Processus Virvoile

Réalisé le 23/04/2024
à Toulouse

Maîtres d’Ouvrage : Mr LEPINARD Gilles et Mr DATO Bruno
Maîtres d’œuvre : Équipe MGMAT DEV

Projet réalisé dans le cadre de l’unité d’enseignement Gestion de Projet dispensée à l’université Toulouse 3 Paul Sabatier.

Responsable : Gabriel

Approbateurs : Maxime, Thomas

Contributeurs : Anil, Mohammad

Table des matières

1. Liste non exhaustive de composants compatible :

- Raspberry Pi 4 (4Go RAM recommandé avec Open Plotter)
- GPS : VK-162 / VK-172 (USB), M10Q-5883 (I2C)
- Anémomètre : Matek ASPD-4525 (vitesse du vent I2C)
- Gyrocompas : MPU-9250 et tout autres gyrocompas à 9 axes
- Capteurs environnementaux (température, humidité et pression) : BME-280 fait

température, humidité et pression pour pas chers

- Servomoteurs : Servomoteurs étanches comme DC5535
- Unité de stockage : micro sd minimum 8Go (16Go recommandé)

2. Échecs de fonctionnement :

Nous ne sommes pas parvenus à faire fonctionner les capteurs de direction et de vitesse du vent. La réception de données se fait via le port RX or nous ne recevons aucune données provenant de ce port. Nous parvenons en revanche à communiquer via le port TX vers les capteurs.

Avec OpenPlotter SignalK n'accède pas au donnée du capteur environnemental malgré que I2C Sensors d'OpenPlotter récupère bien les données.

3. Instruction d'installation software

3.1. ROS 2 Documentation

Le Système d'Exploitation Robotique (ROS) est un ensemble de bibliothèques logicielles et d'outils pour construire des applications robotiques. Des pilotes et des algorithmes de pointe aux outils de développement puissants, ROS offre les outils open-source dont vous avez besoin pour votre prochain projet en robotique. Depuis son lancement en 2007, beaucoup de choses ont changé dans la communauté de la robotique et de ROS. L'objectif du projet ROS 2 est de s'adapter à ces changements, en tirant parti de ce qui est excellent dans ROS 1 et en améliorant ce qui ne l'est pas.

a) Installation

Installation de ROS 2 Humble sur Ubuntu Linux - Jammy Jellyfish (22.04)

b) Set locale

Assurez-vous d'avoir une localisation qui prend en charge l'UTF-8. Nous testons avec les paramètres suivants. Cependant, cela devrait fonctionner si vous utilisez une localisation différente prenant en charge l'UTF-8.

```
locale  # check for UTF-8

sudo apt update && sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8

locale  # verify settings
```

c) Setup Sources

Vous devrez ajouter le dépôt apt ROS 2 à votre système. Tout d'abord, assurez-vous que le repository Ubuntu Universe est activé.

```
sudo apt install software-properties-common
sudo add-apt-repository universe
```

Maintenant, ajoutez la clé GPG ROS 2 avec apt

```
sudo apt update && sudo apt install curl -y
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o
/usr/share/keyrings/ros-archive-keyring.gpg
```

Ensuite, ajoutez le repository à votre liste de sources

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
```

d) Installer packages ROS 2

Mettez à jour les caches de votre repository apt après avoir configuré les repositories

```
sudo apt update
```

Les packages ROS 2 sont construits pour des systèmes Ubuntu fréquemment mis à jour. Il est toujours recommandé de vous assurer que votre système est à jour avant d'installer de nouveaux packages.

```
sudo apt upgrade
```

Installation de ros-humble:

```
sudo apt install ros-humble-desktop
```

e) Setup l'environment

Configurez votre environnement en sourçant le fichier suivant :

```
# Replace ".bash" with your shell if you're not using bash
# Possible values are: setup.bash, setup.sh, setup.zsh
source /opt/ros/humble/setup.bash
```

f) Background

ROS 2 repose sur la notion de combinaison d'espaces de travail en utilisant l'environnement shell. "workspace" est un terme ROS pour l'emplacement sur votre système où vous développez avec ROS 2. Le workspace principal de ROS 2 est appelé le « underlay ». Les workspaces locaux ultérieurs sont appelés les « overlays ». Lorsque vous développez avec ROS 2, vous aurez généralement plusieurs workspaces actifs simultanément.

La combinaison des workspaces facilite le développement avec différentes versions de ROS 2 ou avec différents ensembles de packages. Cela permet également l'installation de plusieurs distributions ROS 2 (ou "distros", par exemple Dashing et Eloquent) sur le même ordinateur et le passage de l'une à l'autre.

Ceci est accompli en sourçant les fichiers de configuration à chaque fois que vous ouvrez un nouveau shell, ou en ajoutant la commande source à votre script de démarrage shell une fois. **Sans sourcer les fichiers de configuration, vous ne pourrez pas accéder aux commandes ROS 2, ni trouver ou utiliser les packages ROS 2. En d'autres termes, vous ne pourrez pas utiliser ROS 2.**

g) Sourcer les ficher de setup

Vous devrez exécuter cette commande à chaque nouvelle ouverture de shell pour avoir accès aux commandes ROS 2, comme ceci :

```
# Replace ".bash" with your shell if you're not using bash
# Possible values are: setup.bash, setup.sh, setup.zsh
source /opt/ros/humble/setup.bash
```

- Sourcez lors de votre script de démarrage de votre shell.

Si vous ne souhaitez pas devoir sourcer le fichier de configuration à chaque fois que vous ouvrez un nouveau shell (en ignorant la tâche précédente), vous pouvez alors ajouter la commande à votre script de démarrage de shell :

```
echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
```

h) Vérifier les variables d'environnement

Le sourçage des fichiers de configuration de ROS 2 définira plusieurs variables d'environnement nécessaires pour faire fonctionner ROS 2. Si vous rencontrez des problèmes pour trouver ou utiliser vos packages ROS 2, assurez-vous que votre environnement est correctement configuré en utilisant la commande suivante :

```
printenv | grep -i ROS
```

- Vérifier que les variables comme ROS_DISTRO et ROS_VERSION soient affectées.

```
ROS_VERSION=2
ROS_PYTHON_VERSION=3
ROS_DISTRO=humble
```

Sources : ROS 2 Documentation (Humble)

3.2. Open Plotter

Open Plotter est une combinaison de logiciel et de matériel conçue pour servir d'aide à la navigation sur les petits et moyens bateaux. Il s'agit également d'un système complet d'automatisation domestique embarqué. Il est open-source, peu coûteux, à faible consommation et fonctionne sur des ordinateurs ARM tels que le Raspberry Pi ou tout ordinateur exécutant une distribution Linux dérivée de Debian. Son design est modulaire, vous permettant ainsi d'implémenter uniquement ce dont votre bateau a besoin.

a) Prérequis

Avant de procéder à l'installation, assurez-vous d'avoir ce qui suit :

- Un Raspberry Pi ou un ordinateur monocarte compatible (avec alimentation et carte SD).
- Une connexion Internet.
- Un clavier, une souris et un écran (facultatif, car vous pouvez également configurer Open Plotter en mode headless).

b) Installation

Tout d'abord, vous devez installer certaines dépendances. Ouvrez un terminal et entrez :

```
sudo apt update
sudo apt install python3-wxgtk4.0 python3-ujson python3-pyudev vlc matchbox-keyboard
```

Maintenant, vous devez installer l'application « Open Plotter Settings » en tapant ceci dans un terminal, en remplaçant « x.x.x-stable « par votre version :

```
sudo dpkg -i openplotter-settings_x.x.x-stable.deb
```

Ouvrez l'application « Open Plotter Settings » en tapant ceci dans un terminal :

```
openplotter-settings
```

sources : Open Plotter 3 Documentation

3.3. OpenSSH Server

a) Introduction

OpenSSH est une puissante collection d'outils pour le contrôle à distance et le transfert de données entre des ordinateurs connectés en réseau. Vous apprendrez également quelques-uns des paramètres de configuration possibles avec l'application serveur OpenSSH et comment les modifier sur votre système Ubuntu.

OpenSSH est une version librement disponible de la famille d'outils du protocole Secure Shell (SSH) pour le contrôle à distance ou le transfert de fichiers entre des ordinateurs. Les outils traditionnels utilisés pour accomplir ces fonctions, tels que telnet ou rcp, sont non sécurisés et transmettent le mot de passe de l'utilisateur en clair lorsqu'ils sont utilisés. OpenSSH fournit un démon serveur et des outils clients pour faciliter les opérations de contrôle à distance et de transfert de fichiers sécurisées et chiffrées, remplaçant ainsi efficacement les outils hérités.

Le composant serveur OpenSSH, sshd, écoute en permanence les connexions des clients provenant de n'importe lequel des outils clients. Lorsqu'une demande de connexion se produit, sshd établit la connexion correcte en fonction du type d'outil client connecté. Par exemple, si l'ordinateur distant se connecte avec l'application client ssh, le serveur OpenSSH établit une session de contrôle à distance après authentification. Si un utilisateur distant se connecte à un serveur OpenSSH avec scp, le démon serveur OpenSSH initie une copie sécurisée de fichiers entre le serveur et le client après authentification. OpenSSH peut utiliser de nombreuses méthodes d'authentification, y compris le mot de passe simple, la clé publique et les tickets Kerberos.

b) Installation

Pour installer l'application serveur OpenSSH et les fichiers de support associés, exécutez cette commande dans un terminal :

```
sudo apt install openssh-server
```

Si le service est en cours d'exécution, vous devriez voir une sortie indiquant qu'il est actif et en cours d'exécution.

Une fois l'installation terminée, vous pouvez vérifier que le serveur OpenSSH est en cours d'exécution en vérifiant son statut. Exécutez la commande suivante :

```
sudo systemctl status ssh
```

Configuration Pare-Feu (Optionnel)

Si vous utilisez un pare-feu sur votre système Ubuntu, tel que UFW (Uncomplicated Firewall), vous devrez peut-être ouvrir le port SSH pour autoriser les connexions entrantes. Vous pouvez le faire en exécutant la commande suivante :

```
sudo ufw allow ssh
```

Cette commande permettra les connexions SSH entrantes sur le port par défaut (22). Si vous avez personnalisé le port SSH dans le fichier de configuration, remplacez ssh par le numéro de port que vous avez choisi.

c) Se connecter à votre système Ubuntu

Avec OpenSSH installé et configuré, vous pouvez maintenant vous connecter à votre système Ubuntu à distance en utilisant un client SSH. Depuis un autre ordinateur ou périphérique, ouvrez une fenêtre de terminal et exécutez la commande suivante pour vous connecter :

```
ssh username@ubuntu_hostname_or_ip
```

Remplacez "username" par votre nom d'utilisateur Ubuntu, et "ubuntu_hostname_or_ip" par le nom d'hôte ou l'adresse IP de votre système Ubuntu.

Vous serez invité à saisir votre mot de passe pour le compte utilisateur Ubuntu. Une fois authentifié, vous aurez une session shell sécurisée vers votre système Ubuntu.

Pour obtenir le « username » et l'ip/hostname entrez les commandes suivantes :

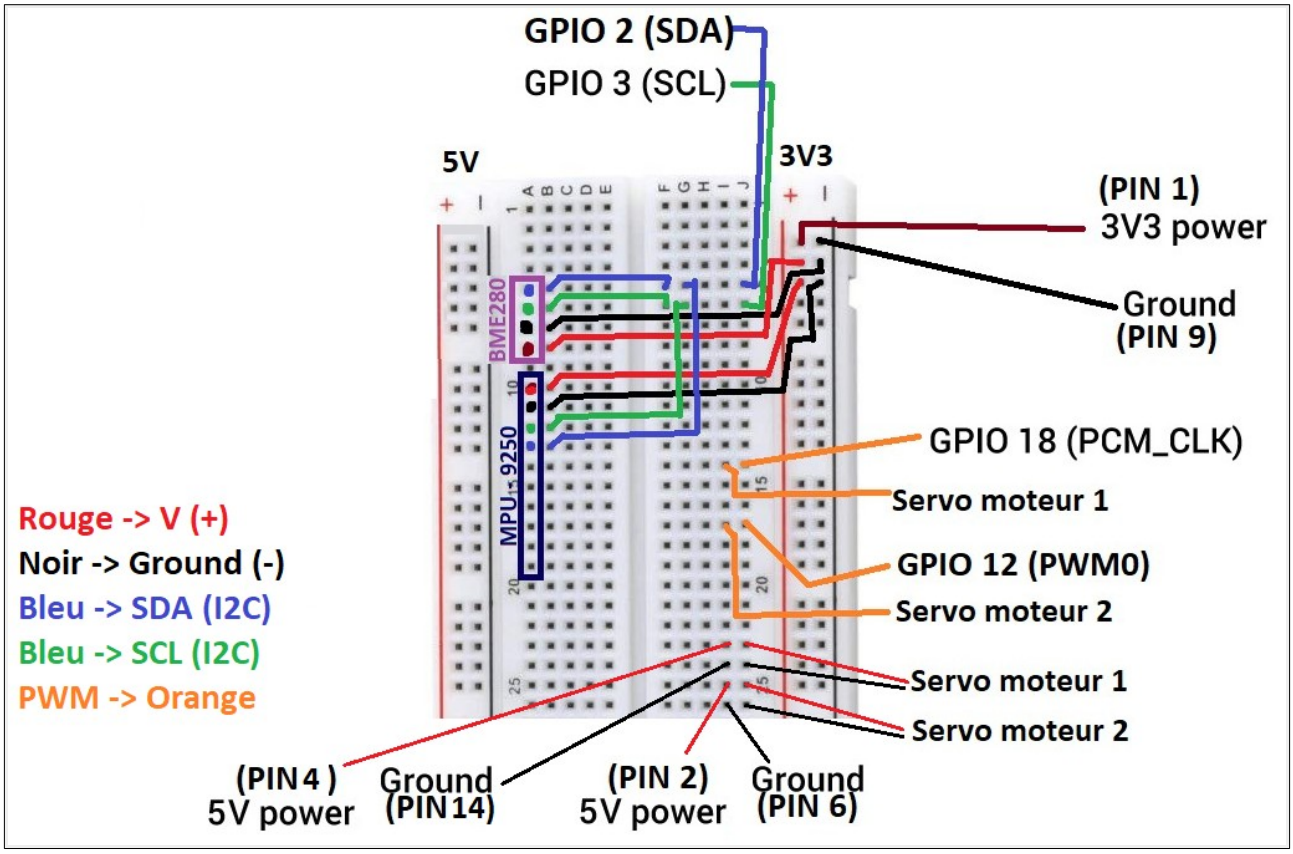
```
whoami
ifconfig
```

4. Instruction câblages

4.1. Schéma global :

Les capteurs de direction et vitesse du vent ne fonctionnent pas avec cette configuration.

4.2. Schéma sur breadboard :



Dans ce schéma on a le capteur environnemental (BME280) et le gyrocompas (MPU9250) connecté au même BUS I2C, on peut également en connecter un sur les pins GPIO 2 et 3 et un sur le pins GPIO 0 et 1 pour être sur deux BUS I2C différent si besoins. Les servomoteurs doivent être connectés à différents ports VCC de la raspberry pour fonctionner.

Bonne ressource pour les pins/GPIO de la Raspberry Pi : https://pinout.xyz/

4.3. Exemple de pont diviseur :

Nous avons besoin de 7V et la batterie a un voltage trop important pour une connexion directe. Il est possible de se brancher à la batterie en faisant un pont diviseur à l'aide de résistances. (ne marche pas avec des servomoteurs connecté en PWM avec la Raspberry)

(Valeurs à titre d'exemple)

On peut utiliser des calculateurs disponibles sur internet pour déterminer les résistances nécessaires pour obtenir le voltage voulu.

exemple de calculateur de pont diviseur:
https://www.digikey.com/en/resources/conversion-calculators/conversion-calculator-voltage-divider
https://ohmslawcalculator.com/voltage-divider-calculator
https://www.allaboutcircuits.com/tools/voltage-divider-calculator/

5. Tests Hardware

5.1. Branchements I2C :

Suivant votre choix dans les branchements vous utiliserez 1 ou 2 bus avec 1 ou plusieurs capteurs par bus.

a) pin gpio 2 et 3 (bus 1)

Afficher l'adresse dans laquelle le capteur écrit les informations :

i2cdetect -y 1

```
mgmatdev@mgmatdev-desktop:~$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

dans cet exemple "68" donc il faudra lire l'adresse 0x68 et adapter le code en fonction.

b) pin gpio 27 et 28 (bus 0)

Afficher l'adresse dans laquelle le capteur écrit les informations :

i2cdetect -y 0

```
mgmatdev@mgmatdev-desktop:~$ i2cdetect -y 0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
```

dans cet exemple "76" donc il faudra lire l'adresse 0x76 et adapter le code en fonction.

Si ça détecte pas dans boot/config.txt ou boot/firmware/config.txt changer la ligne :

dtparam=i2c1=on

5.2. Branchement USB :

Afficher tous les périphériques connectés en USB :

sudo lsusb

```
mgmatdev@mgmatdev-desktop:~$ lsusb
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 006: ID 413c:2113 Dell Computer Corp. KB216 Wired Keyboard
Bus 001 Device 005: ID 1546:01a7 U-Blox AG [u-blox 7]
Bus 001 Device 004: ID 413c:301a Dell Computer Corp. Dell MS116 Optical Mouse
Bus 001 Device 007: ID 04e8:6863 Samsung Electronics Co., Ltd Galaxy series, misc. (tethering mode)
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

On peut voir le champ ID 1546:01a7 pour le gps U-Blox, “1546” est l'idVendor que l’on va utiliser pour trouver le port usb le correspondant avec:

sudo dmesg | grep [idVendor]

```
mgmatdev@mgmatdev-desktop:~$ sudo dmesg | grep 1546
[    3.220913] usb 1-1.3: New USB device found, idVendor=1546, idProduct=01a7, bcdDevice= 1.00
```

On peut voir le port usb correspondant avec :

sudo dmesg | grep [usb 1-1.x] | tty

```
mgmatdev@mgmatdev-desktop:~$ sudo dmesg | grep 1-1.3 | grep tty
[    8.150803] cdc_acm 1-1.3:1.0: ttyACM1: USB ACM device
```

On obtient "ttyACM1" qui est le fichier dans lequel le capteur écrit les données et il faut adapter le code en fonction.

On peut les afficher les données avec la commande suivante :

cat /dev/tty[port serial]

soit dans le cas de l'image ci-dessus :

cat /dev/ttyACM1

6. Configuration Open Plotter

OpenPlotter est un logiciel open-source conçu pour transformer un Raspberry Pi (un petit ordinateur monocarte) en une centrale de navigation maritime ou en un système de contrôle pour un voilier ou un bateau. Il offre une plateforme flexible et personnalisable pour la navigation, en utilisant des composants matériels et logiciels accessibles et peu coûteux.

OpenPlotter peut recevoir des données provenant de diverses sources à bord du bateau, telles que les capteurs de vent, les capteurs de température de l'eau, les GPS, les instruments de navigation, etc. Ces données sont généralement transmises via des interfaces standard telles que NMEA 0183 (un protocole de communication maritime largement utilisé) ou NMEA 2000 (une norme de communication plus récente et plus avancée pour les bateaux).

Le projet du voilier autonome nécessite plusieurs capteurs tel que le capteur environnemental , gyrocompas , GPS etc
Il est possible de récolter ces données et de les transmettre à signal K afin de les rendre visible sur l'interface Signal K accessible à l'adresse *http://localhost:3000* .

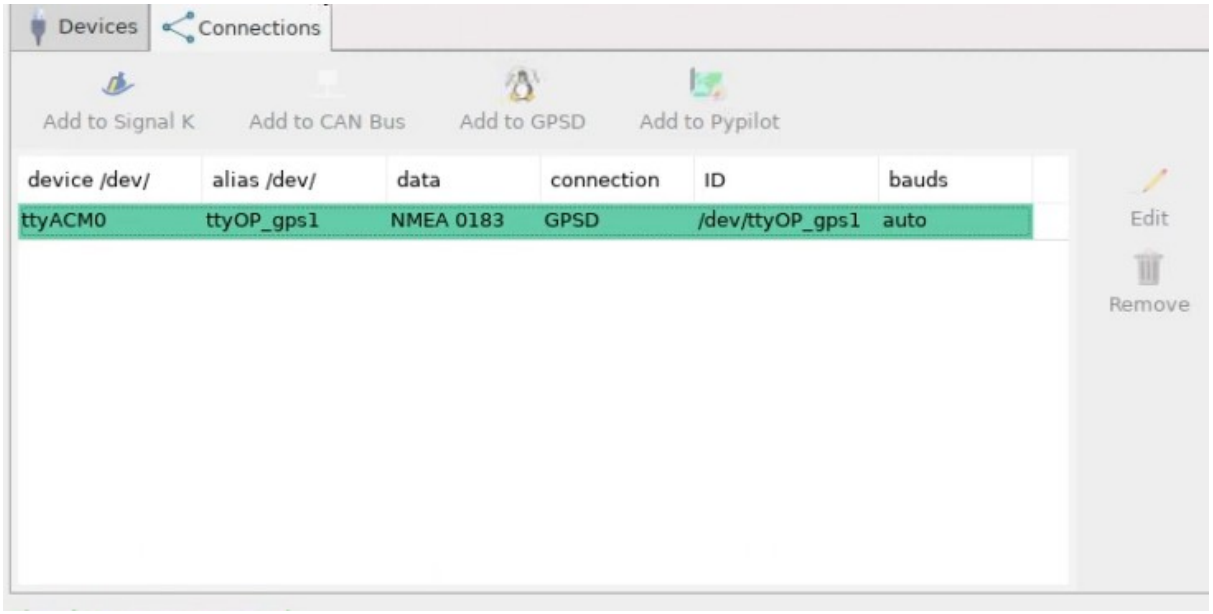
capteur environnemental (BME280) :
Pour avoir accès aux données du capteur environnemental, nous devons aller dans l'onglet I2C Sensors d'openplotter. A partir de là, il est nécessaire de sélectionner l'adresse correspondante au composant, dans notre cas l'adresse 76 (0x76).

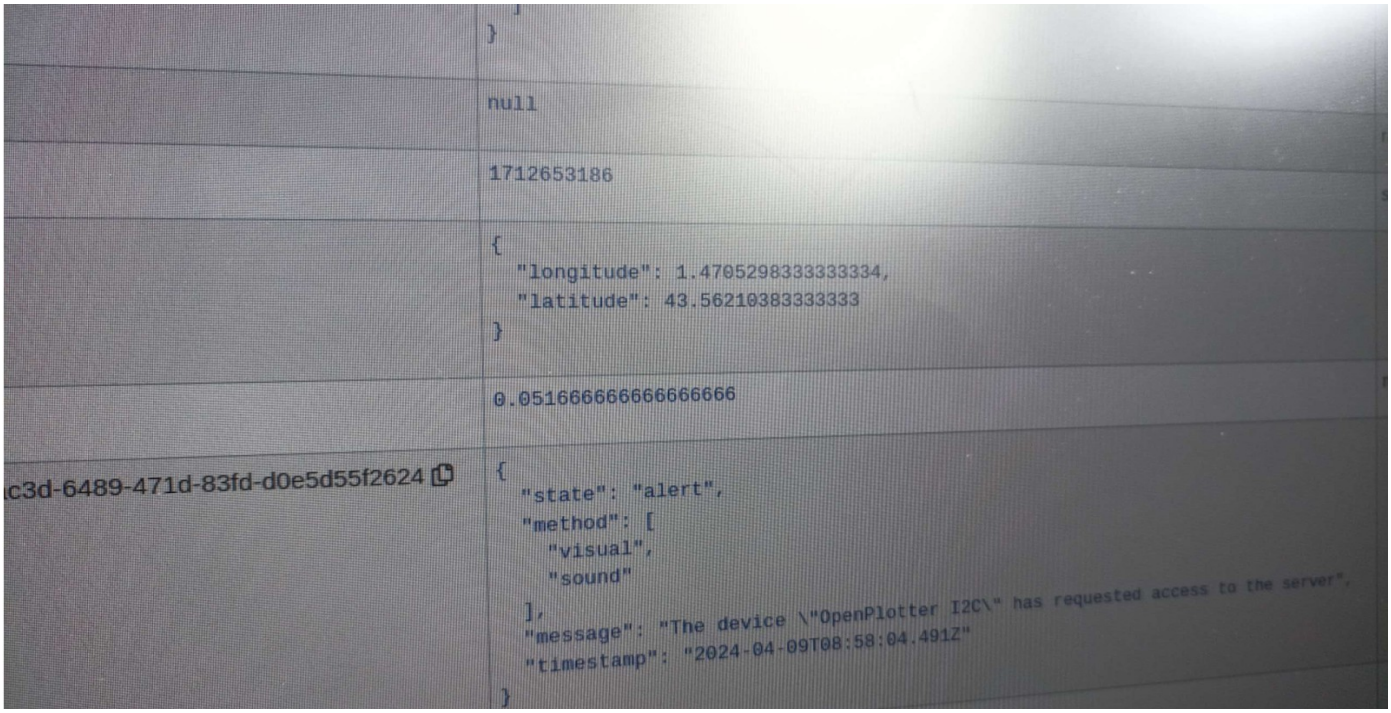
A partir de là, Signal K accède aux données et les affiche dans l'onglet data Browser , voir ci-dessous.

GPS:

Les données GPS sont accessibles via l'onglet « serial », qui traite les données des composants branché en USB .

Le GPS transmet des données via NMEA 0183 , il est donc indispensable de bien choisir le bon type de données . Dans notre projet , nous avons dédié un port spécifique au GPS , il prendra le nom de ttyACM0 . Un fichier ttyACM est un périphérique virtuel de terminal série sur Linux, généralement utilisé pour communiquer avec des périphériques matériels connectés via le port USB . Il est important de sélectionner le bouton « Remember Port » pour associer les paramètres du GPS au GPS lui même , si on coche la case remember device , le GPS aura un nom différent et cela peut engendrer des erreurs . Pour que Signal K reçoive les données du GPS , il doit envoyer les données à GPSD qui s'occupera lui même de l'envoyer à Signal K .





Signal K reçoit bien les données du GPS . Après quelques minutes de calibration , le GPS passe par plusieurs satellites et nous fournit la longitude et latitude suivante qui nous localise au mètre près .

Résultat avant calibration.

Gyrocompas (MPU9250):

Le MPU9250 est un circuit intégré qui combine plusieurs fonctionnalités dans un seul module compact:

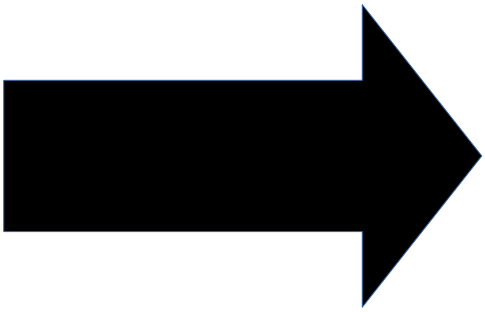
Accéléromètre , Gyroscope , Magnétomètre

Contrairement au capteur environnemental, le paramétrage du gyrocompas se passe sur pypilot . Il n'est pas nécessaire de choisir quel type de données doit être envoyé car pypilot gère ça automatiquement . Cependant , le calibrage se fait manuellement dans l'onglet calibration .

Après paramétrage , les données sont accessible sur l'onglet < Data Browser > :

Cette ligne nous donne notamment le Nord magnétique en radiant , à noter que la puce a un sens .

Les 2 trous doivent être situés vers l'avant du bateau pour la calibration .



Avant du bateau

---

7. Installation des modules pythons

Avant de continuer nous allons exécuter ces commandes qui vont nous permettre d'installer des modules et des librairies python.

```
sudo apt install python-setuptools python3-setuptools
sudo apt-get install build-essential
```

Pour installer un module python on utilise la commande pip3 :

```
pip3 install <nom-module>
```

Dans le code que nous utilisons il y a le module « smbus » permettant le traitement des données du bus I2C pour le gyrocompas et le capteur environemental. Pour l'installer avec pip3 on fait :

```
pip3 install smbus
```

Nous utilisons également la librairie « pigpio » qui nous permet de contrôler les servomoteurs.

Voici les commandes pour l'installer :

```
wget https://github.com/joan2937/pigpio/archive/master.zip
unzip master.zip
cd pigpio-master
make
sudo make install
sudo pigpiod
```

La dernière commande n'affiche pas d'erreur alors installation de « pigpio » est complète.

Vous pouvez retrouver la procédure d'installation et la documentation de « pigpio » sur : https://abyz.me.uk/rpi/pigpio/download.html

Nous utilisons une librairie pour le gyrocompas (imusensor) elle nécessite « i2c-tools », « numpy » et « easydict ». Pour les installer on fait :

```
sudo apt-get install i2c-tools
pip3 install numpy
pip3 install easydict
```

Ensuite pour installer la librairie «imusensor» on fait :

```
wget https://codeload.github.com/niru-5/imusensor/zip/refs/heads/master
unzip master
cd imusensor-master
python3 setup.py install
```

librairie imusensor : https://github.com/niru-5/imusensor

8. Créer un package ros

8.1. Pré-requis:

Il faut initialement installer ros2 et créer les différents dossiers nécessaires.

8.2. Création du package:

Pour chaque capteur, il faut créer un package. Pour se faire, dans le terminal, il faut ouvrir le dossier src dans le dossier ros2. Ensuite, créer un package avec la commande suivante:

```
ros2 pkg create --build-type ament_python --license Apache-2.0 nom_package
```

Le terminal est sensé envoyer un message pour signaler la création du package.

8.3. Création du publisher et du subscriber:

Avec le terminal, aller dans ros2_ws/src/nom_package/nom_package et y mettre le code dans un fichier pour le publisher.

Aller dans ros2_ws/src/nom_package/nom_package et écrire le fichier code du subscriber correspondant.

8.4. Ajouter des dépendances:

Avec le terminal, aller dans le dossier ros2_ws/src/nom_package et ouvrir le fichier package.xml. Ceci sera affiché:

```
<description>Examples of minimal publisher/subscriber using rclpy</description>
<maintainer email="you@email.com">Your Name</maintainer>
<license>Apache License 2.0</license>
```

Cela permet d'ajouter une description, la license et l'email et le nom du créateur du package.

Ensuite, ajouter les imports du fichier python en dessous des lignes au dessus.

Exemple :

```
import rclpy
from rclpy.node import Node
```

Ligne à taper:

```
<exec_depend>rclpy</exec_depend>
<exec_depend>std_msgs</exec_depend>
```

Ensuite, ajouter un point d'entré pour le publisher en ouvrant le fichier setup.py, comme pour le fichier précédent, mais avant cela donnez la description, la licence, l'email et le nom du créateur du package.

```
maintainer='YourName',
maintainer_email='you@email.com',
description='Examples of minimal publisher/subscriber using rclpy',
license='Apache License 2.0',
```

Ensuite, ajoutez le point d'entrée en dessus de ses lignes pour le publisher.

```
entry_points={
        'console_scripts': [
                'talker = nom_package.nom_publisher:main',
        ],
},
```

Et faites de même pour le subscriber.

```
entry_points={
        'console_scripts': [
                'talker = nom_package.nom_subscriber:main',
        ],
},
```

Vérifier dans le fichier setup.cfg que les répertoires soient bien écrits.

```
[develop]
script_dir=$base/lib/nom_package
[install]
install_scripts=$base/lib/nom_package
```

8.5. Exécuter le code:

Taper les lignes suivantes dans le terminal en étant dans ros2_ws/src pour construire le package :

```
rosdep install -i --from-path src --rosdistro humble -y
colcon build --packages-select nom_package
```

Ouvrir un nouveau terminal dans le dossier src du dossier ros2 et Exécutez le talker :

```
source install/setup.bash
ros2 run py_pubsub talker
```

Ouvrir un nouveau terminal au même endroit et exécuter le listener:

```
ros2 run py_pubsub listener
```

9. Installer le code ROS fournit

Créer un package ROS, dans ros2_ws/src/nom_package/nom_package si vous voulez installer un capteur créez un publisher ou un subscriber si c'est un servomoteurs ou le nœuds principal.

Ensuite entrez le contenu du code fournis dans votre publisher ou subscriber correspondant.

Avant d'exécuter chaque nœud , il faut taper la commande sudo pigpiod qui va lancer le service pigpio .

Une fois le code installé il est nécessaire de démarrer un terminal par nœud ros et de taper la commande source/install.bash dans chacun d'entre eux . Ensuite il faut lancer un nœud par terminal (l'ordre de lancement des scripts n'a pas d'importance , un talker peut parler sans que personne ne s'abonne à son topic et un listener peut lire un topic vide) .
Il est nécessaire de compiler tout les packages avant de les utiliser (commande colcon cité en haut) , un code compil.sh est fourni qui va compiler chaque package du projet . Il est nécessaire de nommer les packages comme nous les avons nommés pour que le script marche.
Etape 1 : Dans un terminal quelconque , lancez le script ./compil.sh
Etape 2 : Lancez toutes les commandes ros2 run nom_packages talker/listener
Liste talker = Gyrocompas , gps , enviro_280
Liste listener = Servos , interface
L'interface est le nœud principal qui gère tout le système .

10. Code de test pour les capteurs

10.1. Test ROS GPS

a) Publisher

Executer avec ROS : ros2 run nom_package talker

```
import rclpy  # Importation du module rclpy, qui permet d'utiliser ROS 2 en Python

from rclpy.node import Node  # Importation de la classe Node du module rclpy.node
from std_msgs.msg import String  # Importation du type de message String du package std_msgs
from serial import Serial  # Importation de la classe Serial du module serial


class GPSNode(Node):  # Définition d'une classe GPSNode qui hérite de la classe Node
def __init__(self):  # Définition de la méthode d'initialisation de la classe
```

```
    super().__init__('gps_node')  # Appel du constructeur de la classe mère avec le nom du nœud 'gps_node'
    self.publisher = self.create_publisher(String, 'GPS', 10)  # Création d'un éditeur (publisher) pour publier des messages de type String sur le topic 'read' avec une file d'attente de taille 10
    self.subscription = self.create_subscription(String, 'write', self.callback, 10)  # Création d'un abonnement (subscription) pour recevoir des messages de type String sur le topic 'write' avec une file d'attente de taille 10, et en spécifiant la méthode callback à appeler lorsqu'un message est reçu
    self.serial_port = Serial('/dev/ttyACM0', 9600, timeout=1)  # Initialisation d'un objet Serial pour communiquer avec le port série '/dev/ttyACM0' à une vitesse de transmission de 9600 bauds avec un délai d'attente de 1 seconde

def callback(self, msg):  # Définition de la méthode callback qui sera appelée lorsqu'un message est reçu sur le topic 'write'
    self.get_logger().info('Writing to serial port: %s' % msg.data)  # Enregistrement d'un message de journalisation indiquant que des données sont écrites sur le port série avec les données du message
    self.serial_port.write(msg.data.encode())  # Écriture des données du message encodées sur le port série

def publish_gps_data(self):  # Définition d'une méthode pour publier les données GPS
    while self.serial_port.is_open:  # Boucle tant que le port série est ouvert
        if self.serial_port.in_waiting:  # Vérification si des données sont en attente de lecture sur le port série
            data = self.serial_port.read(self.serial_port.in_waiting).decode()  # Lecture des données disponibles sur le port série et décodage en chaîne de caractères
            self.get_logger().info('GPS Data: %s' % data)  # Enregistrement d'un message de journalisation indiquant les données GPS lues
            msg = String()  # Création d'un objet de message de type String
            msg.data = data  # Assignation des données lues au champ 'data' du message
            self.publisher.publish(msg)  # Publication du message sur le topic 'GPS'

def main(args=None):  # Définition de la fonction principale
rclpy.init(args=args)  # Initialisation de rclpy avec les arguments passés à la fonction
```

```
gps_node = GPSNode()  # Création d'une instance de la classe GPSNode
try:  # Bloc d'essai pour exécuter le code
    gps_node.publish_gps_data()  # Appel de la méthode pour publier les données GPS
finally:  # Bloc de fin, exécuté même en cas d'erreur
    gps_node.destroy_node()  # Destruction du nœud ROS
    rclpy.shutdown()  # Arrêt de l'exécution de ROS 2

if __name__ == '__main__':  # Condition pour exécuter le code si le script est exécuté directement
main()  # Appel de la fonction principale
```

Ce code crée un nœud appelé gps_node qui lit les données GPS à partir d'un port série et les publie sur le topic « GPS ».
Il possède également un abonnement à un autre topic où les données peuvent être écrites pour être envoyées au port série.

b) Subscriber

Executer avec ROS : ros2 run nom_package listener

```
#!/usr/bin/env python

import rospy  # Importation du module rospy, qui permet d'utiliser ROS en Python
from sensor_msgs.msg import NavSatFix  # Importation du type de message NavSatFix du package sensor_msgs

def gps_callback(data):  # Définition d'une fonction callback appelée lorsqu'un message GPS est reçu
rospy.loginfo("Received GPS data - Latitude: %f, Longitude: %f, Altitude: %f",  # Enregistrement d'un message de journalisation avec les données GPS reçues
              data.latitude, data.longitude, data.altitude)

def gps_subscriber():  # Définition d'une fonction principale qui initialise le nœud ROS et crée un abonnement au topic GPS
```

```
rospy.init_node('gps_subscriber', anonymous=True)  # Initialisation du nœud ROS avec le nom 'gps_subscriber' et anonyme=True
rospy.Subscriber('GPS', NavSatFix, gps_callback)  # Création d'un abonnement au topic 'GPS' avec le type de message NavSatFix et le callback gps_callback
rospy.spin()  # Boucle de traitement des événements ROS

if __name__ == '__main__':  # Condition pour exécuter le code si le script est exécuté directement
try:
    gps_subscriber()  # Appel de la fonction principale
except rospy.ROSInterruptException:  # Gestion de l'exception ROSInterruptException
    pass  # Ignorer l'exception et continuer l'exécution du programme
```

Ce code crée un nœud ROS qui s'abonne au topic GPS pour recevoir des messages de type NavSatFix contenant des données GPS.
Lorsqu'un message est reçu, la fonction de rappel gps_callback est appelée pour enregistrer les données GPS dans les journaux.
Le nœud ROS tourne en boucle tant que le nœud maître ROS est en cours d'exécution. Si une exception ROSInterruptException est levée, elle est ignorée et le programme continue son exécution.
Ce code crée un nœud ROS qui s'abonne au topic GPS pour recevoir des messages de type NavSatFix contenant des données GPS.
Lorsqu'un message est reçu, la fonction de rappel gps_callback est appelée pour enregistrer les données GPS dans les journaux.
Le nœud ROS tourne en boucle tant que le nœud maître ROS est en cours d'exécution. Si une exception ROSInterruptException est levée, elle est ignorée et le programme continue son exécution.

10.2. Test ROS Capteur environemental

a) Publisher

Exécuter avec ROS : ros2 run nom_package talker

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
import time
import smbus
from ctypes import c_short
from ctypes import c_byte
from ctypes import c_ubyte

def getShort(data, index):
# return two bytes from data as a signed 16-bit value
 return c_short((data[index+1] << 8) + data[index]).value

def getUShort(data, index):
# return two bytes from data as an unsigned 16-bit value
 return (data[index+1] << 8) + data[index]

def getChar(data,index):
 # return one byte from data as a signed char
 result = data[index]
 if result > 127:
  result -= 256
 return result
```

```
def getUChar(data,index):
  # return one byte from data as an unsigned char
  result =  data[index] & 0xFF
  return result

class BMEPublisher(Node):
    def __init__(self):
        super().__init__('bme_publisher')

        self.publisher = self.create_publisher(String, 'environment_data', 10)

        # Initialise le bus I2C
        self.bus = smbus.SMBus(1)  # Changez le numéro de bus selon votre configuration



    def read_bme280_data(self):
        # Initialisez l'adresse I2C du capteur BME280 (par défaut : 0x76)
        address = 0x76

        # Lire les données de température, d'humidité et de pression
        block = self.bus.read_i2c_block_data(address, 0xF7, 8)
        print(block)
        pres_raw = (block[0] << 16 | block[1] << 8 | block[2]) >> 4
        temp_raw = (block[3] << 16 | block[4] << 8 | block[5]) >> 4
        hum_raw = block[6] << 8 | block[7]

        # Read blocks of calibration data from EEPROM
        # See Page 22 data sheet
        cal1 = self.bus.read_i2c_block_data(address, 0x88, 24)
        cal2 = self.bus.read_i2c_block_data(address, 0xA1, 1)
        cal3 = self.bus.read_i2c_block_data(address, 0xE1, 7)

        # Convert byte data to word values
        dig_T1 = getUShort(cal1, 0)
        dig_T2 = getShort(cal1, 2)
        dig_T3 = getShort(cal1, 4)

        dig_P1 = getUShort(cal1, 6)
        dig_P2 = getShort(cal1, 8)
        dig_P3 = getShort(cal1, 10)
```

```
dig_P4 = getShort(cal1, 12)
dig_P5 = getShort(cal1, 14)
dig_P6 = getShort(cal1, 16)
dig_P7 = getShort(cal1, 18)
dig_P8 = getShort(cal1, 20)
dig_P9 = getShort(cal1, 22)

dig_H1 = getUChar(cal2, 0)
dig_H2 = getShort(cal3, 0)
dig_H3 = getUChar(cal3, 2)

dig_H4 = getChar(cal3, 3)
dig_H4 = (dig_H4 << 24) >> 20
dig_H4 = dig_H4 | (getChar(cal3, 4) & 0x0F)

dig_H5 = getChar(cal3, 5)
dig_H5 = (dig_H5 << 24) >> 20
dig_H5 = dig_H5 | (getUChar(cal3, 4) >> 4 & 0x0F)

dig_H6 = getChar(cal3, 6)

#Calcul temperature
var1 = ((((temp_raw>>3)-(dig_T1<<1)))*(dig_T2)) >> 11
var2 = (((((temp_raw>>4) - (dig_T1)) * ((temp_raw>>4) - (dig_T1))) >> 12) * (dig_T3)) >> 14
t_fine = var1+var2
temperature = float(((t_fine * 5) + 128) >> 8);

# Calcul pression et ajuste la temperature
var1 = t_fine / 2.0 - 64000.0
var2 = var1 * var1 * dig_P6 / 32768.0
var2 = var2 + var1 * dig_P5 * 2.0
var2 = var2 / 4.0 + dig_P4 * 65536.0
var1 = (dig_P3 * var1 * var1 / 524288.0 + dig_P2 * var1) / 524288.0
var1 = (1.0 + var1 / 32768.0) * dig_P1
if var1 == 0:
  pressure=0
else:
  pressure = 1048576.0 - pres_raw
  pressure = ((pressure - var2 / 4096.0) * 6250.0) / var1
  var1 = dig_P9 * pressure * pressure / 2147483648.0
  var2 = pressure * dig_P8 / 32768.0
```

```
        pressure = pressure + (var1 + var2 + dig_P7) / 16.0

        # Calcul humidite
        humidity = t_fine - 76800.0
        humidity = (hum_raw - (dig_H4 * 64.0 + dig_H5 / 16384.0 * humidity)) * (dig_H2 / 65536.0 *
(1.0 + dig_H6 / 67108864.0 * humidity * (1.0 + dig_H3 / 67108864.0 * humidity)))
        humidity = humidity * (1.0 - dig_H1 * humidity / 524288.0)
        if humidity > 100:
          humidity = 100
        elif humidity < 0:
          humidity = 0



        print(temperature/100,"C ",humidity,"% ",pressure/100,"hpa")
        return temperature/100.0, humidity, pressure/100

    def publish_environment_data(self):
        while True:
            # Lire les données du capteur BME280
            #pressure,temperature,humidity  = self.read_bme280_data()
            temperature,humidity,pressure = self.read_bme280_data()



            # Crée un message ROS pour les données de l'environnement
            msg = String()
            msg.data = f'Temperature: {temperature} °C, Humidity: {humidity} %, Pressure: {pressure}
hPa'
            # Publie les données sur le topic 'environment_data'
            self.publisher.publish(msg)

            # Attend un certain temps avant la prochaine lecture des données
            time.sleep(2)

def main(args=None):
    rclpy.init(args=args)
    bme_publisher = BMEPublisher()
    try:
        bme_publisher.publish_environment_data()
    finally:
        bme_publisher.destroy_node()
        rclpy.shutdown()
```

```
if __name__ == '__main__':
    main()
```

b) Subscriber

Exécuter avec ROS : ros2 run nom_package listener

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String


class BMESubscriber(Node):
    def __init__(self):
        super().__init__('bme_subscriber')
        self.subscription = self.create_subscription(
            String,
            'environment_data',
            self.callback,
            10)
        self.subscription  # empêche la suppression du rappel lorsqu'il est inutilisé

    def callback(self, msg):
        self.get_logger().info('Received environment data: %s' % msg.data)
```

```
def main(args=None):
    rclpy.init(args=args)
    bme_subscriber = BMESubscriber()
    rclpy.spin(bme_subscriber)
    bme_subscriber.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

10.3. Test Gyrocompas

Exécuter avec python (python3 nom_fichier.py)

```
import os
import sys
import time
import smbus

from imusensor.MPU9250 import MPU9250
from imusensor.filters import madgwick

sensorfusion = madgwick.Madgwick(0.5)

address = 0x68
bus = smbus.SMBus(1)
imu = MPU9250.MPU9250(bus, address)
imu.begin()

# imu.caliberateGyro()
```

```
# imu.caliberateAccelerometer()
# or load your own caliberation file
#imu.loadCalibDataFromFile("/home/pi/calib_real4.json")

currTime = time.time()
print_count = 0
while True:
        imu.readSensor()
        for i in range(10):
                newTime = time.time()
                dt = newTime - currTime
                currTime = newTime

                sensorfusion.updateRollPitchYaw(imu.AccelVals[0], imu.AccelVals[1],
imu.AccelVals[2], imu.GyroVals[0], \
                                                imu.GyroVals[1],
imu.GyroVals[2], imu.MagVals[0], imu.MagVals[1], imu.MagVals[2], dt)

        if print_count == 2:
                print ("mad roll: {0} ; mad pitch : {1} ; mad yaw : {2}".format(sensorfusion.roll,
sensorfusion.pitch, sensorfusion.yaw))
                print_count = 0

        print_count = print_count + 1
        time.sleep(0.01)
```

10.4. Test Servomoteur (pin 12)

Exécuter avec python (python3 nom_fichier.py)

```
#!/usr/bin/env python

# read_PWM.py
# 2015-12-08
# Public Domain

import time
import pigpio # http://abyz.co.uk/rpi/pigpio/python.html
import read_PWM

PWM_GPIO = 12     #Pin GPIO

pi = pigpio.pi()    #Cree un objet pigpio



while True:
  time.sleep(2)
  pi.set_servo_pulsewidth(PWM_GPIO, 500)     # 0 degree
  time.sleep(2)
  pi.set_servo_pulsewidth(PWM_GPIO, 2500)    #180 degree dans le sens horaire

pi.stop()
```

A la fin du projet , l’architecture du projet ressemble à ça :