



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	2
Introduction	3
I Préparation	4
1 Point de Situation	5
1.1 Contexte	5
1.2 Rappel de notre projet	5
1.3 Architecture Fonctionnelle	6
1.4 Architecture physique	6
1.5 Treillis de détection	7
2 Tests unitaires	8
2.1 Raspberry Pi	8
2.2 PIC	10
2.3 Filtre passe bande	12
2.4 VCO	14
2.5 Down converter	15
3 Test d'intégration	17
II Réalisation	18
4 Radiogoniometre	19
4.1 Down converter	19
5 Interface Web	21
5.1 Analyse	21
5.2 Conception	22
6 Réalisation	26
6.1 Application Android : S.M.A.R.T Comm Center	26

III Organisation d'équipe	31
7 Organisation du travail	32
7.1 Méthode de travail	32
7.2 Outils utilisés	32
7.3 Diagramme de Grant	33
8 Difficultés rencontrées	35
Conclusion	36
Remerciements	37
Annexe	39
A Documentation Technique du Raspberry Pi B+	39
B Documentation Technique à Raspbian	41
C Client.py	42
D Serveur.py	44
E Validation Fonctionnel Usine	47
Table des figures	48

Résumé

Voici le résumé

Introduction

Première partie

Préparation

Point de Situation

1.1 Contexte

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.

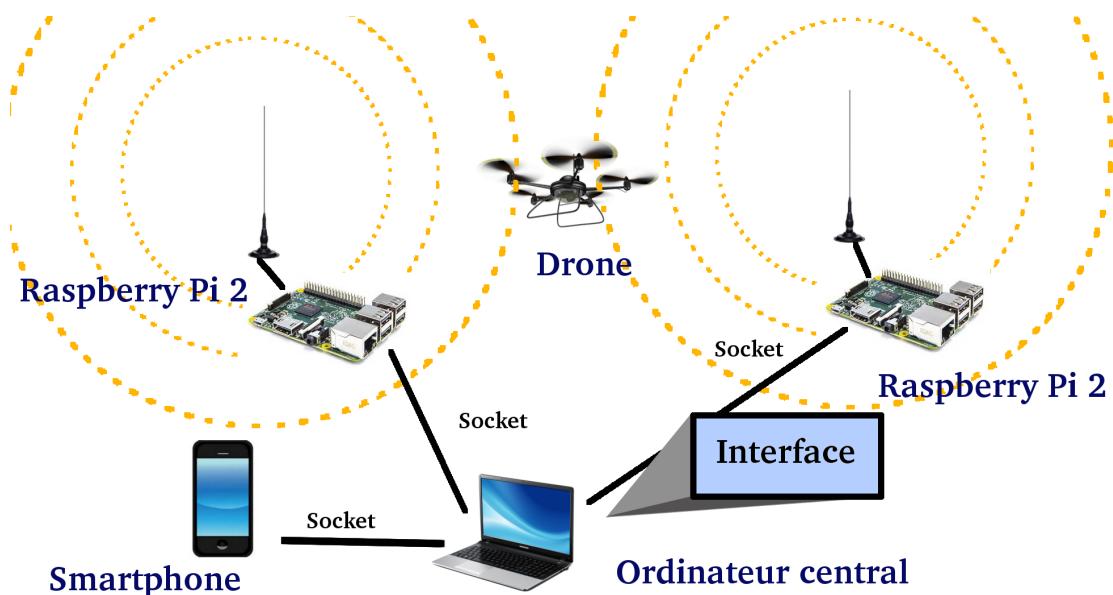


FIGURE 1.1 – Installation de Smart

1. La documentation technique du Raspberry PI est situé en annexe à la page 39

1.3 Architecture Fonctionnelle

1.4 Architecture physique

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

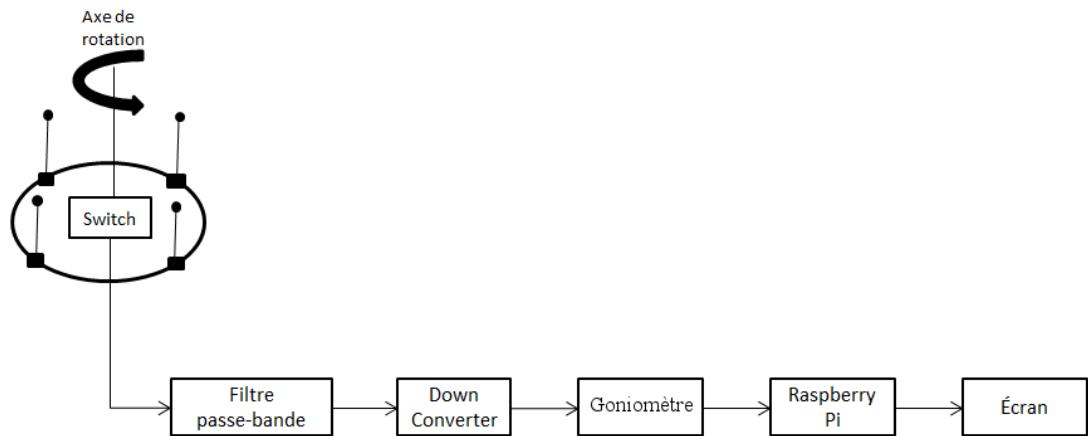


FIGURE 1.2 – Architecture Physique

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 :

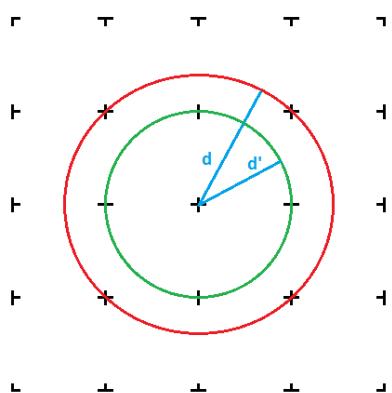
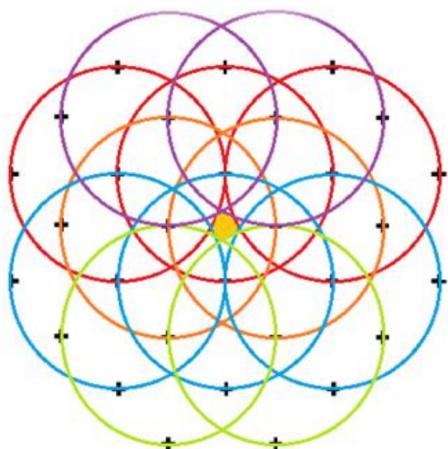


FIGURE 1.3 – Cercle de détection

- 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

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 Raspberry Pi

La documentation technique lié a notre Raspberry Pi es situé en annexe à la page 39

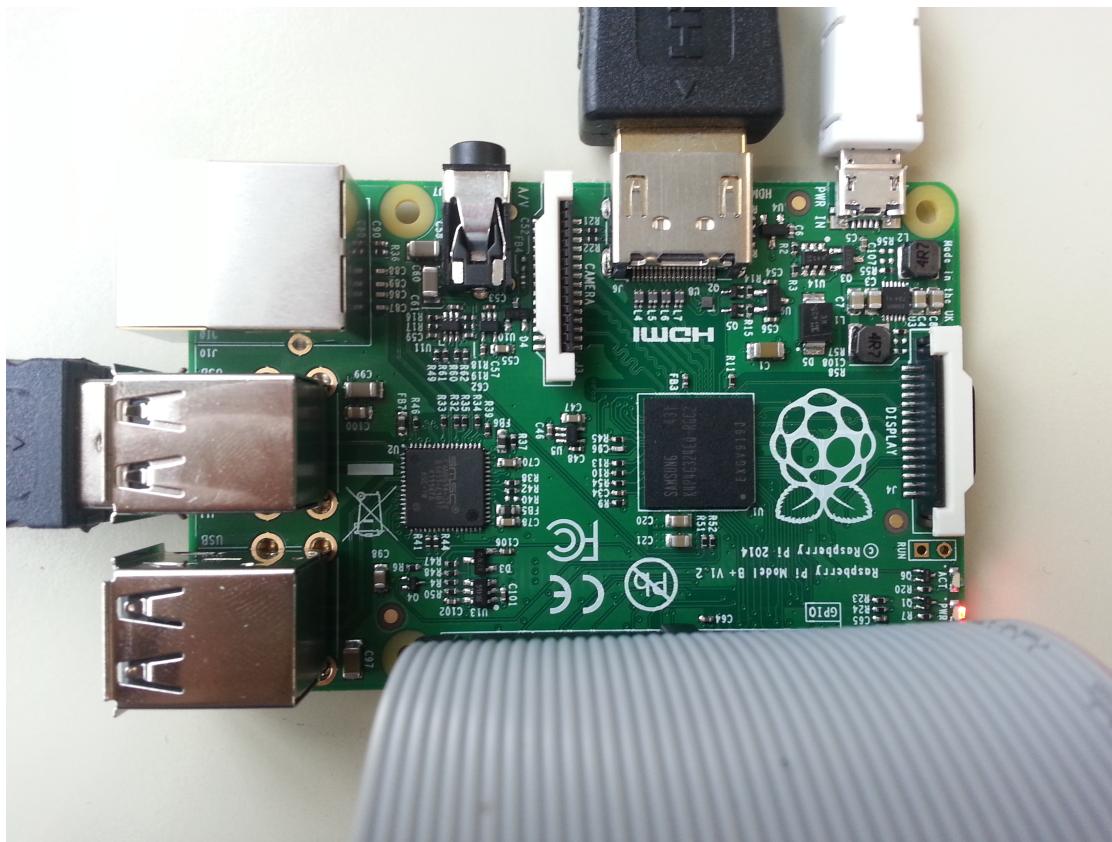


FIGURE 2.1 – Notre Raspberry Pi B+

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 unitaires.

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 2.2, 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 2.3). Nous allons donc placer le Raspberry Pi au niveau des LED pour obtenir les informations délivré par le Montréal 3v2.

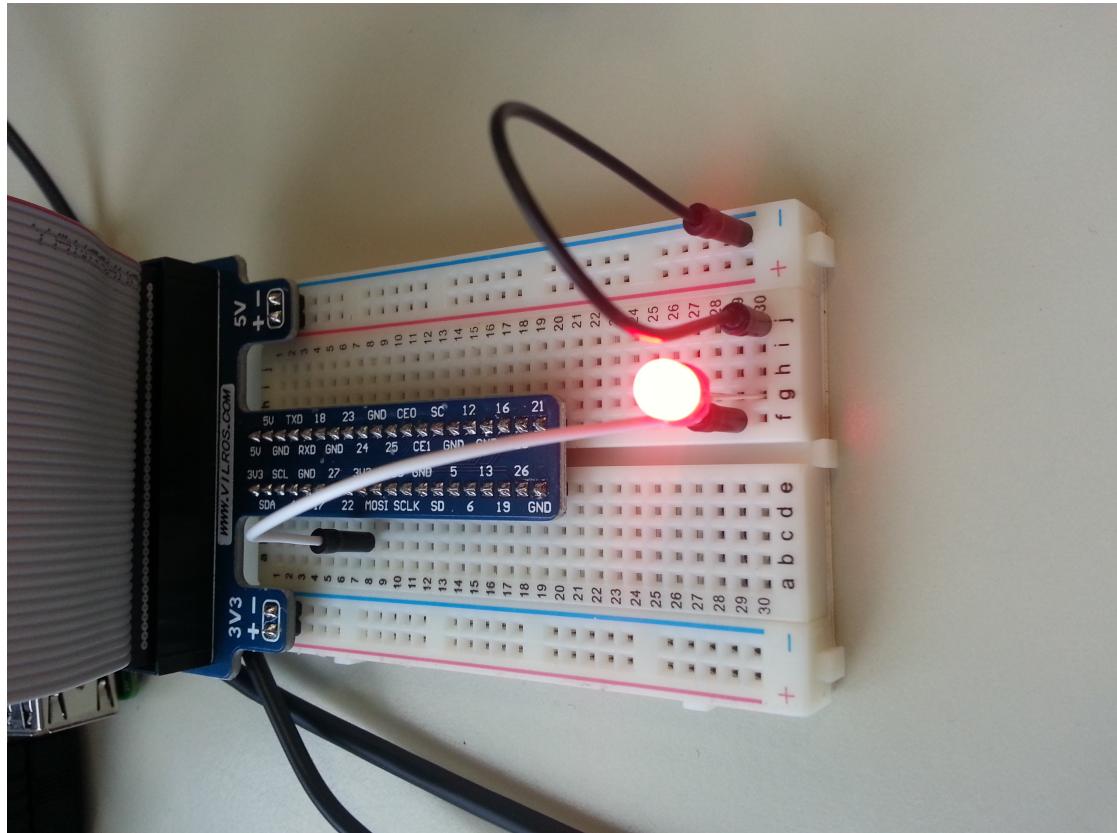


FIGURE 2.2 – Allumage d'une LED par Raspberry Pi

La sortie du Montréal 3v2 est décrite à la figure 2.3. 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 2.4). 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 tester 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 situé en annexe à la page 41

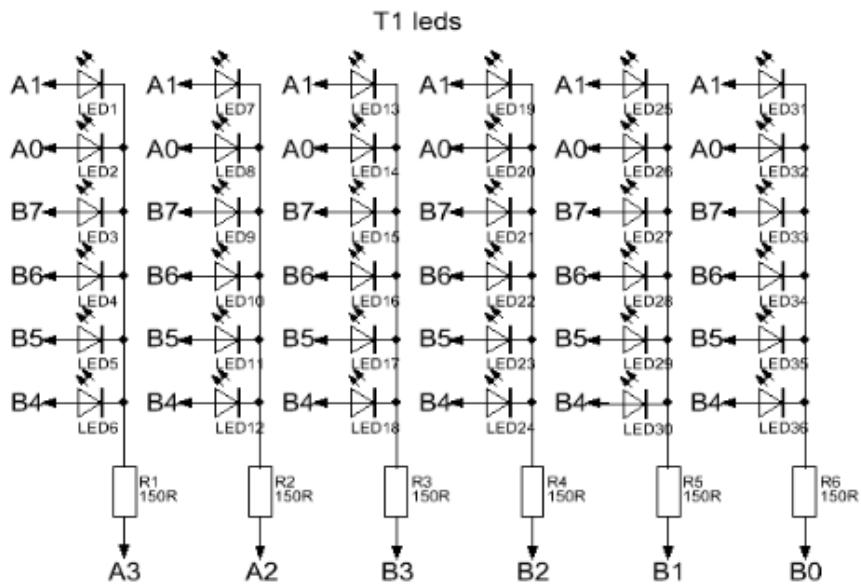


FIGURE 2.3 – Méthode de connexion des leds dans le Montréal

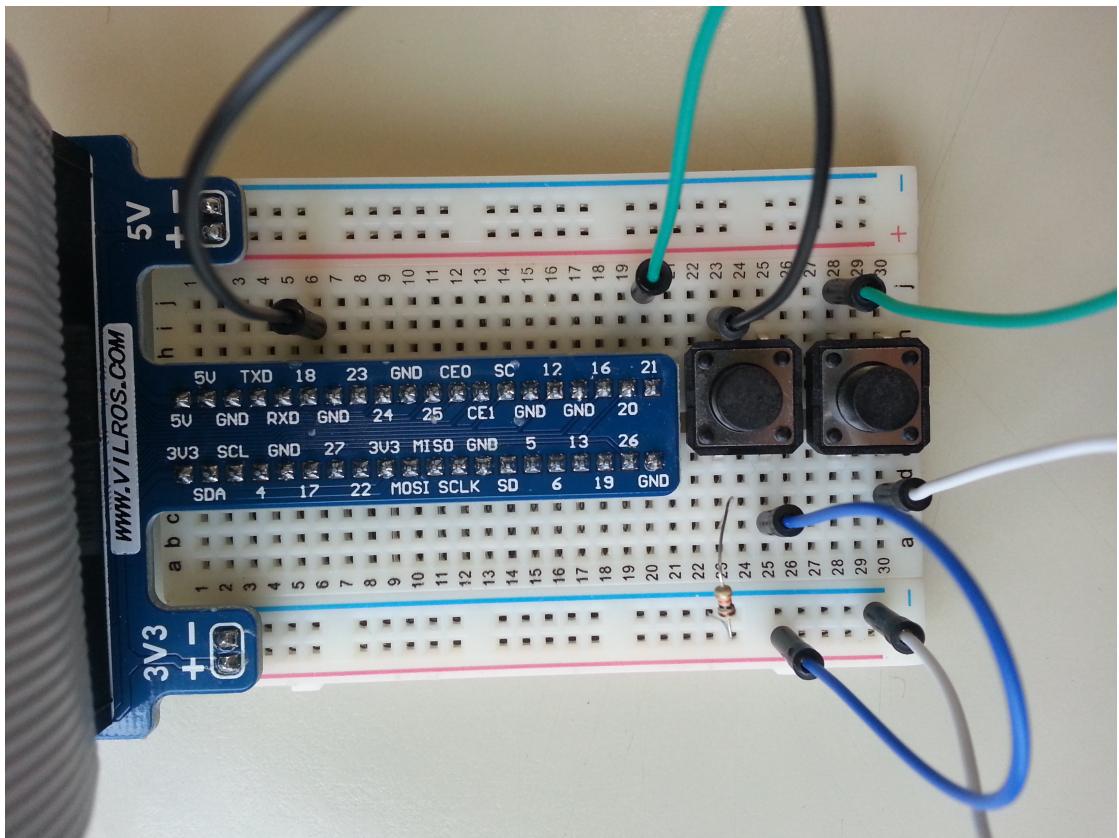


FIGURE 2.4 – Système modélisant une LED

2.2 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 [?].

Nous avons ensuite imaginé et réalisé des tests unitaires sur chacun des PIC pour

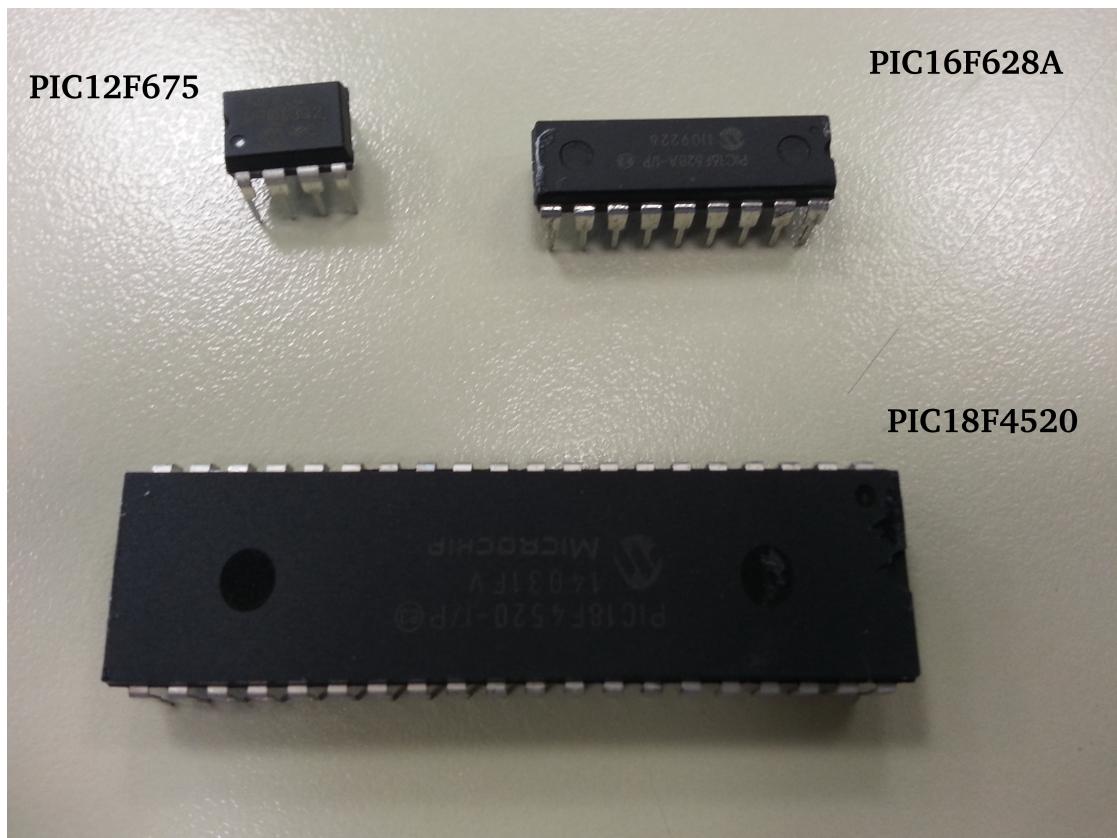


FIGURE 2.5 – 3 PIC programmés

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 réalisé le montage de la figure 2.6. On peut voir à la figure 2.7 le schéma de montage du PIC sur le Montréal 3v2.

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.

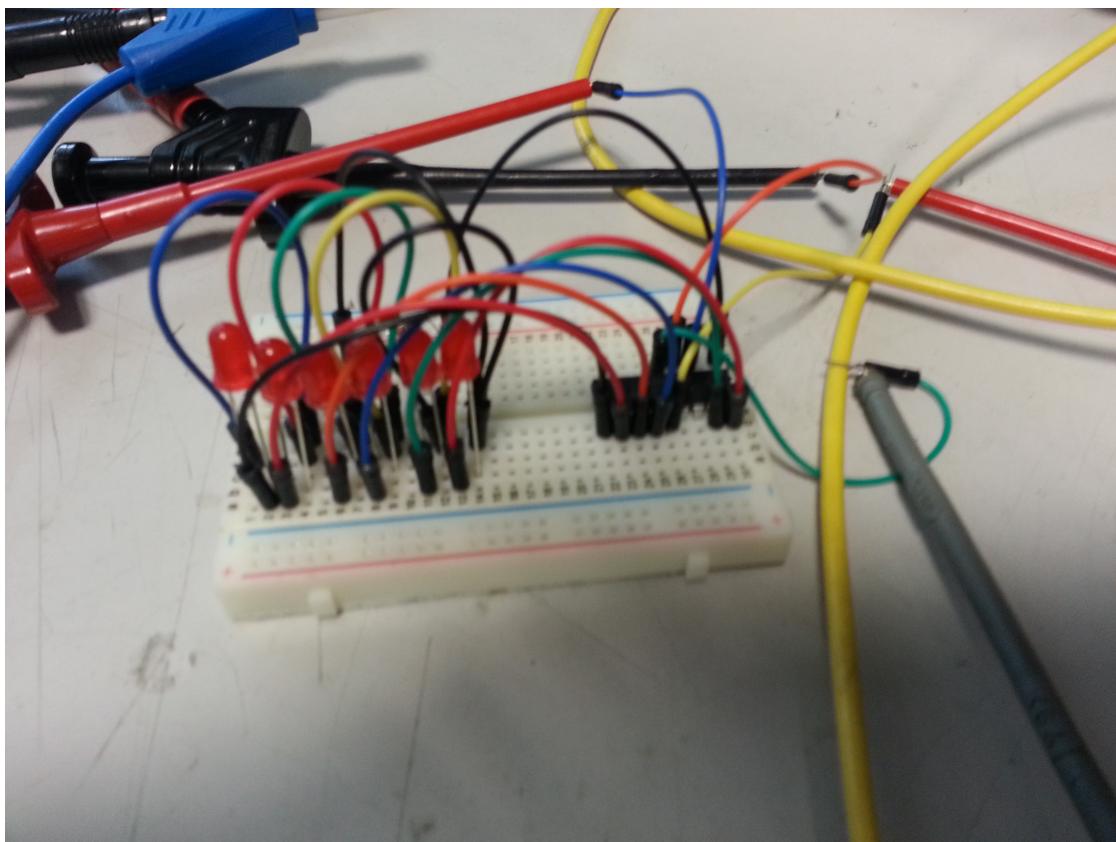


FIGURE 2.6 – Schéma électrique du test unitaire

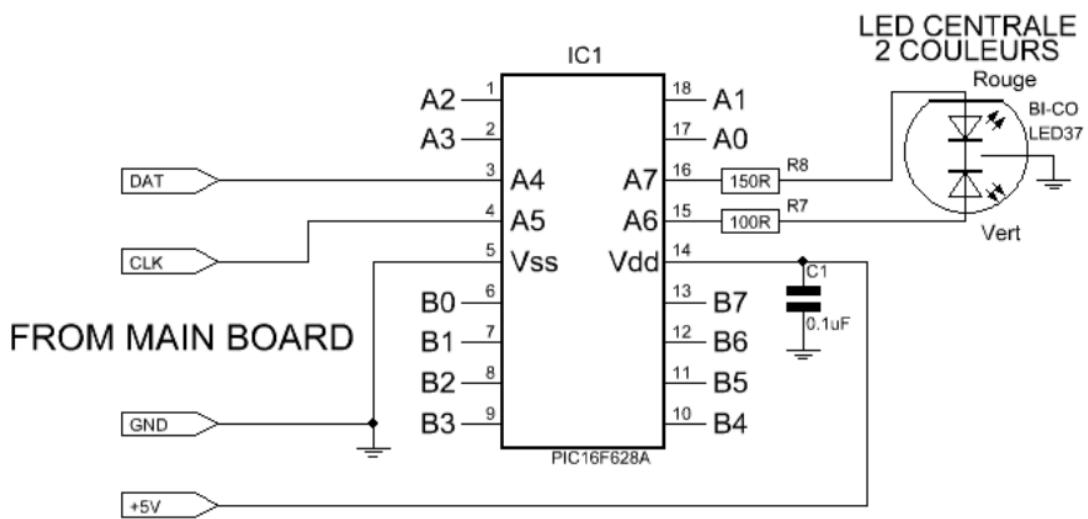


FIGURE 2.7 – Schéma de montage du pic sur le Montréal 3v2

PIC12F675

PIC18F4520

2.3 Filtre passe bande

Pour le filtre passe bande, nous souhaitions un filtre qui couperait tout ce qui se trouve en dehors de notre bande, au final ¹³ le filtre réagit plutôt bien quand le montage

qui y est lié est adapté, ce qui est le cas, le circuit fonctionne bien à 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. Mais ceci n'est pas gênant car les bandes entre 205Ghz et 5 GHz ne sont pas ou peu utilisées en France.

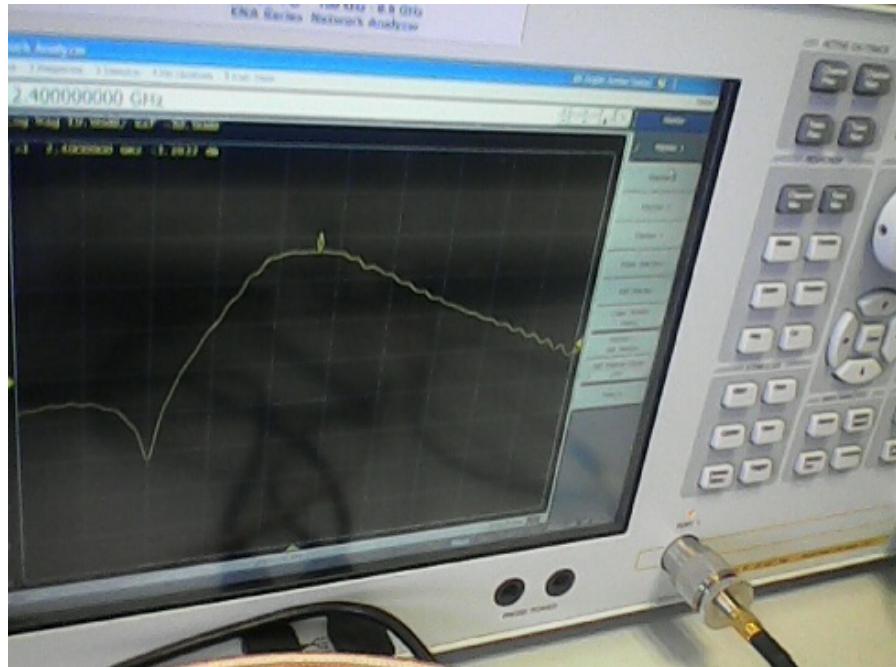


FIGURE 2.8 – Comportement du filtre

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. 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 tout 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 est bon pour le filtre.

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.

Lors du test, nous avions une perte d'environ 2dBm dans la bande de fréquence 2.4-2.5GHz, ce qui est peu, le filtre ne risque donc pas d'occulté ce que l'on souhaite voir.

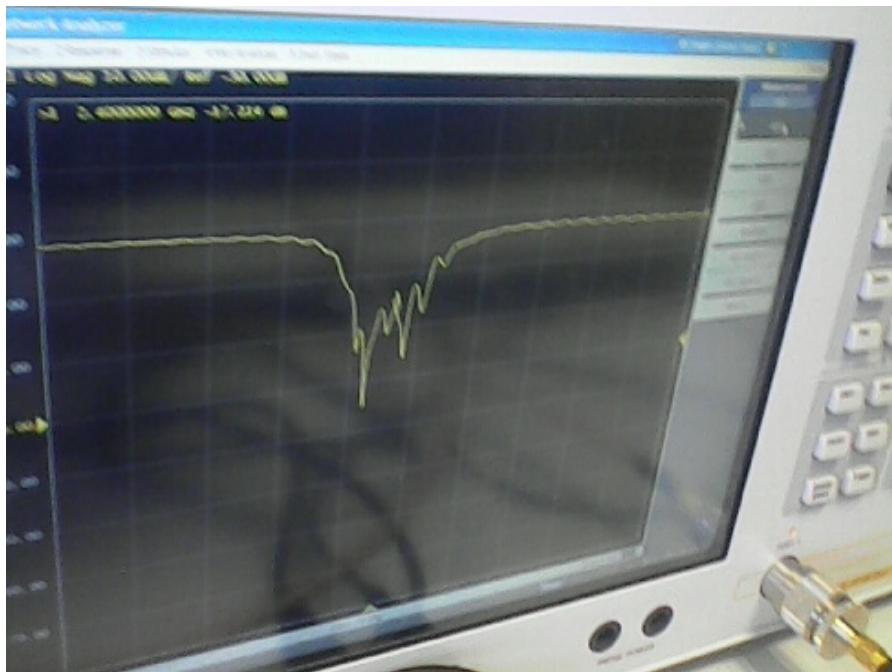


FIGURE 2.9 – S11 du filtre passe bande

2.4 VCO

Le test du VCO est simple mais doit être bien fait car sans lui impossible d'obtenir la bonne fréquence à la fin.

Nous avons commencé par mesurer la fréquence libre, c'est-à-dire la fréquence renvoyée par le VCO s'il est juste alimenté et que la tension d'entrée est nulle. La fréquence libre qui était de 1.35 GHz était plus grande que la fréquence indiquée par le constructeur (1.31Ghz) ensuite nous avons mesuré la tension pour laquelle nous obtenions 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 on a pu obtenir un régulateur à 8.1V ce qui après test donnait 1.91GHz. La bande de fréquence a 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.

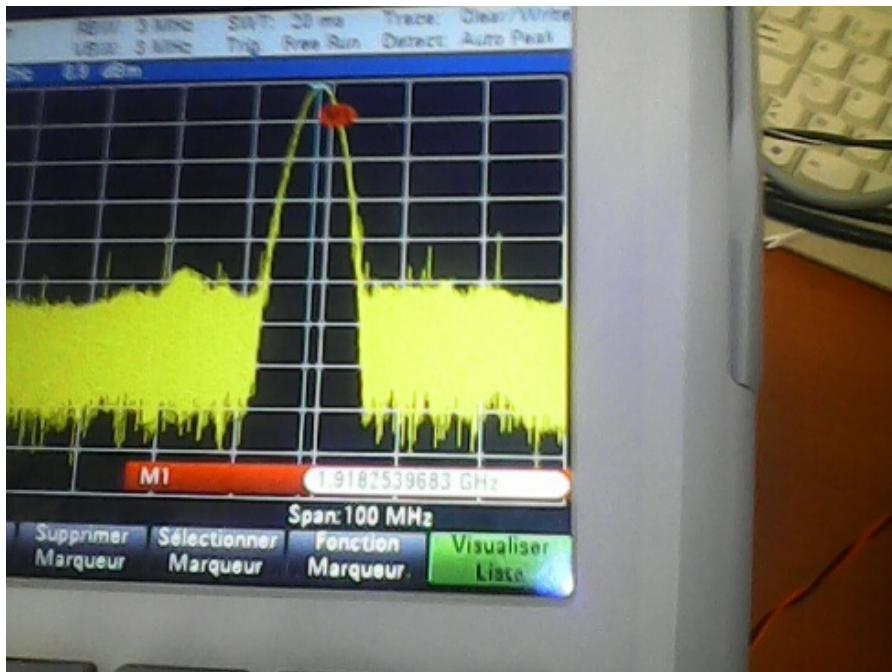


FIGURE 2.10 – Fréquences générées par le VCO alimenté à 8.1V centrée autour de 1.9GHz d'une largeur d'environ 10MHz

2.5 Down converter

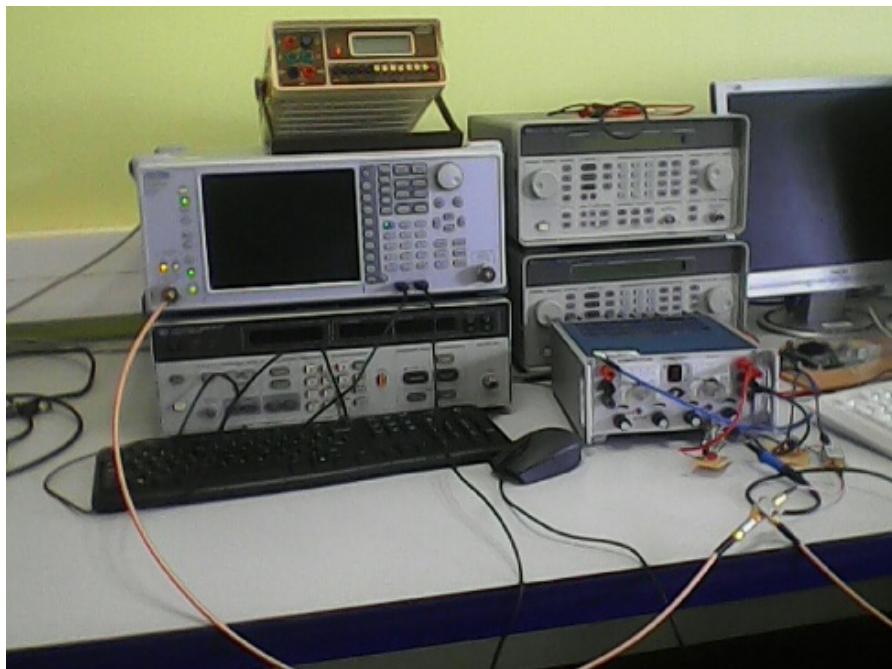


FIGURE 2.11 – Montage de test de l'adaptateur

Le test du down convertisseur a posé certain problème. 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.

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

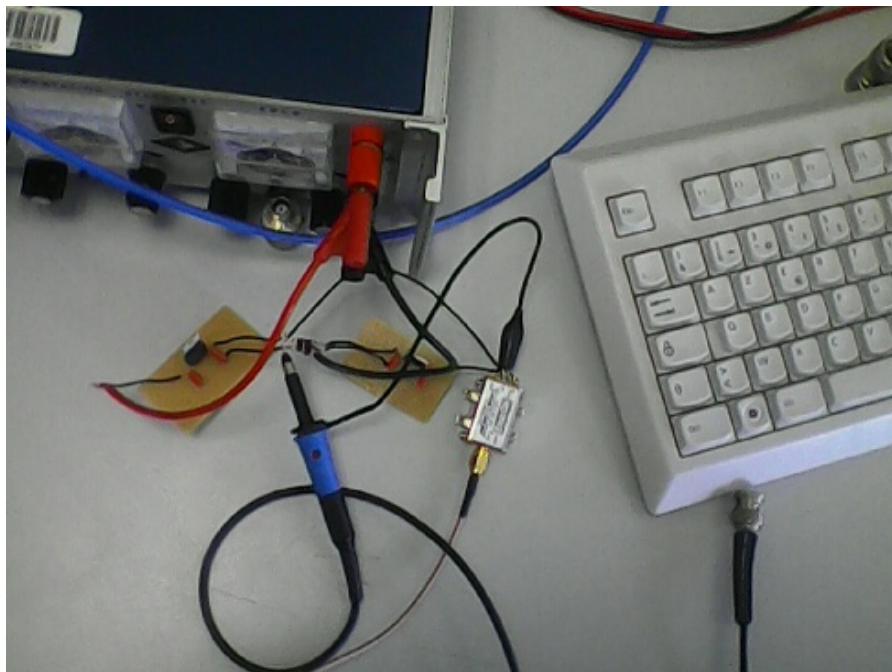


FIGURE 2.12 – Photo du VCO et de son alimentation

celui pour la tension en entrée. Le premier membre de l'équipe a demandé au second de souder le montage mais n'a pas présenté l'ordre dans lequel il devait être placé ce qui a entraîné une inversion. Ceci a retardé la phase de test

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 le griller, nous n'avons pas pu le faire fonctionner.

Enfin, par manque de temps, nous n'avons pas eu l'occasion de refaire ce test.

CHAPITRE **3**

Test d'intégration

Deuxième partie

Réalisation

Radiogoniometre

Pour réaliser notre projet nous avons été obligé d'adapter notre capteur le radiogoniomètre Montréal 3v2 a nos besoins.

4.1 Down converter

Le radiogoniomètre Montréal 3v2 fonctionne à une fréquence de 500Mhz. Il est donc impossible de l'utilisé entre 2.4Ghz et 2.5 GHz, bande de fréquence qu'utilise les drones que nous souhaitons utiliser. Nous avons donc cherché un moyen d'adapter ce radiogoniomètre aux fréquences souhaitées.

Nous avons trouvé une solution applicable à notre système.

D'abord nous avons utilisé un down-converter. Ce composant reçoit deux entrées, le signal dont on veut changer la fréquence(RF) et un signal de fréquence 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.

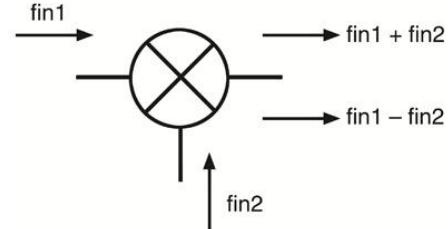


FIGURE 4.1 – schéma de fonctionnement d'un mixer

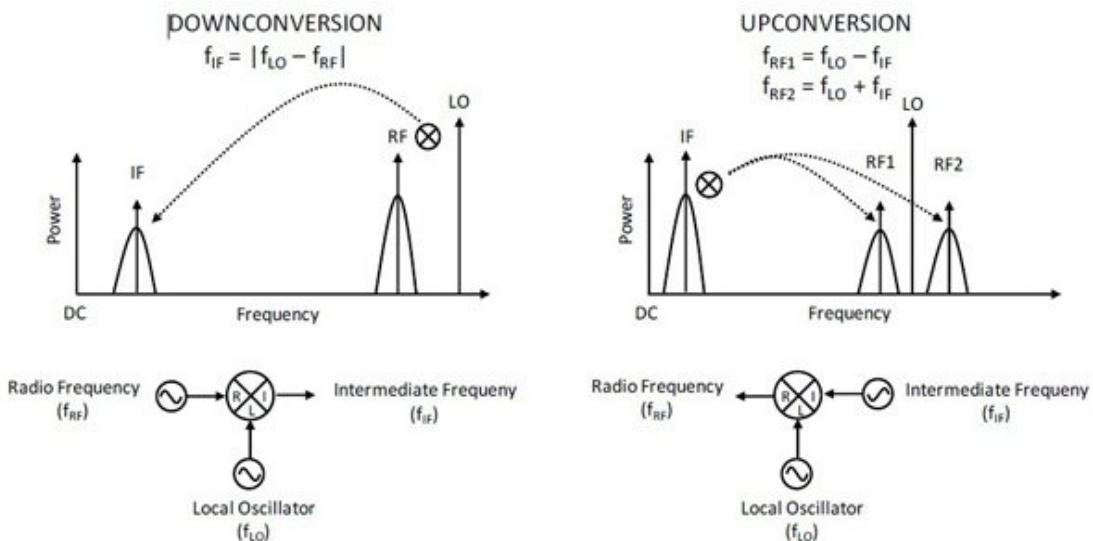


FIGURE 4.2 – principe du fonctionnement d'un mixer

Dans ce cadre on utilise le down-converter pour diminuer la fréquence. La fréquence

du signal Df est donnée par un VCO, 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 stabilisé la tension d'entrée et l'alimentation. On utilise donc un régulateur de tension qui amène une entrée stable. Il en a fallu un second 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 ne pas parasité l'effet doppler, la combinaison des deux indiquerait une mauvaise position. Le régulateur reçoit une tension V, si V est supérieur à la tension Vmax du régulateur, la tension de sortie devient Vmax.

Après il a fallu gérer du signal d'entrée du down-converter et des problèmes de bruit, on a donc placé un filtre devant le down-converter. Ce filtre est un filtre passe bande qui fonctionne autour de 2.4Ghz-2.5Ghz pour ne garder que ce qui nous intéresse.



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

A l'aide de ce montage on peut abaisser la fréquence du signal d'entrée.

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).

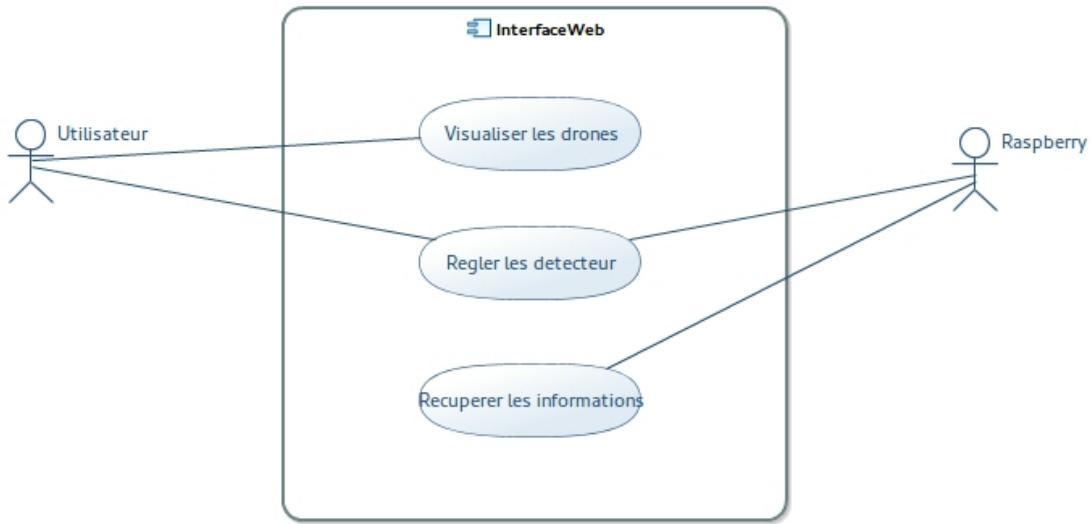


FIGURE 5.1 – Cas d'utilisation de l'interface

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.

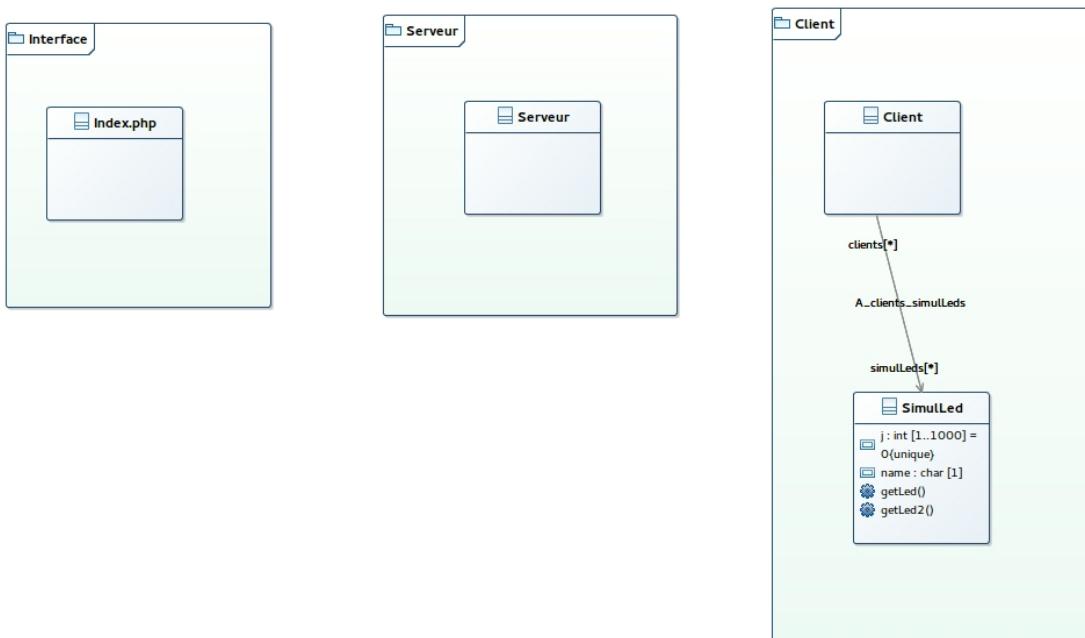


FIGURE 5.2 – Diagramme de classe

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.

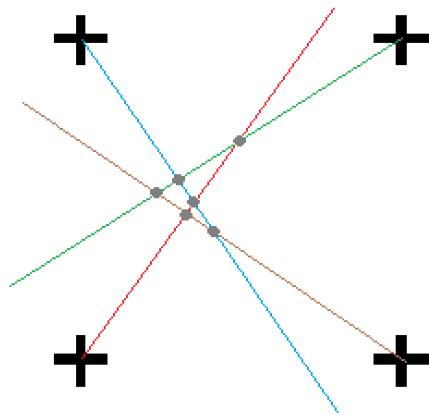


FIGURE 5.3 – triangulation

Au final, nous obtenons la page web suivante :

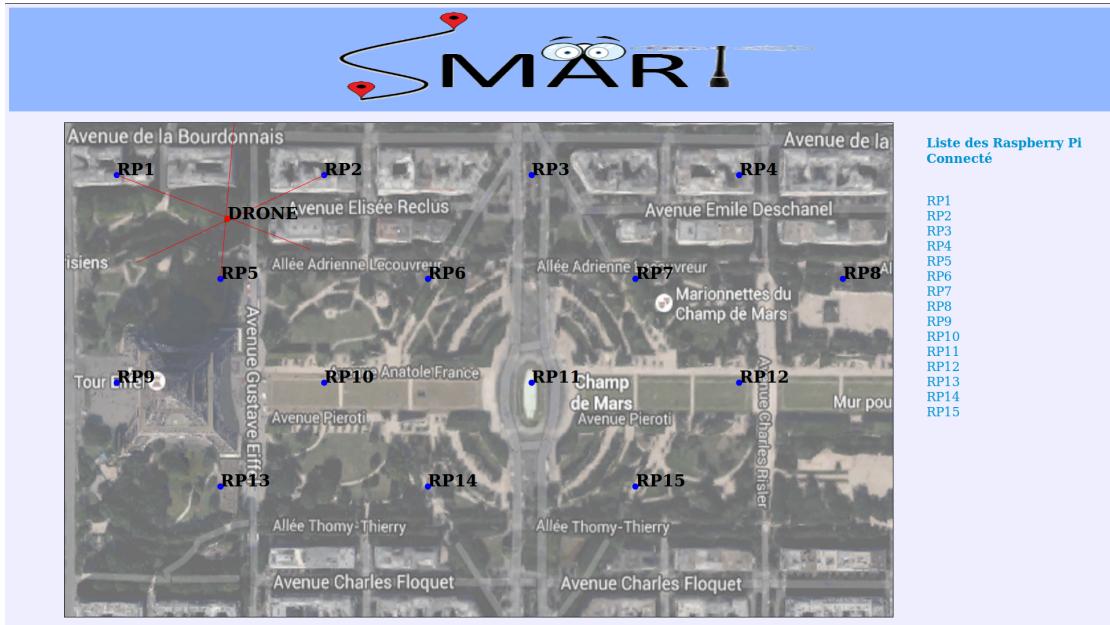


FIGURE 5.4 – Interface Web

Réalisation

6.1 Application Android : S.M.A.R.T Comm Center

Présentation Générale

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 manque, 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 vibrer 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 vibrer 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 développée est Android. En effet, une grande partie du code source est sous licence GPL 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.

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 informations 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 19¹. 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 du client

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, la fenêtre d'affichage des données de localisation du drone et celle des paramètres.

1. 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.



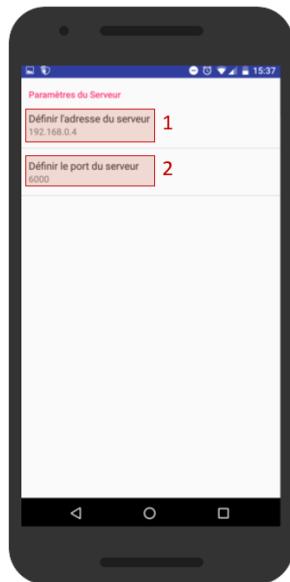
FIGURE 6.1 – Écran d'accueil



FIGURE 6.2 – Écran de localisation

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.

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.



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

FIGURE 6.3 – 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 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

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

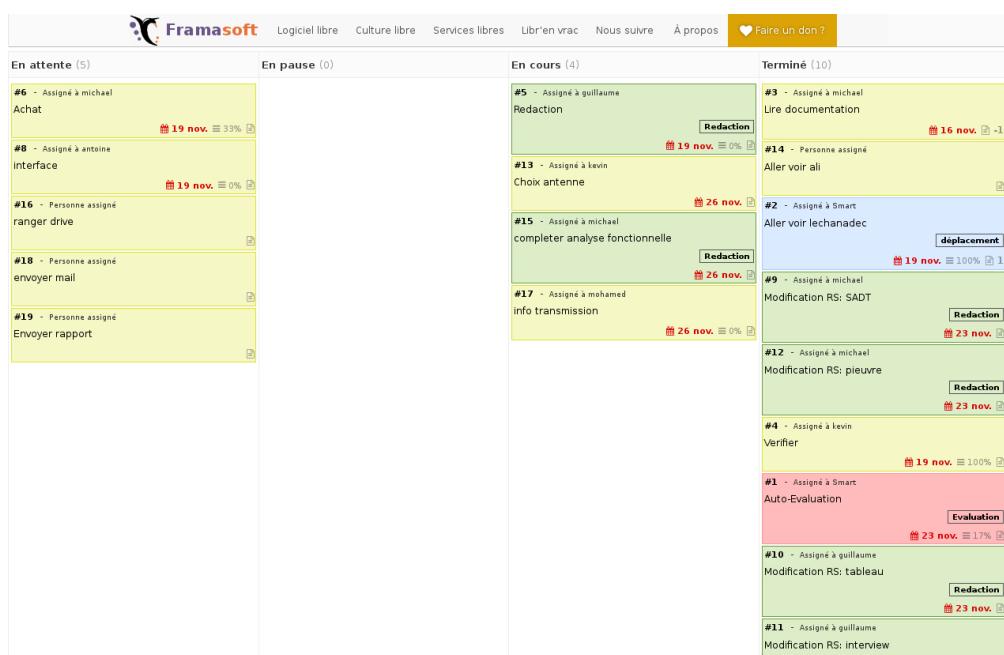


FIGURE 7.1 – Impression d'écran de notre Framaboard

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

GitHub

Dépôt dédié au projet SMART

Auteurs

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

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).

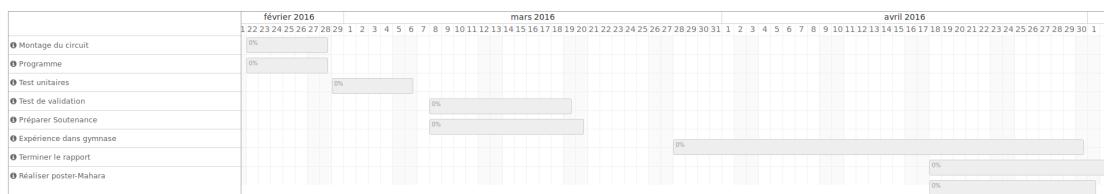


FIGURE 7.3 – Diagramme de Grant des prévision

A la fin du semestre nous obtenons le diagramme 7.4.

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

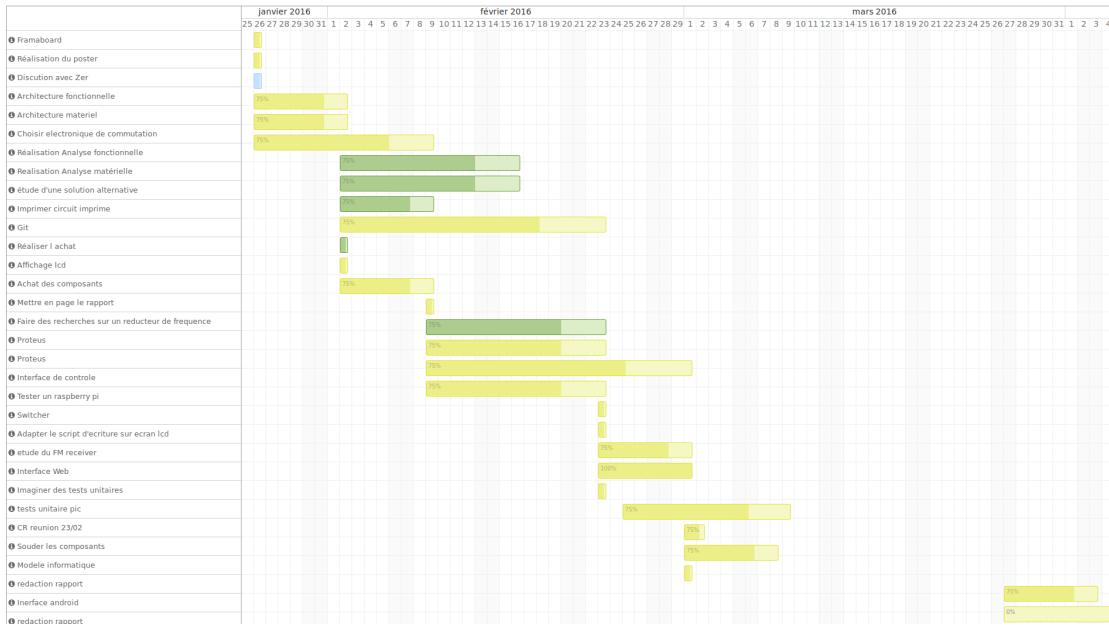


FIGURE 7.4 – Diagramme de Grant (1)

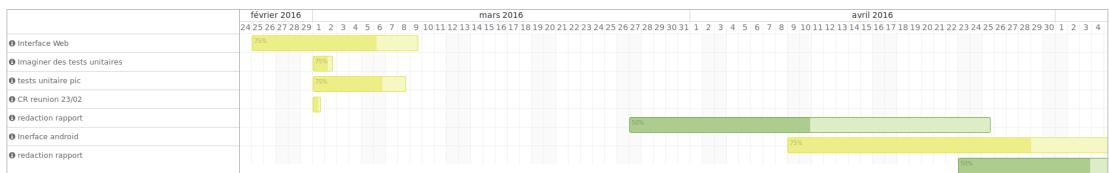


FIGURE 7.5 – Diagramme de Grant (2)

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

CHAPITRE **8**

Difficultés rencontrées

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.



A handwritten cursive signature of the word "Merci" is written in black ink. A small graphic of a pen nib is positioned at the end of the "i" in "Merci".

Annexe

Documentation Technique du Raspberry Pi B+

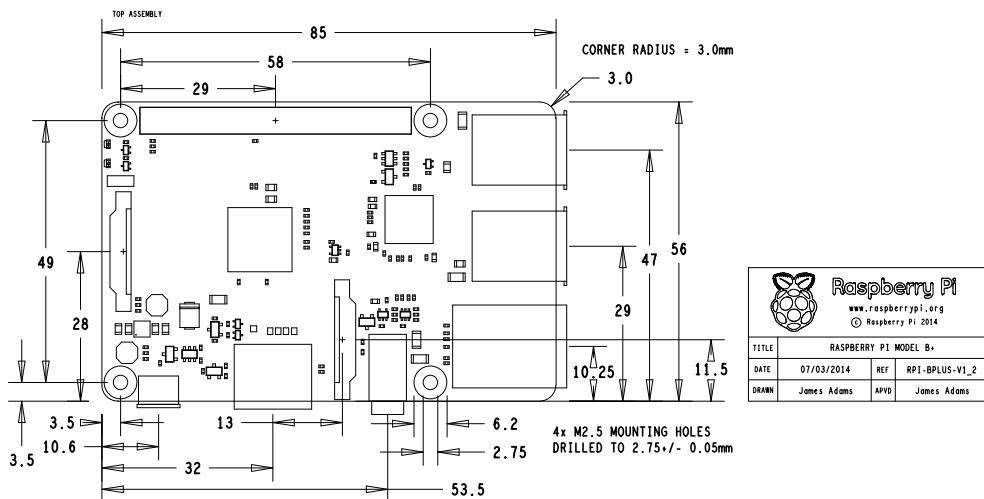


FIGURE A.1 – Dessin mecanique

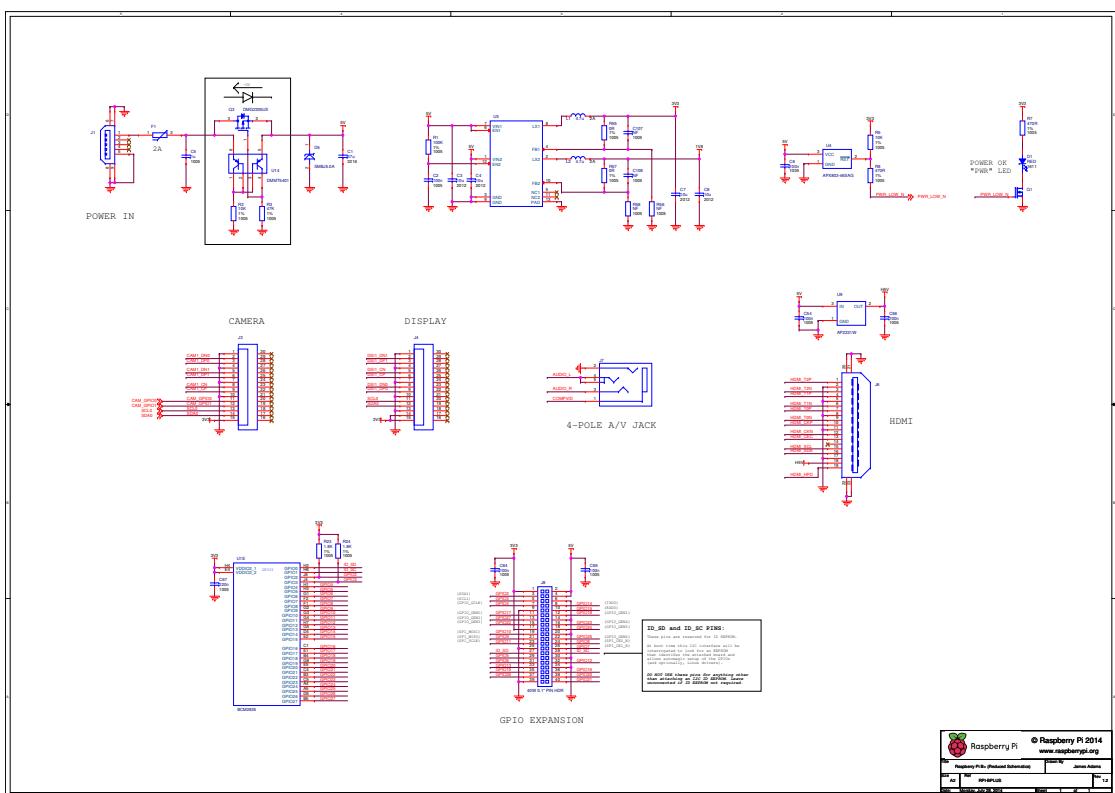


FIGURE A.2 – Schéma technique

Documentation Technique à Raspbian

Raspbian (recommended for Raspberry Pi 1) – is maintained independently of the Foundation ; based on the Debian ARM hard-float (armhf) architecture port originally designed for ARMv7 and later processors (with Jazelle RCT/ThumbEE and VFPv3), compiled for the more limited ARMv6 instruction set of the Raspberry Pi 1. A minimum size of 4 GB SD card is required for the Raspbian images provided by the Raspberry Pi Foundation. There is a Pi Store for exchanging programs.

The Raspbian Server Edition is a stripped version with fewer software packages bundled as compared to the usual desktop computer oriented Raspbian.

The Wayland display server protocol enables efficient use of the GPU for hardware accelerated GUI drawing functions.[104] On 16 April 2014, a GUI shell for Weston called Maynard was released.

PiBang Linux – is derived from Raspbian.

Raspbian for Robots – is a fork of Raspbian for robotics projects with Lego, Grove, and Arduino.

Client.py

```
1 #!/usr/bin/python3
2
3 # Definition d'un client reseau rudimentaire
4 # Ce client dialogue avec un serveur ad hoc
5
6 import socket, sys
7 from script.led.SimulLed import SimulLed
8
9
10 HOST = '127.0.0.1'
11 PORT = 50000
12 NOM = "RP1"
13
14
15 # 1) creation du socket :
16 mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17
18 # 2) envoi d'une requete de connexion au serveur :
19 try:
20     mySocket.connect((HOST, PORT))
21 except socket.error:
22     print("La connexion a echoue.")
23     sys.exit()
24 print("Connexion etablie avec le serveur.")
25
26 # 3) Dialogue avec le serveur :
27
28
29
30 message = str.encode("%s" % (NOM))
31 mySocket.send(message)
32
33 msgServeur = bytes.decode(mySocket.recv(1024))
34 print(msgServeur)
35
36
37 led = ""
38 ledHisto = ""
39 sim = SimulLed(NOM)
40
41 while 1:
42     try:
43         led = sim.getLed()
44         if led!=ledHisto:
```

```
45         print("la led allume:", led)
46         mySocket.send(str.encode(led))
47         ledHisto = led
48
49     except KeyboardInterrupt:
50         mySocket.send(str.encode("FIN"))
51         print("Connexion interrompue.")
52         mySocket.close()
53         break
```

Serveur.py

```

1 #!/usr/bin/python3
2
3 # Definition d'un serveur reseau gerant un systeme de CHAT simplifie.
4
5 # Utilise les threads pour gerer les connexions clientes en
6 # parallele.
7
8
9
10 import socket, sys, threading
11 import pickle
12
13
14 class ThreadClient(threading.Thread):
15     '''derivation d'un objet thread pour gerer la connexion avec un
16     client'''
17     def __init__(self, conn, nom):
18         threading.Thread.__init__(self)
19         self.connexion = conn
20         self.name = nom
21
22     def run(self):
23         while 1:
24             msgClient = bytes.decode(self.connexion.recv(1024))
25             if msgClient.upper() == "FIN" or msgClient == "":
26                 break
27             message = "%s> %s" % (self.name, msgClient)
28             print(message)
29             direction = "donc la direction %s" % (DIRECTION[msgClient])
30             print(direction)
31             POSITION[self.name]=POSITION[self.name][0],POSITION[self.
32                 name][1],POSITION[self.name][2],DIRECTION[msgClient]
33             ecriture()
34             # Fermeture de la connexion :
35             self.connexion.close() # couper la connexion cote serveur
36             del conn_client[self.name] # supprimer son entree dans le
37             # dictionnaire
38             print("Client %s deconnecte." % self.name)
39             # Le thread se termine ici
40
41     def initialisation():

```

```

40     # Initialisation des donnees
41     with open('donnees', 'rb') as fichier:
42         mon_depickler = pickle.Unpickler(fichier)
43         DIRECTION = mon_depickler.load()
44     lecture()
45
46
47 def lecture():
48     fichier = open("position", 'r')
49     for line in fichier:
50         contenue = line.split()
51         POSITION[contenue[0]] = contenue[2],contenue[3],contenue[4],
52             contenue[5]
53     fichier.close()
54
55 def ecriture():
56     fichier = open("position", 'w')
57     for clef in POSITION:
58         message = "%s : %s %s %s\n" %(clef,POSITION[clef][0],
59             POSITION[clef][1],POSITION[clef][2], POSITION[clef][3])
60         fichier.write(message)
61     fichier.close()
62
63 DIRECTION = {}
64 POSITION = {}
65
66 # Cette initialisation permet d'initialiser les dictionnaire qui
67     contiennent toute les variables
68 initialisation()
69
70
71 # Initialisation du serveur - Mise en place du socket :
72 mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
73 try:
74     mySocket.bind((HOST, PORT))
75 except socket.error:
76     print("La liaison du socket a l'adresse choisie a echoue.")
77     sys.exit()
78 print("Serveur pret, en attente de requetes ...")
79 mySocket.listen(5)
80
81
82 # Attente et prise en charge des connexions demandees par les
83     clients :
84 conn_client = {}          # dictionnaire des connexions clients
85 while 1:
86     try:
87         connexion, adresse = mySocket.accept()
88         # Creer un nouvel objet thread pour gerer la connexion :
89         nom = bytes.decode(connexion.recv(1024))
90         th = ThreadClient(connexion, nom)

```

```
90     th.start()
91     # Memoriser la connexion dans le dictionnaire :
92     #it = th.getName()
93     conn_client[nom] = connexion
94     print("Client %s connecte, adresse IP %s, port %s." %\
95           (nom, adresse[0], adresse[1]))
96     # Dialogue avec le client :
97     connexion.send(b'Vous etes connecte. Envoyez vos messages.')
98 except KeyboardInterrupt:
99     print("Fin du serveur")
100    break
```

Validation Fonctionnel Usine

Table des figures

1.1 Installation de Smart	5
1.2 Architecture Physique	6
1.3 Cercle de détection	7
1.4 Maillage de détection	7
2.1 Notre Raspberry Pi B+	8
2.2 Allumage d'une LED par Raspberry Pi	9
2.3 Méthode de connexion des leds dans le Montréal	10
2.4 Système modélisant une LED	10
2.5 3 PIC programmés	11
2.6 Schéma électrique du test unitaire	12
2.7 Schéma de montage du pic sur le Montréal 3v2	12
2.8 Comportement du filtre	13
2.9 S11 du filtre passe bande	14
2.10 Fréquences générées par le VCO alimenté à 8.1V centrée autour de 1.9GHz d'une largeur d'environ 10MHz	15
2.11 Montage de test de l'adaptateur	15
2.12 Photo du VCO et de son alimentation	16
4.1 schéma de fonctionnement d'un mixer	19
4.2 principe du fonctionnement d'un mixer	19
4.3 le down-converter et le filtre passe bande	20
5.1 Cas d'utilisation de l'interface	22
5.2 Diagramme de classe	22
5.3 triangulation	24
5.4 Interface Web	25
6.1 Écran d'accueil	28
6.2 Écran de localisation	29
6.3 Écran de localisation	30
7.1 Impression d'écran de notre Framaboard	32
7.2 Impression d'écran de notre GitHub	33
7.3 Diagramme de Grant des prévision	33
7.4 Diagramme de Grant (1)	34
7.5 Diagramme de Grant (2)	34
A.1 Dessin mécanique	39
A.2 Schéma technique	40