

Université François Rabelais de Tours



Institut Universitaire de Technologie de Tours

Département Génie Électrique et Informatique Industrielle

D4R4 – Rapport de fin de projet

Automatisation du placement d'une parabole



Lise CHAUVIN

Gilles DEVILLERS

Thomas GRAGEON

Alexandre MINOT

Groupe RoboTic 1

Groupe de projet n°304

Promotion 2016-2018

Coach : Vincent GRIMAUD

Jury :

Référent professionnel : Christophe TAILLIEZ

D4R4 – Rapport de fin de projet

Automatisation du placement d'une parabole



Lise CHAUVIN

Gilles DEVILLERS

Thomas GRAGEON

Alexandre MINOT

Groupe RoboTic 1

Groupe de projet n°304

Promotion 2016-2018

Coach : Vincent GRIMAUD

Jury :

Référent professionnel : Christophe TAILLIEZ

Sommaire

Introduction.....	4
1. Organisation.....	5
1.1. Carte mentale.....	5
1.2. Répartition des tâches.....	6
1.3. Budget.....	8
1.4. Choix des technologies informatiques.....	9
2. Acquisition.....	10
2.1. Fonctionnement du récepteur GPS.....	10
2.2. Identification des zones.....	12
2.3. Fonctionnement de la boussole.....	13
2.4. Acquisition de la qualité du signal.....	14
3. Commande.....	15
3.1. Vérin.....	13
3.1.1. Choix du vérin.....	13
3.1.2. Carte d'interface.....	13
3.1.3. Tests réalisés.....	13
3.1.4. Algorithme du vérin.....	13
3.2. Rotor.....	13
3.2.1. Présentation du rotor.....	14
3.2.2. Interface de commande du rotor.....	13
4. Coordination.....	14
4.1 Configuration du Raspberry Pi.....	14
4.2 Interface graphique.....	14
4.3 Carte mère.....	14
Conclusion.....	14
Résumés.....	14
Bibliographie.....	14
Index des illustrations.....	14
Annexes.....	14

Introduction

Dans le cadre du projet tutoré de deuxième année de GEII, nous, Lise Chauvin, Gilles Devillers, Thomas Grageon et Alexandre Minot, travaillons en collaboration avec Strategic Telecom Sécurité civile.

Notre objectif est d'automatiser le positionnement d'une parabole située sur un camion censé se déplacer pendant des interventions. Son positionnement est actuellement effectué manuellement et il nous a été demandé d'y remédier à l'aide d'un rotor et d'un vérin.

Il nous faudra choisir un vérin adapté qui répondra au cahier des charges, ainsi qu'une carte d'interface nous permettant de le piloter afin de régler la parabole en site¹. Commander le rotor sera également nécessaire pour régler la parabole en azimut². Pour nous connecter au satellite, il faudra connaître son type, qui dépend de la position géographique du camion. Une interface graphique sera codée sur le Raspberry Pi pour informer l'utilisateur de plusieurs paramètres et lui permettre de régler la parabole manuellement.

Dans un premier temps, nous allons présenter l'organisation du projet. Nous discuterons ensuite des solutions techniques envisagées pour répondre au cahier des charges.

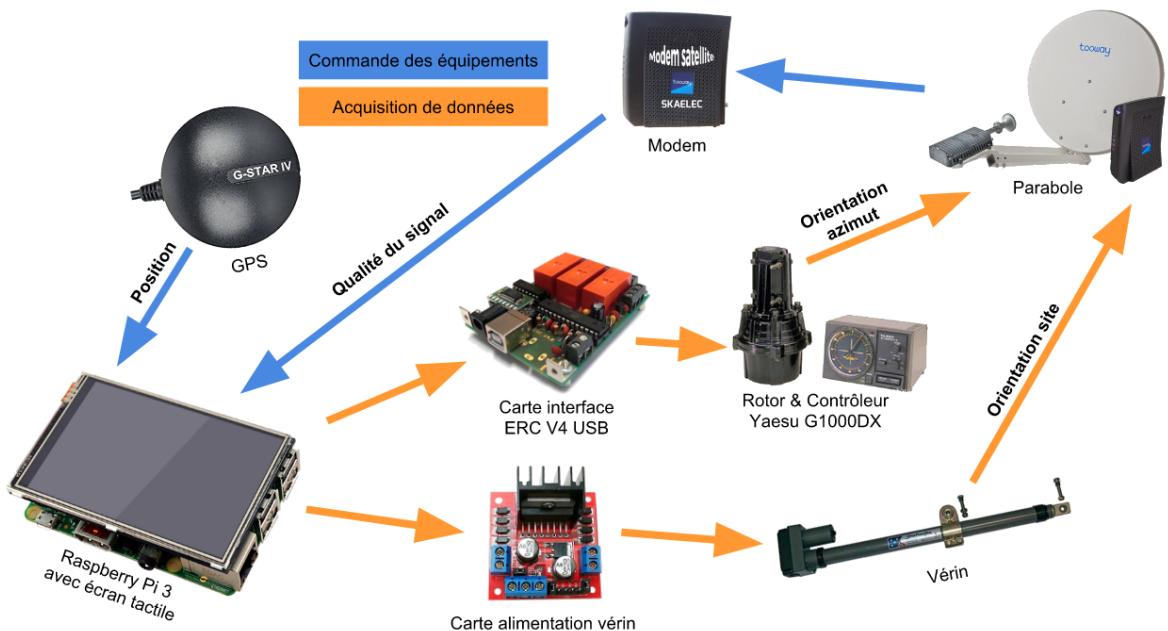


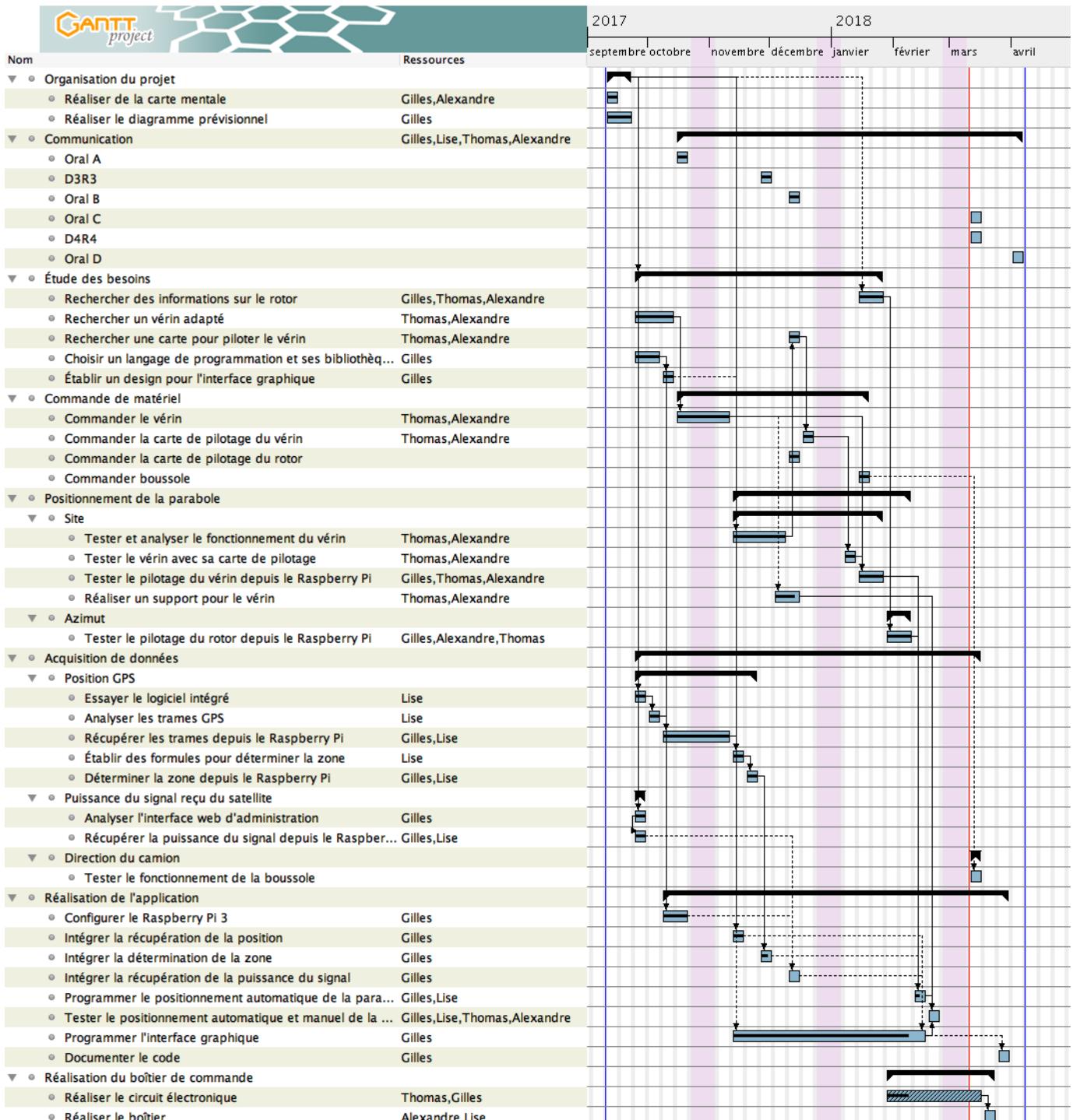
Illustration 1 : Structure du projet

1 site ou élévation : angle dans le plan vertical.

2 azimut ou azimuth : angle dans le plan horizontal

// Carte mentale

1.2. Répartition des tâches



Entre le D3R3 et le D4R4, notre diagramme de Gantt a évolué (diagramme de Gantt du D3R3 disponible en annexe 1). Du retard a été pris à cause de la livraison tardive du rotor et de son contrôleur. D'autres tâches ont été ajoutées telles que la documentation du code source, la réalisation d'un boîtier et la réalisation d'un circuit électronique commun pour rassembler tous les composants du projet.

Gilles s'est occupée du choix des technologies, de la configuration du Raspberry Pi, de la programmation de l'interface graphique, de la rédaction des documentations ainsi que de l'assistance sur la programmation des autres parties (contrôle du rotor, acquisition de la position GPS, acquisition de la qualité du signal...)

Thomas et Alexandre se sont concentrés sur l'étude et le pilotage de la parabole grâce au rotor, au vérin et aux contrôleurs qu'il était nécessaire de dimensionner et de commander. Thomas a également travaillé sur la carte électronique comportant tous les modules de notre installation ainsi que sur l'utilisation de la boussole.

Lise a travaillé sur la détermination du satellite auquel se connecter ainsi qu'aux algorithmes de balayage et aux essais permettant d'obtenir un placement optimal de la parabole.

1.3. Budget

Une partie du matériel nous a été fourni par le client, et nous avons dû commander trois éléments supplémentaires. Vis-à-vis du budget présenté dans le D3R3, seule la boussole a été rajoutée. Voici le budget final.

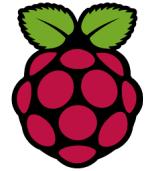
Désignation	Prix TTC
Raspberry Pi 3	35,00 €
Alimentation Raspberry Pi	10,00 €
Carte MicroSD Kingston 16 Go	8,00 €
ERC V4 USB	110,00 €
Écran tactile TFT 3,5"	37,00 €
Rotor Yaesu G-1000DX & Contrôleur	490,00 €
Câble Ethernet RJ45	5,00 €
Kit satellite Tooway (parabole, modem, support, activation)	375,00 €
Abonnement Tooway 25	120€/mois
Vérin SuperJack III	45,00 €
Contrôleur vérin L298N	6,00 €
Boussole HMC5883L	4,00 €
Total	1545 € + 120€/mois

 Matériel fourni par le client

 Matériel que nous avons acheté

1.4. Choix des technologies informatiques

Pour mettre en œuvre tous les composants de notre installation, nous avons besoin d'un **chef** d'orchestre que nous **programmerons** afin d'**acquérir** les données de nos capteurs pour ensuite **commander** la position de notre parabole.



Cet orchestrateur, imposé, est un **Raspberry Pi B+** : un micro-ordinateur simple à mettre en place et à faible consommation. Son système d'exploitation est **Raspbian**, une distribution **GNU/Linux** libre, gratuite et légère basée sur **Debian**³. Cette disposition est la plus optimisée pour cet ordinateur.



Ce Raspberry Pi est muni d'un **écran tactile LCD TFT**⁴ 3,5" qui permet de **contrôler** et **surveiller** le système grâce à son **interface graphique**.

Nous avons choisi de développer le programme du système en **Python**. Celui-ci générera l'**acquisition** des données, la **commande** de la parabole et l'**interface** graphique. Ce langage de **haut-niveau**⁵ est simple à utiliser et permet de développer rapidement grâce à sa structure permissive et les nombreuses **bibliothèques** prêtes à l'emploi disponibles sur Internet. En effet, pour l'interface graphique, nous avons choisi d'utiliser la bibliothèque Pygame pour sa simplicité d'utilisation, sa légèreté et le grand contrôle qu'elle nous offre sur l'affichage. Ainsi, toutes les parties de notre programme sont réalisées avec ce même langage.



³ Debian est une distribution Linux très basique

⁴ TFT (Thin-Film-Transistor Liquid-Crystal Display) : une technologie d'écran peu coûteuse.

⁵ haut-niveau : pas de gestion de la mémoire, et donc un développement plus rapide et plus facile

2. Acquisition

Pour répondre au cahier des charges, il nous est nécessaire d'acquérir plusieurs paramètres comme la position GPS⁶ du camion, son orientation et la qualité du signal satellite reçue. Ces données nous permettront d'identifier le satellite auquel nous connecter, mais aussi d'atteindre un placement optimal par la suite.

2.1. Fonctionnement du récepteur GPS

Afin de déterminer à quel satellite nous allons nous connecter, il faut connaître la position géographique du camion de la Sécurité Civile. Pour cela, un récepteur GPS nous a été fourni par la Sécurité Civile.

Ce récepteur est une tête GPS G-Star IV de chez GlobalSat qui se branche en USB à un ordinateur. Il transmet des trames par liaison selon le protocole de communication standardisé NMEA⁷ 0183. Ces trames sont les suivantes :



Illustration 2 :
Récepteur GPS utilisé

- Trame GGA : fournit l'heure, les coordonnées en longitude et latitude, l'altitude, le nombre de satellites trouvés, la précision de la mesure
- Trame GLL : fournit l'heure, la longitude, la latitude
- Trame GSA : indique le nombre de satellites trouvés et la précision de la mesure
- Trame GSV : indique le nombre de satellites trouvés, la qualité du signal de chaque satellite, son azimut et son élévation
- Trame VTG : indique la direction et la vitesse du récepteur GPS

```
$GPRMC,113307.479,V,,,,,,171017,,,N*41
$GPGGA,113308.480,,,0,00,,,M,0.0,M,,0000*52
$GPGSA,A,1,,,,,,,,,,,*1E
$GPGSV,3,1,12,21,66,094,,26,60,199,,31,57,343,,16,34,224,*75
$GPGSV,3,2,12,27,31,273,,18,14,046,,29,12,125,,20,12,110,*7B
$GPGSV,3,3,12,25,10,069,,10,03,021,,57,03,332,,23,02,246,*72
$GPRMC,113308.480,V,,,,,,171017,,,N*48
$GPGGA,113309.479,,,0,00,,,M,0.0,M,,0000*55
$GPGSA,A,1,,,,,,,,,,,*1E
```

Illustration 3 : Visualisation des différentes trames reçues

6 GPS : Global Positioning System, Système de positionnement par satellite

7 NMEA : National Marine Electronics Association, format de trames GPS

Pour extraire de ces trames les informations qui nous intéressent telles que la longitude, la latitude et le nombres de satellites connectés, nous utilisons la bibliothèque standard **pySerial**. Cette dernière facilite l'acquisition des trames GPS reçues par liaison série. La bibliothèque **pynmea2** nous permet d'exploiter ces trames de données reçues via le protocole NMEA.

```

9
10 while True:
11     nmea = gps.readline()
12
13     if nmea[0:6] == "$GPGGA":
14         gpsData = pynmea2.parse(nmea)
15         print("Latitude : " + "%02d°%02d'%.4f"" % (gpsData.latitude, gpsData.latitude_minutes, gpsData.latitude_seconds))
16         print("Longitude : " + "%02d°%02d'%.4f"" % (gpsData.longitude, gpsData.longitude_minutes, gpsData.longitude_seconds))
17         print("Fix" if gpsData.gps_qual else "No fix")
18         print("Sats : " + gpsData.num_sats)
19         print("")
20
21 gps.close()

```

Illustration 4 : Essai de récupération des coordonnées GPS

Une fois les premiers essais de récupération de position réalisées, nous avons créé une « classe » qui nous permet d'utiliser plus facilement le GPS dans notre programme final. Celle-ci permet de manipuler un « objet » GPS qui configure et initialise automatiquement la connexion USB avec celui-ci. Une méthode « `getInfos()` » permet de récupérer les informations qui nous intéressent.

```

1 class GPS(object):
2     def __init__(self, path):
3         self.path = path
4
5     try:
6         self.gps = serial.Serial(path, 4800)
7         self.gps.readline() # On se place au début d'une trame
8     except serial.SerialException:
9         # Le GPS n'est pas connecté
10        self.gps = None
11
12    def __del__(self):
13        if self.gps is not None:
14            self.gps.close()
15
16    def getInfos(self):
17        if self.gps is not None:
18            while self.gps.in_waiting > 0:
19                serialBuffer = self.gps.readline()
20                if serialBuffer[0:6] == "$GPGGA":
21                    self.data = pynmea2.parse(serialBuffer)
22
23            return self.data
24        else:
25            return None

```

Illustration 5: Classe facilitant l'utilisation du GPS

2.2. Identification de la zone satellite

Désirant connaître la longitude et la latitude du camion, nous avons le choix entre les trames GGA et GLL. Dans l'éventualité où nous voudrions connaître l'altitude du camion, nous envisageons d'exploiter la trame GGA.

Une fois les coordonnées récupérées, nous pouvons localiser assez précisément le camion et déduire à quel satellite nous devons nous connecter pour avoir une réception optimale.

Une capture d'écran du site web qui cartographie les satellites en Europe est présentée ci-contre. Nous nous en servons pour obtenir les coordonnées des satellites et ainsi déterminer à quel satellite nous nous connectons.

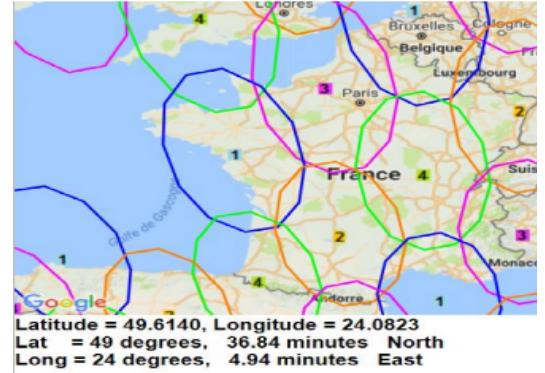
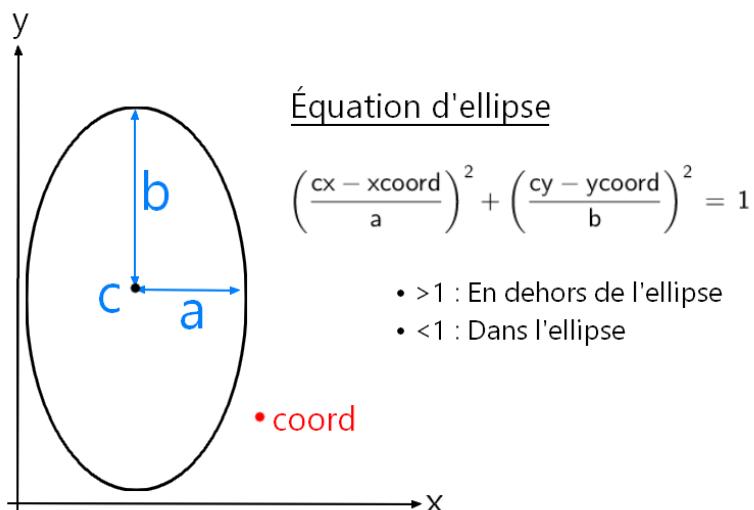


Illustration 6: Zones des satellites en France

Pour cela, nous utilisons un modèle mathématique qui approxime les ellipses inclinées par des ellipses droites, proposant une précision suffisante. En effet, il est possible de savoir s'il on se trouve dans chaque zone ou non en établissant une équation pour chaque ellipse de la carte. Il suffit ensuite de tester chaque équation pour toutes les zones avec les paramètres GPS du camion et d'en déduire la zone optimale.

Comme expliqué sur l'illustration ci-dessous, nous calculons le résultat de l'équation qui dépend du centre de l'ellipse (cx et cy), de ses petit et grand rayons (a et b), et des coordonnées du camion ($xcoord$ et $ycoord$). Le résultat nous indique si le camion (modélisé par le point rouge) se trouve en dedans ou en dehors de la zone du satellite). S'il est supérieur à un, le camion n'est pas dans la zone, s'il y est inférieur, il pourra capter le satellite.



Une fois la zone satellite identifiée, il est possible de calculer l'angle d'azimut à appliquer à la parabole pour qu'elle reçoive le signal satellite. Pour cela, nous avons utilisé une formule trigonométrique extraite du code source d'une page internet proposant d'afficher les angles d'azimut et d'élévation en entrant les coordonnées désirées. Nous avons exploité cette formule pour qu'elle nous fournisse l'angle d'azimut à transmettre au rotor en fonction des coordonnées fournies par le récepteur GPS. Nous avons également tiré parti de la formule de l'élévation, dans l'éventualité où nous en servirions. Code complet en annexe 2.

```
azimuth=180+180*math.atan(-1*math.tan(math.pi*(position-longitude)/180)/math.sin
    (math.pi*latitude/180))/math.pi
elevation1=math.acos(math.cos(math.pi/180*(longitude-position))*math.cos(math.pi
    /180*abs(latitude)));
elevation=180*math.atan((math.cos(elevation1)-(6378000/(6378000+35786000)))/math
    .sin(elevation1))/math.pi;
```

Illustration 7 : Calcul de l'azimut et de l'élévation

2.3. Fonctionnement de la boussole

Afin de pouvoir déterminer l'angle d'azimut à transmettre au rotor, il est nécessaire de connaître l'orientation du camion par rapport au nord, et de tenir compte de ce décalage pour orienter correctement la parabole. Pour cela, nous avons choisi d'équiper le Raspberry Pi d'une boussole qui nous fournit la position du camion par rapport au nord.

Sur l'exemple ci-dessous, l'angle d'azimut recommandé par rapport au nord est de 90° (**Az**), cette valeur est fournie par le programme. Malheureusement, il y a très peu de chance que le camion soit orienté plein nord. La boussole doit donc nous fournir un angle (**dec**) qui nous permettra de déterminer l'angle à transmettre au rotor (**Atr**), tenant compte du décalage du camion par rapport au nord (**dec**).

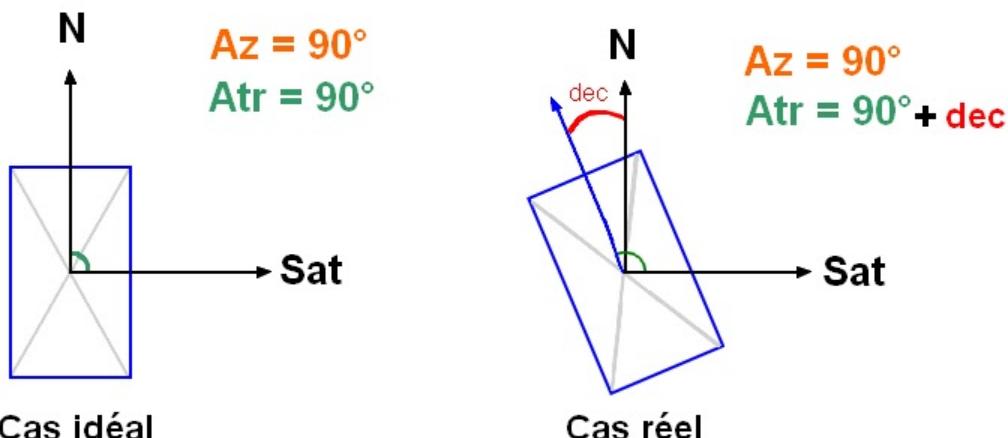


Illustration 8 : Calcul de l'azimut en tenant compte de l'orientation du camion

Le choix de la boussole s'est porté sur le capteur QMC5883l, car il est facile de communiquer avec lui en I2C à partir du Raspberry Pi. De plus, il permet une mesure d'angle précise à 2° près. Il répond également à nos critères d'alimentation car il peut fonctionner en 3,3V ou en 5V, les deux tensions que peuvent fournir les broches du Raspberry Pi.

Pour établir le dialogue avec le Raspberry Pi, nous avons utilisé la bibliothèque Python Smbus qui contient des méthodes simples pour la lecture et l'écriture sur le bus I2C. Avant de pouvoir lire les valeurs renvoyées par la boussole, il faut l'initialiser à l'aide des différents registres.

```
boussole = 0x0d
bus.write_byte(data(boussole, 0x01, 0xB)
```

Illustration 9: Exemple d'initialisation de registre

Ces registres permettent entre autre de gérer les interruptions, de sélectionner le mode de fonctionnement de la boussole (en veille ou continu), établir la fréquence de rafraîchissement des sorties et de choisir la sensibilité de la boussole. Dans notre cas, la boussole sera utilisée en mode continu avec une fréquence de rafraîchissement de 10 Hz, suffisante pour l'application que l'on en fait. Nous n'utilisons pas d'interruptions, et requérons un rapport d'échantillonnage le plus haut possible pour limiter les interférences. Une fois les registres initialisés, nous pouvons récupérer les données correspondant aux différents axes. Celles-ci sont renvoyées sur deux octets chacunes.

00H	Data Output X LSB Register XOUT[7:0]
01H	Data Output X MSB Register XOUT[15:8]

Illustration 10: Extrait de la datasheet de la boussole QMC5883l

Sur cet extrait, on peut voir que les données correspondant à l'axe X sont bien séparées en deux parties, l'octet LSB⁸ et l'octet MSB⁹ ayant une adresse différente. Cette variable recomposée est signée¹⁰, donc non prise en compte par le Python. Pour pouvoir la manipuler, il faut la convertir. Malheureusement, l'orientation obtenue grâce à au traitement de ces données n'est pas convenable et il faudra trouver une meilleure façon de les interpréter.

```
if ((y & 32768)==32768): #signer
    y=y-32768
    y=~y
    y=y-1
    y=-y
```

Illustration 11: Transformation de la variable signée en variable utilisable par Python

8 Least Significant Byte : octet de poids faible

9 Most Significant Byte : octet de poids fort

10 Signée : qui peut prendre des valeurs positives et négatives

2.1. Acquisition de la qualité du signal

Afin d'orienter la parabole de façon **optimale** quand elle sera sur le camion, nous devons accéder à la **qualité** du signal satellite reçu par la parabole. Cela permettra ensuite d'effectuer un **balayage** de la position de la parabole afin de trouver la position optimale qui offre la **meilleure** connexion au satellite.

Pour cela, notre application doit accéder à la page web de configuration du modem de la parabole. Celle-ci permet de configurer le satellite à utiliser en fonction de la position du camion et affiche la qualité du signal avec le satellite en question.

La **qualité** de la connexion, ainsi que la quantité de **bruits** parasites, peuvent être visualisées sur l'interface d'administration du **modem** internet **Tooway** associé à la parabole après avoir choisi la zone satellite **adaptée** à la position du camion.

toowayTM
fast internet everywhere

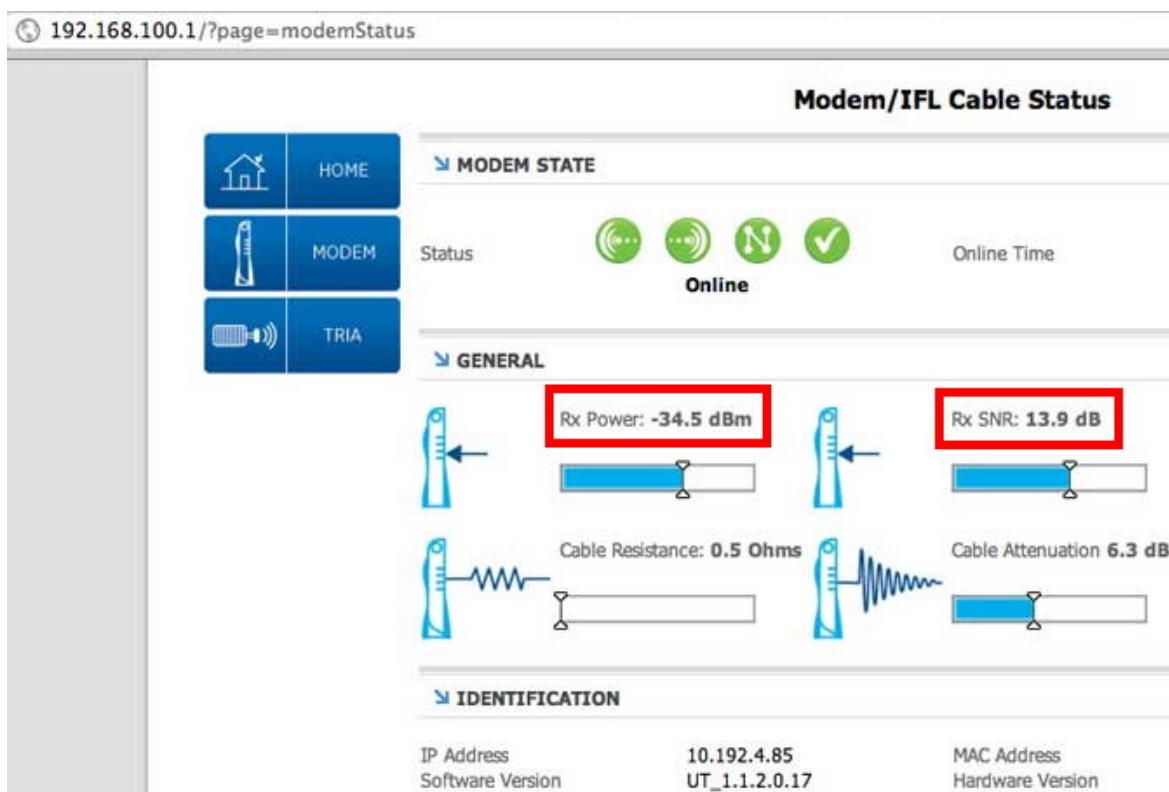


Illustration 12: Page d'administration du modem de la parabole

À partir du code source **JavaScript** de cette page web, nous avons pu trouver la **source** de ces deux informations. Celles-ci sont situées sur la page «cachée» “*index.cgi?page=modemStatusData*”. Nous pouvons ensuite **exploiter** ces informations et les utiliser afin de réaliser notre balayage.

Pour **acquérir** la puissance du signal et la quantité de bruit en Python, nous avons choisi d'utiliser la bibliothèque **Requests**, très bien documentée. Celle-ci permet d'effectuer des requêtes HTTP nécessaires pour récupérer le contenu de la page sur laquelle se trouvent les données qui nous intéressent. Nous obtenons ainsi le corps de la page, qui est composé de plusieurs valeurs séparées par «##». En découplant cette chaîne de données, on peut ensuite récupérer les informations qui nous intéressent.



```

1 import requests
2 from requests_toolbelt.utils import dump
3
4 data = []
5 def getData():
6     global data
7
8     r    = requests.get("http://192.168.100.1/index.cgi?page=modemStatusData")
9     data = r.text.split("##")
10
11 getData()
12 rxPower = data[14]
13 rxSNR   = data[11]
14
15 print("Rx Power : " + rxPower + " dBm\tRx SNR : " + rxSNR + " dB")

```

Illustration 13: Programme de récupération des informations de connexion au satellite

Ce programme va télécharger la page web depuis le modem Tooway pour récupérer la qualité du signal (« Rx Power ») et la quantité de bruits parasites (« RX SNR »). Ces valeurs seront ensuite utilisées pour évaluer le positionnement de la parabole en temps réel.

Rx Power : -43.5 dBm	Rx SNR : 11.2 dB
Rx Power : -43.3 dBm	Rx SNR : 10.7 dB
Rx Power : -43.2 dBm	Rx SNR : 11.8 dB
Rx Power : -43.8 dBm	Rx SNR : 13.2 dB
Rx Power : -42.9 dBm	Rx SNR : 11.4 dB

Illustration 14: Relevé de valeurs de puissance du signal et de bruit

3. Commande

Pour procéder au placement automatique de la parabole, il est nécessaire de la piloter depuis l'interface graphique horizontalement (azimut) et verticalement (site), afin d'avoir une couverture totale de la zone du satellite dans le ciel. Ces déplacements sont effectués grâce à un rotor et un vérin électriques.

3.1. Vérin

3.1.1. Choix du vérin

Il nous faut un vérin possédant des caractéristiques bien précises.

Tout d'abord, le vérin doit avoir une force de poussée suffisante pour pouvoir soulever la parabole qui pèse environ 15 kilogrammes. La force réelle de poussée reste inconnue en l'absence du matériel nécessaire à la mesure de cette force en Newton-mètre (N.m). De plus, cette force varie en fonction de l'endroit de la poussée. En effet, plus la poussée est près du centre, plus sa force est importante.

De plus, le vérin doit pouvoir fonctionner en étant alimenté en 12 ou 24 volts maximum, tensions pouvant être fournies par le camion de la Sécurité Civile. Nous n'avons pas de contrainte en ce qui concerne la taille du vérin. En effet, la fixation du vérin sur la parabole va être réalisée par nos soins et adaptée à la taille du vérin.

Pour répondre aux critères ci-dessus, le vérin retenu est le Super Jack III de chez Jaeger. Les caractéristiques de ce vérin sont les suivantes :

- Taille : 12 pouces
- Charge statique : 225 kg
- Charge dynamique : 135 kg
- Alimentation : 36 volts DC
- Précision du capteur : 76 impulsions par pouce
- Température de fonctionnement : - 30 °C à 50 °C



*Illustration 15:
Représentation 3D du vérin*

Malgré une alimentation supérieure à celle mentionnée dans nos critères, ce vérin fonctionnant en 24 volts, et même s'il perd en charge statique et dynamique, sa force reste malgré tout largement suffisante pour pouvoir orienter la parabole, qui ne pèse que 15 kg. Ce vérin ne peut pas être commandé directement par le Raspberry Pi car celle-ci ne peut fournir la puissance nécessaire (5V au lieu de 12V minimum). C'est pour cela qu'une carte d'interface est obligatoire afin d'adapter la tension et de gérer la polarité.

3.1.2. Carte d'interface du vérin

Le choix de la carte d'interface s'est porté sur le L298N.

Sa tension de conduite s'étend de 5 à 36 volts et elle peut fournir jusqu'à 2A. Sa tension de commande est donc compatible avec le Raspberry Pi.

Voici les branchements :

BROCHE	FONCTION
OUT1, OUT2	Branchement moteur 1
OUT3, OUT4	Branchement moteur 2
IN1, IN2	Choix de polarité du moteur 1
IN3, IN4	Choix de polarité du moteur 2
ENA	Signal PWM pour la vitesse du moteur 1
ENB	Signal PWM pour la vitesse du moteur 2
Entrée + et GND	Alimentation des moteurs
5 V	Entrée 5 V nécessaire pour alimenter la partie commande quand l'alimentation du vérin dépasse 12V

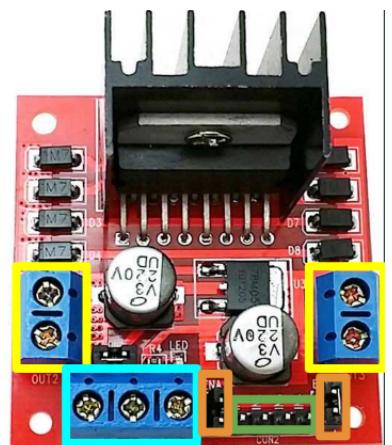


Illustration 16: Carte d'interface du vérin

Cette carte d'interface correspond parfaitement à nos besoins grâce à sa faible tension d'alimentation, qui peut être fournie par le Raspberry Pi. De plus, le L298N gère l'inversion de polarité. Pour le contrôler avec le Raspberry Pi, il faudra envoyer un signal soit sur IN1 soit sur IN2 pour choisir le sens de rotation du moteur (on choisira ainsi de faire sortir ou rentrer le vérin), il faut ensuite envoyer un signal sur ENA pour le faire fonctionner.

3.1.3. Tests réalisés

Plusieurs tests ont été réalisés sur le vérin avec une alimentation stabilisée, dont un test en 24 volts, un test sur le capteur qui détecte les tours effectués par le moteur et un test de la carte d'interface L298N.

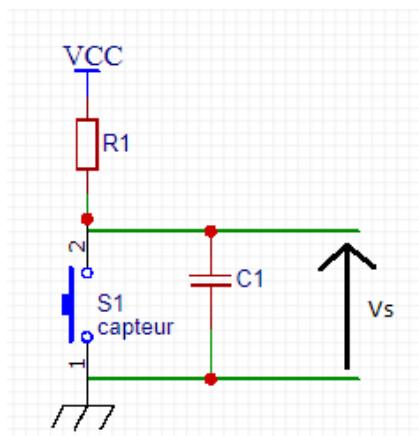


Illustration 17: Schéma de test du capteur à lames souples

Le capteur utilisé dans ce vérin est un interrupteur à lames souples. Le contact est réalisé sous l'effet d'un champ magnétique. Lorsque le moteur tourne, il fait tourner un aimant qui active l'interrupteur à chaque tour qu'il fait et laisse passer le signal que l'on envoie sur l'entrée « sensor » du vérin. Le schéma utilisé pour les tests du capteur est présenté ci-contre.

La résistance remplit la fonction de résistance de tirage et le condensateur C1 lisse la tension. En condition réelle, nous n'avons pas besoin de ces deux composants, en effet, le Raspberry Pi possède déjà une résistance de tirage interne et la tension est suffisamment lisse pour se passer du condensateur.

Ci-dessus un relevé de la sortie moteur OUT1 de la carte L298N alimentée en 12 V. On remarque que lorsque que l'on met l'entrée IN1 à 1 et IN2 à 0, la tension de sortie devient la même que l'alimentation, soit 12 V lors de notre essai. Par ailleurs on voit aussi que lorsque IN1 est à 0 et IN2 à 1, on a une tension résiduelle de l'ordre de 1 V due à l'inversion de polarité. Lorsque que les commandes d'entrée sont toutes les deux à zéro, le vérin s'arrête.

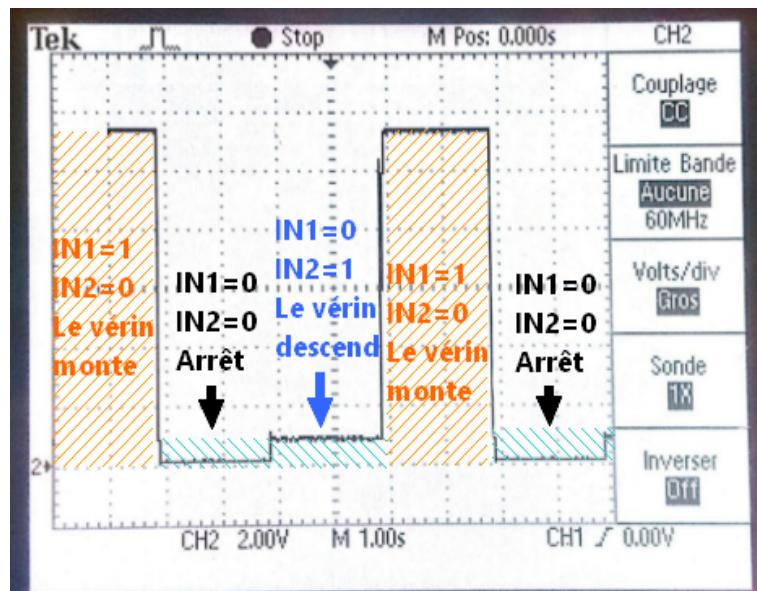


Illustration 18: Relevé de la sortie moteur OUT 1 en fonctions de différentes commandes d'entrée

3.1.4. Algorithme du vérin

Les fonctions de commande du vérin sont relativement simplifiées grâce à la bibliothèque Rpi.GPIO. Pour l'utiliser, il faut d'abord préciser que l'on travaille avec les numéros des GPIO¹¹ ou avec les numéros des broches :

- GPIO.setmode(GPIO.BCM) pour dire qu'on travail avec les numéros des GPIO
- GPIO.setmode(GPIO.BOARD) pour dire qu'on travail avec les numéros des broches

Ensuite, il faut définir le mode de fonctionnement :

- GPIO.setup("numéro du GPIO", GPIO.OUT) définit le GPIO en sortie.
- GPIO.setup("numéro du GPIO",GPIO.IN) définit le GPIO en entrée.

Une fois ces étapes effectuées et si le mode de fonctionnement est configuré en sortie, on peut commander nos sorties avec les commandes suivantes :

- GPIO.output("numéro du GPIO",GPIO.LOW) applique un état logique 0 à notre GPIO
- GPIO.output("numéro du GPIO",GPIO.HIGH) applique un état logique 1 à notre GPIO

```
from RPi import GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
mont = 16
desc = 18
GPIO.setup(mont,GPIO.OUT)
GPIO.setup(desc,GPIO.OUT)
#GPIO.output(mont,GPIO.LOW)
#GPIO.output(desc,GPIO.LOW)

def monter():
    GPIO.output(mont,GPIO.HIGH)
    GPIO.output(desc,GPIO.LOW)

def descendre():
    GPIO.output(mont,GPIO.LOW)
    GPIO.output(desc,GPIO.HIGH)

def arreter():
    GPIO.output(mont,GPIO.LOW)
    GPIO.output(desc,GPIO.LOW)|
```

Pour contrôler le vérin, nous avons créé trois fonctions :

- “monter”, qui met les sorties IN1 = 1 et IN2 = 0
- “descendre”, qui met les sorties IN1 = 0 et IN2 = 1
- “arreter”, qui met les sorties IN1 = 0 et IN2 = 0

Illustration 19: Configuration des GPIO et définition des fonctions

11 General Purpose Input/Output : port d'entrée/sortie

Pour programmer le placement automatique du vérin, deux options s'offrent à nous :

- La première consiste à faire un balayage entier de la zone afin de trouver la position où le signal est le plus fort, puis de resserrer le balayage au fur et à mesure.
- La seconde consiste à positionner la parabole à proximité de l'angle théorique obtenu par le programme d'identification de la zone dans un premier temps, puis faire un balayage autour de cette position.

La deuxième option nécessite la gestion des impulsions générées par l'interrupteur à lames souples du vérin. Nous verrons si la nécessité de se positionner dans un premier temps à l'angle théorique est constatée, en fonction des résultats que nous obtiendrons lors des tests en conditions réelles.

Pour orienter la parabole en site, la solution retenue est donc la première. Nous allons effectuer un balayage en analysant la qualité du signal reçu. Le balayage, ample au départ, sera de plus en plus ciblé et se centrera sur l'angle où le signal sera le plus fort.

Ce fonctionnement est réalisé par des boucles imbriquées dans lesquelles des mesures de la force du signal sont effectuées. La valeur maximale est à chaque fois mémorisée pour que le programme sache autour de quelle valeur resserrer son balayage pour finalement s'arrêter à une position optimale. L'algorigramme ci-contre résume ce programme en version très simplifiée. Le code complet se trouve en annexe 3.

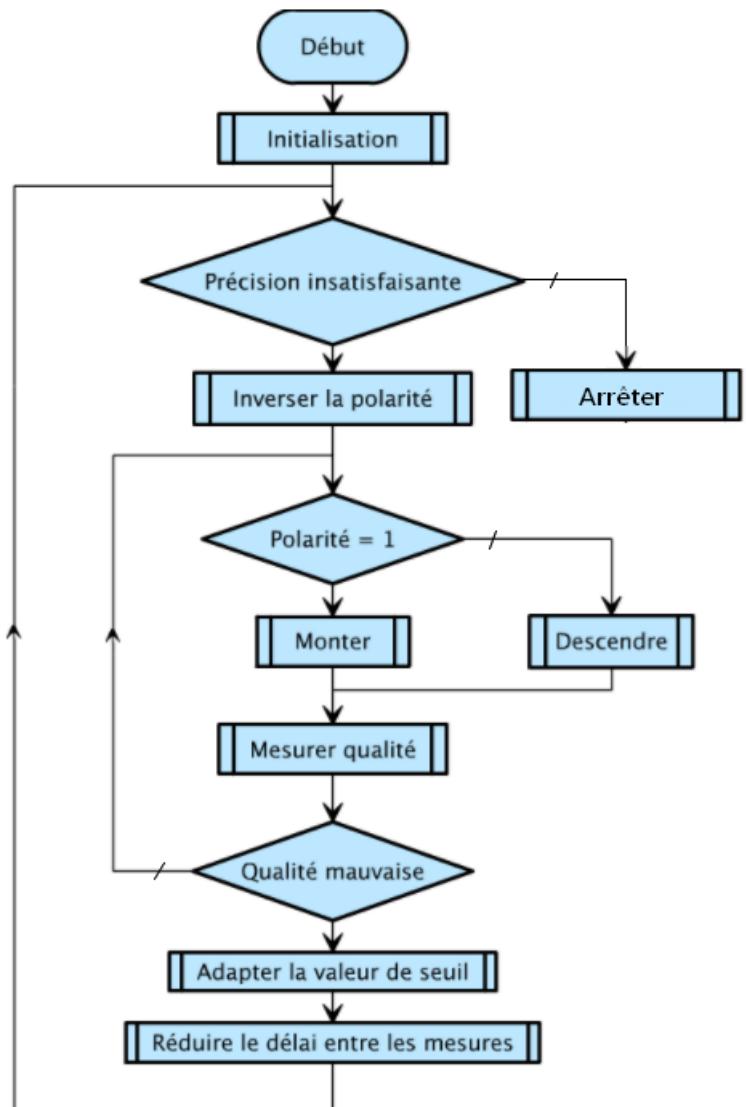


Illustration 20: Algorigramme de placement du vérin

3.2 Rotor

Il ne suffit pas de positionner la parabole en site. Il faut à présent s'intéresser à son orientation en azimut, assurée par le rotor.

3.2.1 Présentation du rotor

Le rotor qui nous est fourni est un YAESU G-1000DX, qui a les caractéristiques suivantes :

- Charge statique : 6000kg/cm
- Charge dynamique : 600 à 1100 kg/cm
- Temps d'une rotation à 360 ° : 43 à 93 secondes
- Diamètre : 186 mm
- Hauteur : 300 mm
- Charge verticale : 200 kg

Pour piloter le rotor, nous devrons utiliser une carte d'interface. Celle qui nous est fournie est un ERC Mini.

3.2.2 Interface de commande du rotor

Le rotor a une commande manuelle, mais dispose également d'un connecteur pour être commandé par un boîtier d'interface. Dans notre cas, on utilise l'ERC Mini 4. Le programme rotor permet de communiquer avec l'ERC Mini. Cet appareil communique en liaison série par un port USB qui se décompose ainsi : un bit de début, huit bits de données, et un bit de fin, le bit de parité étant optionnel. Il s'agit d'une trame en liaison série par un port USB qui se décompose ainsi.



Illustration 21: Trame série

Pour pouvoir utiliser la communication série sur le raspberry Pi nous avons besoin de la librairie “serial” . Elle s'utilise comme suit :

Premièrement nous créons un objet de type “serial” en précisant le port de communication grâce à cette commande : `ser = serial.Serial(port='/dev/ttyUSB1')`. Sans précision de notre part, il y aura huit bits de données, un bit de start, un bit de fin et une vitesse de communication de 9600 baud comme nous le souhaitons.

Ensute nous utilisons la fonction “write” de cet objet pour transmettre nos commandes.

Les commandes que nous allons envoyer sont les suivantes :

- Mxxx<cr> : rotation azimut à xxx degrés
- L<cr> : rotation dans le sens contraire des aiguilles d'une montre
- R<cr> : rotation dans le sens des aiguilles d'une montre
- S<cr> : arrêt du rotor

Le « <cr> » représente le retour chariot indiquant un retour à la ligne qui a la valeur décimale 13 ou 0x0D en hexadécimal. En python il est représenté par “\r\n” .

Lors de la toute première utilisation de l'ERC Mini, on doit procéder à la calibration de l'ERC pour pouvoir utiliser ces commandes :

- Envoyer la commande « sPRO0000 » : définit le protocole de communication avec le rotor, dans notre cas le protocole Yaesu GS232A
- Positionner manuellement le rotor à 0°
- Envoyer la commande « sCAL0000 » : effectue la calibration à 0°
- Positionner manuellement le rotor à 360°
- Envoyer la commande « sCAR0360 » : effectue la calibration à 360°

```
import serial
import time

class rotor:
    def __init__(self,chemin): #Constructeur, initialise la communitation
        self.chemin = chemin
        self.ser = serial.Serial('/dev/ttyUSB0')

    def tourner(self,angle): #Positionne le rotor a l'angle envoye en argument
        angle = int(angle)
        commande="M"+repr(angle)+"\r\n"
        self.ser.write(commande.encode('latin-1'))

    def tournerdroite(self): #Fait tourner le rotor a droite
        self.ser.write(b'R\r\n')

    def tournergauche(self): #Fait tourner le rotor a gauche
        self.ser.write(b'L\r\n')

    def stop(self): #Stop la rotation du rotor
        self.ser.write(b'S\r\n')

    def calibrationdroite(self):#Envoi la commande de calibration droite
        self.ser.write(b'sCAR0360\r\n')

    def calibratongauche(self):#Envoi la commande de calibration gauche
        self.ser.write(b'sCAL0000\r\n')

    def calibrationconfig(self):#Configuration de l'ERC
        self.ser.write(b'sPRO0000')
        self.ser.write(b'sSPA0001')
        self.ser.write(b'sSPL0001')
        self.ser.write(b'sSPH0003')
```

Illustration 22: Classe permettant de commander le rotor

Comme illustré précédemment, la classe “rotor” a été créée pour synthétiser ces différentes commandes. Les commandes secondaires seront détaillées en annexe 4.

La classe rotor comporte plusieurs fonctions :

- une fonction “calibrationgauche”, qui envoie la commande de calibration gauche
- une fonction “calibrationdroite”, qui envoie la commande de calibration droite
- une fonction “tournergauche” qui fait tourner le rotor à gauche
- une fonction “tournerdroite” qui fait tourner le rotor à droite
- une fonction “stop” qui arrête la rotation du rotor
- une fonction “positiondegre” qui envoie la commande pour positionner le vérin au degré passé en paramètre

Toutes ces fonctions ont été testées, et cette classe a été intégrée dans le programme du GPS, ainsi le programme envoi directement l’angle d’azimut au rotor qui se positionne automatiquement à l’angle souhaité.

4. Coordination

L'**acquisition** des données des **capteurs** et le **pilotage** de la parabole sont réalisées par le biais du **Raspberry Pi B+**. Nous avons installé de nombreux composants dessus afin de permettre la **communication** avec tous les **modules** nécessaires ainsi que le développement de l'**interface** graphique. Nous avons également réalisé une **carte** électronique sur laquelle tous ces composants seront placés afin d'avoir un système pratique à manipuler.

4.1. Configuration du Raspberry Pi

L'installation et la configuration du **Raspberry Pi** sont constituées de nombreuses étapes qui peuvent être fastidieuses à réaliser manuellement. Pour cette raison, nous avons réalisé un **mode d'emploi** avec un script qui va installer toutes les dépendances nécessaires à l'exécution du programme principal automatiquement.

Les principales étapes de l'installation et de la configuration sont les suivantes :

- **Flash** de la distribution **Raspbian Jessie** pour écran **PiTFT** sur la carte **microSD** du **Raspberry Pi**
- **Connexion** au **Raspberry Pi** par liaison **SSH** puis transfert du **script** d'installation des dépendances « dependances.sh » et du **programme principal** « parabole.py »
- **Configuration** initiale du **Raspberry Pi**
- **Exécution** du script d'installation des **dépendances** « dependances.sh »
- **Calibration** éventuelle de l'**écran tactile**
- **Exécution** du programme principal « parabole.py »

Le mode d'emploi détaillé est disponible en annexe 5, le script « dependances.sh » est disponible en annexe 6 et le programme principal est disponible en annexe 7.

La configuration du Raspberry Pi a pris plus de temps que prévu, le temps de déterminer les bibliothèques les plus adaptées à notre utilisation, de les installer et de les tester avec le reste de notre matériel.

4.2. Interface graphique

À l'instar du reste du programme, l'**interface graphique** est réalisée en **Python**, grâce à la bibliothèque **Pygame**. Cette interface, utilisée sur un **écran tactile** de 3,5" placé sur notre **Raspberry Pi**, doit afficher les informations suivantes :

- **Position** du camion (latitude & longitude)
- **Nombre** de satellites connectés
- **Direction** du camion
- **Puissance** du signal reçu et quantité de **bruits** parasites
- **Orientation** de la parabole



En plus de ces informations affichées en permanence à l'écran, l'interface permet de **piloter** la parabole **manuellement** : des flèches directionnelles font **tourner** et **inclinent** la parabole. Un bouton «Auto» au centre lance la **procédure** de positionnement **automatique** en fonction de la position du camion et de la qualité du signal reçu.

Dans le cas où la **position GPS** ou la **direction** donnée par la boussole ne seraient pas disponibles, le positionnement manuel peut donc être entièrement réalisé à partir de cet **écran**, sans devoir accéder à l'interface du modem et sans monter sur le toit du camion pour **orienter** la parabole comme c'était le cas avant.

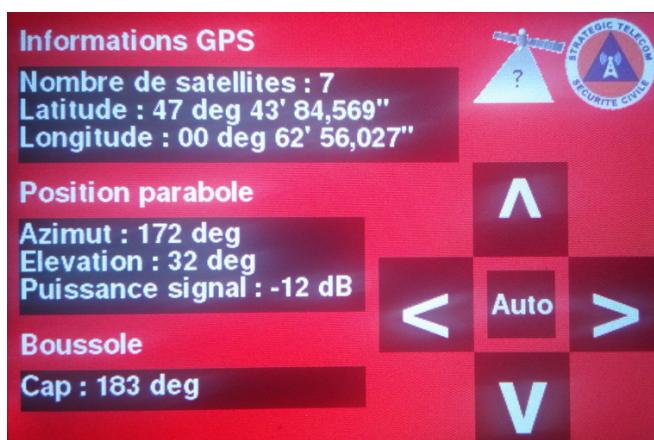


Illustration 24: Interface graphique



Illustration 23: Interface graphique sur l'écran PiTFT

4.3. Carte mère

Tous les composants de notre système, dont le Raspberry Pi, l'écran PiTFT, les contrôleurs et la boussole, ont été regroupés sur une carte électronique qui a été réalisé à l'IUT. Celle-ci a été pensée pour accommoder les points suivants :

- Les câbles doivent sortir sur les côtés de la carte (et donc du futur boîtier) sans qu'il n'y ait de collision entre eux.
- Tous les composants doivent être fixés sur la carte afin d'avoir un unique module facile à installer.
- La boussole doit être éloignée des câbles qui font circuler de fortes puissances, afin de réduire les parasites électromagnétiques qui pourraient la perturber.

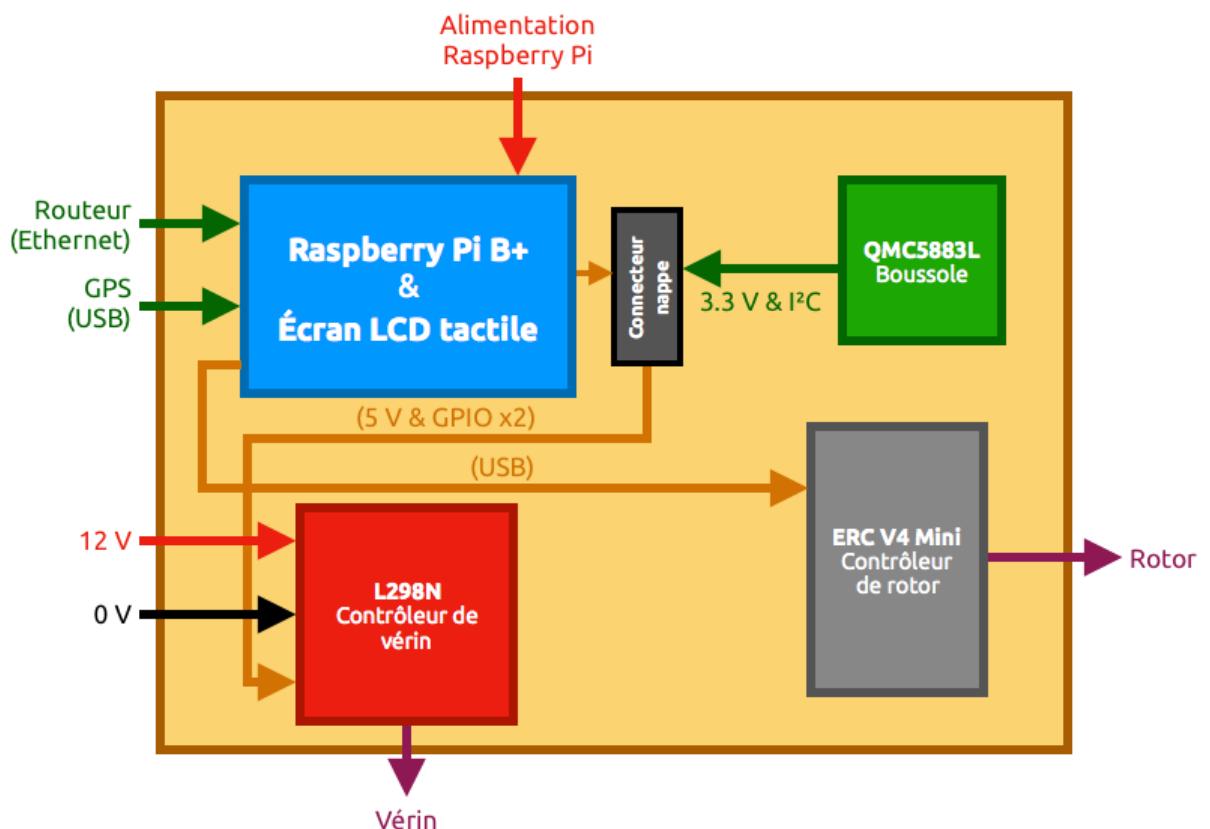


Illustration 25: Schéma d'implantation de la carte électronique



Illustration 26: Carte électronique réalisée à l'IUT

Conclusion

Actuellement, notre système permet de positionner manuellement la parabole au moyen de l'interface graphique, ce qui évite à l'opérateur de monter sur le camion pour régler sa position à la main. Nous arrivons à acquérir la position du camion et à déterminer le satellite auquel se connecter. La récupération de la puissance du signal est opérationnelle et permet de réaliser un balayage afin de trouver le positionnement optimal de la parabole. Le seul problème restant est l'obtention de la direction du camion afin d'effectuer le pré-positionnement qui précède le balayage. Il nous reste à réaliser la mise en commun des programmes des différentes parties ainsi que les essais finaux en situation réelle.

Le projet nous a permis de développer nos compétences, que ce soit en programmation, sur le choix des technologies et du matériel, la mise en place d'un système embarqué avec le Raspberry Pi, le développement d'une interface homme-machine avec son écran tactile ou encore le fonctionnement d'un récepteur GPS.

Nous avons également appris à mieux appréhender la gestion de projet et prévoir les problèmes qui peuvent survenir tels que les délais de livraison retardés ou les ruptures de stocks qui peuvent nous retarder. En travaillant avec la sécurité civile nous avons appris à présenter notre projet devant un client qui à des attentes précises, ce qui donne un enjeu important au projet.

Résumé

Le but de notre projet était d'automatiser le positionnement d'une parabole permettant d'accéder à internet et placée sur le toit d'un camion de la Sécurité Civile de Tours.

Nous utilisons un récepteur GPS afin de récupérer la position du camion. Celle-ci nous permet d'identifier le satellite auquel nous allons nous connecter. Nous obtenons ainsi un angle d'élévation théorique et un angle d'azimut par rapport au nord. Une boussole nous sert à pré-orienter la parabole en prenant en compte l'orientation du camion. Nous récupérons ensuite la valeur de qualité de connexion afin de déterminer son placement optimal.

Afin de positionner la parabole, nous utilisons un rotor, un vérin et leur contrôleur respectif choisis par nos soins, que nous pilotons grâce à un ordinateur embarqué.

À ce jour, nous rencontrons quelques problèmes au niveau du calcul de l'angle que renvoie la boussole. Une fois ce contretemps réglé, il nous restera à mettre toutes nos parties en commun afin de réaliser le programme final.

Nous avons également pris du retard à cause des délais de livraison du rotor et de la carte de contrôle du vérin qui ont été plus longs que prévus. Nous espérons néanmoins que le projet sera finalisé à l'issue des dernières séances.

198 mots

Summary

As part of our project, we worked with “Strategic Telecom Sécurité Civile”. Our work consisted in making a system able to automatically position a satellite dish, located on an intervention truck, used to connect to the Internet. Currently, the dish needs to be oriented manually by getting on top of the truck, which is all but practical for the operators.

To do so, the principal components we use are an actuator and a rotor, both controlled with differents interfaces and protocols by a tiny computer.

What we need to do first is to determine which satellite we must connect to. This depends on the location of the truck that we get from a GPS. We then have to rotate it towards the satellite with the help of an electronic compass. We have to monitor the connection link’s quality between the satellite and the dish, by retrieving this value from the dish’s modem and using it to find the optimal position. The system actually scans the sky, looking for the best signal strength over a predefined area, horizontally and vertically. We stop the automatic positioning process when the signal is the strongest.

The touchscreen has arrows that allow the user to rotate the dish manually however he wants to try to find the best connection by himself.

215 words

Bibliographie

- Documentation Python 2.7 (par Python Software Foundation) dernière MAJ 2 mars 2018)
<https://docs.python.org/2.7/>
- Documentation pySerial
<https://pythonhosted.org/pyserial/>
- Documentation pyGame (dernière MAJ le 28 août 2017)
<https://www.pygame.org/docs/>
- Documentation Requests (par Kenneth Reitz)
<http://docs.python-requests.org/en/master/>
- Documentation Pynmea2 (par Knio, dernière MAJ le 13 octobre 2017)
<https://github.com/Knio/pynmea2>
- Tutoriel d'utilisation de l'écran tactile PiTFT 3,5" (par Adafruit) :
<https://learn.adafruit.com/adafruit-pitft-3-dot-5-touch-screen-for-raspberry-pi/>
- Documentation protocole rotor (par Alba de Schmidt, dernière MAJ le 21 janvier 2017)
http://hintlink.com/manuels/erc_mini
- Carte des satellites
<http://satsig.net/tooway/satellite-dish-pointing-ka-sat-tooway-europe.htm>
- Site duquel nous avons extrait les formules trigonométriques de site et d'azimut
http://www.on4nb.be/content/pointage_satellites.htm
- Bibliothèque RPI.GPIO et tutoriel pour l'utiliser
<https://ouiaremakers.com/posts/tutoriel-diy-gestion-des-gpio-du-raspberry-pi-avec-python3-x>
- Documentation technique de la boussole QMC5883L
<http://osoyoo.com/driver/QMC5883L-Datasheet-1.0.pdf>
- Bibliothèque Python I2C smbus par Bivab
<https://pypi.python.org/pypi/smbus-cffi>
- Tutoriel et bibliothèque pour les liaisons séries
<https://www.google.fr/amp/s/projetsdiy.fr/code-python-lecture-port-serie-raspberry-pi/amp/>

Index des illustrations

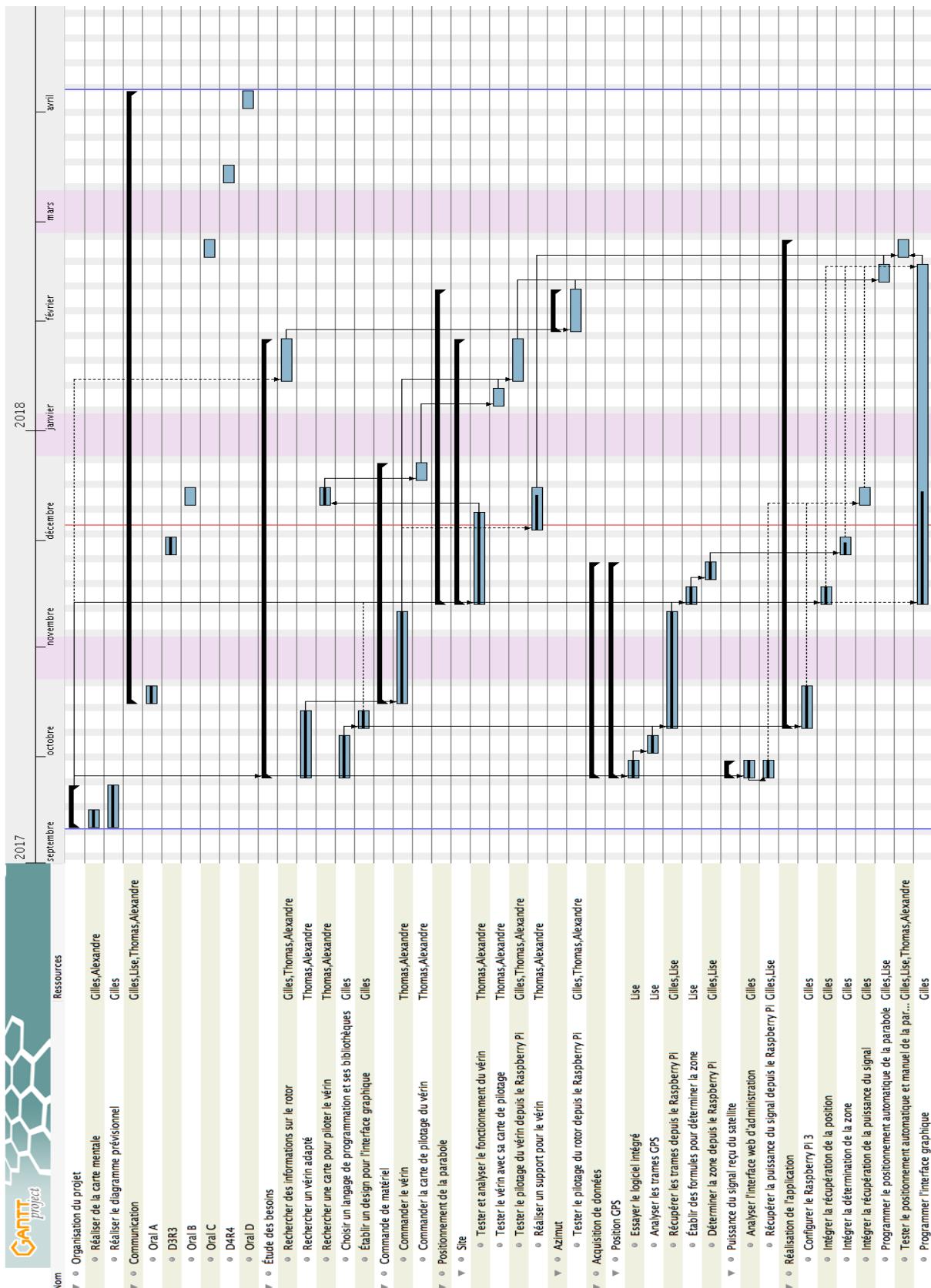
Illustration 1 : Structure du projet.....	5
Illustration 2 : Récepteur GPS utilisé.....	11
Illustration 3 : Visualisation des différentes trames reçues.....	11
Illustration 4 : Essai de récupération des coordonnées GPS.....	12
Illustration 5: Classe facilitant l'utilisation du GPS.....	12
Illustration 6: Zones des satellites en France.....	13
Illustration 7 : Calcul de l'azimuth et de l'élévation.....	14
Illustration 8 : Calcul de l'azimut en tenant compte de l'orientation du camion.....	14
Illustration 9: Page d'administration du modem de la parabole.....	15
Illustration 10: Programme de récupération de la puissance du signal et du bruit.....	16
Illustration 11: Représentation 3D du vérin.....	16
Illustration 12: Schéma de test du capteur.....	18
Illustration 13: Carte d'interface du vérin.....	18
Illustration 14: Relevé de la sortie moteur OUT 1 en fonctions de différentes commandes d'entrée.....	22
Illustration 15: Algorithme de placement du vérin.....	27
Illustration 16: Interface graphique sur l'écran PiTFT.....	29
Illustration 17: Interface graphique.....	29
Illustration 18: Schéma d'implantation de la carte électronique.....	30
Illustration 19: Carte électronique.....	30

Annexes

Index des annexes

1. Diagramme de Gantt du D3R3 (p. 34)
2. Code ce recherche du satellite du calcul de l'azimut (p. 35)
3. Code de placement du vérin (p. 37)
4. Mode d'emploi de la configuration du Raspberry Pi (p. 38)
5. Mode d'emploi de la configuration du Raspberry Pi (p. 40)
6. Script d'installation des dépendances « dependences.sh » (p. 41)
7. Programme principal de l'interface graphique (p. 42)

Annexe 1 : Diagramme de Gantt du D3R3



Annexe 2 : Code de recherche du satellite et du calcul de l'azimut

```
1 import math
2
3 alng= [18.04,20.1,29.69,11.09,24.5,-4.64,6.17,-6.57,15.05,11.61,
4 # longitudes des centres d'ellipses
5 18.5,35.18,-0.18,28.67,21.07,7.27,14.65,-8.1,4.1,30.7,
6 37.82,23.66,-2.87,10.98,17.61,1.21,40.96,7.56,14.06,26.21,
7 20.22,32.59,-1.74,4.41,10.75,28.41,16.61,22.78,34.78,1.7,
8 7.65,13.35,-7.36,19.25,25.17,-8.94,-0.99,16.06,21.65,27.14,
9 4.94,-3.86,10.39,34.21,29.9,37.68,2.31,12.9,24.09,47.37,
10 7.98,40.573,18.66,-6.09,-0.24,26.35,15.6,20.72,-8.5,32.46,
11 -2.75,28.88,23.47,13.03,-5.2,34.61,2.44,25.24,10.03,-6.61,
12 21.13,31.05,53.8]
13 alat= [65.81,62.91,62.79,62.07,60.3,59.39,59.81,56.33,58.92,55.95,
14 # latitudes des centres d'ellipses
15 55.74,53.4,53.92,53.61,53.14,53.28,53.07,52.71,51.58,50.98,
16 50.25,50.42,51.34,50.76,50.57,49.27,48.85,48.58,48.36,47.98,
17 48.02,48.15,47.24,46.6,46.51,46,46.32,46.25,45.97,44.6,
18 44.65,44.27,43.85,44.09,44.06,42.18,43.09,42.57,42.42,42.25,
19 42.44,41.94,42.39,40.9,40.33,40.06,40.9,40.57,40.39,38.90,
20 40.53,39.247,40.47,40.27,39.35,38.65,38.29,38.64,38.91,37.6,
21 37.64,36.54,36.79,36.28,36.44,35.77,35.12,34.83,34.3,32.3,
22 30.83,29.77,25.59]
23 anum= [71,70,75,65,76,45,56,35,64,55, # numéro du spot
24 63,74,34,69,62,44,54,24,33,68,
25 73,61,23,43,53,22,72,32,42,60,
26 52,67,16,21,31,59,41,51,66,15,
27 20,30,9,40,50,6,11,29,39,49,
28 14,8,19,58,48,57,10,18,38,83,
29 13,77,28,5,7,37,17,27,2,47,
30 4,36,26,12,1,46,3,25,79,78,
31 80,81,82]
32 atyp= [2,1,4,4,2,3,2,1,3,1, # type du spot
33 4,3,2,2,3,3,2,3,1,1,
34 4,4,4,4,1,3,3,2,3,3,
35 2,2,1,4,1,4,4,1,1,2,
36 3,2,1,3,2,3,4,1,4,1,
37 1,2,4,4,2,3,3,3,3,4,
38 2,1,2,4,1,4,4,1,2,1,
39 3,3,2,1,1,2,4,1,3,4,
40 3,1,2]
41
42 position=9      ## constante en fonction de la technologie de satellite
                  (correspond à la longitude 9)
43
44 a=2.5          # diamètre moyen x de l'ellipse
45 b=2.7          # diamètre moyen y de l'ellipse
46
47 # Entrée des coordonnées actuelles
48 cy=float(input("Entrer la lattitude : "))
49 cx=float(input("Entrer la longitude : "))
50
51 # application de la formule des ellipses droites
52 resultat=0      # stocke le résultat de la formule
53 affichage=0     # autorise l'affichage si un satellite est détecté
54 i_mem=0         # garde en mémoire le numéro du satellite détecté dans la boucle
55 res_mem=1       # garde en mémoire le résultat de la formule quand un satellite a
                  été détecté
```

```

56 for i in range(83):
57     resultat=((cx-alng[i])/a)**2+((cy-alat[i])/b)**2
58     if resultat<1:
59         affichage=1 # si un satellite est détecté, on affiche un résultat
                      à la fin
60     if resultat<res_mem: # car plus le résultat est proche de 1, plus on est
                      proche du centre de l'ellipse
61         i_mem=i # remplacement par un résultat plus proche
62
63 if (affichage==1): # teste s'il y a quelquechose à afficher ou pas
64     latitude=alat[i_mem]
65     longitude=alng[i_mem]
66     ## calculs avec précision au degré près
67     azimuth=180+180*math.atan(-1*math.tan(math.pi*(position-longitude)/180)/math
68         .sin(math.pi*latitude/180))/math.pi
69     elevation1=math.acos(math.cos(math.pi/180*(longitude-position))*math.cos(math
69         .pi/180*abs(latitude)));
70     elevation=180*math.atan((math.cos(elevation1)-(6378000/(6378000+35786000
70         ))/math.sin(elevation1))/math.pi);
71     ## affichages
72     print("i: " + str (i_mem))
73     print("type : "+ str (atyp[i_mem]))
74     print("Latitude du centre de l'ellipse : " + str(latitude)) # affiche
                      latitude
74     print("Longitude du centre de l'ellipse : " + str(longitude)) # affiche
                      longitude
75     print("Azimuth : " + str(azimuth)) # affiche l'azimuth
76     print("Elévation : " + str(elevation)) # affiche l'élévation
77 else :
78     print("Aucun satellite détecté")
79
80 print(int(azimuth))

```

Annexe 3 : Code du placement du vérin

```
1 import time
2
3 Nb_mesures=0 # nombre de mesures effectuées
4 delai_mesures=0.5 # temps laissé au vérin entre deux mesures
5 mesure_qte=[] # tableau des valeurs de la qualité du signal
6 qte=0 # init qte à une valeur nulle
7 polarity=0 # sens 0 descend, 1 monte
8 erreur=0.5 # écart entre qte_min et qte_max satisfaisant
9 seuil_min=4.0 # seuil pour lequel on commence à enregistrer les valeurs
10 qte_max=25 # init à une grande valeur pour la première itération
11 qte_min=0 # pour la première itération
12 qte=0 # mesure la qualité du signal
13 seuil=seuil_min # init à une valeur fixée pour la première itération
14 capteur_v=0 # capteur de fin de course
15 IN1=0 # Moteur 1
16 IN2=0 # Moteur 2
17
18 ##### Initialisation, départ en position basse #####
19 print("init")
20 while capteur_v==0:
21     IN1=0
22     IN2=1 # commande de descente
23
24 print("Fin de course basse ok")
25
26 while capteur_v==1: # attend de ne plus capter le capteur
27     IN1=1
28     IN2=0 # commande de montée
29
30 print("A commencé à monter")
31
32
33 ##### Début du fonctionnement normal #####
34
35 while 1:
36     print(str(qte_max-seuil))
37     print(str(erreur))
38
39 while (abs(qte_max-qte_min)>erreur):
40
41     polarity=not polarity # le vérin va monter
42     print("sens :" + str(polarity))
43     print("Delai entre les mesures : " + str(round(delai_mesures, 3)) + " secondes")
44 while qte<seuil: # le vérin monte tant que le signal n'est pas correct
45     qte=float(input("Entrer la qualité : "))
46     mesure_qte.append(qte) # récupère la première valeur au dessus du seuil
47     time.sleep(delai_mesures) # temps entre chaque mesure
48
49 while qte>=seuil/1.2: # marge hysteresis
50     qte=float(input("Entrer la qualité : ")) # rentrée à la main mais lecture directe quand mise
      en commun du code
51     mesure_qte.append(qte) # agrandissement du tableau au fur et à mesure
52     time.sleep(delai_mesures) # temps entre chaque mesure
53
54 del mesure_qte[0] # Efface l'ancienne donnée qui est en première position
55 Nb_mesures=len(mesure_qte) # pour que les boucles itèrent autant de fois
56 qte_max=0
57
58
59 for k in range(Nb_mesures):
60     if mesure_qte[k]>=qte_max:
61         qte_max=mesure_qte[k]
62
63 seuil=(qte_max+seuil)/2
64 delai_mesures=delai_mesures/1.2
65 #print("Valeur max de la qte : " + str(qte_max))
66 print("Valeur seuil : " + str(seuil))
```

```
67
68  for m in range(Nb_mesures-1):      # Efface toutes les valeurs du tableau sauf une
69      mesure_qte.pop()
70
71
72  if(abs(mesure_qte[0]-seuil)<=erreur):      # dernier positionnement
73      while(mesure_qte<seuil):
74          polarity=not polarity
75          commande_v=1
76
77  if(abs(mesure_qte[0]-seuil)<2*erreur):      # en cas d'erreur
78      break
79
80 commande_v=0      ## on arrête le vérin quand le positionnement est fini
81
```

Annexe 4 : Commandes secondaires de configuration de l'ERC

Yaesu GS232 A and B

The only difference in A and B is the format how a position is returned from the ERC

Command to ERC	Description	Returned from ERC
A<cr>	Stop rotation azimuth	<cr>
C<cr> (in GS232A-mode)	Request position azimuth	+0aaa<cr><lf>
C<cr> (in GS232B-mode)	Request position azimuth	AZ=aaa<cr><lf>
C2<cr> (in GS232A-mode)	Request position azimuth + elevation	+0aaa+Deee<cr><lf>
C2<cr> (in GS232B-mode)	Request position azimuth + elevation	AZ=aaa<s><s>EL=eee<cr><lf>
L<cr>	Rotate CCW 1 st axis	<cr>
Maaa<cr>	Rotate azimuth to aaa	<cr>
R<cr>	Rotate CW 1 st axis	<cr>
S<cr>	Stops rotation	<cr>
Waaa<s>eee<cr>	Rotate azimuth to aaa and Rotate elevation to eee	<cr>

Commands to configure and calibrate the ERC-Mini:

API ERC-Mini V12	to ERC-Mini			
Command	ASCII			
Set Baudrate	s B A U 9 6 0 0 (cr)			
Set Protocol	s P R O 0 0 0 1 (cr)			
Set Delay before Move	s D B M 1 0 0 0 (cr)			
Set Programmable Stop Right	s P S R 0 0 0 0 (cr)			
Set Programmable Stop Left	s P S L 0 0 0 0 (cr)			
Set Tolerance	s T O L 0 0 0 2 (cr)			
Set Antenna-Offset	s A O F 0 0 0 0 (cr)			
Set Speedfunction	s S P F 0 0 0 1 (cr)			
Set Speed angle	s S P A 0 0 0 3 (cr)			
Set Speed at Low-speed	s S P L 0 0 0 1 (cr)			
Set Speed at High-speed	s S P H 0 0 0 0 (cr)			
Set Calibration Left	s C A L 0 0 0 0 (cr)			
Set Calibration Right	s C A R 0 3 6 0 (cr)			
Set Factory Default Values	s F D V 0 0 0 0 (cr)			

Annexe 5 : Mode d'emploi de la configuration du Raspberry Pi

Installation

=====

Sur un ordinateur *nix

Flasher le système d'exploitation sur la carte microSD du Raspberry Pi
(Raspbian Jessie Lite pour PiTFT 3,5" <https://learn.adafruit.com/adafruit-pitft-3-dot-5-touch-screen-for-raspberry-pi/easy-install>) :

```
sudo dd if=2016-11-08-pitft-35r.img of=/dev/diskX bs=1M
```

Pour utiliser la connexion SSH, créer un fichier «ssh» à la racine de la partition «boot» créée :

```
touch ssh
```

Alimenter le Raspberry Pi

Mettre les fichiers «dependances.sh» et «parabole.py» sur le Raspberry Pi, par exemple en utilisant la connexion SSH avec l'adresse IP de Raspberry Pi 192.168.2.3 :

```
scp dependances.sh parabole.py pi@192.168.2.3:~/
```

Mot de passe par défaut : raspberry

Sur le Raspberry Pi (en SSH ou en direct)

sudo raspi-config
«Expand filesystem»

```
sudo chmod +x dependances.sh  
sudo ./dependances.sh
```

Pour calibrer l'écran tactile :

```
sudo TSLIB_FBDEVICE=/dev/fb1 TSLIB_TSDEVICE=/dev/input/touchscreen  
ts_calibrate
```

Pour tester la calibration de l'écran tactile:

```
sudo TSLIB_FBDEVICE=/dev/fb1 TSLIB_TSDEVICE=/dev/input/touchscreen ts_test
```

Utilisation

=====

Exécution du script :

```
sudo ./parabole.py
```

Annexe 6 : Script d'installation des dépendances « dependences.sh »

```
#!/bin/bash

# Mises a jour initiales
sudo apt-get update
sudo apt-get -y --force-yes upgrade

# Installation de Python et Pip
sudo apt-get -y --force-yes install python-pip
sudo apt-get -y --force-yes install python-pygame

# Dépendances pour le bon fonctionnement de l'écran tactile avec
Pygame
# Activation des sources de paquets Wheezy
echo "deb http://archive.raspbian.org/raspbian wheezy main
" > /etc/apt/sources.list.d/wheezy.list
# Réglage de la source des paquets par défaut comme wheezy
echo "APT::Default-release \"oldstable\";
" > /etc/apt/apt.conf.d/10defaultRelease
# Réglage de la priorité de libsdl wheezy supérieure à celle du paquet
jessie
echo "Package: libsdl1.2debian
Pin: release n=jessie
Pin-Priority: -10
Package: libsdl1.2debian
Pin: release n=wheezy
Pin-Priority: 900
" > /etc/apt/preferences.d/libsdl
# Installation de SDL 1.2
apt-get update
apt-get -y --force-yes install libsdl1.2debian/wheezy

# Installation de la bibliothèque Python PySerial (communication avec
le GPS & l'ERC V4)
sudo python -m pip install pyserial
# Installation de la bibliothèque Python Pynmea2 (exploitation des
trames GPS)
sudo pip install pynmea2
# Installation de la bibliothèque Python bitstring (conversion des
valeurs de la boussole)
sudo pip install bitstring

# Installation de la bibliothèque Python GUI PGU
wget https://storage.googleapis.com/google-code-archiver-
downloads/v2/code.google.com/pgu/pgu-0.18.zip
sudo pip install pgu-0.18.zip

# Nettoyage
rm pgu-0.18.zip
```

Annexe 7 : Programme principal de l'interface graphique

```
#!/usr/bin/python
# coding: utf-8

import pygame
from pygame.locals import *
import os
import sys
from time import sleep
from helpers import *
import serial
import pynmea2
import string

class GPS(object):

    def __init__(self, path):
        self.path = path

        try:
            self.gps = serial.Serial(path, 4800)
            self.gps.readline() # On se place au début d'une trame
        except serial.SerialException:
            # Le GPS n'est pas connecté
            self.gps = None

    def __del__(self):
        if self.gps is not None:
            self.gps.close()

    def getInfos(self):
        if self.gps is not None:
            while self.gps.in_waiting > 0:
                serialBuffer = self.gps.readline()
                if serialBuffer[0:6] == "$GPGGA":
                    self.data = pynmea2.parse(serialBuffer)

            return self.data
        else:
            return None

    # Check if executed with root access
    if not "SUDO_UID" in os.environ.keys():
        print("Veuillez exécuter ce programme avec les droits root : sudo ./parabole.py")
        sys.exit(1)
```

```

# GPS initialization
gps = GPS('/dev/ttyUSB0')

# Colors
WHITE = (255, 255, 255)

# Screen setup
os.putenv("SDL_FBDEV", "/dev/fb1")
os.putenv("SDL_MOUSEDRV", "TSLIB")
os.putenv("SDL_MOUSEDEV", "/dev/input/touchscreen")

# Pygame initialization and configuration
pygame.init()
pygame.mouse.set_visible(False)
clock = pygame.time.Clock()
lcd = pygame.display.set_mode((480, 320)) # Taille de la fenêtre
lcd.fill((229, 50, 46)) # Couleur d'arrière-plan
pygame.display.update()
font = pygame.font.Font(None, 30)

# Format des boutons : "Label": ((x, y), (w, h), TaillePolice, callback())
buttonsDef = {
    ("^", (340, 110), (70, 70), 100),
    (">", (410, 180), (70, 70), 100),
    ("v", (340, 250), (70, 70), 100),
    ("<", (270, 180), (70, 70), 100),
    ("Auto", (350, 190), (50, 50), 30),
}

buttons = []
for button in buttonsDef:
    btn = makeButton(button)
    buttons.append(btn)
    lcd.blit(btn[0], btn[1])

# GPS informations
gpsInfosLabel = pygame.font.Font(None, 30).render("Informations GPS", True, WHITE)
lcd.blit(gpsInfosLabel, gpsInfosLabel.get_rect().move((10, 10)))
gpsInfosSurface = pygame.Surface((320, 70))

# Antenna's position and signal strength
antennaPosLabel = pygame.font.Font(None, 30).render("Position parabole", True, WHITE)
lcd.blit(antennaPosLabel, antennaPosLabel.get_rect().move((10, 120)))
antennaPosSurface = pygame.Surface((250, 70))

# Compass
compassLabel = pygame.font.Font(None, 30).render("Boussole", True, WHITE)
lcd.blit(compassLabel, compassLabel.get_rect().move((10, 230)))
compassSurface = pygame.Surface((250, 30))

# Logo
logo = pygame.image.load("images/StrategicTelecom.png").convert_alpha()

```

```

logo = pygame.transform.scale(logo, (76, 76))
rect = logo.get_rect()
rect = rect.move((405, 0))
lcd.blit(logo, rect)

# Zone
zone = pygame.image.load("images/zoneInconnue.png").convert_alpha()
zoneRect = zone.get_rect()
zoneRect = zoneRect.move((340, 10))
lcd.blit(zone, zoneRect)

pygame.display.update()

zone2 = 0
timer = 0
showZone = False

while True:
    clock.tick(1000)
    timer = timer + 1

    # Zone image
    if (zone2 == 0):
        if timer % 10 == 0:
            if showZone:
                showZone = False
                pygame.draw.rect(lcd, (229, 50, 46), zoneRect)
            else:
                showZone = True
                lcd.blit(zone, zoneRect)

    # Get GPS informations
    gpsData = gps.getInfos()

    if gpsData is not None:
        print("Lat : " + "%02d%02d'%07.4f\"" % (gpsData.latitude,
        gpsData.latitude_minutes, gpsData.latitude_seconds))
        print("Lon : " + "%02d%02d'%07.4f\"" % (gpsData.longitude,
        gpsData.longitude_minutes, gpsData.longitude_seconds))
        print("Fix" if gpsData.gps_qual else "No fix")
        print("Sats : " + gpsData.num_sats)
        print("")

    gpsInfosSurface.fill((127, 0, 0))

    gpsInfosText = font.render("Nombre de satellites : " + gpsData.num_sats, True,
    WHITE)
    gpsInfosSurface.blit(gpsInfosText, gpsInfosText.get_rect())

    if gpsData.gps_qual:
        gpsInfosText = font.render("Latitude : %02d deg %02d' %07.4f\"" %
        (gpsData.latitude, gpsData.latitude_minutes, gpsData.latitude_seconds), True,

```

```

WHITE)
gpsInfosSurface.blit(gpsInfosText, gpsInfosText.get_rect().move((0, 20)))

gpsInfosText = font.render("Longitude : %02d deg %02d' %07.4f\"" %
(gpsData.longitude, gpsData.longitude_minutes, gpsData.longitude_seconds), True,
WHITE);
gpsInfosSurface.blit(gpsInfosText, gpsInfosText.get_rect().move((0, 40)))
else:
gpsInfosText = font.render("Pas de fix", True, WHITE)
gpsInfosSurface.blit(gpsInfosText, gpsInfosText.get_rect().move((0, 30)))

gpsInfosRect = gpsInfosSurface.get_rect().move((10, 40))
lcd.blit(gpsInfosSurface, gpsInfosRect)

pygame.display.update()

# Antenna informations

antennaPosSurface.fill((127, 0, 0))

antennaPosText = font.render("Azimut : {} deg".format(azimut), True, WHITE)
antennaPosSurface.blit(antennaPosText, antennaPosText.get_rect())

antennaPosText = font.render("Elevation : {} deg".format(elevation), True, WHITE)
antennaPosSurface.blit(antennaPosText, antennaPosText.get_rect().move((0, 20)))
antennaPosText = font.render("Puissance signal : {} dB".format(rxPower), True,
WHITE)
antennaPosSurface.blit(antennaPosText, antennaPosText.get_rect().move((0, 40)))

lcd.blit(antennaPosSurface, antennaPosSurface.get_rect().move((10, 150)))

# Compass informations

compassSurface.fill((127, 0, 0))
compassText = font.render("Cap : {} deg".format(direction), True, WHITE)
compassSurface.blit(compassText, compassText.get_rect())

lcd.blit(compassSurface, compassSurface.get_rect().move((10, 260)))

# Scan touchscreen events

for event in pygame.event.get():

if(event.type is MOUSEBUTTONDOWN):
pos = pygame.mouse.get_pos()
print(pos)

if (gpsInfosRect.collidepoint(pos)):
print("Bouton")

for button in buttons:

if button[1].collidepoint(pos):
print(button[2])

```


Université François Rabelais de Tours

Institut Universitaire de Technologie de Tours

Département Génie Électrique et Informatique Industrielle



D4R4 – Rapport de fin de projet

Automatisation du placement d'une parabole



Lise CHAUVIN

Gilles DEVILLERS

Thomas GRAGEON

Alexandre MINOT

Groupe RoboTic 1

Groupe de projet n°304

Promotion 2016-2018

Coach : Vincent GRIMAUD

Jury :

Référent professionnel : Christophe TAILLIEZ