

Rapport S.A.E R5.01 Projet 2  
Pointage automatique  
Gonon Bastien, Carrière Lilian  
BUT 3 GEII ESE B  
Année 2024/2025

## Table of Contents

1. Introduction.....	3
2. La carte électronique .....	4
2.1 Schéma de la Nano Pi .....	4
<b>2.2</b> Fonctionnement de chaque partie de la carte.....	6
3. Mise en place de la Nano Pi Neo .....	15
<b>3.1</b> Attribution des broches sur le schéma .....	15
<b>3.2</b> Installation de l'os et initialisation.....	17
<b>3.3</b> Configuration du réseau.....	19
<b>3.4</b> Ajout des packages.....	21
4. Programmes de la carte.....	22
5. Conclusion.....	33

# 1. Introduction

L'objectif de ce projet d'équipe est de proposer une nouvelle version de la station satellite, déjà existante et installée au bâtiment Z. Les principales modifications seront quant au choix du contrôleur de la carte électronique, l'antenne qui sera utilisée, ainsi que les logiciels et langages de programmation utilisés.

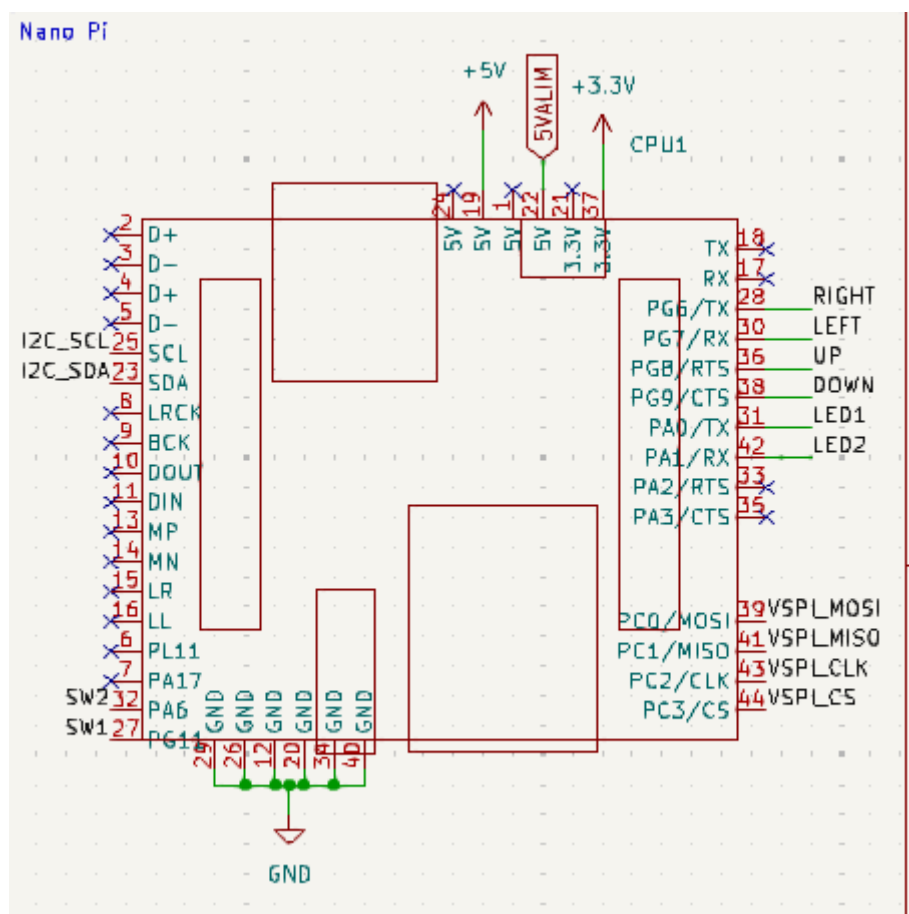
En effet, la précédente station est équipée d'une carte ESP32 fonctionnant sous Arduino. Elle utilise deux antennes à polarisation circulaire placées au sommet d'un pylône. Un moteur est disposé entre le haut du pylône et la barre qui soutient la base des antennes afin de pouvoir modifier leur orientation par un contrôle de leur azimuth et de leur élévation. Ce moteur est conçu de façon qu'il renvoie une tension vers la boîte contrôle permettant ainsi de connaître la position exacte des deux axes du moteur. La carte électronique conçue par des étudiants de BUT 3 au cours de l'année 2023-2024 permet de faire le lien entre la carte moteur et l'ESP32. Elle est équipée de plusieurs entrées analogiques qui arrivent depuis la carte motrice, les données transmises sur ces entrées sont ensuite traitées par des ADC (ou CAN) afin que l'ESP32 puisse traiter ces données et les garder en mémoire. Lorsque l'ESP32 reçoit les informations concernant le passage du satellite qu'il doit suivre, il ajuste alors la position des antennes en utilisant les relais connectés à la carte moteur pour faire pivoter ce dernier. Le travail demandé est donc de modifier cette carte électronique afin qu'elle puisse accueillir une NanoPi NeoCore.

Afin d'effectuer ce travail, le logiciel Kicad sera utilisé pour représenter l'ensemble des connexions.

## 2. La carte électronique

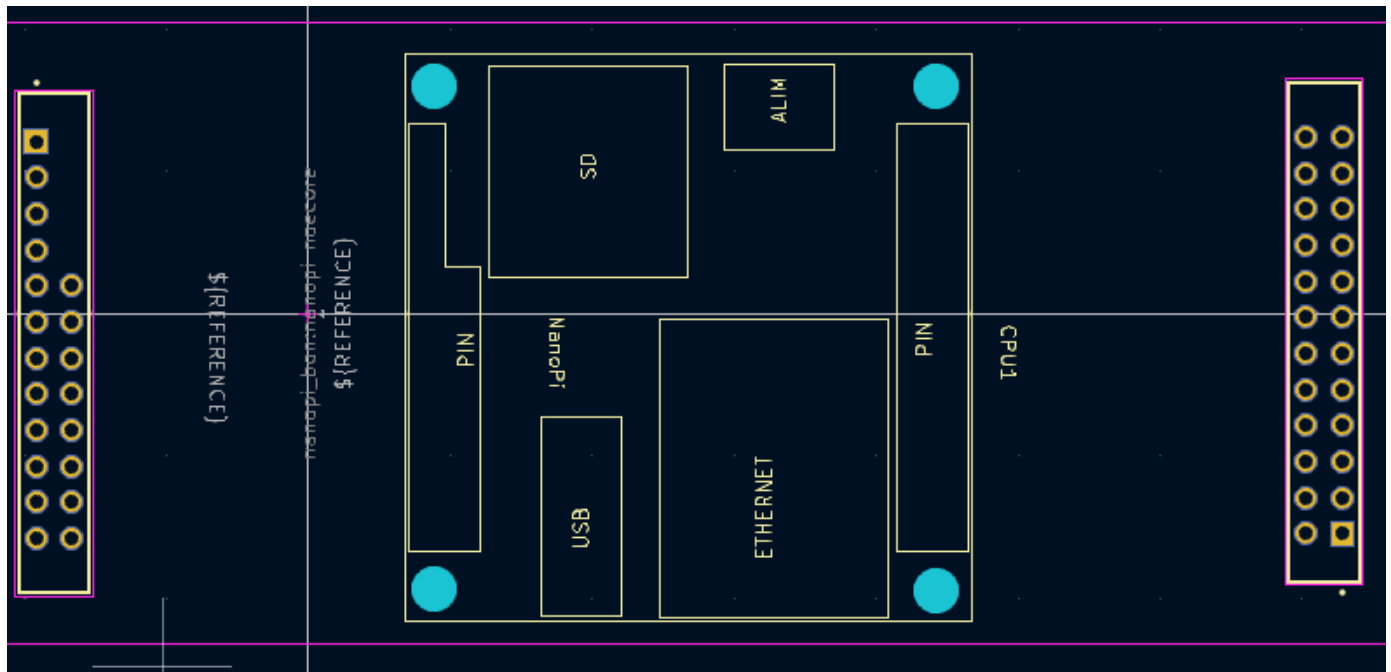
### 2.1 Schéma de la Nano Pi

Pour concevoir cette carte électronique, il a d'abord été nécessaire récupérer le schéma de la carte électronique précédente qui contenait le microcontrôleur ESP32. A partir de l'ancien schéma KiCad, il a fallu créer le schéma et l'empreinte du contrôleur Nano Pi qui n'était pas présent sur le logiciel, ni sur internet.



Annexe 1 : Schéma de la Nano Pi

Pour réaliser cette empreinte, il a fallu récupérer les mesures sur la carte, à la fois dans la documentation présente sur internet, mais aussi avec des mesures prises directement sur la Nano Pi avec un pied à coulisse. Le contrôleur est un carré de 39.8 cm. Un trou est ajouté pour laisser la place pour le processeur (car la Nano Pi s'apparente à un petit ordinateur en termes de puissance), et donc la pâte thermique. 4 autres trous circulaires d'un diamètre de 3.2 mm. Enfin, pour la partie de la Nano Pi, 2 nappes ont été ajoutées pour connecter les pins de la Nano Pi aux composants présent sur la carte. Il a donc fallu aussi créer l'empreinte de ces connecteurs (FC-24P 2\*12 PIN) fournie pour la réalisation du projet. Il est donc primordial de préciser les éléments principaux présents sur la Nano Pi pour ne pas se tromper de sens lors de son installation. Enfin, les écarts entre les connecteurs FC-24P et les connecteurs présents sur la Nano Pi ont été réfléchis pour éviter d'avoir du jeu, et donc de ne pas abimer ces nappes.

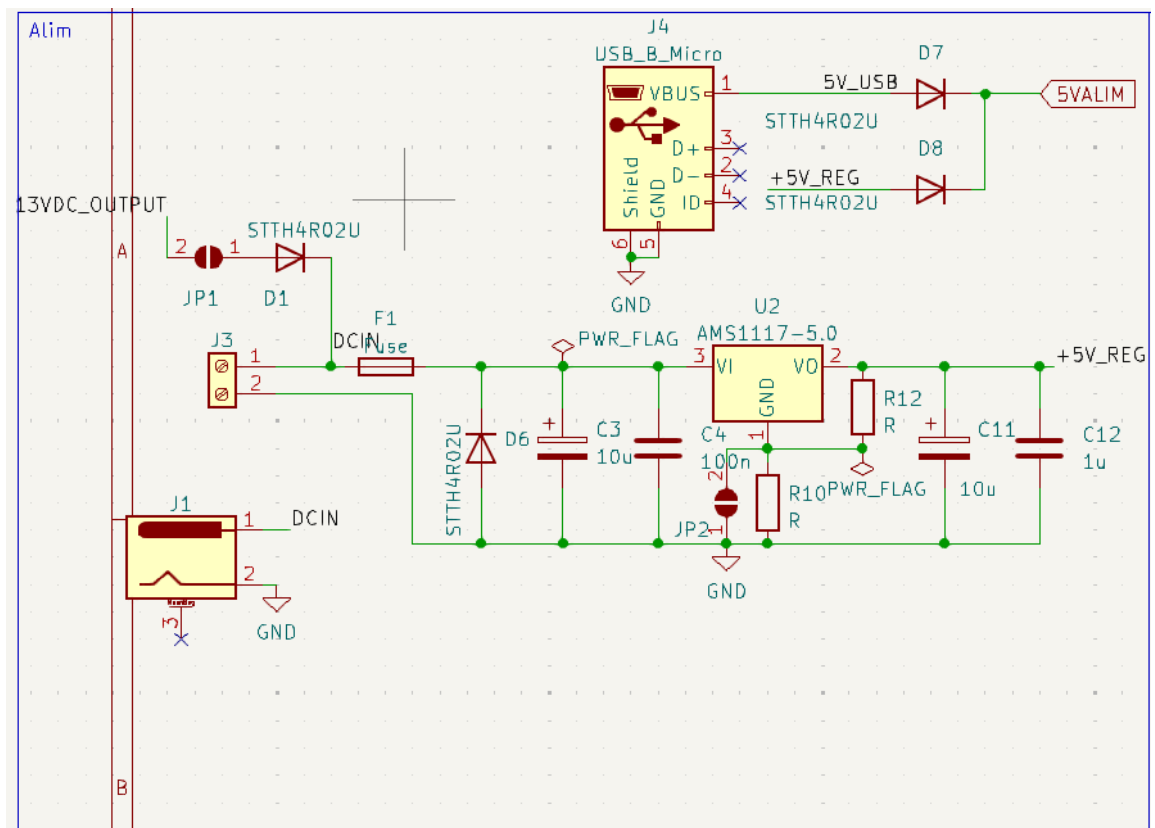


Annexe 2 : Empreinte de la Nano Pi

## 2.2 Fonctionnement de chaque partie de la carte

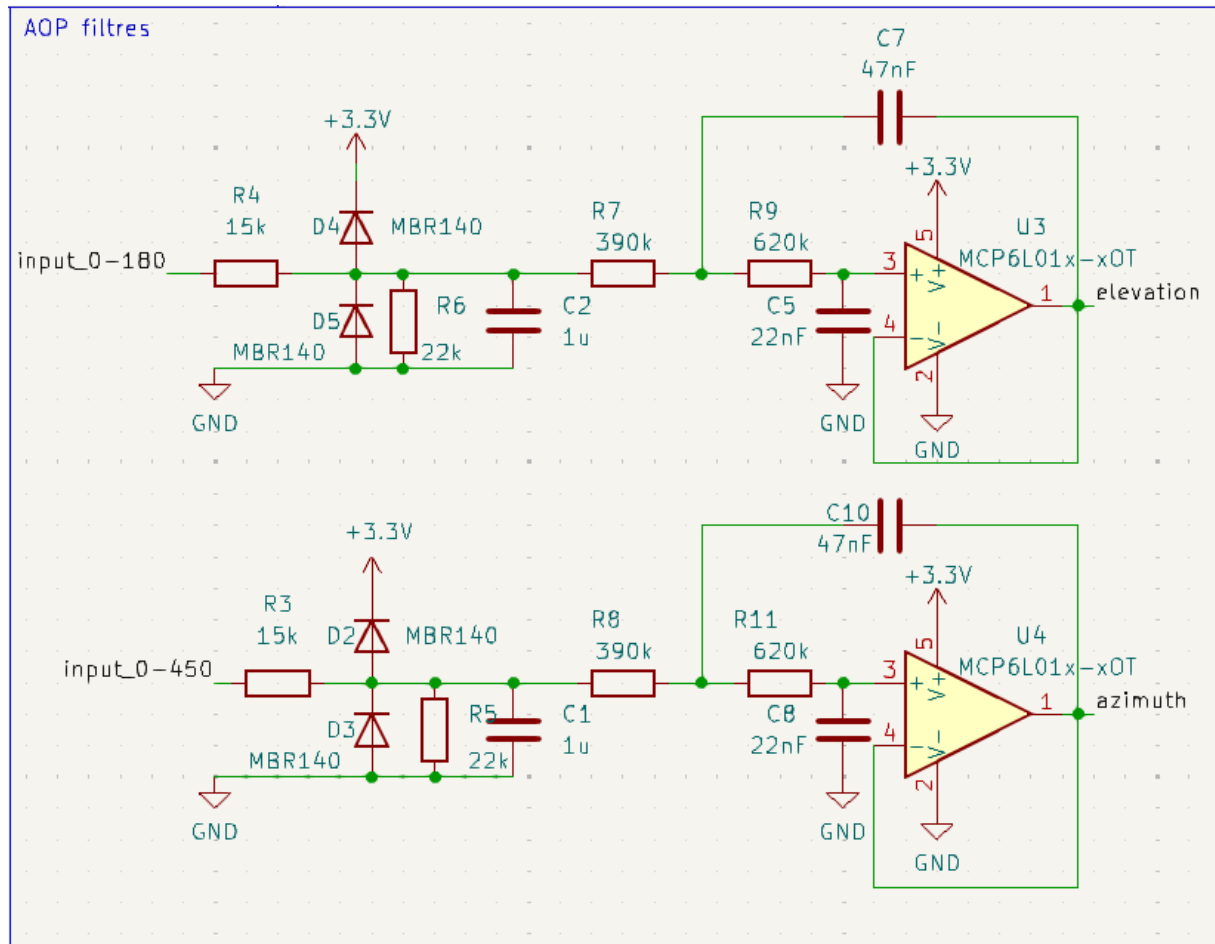
L'objectif principal de la carte est de permettre de piloter les moteurs afin que l'antenne suive automatiquement un satellite, le tout en Ethernet (dont le choix de la Nano Pi). Mais pour que l'on puisse atteindre cet objectif, il faut regarder la carte en détail pour comprendre son fonctionnement et toutes ses fonctionnalités.

La carte est composée de 6 principales parties. La première et la plus importante, la Nano Pi qui a été présentée ci-dessus. Ensuite, il y a les différents types d'alimentation. Cette carte en contient 3. La première est une alimentation tout ce qu'il y a de plus classique, une alimentation par un câble USB type B (J4 sur l'annexe ci-dessous). On peut aussi alimenter par la prise jack (J1) ou encore par le bornier J3 auquel on branche une alimentation de 8/9 Volts qui est ensuite régulé à 5 Volts grâce au circuit suivant le boîtier.



Annexe 3 : Schéma des 3 types d'alimentation

Une autre des parties est la partie des filtres. Ces filtres (MCP6L01) sont des amplificateurs opérationnels (ampli-op). Dans ce contexte, ils servent à rehausser les niveaux des tensions permettant au contrôleur de connaître la position de l'antenne.

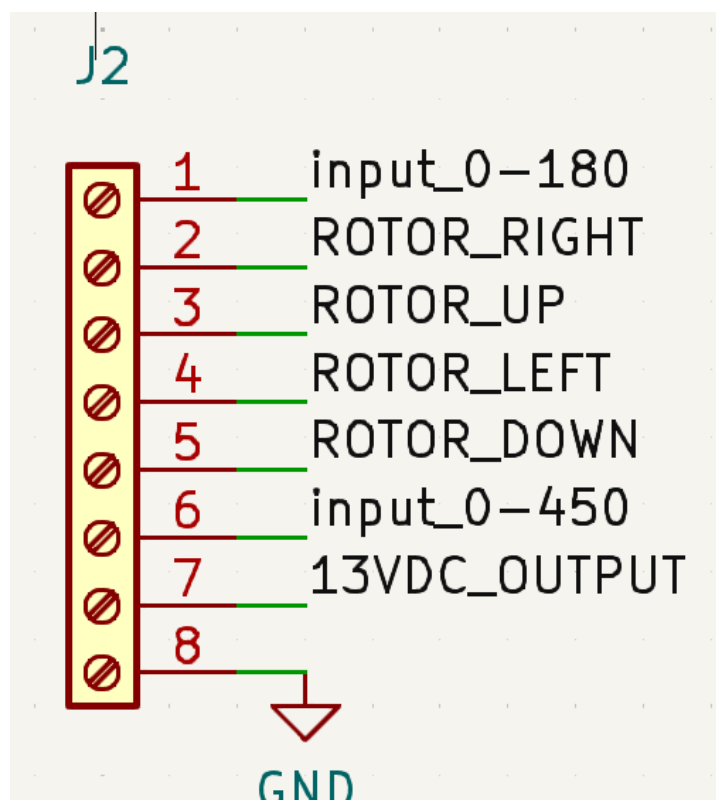


Annexe 4 : Schéma des filtres

Les entrées des filtres correspondent à l'orientation de l'antenne convertie en une tension comprise entre 2 et 4.5 Volts. Sur cette carte électronique, il y a 2 amplificateurs, l'un pour le côté azimuth de l'antenne (orientation gauche et droite de l'antenne) et l'autre pour l'élévation (orientation de l'antenne vers le haut et vers le bas).

Pin	Function
6	Provides 2 to 4.5 VDC corresponding to 0 to 450 °
1	Provides 2 to 4.5 VDC corresponding to 0 to 180 °

Annexe 5 : Schéma des 3 types d'alimentation

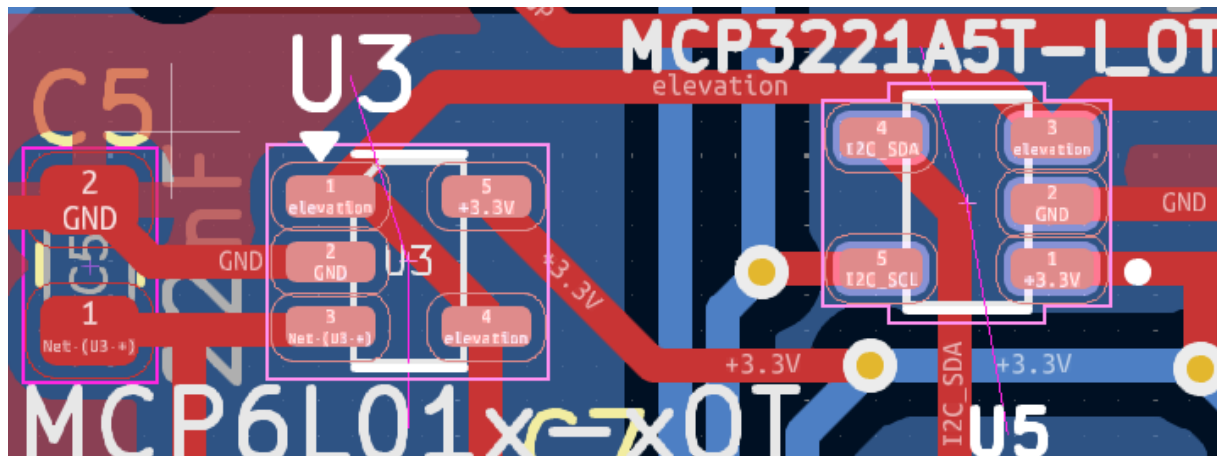


Le bornier J2 permet de récupérer des tensions en sorties des potentiomètres de moteurs pour connaître leurs positions. Le principe est simple, il s'agit de comparer le niveau de tension et de l'associer à une valeur d'angle. Si l'angle est de 0°, alors la tension en sortie du bornier sera de 0. Si l'angle est au maximum, alors la tension sera comprise entre 2 et 4.5V tout dépendant du réglage du G-5500. Il faut donc prendre en compte ce réglage pour connaître la position exacte de chaque moteur.

Annexe 6 : Connecteur J2 reliant la carte au moteur



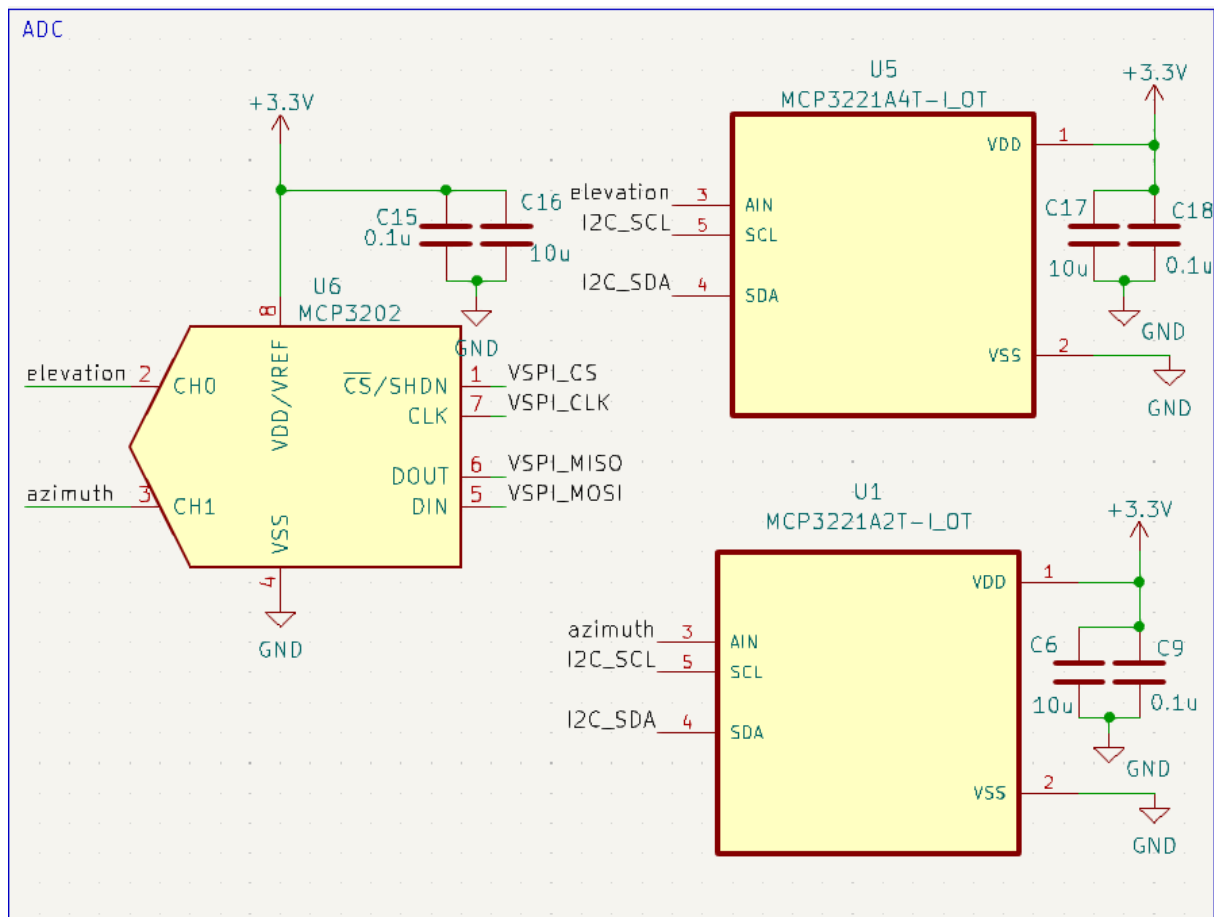
Le filtre AOP suiveur permet de recevoir la valeur sans la modifier, le tout en filtrant les interférences comme le bruit. Cela améliore le niveau de tension reçu. Pour éviter de perdre le signal, ou d'avoir un niveau de signal trop faible, les filtres sont placés au plus proche des ADC sur le schéma.



Annexe 7 : Placement du filtre (U3) et de l'ADC (U5) sur la carte

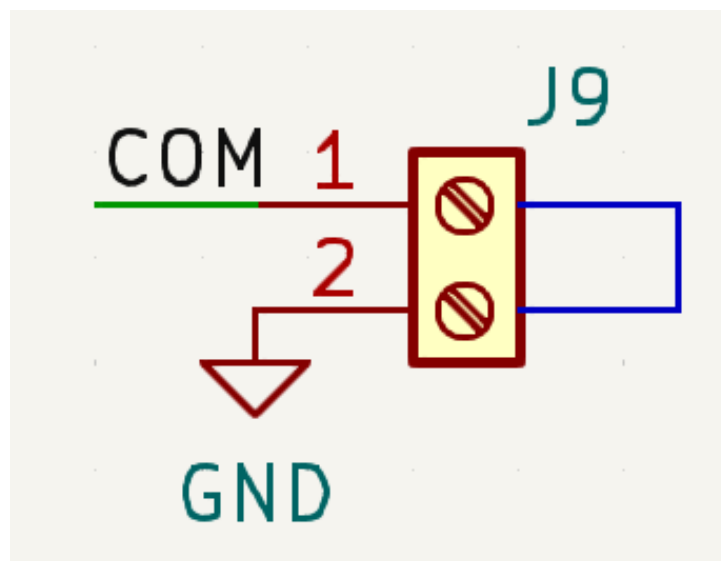
Les ADC sont utilisées dans le but de convertir les données (tension des moteurs, donc position de l'antenne) en données numériques afin de pouvoir les comparer avec les valeurs permettant de suivre le satellite. Des corrections sont ensuite transmises au G-5500 (le moteur de l'antenne) pour faire bouger les moteurs au bon endroit.

Sur la carte, sont disposés 2 différents types d'ADC. Il y a une communication I2C par les ADC (MCP3221). Ils sont aux nombres de 2, l'un dédié à la partie azimut et l'autre pour la partie élévation. C'est ce type de communication qui est en place sur la carte électronique. En effet, l'autre type de communication, la communication SPI (le MCP3202), qui est présente en un seul exemplaire, car ce composant est composé de 2 canaux, l'un pour l'élévation et l'autre pour l'azimut. Cette communication n'est pas disponible, car le composant n'était pas en stock lors de ce projet. Il n'a donc pas été testé.



Annexe 8 : ADC présent sur la carte

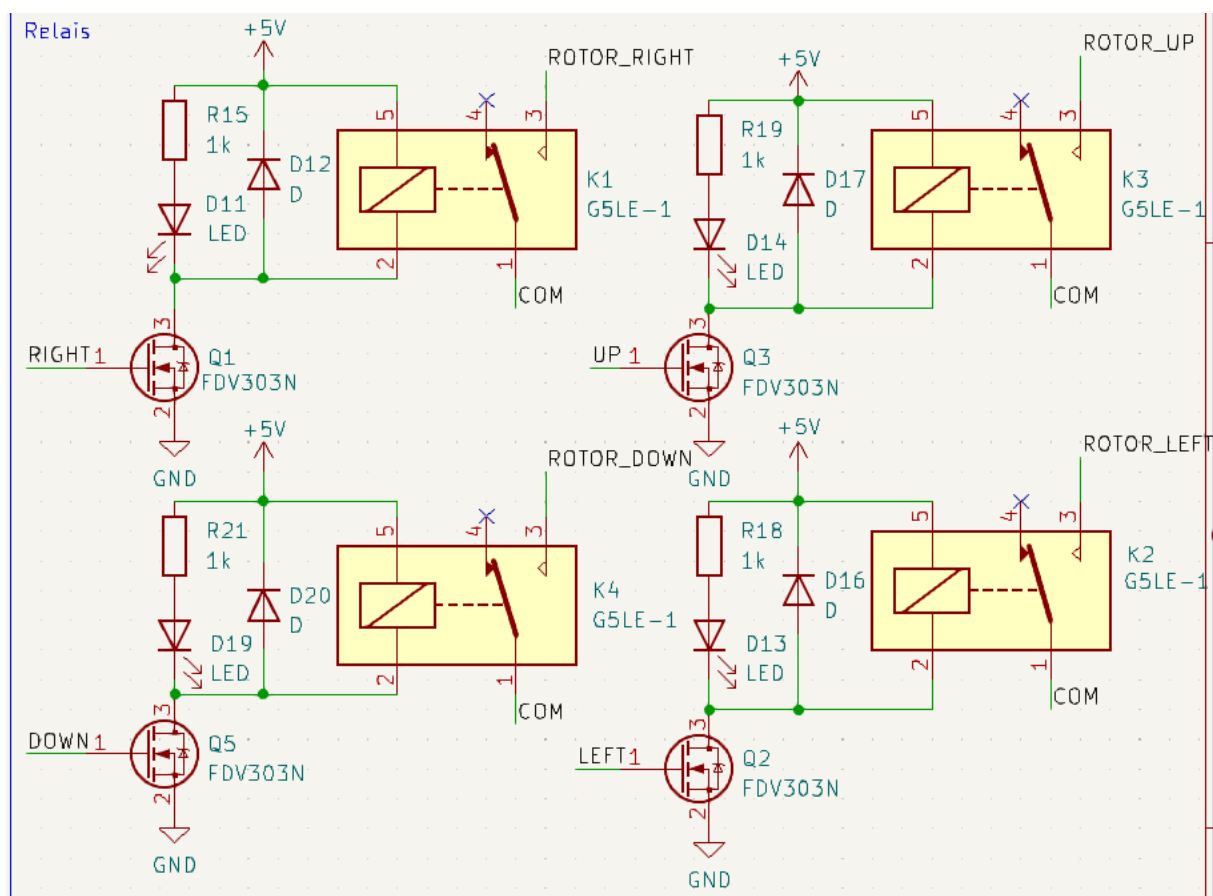
Un bornier (le bornier J9) est un bornier en mode « loopback », qui permet de réaliser un retour à la masse pour tous les relais. Il permet aussi de bloquer le système si nécessaire.



Annexe 9 : Bornier permettant le « loopback »

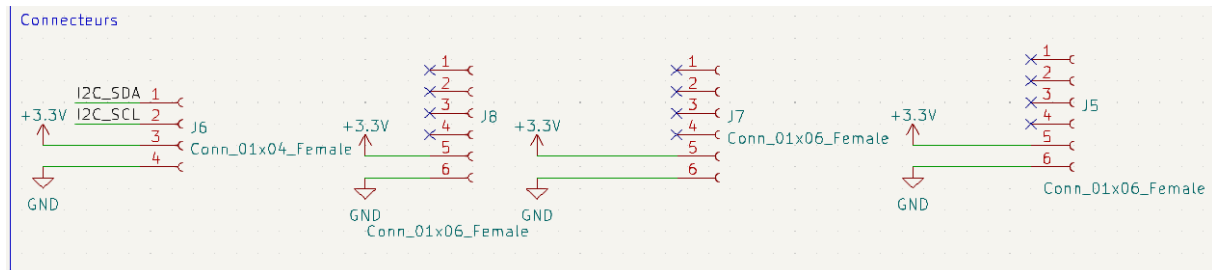
Afin de pouvoir piloter les moteurs à distance, il y a quatre relais en parallèle des boutons du G-5500. Ces relais sont accompagnés par des leds, de diode et de mosfet. Les diodes permettent de protéger les surtensions. Les leds permettent d'avoir un aspect visuel sur le fonctionnement des moteurs. Chaque relais a une led qui correspond à son fonctionnement, et donc à un moteur.

Les relais sont utilisés pour activer ou désactiver chaque moteur individuellement (4 relais pour les 4 moteurs et donc les mêmes directions qu'expliquer précédemment). Ils agissent comme des interrupteurs qui activent et désactive les moteurs pour gérer leur direction. L'avantage des MOSFET et qu'ils peuvent commuter très rapidement, ce qui est essentiel pour le contrôle précis des moteurs.



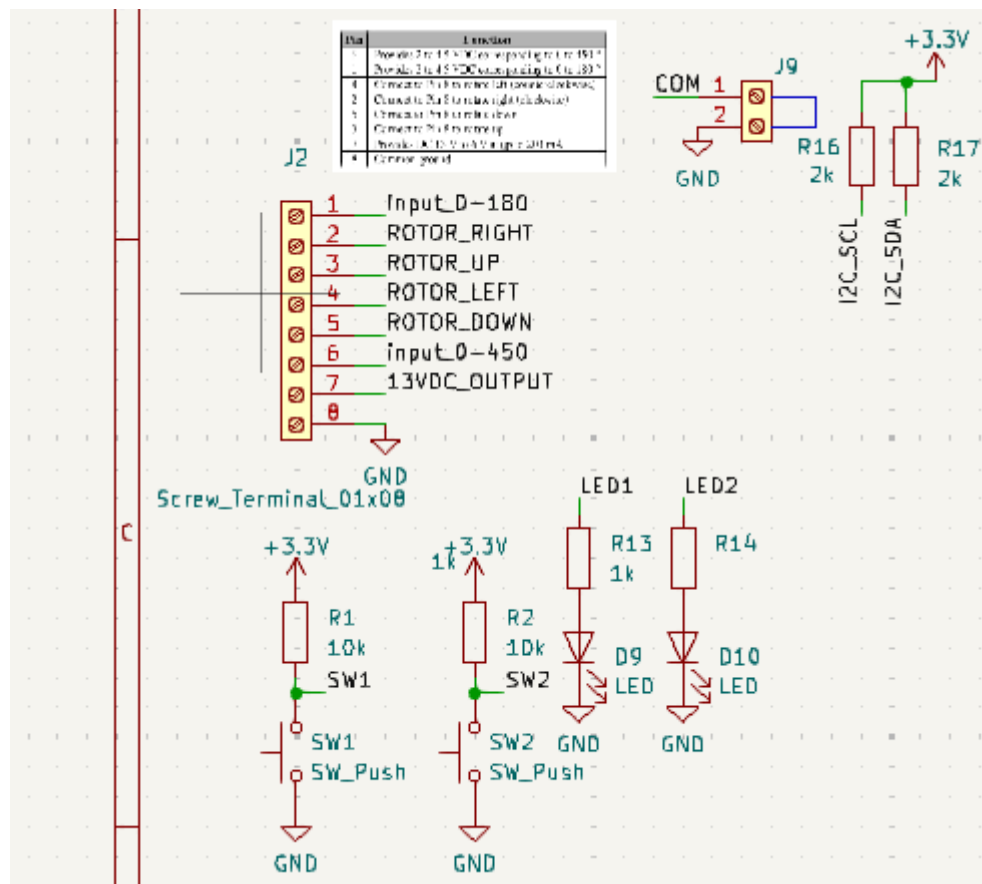
Annexe 10 : Schéma des relais

D'autres ajouts sont présents sur la carte comme 4 connecteurs (J5, J6, J7 et J8). Ces connecteurs permettent de réaliser des tests. De plus, tous ces connecteurs ont une broche de masse associée avec une broche alimentation de 3.3V. Cela permet d'avoir une alimentation supplémentaire pour réaliser divers tests.



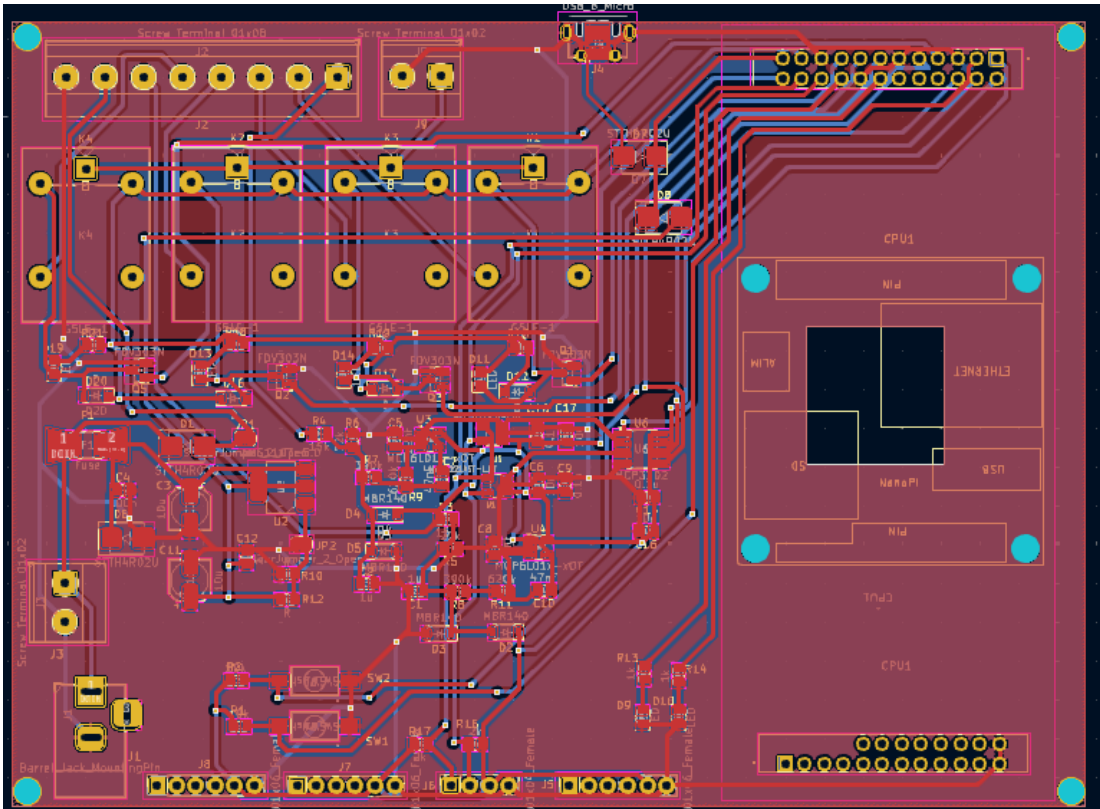
Annexe 11 : Connecteurs supplémentaires

Deux leds sont présentes sur la carte qui ont permis de visualiser le fonctionnement de certains éléments sur la carte durant la phase de conception et de test de cette carte, idem pour les boutons qui servent uniquement pour les tests. Enfin, le bornier J2 permet de faire le lien entre la carte électronique et le moteur de l'antenne.

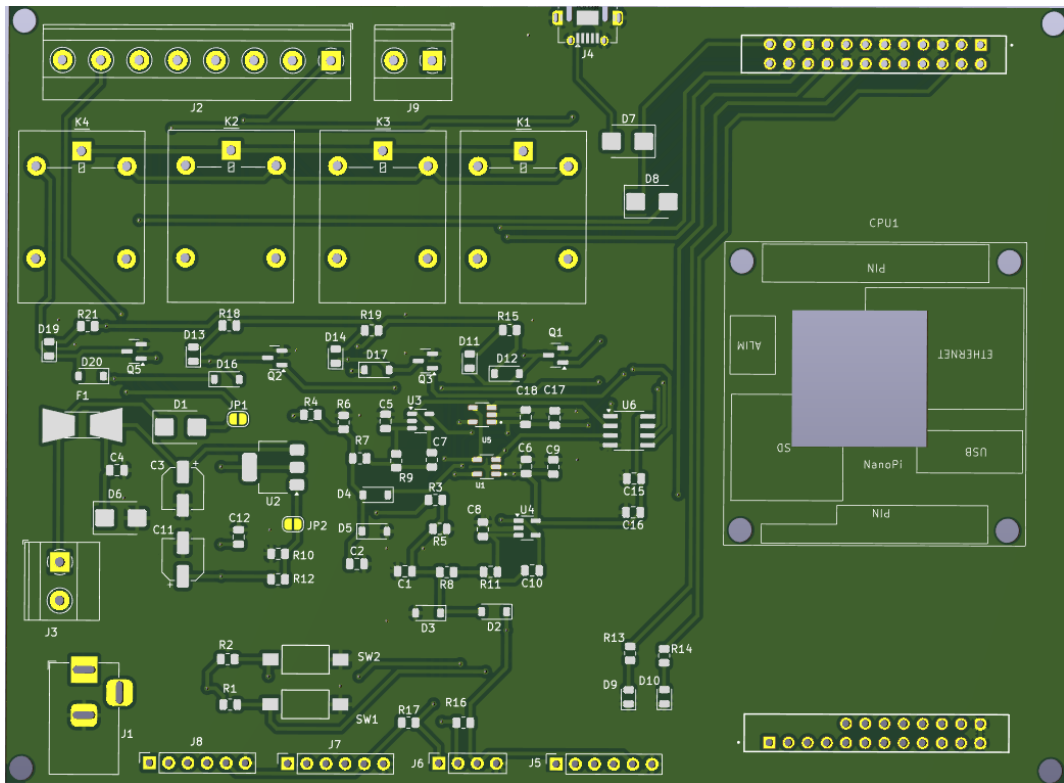


Annexe 12 : derniers ajouts présents sur la carte

Enfin, voici la carte électronique complète et sa visualisation 3D.



## Annexe 13 : Carte finale



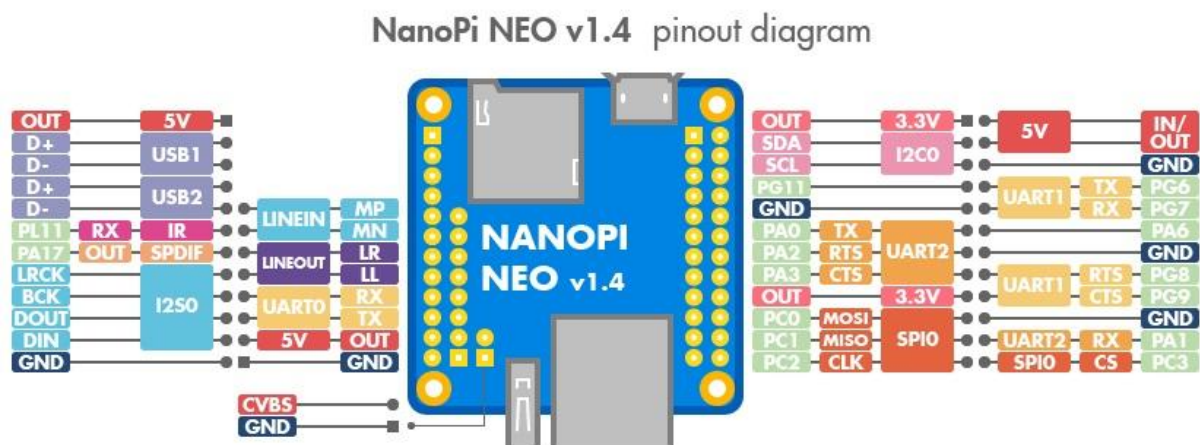
## Annexe 14 : Visualisation 3D

Une fois le schéma et le PCB achevé, les fichiers de créations ont alors été transmis afin de pouvoir procéder à la fabrication. Une fois la carte arrivée, il fut possible de passer à l'étape d'assemblage et de vérification des différents composants de la carte. Des tests de continuités, de tensions et de consommation ont été menés afin d'attester de la bonne réalisation de la carte.

### 3. Mise en place de la Nano Pi Neo

#### 3.1 Attribution des broches sur le schéma

Afin que la Nano Pi puisse interagir avec la carte précédemment conçue, il faut commencer par comprendre la disposition des différentes entrées et sorties du mini-ordinateur. Le site FriendlyElec, sur lequel est vendu cette carte, fournit alors le schéma suivant :



Selon celui-ci, la carte dispose déjà de pins dédiés à l'utilisation de la norme SPI, ainsi que deux broches pour la norme I2C. Les 3 ADC présents sur la carte électronique précédente ont donc été rattachés à la Nano Pi en accord avec la disposition présente ci-dessus. Le contrôleur dispose aussi de plusieurs entrées/sorties d'UARTS lui permettant de réaliser de la communication par broches RX-TX et donc l'utilisation d'un port série. La NanoPi Neo dispose aussi de plusieurs sorties de tensions en 3.3V et en 5V ainsi que d'entrer dans ces mêmes tensions. Cela permettra de l'alimenter directement depuis la carte électronique et de fournir la tension nécessaire ou autres composants. La dernière incertitude réside dans le branchement des pins numériques qui permettent de contrôler les relais. Ce sont quatre sorties numériques qui partent du contrôleur jusqu'à un transistor qui permet de faire commuter le relais correspondant lorsqu'on lui injecte une certaine tension. Le site FriendlyElec ainsi que plusieurs autres permettent d'indiquer que les broches PA, PG, et PC peuvent être définies en entrées ou en sorties au bon vouloir de l'utilisateur (à l'exception de PA17 pour laquelle il n'est pas possible de confirmer cette affirmation).

En effet, les broches PC et PG correspondent à des pins pouvant être définies comme des entrées et des sorties numériques, tandis que les broches PA correspondent aux entrées et aux sorties analogiques. Dans le cas de ce projet, il n'y a pas le besoin d'utiliser de broches analogiques, car les informations venant du boîtier du moteur seront traitées par les ADC SPI et I2C, et que les données sortantes de la Nano Pi ne demanderont pas l'usage d'une PWM étant donné que le moteur sera piloté depuis les relais. Le choix sera donc fait de n'utiliser que 4 pins numériques pour raccorder la Nano Pi aux transistors qui contrôlent les relais.

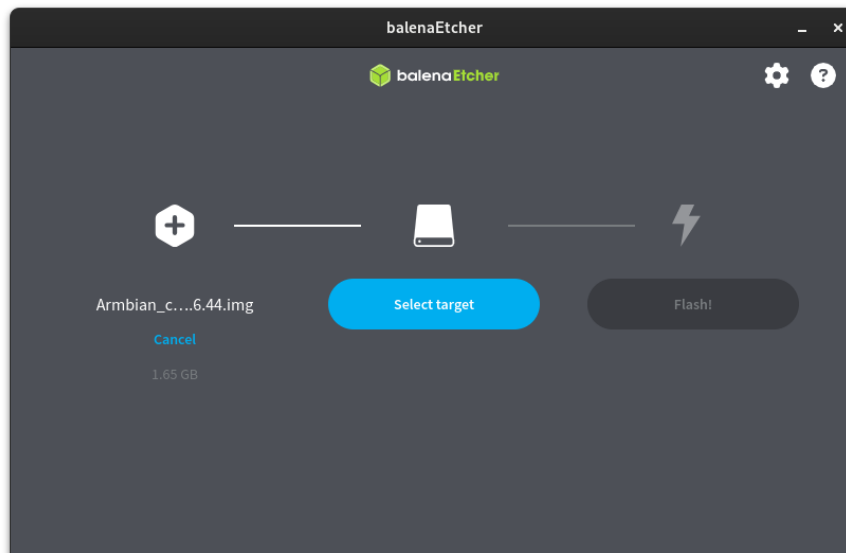


## 3.2 Installation de l'os et initialisation

La Nano Pi Néo vendue par FriendlyElec arrive sans être pourvu d'un système d'exploitation, il faut donc en ajouter un. Les mini-ordinateurs de ce genre fonctionnent généralement sous des versions modifiées et adaptées de Linux. Dans le cas des RaspBerry, il s'agit d'un système d'exploitation modifié par le constructeur afin qu'il soit parfaitement adapté à la carte. Dans le cas de la Nano Pi Neo, il n'y a pas de système particulièrement indiqué pour son usage. Cependant, il existe une multitude de distribution Linux spécialisée dans les systèmes embarqués. Ces versions sont ajustées pour prendre le moins d'espace possible sur le stockage, et pour fonctionner en utilisant le minimum de ressources. Parmi ces distributions, on retrouve notamment Armbian. Armbian se base sur Ubuntu, mais retire tout l'aspect graphique et supprime une grande partie des extensions pour ne laisser que l'essentiel et permettre à une carte comme la Nano Pi Neo de pouvoir fonctionner en utilisant le terminal de commande. C'est donc le choix qui fut retenu.

L'installation d'Armbian est assez simple, il faut d'abord se rendre sur le site de <https://www.armbian.com/nanopi-neo/>. Le site indique qu'il s'agit d'une version prévue pour un usage de serveur et qu'elle inclue tous les « packages » permettant de travailler en réseau. Cela correspond donc parfaitement à ce qui est recherché pour le projet.

La Nano Pi Neo utilise un stockage par carte Micro SD, il faut donc mettre la distribution Armbian dans une carte Micro SD qui sera ensuite mise dans la Nano Pi Neo, c'est une tâche réalisable facilement grâce au logiciel BalenaEtcher. Son interface est la suivante :



Annexe 16 : Interface de BalenaEtcher

Son usage est assez simple, il suffit d'indiquer le fichier que l'on souhaite transférer, puis dans quel périphérique (dans ce cas, il s'agit d'un adaptateur Micro SD vers USB). Le logiciel se charge ensuite d'extraire les fichiers de l'archive zip et de les installer sur la carte Micro SD. Une fois le transfert effectué, il est alors possible de mettre le stockage dans la Nano Pi et de passer à la configuration du système d'exploitation.

Après avoir alimenté la carte, il faut se brancher sur cette dernière depuis un ordinateur (de préférence sous Linux), grâce au logiciel GTK Term qui permet d'observer ce qu'il se passe sur le terminal de commande de la Nano Pi. Lors du premier démarrage, le programme d'installation demande de renseigner plusieurs informations qui sont les suivantes :

- Le mot de passe du super utilisateur root (Il s'agit du compte administrateur)
- Le nom et le mot de passe d'un utilisateur « standard »
- La langue et la position géographique de la Nano Pi

Ces informations devront être entrées dans le terminal de commande GTK Term et elles seront les suivantes :

- Mot de passe de root : geii
- Nom de l'utilisateur standard : geii
- Mot de passe de l'utilisateur geii : geii
- Langue : EN (anglais)
- Position : Europe/France

Bien que très peu sécuritaire, le mot de passe du super utilisateur et de geii sont les mêmes afin de simplifier les opérations et l'échange du projet entre différents groupes.

Une fois que le programme dispose de ces informations, le terminal demande le mot de passe de geii puis affiche une « page d'accueil » qui indique les capacités du système. La phase de configuration et d'initialisation est donc finie, la Nano Pi fonctionne maintenant sous Armbian et est prête pour la suite du projet.

### 3.3 Configuration du réseau

Pour que la Nano Pi puisse contrôler le moteur, il faut d'abord qu'elle puisse utiliser les différentes liaisons avec la carte électroniques conçue précédemment. Pour commencer, afin de simplifier la connexion au terminal de la Nano Pi, il faut commencer par configurer la connectivité réseau de celle-ci. Etant donné que la carte sera utilisée sur le réseau Ethernet de l'IUT, il faut lui donner une adresse IP qu'elle va garder après chaque redémarrage. Il faut aussi configurer le proxy afin qu'elle puisse installer depuis des serveurs externes. Les étapes sont alors les suivantes :

Paramétrage du proxy :

La ligne « Acquire::http::Proxy "http://192.168.0.1:3128" ; » doit être ajoutée à la fin dans un fichier proxy dont le chemin d'accès sera le suivant : /etc/apt/apt.conf.d/proxy . Pour pouvoir créer un tel fichier, la commande à utiliser est :

```
“sudo nano /etc/apt/apt.conf.d/proxy”
```

Maintenant que le proxy est configuré, il est possible d'installer des packages comme net-tools qui donne accès à la commande « ifconfig » permettant d'afficher des informations sur le réseau de la carte. La prochaine étape est le passage en IP fixe. La commande utilisée au cours des prochaines étapes sera « nmcli » qui permet elle aussi d'obtenir des informations sur la configuration du réseau de la Nano Pi en ajoutant des paramètres :

```
« nmcli connection show “Wired connection 1” ».
```

Il est possible de trouver des informations concernant l'ipv4 de la carte et de voir que celui-ci est de base en mode automatique. L'adresse choisie pour la Nano Pi sera la suivante : 192.168.0.199. Cette adresse a été définie afin de respecter les emplacements disponibles sur le réseau de l'IUT. La commande pour modifier l'adresse ipv4 est donc :

```
« nmcli connection modify "Wired connection 1" ipv4.addresses  
192.168.0.199/24 »
```

Puis, pour changer le mode d'attribution vers manuel :

```
« nmcli connection modify "Wired connection 1" ipv4.method manual »
```

La carte à maintenant l'adresse 192.168.0.199 et elle ne changera plus après un redémarrage. Il est donc maintenant possible de passer par une connexion en ssh depuis le terminal de commande d'un ordinateur du réseau de l'IUT.

### 3.4 Ajout des packages

Maintenant que la carte est connectée et accessible depuis le réseau, il faut ajouter toutes les extensions qui permettent d'utiliser les fonctionnalités de la carte.

Pour commencer, si l'on souhaite pouvoir communiquer via SPI et I2C, il faut alors ajouter les packages suivants :

- spi-tools
- i2c-tools

Puis, pour configurer ces éléments, il faut alors utiliser la commande suivante :

« armbian-config »

Cette commande ouvre une nouvelle page dans laquelle il est possible de naviguer avec les touches du clavier. Le chemin à suivre est « Système > HardWare », puis il faut cocher les cases i2c0 et spi-spidev.

Une fois ceci fait et les modifications enregistrées, le logiciel propose de faire un reboot de la carte, ce qui doit être fait.

Pour vérifier que l'I2C fonctionne, il faut alors connecter un ou plusieurs module I2C sur les broches correspondantes et utiliser la commande suivante :

« sudo i2cdetect 0 »

Cela va afficher un tableau dans lequel devrait se trouver les valeurs des identifiants des modules I2C connectés.

Pour le SPI, la version d'Armbian utilisée n'a pas décidé de reconduire la fonctionnalité spidev, il n'est donc pas possible d'utiliser le SPI sans avoir à recoder l'ensemble du protocole à la main. L'ADC SPI ne sera donc pas utiliser lors de ce projet et seul les 2 ADC I2C le seront.

Enfin, les programmes utilisés pour le contrôle du moteur seront réalisés via Python, il faut donc installer Python 3 et quelques librairies (Periphery, Time, SMBus, Socket). Pour ce faire :

« sudo apt install python3 »

Pour les librairies :

« sudo apt install python3-nomdelalibrairie »

## 4. Programmes de la carte

Le premier programme python à être mis en place est « pinassign.py » ce programme permet de configurer les broches GPIOs de la Nano Pi.

La librairie Periphery comporte une classe d'éléments appelée GPIO et qui permet de faire l'initialisation d'une broche en une ligne. Par exemple, l'initialisation des pins numériques ressemble à ceci :

```
gpio0 = GPIO("/dev/gpiochip0",64,"out")
gpio1 = GPIO("/dev/gpiochip0",65,"out")
gpio2 = GPIO("/dev/gpiochip0",66,"out")
gpio3 = GPIO("/dev/gpiochip0",67,"out")
gpio6 = GPIO("/dev/gpiochip0",198,"out")
gpio7 = GPIO("/dev/gpiochip0",199,"out")
gpio8 = GPIO("/dev/gpiochip0",200,"out")
gpio9 = GPIO("/dev/gpiochip0",201,"out")
gpio11 = GPIO("/dev/gpiochip0",203,"out")
```

Annexe 17 : Extrait de pinassign.py

Cette initialisation prend comme paramètre le chemin d'accès dans les fichiers Linux, s'il s'agit d'une entrée ou d'une sortie et surtout le numéro de la broche que l'on souhaite initialiser. Ces numéros peuvent être trouvés sur le site FriendlyElec avec le tableau ci-dessous :

Pin#	Name	Linux gpio	Pin#	Name	Linux gpio
1	SYS_3.3V		2	VDD_5V	
3	I2C0_SDA		4	VDD_5V	
5	I2C0_SCL		6	GND	
7	GPIOG11	203	8	UART1_TX/GPIOG6	198
9	GND		10	UART1_RX/GPIOG7	199
11	UART2_TX/GPIOA0	0	12	GPIOA6	6
13	UART2_RTS/GPIOA2	2	14	GND	
15	UART2_CTS/GPIOA3	3	16	UART1_RTS/GPIOG8	200
17	SYS_3.3V		18	UART1_CTS/GPIOG9	201
19	SPI0_MOSI/GPIOC0	64	20	GND	
21	SPI0_MISO/GPIOC1	65	22	UART2_RX/GPIOA1	1
23	SPI0_CLK/GPIOC2	66	24	SPI0_CS/GPIOC3	67

## Annexe 18 : Tableau de correspondance des GPIO

La méthode est la même pour les pins analogiques.

Ce premier programme sera exécuté par la Nano Pi après chaque démarrage. Pour cela, il faut modifier le fichier rc.local de la façon suivante :

« sudo nano /etc/rc.local »

Puis il faut ajouter la ligne suivante au programme :

« /usr/bin/python3 /home/geii/pinassign.py »

Ou bien :

« python3 pinassign.py »

Après avoir fait toutes ces étapes, le programme sera lancé une fois à chaque démarrage de la Nano Pi permettant ainsi d'initialiser les pins et de les remettre à zéro.

Le second programme à être mis en place dans la Nano Pi est celui qui permet de mettre en place un serveur TCP/IP. Effectivement, le projet précédent pouvait être contrôlé depuis un n'importe quel poste qui était connecté au même réseau que l'esp32. Pour pouvoir en faire de même ici, il faut alors de nouveau héberger un serveur TCP/IP sur la Nano Pi pour qu'elle puisse recevoir les commandes émises depuis un autre ordinateur.

Pour mettre en place un serveur de ce type en pytho, il faut utiliser la librairie socket qui permet de gérer les processus de communication entre un serveur et des clients. Le début du programme serveur.py est le suivant :

```
import time
import socket

ADRESSE = ''
PORT = 6790

serveur = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serveur.bind((ADRESSE,PORT))
serveur.listen(1)
client, adresseClient = serveur.accept()
print(f"Connection de : {adresseClient}")
```

Annexe 19 : Extrait du début du programme serveur.py

Cette première partie de programme permet de définir l'adresse qui est utilisée, ainsi que le port de communication dédié. Dans le cas de ce projet, le serveur sera local à la Nano Pi, il n'y a donc pas d'adresse à renseigner, car il s'agira du localhost de la Nano PI. Pour le port, il suffit de prendre un port libre qui permet de faire ce genre d'échange.

Ensuite, la mise en place du socket permet de créer un objet serveur dans lequel on injecte les paramètres précédemment indiqués via le .bind(). Le serveur se met ensuite en écoute.

Il faut aussi autoriser l'accès à des clients et récupérer leurs adresses afin de pouvoir communiquer en retour.

La suite du programme se constitue d'une boucle While permettant de garder le serveur en vie jusqu'à ce que la commande « fin » soit envoyée par le client, ou que celui-ci se déconnecte. En cas de déconnexion, le programme sort de la boucle pour arriver à cette partie :



```
print("Fermeture de la connexion avec le client")
client.close()
print("Arret du serveur")
serveur.close()
```

Annexe 20 : Extrait de la fin du programme serveur.py

Le programme referme la liaison avec le client puis arrête le serveur.

A l'intérieur de la boucle While, il est possible de retrouver la partie du programme qui se charge de décoder ce que le client transmet au cours de la liaison par Telnet.

```
donnees = client.recv(1024)
if not donnees :
    print("Erreur de réception")
else :
    print(f"Réception de : {donnees}")
    donnees = donnees.upper()
    commande = donnees.decode("utf-8")
    file = open("transfert.txt", "w")
    file.write(commande)
    file.close()
    donnees = donnees.strip()
```

Annexe 21 : Extrait de la boucle While

Lorsque le client envoie une commande, le serveur se charge de convertir l'ensemble des lettres minuscules en majuscules afin d'éviter la casse lors du traitement des requêtes. Le serveur inscrit ensuite cette commande dans un fichier transfert.txt qui permet de faire un lien entre lui et le programme qui agit sur le moteur.

Enfin, plusieurs cas de commandes sont répertoriés dans la boucle While, ces cas nécessitent des actions supplémentaires de la part du serveur, comme de la lecture d'information et la retransmission de celles-ci vers le client. Par exemple le cas de la commande C :

```

if donnees == b"C" :
    time.sleep(0.2)
    fileaz = open('Azimut.txt','r')
    azimuth = fileaz.read()
    azimuth = azimuth.strip()
    fileaz.close()
    print(f"Azimuth : {azimuth}")

```

## Annexe 22 : Extrait de la boucle While pour C

Si le client envoie la commande C vers le serveur, alors celui-ci commence par décoder le texte reçu et l'inscrit dans le fichier transfert.txt, puis il lit le fichier Azimut.txt dans lequel se trouve la position en azimuth du moteur transmise par le programme python controlrotor.py. Le programme traite ensuite cette information avant de la transmettre au client. Il existe ainsi une commande C2 qui effectue la même opération, mais qui inclue aussi l'élévation du moteur et une commande W qui permet de définir la position cible que l'antenne doit atteindre. Tous les transferts de données entre serveur.py et controlrotor.py se font par l'intermédiaire de fichier txt dans lesquelles se trouvent les informations. Cette méthode pourra être remplacée par une mémoire partagée permettant d'éviter la réécriture de fichier txt pour chaque échange.

Ce programme permet donc de faire le lien entre le client et le programme controlrotor.py qui permet de faire tous les calculs et les opérations permettant le déplacement de l'antenne.

Ce dernier programme python est le plus important. En effet, c'est dans ce programme que se font l'ensemble des calculs permettant de définir dans quel sens devra se déplacer l'antenne, de savoir où se trouve sa cible et de prendre les décisions adéquates. Ce programme peut être décomposé en plusieurs sections. Il y a d'abord la section qui permet de faire l'acquisition des données en provenance du moteur et de réaliser les conversions, puis la partie qui gère les déplacements du moteur, et enfin, la partie qui réagit permet de convertir les commandes en actions.

La partie d'acquisition des données commence par la mise en place des ADC. Comme dit précédemment, la liaison SPI n'est pas prise en charge par des librairies et des packages sur la version utilisée ici. L'ADC SPI n'est donc pas mis en place dans ce programme, mais il pourra être ajouté une fois qu'une gestion de la communication par SPI aura été implantée dans la Nano Pi. Les seuls ADC

utilisés sont donc les I2C. Leur implantation se fait via la librairie Python SMBus. Cette librairie ajoute des fonctions qui permettent de mettre en place la communication avec les ADC en ne se basant que sur leurs adresses respectives. Cette initialisation se fait de la façon suivante :

```
i2c_ch = 0
ELaddress = 0x4c
AZaddress = 0x4a

bus = smbus.SMBus(i2c_ch)
bus.write_byte(AZaddress, 0x02)
bus.write_byte(ELaddress, 0x02)
```

#### Annexe 24 : Initialisation de la liaison I2C dans controlrotor.py

La première étape est donc d'indiquer l'adresse relative à chacun des deux I2C. Puis, le module SMBus permet de créer un objet bus qui contient l'ensemble des fonctions permettant de communiquer avec les ADC. Pour commencer la communication, il suffit alors d'envoyer des données vers les I2C pour qu'ils entament la discussion avec la Nano Pi.

Les ADC permettent de convertir les tensions de sortie de la boîte de contrôle du moteur en valeur numérique codée sur 12 bits, soit une valeur maximale de 4096. Pour récupérer cette valeur, la librairie SMBus dispose de la commande :

```
bus.read_word_data()
```

Cette commande retourne la valeur sur 12 bits. Lorsqu'on tente d'afficher cette valeur, on constate assez vite que la valeur qui devra normalement caper à 4096 à une valeur s'approchant des 64 000. La raison pour cette valeur anormalement élevée est que la lecture de ces bits ne s'effectue pas dans le même sens que ce que le constructeur des ADC l'écrit. Cela signifie que le bit de poids fort devient le bit de poids faible et vice-versa. Pour contrer cela, il faut alors ajouter une étape de retournement des bits pour les remettre dans le bon sens. Sur Python, il existe une commande `.tobytes()` qui prend en paramètres la donnée à convertir, et le sens de conversion, il y a deux sens possibles, le Little Endian, et le Big Endian. Après plusieurs essais, il est possible de conclure que le sens qui permet la lecture de la valeur souhaitée est le Big Endian. La conversion en entier (int) et l'inversion des bits s'effectuent de la façon suivante :

```

azimuth = bus.read_word_data(AZaddress,0)
azimuthbytes = (azimuth.to_bytes(2,'little'))
azimuthint = (int.from_bytes(azimuthbytes,'big'))
elevation = bus.read_word_data(ELaddress,0)
elevationbytes = (elevation.to_bytes(2,'little'))
elevationint = (int.from_bytes(elevationbytes,'big'))

```

#### Annexe 25 : Inversion et conversion des valeurs

En procédant de la sorte, la valeur est inversée pour correspondre au sens Big Endian et elle adopte le type int et elle est donc maintenant exploitable par le reste du programme.

Dans l'état actuel, il est tout à fait possible d'utiliser la valeur telle quelle pour faire les comparaisons avec la cible et déplacer l'antenne. Or, cela implique qu'il faut convertir les coordonnées de la cible en valeur numérique sur 12 bits et que les calculs devront traiter des chiffres abstraits. Pour rendre les calculs plus faciles à comprendre et pour permettre de clairement identifier les opérations effectuées, les valeurs récupérées sur les ADC seront alors converties en degrés. Il sera alors possible de comparer des degrés avec des degrés et donc de simplifier la partie de gestion des commandes. Ne connaissant pas directement le lien entre les degrés du moteur et la valeur numérique, il faut alors passer par des intermédiaires. Deux fonctions doivent être utilisées pour passer de numérique à degrés. D'abord, la fonction qui fait le lien entre la valeur lue par la Nano Pi et la tension de sortie du boîtier de contrôle du moteur. Puis, la fonction qui établit le lien entre la tension de sortie de ce boîtier et la position du moteur. Pour ce faire, une première mesure est alors effectuée sur la carte électronique avec l'aide d'une alimentation stabilisée, et de la Nano Pi. En faisant varier la tension et en relevant les valeurs numériques associées, il est possible de tracer une courbe sur Excel et de demander à l'application de donner son équation approchée. Deux fonctions sont donc obtenues correspondant à un des 2 ADC I2C. Puis, une opération similaire est menée, mais cette fois-ci en utilisant un voltmètre sur les sorties de tensions du moteur. Ainsi, avec les extremums de chacun des axes, il est possible de tracer deux courbes et d'obtenir leurs équations respectives.

En mettant ces équations l'une dans l'autre, il est donc maintenant possible d'associer une valeur à un angle et donc de poursuivre.

Ces équations cumulées sont alors les suivantes :

Pour l'élévation :

$$\text{Elévation} = 32.979 \times (0.013 \times \text{Enum} - 0.0084)^\circ$$

Pour l'azimut :

$$\text{Azimut} = 88.373 \times (0.0013 \times \text{Anum} - 0.0023)^\circ$$

L'intégration de ces calculs se fait de la façon suivante :

```
print(f"Azimuth brut : {azimuth}, Elevation brut :  
  
azimuthvolt = 0.0013*azimuthint - 0.0084  
elevationvolt = 0.0013*elevationint - 0.0023  
  
print(f"Azimuth tension : {azimuthvolt}, Elevation  
  
azimuthdegre = azimuthvolt * 88.373  
elevationdegre = elevationvolt * 32.979  
  
azimuthreel = round(azimuthdegre,2)  
elevationreel = round(elevationdegre,2)
```

#### Annexe 26 : Conversion des valeurs en degrés

Une fois ce souci de conversion réglé, il est possible de passer à la partie de contrôle des axes du moteur.

L'antenne doit pouvoir être contrôlée dans différents modes. Le premier est le mode manuel, dans ce mode, l'utilisateur envoie des commandes type « R », « L », « U » ou « D » pour activer la rotation du moteur. Cette rotation continue jusqu'à ce que la commande « A » soit reçue, ou que l'on change le mode de contrôle de la station. La commande « A » ordonne un arrêt total du moteur et passe la station en mode standby puis en mode manuel. Ce mode manuel permet donc de changer la direction de l'antenne de façon peu précise, cela peut être utile à la fin d'une séance d'utilisation pour ranger l'antenne dans une position sécurisée.

Le second mode de fonctionnement de la station est le mode automatique. Dans ce mode, l'utilisateur envoie les coordonnées d'une cible et la station se déplacera jusqu'à atteindre la position renseignée. Dans les faits, une zone morte, ou une marge d'erreur a été intégrée à proximité des coordonnées de la cible afin d'éviter

les oscillations du moteur en cas de dépassement ou de perturbations dues au vent. Dans le programme, cette marge prend la forme suivante :

```
if azimuthreel < (azimuthcible - deltaaz) :  
    LEFT = 0  
    RIGHT = 1  
elif azimuthreel > (azimuthcible + deltaaz) :  
    RIGHT = 0  
    LEFT = 1
```

Annexe 27 : Mise en place de la marge d'erreur

Dans les deux cas, la méthode permettant d'actionner les relais reste la même :

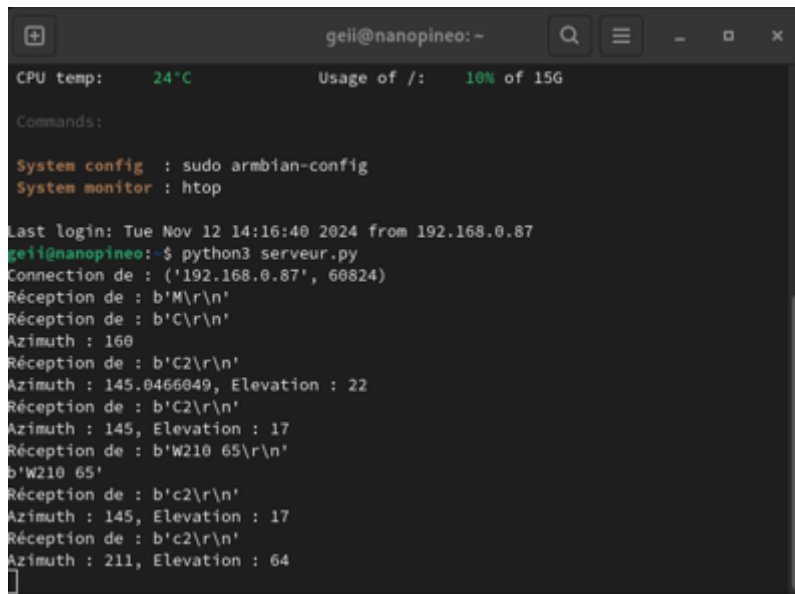
```
if LEFT == 1 :  
    gpio7.write(True)  
else :  
    gpio7.write(False)
```

Annexe 28 : Exemple du contrôle d'un relai

Comme pour le programme serveur.py, les actions que peut effectuer la station sont définies via des conditions if et elif. Les fonctions C et C2 font donc l'opposer de leurs homonymes dans le programme du serveur. Et la fonction W récupère les informations transmises dans les fichiers de target afin de les définir comme nouvelles coordonnées de la cible. Une sécurité est mise en place afin d'éviter que les coordonnées de la cible ne dépassent les limites atteignables par la station.

Avec tous ces programmes, il est donc possible de mettre en œuvre la station. Le processus de démarrage nécessite l'utilisation de deux terminaux de commandes sur la Nano Pi, et d'une liaison Telnet depuis un autre poste, ou depuis la Nano Pi elle-même. Les programmes controlrotor.py et serveur.py doivent être lancés chacun de leurs côtés sur leurs terminaux respectifs.

Lors d'une phase d'utilisation réelle, voici ce à quoi ressemble le terminal où serveur.py est actif :



```
geii@nanopineo: ~
CPU temp: 24°C Usage of /: 10% of 15G

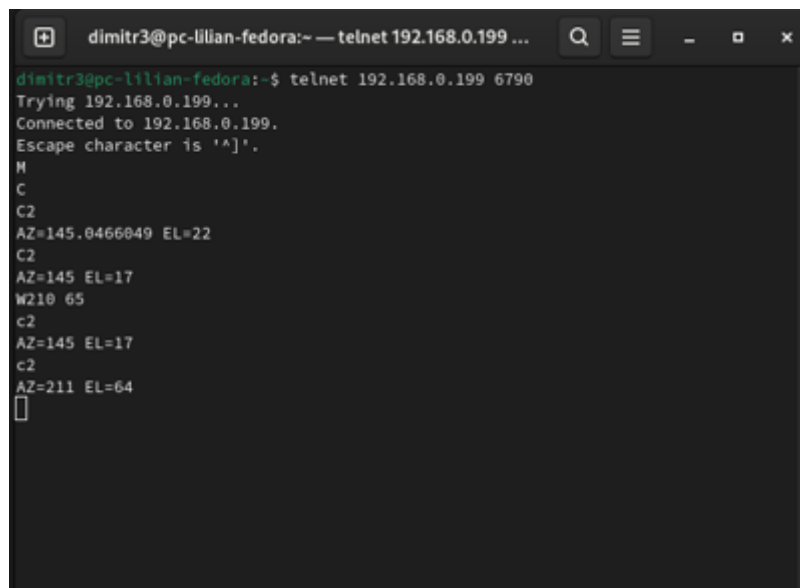
Commands:

System config : sudo armbian-config
System monitor : htop

Last login: Tue Nov 12 14:16:40 2024 from 192.168.0.87
geii@nanopineo:~$ python3 serveur.py
Connection de : ('192.168.0.87', 60824)
Réception de : b'M\r\n'
Réception de : b'C\r\n'
Azimuth : 160
Réception de : b'C2\r\n'
Azimuth : 145.0466049, Elevation : 22
Réception de : b'C2\r\n'
Azimuth : 145, Elevation : 17
Réception de : b'W210 65\r\n'
b'W210 65'
Réception de : b'c2\r\n'
Azimuth : 145, Elevation : 17
Réception de : b'c2\r\n'
Azimuth : 211, Elevation : 64
█
```

Annexe 29 : Terminal serveur.py en fonctionnement

Le serveur indique l'adresse du client qui s'est connecté et le port sur lequel il lui répond. Puis, il indique les caractères reçus et répond au client si cela est demandé. Par exemple, lorsque le client effectue l'envoi de la commande C2, le serveur lit les valeurs d'azimut et d'élévation et les renvoie vers le terminal du client. Du point de vue du client, cela donne donc ceci :



```
dimitr3@pc-lilian-fedora:~ — telnet 192.168.0.199 ...
dimitr3@pc-lilian-fedora:~$ telnet 192.168.0.199 6790
Trying 192.168.0.199...
Connected to 192.168.0.199.
Escape character is '^]'.
M
C
C2
AZ=145.0466049 EL=22
C2
AZ=145 EL=17
W210 65
c2
AZ=145 EL=17
c2
AZ=211 EL=64
█
```

Annexe 30 : Terminal du client en fonctionnement

Le client ne voit que les requêtes qu'il envoie et les réponses du serveur lorsque la commande le demande. Avec le même exemple, la commande C2 indique alors la position en élévation et en azimuth conformément au protocole de communication GS-232 dont sont tiré l'ensemble des commandes. Enfin, du point de vue du programme controlrotor.py voici ce qu'il se passe :

```
Mode : Manuel
Cmd = C2
Azimuth brut : 12551, Elevation brut : 52485
Azimuth bytes : b'\x071', Elevation bytes : b'\x05\xcd'
Azimuth int : 1841, Elevation int : 1485
Azimuth tension : 2.3849, Elevation tension : 1.9282
Azimuth degrés : 210.7607677, Elevation degrés : 63.5901078
Azimuth de la cible : 210, Elevation de la cible : 65
Mode : Manuel
Cmd = C2
Azimuth brut : 12551, Elevation brut : 52485
Azimuth bytes : b'\x071', Elevation bytes : b'\x05\xcd'
Azimuth int : 1841, Elevation int : 1485
Azimuth tension : 2.3849, Elevation tension : 1.9282
Azimuth degrés : 210.7607677, Elevation degrés : 63.5901078
Azimuth de la cible : 210, Elevation de la cible : 65
Mode : Manuel
```

#### Annexe 31 : Terminal controlrotor.py en fonctionnement

Ce programme effectue la dernière action qu'il reçoit en boucle, il affiche donc constamment la position de la cible. Cependant, cet affichage ne se fait pas pour le serveur comme vu précédemment. En effet, le serveur n'affiche les coordonnées du rotor qu'une seule fois après que la commande eu été envoyée. Le client doit donc effectuer plusieurs C2 s'il veut suivre le déplacement de la station en temps réel. Cette façon de fonctionner est faite pour être compatible avec les travaux réalisés par d'autres groupes d'étudiants en parallèle du développement de ce projet. C'est pour cela que le protocole GS-232 sert de référence dans la programmation des différentes commandes. La station est donc pilotable à la main depuis un terminal sérié et une liaison Telnet, ou par le logiciel développé par les autres équipes.



## 5. Conclusion

En conclusion, ce projet de mise à niveau de la station de suivi de satellite a permis de mettre en œuvre les compétences de conception, de vérification et de programmation vues tout au long du BUT. Il aura permis d'approfondir des connaissances et d'en développer de nouvelles, ainsi que de renforcer la capacité à travailler en équipe et en autonomie des étudiants.

Pour ce qui est de la complétion du projet, l'objectif initial était de mettre à jour l'ensemble de la station, c'est-à-dire la partie de contrôle du moteur, mais aussi l'antenne en elle-même. Cependant, par manque de temps et à la suite de retards vis à vis de l'installation dans le nouveau bâtiment, cette partie du projet fut abandonnée. Le reste du projet à quant à lui été mené à bien dans son ensemble, la station est fonctionnelle et permettra aux étudiants des années suivantes de reprendre le travail d'amélioration.