

Université De Tours

Institut Universitaire technologique de Tours

Département Génie Électrique et Informatique Industrielle

Automatisation du placement d'une parabole et d'une antenne râteau pour la station spatiale internationale

ISS



D4R4

Thomas BARRIER

Olivier BERTHIAU

Augustin DECAUX

Grégoire REBOUL

Groupe 302

2017/2019

Projet tutoré 2^{ème} année

Mme AUGER Véronique

Coach : Mr GRIMAUD Vincent

Christophe Tailliez

Université De Tours

Institut Universitaire technologique de Tours

Département Génie Électrique et Informatique Industrielle

Automatisation du placement d'une parabole et d'une antenne râteau pour la station spatiale internationale

ISS



D4R4

Thomas BARRIER
Olivier BERTHIAU
Augustin DECAUX
Grégoire REBOUL
Groupe 302
2017/2019

Projet tutoré 2^{ème} année
Mme AUGER Véronique
Coach : Mr GRIMAUD Vincent
Christophe Tailliez

Sommaire

Introduction.....	5
Diagramme de Gantt.....	6
Carte mentale	7
Budget	9
1. Reprise du sujet.....	10
1.1. Câblage du système.....	10
1.2. Tests effectués.....	12
2. Acquisition des informations	13
2.1. Utilisation du récepteur GPS	13
2.2. Algorithme de positionnement automatique de la parabole.....	16
2.3. Boussole	19
3. Pilotage	23
3.1. Pilotage du rotor en angle	23
3.2. Pilotage du vérin en angle	28
3.3. Interface Homme Machine.....	32
Conclusion	36
Résumé	37
Bibliographie	38
Index des mots clés.....	Erreur ! Signet non défini.
Index des illustrations.....	40
Annexes.....	42



Introduction

Dans le cadre du projet Super Pouet, notre groupe composé de Olivier Berthiau, Grégoire Reboul, Augustin Decaux et Thomas Barrier travaillons sur le placement automatique d'une parabole pour Strategic Telecom Sécurité Civile.

Cette parabole sera située sur un camion, nous devrons alors mettre en place un algorithme permettant le positionnement automatique de la parabole quelle que soit sa position géographique.

Le système est équipé d'un vérin permettant de contrôler la parabole verticalement, mais aussi d'un rotor pouvant diriger la parabole horizontalement. Le code effectué sera programmé en langage Python et exécuté sur une Raspberry Pi¹.

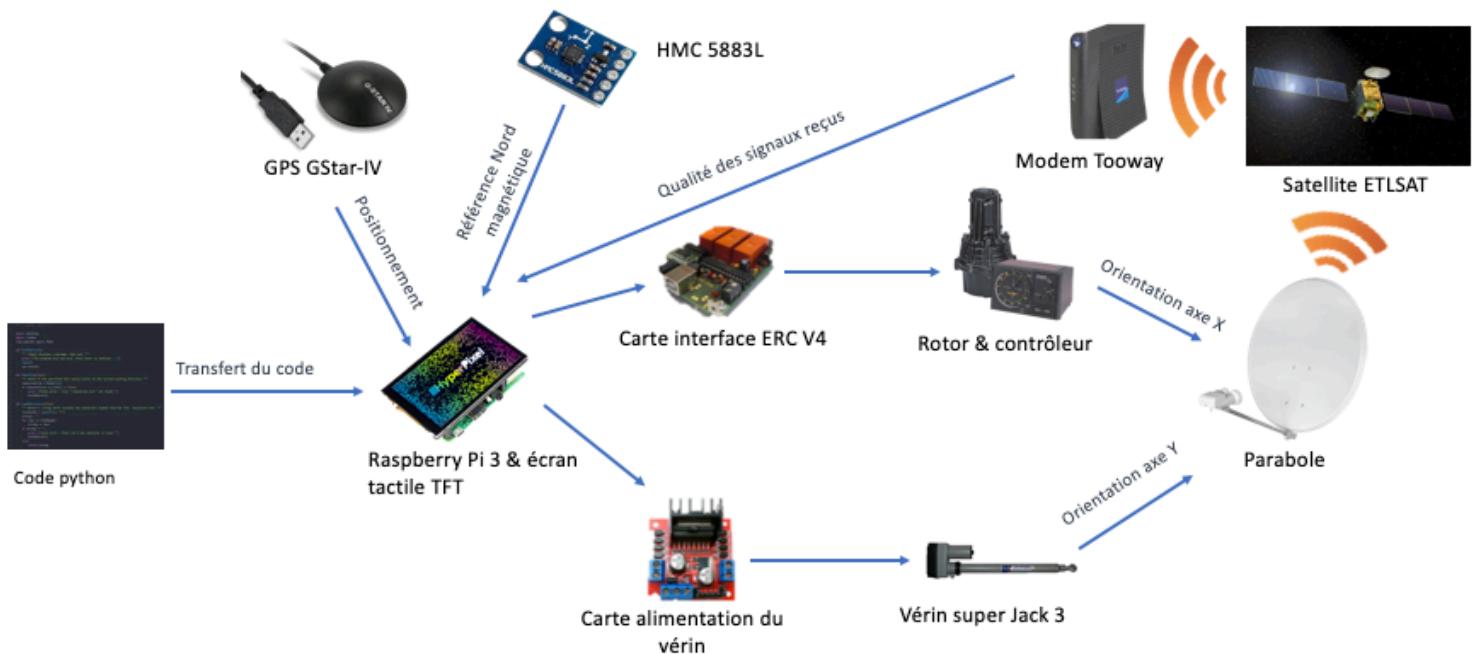


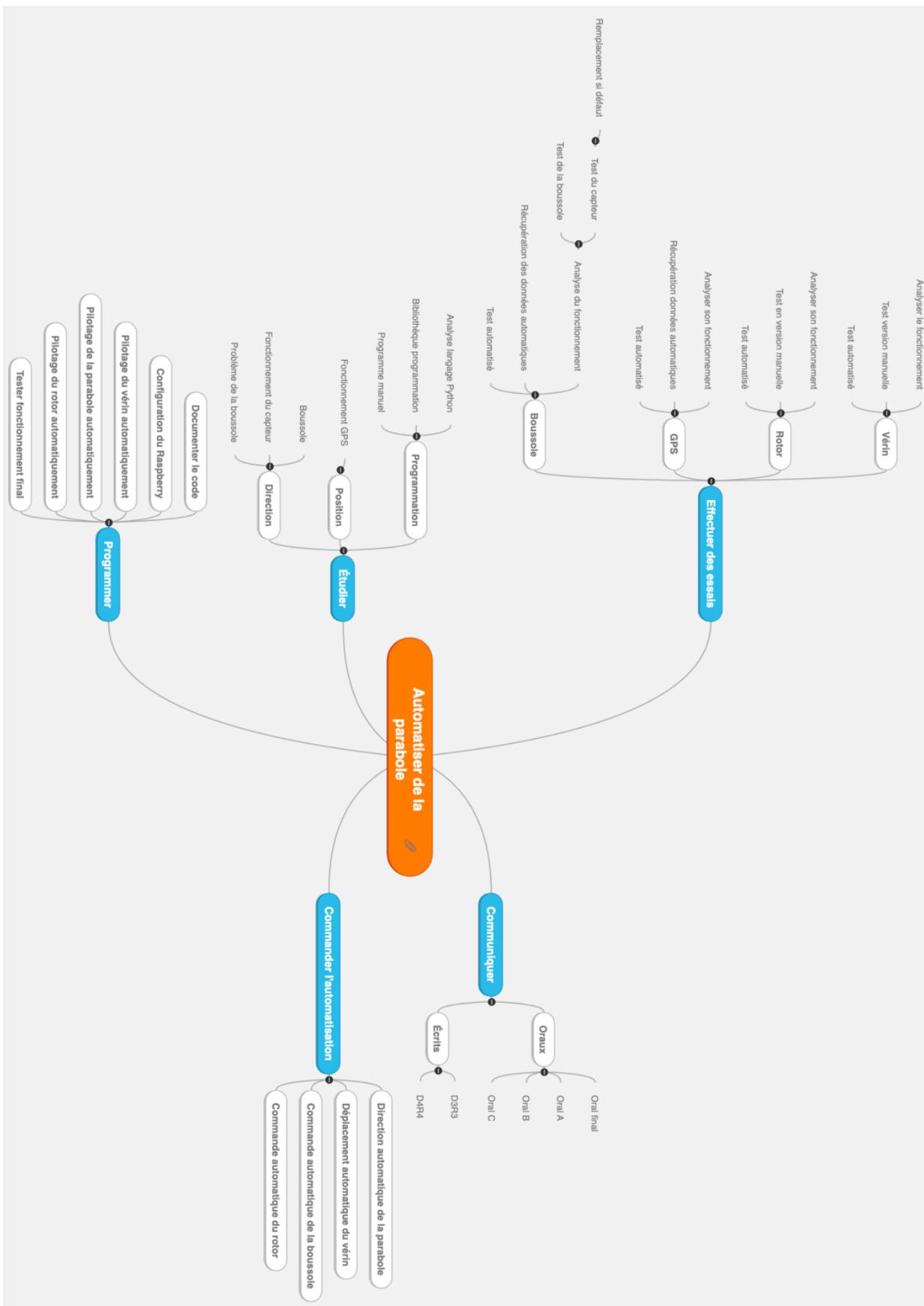
Illustration 1 : Schéma structurel du système

¹ La Raspberry Pi est un nano-ordinateur

Diagramme de Gantt



Carte mentale



Par rapport au diagramme de Gantt présenté dans le D3R3 (voir annexe 1), nous avons pris du retard suite aux différents problèmes rencontrés. Nous détaillerons ces problèmes dans le développement de notre rapport.

De plus, des tâches au fur et à mesure de l'avancée du projet se sont rajoutées afin de rendre le système fonctionnel et performant comme le pilotage du rotor en degré, les modifications apportées aux différentes boussoles numériques testées et la reprise complète du câblage système avec rédaction de documents techniques.

Olivier Berthiau s'est occupé du développement sous Raspberry Pi en langage Python d'un algorithme permettant d'avoir l'azimut² approximatif pour connecter la parabole au satellite.

Il a également, avec l'aide d'**Augustin Decaux** développé un code permettant de commander le rotor en angle et donc commencer l'automatisation.

Thomas Barrier a pris en charge les tests de la boussole permettant d'avoir une référence grâce au nord magnétique³. Différents codes ont été réalisés afin de pallier aux problèmes de fiabilité de la boussole et une solution a donc dû être trouvée. Nous y reviendrons dans la partie « Boussole » de notre rapport.

De plus il s'est occupé de la rédaction des documents techniques concernant le câblage du système et également celui de la boussole.

Gregoire Reboul quant à lui s'est occupé de la partie IHM⁴. Il a donc développé sous Python une interface reprenant les programmes afin de rendre le système plus ergonomique pour l'utilisateur. Il a également aidé sur la partie boussole pour les différents tests menés et sur la réflexion d'une nouvelle solution technique.

² Azimut : angle dans le plan horizontal

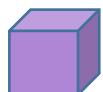
³ Le nord magnétique est un point défini sur la surface de la Terre où le champ magnétique terrestre pointe vers le bas

⁴ IHM : Interface Homme Machine

Budget

Une grande partie du matériel nous a été fournie par Strategic Telecom. Nous avons donc repris le matériel de l'année dernière avec comme seul changement la boussole. Nous avons également revu l'abonnement mensuel pour le module Tooway⁵.

Désignation	Prix TTC en €
Raspberry Pi 2B	35€
Alimentation Raspberry Pi	10€
Carte micro SD Kingstone 16 Go	8€
ERC V4 USB	110€
Ecran tactile TFT 3.5 pouces	37€
Rotor Yaesu G-1000DX & contrôleur	490€
GPS GSTAR IV	39€
Kit satellite Tooway (parabole, modem, support, activation)	375€
Abonnement Tooway 25	45€/mois
Vérit superJack III	45€
Contrôleur vérit L298N	6€
Boussole CMPS 12	32€
Total	1204€



Matériel fourni de l'année dernière



Matériel acheté cette année



Budget final du projet

⁵ Tooway est un service proposant la connexion internet par satellite

1. Reprise du sujet

Avant de nous consacrer pleinement dans l'automatisation de la parabole, nous avons d'abord dû étudier les différents programmes effectués par le groupe précédent. Pour cela nous avons fait des tests à l'aide de la Raspberry Pi et de l'IHM présent dans le matériel fourni.



Illustration 1 : IHM réalisée par le groupe précédent

1.1. Câblage du système

Une fois le système mis à notre disposition, nous avons commencé l'étude du fonctionnement de celui-ci. Le branchement en USB du contrôleur vérin L298N et de la carte ERC Mini pour le rotor sur la Raspberry étaient nécessaire afin de pouvoir les piloter avec l'interface graphique.

Après avoir effectué le câblage, nous avons dû nous atteler au branchement du vérin sur le contrôleur. Or, aucune documentation technique nous avait été fourni.

La rédaction d'une documentation technique lors de la mise en œuvre d'un système est primordiale pour la bonne compréhension de ce dernier. Après l'étude et la lecture des documents constructeurs, nous avons refait l'intégralité du câblage pour faire communiquer le vérin avec la Raspberry.

Étant handicapé par le manque de documentation technique, nous avons décidé dans un premier temps de mettre le câblage du vérin avec le contrôleur sur papier pour pouvoir faciliter la tâche à un éventuel groupe qui prendrait la relève de notre projet.

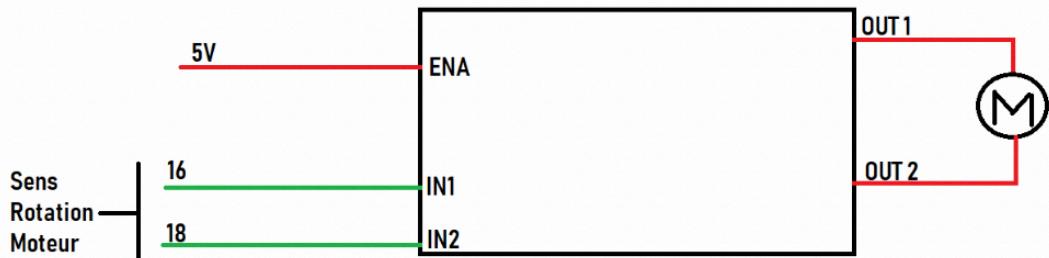


Illustration 2 : Schéma de câblage du vérin avec le contrôleur

Broches	Fonction
OUT1 et OUT2	Liaison au moteur 1
IN1 et IN2	Sens de rotation du moteur
Entrée + et GND	Alimentation du moteur vérin
5V	Alimentation de la partie commande

La carte est alimenté en USB au port 5 volts de la Raspebrry Pi. Nous nous servons des entrées IN1 et IN2 directement branchés aux GPIO 16 et 18 de la Rapsberry pour pouvoir piloter le moteur du vérin dans le sens horaire et anti horaire. De plus, le moteur vérin est alimenté par 2 batteries de 12V en série pour lui fournir une tension d'alimentation de 24 volts.

Concernant le rotor, celui-ci dispose d'une commande manuelle qui nous a été fournie par Strategic Telecom. Le rotor est relié à un contrôleur qui renvoi l'angle exact de rotation du rotor et qui permet le pilotage manuel.

Ce contrôleur sera relié à la carte ERC Mini par un câble antenne. Pour pouvoir communiquer avec la Raspberry, l'ERC Mini nécessite une liaison série par USB.

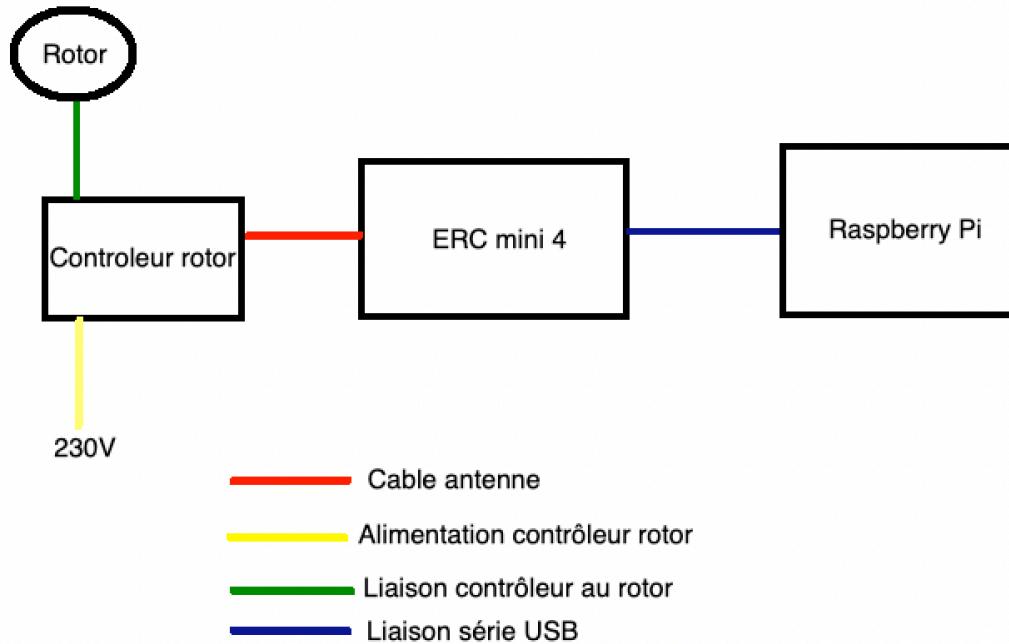


Illustration 3 : Schéma de câblage du rotor à la Raspberry

Une fois le câblage effectué et le système fonctionnel, nous avons commencés les tests des programmes de l'an passé.

1.2. Tests effectués

La reprise des éléments de l'année dernière a commencé par des tests permettant d'évaluer le fonctionnement du système.

Nous avons décidé de lire la carte SD du groupe de l'an passé afin de tester l'interface homme-machine et le code du projet. Cependant, une erreur de script rendait le lancement du code final « Parabole.py » impossible.

Nous avons pris la décision d'installer un programme nommée « Install.sh ».

Ce programme est une commande shell⁶ permettant d'installer des dépendances et de configurer l'interface graphique. Malheureusement cet

⁶ Un script shell permet d'automatiser une série d'opération et d'exécuter les commandes de manière sequentielle

essai n'a pas été concluant. Suite à ce problème, nous nous sommes lancés dans la réalisation d'une nouvelle interface graphique que nous présenterons ultérieurement dans le rapport.

Néanmoins, il a été possible de récupérer les programmes de commandes manuelles afin de les intégrer dans notre interface.

2. Acquisition des informations

2.1. Utilisation du récepteur GPS

Dans le cadre de l'automatisation de la parabole, nous devons connaître la position géographique du camion. Pour cela, nous utilisons un module GPS qui nous a été fourni.

Ce module GPS est un G-Star IV de chez GlobalSat et se connecte à la Raspberry Pi par un port USB. Cet appareil transmet des trames avec le protocole de communication standardisé NMEA 0183. Une trame NMEA⁷ est toujours composée de caractères ASCII⁸ donc elle est directement lisible par la Raspberry.



Illustration 4 : Module GPS G-Star IV

Avant de réaliser un programme en Python, nous avons d'abord préféré observer les trames envoyées par le module GPS. Pour cela, nous avons utilisé le moniteur série de l'IDE Arduino qui permet de lire des trames provenant d'un port série.

⁷ La norme NMEA (National Marine Electronics Association) est un protocole de communication entre équipements marins

⁸ L'ASCII est une norme de codage de caractères informatique

```
$GPRMC,203901.000,A,4720.7766,N,00044.5702,E,0.04,221.17,100319,,,A*6F  
$GPGGA,203902.000,4720.7766,N,00044.5701,E,1,06,1.4,99.5,M,47.9,M,,0000*68  
$GPGSA,A,3,16,20,26,21,10,08,,,,,,3.4,1.4,3.1*39  
$GPRMC,203902.000,A,4720.7766,N,00044.5701,E,0.03,221.17,100319,,,A*68  
$GPGGA,203903.000,4720.7766,N,00044.5700,E,1,06,1.4,99.2,M,47.9,M,,0000*6F
```

Illustration 5 : Trames renvoyées par le module GPS

Comme on peut le voir sur l'illustration 5, le module GPS envoie plusieurs trames à la fois que nous allons expliquer. Tout d'abord, le type d'équipement est défini par les deux caractères qui suivent le caractère « \$ », ici c'est « GP » ce qui signifie c'est une trame provenant d'un GPS. Les trois caractères suivant définissent le type de trame :

- Trame GGA : fournit l'heure, la longitude, la latitude, l'altitude, le nombre de satellites utilisés pour calculer les coordonnées, la précision de la mesure
- Trame GLL : fournit l'heure, la longitude, la latitude
- Trame GSA : indique le nombre de satellites utilisés pour calculer les coordonnées et la précision de la mesure
- Trame GSV : indique le nombre de satellites utilisés pour calculer les coordonnées, la qualité du signal de chaque satellite, son azimut et son élévation⁹

Pour chaque trame, les arguments sont séparés par une virgule. Pour la suite, nous ne garderons que la trame GGA car c'est celle qui est le plus souvent utilisée pour déterminer la position courante d'un récepteur GPS.

Après avoir effectués les tests sur le moniteur série, nous nous sommes concentré sur la programmation Python du module GPS. Nous avons réutilisé le code réalisé par le groupe précédent car celui-ci fonctionne parfaitement. Nous avons néanmoins apporté quelques modifications ayant pour but de renvoyer les coordonnées en dehors de la fonction et d'y avoir accès dans d'autres fonctions.

⁹ Elévation : angle dans le plan vertical

```

def gps():
    gps = serial.Serial('/dev/ttyUSB0', 4800) #configuration du port serie avec un baudrate 4800
    nmea = gps.readline() #on lis la ligne ou est present la trame NMEA
    if nmea[0:6] == "$GPGGA": #On compare la trame GPGGA
        print nmea
        gpsData = pynmea2.parse(nmea) #On parcourt la trame qu'on affecte a une variable gpsDATA
        lata = float(gpsData.latitude) #permet de retrouver dans notre trame la latitude et la traduire
        longitu = float(gpsData.longitude) #permet de retrouver dans notre trame la longitude et la traduire
        print 'latitude = ' , lata
        print 'longitude = ' , longitu
    return lata, longitu #on retourne ces deux variables pour le futur programme
    gps.close()

```

Illustration 6 : Code de la classe "gps"

Nous allons expliquer le fonctionnement de la classe¹⁰ « gps » présentée ci-dessus. Il faut tout d'abord initialiser la connexion entre le module GPS et la Raspberry. Nous utilisons la librairie « Serial » pour laquelle il faut spécifier le chemin d'accès au port USB ainsi que la valeur du baudrate¹¹ nécessaire pour communiquer.

```

gps = serial.Serial('/dev/ttyUSB0', 4800)

```

Illustration 7 : Création d'un objet de type Serial

Par la suite, nous lisons les trames envoyées par le module GPS et vérifions si la trame reçue commence par « \$GPGGA » car nous souhaitons extraire seulement la trame GGA.

```

nmea = gps.readline() #on lit la trame
if nmea[0:6] == "$GPGGA":

```

Illustration 8 : Lecture et analyse des trames reçues

Pour continuer, nous découpons la trame grâce à la librairie « pynmea2 » qui est parfaitement adaptée pour classer automatiquement les données de trames GGA.

```

gpsData = pynmea2.parse(nmea)

```

Illustration 9 : Découpage de la trame

¹⁰ En informatique, une classe permet de regrouper dans une même entité des données et des méthodes (aussi appelé fonctions)

¹¹ Le baudrate définit la vitesse de transmission en bits par seconde d'une liaison série

Ainsi, grâce à cette librairie, nous pouvons retrouver facilement au sein de notre trame les valeurs de latitude et de longitude.

```
lata = float(gpsData.latitude) #pe
longitu = float(gpsData.longitude)
```

Ces deux variables sont ensuite retournées pour être utilisées dans l'algorithme de placement automatique de la parabole que nous allons expliquer.

2.2. Algorithme de positionnement automatique de la parabole

On souhaite maintenant mettre en place un algorithme permettant de calculer l'azimut et l'élévation de notre antenne en fonction de la position géographique. Pour déterminer la latitude et la longitude du camion, nous exploiterons le contenu de la trame GGA du récepteur GPS présenté précédemment.

Pour réaliser l'algorithme décrit ci-dessus, nous avons utilisé un site internet appartenant à l'entreprise Tooway qui est spécialisée dans la connexion internet par satellite. Une capture d'écran du site web est présentée ci-dessous.

The screenshot shows the Tooway KA-Sat Finder website. At the top, there are icons for iOS and Android, and a language selection bar with various flags. Below the header, there are two main sections: "Carte" and "Coordonnées". The "Coordonnées" section is active, containing fields for "Adresse", "Ou", "Latitude", and "Longitude", along with a "Calculer" button and an "Imprimer" button. To the right of this is a "Résultats" panel showing "Elevation" (35.00), "Azimut géographique" (168.88), and a "Configuration du spot" diagram. A note at the bottom left of the "Coordonnées" section states: "Cette zone n'est pas couverte par le satellite".

Copyright (c) System Integration Team 2011@Eutelsat Portail Tooway Installers - Vidéos youtube - Eutelsat apps - Campus Tooway - Facebook FR - EN - Twitter FR - EN

Illustration 10 : Site web contenant l'algorithme

Illustration 11 : Exemple de détermination de l'azimut et de l'élévation

Comme on peut le voir, il suffit de rentrer la latitude et la longitude et le site internet nous renvoie automatiquement l'azimut et l'élévation pour le placement de la parabole. On obtient également la configuration du spot qui correspond à la zone dans laquelle se trouve le satellite. En France, il y a 4 zones de satellites représentées ci-dessous et l'algorithme choisit toujours le satellite situé dans la zone la plus optimale.

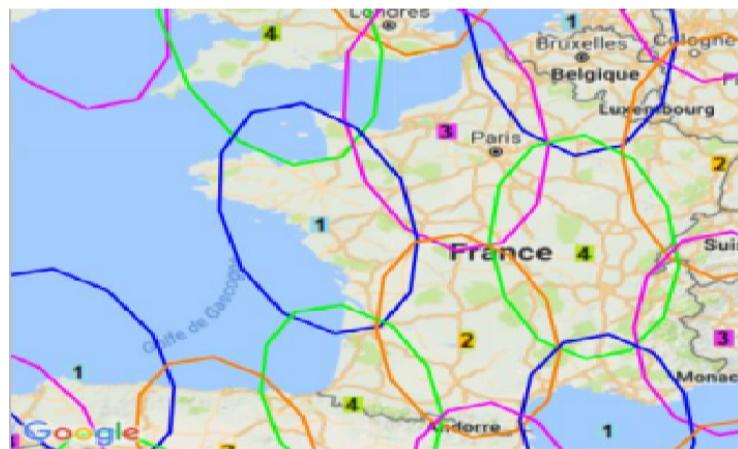


Illustration 12 : Carte des zones de satellites en France

Avant d'implémenter l'algorithme à notre projet, nous nous sommes entretenus avec Monsieur Christophe TAILLIEZ, commanditaire du projet, qui nous a confirmé la fiabilité de ce dernier. Nous avons donc dû extraire les fichiers du site web pour permettre d'utiliser l'algorithme hors ligne et l'intégrer dans notre système embarqué.

Après une étude approfondie des différents fichiers, nous avons trouvé que l'algorithme était présent dans un fichier nommé « script.js ». L'extension « .js » signifie que c'est un fichier écrit en JavaScript or nous utilisons une Raspberry Pi qui utilise le langage Python. Nous devons donc traduire le programme en Python pour pouvoir l'exécuter.

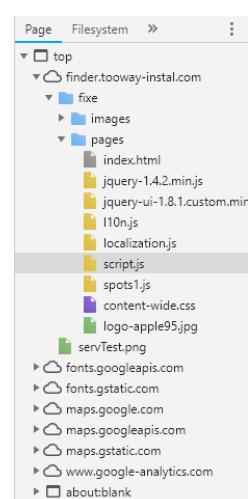


Illustration 13 : Liste des fichiers contenus dans le site web

```

var azi;
var ele;
var g = (-9 + n);
var grad = (g / 57.29578);
var lrad = (l / 57.29578);
azi = (3.14159 -
    (-Math.atan((Math.tan(grad) / Math.sin(lrad)))));
//alert(azi*57.29578);
form.azimuth.value = (azi * 57.29578).toFixed(2);
azimuth = (azi * 57.29578);

a = Math.cos(grad);
b = Math.cos(lrad);
ele = Math.atan((a * b - .1512) / (Math.sqrt(1 - (a * a) * (b * b))));
form.elevation.value = (ele * 57.29578).toFixed(2);

```

Illustration 14 : Code de l'algorithme en JavaScript

```

1  from math import *
2
3  lat = 1
4  long = 1
5
6  azimuth=0
7  elevation=0
8
9  g = (-9+long)
10 grad = (g / 57.29578)
11 lrad = (lat / 57.29578)
12
13 azi= (pi - (-atan((tan(grad) / sin(lrad)))))
14 azimuth = (azi*57.29578)
15
16 a = cos(grad)
17 b = cos(lrad)
18 ele = atan((a*b-.1512)/(sqrt(1-pow(a,2)*pow(b,2))))
19 elevation = ele*57.29578
20
21 print 'Elevation = ' + str(elevation)
22 print 'Azimuth =' +str(azimuth)

```

Illustration 15 : Code de l'algorithme en Python

Ci-dessus se trouve l'algorithme écrit en JavaScript¹² ainsi que son équivalence en Python. Pour vérifier le bon fonctionnement du code en Python, nous avons réalisé des tests afin de contrôler que nous obtenions bien les mêmes valeurs d'azimut et d'élévation en Javascript et en Python.

The screenshot shows a user interface for calculating geolocation parameters. It has two main sections: 'Carte' (Map) and 'Coordonnées' (Coordinates). In the 'Coordonnées' section, there are fields for 'Adresse' (Address), 'Latitude' (45.5), and 'Longitude' (7.2). A 'Calculer' (Calculate) button is present. To the right, under 'Résultats' (Results), are fields for 'Elévation' (37.59) and 'Azimut géographique' (177.48), along with a 'Configuration du spot' (Spot configuration) icon and an 'Imprimer' (Print) button.

Illustration 16 : Test de l'algorithme sous JavaScript



The screenshot shows a terminal window titled 'Result'. It displays the command '\$python main.py' followed by the output: 'Elevation = 37.5911798651' and 'Azimuth = 177.477144947'.

Illustration 17 : Test de l'algorithme sous Python

Dans cet exemple, nous avons pris une latitude de 45,5 et une longitude de 7,2. On voit que les valeurs d'azimut et d'élévation obtenues sont identiques. Nous avons donc réalisé avec succès l'extraction du code de l'algorithme et sa traduction en langage Python.

¹² Le JavaScript est un langage de programmation souvent employé dans les pages web

2.3. Boussole

La boussole joue un rôle crucial dans notre projet et c'est également la partie la plus critique. En effet, après la prise de connaissance du sujet, le professionnel nous a affirmé que la boussole utilisée l'année dernière (HMC5883L) ne donnait pas de résultats concluants lors des tests.

La boussole est présente pour établir un repère par rapport au Nord magnétique et transmettre l'angle d'azimut au système.

Le but sera donc de déterminer l'angle à transmettre au rotor en prenant compte du décalage du camion.

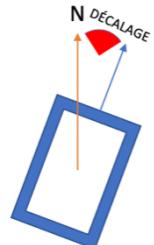


Illustration 18 :
Décalage par rapport
au nord

Dans un premier temps, nous avons décidé, suite aux problèmes, de changer de boussole et de partir sur un modèle plus précis et moins sensible aux perturbations.

Notre choix s'est porté sur une CMPS12. Elle est composée de :

- 3 axes x, y et z
- Un gyroscope sur 3 axes
- Un accéléromètre sur 3 axes

Elle possède une précision d'environ 99% et peut communiquer avec un bus I2C¹³ ou un port série.

Pour des questions d'adaptation avec le câblage de l'ancienne boussole, nous avons décidé d'utiliser la communication I2C.



Illustration 19 : Boussole CMPS12

¹³ Le bus I2C (Inter-Integrated Circuit) permet de relier facilement un microprocesseur avec d'autres composants électronique

Pour établir une connexion sur le bus I2C de la Raspberry Pi, nous avons relié le 3.3 volts, les lignes de données bidirectionnelles SDA et SCL, le Mode (qui est la sélection du mode I2C ou série) et le 0V sur la Raspberry Pi.

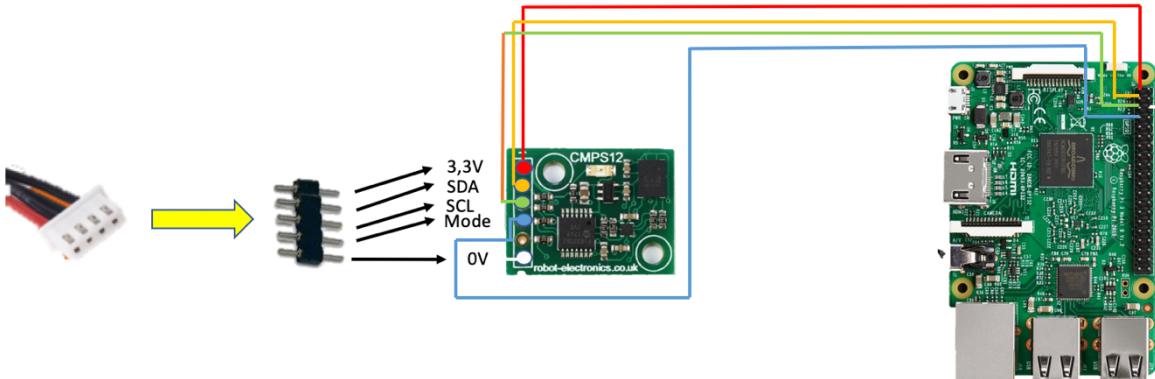


Illustration 20 : Schéma de cablage de la boussole CMPS12

Pour établir le dialogue avec cette boussole, nous utilisons la bibliothèque « `smbus` » qui permet de lire et d'écrire facilement sur le bus I2C.

L'angle et le nord magnétique sont déterminables grâce aux registres¹⁴ 0x1A et 0x1B présents dans la documentation du constructeur. Ces registres permettent de nous donner une valeur comprise entre 0 et 5759 qui devra être ensuite divisée par 16 pour donner une valeur en degré.

0x1A, 0x1B	Compass Bearing 16 bit This is the angle Bosch generate in the BNO055 (0-5759), divide by 16 for degrees
------------	--

Le code que nous avons effectué concatène¹⁵ le bit de poids fort et de poids faible pour le stocker dans une variable. La variable obtenue correspond à l'angle par rapport au nord magnétique.

Malgré de nombreux tests de calibrage et de variantes dans le code de notre programme, les valeurs rentrées étaient toujours erronées.

¹⁴ Un registre est un emplacement de mémoire très réduit présent au sein d'un processeur

¹⁵ Le terme concaténation désigne l'action de mettre au minimum deux chaînes de caractères bout à bout pour n'en créer qu'une seule

Nous avons donc décidé de revenir sur la boussole de l'année dernière jugeant la documentation de la CMPS12 peu détaillée et parfois contradictoire. Nous sommes donc repartis sur une boussole HMC5883L. Contrairement à l'année dernière, avant d'effectuer des mesures, nous avons calibré la boussole à l'aide d'un script effectuant 500 mesures des axes x, y et z.

```
pi@raspberrypi:~/Desktop $ sudo python boussole_step1.py
minx: -214
miny: -387
maxx: 124
maxy: 0
x offset: -45
y offset: -194
```

Illustration 21 : Résultat de la calibration de la boussole

Le but étant d'obtenir les valeurs de « x offset » et « y offset » obtenues en appliquant les formules suivantes :

$$x \text{ offset} = \frac{\text{maxx} + \text{minx}}{2}$$

$$y \text{ offset} = \frac{\text{maxy} + \text{miny}}{2}$$

Pour obtenir le nord magnétique nous devons lire la valeur des axes. Chaque axe est écrit dans deux registres comme le montre le tableau ci-dessous :

Address Location	Name	Access
00	Configuration Register A	Read/Write
01	Configuration Register B	Read/Write
02	Mode Register	Read/Write
03	Data Output X MSB Register	Read
04	Data Output X LSB Register	Read
05	Data Output Z MSB Register	Read
06	Data Output Z LSB Register	Read
07	Data Output Y MSB Register	Read
08	Data Output Y LSB Register	Read
09	Status Register	Read
10	Identification Register A	Read
11	Identification Register B	Read
12	Identification Register C	Read

Illustration 22 : Tableau des registres de la boussole

Pour lire les deux bits simultanément, nous avons réalisé une fonction « `read_word()` » qui prend en paramètre l'adresse d'un registre. Le but étant de lire le bit de poids fort et de poids faible puis de signer la valeur obtenue par l'intermédiaire d'une condition « Si ».

Nous avons également écrit une fonction « `read_axes()` » qui effectue les actions suivantes :

- Lire les valeurs des axes x, y et z
- Calcul du nord en utilisant une formule trigonométrique
- On applique un modulo 360 sur la valeur de l'angle afin qu'elle soit comprise entre 0° et 360°

```
def read_word(addr):
    high = bus.read_byte_data(address, addr)
    low = bus.read_byte_data(address, addr+1)
    val = (high << 8) | low
    if (val >= 0x8000):
        return -((65535 - val) + 1)
    else:
        return val
```

Illustration 23 : Code de la fonction "read_word"

```
def read_axes():
    axeX = (read_word(0x03)-x_offset)*scale
    axeY = (read_word(0x07)-y_offset)*scale
    axeZ = read_word(0x05)*scale
    angle = math.atan2(axeX,axeY)*180/math.pi;

    if (angle<0):
        angle+=360

    if (angle > 360):
        angle-=360

    return(axeX,axeY,axeZ, angle)
```

Illustration 24 : Code de la fonction "read_axes"

Une fois les tests effectués, nous nous sommes rendus compte que la boussole était très sensible aux perturbations électromagnétiques. Il était donc possible d'obtenir le nord magnétique néanmoins l'impact de l'environnement restait important. Les valeurs rentrées étaient très aléatoires et donc peu fiables.

Nous avons donc réfléchi à une autre solution beaucoup plus juste. La création d'un indicateur numérique sur la partie IHM permettrait à l'utilisateur de rentrer lui-même l'angle de direction du camion à l'aide d'une boussole physique afin que l'algorithme compense et donne la référence du nord magnétique. Cette solution sera détaillée dans la partie IHM.

3. Pilotage

Dans cette partie, nous allons traiter du pilotage à travers une IHM d'un vérin et d'un rotor pour procéder au placement automatique de la parabole.

3.1. Pilotage du rotor en angle

Pour assurer l'orientation en azimut de la parabole, nous avons utilisé un rotor YAESU G-1000DX dont les caractéristiques techniques sont présentées ci-après.

3.1.1. Caractéristiques techniques du rotor

Le rotor YAESU G-1000DX possède 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



Illustration 25 : Rotor
YAESU G-1000DX

Ce rotor correspond bien à nos besoins car il est capable de supporter le poids de la parabole. Néanmoins, on ne peut pas relier directement le rotor à la Raspberry. Pour pallier à ce problème, nous avons utilisé une carte d'interface ERC Mini.

3.1.2. Commande du rotor

La carte ERC Mini est relativement simple à utiliser avec le rotor car cette dernière contient déjà des registres pour communiquer avec les rotors YAESU. Pour pouvoir envoyer les commandes souhaitées de la Raspberry vers l'ERC, Nous nous servons de la bibliothèque « Serial » pour établir une liaison série.

Pour créer un objet de type « serial », il faut spécifier le port USB sur lequel est connecté l'ERC Mini avec cette commande :

- rotor = serial.Serial("/dev/ttyUSB<n° port>").

En annexe 2 se trouve la liste complète des commandes disponibles pour communiquer avec le rotor. Parmi toutes ces commandes, voici celles que nous utilisons :

- Mxxx<cr> : faire tourner le rotor en 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
- sCAL0000<cr> : effectuer la calibration du rotor à 0°
- sCAR0360<cr> : effectuer la calibration du rotor à 360°

Le <cr> signifie un retour chariot et se symbolise par « /r/n » sous Python.

Lors de la première connexion de l'ERC Mini avec le rotor, il est indispensable d'effectuer un calibrage pour que la commande du rotor en angle soit fonctionnelle. Avant de nous concentrer pleinement sur la programmation de l'ERC, nous avons préféré analyser le fonctionnement de ce dernier en le connectant à un ordinateur pour utiliser l'exécutable « Service-Tool ERC-Mini ».

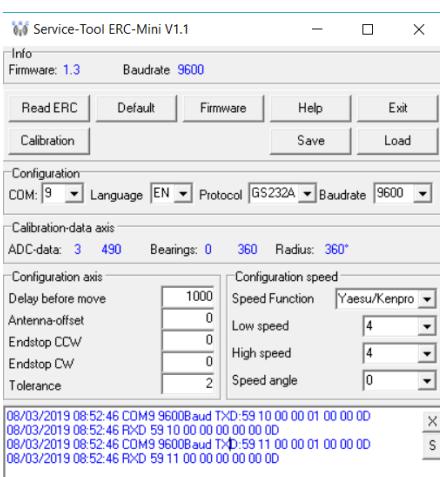


Illustration 27 : Capture d'écran du logiciel pour l'ERC

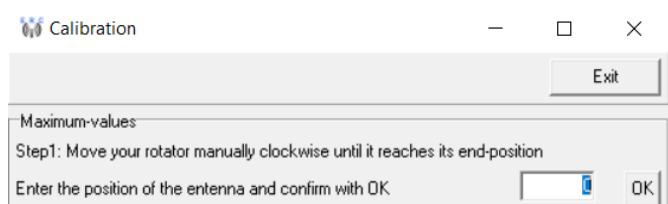


Illustration 26 : Capture d'écran de la boîte de dialogue pour la calibration de l'ERC

Sur cet exécutable, certaines options sont configurables comme nous le voyons sur l'illustration 26. Tout d'abord, le COM 9 correspond au port de connexion à l'ordinateur de l'ERC Mini. Par défaut, on utilise le protocole GS232A qui est le plus souvent utilisé pour contrôler des rotors ainsi qu'un baudrate de 9600. Mais nous nous intéressons à la calibration de la carte d'interface, celle-ci se fait en deux étapes :

- Premièrement, une boîte de dialogue s'ouvre et demande à l'utilisateur de faire tourner manuellement le rotor dans le sens horaire jusqu'à atteindre 0 degré puis le logiciel envoie la commande « sCAL0000 »
- Ensuite, une seconde boîte de dialogue demande de faire tourner le rotor dans le sens antihoraire jusqu'à atteindre 360 degrés et le logiciel envoie la commande « sCAR0360 »

Pour résumer, voici les étapes à suivre pour calibrer l'ERC Mini :

- Positionner manuellement le rotor à 0°
- Envoyer la commande « sCAL0000 »
- Positionner manuellement le rotor à 360°
- Envoyer la commande « sCAR0360 »

Pour commander l'ERC Mini avec la Raspberry, nous avons repris la classe qui a été rédigée par le groupe précédent. Ci-contre se trouve les prototypes des fonctions de cette classe :

```

def tourner(self, angle):
    if self.ser is not None:
        angle = int(angle)
        if (angle <100):
            commande = "M0" + repr(angle) + "\r\n"
        else:
            commande = "M" + repr(angle) + "\r\n"
        print(commande)
        self.ser.write(commande.encode('latin-1'))

def angle(self):
    if self.ser is not None:
        self.ser.write(b'C\r')
        return int(self.ser.readline())

def tournerHoraire(self):
    if self.ser is not None:
        self.ser.write(b'R\r\n')

def tournerAntiHoraire(self):
    if self.ser is not None:
        self.ser.write(b'L\r\n')

def stop(self):
    if self.ser is not None:
        self.ser.write(b'S\r\n')

def calibrationDroite(self):
    if self.ser is not None:
        input('Déplacez votre rotor dans le sens des aiguilles d\'une montre (360°)');
        self.ser.write(b'sCAR0360\r\n')

def calibrationGauche(self):
    if self.ser is not None:
        input('Déplacez le rotator dans la position la plus anti-horaire (0°)')
        self.ser.write(b'sCAL0000\r\n')

```

Illustration 28 : Classe du rotor

Voici la description des différentes fonctions :

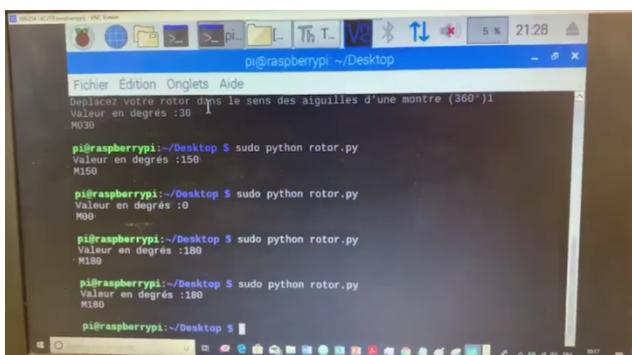
- « tourner » : faire tourner le rotor jusqu'à un angle donné
- « angle » : lire la valeur de l'angle du rotor
- « tournerHoraire » : tourner le rotor dans le sens des aiguilles d'une montre

- « tournerAntiHoraire » : tourner le rotor dans le sens inverse des aiguilles d'une montre
- « stop » : arrêter la rotation du rotor
- « calibrationDroite » : calibration du rotor à 0°
- « calibrationGauche » : calibration du rotor à 360°

Par rapport au programme de l'année dernière, nous avons ajouté des temporisations dans les fonctions de calibration pour laisser le temps à l'utilisateur de pouvoir déplacer le rotor manuellement. Ces temporisations sont caractérisées par la commande « input » qui attend une saisie de l'utilisateur avant d'envoyer les commandes. Nous avons également corrigé la fonction « tourner » pour ajuster la commande à envoyer en fonction de la valeur de l'angle :

- Pour un angle inférieur à 100°, on doit envoyer une commande de la forme « M0xx » où xx est la valeur de l'angle
- Pour un angle supérieur à 100°, la commande à envoyer est de la forme « Mxxxx » où xxxx est la valeur de l'angle

Maintenant que la calibration a été réalisée, nous pouvons simplement contrôler le rotor en angle grâce à la fonction « tourner » qui prend en paramètre la valeur de l'angle souhaité. Voici les tests montrant que notre programme est totalement fonctionnel.



```

pi@raspberrypi:~/Desktop$ sudo python rotor.py
Valeur en degrés :30
M030
pi@raspberrypi:~/Desktop$ sudo python rotor.py
Valeur en degrés :150
M150
pi@raspberrypi:~/Desktop$ sudo python rotor.py
Valeur en degrés :180
M180
pi@raspberrypi:~/Desktop$ sudo python rotor.py
Valeur en degrés :180
M180
pi@raspberrypi:~/Desktop$ 

```

Illustration 30 : Test de l'envoi d'une commande au rotor



Illustration 29 : Résultat de la commande envoyée

3.2. Pilotage du vérin en angle

Dans cette partie nous allons expliquer les démarches effectuées pour permettre de contrôler le vérin en angle.

Pour piloter la parabole en élévation, nous avons besoin d'un vérin ayant une force de poussée suffisante pour pouvoir soulever la parabole qui a une masse d'environ 15 kilogrammes. Notre choix s'est porté sur le vérin SuperJack III de chez Jaeger qui possède les caractéristiques 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 31 : Vérin SuperJack III

Comme la parabole doit être installée sur le camion, il faut que le vérin soit alimenté avec la tension pouvant être fournie par le camion soit 24 volts maximum. L'alimentation du vérin est donc inférieure à celle recommandée par le constructeur. Néanmoins, cette alimentation suffit pour que le vérin émette une force permettant l'orientation de la parabole, qui ne pèse que 15 kilogrammes.

Pour pouvoir commander le vérin dans notre système, nous devons utiliser une carte d'interface car la Raspberry Pi ne peut fournir la puissance nécessaire (5 volts au lieu de 12 volts minimum). La carte d'interface qui a été retenue est la L298N car celle-ci a besoin d'une alimentation minimum de 5 volts que peut fournir la Raspberry Pi. Nous avons dessiné un schéma de câblage car nous avons eu des difficultés à reprendre le montage du groupe précédent qui n'en avait pas effectué.

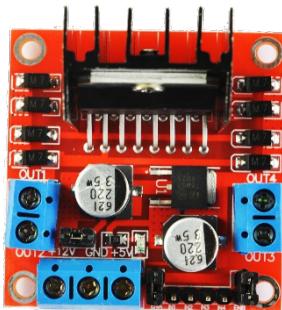


Illustration 32 : Carte d'interface L298N

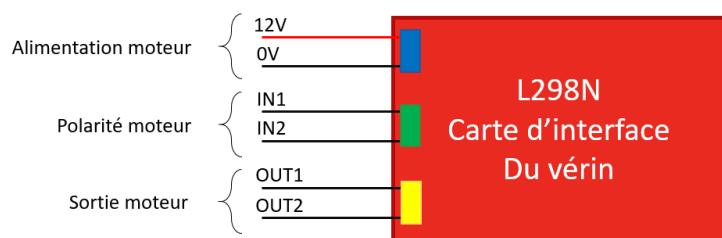


Illustration 33 : Schéma de cablage de la carte d'interface

Cette carte comunique avec la Raspberry par l'intermédiaire de GPIO¹⁶.

Pour choisir le sens de rotation du moteur, il faut envoyer un signal soit sur IN1 soit sur IN2.

Pour pouvoir simplifier la commande du vérin dans notre système, nous avons repris la classe « Verin » que nous avons adapté pour correspondre au cahier des charges. En annexe 3 se trouve le code complet de cette classe. Pour l'utiliser, on indique tout d'abord qu'on travaille avec des numéros de broches et non pas avec des numéros de GPIO grâce à la commande suivante :

- `GPIO.setmode(GPIO.BCM)`

Ensuite, nous devons envoyer des informations vers la carte d'interface donc il faut configurer les GPIOs en sortie avec la ligne suivante :

- `GPIO.setup(GPIO.OUT)`

A présent, on peut affecter un état logique aux GPIOs :

- `GPIO.output(« numéro du GPIO », GPIO.HIGH)` pour affecter à 1 le GPIO
- `GPIO.output(« numéro du GPIO », GPIO.LOW)` pour affecter à 0 le GPIO

¹⁶ Les ports GPIO (General Purpose Input/Output) sont des ports d'entrées/sorties permettant de communiquer avec des composants électroniques externes au système

Pour contrôler le vérin, trois fonctions sont à notre disposition :

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

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)
```

- La fonction « monter » met les sorties IN1 à 1 et IN2 à 0
- La fonction « descendre » met les sorties IN1 à 0 et IN2 à 1
- La fonction « arreter » met les sorties IN1 à 0 et IN2 à 0

Illustration 34 : Extrait de la classe "Verin"

Pour procéder au placement automatique de la parabole en élévation, nous avons dû réfléchir à la mise en place d'une fonction permettant de contrôler le vérin en angle. Pour cela, nous avons d'abord effectué des tests en prenant en compte la valeur de l'angle et la durée que met le vérin à orienter la parabole. Voici le résultat de nos premières mesures :

- Pour parcourir 10°, le vérin met 15 secondes
- Pour parcourir 20°, le vérin met environ 30 secondes

On voit que le système est linéaire pour des petites valeurs d'angles. Malheureusement, nous ne pouvons pas démontrer la linéarité pour toutes les valeurs d'angles car nous n'avons pas eu le temps d'effectuer d'autres mesures.

A partir des valeurs précédemment trouvées, on voit qu'il faut appliquer un coefficient de 1.5 pour obtenir le temps nécessaire en fonction de l'angle.

Nous avons résumé cela au sein d'une fonction nommée « auto_verin ». Voici l'explication de cette fonction :

- La fonction prend en paramètre une valeur d'angle
- Le programme convertit la valeur de l'angle en une valeur temporelle
- Le vérin descend jusqu'à arriver en butée (0°)
- Le vérin monte pendant la durée nécessaire pour arriver à la valeur d'angle souhaitée puis s'arrête

```
def auto_verin():
    total_verin = int(afficher())
    auto_verin = total_verin*1.5
    verin.descendre()
    sleep(90)
    verin.monter()
    sleep(auto_verin)
    verin.arreter()
```

Illustration 35 : Fonction "auto_verin()"

La commande du rotor en angle fonctionne mais est approximative. Lors de la connexion avec un satellite, nous effectuerons un balayage en angle pour obtenir plus de précision.

3.3. Interface Homme Machine

Dans le cadre de notre projet, l'usager doit pouvoir piloter l'antenne **manuellement et automatiquement**. Pour cela, il est nécessaire de réaliser un affichage graphique. Cet affichage aura comme support un écran tactile d'une taille de 480 par 320 pixels. Pour respecter les contraintes de compatibilité avec les autres programmes, nous avons choisi de garder le **langage de programmation Python**. Une interface graphique a été réalisée par le groupe de l'année dernière. Elle était fonctionnelle, mais par souci de flexibilité nous n'avons conservé que la librairie graphique nommée « **Pygame** ».



Illustration 36 : Logo pyGame

Notre interface graphique a évolué au fil du temps. **Notre première ébauche** n'avait que pour but de tester les différents équipements de notre projet (Contrôle du vérin et du rotor). Cette interface était une première approche de la librairie graphique Pygame. Celle-ci nous a permis de **mettre en place certaines notions importantes** pour la future solution graphique telles que la programmation des boutons et l'affichage des variables.

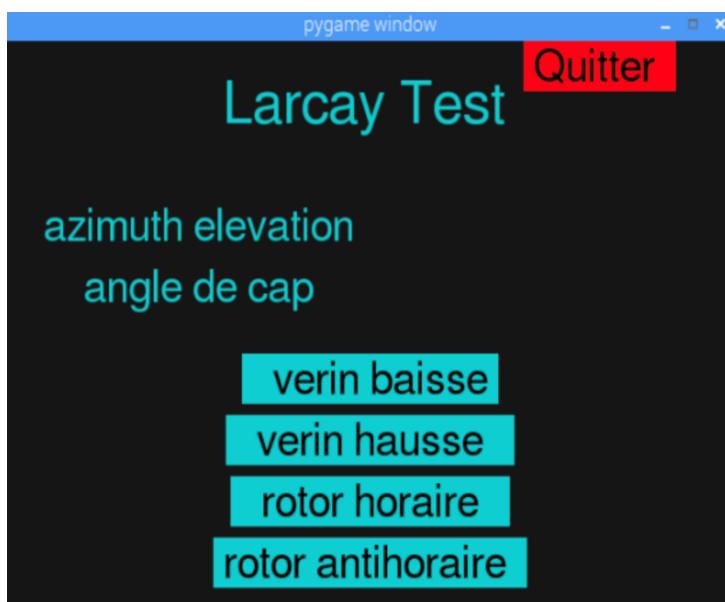


Illustration 37 : Première IHM réalisée

On peut voir sur cette interface les différents boutons mis en place afin de commander manuellement le vérin et le rotor. Le code de cette interface ne sera pas détaillé, car peu intéressant.

Par la suite, nous nous sommes attelés à réaliser une interface graphique qui sera celle fournie à l'utilisateur. Nous avons donc listé les besoins de cette interface :

- Affichage de la position du vérin et du rotor
- Puissance et qualité du signal
- Cordonnées GPS du camion
- Clavier avec indicateur numérique
- Elévation et azimut où l'on devrait pointer d'après l'algorithme

Après avoir mis en place ce cahier des charges, nous avons choisi contrairement au groupe précédent de découper le programme en classe. La Programmation orientée objet (code découpé par classe) permet d'avoir une meilleure visibilité du code ainsi que la possibilité de le modifier le cas échéant.

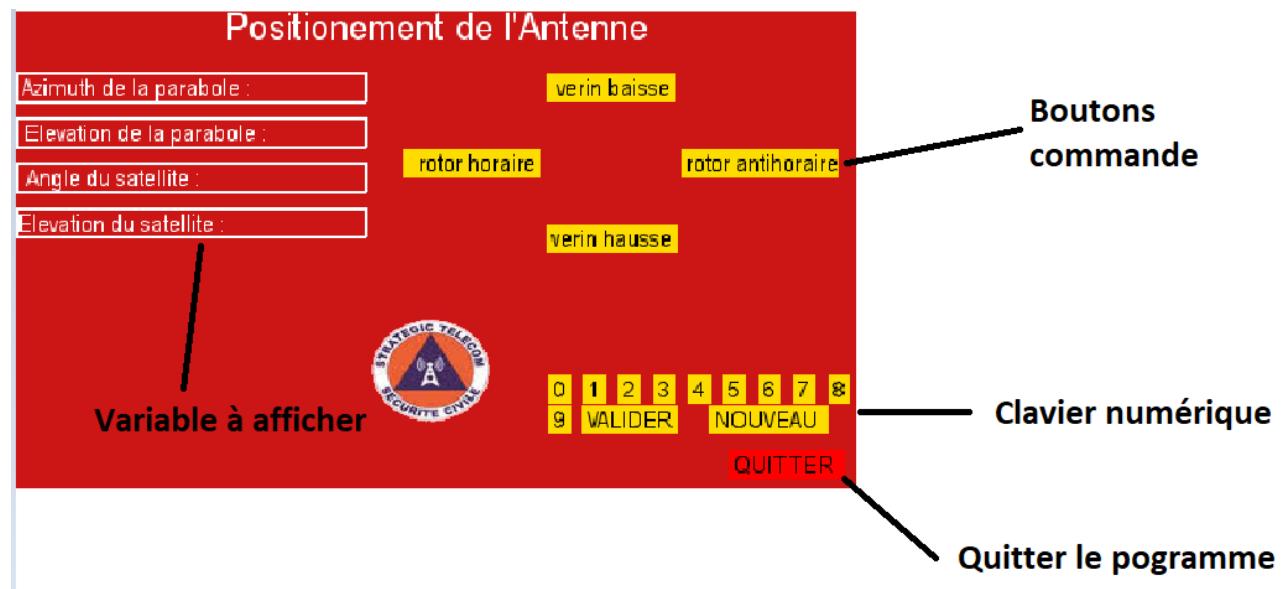


Illustration 38 : Dernière version de l'IHM

Sur cette capture d'écran la partie graphique est quasiment terminée mais il manque le bouton automatique et l'affichage de certaines variables.

Pour créer des boutons, nous avons écrit une classe « Bouton » dont le code complet est en annexe 4. Celle-ci permet d'obtenir des boutons affichables sur l'interface et se décompose en 3 fonctions :

- « Init » : créer un bouton en choisissant des paramètres tels que la taille de la police et la couleur
- « Update_button » : détecte l'appui sur le bouton
- « Display_button » : affiche le bouton sur l'interface

Ce code est complété par une autre classe nommée IHM qui ne sera pas détaillée au sein du D4R4. Elle permet de relier des fonctions au bouton et de donner un aspect esthétique à notre interface.

Ensuite, pour parer aux problèmes évoqués dans la partie boussole, nous avons décidé de mettre en place un indicateur numérique où l'utilisateur doit pouvoir rentrer la valeur d'angle de la boussole obtenue en autonomie. La tâche fut complexe à réaliser puisque nous devions récupérer la valeur rentrée et anticiper une faute de l'utilisateur.

Le code est décomposé en 10 fonctions. Chaque fonction correspond à un numéro, de 0 à 9. Le programme est complété par des fonctions de saisie et de correction.

Après une étude des différentes solutions possibles, nous avons choisi de créer un tableau nommé « garder » dans lequel on ajoute une valeur à chaque clic sur un bouton numéroté. Par exemple, si on appuie successivement sur les boutons « 2 », « 4 » et « 7 », le tableau « garder » possédera trois lignes composés des valeurs saisies précédemment comme le montre le résultat ci-dessous :

['2', '4', '7']

Illustration 40 : Exemple d'une saisie de numéros

```
garder = []

def nouveau():
    del garder[0:100]
    return garder

def saisie(para):
    garder.append(para)
    print (garder)
    return garder

def numero0():
    a = 0
    saisie(a)
    #numero0()
def numero1():
    b = "1"
    saisie(b)
def numero2():
    c = "2"
    saisie(c)
```

Illustration 39 : Fonctions permettant de saisir les numéros

Ce tableau conserve la valeur d'angle saisit par l'utilisateur. Néanmoins, il faut maintenant transformer ce tableau en nombre entier pour pouvoir l'utiliser dans les autres parties de notre programme. Pour cela, au moment où l'utilisateur clique sur le bouton « VALIDER », on extrait les 3 premières valeurs du tableau et on les concatène pour l'affecter à une unique variable. Cette variable est ensuite retournée pour être utilisée par l'algorithme. Dans le cas où l'utilisateur souhaite rentrer une nouvelle valeur, il suffit de cliquer sur « Nouveau » pour supprimer l'intégralité des valeurs comprises dans le tableau « garder » et en saisir de nouvelles. Ci-après se trouve le code illustrant nos propos et un exemple prouvant que le code est fonctionnel.

```
def afficher():
    total = []
    total = saisie("")
    total.append("")
    total.append("")
    total.append("")
    premier_chiffre = total[0]
    second_chiffre = total[1]
    troisieme_chiffre= total[2]
    Angle = premier_chiffre+second_chiffre+troisieme_chiffre
    print(Angle)
    return Angle
```

Illustration 41 : Extrait de la classe "afficher"

```
[ '2'
[ '2', '4'
[ '2', '4', '7'
[ '2', '4', '7', ''
247
```

Illustration 42 : Résultat de la concaténation des numéros

Notre IHM est presque complète et fonctionnelle, il faudra ajouter dans le futur l'affichage des variables sur l'interface graphique.

Conclusion

En conclusion, notre système permet de piloter les différents organes manuellement mais également automatiquement. Il est donc possible de piloter le rotor en angle mais également de pouvoir piloter le vérin à volonté. Cette étape est primordiale pour l'automatisation.

Néanmoins, un temps considérable a été perdu sur l'étude de la boussole numérique. En effet, beaucoup de tests et de codes ont été réalisés afin de rendre le projet le plus autonome possible. Suite à ce problème, le groupe a donc dû prendre une décision sur la stratégie à adopter afin de répondre au mieux au cahier des charges du client.

Ce projet qui était à prédominance informatique nous a permis de mettre à profit nos connaissances en terme de programmation orientée objet mais également de se familiariser à un nouveau langage de programmation qu'est Python.

La cohésion dans notre groupe fut primordiale pour rendre notre travail efficace et coordonné. Chaque personne avait une tâche assignée ce qui a permis d'avancer en équipe et de rendre le projet fonctionnel. Afin d'appréhender les différents problèmes, cet exercice a nécessité une bonne organisation de notre part et nous a fait découvrir la gestion de projet.

Cette première immersion professionnelle au sein de Strategic Telecom a permis de développer notre relationnel et de s'adapter au cahier des charges du client.

Cette expérience a été très formatrice et se révèlera certainement très utile lors de notre stage de fin d'études.

Nous remercions Monsieur Tailliez pour sa gentillesse et sa confiance.

Résumé

Notre projet consiste à automatiser le positionnement d'une parabole dans le but d'une connexion internet par satellite en collaboration avec Strategic Telecom. Un boitier a été réalisé afin de relier notre Raspberry aux différents modules permettant de commander notre parabole.

Tout d'abord, nous avons trouvé un site qui donne l'azimut et l'élévation en fonction de notre position géographique qui est donnée par le module GPS présent dans le boitier. Ces données permettront d'orienter le vérin et le rotor.

Nous avons donc réalisé un programme permettant de positionner la parabole en azimut. Ce programme comprend le calibrage et la commande de l'angle du rotor.

Pour respecter l'élévation, nous avons programmé le vérin à partir d'un coefficient. Ce coefficient est calculé grâce à des mesures que nous avons effectuées sur le vérin. Il est déterminé par le temps que met le vérin à atteindre un angle donné.

Nous avons pris du retard suite aux mauvaises qualités des données renvoyées par la boussole. Nous avons découvert que celle-ci était facilement perturbée par certaines sources magnétiques. Nous avons donc décidé de réaliser une commande numérique permettant à l'utilisateur de rentrer manuellement la valeur de l'angle du camion par rapport au nord magnétique.

Enfin, la commande numérique a été développée sur une nouvelle interface graphique ayant comme support un écran tactile. Celle-ci contient des informations sur le positionnement de la parabole mais également une commande manuelle permettant à l'utilisateur de modifier le positionnement du système le cas échéant.

243 mots

Bibliographie

- Documentation de la bibliothèque « pySerial »
<https://pythonhosted.org/pyserial/>
- Documentation de la bibliothèque « pyGame »
<https://pythonhosted.org/pyserial/>
- Documentation de la bibliothèque « Requests » (par Kenneth Reitz)
<http://docs.python-requests.org/en/master/>
- Documentation de la bibliothèque « Pynmea2 » (par Knio, dernière MAJ le 6 février 2019)
<http://docs.python-requests.org/en/master/>
- Documentation de la bibliothèque « Pynmea2 » (par Knio, dernière MAJ le 6 février 2019)
<http://docs.python-requests.org/en/master/>
- Documentation de la bibliothèque « RPi.GPIO » (par Ben Crostron, dernière MAJ le 16 novembre 2018)
<https://pypi.org/project/RPi.GPIO/>
- Site duquel nous avons extrait l'algorithme de positionnement automatique de la parabole
<http://finder.tooway-instal.com/fixe/pages/index.html>
- Documentation constructeur de la boussole HMC5883L
<http://www.farnell.com/datasheets/1683374.pdf>
- Site constructeur de la carte d'interface ERC Mini
<https://www.schmidt-alba.de/eshop/>

Index des mots clés

A

algorithme 5, 8, 16, 17, 18, 22, 35, 38
angle8, 11, 14, 19, 20, 22, 23, 24, 26, 27, 28, 30, 31,
34, 35

automatisation 8, 10, 13
azimut 8, 14, 16, 17, 18, 19, 23, 24, 33

B

boussole 8, 9, 19, 20, 21, 22, 34, 38

C

câblage 8, 10, 11, 12, 19, 28
calibrage 20, 24
classe 15, 25, 29, 33, 34, 45, 46

D

développement 8
documentation 10, 11, 20, 21

E

ERC Mini 10, 12, 23, 24, 25, 38, 44

F

fonction 14, 16, 22, 27, 30, 31, 34

G

GPS 13, 14, 15, 16, 33

I

IHM 8, 10, 22, 23, 33, 34, 35
interface8, 10, 12, 13, 23, 25, 28, 29, 32, 33, 34, 35,
38

L

latitude 14, 16, 17, 18

librairie 15, 32

longitude 14, 16, 17, 18

N

nord 8, 20, 21, 22

P

parabole 5, 8, 10, 13, 16, 17, 23, 28, 30, 38
pilotage 8, 23
placement 5, 16, 17, 23, 30
position géographique 5, 13, 16
positionnement 5, 16, 38
programme 12, 13, 17, 20, 27, 31, 33, 34, 35

R

Raspberry 5, 8, 10, 12, 13, 15, 17, 20, 23, 25, 28, 29
registres 20, 21, 23
rotor 5, 8, 10, 11, 19, 23, 24, 25, 26, 27, 28, 30, 31,
32, 33

S

satellite 8, 14, 16, 17, 31

système 5, 8, 10, 12, 17, 28, 29, 30

T

tests 8, 10, 12, 14, 18, 19, 20, 22, 27, 30
trame 13, 14, 15, 16
trames 13, 14, 15

U

USB 10, 11, 12, 13, 15, 24

V

verin 30, 31

Index des illustrations

Illustration 1 : IHM réalisée par le groupe précédent.....	10
Illustration 2 : Schéma de câblage du vérin avec le contrôleur.....	11
Illustration 3 : Schéma de câblage du rotor à la Raspberry	12
Illustration 4 : Module GPS G-Star IV	13
Illustration 5 : Trames renvoyées par le module GPS.....	14
Illustration 6 : Code de la classe "gps"	15
Illustration 7 : Création d'un objet de type Serial.....	15
Illustration 8 : Lecture et analyse des trames reçues	15
Illustration 9 : Découpage de la trame	15
Illustration 10 : Site web contenant l'algorithme.....	16
Illustration 11 : Exemple de détermination de l'azimut et de l'élévation	16
Illustration 12 : Carte des zones de satellites en France	17
Illustration 13 : Liste des fichiers contenus dans le site web.....	17
Illustration 14 : Code de l'algorithme en JavaScript	18
Illustration 15 : Code de l'algorithme en Python.....	18
Illustration 16 : Test de l'algorithme sous JavaScript	18
Illustration 17 : Test de l'algorithme sous Python	18
Illustration 18 : Décalage par rapport au nord.....	19
Illustration 19 : Boussole CMPS12.....	19
Illustration 20 : Schéma de cablage de la boussole CMPS12.....	20
Illustration 21 : Résultat de la calibration de la boussole	21
Illustration 22 : Tableau des registres de la boussole	21
Illustration 23 : Code de la fonction "read_word".....	22
Illustration 24 : Code de la fonction "read_axes".....	22
Illustration 25 : Rotor YAESU G-1000DX	23
Illustration 26 : Capture d'écran de la boite de dialogue pour la calibration de l'ERC	24
Illustration 27 : Capture d'écran du logiciel pour l'ERC.....	24
Illustration 28 : Classe du rotor.....	26
Illustration 29 : Résultat de la commande envoyée	27

Illustration 30 : Test de l'envoi d'une commande au rotor.....	27
Illustration 31 : Vérin SuperJack III	28
Illustration 32 : Carte d'interface L298N	29
Illustration 33 : Schéma de cablage de la carte d'interface.....	29
Illustration 34 : Extrait de la classe "Verin"	30
Illustration 35 : Fonction "auto_verin()"	31
Illustration 36 : Logo pyGame	32
Illustration 37 : Première IHM réalisée	32
Illustration 38 : Dernière version de l'IHM.....	33
Illustration 39 : Fonctions permettant de saisir les numéros	34
Illustration 40 : Exemple d'une saisie de numéros	34
Illustration 41 : Extrait de la classe "afficher".....	35
Illustration 42 : Résultat de la concaténation des numéros	35

Annexes

Index des annexes

Annexe 1 : Diagramme de Gantt du D3R3.....	43
Annexe 2 : Liste des commandes de l'ERC Mini	44
Annexe 3 : Code de la classe « Verin ».....	45
Annexe 4 : Code de la classe « Button »	46

Annexe 1 : Diagramme de Gantt du D3R3



Annexe 2 : Liste des commandes de l'ERC Mini



ERC-Mini V2.0

Manual

Appendix 5: API (application programming interface)

If another OS than Windows is used (e.g. MacOS or Linux), the Service-Tool that came with your ERC-Mini is of no use.

In order to perform configuration- and calibration-tasks, ERC-Mini provides a programming-interface, which can be easily used with a terminal-program.

Remarks for the annotation in this document:

<cr> = carriage return and represent the Ascii-code 13 = Hex-code 0D

Start the terminal-program, set the right COM-port and the communication-speed (default speed is 9600 Baud)

These are the commands provided by the API:

Commands to read the settings of the ERC-Mini:

API ERC-Mini V12	to ERC-Mini	from ERC-Mini	Example	Range	Explanation
Command	ASCII	ASCII			
Read Firmware-version	r F M W (cr)	a F M W 0 1 0 0 (cr)	V1.00	1.00...9.99	
Read Baudrate	r B A U (cr)	a B A U 9 6 0 0 (cr)	9600	4800/9600	
Read Protocol	r P R O (cr)	a P R O 0 0 0 1 (cr)	1=GS232B	0/1/2	0=GS232A, 1=GS232B, 3=DCU1
Read Angle Right	r A N R (cr)	a A N R 0 3 6 0 (cr)	360°	0...360	
Read Angle Left	r A N L (cr)	a A N L 0 0 0 0 (cr)	0°	0...360	
Read Calibration Right	r C A R (cr)	a C A R 1 0 1 0 (cr)	ADC=1010	0...1023	
Read Calibration Left	r C A L (cr)	a C A L 0 0 0 5 (cr)	ADC=5	0...1023	
Read Delay before Move	r D B M (cr)	a D B M 1 0 0 0 (cr)	1000	0...5000	
Read Programmable Stop Right	r P S R (cr)	a P S R 0 0 0 5 (cr)	5	0...179	
Read Programmable Stop Left	r P S L (cr)	a P S L 0 0 1 0 (cr)	10	0...179	
Read Tolerance	r T O L (cr)	a T O L 0 0 0 2 (cr)	2	0...10	
Read Antenna-Offset	r A O F (cr)	a A O F - 0 9 0	-90°	-180...+180	
Read Speedfunction	r S P F (cr)	a S P F 0 0 0 1 (cr)	1	0/1/2	0=no speed function, 1=Yaesu/Kenpro, 2>Create
Read Speed angle	r S P A (cr)	a S P A 0 0 0 3 (cr)	3	0/1/2/3	0=0°, 1=10°, 2=20°, 3=30°
Read Speed at Low-speed	r S P L (cr)	a S P L 0 0 0 1 (cr)	1	0/1	0=speed1, 1=speed2, 2=speed3, 3=speed4
Read Speed at High-speed	r S P H (cr)	a S P H 0 0 0 0 (cr)	0	0/1	0=speed1, 1=speed2, 2=speed3, 3=speed4

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 3 : Code de la classe « Verin »

```
class Verin:
    def __init__(self, pinPush, pinPull, pinInterrupt):
        self.pinPush = pinPush
        self.pinPull = pinPull
        self.pinInterrupt = pinInterrupt
        GPIO.setup(pinPush, GPIO.OUT)
        GPIO.setup(pinPull, GPIO.OUT)
        GPIO.setup(pinInterrupt, GPIO.IN, pull_up_down=GPIO.PUD_UP)
        GPIO.output(pinPush, GPIO.LOW)
        GPIO.output(pinPull, GPIO.LOW)
        self.compteur = 100
        GPIO.add_event_detect(pinInterrupt, GPIO.FALLING, callback=self.impulsion, bouncetime=50)

    def impulsion(self, channel):
        self.compteur += self.increment
        print("Impulsion {}".format(self.compteur))

    def monter(self):
        self.increment = 1
        GPIO.output(self.pinPush, GPIO.HIGH)
        GPIO.output(self.pinPull, GPIO.LOW)

    def descendre(self):
        self.increment = -1
        GPIO.output(self.pinPush, GPIO.LOW)
        GPIO.output(self.pinPull, GPIO.HIGH)

    def arreter(self):
        GPIO.output(self.pinPush, GPIO.LOW)
        GPIO.output(self.pinPull, GPIO.LOW)

    def stopverin():
        print("stop verin")

    def stoprotor():
        print("stop rotor")

    def rotorhor():
        print("rotor horaire")
        tournerHoraire()

    def rotorant():
        print("rotor antihoraire")

    def baisserverin():
        print("baisser verin")

    def monterverin():
        print("monter verin")
        monter()

    def Quitter():
        print("Quit")
        pygame.quit()
        sys.exit()
```

Annexe 4 : Code de la classe « Button »

```
class Button:  
    def __init__(self, fond, text, color, font, dx, dy):  
        self.fond = fond  
        self.text = text  
        self.color = color  
        self.font = font  
        self.dec = dx, dy  
        self.state = False # enable or not  
        self.title = self.font.render(self.text, True, BLACK)  
        textpos = self.title.get_rect()  
        textpos.centerx = self.fond.get_rect().centerx + self.dec[0]  
        textpos.centery = self.dec[1]  
        self.textpos = [textpos[0], textpos[1], textpos[2], textpos[3]]  
        self.rect = pygame.draw.rect(self.fond, self.color, self.textpos)  
        self.fond.blit(self.title, self.textpos)  
  
    def update_button(self, fond, action=None):  
        self.fond = fond  
        mouse_xy = pygame.mouse.get_pos()  
        over = self.rect.collidepoint(mouse_xy)  
        if over:  
            action()  
            if self.color == RED:  
                self.color = GREEN  
                self.state = True  
            elif self.color == GREEN:  
                # sauf les + et -, pour que ce soit toujours vert  
                if len(self.text) > 5: # 5 char avec les espaces  
                    self.color = RED  
                self.state = False  
            # à la bonne couleur  
            self.rect = pygame.draw.rect(self.fond, self.color, self.textpos)  
            self.fond.blit(self.title, self.textpos)  
  
    def display_button(self, fond):  
        self.fond = fond  
        self.rect = pygame.draw.rect(self.fond, self.color, self.textpos)  
        self.fond.blit(self.title, self.textpos)
```