 1e --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- UU -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- 77
```

On peut tester la lecture du MPU en lui demandant de nous renvoyer sa propre adresse :

```
$ sudo i2cget -y 1 0x68 0x75
0x68
```

13 DIFFUSION DU FLUX VIDÉO

L'enregistrement de la vidéo s'effectue à l'aide du module Python picamera. Un problème a cependant été rencontré pour l'envoi de l'image sur la sortie vidéo. Deux flux doivent être envoyés : la vidéo et le calque des informations de navigation. Bien entendu, le calque de navigation doit être rendu au-dessus de la vidéo pour qu'il soit visible.

C'est le *display manager* qui va se charger de composer l'image finale à partir des deux flux. Pour connaitre l'ordre des couches, celui-ci se réfère aux indications données par le programme qui envoie le flux. Cette information est généralement définie statiquement dans les programmes. Dans Qt, elle se situe dans : *qtbase/mkspecs/devices/linux-rasp-pi-g++/qeglfshooks_pi.cpp* :

```
static DISPMANX_DISPLAY_HANDLE_T dispman_display = 3;
```

Le problème est qu'il est alors nécessaire de recompiler Qt pour que la modification soit effective, ce que nous aimerions éviter, afin de rester compatibles avec les mises à jour des packages.

Dans Python, il n'a pas été possible d'identifier où est défini le niveau, mais nous serions de toute façon confrontés au même problème : recompiler Python. Bien heureusement, une mise à jour du module Picamera est venue apporter une solution à ce problème sous forme de paramètre qui permet d'indiquer le display layer voulu.

```
self.camera.preview_layer = 0
```

Un fichier de configuration permet de paramétrer le comportement de la caméra :

```
[camera]
rotation = 0
resolution_x = 1280
resolution_y = 720
preview = True
record = False
```

Une alternative qui a été utilisée est de recompiler raspivid en modifiant le layer dans :

```
userland/host_applications/linux/apps/raspicam/RaspiPreview.h
```

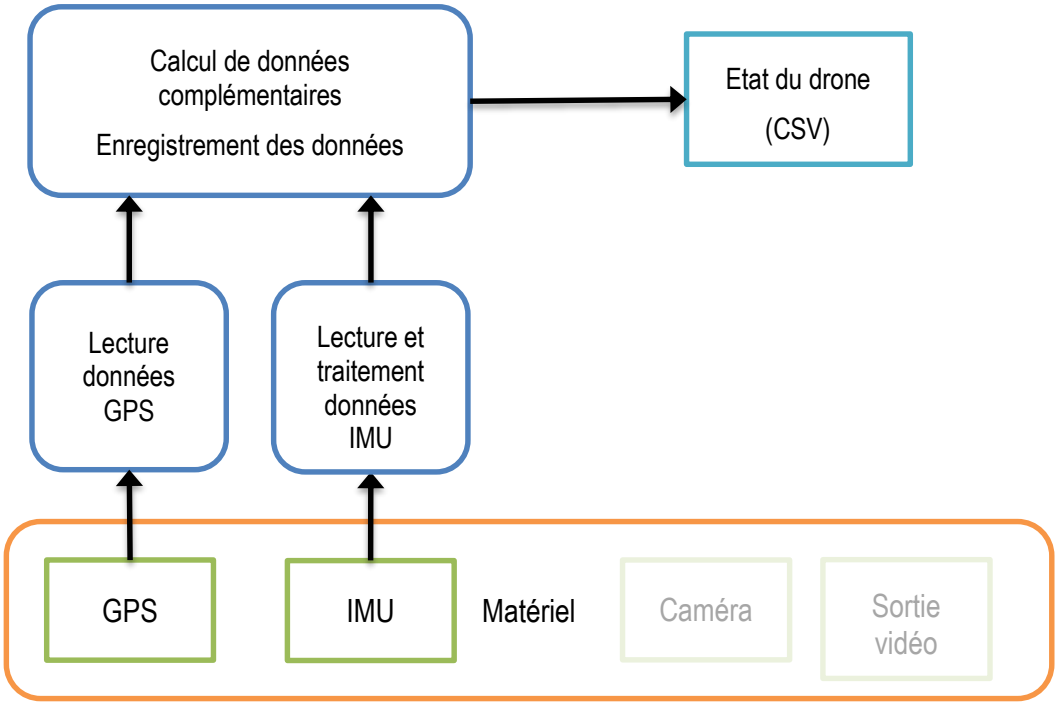
14 CALCUL ET ENREGISTREMENT DES DONNÉES DE NAVIGATION

Cette étape s'effectue à l'aide d'un programme en Python basé sur la librairie d'accès au GY80 et au MPU6050 développée par Bitify[16], qui implémente une partie des fonctionnalités décrites ci-dessous.

Le programme est séparé en plusieurs tâches :

- Lecture des données de l'IMU
- Lecture des données du GPS
- Calcul des données de navigation

Les lectures de données sont effectuées par des threads dédiés qui indiquent également la présence de nouvelles celles-ci. Le thread principal se charge d'écrire le fichier CSV après avoir calculé les données complémentaires.



14.1 Lecture et traitement des données de l'IMU

L'IMU (GY88) est composée de trois composants reliés sur le même bus I2C :

- MPU-6050 (6-Axis Angular Rate Sensor & Accelerometer)
- HMC5883L (3-Axis Digital Compass)
- BMP085 (Barometric Pressure Sensor)

Chacun de ces capteurs a une classe Python permettant l'accès sur le bus pour récupérer les données avec un premier traitement. La lecture s'effectue à une fréquence paramétrable définie par défaut à 25 Hz.

[16] https://github.com/bitify/raspi

A) MPU-6050 – accéléromètre et gyroscope

Roulis, tangage et lacet : ces trois notions déterminent l'état de position du drone, c'est-à-dire l'angle de rotation sur chacun des trois axes. Nous utiliserons ici également les termes en anglais : yaw (lacet), pitch (tangage), roll (roulis).

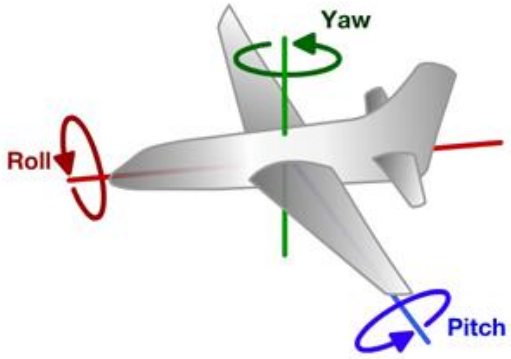


Image 2[17]

Ces informations sont élémentaires dans le domaine de l'aéronautique. Pour les déterminer, il faut utiliser une combinaison des données de nos capteurs.

Le MPU nous donne des informations d'accélération et de rotation. Ceci nous permet de connaitre, lors d'un mouvement linéaire, l'orientation du drone sur pitch et roll. Lors d'un mouvement non-linéaire, avec des accélérations, cette information est modifiée, car en plus de l'accélération terrestre, on retrouve l'accélération du mouvement. Un choc sur le drone, qui générera des pics d'accélération très importants, ou simplement le bruit pourront alors modifier l'orientation du drone, ce qui n'est pas souhaitable.

Pour éviter cela, les informations d'accélération ne sont utilisées que pour le long terme. Les informations de rotation rapides sont tirées des gyroscopes, plus fiables sur de courtes durées. Ceux-ci étant sujets à des dérives dans le temps, ils ne sont pas utilisés sur le long terme. Cette opération est effectuée par un filtre, représenté par La Figure 1 ci-dessous. Cette solution est implémentée dans la librairie Bitify et semble tout à fait adaptée pour notre utilisation du capteur.

Les capteurs peuvent parfois renvoyer des valeurs erronées, due à l'imperfection du matériel. On remarque qu'une mauvaise valeur unique renvoyée par les accéléromètres aura un impact négligeable sur le résultat. Cependant, une valeur erronée de gyroscope, même unique, modifiera très rapidement le résultat, ce qui peut s'avérer problématique, la correction apportée par les accéléromètres étant très lente.

Des valeurs erronées de gyroscope ont été détectées à plusieurs reprises dans notre cas, il était donc indispensable de filter ces valeurs afin de garder des données de sortie correctes. La Figure 2, représentant les valeurs du gyroscope, permet de se rendre compte de la grandeur des valeurs erronées. Entre 100 et 400, une rotation lente sur chaque axe a été faite successivement et entre 600 et 900 une rotation rapide[18]. On remarque alors que les « glitchs » sortent fortement du lot. Alors que les valeurs d'un mouvement même rapide ne dépassent pas les 10°/ms, les valeurs erronées naviguent entre 15 et 35°/ms. En pratique, il est très difficile pour un drone d'atteindre une telle vitesse de rotation par lui-même, les données supérieures à 15° seront donc automatiquement ignorées.

[17] Source : http://www.touringmachine.com/

[18] La mesure initiale est faite sur plus de 10'000 mesures, mais les parties inintéressantes ont été tronquées

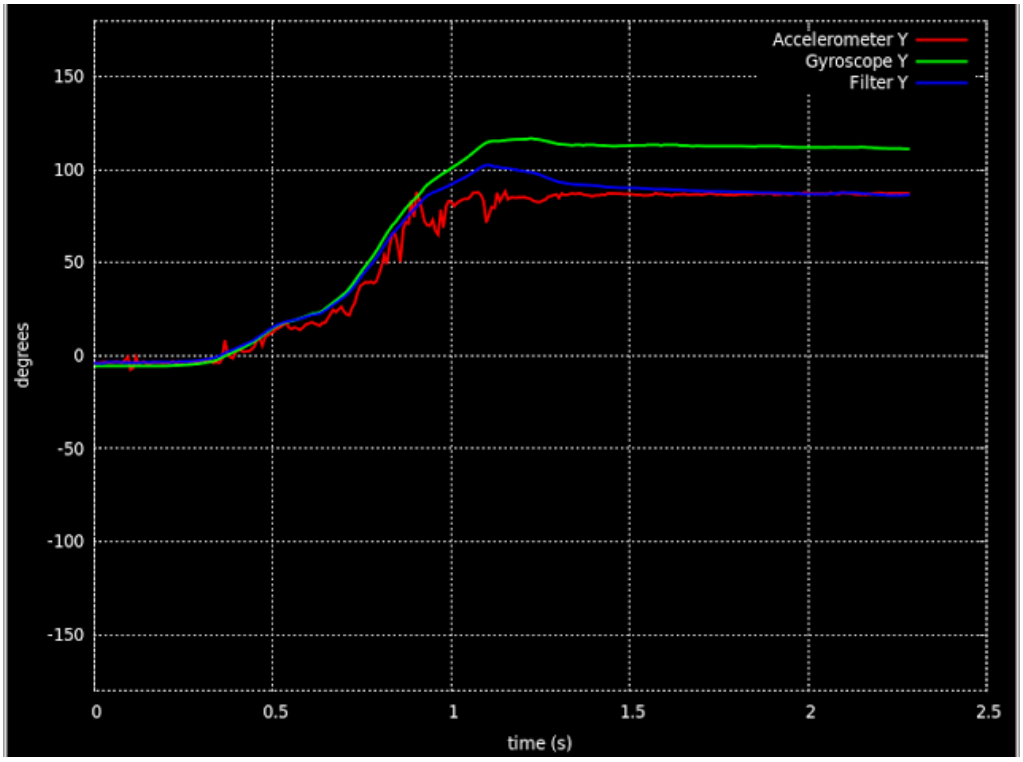


Figure 1[19] – effet du filtre compensatoire sur l'axe y lors d'une rotation de 90°

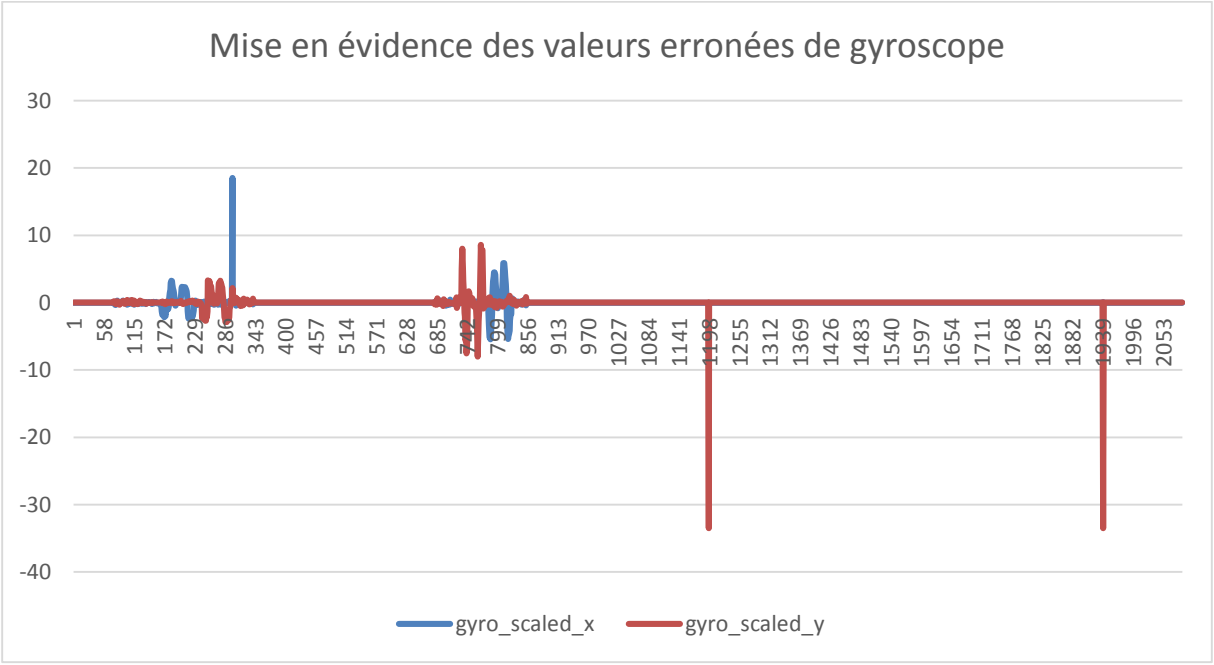


Figure 2 – mise en évidence des valeurs erronées de gyroscope

Blocage de cadran

A cause de l'utilisation d'angles d'Euler (yaw, pitch, roll), on a un blocage de cadran lorsque pitch et yaw font référence à la même information (soit quand on est à 90°). On perd à ce moment un degré de liberté (et donc

[19] Source : http://blog.bitify.co.uk/2013/11/using-complementary-filter-to-combine.html

d'information) qui nous empêche de distinguer pitch de roll. Concrètement, à partir du moment où on veut corriger un des deux, la correction va s'appliquer à l'autre également.

Pour éviter de blocage, il faut remplacer cette représentation des rotations par une représentation par des quaternions. Cela demande la refonte de toutes les parties qui entrent en compte dans le calcul de position.

B) HMC5883L - boussole

Calcul de la rotation sur Z

La boussole nous renseigne sur la rotation autour de l'axe Z, dans le cas où le drone est à plat, cette rotation est la même que celle de l'axe z du drone. Dans le cas où le drone n'est pas à plat, on peut déterminer la rotation sur l'axe z à l'aide des informations de rotation sur les axes x et y et Z. Ceci est déjà fait par la librairie Bitify, il n'y a donc rien à ajouter.

Correction des perturbations magnétiques

Un script de calibrage, basé sur celui de la librairie de Bitify, a été ajouté afin de calculer les offsets à appliquer pour corriger le décalage dû à l'environnement. Le principe de cette calibration et de mesurer la force du champ magnétique sur tous les axes. Le champ magnétique étant fixe, les valeurs devraient être les mêmes dans chaque direction, ce qui équivaut à dire que les valeurs max de chaque axe sont identiques. La Figure 3 montre l'influence de l'environnement sur le champ magnétique (sur deux axes), et le décalage apparent du centre sur l'axe y (d'environ 150).

Cette mesure nous permet alors de calculer le décalage sur chaque axe. Ces offset sont ensuite stockés dans un fichier de configuration qui est lu par l'enregistreur qui appliquera ces offsets aux valeurs brutes récoltées.

Ce script doit être lancé manuellement par l'utilisateur à chaque fois que le capteur change d'environnement de montage. Bien entendu, seules les perturbations du support de montage (et tout ce qui est fixé dessus) peuvent être compensées, l'influence d'éléments fixes de l'environnement, telle qu'une ligne à haute tension, ne peut pas être corrigé. Dans le cas où le fichier avec les valeurs de corrections à apporter n'est pas trouvé, l'utilisateur en est informé par la console.

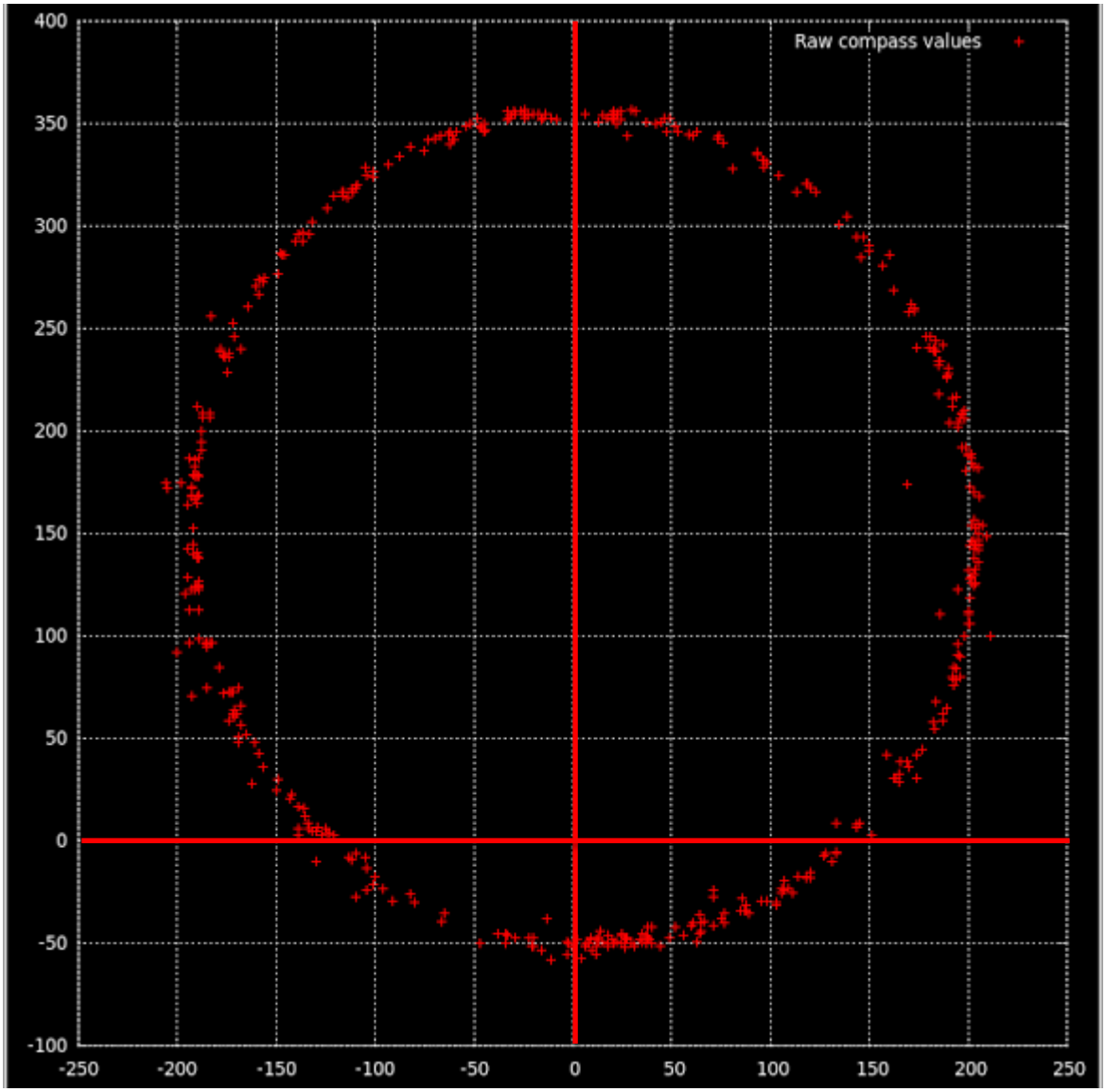


Figure 3[20] – mesure du champ magnétique sur deux axes, sans correction

Prise en compte de la déclivité magnétique

La déclivité magnétique est ajoutée au paramètre d'offset du nord (noth_offset) dans le fichier de configuration. Il faut additionner la déclivité magnétique, la déviation du compas et la différence entre l'orientation du capteur et l'avant du drone et la saisir comme paramètre.

C) BMP085 – baromètre et thermomètre

La librairie de Bitify se charge de lire les données et de calculer la pression atmosphérique et la température à partir de ces dernières, il n'y a donc rien de plus à faire à ce stade.

[20] Source : http://blog.bitify.co.uk/2013/11/connecting-and-calibrating-hmc5883l.html

14.2 Lecture des données GPS

Rappelons-le, le GPS est connecté par une paire unique en série. L'accès au GPS se fait à travers le driver série sur /dev/ttyAMA0. Le GPS envoie des données au format NMEA. La librairie gpsd permet de lire les trames NMEA pour en extraire les informations plus facilement. Gpsd contient également un module python.

A) Lecture des données

Les données sont récupérées par un thread qui vérifie qu'elles ne soient pas nulles. Une méthode permet de demander au thread si de nouvelles données sont disponibles. Après une lecture, le thread se rendort jusqu'à la prochaine lecture, une seconde plus tard. Selon la documentation du GPS, la fréquence de rafraichissement de ce dernier peut être paramétrée entre 1 et 10 Hz, mais malgré plusieurs tentatives, il n'a pas été possible de modifier ces paramètres. Les commandes NMEA envoyées au contrôleur GPS sont ignorées, ou reçoivent comme seule réponse « Device type has no rate switcher ». Il parait cependant improbable que le contrôleur ne supporte pas cette opération, la spécification et de nombreux témoignages faisant état du contraire. Il n'a pas pu être déterminé à quoi est du cette impossibilité.

14.3 Fusion des données

Les données récupérées de ces deux capteurs sont mises en commun afin de déterminer plus précisément l'état du drone. Les valeurs que nous cherchons à déterminer sont :

- Date et heure de la mesure
- Pitch, roll, yaw
- Latitude / longitude
- Vitesse
- Altitude

Pour la date et l'heure, c'est celle de la RPi qui est utilisée au moment de la mesure. Raw, pitch et yaw ont déjà été déterminés alors que la latitude/longitude provient directement du GPS.

Pour les autres valeurs, il faut les déterminer de manière composée, car on aimerait à la fois une grande précision et une grande fréquence de rafraichissement.

A) Vitesse

Afin d'obtenir une mesure précise qui ne dérive pas, la vitesse du GPS est utilisée comme référence, modulée par l'accélération effective durant la seconde qui la sépare de la prochaine mesure GPS. De cette manière, on peut espérer avoir une valeur avec un bon taux de rafraichissement sans trop de dérive cumulative.

La vitesse peut être déterminée en dérivant l'accélération sur l'axe y (avant-arrière). Toutefois, l'accéléromètre ne mesure pas la seule accélération due au changement de vitesse, mais également l'accélération normale G. Celle-ci est nulle sur l'axe y lorsque le drone est à plat, mais change dès qu'on incline le drone sur x. Nous avons donc une addition de la composante d'accélération due à la vélocité et d'une fraction de la composante d'accélération normale.

La composante d'accélération normale peut être déterminée à l'aide de l'inclinaison sur l'axe x, soit le « pitch ». Avec un angle de 0° avec l'horizontale, la composante de l'accélération normale est nulle, et avec un angle de 90°, elle est maximale. L'accélération normale (9,81m/s^2) étant plutôt élevée par rapport aux accélérations de vitesse (<5m/s^2), une petite erreur d'angle de pitch aura de fortes conséquences sur la vitesse déterminée, ce qui rend cette technique assez peu précise.

B) Altitude

Nous avons également plusieurs manières de déterminer l'altitude. La première est en fonction de la pression atmosphérique, qui varie en fonction de l'altitude. Mais elle est également sujette aux variations de températures et de conditions météorologiques. La pression atmosphérique donne donc une mauvaise indiction absolue (à moins d'avoir une station météo avec soi), mais une bonne indication relative : on peut rapidement faire varier la pression en changeant d'altitude dans la seconde, alors que les conditions météorologiques changent de manière insignifiante durant ce temps.

La seconde est la valeur mesurée par le GPS, avec une grande précision absolue, mais une fréquence de rafraichissement faible.

A nouveau, nous pouvons combiner ces deux mesures afin d'en déduire l'altitude avec une bonne précision absolue et un bon taux de rafraichissement. Pour cela, la pression est mémorisée à chaque nouvelle donnée d'altitude du GPS, qui sert de référence. Jusqu'à la prochaine mesure GPS, l'altitude est modulée en fonction de la différence de pression entre la dernière mesure et la pression de référence. La Table 1 nous donne les variations de pression en fonction de l'altitude pour différentes altitudes et températures. Notre modulation d'altitude n'a pas besoin d'être extrêmement précise, car elle a une nouvelle référence chaque seconde, on admettra donc une variation fixe de 8,7 m/hPa.

z	−15 °C	0 °C	15 °C	30 °C
0 m	7,5	7,9	8,3	8,8
500 m	7,9	8,3	8,7	9,2
1000 m	8,3	8,7	9,2	9,6
2000 m	9,3	9,7	10,1	10,6
3000 m	10,4	10,8	11,2	11,6

Table 1[21] - Echelon de nivellement barométrique [m/hPa]

[21] Source : http://fr.wikipedia.org/wiki/Variation_de_la_pression_atmosphérique_avec_l'altitude

15 AFFICHAGE DES INFORMATIONS DE NAVIGATION

15.1 Décomposition du programme

L'affichage des éléments de navigation est assuré par un programme en Qt5, à l'aide de QML. Nous avons 3 tâches à assurer :

1. Définir le comportement visuel à l'aide de composants graphiques : « Quelles informations voulons-nous afficher et de quelle manière ? »
2. Lire l'état du drone depuis le fichier XML
3. Mettre à jour les composants graphiques à l'aide des données de l'état du drone

Le comportement visuel est régit par une description QML. Celle-ci intègre le point 3 sous forme d'un timer cyclique, se déclenchant toutes les 4ms (25Hz) qui va déclencher la lecture du fichier CSV et mettre à jour la valeur et l'aspect des composants QML. La lecture du fichier CSV s'effectue à l'aide d'une classe C++ utilisée depuis QML.

15.2 Lecture des données

Une classe FileIO permet de lire dans le fichier CSV. Cette classe est mise à disposition de QML qui en instancie un objet et définit le fichier CSV à lire. Plusieurs modes de lecture ont été implémentés.

A) Lecture en flux tendu

La classe FileIO va lire la dernière ligne du fichier CSV. Le fichier pouvant avoir de nombreuses lignes, le pointeur est placé à la fin du fichier est remonte de deux lignes. On lit alors la ligne au pointeur : l'avant dernière ligne. On ne souhaite pas lire la dernière ligne car, le fichier étant en train d'être écrit, elle pourrait être incomplète. Avec cette méthode, on garde un temps de lecture indépendant de la taille du fichier.

Dans une lecture en direct de ce type, il est important que l'écriture du fichier CSV par l'enregistreur soit immédiate. En général, il faudrait donc faire un « flush » pour vider le buffer après chaque écriture.

B) Lecture différée

Il y a également la possibilité de faire une lecture différée afin de visualiser une navigation passée. Dans ce cas, le fichier CSV est lu ligne par ligne depuis le début. Afin de rejouer le vol à vitesse réelle, le temps de la première donnée est mémorisé et un timer compte le temps « réel ». On ne passe à la lecture de la prochaine donnée que si elle est dans le « passé ». De cette manière, on est indépendant de la fréquence d'enregistrement et des éventuels « trous » de données.

15.3 Composantes de l'affichage

A) L'horizon

L'horizon artificiel permet de représenter la ligne d'horizon et de connaître le roulis du drone. C'est particulièrement utile si le référentiel visuel disparait (visibilité, dévers, champ de vision, …). L'horizon est représenté par un simple trait (un rectangle en QML) auquel on applique une rotation inverse au « roll » afin qu'il reste toujours aligné avec l'horizontale de la vidéo.

L'horizon permet également de savoir si l'avion pointe vers le haut ou vers le bas : le « pitch ». Pour ce faire, un offset vertical (inverse au pitch) y est appliqué. Cependant, nous n'avons pas une vue sphérique sur 360°, on ne peut donc pas représenter l'horizon si on le perd de vue. Or, c'est justement là qu'il serait utile de l'avoir, on applique donc une limite haute et basse sous forme d'arctangente de pitch. Utiliser une arctangente permet de se rendre

compte d'un mouvement dans tous les cas. Avec un « plafond », une variation dans les valeurs supérieures à celui-ci ne serait pas visible.

B) Les règles

Afficher un paramètre sous forme numérique permet difficilement de se rendre compte des variations de ce dernier. Une technique fréquemment utilisée est celle de la règle graduée. La règle se déplace lorsque la valeur change, ce qui permet de facilement voir les variations.

Un modèle de règles a été implémenté dans le QML avec divers paramètres pour spécifier le pas de graduation, l'étendue, remplacer les valeurs par des labels, inverser la direction ou encore le rendre cyclique. On représente ainsi la vitesse, l'altitude et la direction grâce à cet unique modèle de règle.

C) Affichage de la direction du point de départ

Les coordonnées du point de départ sont mémorisées au démarrage de l'application d'affichage. Durant le vol, on calcule la direction de la cordonnée de départ par rapport à la position actuelle et, grâce à la boussole, on indique la direction du point de départ par rapport à notre direction. Comme nous n'avons pas une vue à 360°, la direction est représentée à plat, vue du dessus, on peut ainsi toujours connaitre la direction de retour sans avoir à la chercher en tournant en rond avec le drone.

D) Information complémentaires

Les autres informations, non essentielles au pilotage, sont affichées sous forme de texte.

Partie 3 - Discussion

16 PISTES EXPÉRIMENTÉES

16.1 MPU5060 en master I2C

La documentation du MPU spécifie qu'il peut travailler en I2C master sur le bus pour interroger le magnétomètre afin de déterminer son état sur les trois axes. De cette manière, le MPU peut utiliser des algorithmes de reconnaissance de geste. Malheureusement, seules les données de reconnaissance de geste sont accessibles, les données d'état du capteur ne peuvent pas être récupérées. Toutefois, si cela avait été possible, on aurait pu utiliser cette puissance de calcul et lieu de notre implémentation en Python.

16.2 OpenVG

OpenVG est une librairie pour le dessin vectoriel dont le rendu se fait par le GPU. Un wrapper, mis à disposition par Anthony Starks[22], permet de créer directement des formes diverses qui correspondent à nos besoins. Le traitement vectoriel induit cependant une plus grande utilisation des ressources. Le test de performance suivant démontre les trop faibles performances de cet outil.

- Le rendu de 100 triangles sur une image full HD s'effectue à 27 ips.
- Le rendu de 200 triangles s'effectue à 13 ips.
- Le rendu de 1000 triangles s'effectue à 2 ips.
- Le rendu de 1000 mots indépendants s'effectue à 60 ips.

Notre OSD contient environ une centaine d'éléments, nous sommes donc aux limites des performances d'OpenVG sur notre plate-forme.

[22] https://github.com/ajstarks/openvg

17 AMÉLIORATIONS FUTURES

17.1 Calcul de vitesse et filtre de Kalman

Nous avons constaté que notre calcul de vitesse était sujet à une imprécision cumulative alors que le GPS fait des « sauts » du à son imprécision et à sa mesure à faible fréquence.

Une méthode généralement utilisée dans ce genre de cas et le filtre de Kalman. Alors que nous étions indépendant des lois de la physique et des possibilités de notre drone, le filtre de Kalman se base sur un modèle de contraintes liées au drone. Ainsi, l'accélération n'a plus besoin d'être dérivée et l'accumulation d'erreur disparait, alors que le GPS voit ses valeurs réinterprétées en fonction de ce qui est réellement le cas, à l'aide des données d'accélération linéaire.

18 CONCLUSION

Flexibilité et fonctionnalités ne sont pas nécessairement opposées à de bonnes performances. Le problème est généralement que ces outils prennent du temps à développer, c'est pourquoi il peut être bénéfique de passer du temps à trouver des outils existants. Dans notre cas, la recherche fut fructueuse car Qt et QML se sont avérés particulièrement efficaces dans la réalisation de l'OSD.

Les difficultés pour connaitre la position d'un appareil sont bien plus nombreuses qu'on ne peut l'imaginer au départ. Notre perception des notions d'accélération, de rotation ou de vitesse nous est innée, ce qui donne une illusion de simplicité. En réalité, chacun des capteurs comporte son lot de problèmes, mais heureusement chaque capteur à des problèmes différents ce qui permet au final de retrouver les informations utiles, parfois au prix de la précision. Ces difficultés nous ont amenés à faire un mauvais choix stratégique au départ : celui de l'utilisation d'angles d'Euler « yaw, pitch et roll » pour déterminer la position. En dehors de toute attente, ils n'ont pas permis de mesurer les données correctement dans le cas où le drone fait un looping, ce qui ne parait instinctivement pas être un cas particulier.

Au final, nous obtenons une solution qui répond au cahier des charges pour une utilisation standard. Il n'a malheureusement pas été possible de s'étendre plus en détails dans toutes les parties de l'implémentation avant le délai imparti. Néanmoins, cette solution pause de solides bases et peut facilement être personnalisée et améliorée afin de convenir à des utilisations ayant des besoins divers.

Lausanne le 6 juin 2014

Christian MULLER

19 RÉFÉRENCES

A) RPi

Wiki non officiel : http://elinux.org/RPi_Hub

B) Camera

Installation de la camera sur la RPi : http://www.raspberrypi.org/help/camera-module-setup/

Documentation du module Python Picamera : http://picamera.readthedocs.org/

C) GPS

Le GPS Adafruit sur la Raspberry :

- https://learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps-on-the-raspberry-pi.pdf

D) IMU

CH Robotics : Using Accelerometers to Estimate Position and Velocity :

- http://www.chrobotics.com/library/accel-position-velocity

Librairie python pour la lecture de l'IMU, avec explication de l'implémentation : http://blog.bitify.co.uk/

Vidéo de présentation de Google concernant l'utilisation des capteurs d'une IMU : http://youtu.be/C7JQ7Rpwn2k

Partie 4 - Annexes

ANNEXE A – PERFORMANCES DES LANGAGES SUR RPI

Un programme calculant les 10000 premiers nombres premiers a été créé dans chacun des langages afin d'en évaluer les performances. Le temps d'exécution est évalué avec le programme *time*.

19.2 C

A) Code

```
int result = 0;

int main() {
    result |= 2;
    int i;
    for (i=3;i<10001;i++) {
        int prime = 1;
        int j;
        for (j=2;j<i;j++) {
            int k = i/j;
            int l = k*j;
            if (l==i) prime = 0;
        }
        if (prime) result |= i;
    }
    printf("%i\n", result);
    return 0;
}
```

B) Compilateur

- gcc version 4.6.3 (Debian 4.6.3-14+rpi1)
- Target: arm-linux-gnueabihf

C) Résultat

```
16383

real 0m5.475s
user 0m5.260s
sys 0m0.030s
```

19.3 C++

A) Code

```
int result = 0;

int main() {
    result |= 2;
    int i;
    for (i=3;i<10001;i++) {
        int prime = 1;
        int j;
        for (j=2;j<i;j++) {
            int k = i/j;
            int l = k*j;
            if (l==i) prime = 0;
        }
        if (prime) result |= i;
    }
    printf("%i\n", result);
    return 0;
}
```

B) Compilateur

- g++ (Debian 4.6.3-14+rpi1) 4.6.3

C) Résultat

```
16383

real 0m6.516s
user 0m6.370s
sys 0m0.020s
```

19.4 Python 2.7.3

A) Code

```
result = 0
result |= 2
for i in range(3,10001):
    prime = 1
    for j in range(2, i):
        k = i/j
        l = k*j
        if l == i:
            prime = 0
    if prime == 1:
        result |= i

print result
```

B) Résultat

```
16383

real 8m14.478s
user 7m59.510s
sys 0m2.470s
```

19.5 Python 3.2.3

A) Code

```
result = 0
result |= 2
for i in range(3,10001):
    prime = 1
    for j in range(2, i):
        k = i/j
        l = k*j
        if l == i:
            prime = 0
    if prime == 1:
        result |= i

print result
```

B) Résultat

```
16383

real 10m35.079s
user 10m12.670s
sys 0m3.220s
```

ANNEXE B – PROFILAGE DE PI3D

Le profilage du programme Python est effectué à l'aide de cProfile.

```
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    20351   14.291    0.001   14.291    0.001 {numpy.core.multiarray.array}
     2031    3.503    0.002   12.411    0.006 Shape.py:18(  init  )
     2026    3.194    0.002    3.564    0.002 Buffer.py:146( load opengl)
     2356    1.947    0.001    9.524    0.004 Shape.py:128(draw)
     2356    1.805    0.001    6.334    0.003 Buffer.py:212(draw)
     2031    1.391    0.001   10.303    0.005 Buffer.py:22(__init__)
     2031    1.188    0.001    1.197    0.001 Ctypes.py:19(c_floats)
       64    0.719    0.011   23.888    0.373 osd.py:137(set_value)
      504    0.638    0.001    0.638    0.001 {method 'render' of 'Font'
objects}
       30    0.626    0.021    0.699    0.023 Display.py:251(_tidy)
    12337    0.607    0.000    0.607    0.000 {numpy.core._dotblas.dot}
     2031    0.581    0.000    0.581    0.000
{numpy.core.multiarray.concatenate}
     2031    0.518    0.000    0.526    0.000 Ctypes.py:27(c_shorts)
```

Grâce à ce profilage, on remarque que l'essentiel du temps est passé dans ces quelques fonctions :

- numpy array : on accède aux tableaux de numpy lors de l'initilisation des shapes
- init : constructeur des fonctions, il y a effectivement beaucoup d'objets créés à chaque rendu
- Buffer : chargement des textures à l'init

L'init représente un temps total de 20 environ, sur un total de 41. Réduire le nombre d'objets créés peut donc booster les performances.

Du côté de l'utilisation, on peut optimiser le traitement en déplaçant les lignes des règles au lieu de les recréer, cela nous supprimera déjà la création de nombreux objets. L'autre point où l'on peut influencer, c'est la recréation des String lors de la modification du texte. Pour ce faire, il faudrait modifier la librairie Python en lui ajoutant les méthodes de modification. Pour simuler ces améliorations afin d'en mesurer les bénéfices, les valeurs des textes et règles ne sont pas modifiées, les objets sont ainsi redessinés, mais pas recréés à chaque itération.

```
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     2356    1.602    0.001    2.461    0.001 Buffer.py:212(draw)
     2356    1.000    0.000    3.783    0.002 Shape.py:128(draw)
      504    0.628    0.001    0.628    0.001 {method 'render' of 'Font'
objects}
      801    0.523    0.001    0.523    0.001 {numpy.core.multiarray.array}
     1011    0.392    0.000    0.392    0.000 {method 'getsize' of 'Font'
objects}
     2432    0.242    0.000    0.242    0.000 Buffer.py:169(_select)
      504    0.224    0.000    0.224    0.000 {method 'draw_bitmap' of
'ImagingDraw' objects}
        1    0.221    0.221    1.793    1.793 Font.py:30(  init  )
     9424    0.206    0.000    0.563    0.000 Loadable.py:37(load_opengl)
       76    0.184    0.002    0.199    0.003 Buffer.py:146(_load_opengl)
     2358    0.162    0.000    0.162    0.000 Shader.py:131(use)
        1    0.156    0.156    0.162    0.162 DisplayOpenGL.py:154(destroy)
      262    0.149    0.001    0.196    0.001 function_base.py:3181(add_newdoc)
        9    0.132    0.015    3.644    0.405   init  .py:1()
       76    0.127    0.002    0.457    0.006 Shape.py:18(  init  )
     4712    0.109    0.000    0.319    0.000 Texture.py:89(tex)
```

Le temps est fortement descendu, cependant, on remarque visuellement que le rendu n'est toujours pas suffisamment performant (10 fps).

A ce stade, aucune autre amélioration n'est envisagée. Pour avoir un rendu fluide avec cette solution, il faudrait réduire le nombre d'éléments affichés par l'OSD.

ANNEXE C – TEST DE PERFORMANDE DE QT AVEC QML

C) Code

```
import QtQuick 2.2

Rectangle {
    id: page
    width: 1280
    height: 720

    Row {
        spacing: 150 // a simple layout do avoid overlapping

        Repeater {
            model: 5 // just define the number you want, can be a variable too

            Column {
                spacing: 0 // a simple layout do avoid overlapping

                Repeater {
                    model: 5// just define the number you want, can be a
variable too

                    delegate: Rectangle {
                        id: rect
                        width: 1
                        height: 100
                        color: "red"
                        antialiasing: true
                        anchors.centerIn: page.Center

                        NumberAnimation on rotation {
                            from: 0
                            to: 360
                            duration: 5000
                            loops: Animation.Infinite
                        }
                    }
                }
            }
        }
    }

    Row {
        spacing: 50 // a simple layout do avoid overlapping

        Repeater {
            model: 5 // just define the number you want, can be a variable too


            Column {
                spacing: 100 // a simple layout do avoid overlapping

                Repeater {
                    model: 5 // just define the number you want, can be a
variable too
                    delegate: Text {
                        text: "Hello world !"

                        NumberAnimation on rotation {
                            from: 360
                            to: 0
                            duration: 2000
                            loops: Animation.Infinite
                        }
                    }
                }
            }
        }
    }
}
```

D) Résultats

- Résolution : 1024x768 (résolution de l'affichage de X)
- FPS : 62 (données du profiler Qt)
- UC usage : 39% (*top* sur la RPi)