



System with Multi Antennas to Reorient a Target

D'Acremont Antoine
Cotten Guillaume
Legay Kevin
Kennan Aya
Shehade Mohammed
Rigaud Michaël

Table des matières

| | |
|---|-----------|
| Table des matières | 1 |
| Résumé | 3 |
| Introduction | 4 |
| | |
| I Préparation | 5 |
| 1 Point de Situation | 6 |
| 1.1 Contexte | 6 |
| 1.2 Rappel de notre projet | 6 |
| 1.3 Architecture Fonctionnelle | 7 |
| 1.4 Architecture physique | 7 |
| 1.5 Treillis de détection | 8 |
| 2 Tests unitaires | 9 |
| 2.1 PIC | 9 |
| 2.2 Filtre passe bande | 11 |
| 2.3 VCO | 13 |
| 2.4 Down converter | 14 |
| 3 Test d'intégration | 16 |
| 3.1 Raspberry Pi | 16 |
| 3.2 Android | 18 |
| | |
| II Réalisation | 21 |
| 4 Radiogoniometre | 22 |
| 4.1 Module Montréal | 22 |
| 4.2 Down converter | 22 |
| 5 Interface Web | 24 |
| 5.1 Analyse | 24 |
| 5.2 Conception | 25 |
| 6 Application Android | 29 |
| 6.1 Présentation du S.M.A.R.T Comm Center | 29 |
| 6.2 Fonctionnement de l'application | 30 |

| | |
|---|-----------|
| III Organisation d'équipe | 33 |
| 7 Organisation du travail | 34 |
| 7.1 Méthode de travail | 34 |
| 7.2 Outils utilisés | 34 |
| 7.3 Diagramme de Grant | 36 |
| 8 Difficultés rencontrées | 37 |
| Conclusion | 38 |
| Remerciements | 39 |
| | |
| Annexe | 41 |
| A Validation Fonctionnel Usine | 41 |
| B Validation Fonctionnelle | 42 |
| C Documentation Technique du Raspberry Pi B+ | 43 |
| C.1 Définition | 43 |
| C.2 Caractéristique et connectiques | 43 |
| C.3 Schéma technique | 44 |
| D Documentation Technique à Raspbian | 45 |
| D.1 Définition | 45 |
| D.2 Logo | 45 |
| E Client.py | 46 |
| F Serveur.py | 48 |
| Table des figures | 51 |

Résumé

Ce document présente notre rapport sur le projet SMART « System with Multi Antennas to Reorient a Target » encadré par M. MANSOUR Ali. Ce projet comporte deux phases : la détection d'un drone puis sa neutralisation. Compte tenu du temps imparti nous nous sommes concentré sur la détection. De plus, nous avons choisi de réaliser notre détection avec de la radiogoniométrie, et de proposer à nos utilisateurs deux interfaces de contrôle : une application web, et une application android.

A cause de retard dans la réception de certains composants nous n'avons pas pu concevoir le capteur qui permet de détecter les drones, mais nous avons implémenté tout le système de visualisation et d'alerte, et nous avons testé au maximum les composants qui nous ont été livrés. De cet manière nous pouvons assuré que l'arrivée des quelques composants manquant suffira pour terminer ce projet.

Nous espérons que vous prendrez autant de plaisir à lire ce rapport que nous en avons pris durant tout le déroulement de ce projet.

Mots clés :

Détection, drone, application Android, radiogoniomètres, application web

Introduction

L'actualité récente a montré que l'intrusion de drones dans des sites sécurisés représentaient un risque de sécurité majeur. Des gouvernements et entreprises privées se sont lancés dans la mise au point de systèmes de détection et de neutralisation de ces drones. L'équipe Smart, constituée de Rigaud Michaël, D'Acremont Antoine, Cotten Guillaume, Legay Kevin, Kenaan Aya, et Shehade Mohamed, a cherché également à répondre à cette problématique.

Mais compte tenu du temps imparti, nous avons choisi de nous concentrer dans un premier temps sur la détection d'un drone. De plus, après notre état de l'art effectué au semestre 3 nous avons choisi d'utiliser la technologie de la radiogoniométrie. Plus exactement, nous avons choisi de réaliser la détection en plaçant un champ de radiogoniomètre. Leur données seront transmises à un ordinateur central qui calculera la position du drone et le signalera aux utilisateurs à travers des interfaces graphiques.

Ce rapport présentera dans un premier temps un rappel de notre travail d'analyse effectué au semestre 3 ainsi que les différents tests qui ont été effectué sur le matériel que nous avons acheté. Puis nous présenterons le travail qui a été réalisé, c'est-à-dire le travail pour adapter le Montréal 3v2 à notre problème, la conception de l'interface web et l'élaboration de l'application android. Enfin nous expliquerons comment nous avons organiser notre travail pour obtenir ce résultat, ainsi que les difficultés que nous avons rencontrés.

Première partie

Préparation

Point de Situation

1.1 Contexte

La première partie du projet nous a permis de redéfinir le sujet et les attentes, mais également de réfléchir aux solutions technologiques que nous pourrions utiliser. Après avoir observé les solutions existantes et compte tenu du budget qui nous est imposé, la solution générale retenue sera donc la suivante : les ondes radio captées par les antennes, seront traitées par un filtre et un down-converter avant de passer par le radiogoniomètre et obtenir une position angulaire qui sera envoyée au serveur puis transmise à l'IHM (voir dans le chapitre 1 la partie 1.4).

Cependant, de nombreuses zones d'ombres planent encore sur la réalisation technique, notamment car nous ne maitrisons pas le domaine. En effet, nous utiliserons des plans de carte électronique de radiogoniomètre amateur afin de réaliser notre solution technologique, car réaliser un traitement entièrement numérique ne semble pas réalisable dans le temps impartit.

De plus, actuellement, nous ne nous sommes pas encore réellement penchés sur le traitement des informations ainsi que sur la structure du serveur et de l'IHM. Nous avons seulement convenu que le transfert des informations se fera via Arduino. L'un d'entre eux sera le serveur et dialoguera avec l'ensemble des autres Arduino positionnés sur les antennes.

Actuellement, nous ne possédons aucune partie physique du projet et nous réalisons des listes de commandes afin de pouvoir commencer les tests unitaires le plus rapidement possible.

1.2 Rappel de notre projet

Suite à notre état de l'art, nous avons décidé de réaliser notre système de détection en installant un maillage de capteur qui se baseront sur le système du Montréal 3V2. Chaque capteur sera connecté à un Raspberry Pi 2¹. De plus, chaque Raspberry Pi communiquera avec un ordinateur central qui traitera les données pour les afficher sur une interface graphique. Les données qui seront transmises sont : le numéro du Raspberry Pi , la position du capteur, et le gisement du drone par rapport au capteur. Enfin, l'ordinateur central communiquera avec une application android qui notifiera le client de la présence d'un drone comme on peut le voir sur la figure 1.1.

1. La documentation technique du Raspberry PI est situé en annexe à la page 43

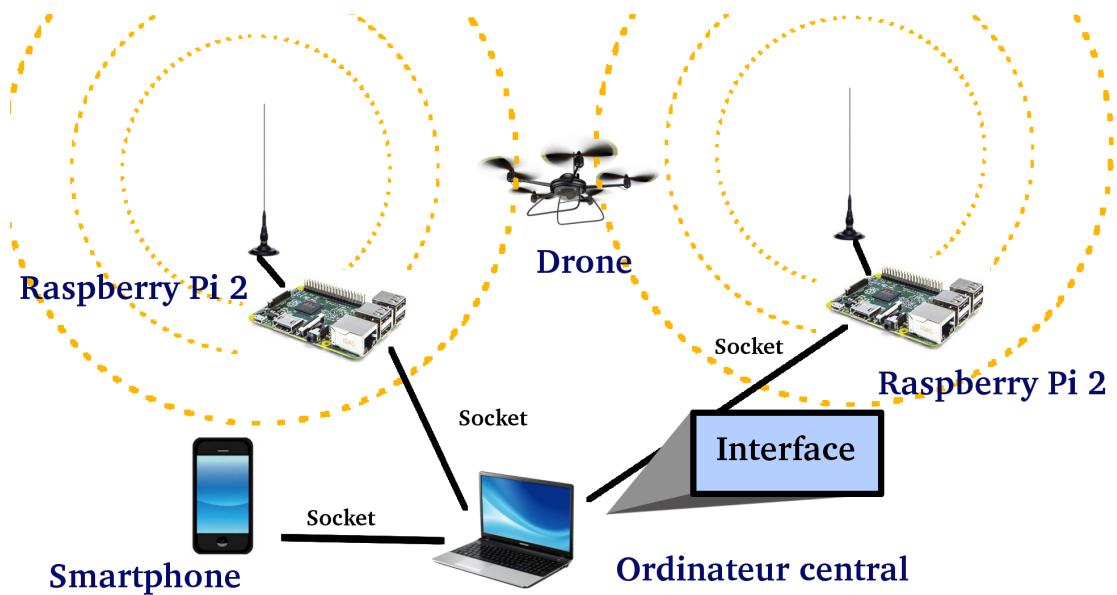


FIGURE 1.1 – Installation de Smart

1.3 Architecture Fonctionnelle

1.4 Architecture physique

L'architecture physique du système est présenté à la figure 1.2.

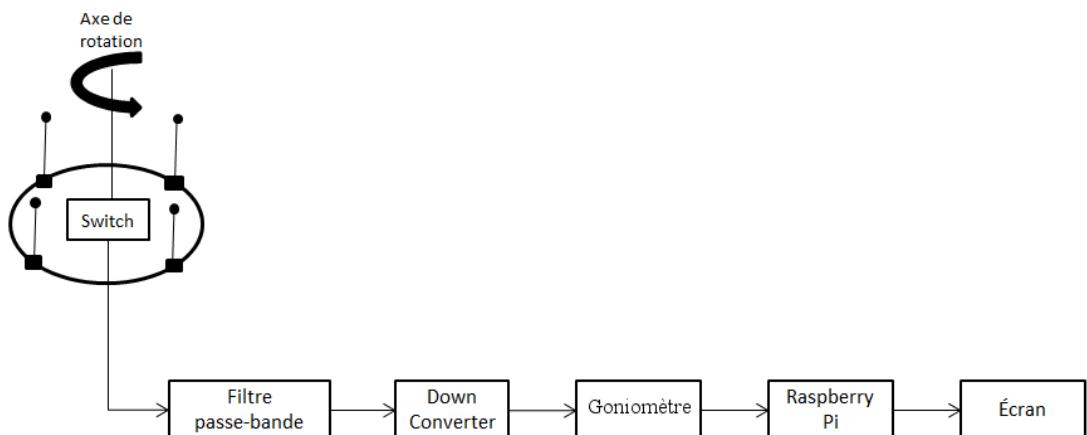


FIGURE 1.2 – Architecture Physique

1.5 Treillis de détection

Pour répondre au besoin de détection et s'assurer d'un correct positionnement de la cible, la mise en place d'une couverture de détection répondant à nos besoins était nécessaire. Les critères retenus pour cette dernière sont les suivants :

- A l'intérieur de la zone de détection, la cible doit être en permanence sous la couverture de détection de 4 radiogoniomètres
- Tenter une optimisation de la couverture afin d'éviter l'installation d'un trop grand nombre de radiogoniomètre

Très peu de sujets similaires ont pu être trouvé bien que le problème soit récurrent dans de nombreux projets.

Cependant, après plusieurs essais, le choix de treillis fut le suivant :

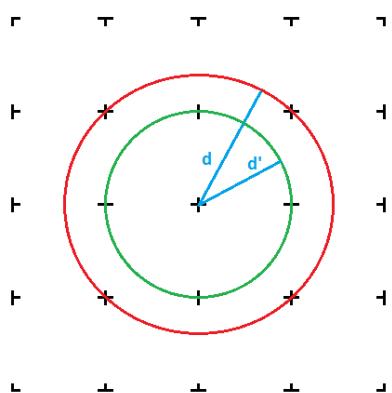
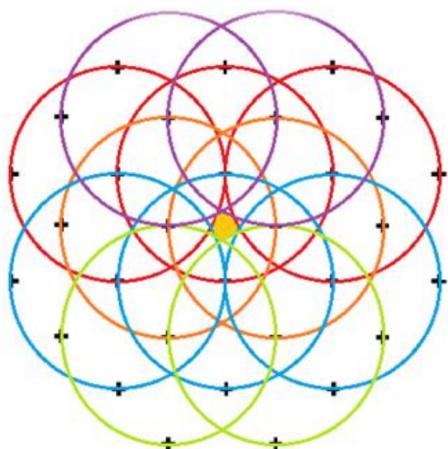


FIGURE 1.3 – Cercle de détection

- Dans les deux cas, les distances d et d' représentent la distance de couverture maximale d'une antenne pour une configuration de treillis particulière, distance au-delà de laquelle nous ne sommes pas sûr d'assurer la détection d'un drone.
- Chaque croix noire représente une antenne. Ces dernières forment ainsi la zone de détection, zone à l'intérieur de laquelle, le drone se doit d'être repéré.

Initialement, nous avions prévu que les antennes radiogoniométrique assureraient la détection jusqu'à son plus proche voisin. Cependant, cette configuration ne permet pas d'assurer qu'un drone traversant la zone soit sous la couverture d'au moins quatre antennes en tout temps. Nous avons donc choisi la configuration représentée par le cercle rouge, c'est-à-dire en rapprochant les antennes les unes des autres. On obtient ainsi la couverture suivante :



Dans cette configuration, les zones de plus faible couverture sont situées sur les antennes elles même. En effet, au-dessus de chaque antenne, la couverture n'est assurée que par quatre d'entre elles. En dehors de celle-ci, la couverture est assurée par cinq à six antennes.

FIGURE 1.4 – Maillage de détection

CHAPITRE 2

Tests unitaires

Suite au choix que nous avons réalisé dans la partie précédente, nous avons commandé notre matériel. Dès la réception de celui-ci nous avons effectué des tests unitaires pour vérifier leur bon fonctionnement. Voici la liste de l'ensemble des tests que nous avons réalisé.

2.1 PIC

Dans le schéma du Montréal 3v2 nous avons pu constater qu'il y avait 3 PIC programmés. Nous avons commandé les PIC programmés au près de l'entreprise F1LVT [?].

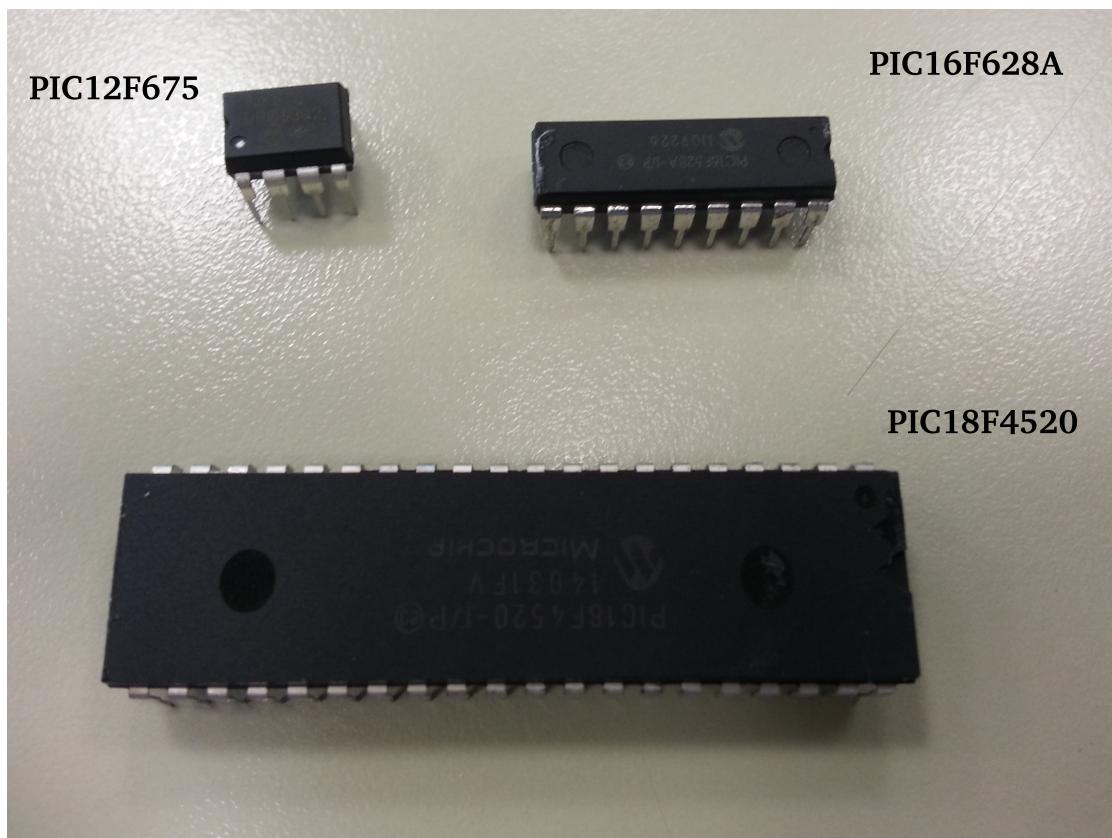


FIGURE 2.1 – 3 PIC programmés

Nous avons ensuite imaginé et réalisé des tests unitaires sur chacun des PIC pour vérifier qu'ils ont bien été programmés et qu'aucune erreur n'est apparue sur ce système de décision critique pour le système.

PIC16F628A

Ce PIC sert à réaliser l'affichage sur les LED. Pour tester ce PIC, nous avons réaliser le montage de la figure 2.2. On peut voir à la figure 2.3 le schéma de montage du PIC sur le Montréal 3v2.

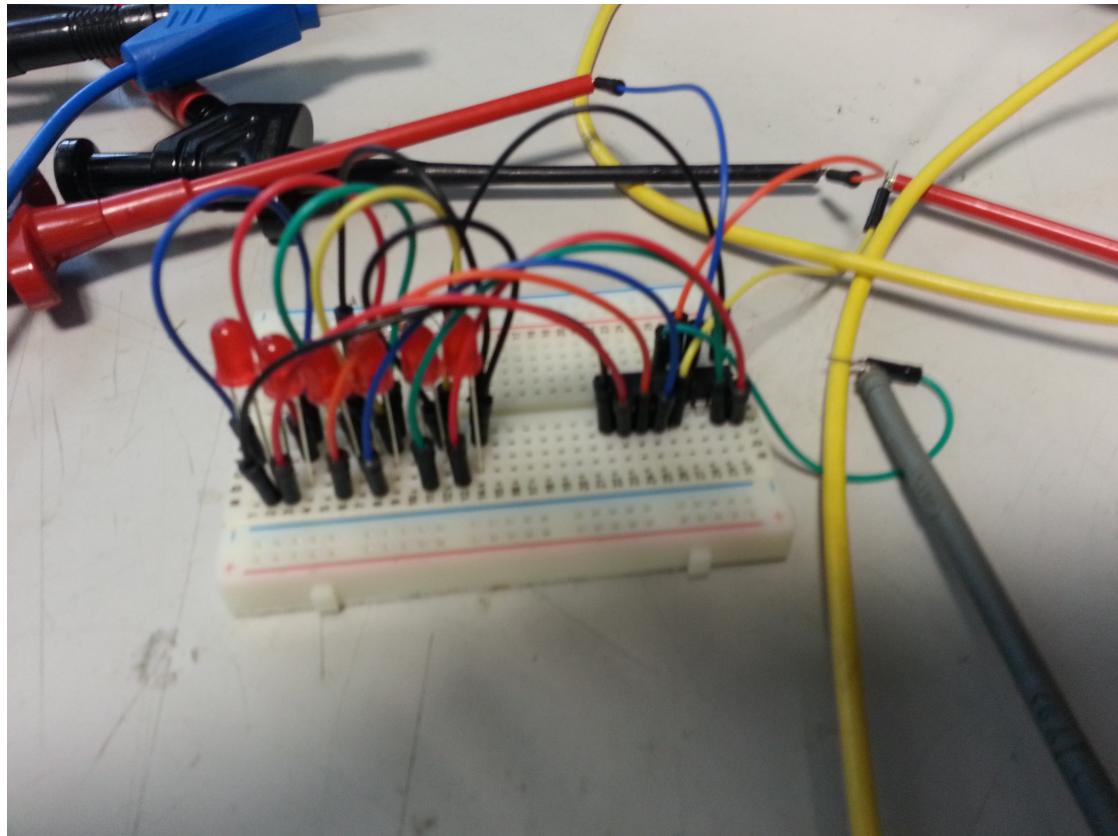


FIGURE 2.2 – Schéma électrique du test unitaire

Pour tester ce composant, nous avons donc choisi de monter une partie des LED situés en sorti, de configurer le « clock » sur un signal carré de fréquence 1 MHz, et de faire varier la fréquence de l'entrée « data ».

Malheureusement nous n'avons pas pu observer de LED s'allumer pendant notre expérience.

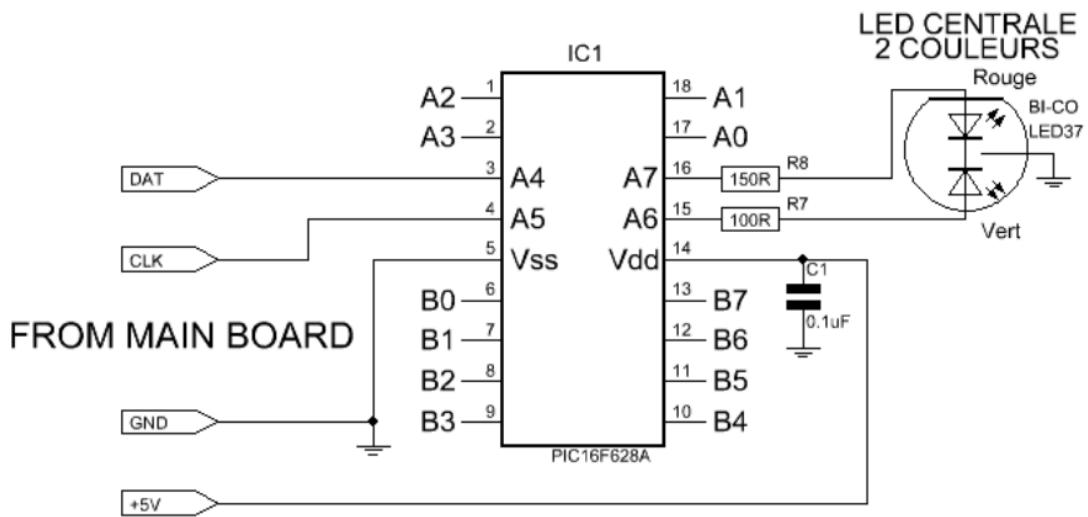


FIGURE 2.3 – Schéma de montage du pic sur le Montréal 3v2

PIC12F675 et PIC18F4520

Ces deux PIC ont également été testé avec le même procédé. Nous ne développerons pas plus ces tests car ils nous ont donné les mêmes résultats.

2.2 Filtre passe bande

Pour le filtre passe bande, nous souhaitions un filtre qui couperait tout ce qui se trouve en dehors de notre bande. Les tests ont montré que ce filtre réagit plutôt bien quand le montage qui y est lié est adapté, ce qui est le cas, le circuit fonctionne bien avec une impédance de 50 Ohm.

Le test unitaire était simple on a branché le filtre sur un analyseur que envoyais un signal et recevait ce même message. Il est alors simple d'obtenir le comportement du filtre. Nous avons obtenu que le filtre diminuait très bien ce que ce trouve avant 2.4GHz mais plutôt mal ce qui vient après. Ceci n'est pas gênant car les bandes entre 2.5Ghz et 5 GHz sont peu utilisées en France.

Sur la photo le curseur sur la courbe est à 2.45Ghz et le plat est un peu plus grand que la bande.

Nous avons mesuré deux paramètres supplémentaires, Le S11 et le S21 qui sont des paramètres permettant de mesurer la perte d'amplitude liée au composant. Le S11 est le coefficient de réflexion à l'entrée lorsque la sortie est adaptée. Dans l'idéal il vaut 0, il n'y a alors aucune réflexion et toute l'amplitude du signal sort du filtre, on obtient le S11 de la photo suivante.

On peut voir ici que le log du S11 est très faible entre 2.4 et 2.5 GHz ce qui indique un faible taux de réflexion et donc que le signal en entrée sera peu atténué.

Les deux plus grands pics vers le bas correspondent aux limites de la bande 2.4-2.5GHz.

Le S21 est le coefficient de transmission direct lorsque la sortie est adaptée, pour celui-ci le but est d'avoir ce nombre le plus proche de 1 et donc son logarithme le plus proche de zéro possible.

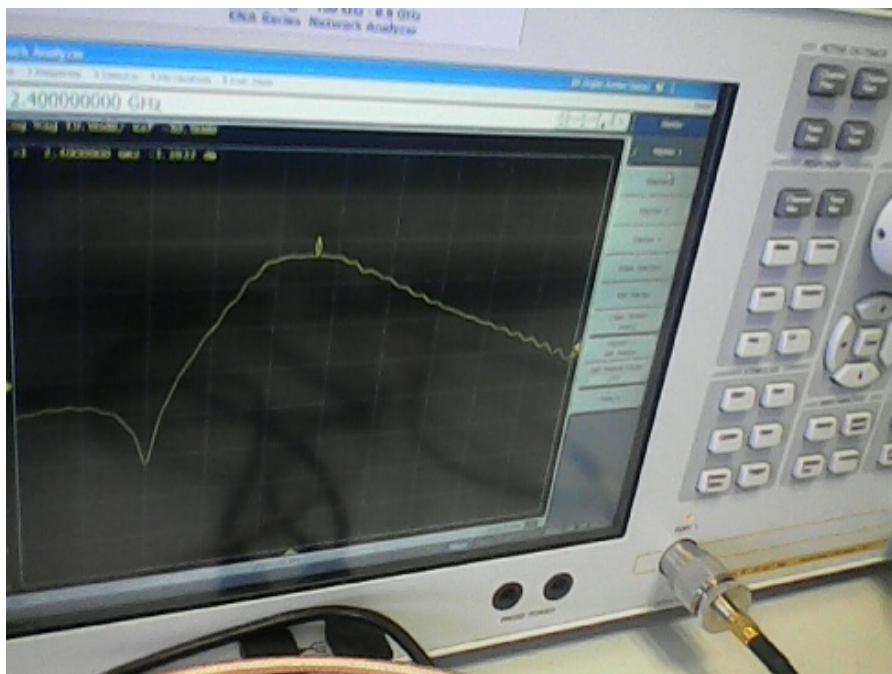


FIGURE 2.4 – Comportement du filtre

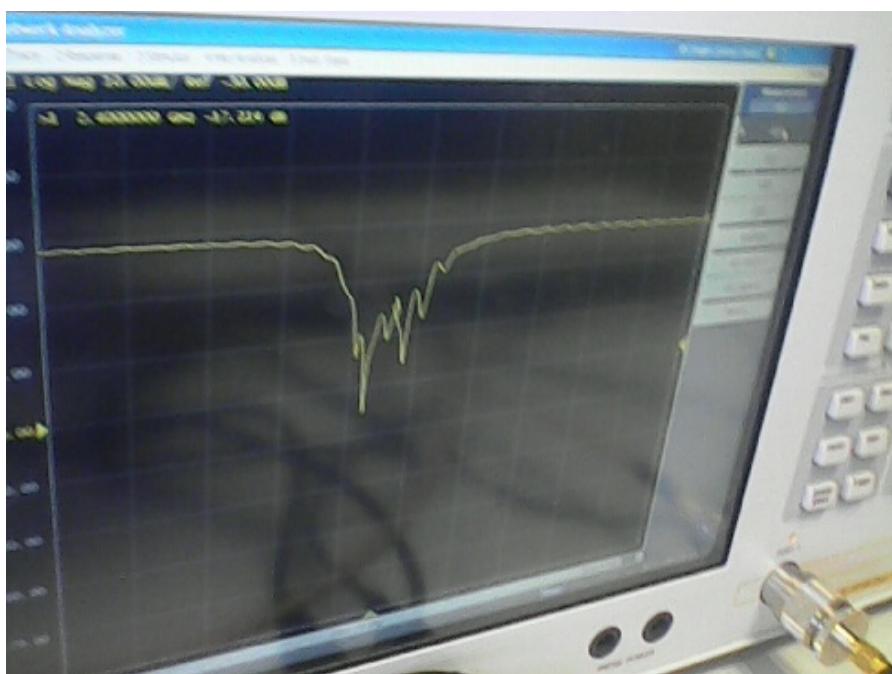


FIGURE 2.5 – S11 du filtre passe bande

Lors du test, nous avions une perte d'environ 2dBm dans la bande de fréquence 2.4-2.5GHz, ce qui est relativement faible, le filtre ne risque donc pas d'occuper ce que l'on souhaite voir en atténuant trop fortement le signal utile.

2.3 VCO

Le test du VCO est simple mais doit être bien fait car sans lui impossible d'obtenir la bonne fréquence de travail en entrée du radio-goniomètre.

Nous avons commencé par mesurer la fréquence libre, c'est-à-dire la fréquence renvoyée par le VCO lorsqu'il est alimenté avec une tension nulle en entrée. La fréquence libre mesurée sur notre matériel était de 1.35 GHz donc plus importante que la fréquence indiquée par le constructeur (1.31Ghz). L'étape suivante consistait à mesurer la tension pour laquelle nous obtenions une fréquence de sortie de 1.9Ghz et nous avons obtenu environ 8V ce qui nous a permis de choisir le bon régulateur de tension pour la suite. Les régulateurs sont calibrés, il est donc difficile d'en trouver un qui corresponde parfaitement mais nous avons pu obtenir un régulateur à 8.1V qui après test donnait en sortie du VCO une fréquence de 1.91GHz. La bande de fréquence à transférer étant de 100 MHz nous n'étions pas à 10 MHz près et il aurait été difficile et coûteux de trouver un meilleur moyen de le faire.

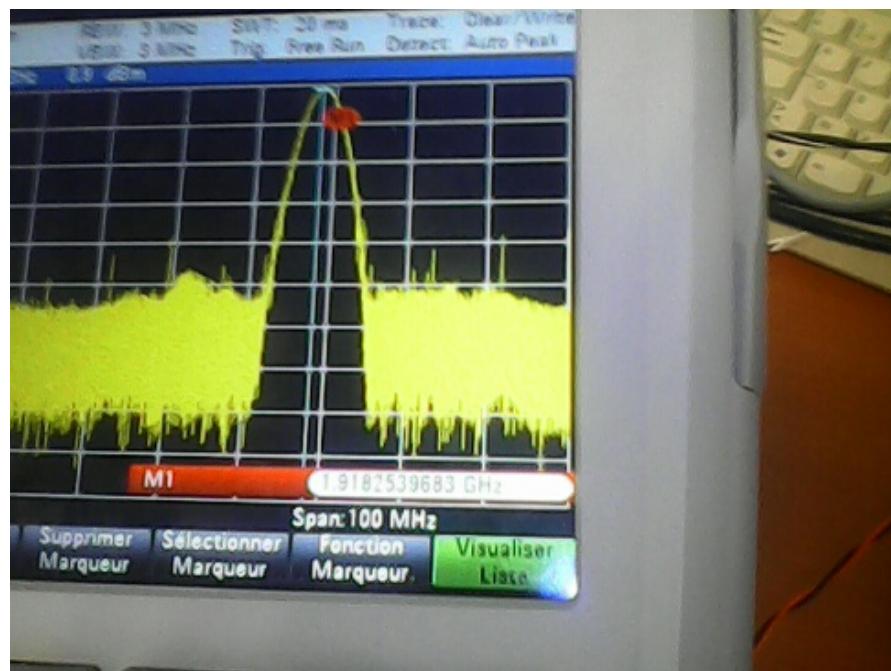


FIGURE 2.6 – Fréquences générées par le VCO alimenté à 8.1V centrée autour de 1.9GHz d'une largeur d'environ 10MHz

2.4 Down converter

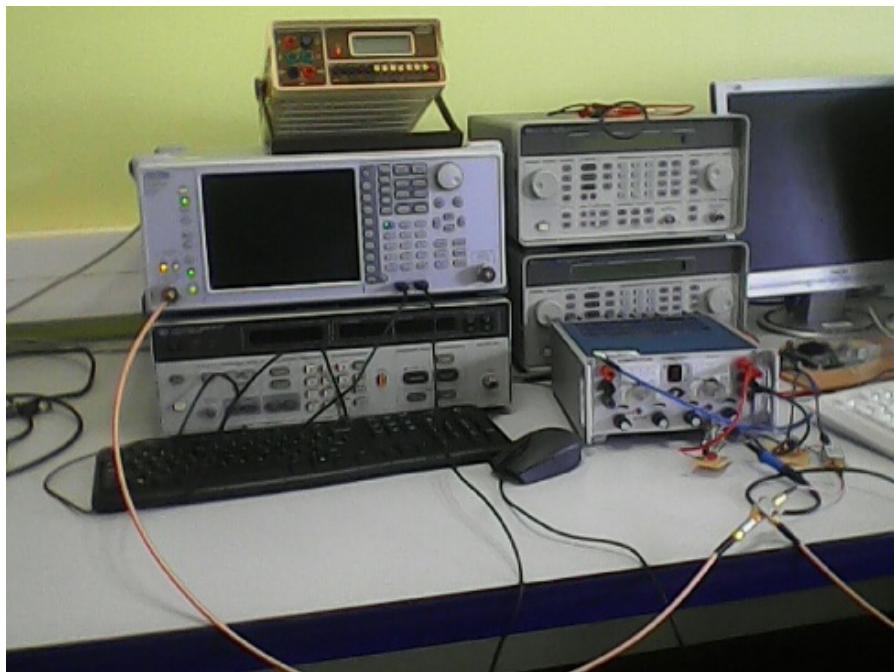


FIGURE 2.7 – Montage de test de l'adaptateur

Le test du down-converter a posé quelques problèmes. En effet le seul moyen de le tester est de le tester dans son cas d'utilisation pratique. Il est, en effet, nécessaire de l'alimenter et ceci ne peut se faire sans le VCO, de même le bruit risque de gêner l'observation, il faut donc utiliser le filtre.

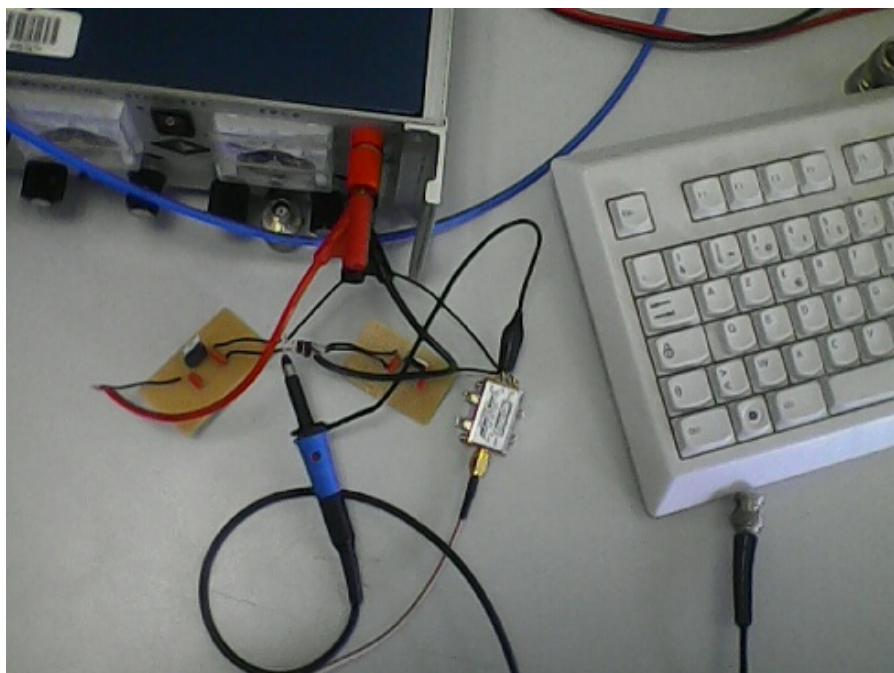


FIGURE 2.8 – Photo du VCO et de son alimentation

Nous avons rencontré des problèmes lors du test, premièrement, des problèmes de communication entre membre de l'équipe ont retardé le test d'une semaine, en effet il a fallu créer un montage avec les deux régulateurs de tension, celui pour l'alimentation et celui pour la tension en entrée. Il manquait cependant une information relative à

l'orientation d'un composant sur les schémas fourni au membre de l'équipe chargé de la soudure et cela a entraîné une erreur de montage avec une augmentation des délais liés à la réalisation de ces filtres.

Deuxièmement, nous n'avions pas prévue les problèmes dus aux câbles. Il a fallu en effet cherché des câbles en n'étant pas certain que le câblage utilisé ne ferait pas griller le matériel.

Troisièmement, le professeur nous ayant aidé lors de la conception théorique du montage était en déplacement, il n'a donc pas pu nous aider lorsque des hésitations se sont fait sentir. Devant le prix du matériel et la possibilité de l'endommager, nous avons choisis d'attendre.

Test d'intégration

Nous n'avons malheureusement pas pu faire de nombreux tests d'intégrations puisque à la fin de notre projet nous n'avions pas la totalité des composants pour monter notre radiogoniomètre. Néanmoins nous avons pu tester l'intégration du radiogoniomètre au Raspberry Pi en plaçant des leds sur les ports GPIO, ainsi que l'intégration de l'application mobile au serveur central.

3.1 Raspberry Pi

La documentation technique lié a notre Raspberry Pi es situé en annexe à la page 43

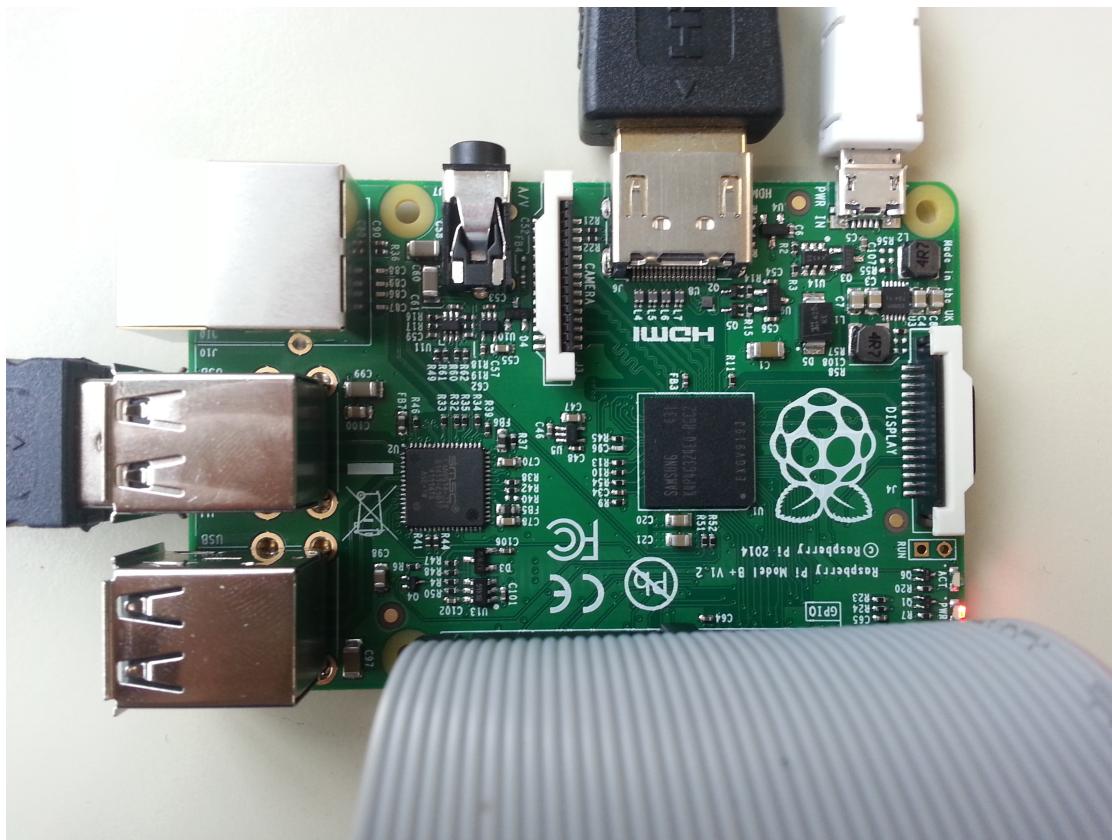


FIGURE 3.1 – Notre Raspberry Pi B+

Pour s'assurer que notre Raspberry Pi répond aux spécifications fonctionnelles et qu'il fonctionne correctement en toutes circonstances pour notre projet, nous y avons réalisé des tests intégrations.

Après avoir enfin installé le système d'exploitation Raspbian¹ sur notre Raspberry Pi B+, nous avons tenté de tester les ports GPIO. Pour cela, dans un premier temps, nous avons allumé des LED grâce à un script python à travers différents ports GPIO. Sur la figure 3.2, on peut observer que nous avons allumé une LED grâce au port 22.

Dans notre projet le Raspberry Pi sera placé entre le radio-goniomètre à effet Doppler et l'utilisateur. Il aura deux tâches, corrélérer les données entre tous les dispositifs pour obtenir la position du drone et afficher le résultat à l'utilisateur. Pour cela il doit récupérer la direction qui est donné par le Montréal 3v2. Cette position est donnée à travers des LED (voir figure 3.3). Nous allons donc placer le Raspberry Pi au niveau des LED pour obtenir les informations délivré par le Montréal 3v2.

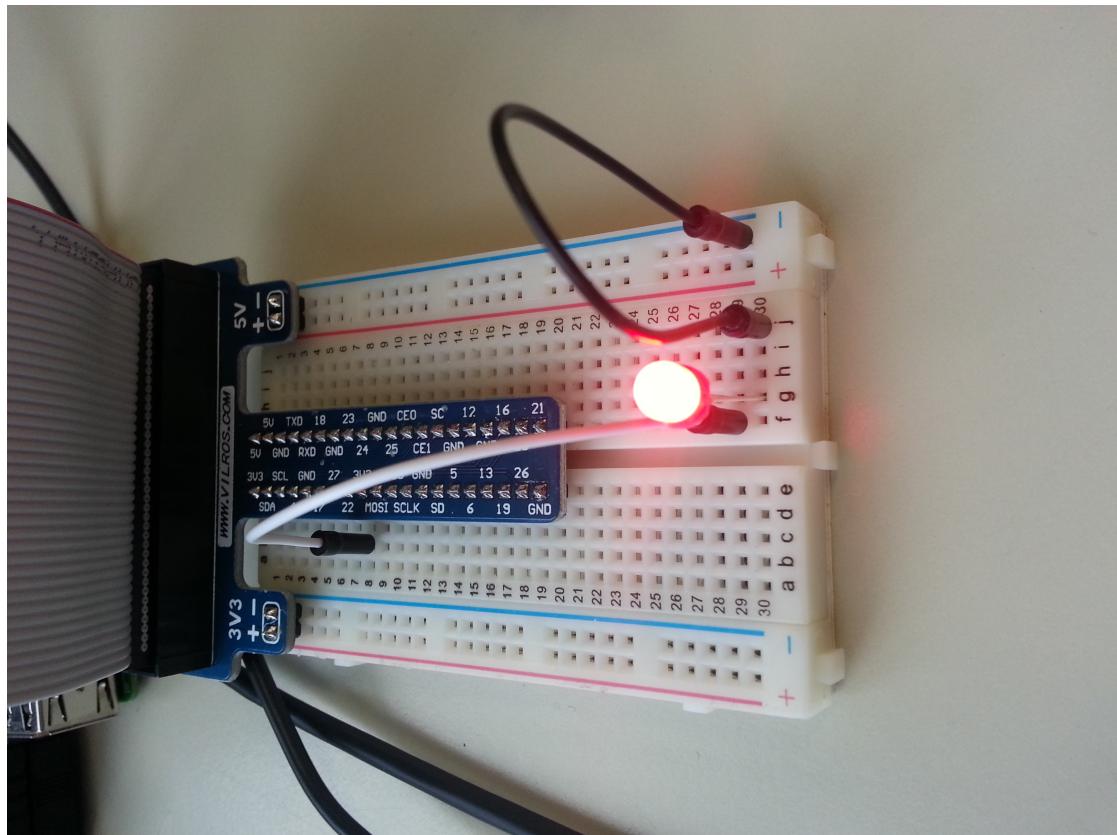


FIGURE 3.2 – Allumage d'une LED par Raspberry Pi

La sortie du Montréal 3v2 est décrite à la figure 3.3. On constate que le pic qui permet l'affichage à 12 sorties qui lui permet de gérer 24 LED. Pour connaître quelle LED est allumé, il faut savoir laquelle des entrées A1,A0,B7,B6,B5,B4 à un front montant et laquelle des entrées B0,B1,B2,B3,A3,A2 à un front descendant.

Pour modéliser une LED en entrée du Raspberry Pi, nous avons positionné 2 boutons poussoirs (voir figure 3.4). Le premier permet de réaliser le front montant et le second le front descendant. Ainsi en positionnant ces boutons au bon endroit par rapport au port GPIO du raspberry il est possible de connaître quelle LED on a simulé.

Nous avons réalisé un script python qui lie les entrées du raspberry avec les sorties du pic. Puis nous avons testé en simulant une LED comme décrit précédemment.

On peut constater que l'expérience est un succès car le raspberry pi nous renvoie bien le numéro de la LED que nous voulions tester.

1. La documentation lié à Raspbian est située en annexe à la page 45

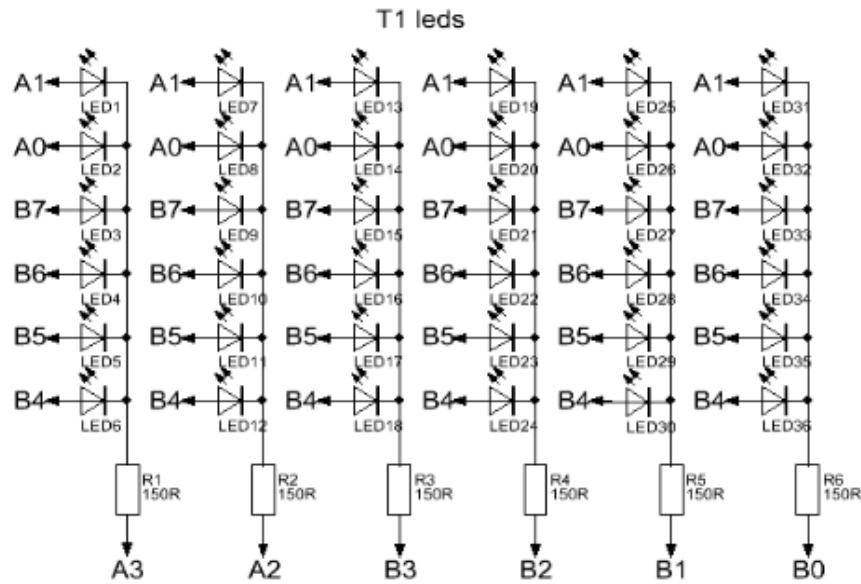


FIGURE 3.3 – Méthode de connexion des leds dans le Montréal

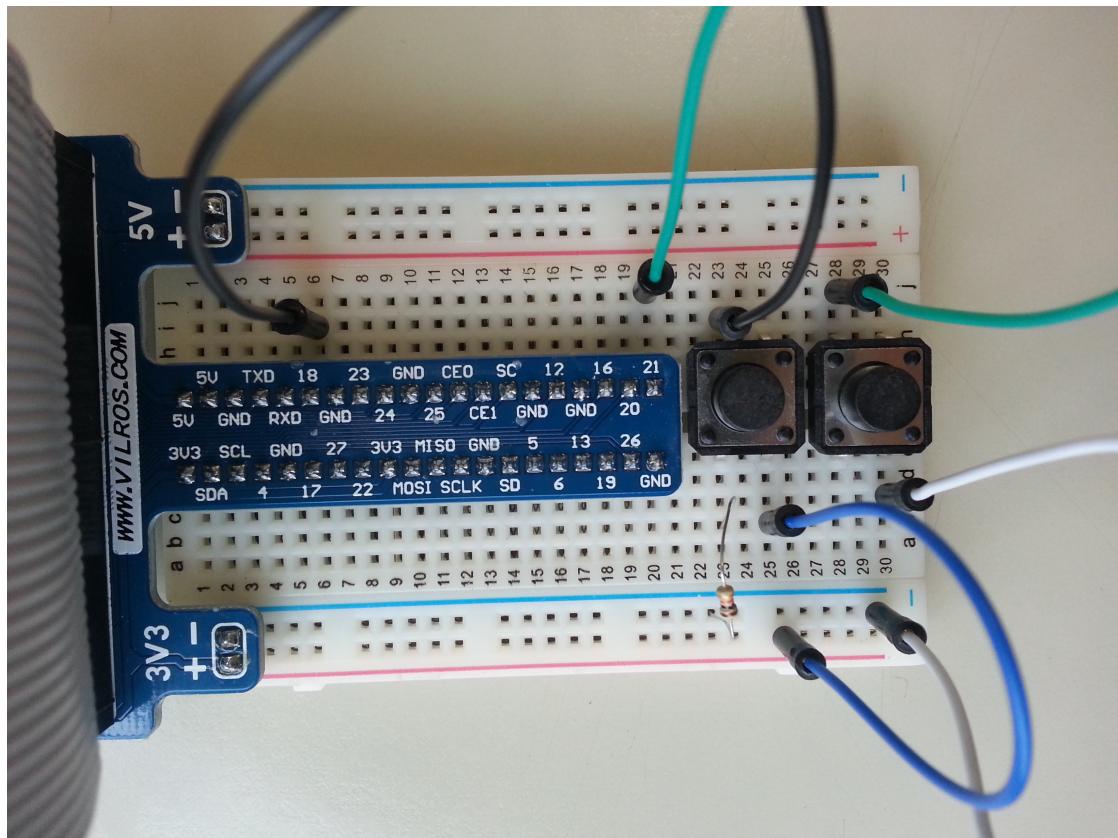


FIGURE 3.4 – Système modélisant une LED

3.2 Android

Les premiers tests de l’application Android étaient de simples tentatives de connexions avec transmission de messages simples pour vérifier le bon fonctionnement des socket Web. Une fois l’application terminée une série de tests plus complets ont été effectués pour simuler le comportement global de l’application en conditions réelles.

Le premier test d'intégration réalisé simule une succession de quatres instances de communication entre le client Android et le serveur. L'objectif est de simuler la transition d'un état "Rien à Signaler" vers un état "Drone détecté", et de vérifier l'exactitude des informations transmises. Pour visualiser le comportement du téléphone nous avons utilisé l'outil "logcat" du logiciel de programmation Android Studio qui permet de visualiser en temps réel les registres du téléphone.

```

Android Monitor
LGE Nexus 5X Android 6.0.1, API 23 | com.example.smart.smartcommcenter (6603) | Verbose
logcat Monitors →
[red box] > I/System.out: Demande de connexion
[red box] > I/System.out: Vous etes connecte.
[red box] > I/System.out: 0
[red box] > I/System.out: Fin de communication
[red box] > I/System.out: Demande de connexion
[red box] > I/System.out: Vous etes connecte.
[red box] > I/System.out: 0
[red box] > I/System.out: Fin de communication
[green box] > D/OpenGLRenderer: endAllStagingAnimators on 0x7f6b285800 (RippleDrawable) with handle 0x7f81d3bf00
[green box] > I/System.out: Demande de connexion
[green box] > I/System.out: Vous etes connecte.
[green box] > I/System.out: 1 128.0324998887 95.057113932915
[green box] > I/System.out: Fin de communication
[green box] > I/System.out: X=128.0324998887 Y=95.057113932915
[green box] > D/OpenGLRenderer: endAllStagingAnimators on 0x7f6c75ac00 (RippleDrawable) with handle 0x7f77dfe1a0
[green box] > D/OpenGLRenderer: endAllStagingAnimators on 0x7f6a0a8000 (RippleDrawable) with handle 0x7f77e6e840

```

Premier échange
Message "0" renvoyé par le serveur correspondant à une absence de détection

Second échange

Troisième échange
Message serveur avec position

Information renvoyée à l'utilisateur

FIGURE 3.5 – Premier test : communication

Sur cette image on peut visualiser les trois premiers échanges. Les deux premières requêtes ne donne lieu à aucun changement sur l'application car aucun drone simulé n'a été détecté. Le troisième échange résulte de l'entrée d'un drone dans la zone simulée. On d'abord peux visualiser le message brut reçu par l'application puis dans un second temps la chaîne de caractères formatée par l'application avant qu'elle soit affichée par l'interface.

Le second test d'intégration lié à l'application avait pour but de vérifier le bon affichage des erreurs de connexion sur l'application Android pour éviter toute ambiguïté.

```

Android Monitor
LGE Nexus 5X Android 6.0.1, API 23 | com.example.smart.smartcommcenter (6603) | Verbose
logcat Monitors →
[red box] E: java.net.ConnectException: failed to connect to /192.168.9.30 (port 6666): connect failed
E:     at libcore.io.IoBridge.connect(IoBridge.java:124)                         Message d'erreur
E:     at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:183)
E:     at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:163)
E:     at java.net.Socket.startupSocket(Socket.java:592)
E:     at java.net.Socket.tryAllAddresses(Socket.java:128)
E:     at java.net.Socket.<init>(Socket.java:178)
E:     at java.net.Socket.<init>(Socket.java:150)
E:     at com.example.smart.smartcommcenter.Client.doInBackground(Client.java:67)

```

FIGURE 3.6 – Second test : erreurs

En cas d'erreur le message est bien visible dans les logs et L'application ne s'est pas bloquée. Celle-ci sera donc capable de bien différencier une absence de détection d'une erreur de connections tout en restant prête à renvoyer une requête au serveur. De plus, le message d'erreur est bien affiché dans son intégralité sur l'interface de l'application. elle est ici réglée en mode "Debug" pour afficher un maximum d'informations.

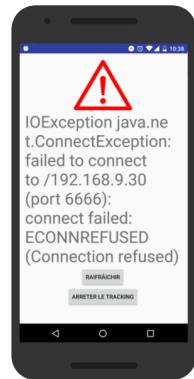


FIGURE 3.7 – Message d'erreur sur l'application

Deuxième partie

Réalisation

Radiogoniometre

4.1 Module Montréal

Le module Montréal comprends deux cartes électroniques : la carte principale qui est chargée de toutes les tâches relatives au traitement du signal reçu par les antennes et la carte destinée à l'affichage de la direction.

Les supports des cartes sont en résine époxy sur laquelle ont été apposée des pistes en cuivre en configuration mono-couche (les pistes ne sont présentes que sur une face de la plaque). Un étamage par dépôt chimique des piste a été réalisé pour les protéger de l'oxydation. L'ensemble de ces opération ont été effectuées à l'école avec l'aide de Mr. Gallou.

Le PIC18F4520 chargé de la majorité des traitements et le PIC12F675 ont été soudés sur la carte principale. Le PIC16F628A, chargé de l'affichage des LEDs a été soudé sur la carte d'affichage de direction. Le montage choisi pour ce dernier PIC permet de le désolidariser de la plaque afin d'effectuer les tests d'allumage des LEDs de direction.

L'ensemble des composants que nous avions a disposition (LEDs, condensateurs et résistances) ont été ensuite soudés sur les deux plaques.

Par rapport au montage d'origine nous n'utiliseront pas le RS232. Toutefois, lors de la rédaction de ce rapport, il manque deux filtres sur la carte principale qui nous empêchent de valider son fonctionnement.

4.2 Down converter

Le radiogoniomètre Montréal 3v2 fonctionne à une fréquence de 500Mhz. Sans modification il est impossible de l'utiliser entre 2.4Ghz et 2.5 GHz, bande de fréquence utilisée par les drones que nous souhaitons détecter. Nous avons donc cherché un moyen d'adapter ce radio-goniomètre aux fréquences souhaitées.

Une solution applicable à notre système est l'utilisation d'un down-converter. Ce composant reçoit deux entrées, le signal dont on veut changer la fréquence(RF) et un signal de fréquence fixe Df(LO). Le down-converter diminue la fréquence du premier signal de celle du second. La sortie(IF) correspond au signal modifié. Son principe de fonctionnement est illustré à la figure 4.2.

On utilisera donc le down-converter pour abaisser la fréquence reçue par les antennes (2.4GHz) à la fréquence de travail du radio-goniomètre.

La fréquence du signal Df est donnée par un VCO (Voltage Controlled Oscillator

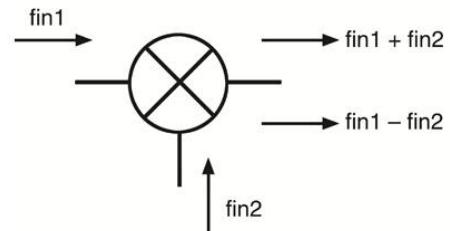


FIGURE 4.1 – schéma de fonctionnement d'un mixer

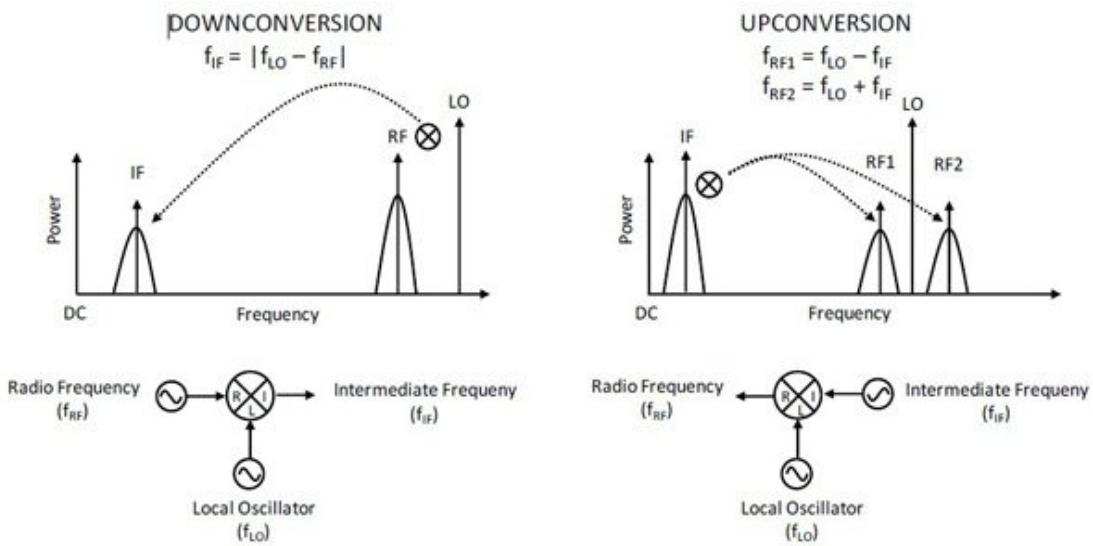


FIGURE 4.2 – principe du fonctionnement d'un mixer

ou oscillateur contrôlé en tension), le VCO reçoit en entrée une tension et donne en sortie une sinusoïdal de fréquence dépendante de la tension d'entrée. Le VCO étant très sensible, il est nécessaire de stabiliser la tension d'entrée et l'alimentation. On utilise donc un régulateur de tension qui amène une entrée stable. Le régulateur est un composant qui viendra limiter à une valeur seuil V_{max} la tension qu'il reçoit en entrée si celle-ci dépasse ce seuil. Dans le cas où $V < V_{max}$ la sortie du régulateur sera égale à l'entrée.

Le VCO étant un composant actif et sensible aux variations dans sa tension d'alimentation, un second régulateur a été utilisé pour alimenter le VCO, toujours dans le but d'obtenir une fréquence stable ne variant pas pendant le processus. Il est en effet indispensable que cette fréquence reste fixe pour que l'effet doppler soit toujours visible et exploitable. Des fluctuations incontrôlées de la tension d'alimentation ou de la fréquence pourraient perturber la localisation.

Pour améliorer la mesure, il est utile de filtrer au maximum toutes les fréquences pouvant perturber la mesure et les différents bruits électromagnétiques. Ces phénomènes ont été atténusés en positionnant en entrée du down-converter un filtre passe-bande qui permet de conserver uniquement la portion du spectre qui nous intéresse soit la bande située entre 2.4Ghz et 2.5Ghz.

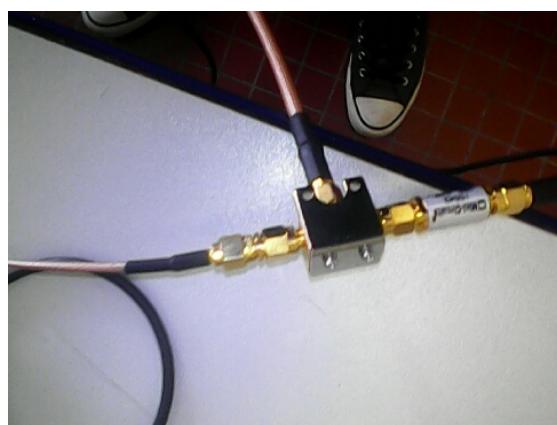


FIGURE 4.3 – le down-converter et le filtre passe bande

A l'aide de ce montage on peut améliorer le signal en entrée pour qu'il puisse être traité par le radio-goniomètre.

Interface Web

Une fois le capteur terminé nous nous sommes penchés sur l'interface Homme/Machine. Dans une volonté de délivrer à l'utilisateur une interface agréable et lisible, nous avons décidé de proposer dans un premier temps une interface web.

5.1 Analyse

Dans un premier temps nous nous sommes penchés sur une phase d'analyse.

Dans la phase d'analyse, on cherche d'abord à bien comprendre et à décrire de façon précise les besoins des utilisateurs ou des clients concernant cette interface. Que souhaitent-ils faire avec le logiciel ? Quelles fonctionnalités veulent-ils ? Pour quel usage ? Comment l'action devrait-elle fonctionner ? C'est ce qu'on appelle « l'analyse des besoins ». Après validation de notre compréhension du besoin, nous imaginons la solution. C'est la partie analyse de la solution.

Dans la phase de conception, on apporte plus de détails à la solution et on cherche à clarifier des aspects techniques, tels que l'installation des différentes parties logicielles à installer sur du matériel. Pour réaliser ces deux phases dans un projet informatique, nous utilisons des méthodes, des conventions et des notations. UML fait partie des notations les plus utilisées aujourd'hui. Pour faciliter à nos clients d'obtenir la direction des drones on a créé une interface web qui répond à leur besoin.

Uml

Pour décrire au mieux ce besoin, nous avons commencé par réaliser un cas d'utilisation de l'interface (figure 5.1).

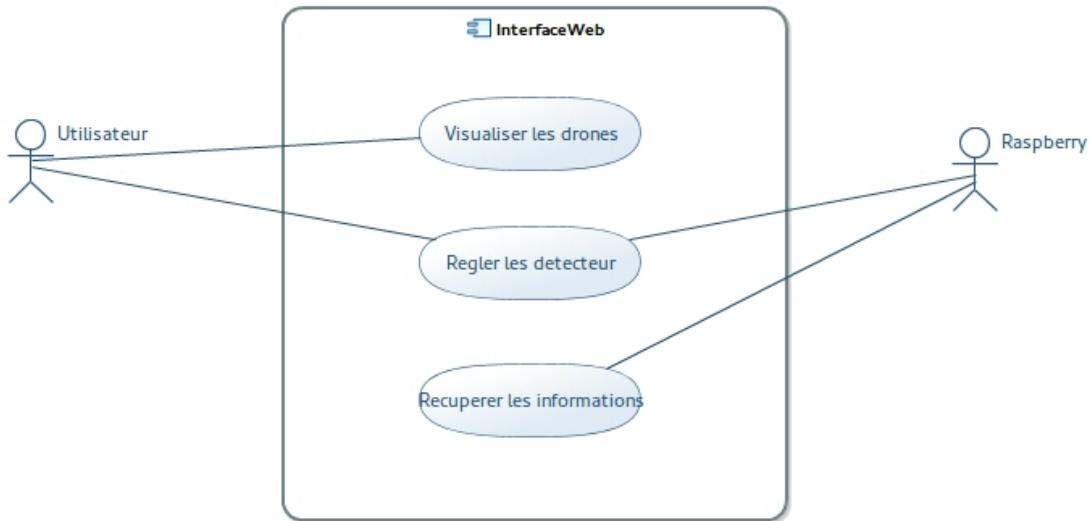


FIGURE 5.1 – Cas d'utilisation de l'interface

Ensuite, nous avons cherché à réaliser un diagramme de classe de notre interface. Pour cela nous avons défini 3 classes principales :

- index.php, qui réalise l'affichage dans un navigateur
- serveur.py, qui récupère les données de chacun des radiogoniomètres
- client.py, installé sur chaque radiogoniomètre il envoie les données des capteurs à travers un socket au serveur.

Le diagrammes de classe de la figure 5.2, montre ce fonctionnement.

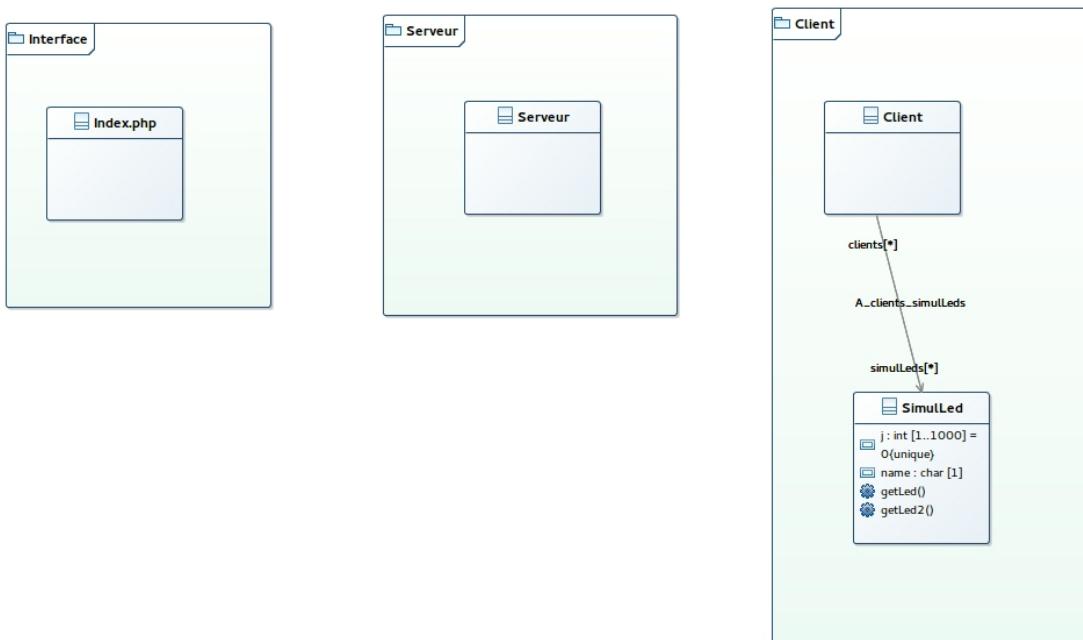


FIGURE 5.2 – Diagramme de classe

5.2 Conception

Client-Serveur

Pour commencer nous avons cherché à donner une interface à nos Raspberry Pi pour communiquer avec l'ordinateur central, c'est l'interface Client/Serveur.

Cette interface a été codé en python. Nous avons choisi le python car c'est un langage souple et rapide. Nous avons réalisé la connection à l'aide d'un socket TCP/IP. De plus, le serveur se base sur du multi thread pour accepter plusieurs client. Le client se connecte donne, son identifiant puis sa communication est placé dans un thread. Ainsi, on a un serveur qui peut accepter une infinité de client.

Pour lancer le serveur il faut lancer le programme *serveur.py* qui se met en écoute de client. Pour lancer le client il faut lancer le programme *client.py*. Une illustration du terminal lors de l'exécution de ces commandes est donnée ci-dessous.

```
1 ./serveur.py
2 Serveur pret, en attente de
   requetes ...
3 Client RP1 connecte, adresse IP
   127.0.0.1, port 51632.
4 RP1> led21
5 donc la direction 200
6 [...]
7 Client RP1 deconnecte.
8 [...]
9 Fin du serveur
```



```
1 ./client.py
2 Connexion etablie avec le serveur.
3 Vous etes connecte. Envoyez vos
   messages.
4 la led allume: led21
5 [...]
6 Connexion interrompue.
```

On peut constater que la seule donnée que le Raspberry Pi envoie en continue est le numéro de la led qui s'est allumé. En effet, le modèle du Montréal 3v2 affiche le gisement sur un cadrant de 36 led. Dans notre solution nous avons décidé de garder ce cadrant et de placer le Raspberry Pi en parallèle du circuit pour détecter laquelle led s'est allumé. De plus, nous avons décidé de réaliser la conversion led/gisement au niveau du serveur pour des questions de modularités.

Une fois les données reçues par le serveur, celui-ci les écrit dans un fichier qui s'appelle *position*.

Web

Ensuite, nous nous sommes penchés sur notre page web. Cette page web doit permettre à notre utilisateur de visualiser en temps réel la position du drone dans notre maillage. Pour cela nous avons fait le choix d'utiliser des solutions dynamiques tel que PHP, Javascript et JQuery.

Pour nous faciliter dans la réalisation du schéma modélisant notre treillis, nous avons choisi SVG. En effet, SVG nous permet de réaliser notre schéma de manière vectorielle. C'est-à-dire que l'on peut donner avec précision la position des capteurs ainsi que de leur droite de détection, mais surtout on peut rendre cela dynamique.

Le script ouvre le fichier *position* qui a été précédemment rempli, et ajoute les points qui sont inscrit dans le fichier avec leur droite de détection. Mais cela ne suffit pas. En effet, si l'on s'arrête là on n'a qu'une position visible par un être humain il reste encore à la positionner. Pour cela nous utiliserons la triangulation (voir à la page 5.2).

Enfin, nous affichons de manière classé l'ensemble des Raspberry Pi qui étaient présent dans le fichier *position* à la droite de l'écran.

Triangulation

Les différents radiogoniomètres nous donnant un gisement de la détection, nous pouvons donc réaliser une triangulation de la position du drone lorsqu'un nombre suffisant d'antenne détecte le drone. Bien qu'une première estimation de la position peut-être obtenu à partir de deux drones, nous considérons que le drone doit être détecté par au moins quatre senseurs pour que la position soit acceptable.

Cependant, avant de pouvoir réaliser toute triangulation, l'acquisition des points d'intersection entre les droites de détection issue du gisement fourni par les différents radiogoniomètres est nécessaire. Pour se faire, à partir des angles, nous formulons une équation de droite plan, passant par les radiogoniomètres respectifs, et tentons de trouver une solution à chaque système composé de deux droites.

Suite à la résolution de ces différents systèmes, nous obtenons une liste de différentes solutions, solutions ici schématisé avec des points de couleur grise. Nous observons que, du fait de la portée de détection, ainsi que la géométrie de notre treillis, certains points sont incohérents ou très imprécis.

Notre première approche fut donc de positionner le drone à la moyenne de l'ensemble des positions solutions d'un des systèmes précédent. Cependant, après simulation, il s'est avéré que, de par l'imprécision relative des radiogoniomètres (les angles sont donnée à 5° près), la moyenne de donnait qu'une idée toute relative de la position du drone et souvent loin de la vérité, et cela à cause d'intersections multiples entre les droites de détection. Pour corriger cela nous avons fait le choix d'utiliser la médiane afin de supprimer tous les résultats incohérents. A partir de la médiane des résultats, nous appliquons un gabarit circulaire et retenons tous les points d'intersection compris dans ce gabarit. Une moyenne est alors appliquée à l'ensemble de ces résultats nous permettant d'obtenir un résultat plus cohérent et moins sensible aux erreurs.

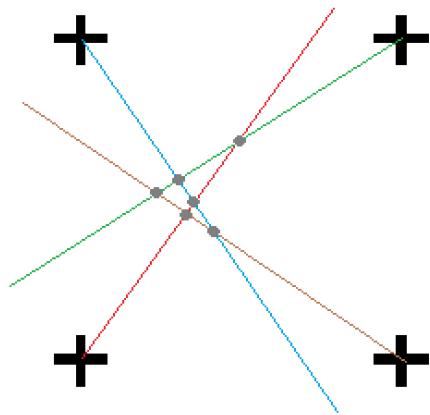


FIGURE 5.3 – triangulation

Au final, nous obtenons la page web suivante :

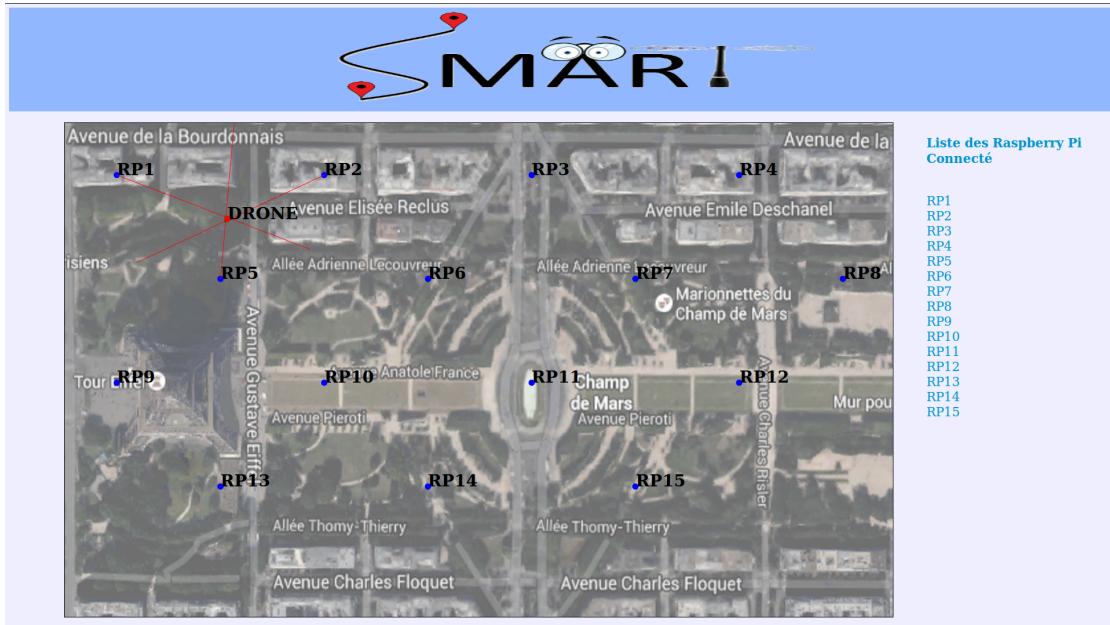


FIGURE 5.4 – Interface Web

Application Android

6.1 Présentation du S.M.A.R.T Comm Center

Le système SMART se base sur un ensemble de capteurs reliés à une station centrale dont les données sont accessibles depuis internet. Cela permet d'accéder aux paramètres du système à distance depuis un autre appareil relié à internet de préférence un ordinateur. Toutefois, le système, étant donné son coût réduit se destine à être utilisé au sein de structures de petites taille ou le personnel en charge de la sécurité du site doit parfois se déplacer en fonction de ses autres obligations et ne peut être constamment en train de surveiller l'état du S.M.A.R.T depuis un ordinateur.

Pour pallier à ce manqueument, il semblait intéressant de proposer une solution sur téléphone mobile qui permettrait d'avertir l'utilisateur final où qu'il se trouve. Deux options existent :

- Une version mobile de l'interface web
- Une application dédiée

Ces deux utiliseraient les APIs des différentes plateformes mobiles existantes (Windows phone, Android, iOS) pour avertir l'utilisateur en utilisant les fonctions vibreur ou la sonnerie du téléphone. Toutefois, les capacités du site mobile sont assez limitées car de nombreux éléments de sécurité peuvent restreindre l'accès à certaines fonctionnalités du téléphone comme l'accès au vibreur ou la possibilité de s'exécuter en tache de fond. De plus ces restrictions varient en fonction de la plateforme mobile visée.

L'application mobile a donc l'avantage d'offrir plus de latitude au développeur et d'implémenter plus facilement différents moyens d'alerte pour l'utilisateur. Il faut cependant garder à l'esprit le fait que ce choix de développement implique de réaliser une application par système d'exploitation mobile existant.

Le choix final pour la version actuelle du système S.M.A.R.T a donc été celui de l'application mobile. Compte tenu des équipements dont nous disposons le système sur lequel l'application a été développé est Android. En effet, une grande partie du code source est sous licence GPL¹ et le codage des applications se fait en Java dans sa version 1.7. De plus, ce système d'exploitation mobile représente en Janvier 2016 64 % du parc mobile français.

1. GPL : "General Public License" régissant la distribution des logiciels libres

6.2 Fonctionnement de l'application

Vue d'ensemble

Le système S.M.A.R.T utilise un serveur web pour gérer l'affichage des données à destination de l'utilisateur final, il est en mesure de créer des connexions vers plusieurs appareils distants. L'application jouera le rôle de client et recevra du serveur les information de position du drone en cas d'intrusion. Dans l'état actuel la réception des données de position par l'application se fait en mode "pull". Cela signifie que c'est l'utilisateur qui lance la demande d'information et le serveur répond ensuite à la requête.

Un mode automatique, avec un rafraîchissement régulier de l'information, a été codé et implémenté mais n'a pas été utilisé dans la version finale de l'application. Il est aussi possible de passer par un mode "push", où le serveur envoie l'information de lui même vers le client dès que celle-ci est mise à jour. Les raisons de ces choix technologiques seront détaillées par la suite.

Choix du niveau d'API

La version actuelle de S.M.A.R.T Comm Center a été développé avec un niveau d'API Android minimum de 19². Le choix d'un niveau aussi élevé d'API a été déterminé par trois éléments importants : la gestion des "threads", des tâches asynchrones et des socket. En effet depuis l'API 19 Google, qui édite et maintient le code d'Android, a modifié la façon dont les connexions réseau étaient gérés sous Android et le fonctionnement actuel qui sera détaillé par la suite nous convenait mieux.

Il faut cependant noter qu'un tel choix limite le nombre de smartphones qui seront en mesure de lancer l'application. L'API 22 par exemple représente seulement 34 % du nombre total d'appareils Android activés. Dans un souci de faciliter la réalisation de l'application nous avons maintenu ce choix, sachant que l'ensemble des téléphones pouvant exécuter du code écrit avec un niveau d'API de 19 représente en Janvier 2016 75.6% du nombre total des téléphones Android activés dans le monde. Ce choix n'est donc pas si restrictif au vu du nombre d'appareils touchés (plus d'un milliard).

Achitecture détaillée

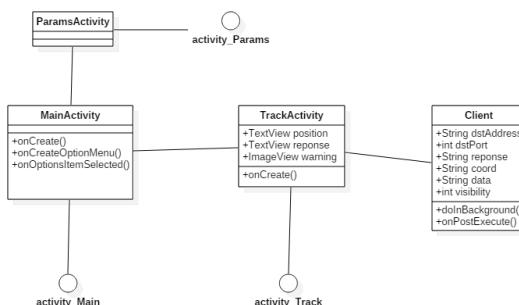


FIGURE 6.1 – Diagramme de classe de l'application

La structure globale de l'application est visible dans le diagramme ci-dessus. Elle regroupe quatre classes, trois liés aux différents écrans de l'application (XxxActivity) et une pour le client web. À chaque classe liée à un écran est associé un fichier .xml qui définit l'aspect visuel de l'écran présenté et appelé activity_xxxx. Ce fichier est chargé via les méthodes "onCreate()" des différentes classes d'activité.

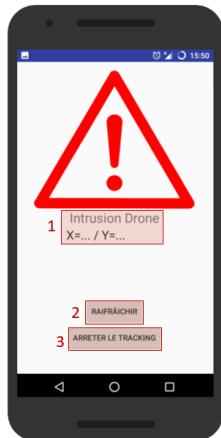
2. L'API correspond à la version d'Android ciblée et détermine donc les fonctionnalités disponibles pour le développeur. le niveau 19 correspond à Android 4.4 KitKat.

Les applications Android se basent sur un système d'"activités" et une activité correspond à une fenêtre visible par l'utilisateur. L'application en possède trois : La fenêtre d'accueil (MainActivity), la fenêtre d'affichage des données de localisation du drone (TrackActivity) et celle des paramètres (ParamsActivity).



La fenêtre ou écran d'accueil ne comprends qu'un bouton permettant de lancer l'activité de localisation et donc le client web et un bouton pour quitter l'application, accompagnés du logo du projet. Un menu déroulant a été implémenté de façon à pouvoir accéder aux paramètres.

FIGURE 6.2 – Écran d'accueil



La fenêtre ou écran de localisation est un peu plus complexe. Pour l'utilisateur, les trois éléments clef de l'interface sont : les informations concernant l'état du système, le bouton "rafraîchir" et le bouton "retour". Un indicateur visuel est aussi présent en cas de détection d'intrusion.

FIGURE 6.3 – Écran de localisation



La fenêtre de paramètres permet de modifier les paramètres de connexion du client comme l'adresse et le port du serveur visé.

FIGURE 6.4 – Écran de localisation

Détails concernant la fenêtre de localisation

La fenêtre de localisation comporte une particularité essentielle. En effet, en même temps que celle ci s'affiche, le client web (classe Client) est lancé en arrière plan grâce à une "AsyncTask" (Tâche asynchrone). L'intérêt d'un tel mode de fonctionnement présente l'avantage de ne pas avoir à créer de "Thread" (Fil d'exécution) supplémentaire et de découpler le serveur de l'affichage. Ainsi, en cas d'erreur de connexion ou de problème irrécupérable, l'interface continue de répondre et permet à l'utilisateur de relancer la connexion. Le système Android gèrera en parallèle l'affichage et le serveur, le fonctionnement des AsyncTask permet de s'assurer que tant que la fenêtre de l'application est visible par l'utilisateur le client fonctionnera en arrière plan.

Troisième partie

Organisation d'équipe

Organisation du travail

7.1 Méthode de travail

Au cours de ce semestre nous avons défini quatre grands axes de travail.

- Assembler le radiogoniomètre,
- Concevoir l'interface Web,
- Implémenter l'application Android,
- Et réaliser les tests unitaires et tests d'intégration.

Nous nous sommes penché en priorité sur l'assemblage du radiogoniomètre. Mais voyant que les différentes pièces n'arrivaient pas dans les temps nous nous sommes réparti le travail. Les différentes tâches que nous avons effectué sont affichable sur notre framaboard.

7.2 Outils utilisés

Au début de notre projet nous avons choisi d'utiliser plusieurs outils de travail en collaboration.

- Nous utilisons L^AT_EX pour la rédaction de nos rapports.
- Nous avons hébergé notre projet sur GitHub. Cela nous permet de travailler de manière collaborative avec un versionning.
- Enfin nous utilisons un Framaboard du groupe Framasoft pour gérer les tâches de notre projet.

Framaboard

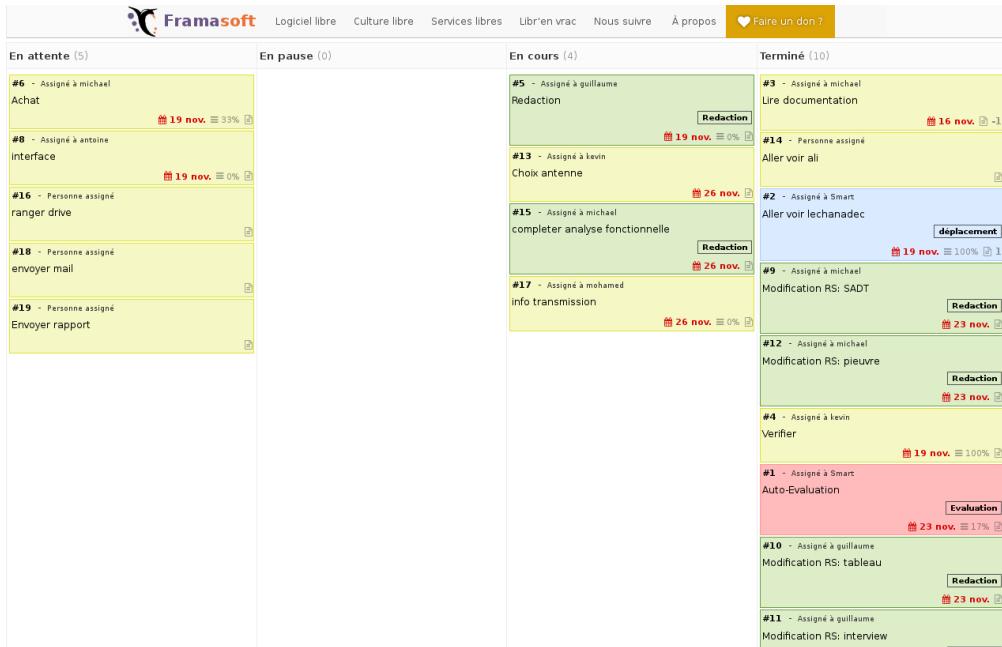


FIGURE 7.1 – Impression d’écran de notre Framaboard

Il est possible d’avoir accès en lecture à notre page Framaboard en cliquant *ici*

GitHub

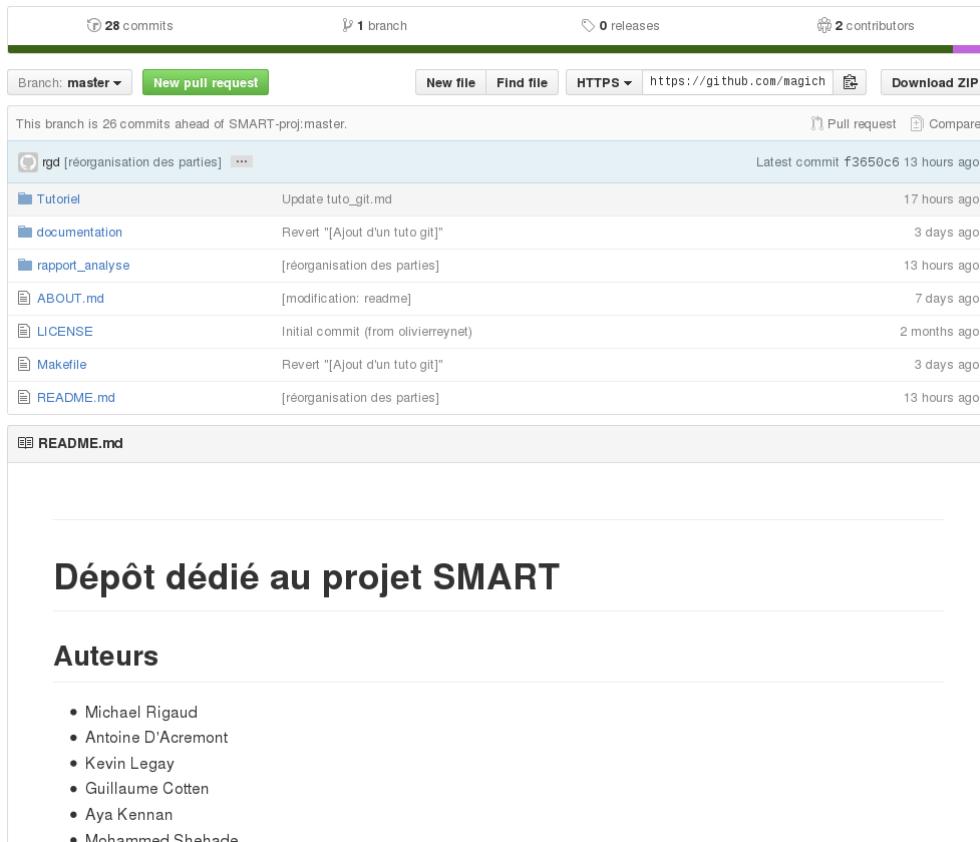


FIGURE 7.2 – Impression d’écran de notre GitHub

Il est possible d’avoir accès à notre page GitHub en cliquant *ici*

7.3 Diagramme de Grant

Le logiciel que nous avons utilisé durant notre projet permet de réaliser des TODO List, mais également d'afficher les tâches enregistrées à travers un diagramme de Grant.

Nous avons au début du semestre 4 réalisé un diagramme de Grant de Prévision des tâches principales à réaliser (figure 7.3).

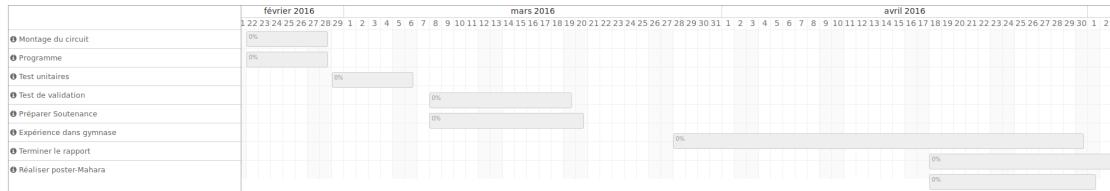


FIGURE 7.3 – Diagramme de Grant des prévision

A la fin du semestre nous obtenons le diagramme 7.4.

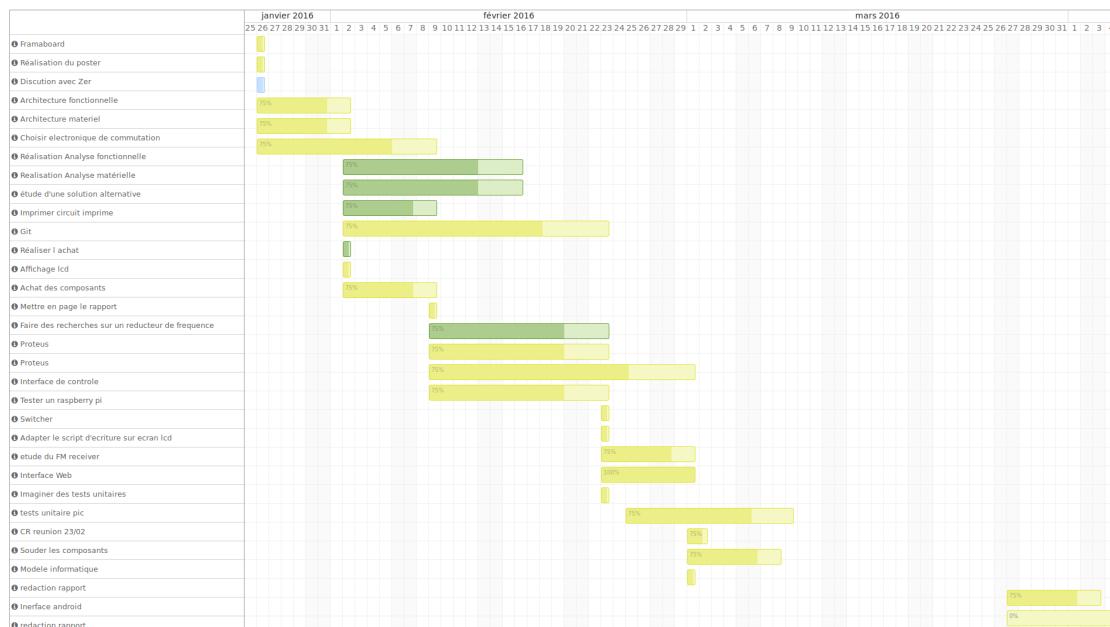


FIGURE 7.4 – Diagramme de Grant (1)

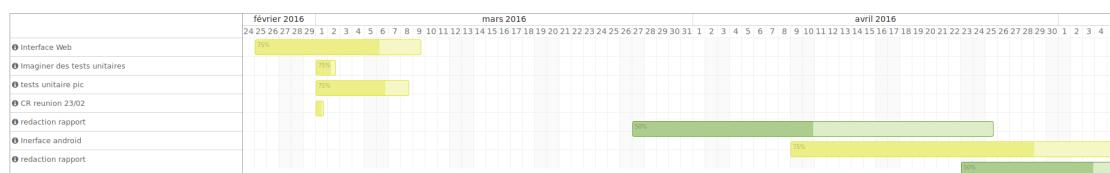


FIGURE 7.5 – Diagramme de Grant (2)

Il est possible de constater qu'il y a une certaine chute d'activités au mois de mars. Une explication sera donnée dans le chapitre suivant.

De plus, on peut constater que certains objectifs n'ont pas été tenus. En effet, nous espérions pouvoir réaliser des tests de fonctionnement dès le mois de mars, or à cause de retards de livraison nous n'avons pas pu réaliser un modèle testable. À cause de ces difficultés¹ nous avons du repenser notre projet et nos prévisions ce qui a entraîné du retard dans notre travail.

1. Elles seront détaillées au chapitre 8

Difficultés rencontrées

Durant ce semestre, de nombreux obstacles à la réalisation de notre projet ont été rencontré. Ces difficultés sont de sources et de natures différentes, et ont conduit à une évolution des attentes et des objectifs de l'équipe pour atteindre le résultat actuel.

Les premières difficultés sont de natures académiques et intellectuelles. En effet, nous avions pour objectif de réaliser notre propre radiogoniomètre à effet Doppler, couvrant le 2.4 GHz ambition réalisable mais complexe dans le temps imparti. La réalisation d'un tel système implique une solide maîtrise en électronique et théorie des ondes, qualité que nous aurions pu acquérir avec plus de délais. De l'implémentation numérique de composants absents du logiciel Proteus, en passant par la réalisation du schéma électrique puis du circuit imprimé, nos notions en électroniques étaient trop sommaires pour se lancer dans une telle réalisation. Cela demeurait cependant la seule solution pour obtenir un système complet en fin d'année, le prix d'un tel radiogoniomètre avoisinant les 2000€ sur le marché. La complexité de ce système a été accentuée par des retards de livraisons et de commande. Si certains de ces retards sont entièrement dus à notre innocence dans la conduite d'un projet, d'autres incombent uniquement aux fournisseurs. En effet, la commande de nos antennes accuse plus d'un mois de retard et devrait donc être annulé. Le dialogue avec le fournisseur n'y changeant rien, ce retard nous a d'autant plus conduit à faire évoluer nos attentes et nous recentrer sur la deuxième partie de notre projet, la gestion de la détection. D'une détection réelle, nous avons donc pris la décision de présenter une détection simulée, décision approuvée lors de la soutenance intermédiaire.

Dans un second temps, les difficultés furent de natures humaines et relationnelles. Chaque groupe possède son élément moteur. Or, suite à des problèmes de santé, Michael RIGAUD a dû se retirer temporairement du groupe suite à sa période de convalescence. Cette période, démarrant en même temps que nos évolutions d'objectifs, a donc vu notre productivité et notre motivation diminuer à cause du retard accumulé sur la réalisation des goniomètres pour au final en abandonner la réalisation. Cette période aurait à l'inverse dû être un second départ pour l'ensemble du groupe afin de se relancer. Toutes ces difficultés rencontrées nous auront ouvert les yeux sur les obstacles à la réalisation d'un projet, obstacles auxquels nous pourront dorénavant nous préparer à l'avenir.

Conclusion

Remerciements

Avant de commencer la présentation de notre travail, nous profitons de l'occasion pour adresser nos remerciements à toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce projet.

Nous tenons à exprimer nos vifs remerciements pour notre respectueux professeur, M. Mansour Ali, d'avoir accepté de nous encadrer, suivre notre travail, nous diriger, afin que nous puissions mener ce projet à terme, ainsi que pour son soutien, ses remarques pertinentes et son encouragement.

Nos remerciements vont aussi à M. Le Chenadec Gilles , qui nous a accompagné de près durant tout ce travail, pour sa disponibilité, pour la confiance qu'il a su nous accorder et les conseils précieux qu'il nous a prodigués tout au long de la réalisation de ce projet.

Nos remerciements vont aussi à tous les professeurs, enseignants et toutes les personnes qui nous ont soutenus jusqu'au bout, et qui n'ont pas cessé de nous donner des conseils très importants en signe de reconnaissance. Nous souhaitons que le travail réalisé soit à la hauteur de leurs espérances ainsi qu'aux attentes de notre encadrant.



Annexe

Validation Fonctionnel Usine

| | |
|----------------------|---|
| Date | 17 mai 2016 |
| Lieu | Ensta Bretagne |
| Nom | Beaudoin |
| Prénom | Maxime |
| Écart constaté | <p>Il m'a été montré durant cette démonstration l'interface de SMART. J'ai pu constater l'absence des capteurs qui doivent détecter les drones. Cette absence a malheureusement était pénalisant pour la démonstration. Néanmoins l'idée est très intéressante et le travail qui a été effectué m'a convaincu quant à la réalisation du projet. J'ai également pu noter des points perfectibles, tel que :</p> <ul style="list-style-type: none"> — l'actualisation automatique dans l'application android, — la détection des altitudes des drones, — la détection de plusieurs drones, — l'affichage de plus d'information dans l'interface web. |
| Correction envisagée | <p>« <i>L'actualisation automatique dans l'application android</i> »</p> <p>Il est possible de réaliser une actualisation automatique, mais pour cela nous devrions utiliser le service google cloud messenger qui est payant. Pour des questions budgétaires nous avons choisi d'avoir pour le moment une actualisation manuelle.</p> <p>« <i>La détection de altitudes</i> »</p> <p>Nous avons envisagé de corriger ce problème en plaçant certains capteur de manière horizontal. Ainsi nous aurions leur altitude.</p> <p>« <i>Détection de plusieurs drones</i> »</p> <p>Pour régler ce problème il nous faudrait repenser la méthode de calcul de la position du drones.</p> <p>« <i>Afficher plusieurs information dans l'interface</i> »</p> <p>Nous avons déjà envisagé une amélioration dans ce sens, mais elle nous semblait pas essentiel dans le cadre de ce projet.</p> |

Validation Fonctionnelle

Documentation Technique du Raspberry Pi B+

C.1 Définition

« Le Raspberry Pi est un nano-ordinateur monocarte à processeur ARM conçu par le créateur de jeux vidéo David Braben, dans le cadre de sa fondation Raspberry Pi.

Cet ordinateur, qui a la taille d'une carte de crédit, est destiné à encourager l'apprentissage de la programmation informatique²; il permet l'exécution de plusieurs variantes du système d'exploitation libre GNU/Linux et des logiciels compatibles. Il est fourni nu (carte mère seule, sans boîtier, alimentation, clavier, souris ni écran) dans l'objectif de diminuer les coûts et de permettre l'utilisation de matériel de récupération.

Son prix de vente était estimé à 25 \$, soit 19,09 €, début mai 2011. Les premiers exemplaires ont été mis en vente le 29 février 2012 pour environ 25 €. Début 2015, plus de cinq millions de Raspberry Pi ont été vendus. De multiples versions ont été développées (voir la liste ci-dessous), on trouve les dernières à un peu plus de 25 € pour le B+, à un peu plus de 30 € pour le Pi 2 (2015) et à un peu plus de 45 € pour le Pi 3 (2016) » Wikipédia [?]

C.2 Caractéristique et connectiques

| Caractéristiques | |
|-------------------|------------------------------|
| Micro-contrôleur | Broadcom BCM2835 ARM1176JZFS |
| Vitesse d'horloge | 700 MHz |
| RAM | 512 Mo |

| Connectiques | |
|--------------------------------|-----------------------|
| Port(s) USB | 4 |
| Port Ethernet / RJ45 | 1 |
| Connecteur(s) audio analogique | 1 sortie jack 3,5 mm |
| HDMI | 1 |
| Port pour carte mémoire | 1 port micro SD |
| Alimentation | via port micro USB 5V |

C.3 Schéma technique

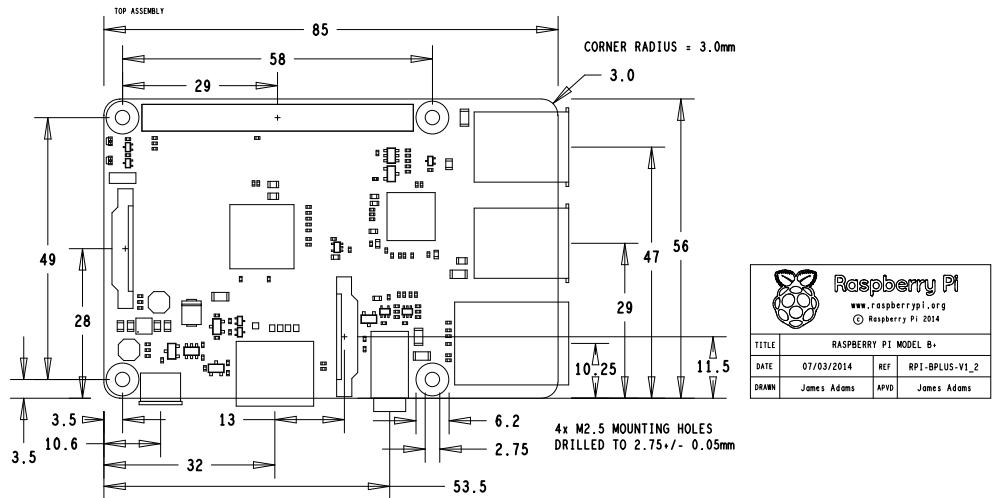


FIGURE C.1 – Dessin mecanique

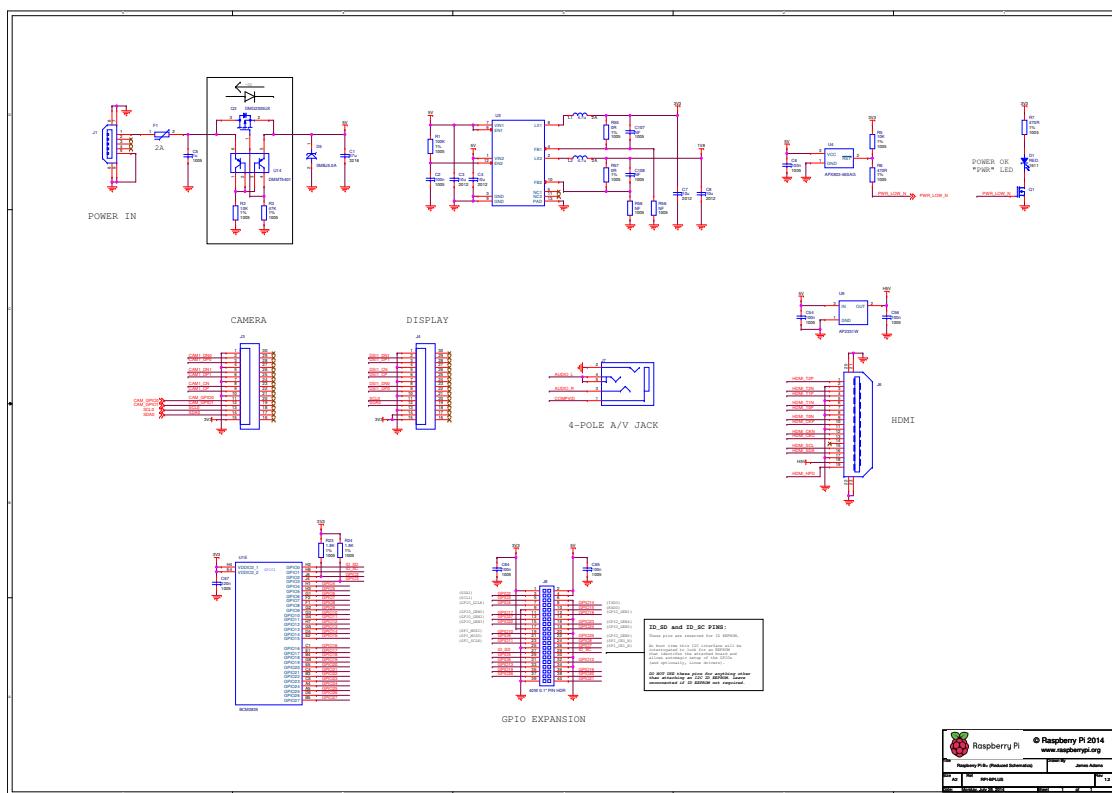


FIGURE C.2 – Schéma technique

Documentation Technique à Raspbian

D.1 Définition

« Raspbian est un système d'exploitation libre basé sur la distribution GNU/Linux Debian, et optimisé pour le plus petit ordinateur du monde, la Raspberry Pi.

Raspbian ne fournit pas simplement un système d'exploitation basique, il est aussi livré avec plus de 35 000 paquets, c'est-à-dire des logiciels pré-compilés livrés dans un format optimisé, pour une installation facile sur votre Raspberry Pi via les gestionnaires de paquets. La première version des 35 000 paquets Raspbian, optimisés pour la Raspberry Pi, a été achevée en Juin 2012.

Néanmoins, Raspbian est toujours en développement, avec une priorité à l'amélioration de la stabilité et des performances d'un maximum de paquets Debian possibles » raspbian-france[?]

D.2 Logo



FIGURE D.1 – logo raspbian

Client.py

```

1 #!/usr/bin/python3
2 ##### SERVEUR.py #####
3 # auteur: Equipe Smart
4 # date: 5 avril
5 #
6 # Description: cette application sert de client
7 #####
8
9 import socket, sys
10 from script.led.SimulLed import SimulLed
11
12 # Adresse de connexion
13 HOST = '127.0.0.1'
14 # Port de connexion
15 PORT = 50000
16 # Nom du RPI se connectant
17 NOM = "RP1"
18
19
20 mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
21
22 # Realisaion de la connection
23 try:
24     mySocket.connect((HOST, PORT))
25 except socket.error:
26     print("La connexion a echoue.")
27     sys.exit()
28 print("Connexion etablie avec le serveur.")
29
30 # identification du RPI
31 message = str.encode("%s" % (NOM))
32 mySocket.send(message)
33
34 msgServeur = bytes.decode(mySocket.recv(1024))
35 print(msgServeur)
36
37
38 led = ""
39 ledHisto = ""
40 # Permet de simuler l'acquisition as port GPIO
41 # Il suffit de changer cette fonction pour que le code soit adapter
42 # au radiogoniometre branche au port GPIO
43 sim = SimulLed(NOM)
44

```

```
45 while 1:  
46     try:  
47         led = sim.getLed() # recuperation de l'etat des led a cet  
        instant  
48         if led!=ledHisto:  
49             print("la led allume:", led)  
50             mySocket.send(str.encode(led))  
51             ledHisto = led  
52     except KeyboardInterrupt:  
53         # fin de la socket sur une interruption de type Ctrl-C  
54         mySocket.send(str.encode("FIN"))  
55         print("Connexion interrompue.")  
56         mySocket.close()  
57         break
```

Serveur.py

```
1 #!/usr/bin/python3
2 ##### SERVEUR.py #####
3 # auteur: Equipe Smart
4 # date: 5 avril
5 #
6 # Description: cette application sert de serveur avec multi thread
7 #####
8
9 # Adresse de connexion
10 HOST = '127.0.0.1'
11 # Port de Connexion
12 PORT = 50000
13
14
15 import socket, sys, threading
16 import pickle
17
18
19
20 class ThreadClient(threading.Thread):
21     '''objet thread servan a gerer les connexions clients'''
22     def __init__(self, conn, nom):
23         threading.Thread.__init__(self)
24         self.connexion = conn # contient le nom de la connexion
25         self.name = nom # le nom du rpi client
26
27     def run(self):
28         while 1:
29             msgClient = bytes.decode(self.connexion.recv(1024))
30             if msgClient.upper() == "FIN" or msgClient == "":
31                 break
32             message = "%s> %s" % (self.name, msgClient)
33             print(message)
34             direction = "donc la direction %s" % (DIRECTION[msgClient])
35             print(direction)
36             POSITION[self.name]=POSITION[self.name][0],POSITION[self.
37                 name][1],POSITION[self.name][2],DIRECTION[msgClient]
38             ecriture()
39             # Fermeture de la connexion :
40             self.connexion.close() # couper la connexion cote serveur
41             del conn_client[self.name] # supprimer son entree dans le
42                 dictionnaire
43             print("Client %s deconnecte." % self.name)
44             # Le thread se termine ici
```

```

43
44
45 def initialisation():
46     '''Initialisation des donnees'''
47     with open('donnees', 'rb') as fichier:
48         mon_depickler = pickle.Unpickler(fichier)
49         DIRECTION = mon_depickler.load()
50     lecture()
51
52
53 def lecture():
54     ''' Lecture du fichier position contenant la position de rpi avec
55         leur angle de detection'''
56     fichier = open("position", 'r')
57     for line in fichier:
58         contenue = line.split()
59         POSITION[contenue[0]] = contenue[2],contenue[3],contenue[4],
60         contenue[5]
61     fichier.close()
62
63 def ecriture():
64     ''' Ecriture dans le fichier position de le position des rpi avec
65         leur angles de detection'''
66     fichier = open("position", 'w')
67     for clef in POSITION:
68         message = "%s : %s %s %s\n" %(clef,POSITION[clef][0],
69             POSITION[clef][1],POSITION[clef][2], POSITION[clef][3])
70     fichier.write(message)
71     fichier.close()
72
73 # Cette initialisation permet d'initialiser le dictionnaire POSITION
74 # qui contiennent toute les variables
75 initialisation()
76
77 DIRECTION = {"led1": 0, "led2": 10,"led3": 20,"led4": 30,"led5": 40,
78 "led6": 50,"led7": 60,"led8": 70,"led9": 80,"led10": 90,"led11": 100,
79 "led12": 110,"led13": 120,"led14": 130,"led15": 140,"led16": 150,
80 "led17": 160,"led18": 170,"led19": 180,"led20": 190,"led21": 200,
81 "led22": 210,"led23": 220,"led24": 230,"led25": 240,"led26": 250,
82 "led27": 260,"led28": 270,"led29": 280,"led30": 290,"led31": 300,
83 "led32": 310,"led33": 320,"led34": 330,"led35": 340,"led36": 350,
84 "ledNULL":360}
85
86 # Initialisation du serveur - Mise en place du socket :
87 mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
88 try:
89     mySocket.bind((HOST, PORT))
90 except socket.error:
91     print("La liaison du socket a l'adresse choisie a echoue.")
92     sys.exit()
93 print("Serveur pret, en attente de requetes ...")
94 mySocket.listen(5)
95
96 # Attente et prise en charge des connexions demandees par les
97 # clients :

```

```
91 conn_client = {}           # dictionnaire des clients connectes
92 while 1:
93     try:
94         connexion, adresse = mySocket.accept()
95         # Identification du client avec d'ouvrir le thread
96         nom = bytes.decode(connexion.recv(1024))
97         th = ThreadClient(connexion, nom)
98         # ouverture du thread
99         th.start()
100        conn_client[nom] = connexion
101        print("Client %s connecte, adresse IP %s, port %s." %\
102              (nom, adresse[0], adresse[1]))
103        connexion.send(b'Vous etes connecte. Envoyez vos messages.')
104    except KeyboardInterrupt:
105        print("Fin du serveur")
106        break
```

Table des figures

| | | |
|-----|---|----|
| 1.1 | Installation de Smart | 7 |
| 1.2 | Architecture Physique | 7 |
| 1.3 | Cercle de détection | 8 |
| 1.4 | Maillage de détection | 8 |
| 2.1 | 3 PIC programmés | 9 |
| 2.2 | Schéma électrique du test unitaire | 10 |
| 2.3 | Schéma de montage du pic sur le Montréal 3v2 | 11 |
| 2.4 | Comportement du filtre | 12 |
| 2.5 | S11 du filtre passe bande | 12 |
| 2.6 | Fréquences générées par le VCO alimenté à 8.1V centrée autour de 1.9GHz d'une largeur d'environ 10MHz | 13 |
| 2.7 | Montage de test de l'adaptateur | 14 |
| 2.8 | Photo du VCO et de son alimentation | 14 |
| 3.1 | Notre Raspberry Pi B+ | 16 |
| 3.2 | Allumage d'une LED par Raspberry Pi | 17 |
| 3.3 | Méthode de connexion des leds dans le Montréal | 18 |
| 3.4 | Système modélisant une LED | 18 |
| 3.5 | Premier test : communication | 19 |
| 3.6 | Second test : erreurs | 19 |
| 3.7 | Message d'erreur sur l'application | 20 |
| 4.1 | schéma de fonctionnement d'un mixer | 22 |
| 4.2 | principe du fonctionnement d'un mixer | 23 |
| 4.3 | le down-converter et le filtre passe bande | 23 |
| 5.1 | Cas d'utilisation de l'interface | 25 |
| 5.2 | Diagramme de classe | 25 |
| 5.3 | triangulation | 27 |
| 5.4 | Interface Web | 28 |
| 6.1 | Diagramme de classe de l'application | 30 |
| 6.2 | Écran d'accueil | 31 |
| 6.3 | Écran de localisation | 31 |
| 6.4 | Écran de localisation | 31 |
| 7.1 | Impression d'écran de notre Framaboard | 35 |
| 7.2 | Impression d'écran de notre GitHub | 35 |
| 7.3 | Diagramme de Grant des prévision | 36 |

| | | |
|-----|----------------------------------|----|
| 7.4 | Diagramme de Grant (1) | 36 |
| 7.5 | Diagramme de Grant (2) | 36 |
| C.1 | Dessin mecanique | 44 |
| C.2 | Schéma technique | 44 |
| D.1 | logo raspbian | 45 |