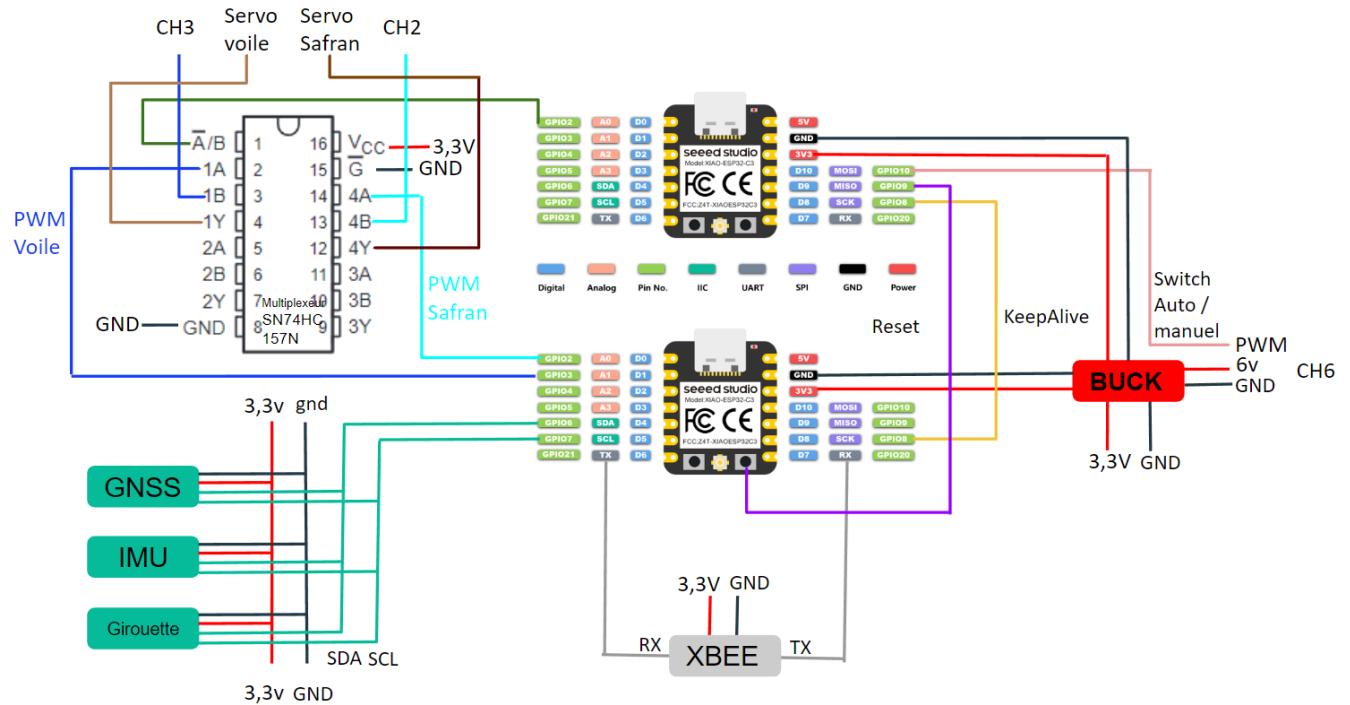




Schéma général :



- Microcontrôleurs : 2 ESP32C3 seeed studio :
<https://www.seeedstudio.com/Seeed-XIAO-ESP32C3-p-5431.html>
- Convertisseur BUCK :
<https://www.digikey.fr/fr/products/detail/dfrobot/DFR0570/9559260>
- Module XBEE :
<https://www.gotronic.fr/art-module-xbee-serie-2c-xb24cawit-001-31605.htm>

- GNSS :
- IMU :
- Capteur Girouette :
- Multiplexeur :
<https://www.mouser.fr/ProductDetail/Texas-Instruments/SN74HC157N?qs=UGVLDq%2F29uh%252BIECfoxfSbw%3D%3D>

I. Principe de fonctionnement

Nous avons deux microcontrôleurs, un pour collecter les données des différents capteurs et faire fonctionner l'algorithme de navigation, "Décision", le second pour assurer la sécurité du système et le basculement du mode automatique au mode manuel , "Safety".

Tout notre système fonctionne avec une alimentation en 3,3V.

Les capteurs empruntent tous le même bus I2C. (voir section Librairies)

Le module XBEE permet de réaliser la commande

CH2, CH3, et CH6 correspondent aux signaux que l'on récupère en sortie du récepteur RF de la télécommande.

Servo voile et Servo Safran correspondent aux deux sorties pilotables du bateau. (voir section Librairies -> CommandeServo)

Le multiplexeur sert à basculer de la commande automatique générée par l'algorithme de navigation à la commande manuelle envoyée par la télécommande. En activant le switch lié à "CHANNEL 6" sur la télécommande, on bascule du mode automatique au mode manuel. Cette bascule est réalisée via le microcontrôleur "Safety"

Un signal "KeepAlive" est envoyé périodiquement depuis le microcontrôleur "Décision" vers le microcontrôleur "Safety", pour garantir que le fonctionnement du microcontrôleur "Décision" n'est pas altérée (plantage, boucle infinie...). Si le signal n'est plus reçu, le microcontrôleur "Safety", bascule la commande du bateau en mode manuel et lance un reset du microcontrôleur "Décision". Une fois le reset effectué, il change à nouveau la commande du bateau en mode automatique. (Voir section Librairies -> microcontrôleur Safety)

II. Microcontrôleurs et IDE

L'intégralité des codes ont été réalisés sur et pour l' ESP32C3 seeed studio. Ils sont donc compatibles avec la plupart des microcontrôleurs de la gamme ESP32.
L'IDE utilisé est ARDUINO.

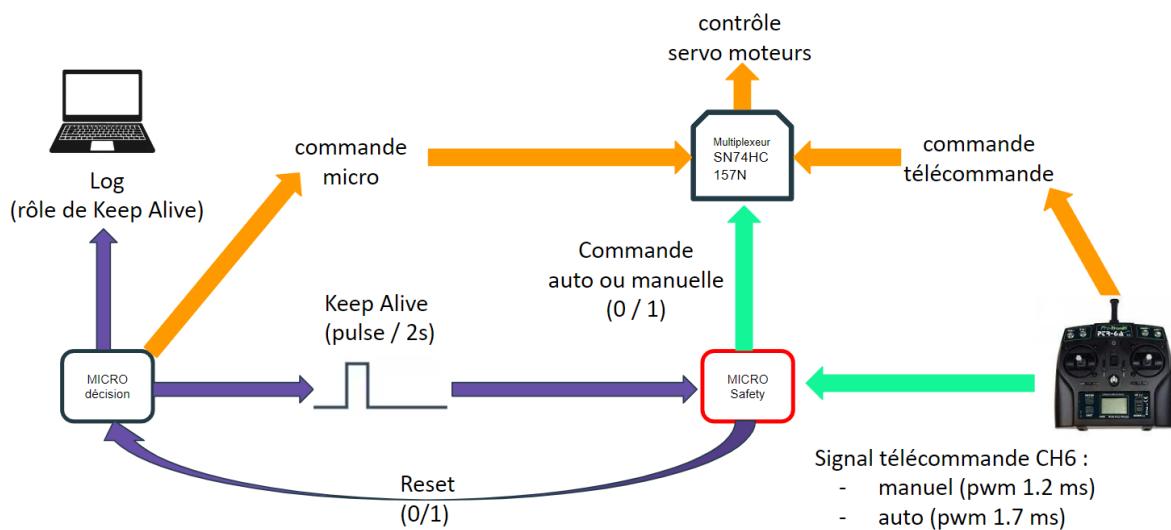
SETING ARDUINO

III. Les librairies (projet_voilier_main -> SRC)

1. Microcontrôleur “Safety”

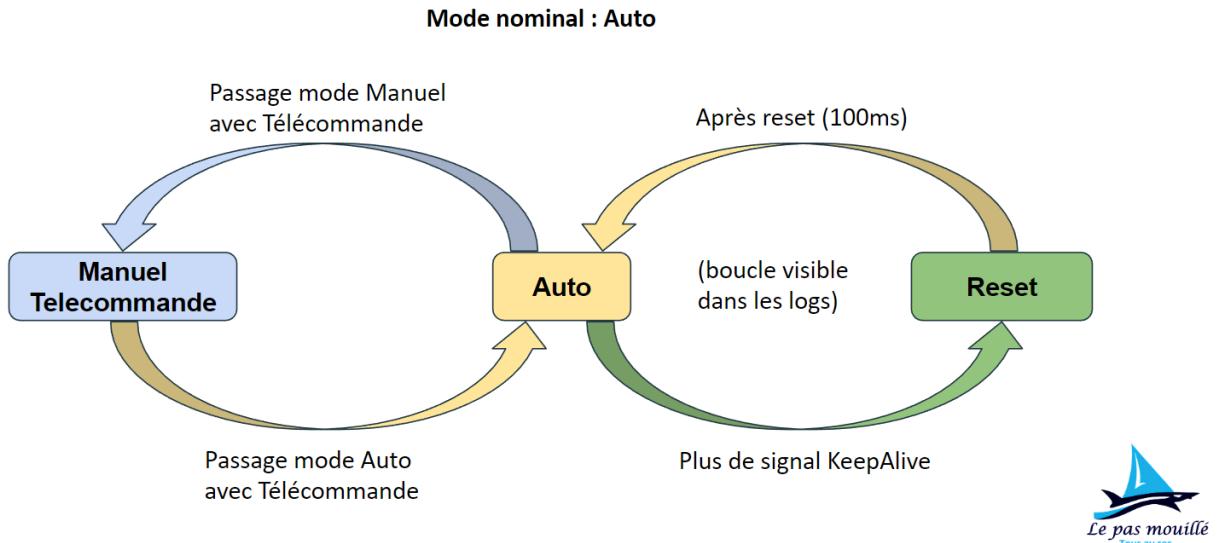
Ce microcontrôleur permet de gérer la sécurité de fonctionnement du microcontrôleur “Décision” et le basculement du mode auto au mode manuel.

Les logs que l'on reçoit depuis le microcontrôleur “Décision” via la communication XBEE permettent également de s'assurer le bon fonctionnement du système.



Le signal KeepAlive est une pulsion envoyée périodiquement depuis le microcontrôleur “Décision” vers le microcontrôleur “Safety”, pour garantir que le fonctionnement du microcontrôleur “Décision” n'est pas altérée (plantage, boucle infinie...). Si le signal n'est plus reçu, le microcontrôleur “Safety”, bascule la commande du bateau en mode manuel et lance un reset du microcontrôleur “Décision”. Une fois le reset effectué, il change à nouveau la commande du bateau en mode automatique.

Le basculement du mode auto au mode manuel suit la machine à état suivante :



Le mode nominal de la machine à état est le mode Auto, il peut également être assigné au mode manuel.

Le mode Reset est en réalité un mode manuel temporaire. Pendant le reset, le mode manuel est activé jusqu'à ce qu'il soit terminé.

Le reset coupe également la communication XBEE et donc l'envoi de logs.

La sécurité est donc également assurée par la visualisation des logs. S'ils sont constamment en reset, nous passons au mode manuel.

Si au bout de 4 reset il n'y a toujours pas de signal KeepAlive, le système passe automatiquement en mode Manuel.

Le code se trouve dans le répertoire Safety, le fichier est un Safety.ino

2. Algorithme de navigation

2.1 Introduction

L'objectif de cette librairie est de calculer les sorties nécessaires au pilotage d'un bateau autonome en fonction des entrées de différents capteurs. Le bateau est équipé de trois capteurs :

- Un GPS : renvoie la position GPS du bateau
- Une IMU : renvoie l'orientation du bateau par rapport au Nord
- Une girouette : renvoie la direction du vent par rapport au bateau

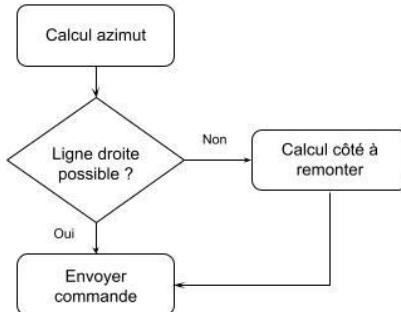
Les deux actionneurs du bateau sont deux moteurs pilotés par PWM, un pour le safran et un autre pour le réglage des deux voiles.

Tous les angles dans la librairie sont exprimés en degrés, dans la plage [-180;180] avec 0° étant l'étrave du bateau.

2.2 Concept de base de navigation

La première chose à déterminer est la direction vers laquelle se situe la bouée, l'azimut. L'azimut est un angle entre le Nord, le bateau et la destination. Les angles sur le bateau sont

exprimés sur la plage [-180;180] avec le 0° étant l'avant du bateau. Une fois l'azimut déterminé, il faut vérifier que la trajectoire en ligne droite est faisable en fonction du vent (le bateau ne peut pas remonter à plus de +/-35° de vent). Si la ligne droite est faisable alors le cap donné est l'azimut, sinon il faut choisir de quel côté remonter le vent.



2.3 Utilisation de la librairie

La librairie met à disposition un objet Navigator ainsi que trois fonctions publiques. L'objet Navigator prend en paramètre une destination initiale.

La première fonction à utiliser est `navigate(Point boat, float wind_angle, float course)`. Cette fonction prend en paramètre la position du bateau, la direction du vent par rapport au bateau ainsi que le cap du bateau par rapport au Nord. Elle renvoie un objet `CommandOutput` qui contient la valeur à passer au safran ainsi qu'aux voiles.

La deuxième fonction utile est `arrived(float precision)`, cette fonction renvoie si le bateau est arrivé aux coordonnées de sa destination plus ou moins la précision, en fonction de la position donnée à `navigate()`.

La dernière fonction à utiliser est la fonction `setBuoy(Point buoy)`, cette fonction redéfinit la destination du Navigator.

2.4 Exemple d'utilisation de la librairie

```

#include <Navigator.h>

// Création de la destination avec ses coordonnées géographiques
Point buoy = {47.2734979, -2.213848};
// Création de l'objet Navigator avec la destination initiale
Navigator nav = Navigator(buoy);
// Récupération des commandes des actionneurs à partir des données d'entrée
CommandOutput commands = nav.navigate(boat, wind_angle, course);
if (nav.arrived()) {
    nav.setBuoy(nextPoint);
}
// Utilisation des commandes
safran = commands.safran;
sails = commands.sails
  
```

3. Send Logs

Le but de cette librairie est de transmettre, via une liaison série, l'ensemble de la télémétrie des capteurs intégrés dans le bateau. Cette télémétrie servira à la station sol pour afficher la position en temps réel du bateau et l'ensemble des mesures des capteurs.

La librairie est composée des fonctions suivantes :

- `SendLogs();`

Cette fonction est le constructeur de la classe, elle permet d'initialiser les 3 variables de statut des 3 capteurs à 1 (fonctionnel), la variable de l'état de pilotage à 0 (mode AUTO) et d'établir une liaison série configurée à 38400 bauds.

- void setStatusSensors(bool S_IMU, bool S_GPS, bool S_GIR);

Permet de mettre à jour, dans la librairie, l'état des 3 capteurs (IMU, GPS, Girouette). Un état à 1 signifie que le capteur est fonctionnel, un état à 0 signifie que le capteur est en défaut.

- void setStatusMode(bool S_MODE);

Permet de mettre à jour, dans la librairie, l'état de pilotage. Un état à 1 signifie que le pilotage est dans le mode MANUEL, un état à 0 signifie que le pilotage est dans le mode AUTO.

- void sendLogs(float GPS_lat,
 float GPS_long,
 float IMU_degNord,
 float GIR_degVent);

Permet d'envoyer sur la liaison série, l'ensemble de la télémétrie des 3 capteurs en plus de leur statut (OK ou KO). Les données envoyées sont pré formalisées pour l'interprétation et la reconstruction d'un format JSON sur la réception.

En fonction de l'état des 3 variables d'état des capteurs, la fonction enverra ou non les valeurs de la télémétrie.

- void printDouble(double val, unsigned int precision);

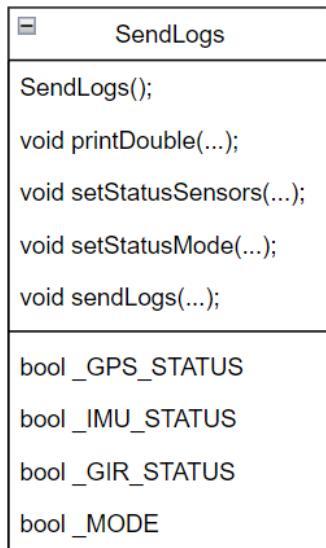
Cette fonction interne à la librairie permet d'envoyer sur la liaison série des nombres à décimales, ce qui n'est pas possible avec les fonctions de base.

La fonction prend en entrée le nombre à transmettre (type float), et la précision souhaitée (entier non signé).

Exemple : printDouble(3.1415, 100); // Affiche 3.14 (Avec deux décimales)

Exemple : printDouble(3.1415, 100000); // Affiche 3.14150 (Avec cinq décimales)

Structure de la librairie :



Utilisation :

ATTENTION : Cette librairie n'est pas chargée de vérifier le fonctionnement des capteurs. Il est nécessaire d'actualiser l'état dans le programme principal et de mettre à jour 3 variables. Ces 3 variables sont ensuite utilisées pour la fonction **setStatusSensors**.

Dans la fonction principale, il faut créer un objet de type **SendLogs** :

```
SendLogs comXbee;
```

Au niveau de l'initialisation, il faut initialiser l'objet avec son constructeur :

```
void setup() {
    comXbee = SendLogs();
}
```

A chaque mise à jour des valeurs d'un des capteurs, il est nécessaire de mettre à jour les statuts des capteurs et de transmettre l'ensemble des données.

```
void loop() {
    comXbee.setStatusSensors(Etat_GPS,Etat_IMU,Etat_Gir);
    comXbee.setStatusMode(Mode);
    comXbee.sendLogs(data1,data2,data3,data4);
    delay(500);
}
```

4. IMU

L'imu est implémenté au sein de la librairie projet_voilier_imu et dispose de la classe suivante :

```
class Imu_Voilier {  
private:  
    int16_t angle_value;  
    bool calibrated;  
public :  
    Imu_Voilier();  
    bool imu_update_data();  
    int16_t imu_get_data();  
    int16_t imu_get_data_with_negative();  
    void imu_display_data();  
    bool read_calibration_status();  
};
```

Utiliser l'IMU s'effectue en trois temps :

- On vérifie que l'IMU est opérationnel avec la commande `read_calibration_status()` qui est censée retourner vrai si l'IMU est prête.
- On met à jour la valeur de l'IMU avec `imu_update_data()`
- On lit la valeur de l'angle du nord magnétique avec `imu_get_data()`. `imu_get_data()` retourne une valeur entre 0 et 3600 et `imu_get_data_with_negative()` retourne la valeur de l'angle au format -1800/1800.

5. GPS

L'imu est implémenté au sein de la librairie projet_voilier_gps et dispose de la classe suivante :

```
class Gps_Voilier {  
public :  
    SFE_UBLOX_GNSS myGNSS;  
    GPS_DATA gps_data;  
    byte gps_status;  
    byte gps_reconnecting_count;  
    bool booted;  
    int update_data_gps();  
    Gps_Voilier();  
    void gps_display_data();  
    GPS_DATA gps_get_data();  
    bool gps_update_status();
```

```
};
```

Utiliser le GPS s'effectue en trois temps :

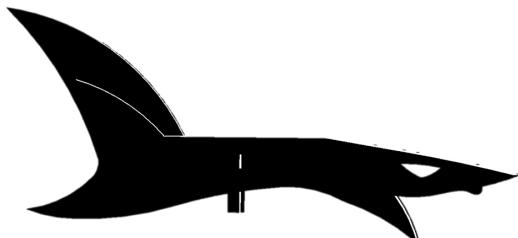
- On vérifie que l'IMU est opérationnel avec la commande `gps_update_status()` qui est censée retourner vrai si le GPS est prêt.
- On met à jour la valeur des coordonnées avec `update_data_gps()`. La fonction retourne 0 si des données ont été récupérées, et différentes valeurs d'erreur autrement
- On lit la valeur du GPS avec la fonction `gps_update_status()`. Cette fonction retourne la structure suivante :

```
struct GPS_DATA {  
    int32_t latitude;  
    int32_t longitude;  
    int32_t altitude;  
    byte SIV;  
};
```

Cette structure permet de récupérer les différentes données utiles à la navigation

6. Girouette

La girouette est imprimée en résine et tourne librement à l'aide d'un roulement à billes. Elle présente un encodeur magnétique lui permettant de connaître l'orientation du vent sans affecter sa rotation. Concernant sa modélisation, nous nous sommes inspirées du logo, que nous avons modifié pour améliorer sa prise au vent.



Le capteur utilisé est un AS5600, dont nous utilisons l'interface I2C afin de récupérer l'angle de l'aimant relatif au capteur. Nous obtenons alors une précision de 2^{12} (4096 valeurs), mais pour simplifier les calculs en aval nous décidons de passer à une précision au degré près.



Concernant la disposition de la girouette sur le bateau, nous avions dans un premier temps décidé de la placer à l'avant pour éviter d'apporter un stress trop important sur le mât. Après plusieurs itérations de celle-ci nous avons remarqué que sa masse était très faible et qu'une disposition sur le mât était envisageable. Nous sommes alors partis sur cette option afin de maximiser sa prise au vent.

7. CommandeServo

ATTENTION : Pour que cette librairie fonctionne, il est nécessaire d'installer la librairie ServoESP32.

La librairie est composée des fonction suivante :

- void set_gouv(int pin_g);
- void set_voile(int pin_v);

Elles servent à définir les Pins utilisées pour commander les servomoteurs liées à la voile et au safran avec des signaux PWM.

Dans notre cas, voile : GPIO 3, safran : GPIO2.

- void cmd_gouv(int cmd_g);
- void cmd_voile(int cmd_v);

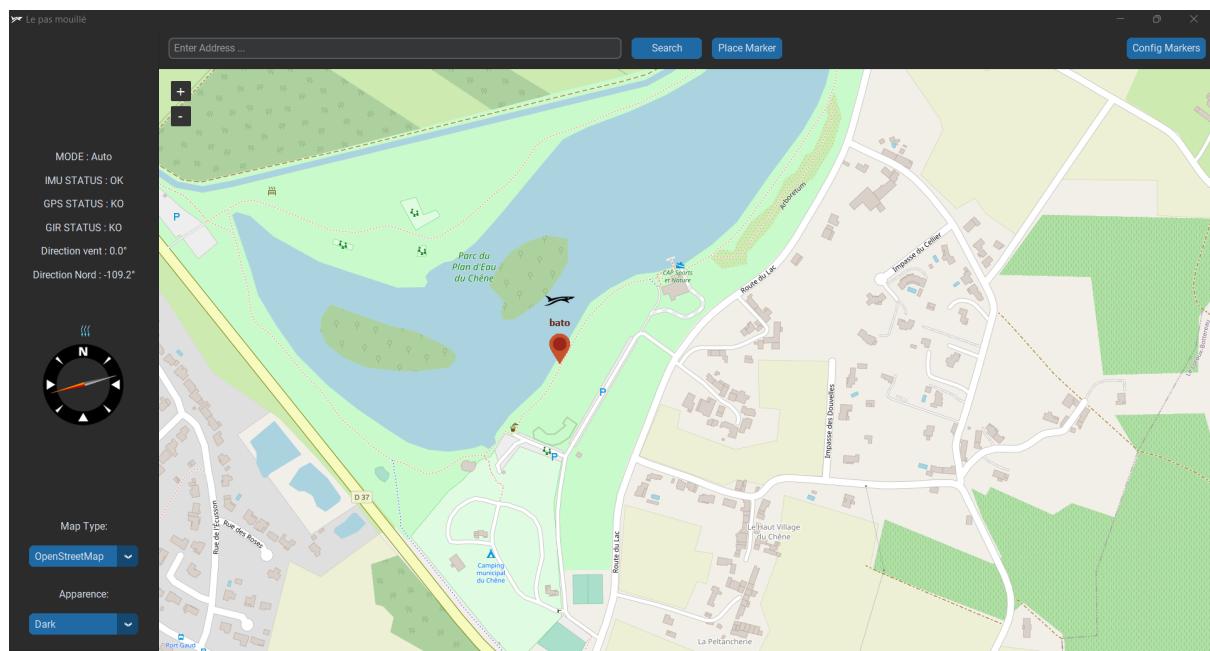
Elles servent à traduire une commande entre 0% (gauche / bâbord) et 100% (droite / tribord), en un signal PWM à envoyer aux servomoteurs de la voile et du safran.

IV. Interface et visualisation des logs

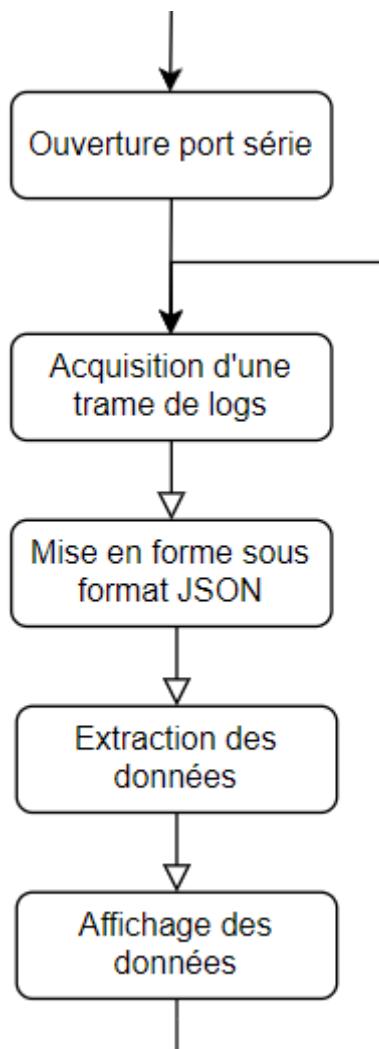
L'interface est réalisée en Python avec les librairies suivantes :

- tkinter (interface graphique)
- serial (communication port série)
- json (manipulation de données sous format JSON)
- Thread (gestion de tâches)

L'interface graphique :



Principe de fonctionnement :



Le programme ouvre le port série défini par l'utilisateur, et commence l'acquisition des données, ligne par ligne.

Une trame commence par le champ "SOF" et se termine par le champ "EOF"

Le programme fait la détection du début de trame pour concaténer l'ensemble des données reçues lignes par lignes jusqu'à la fin de la trame.

A la fin de cette étape, le programme enregistre toutes les données dans un fichier sous format JSON :

```
{  
    "SYSTEM_STATUS" : {  
        "GPS" : 1,  
        "IMU" : 1,  
        "GIR" : 1,  
        "MODE" : 0  
    },  
    "GPS" : {  
        "latitude" : 47.24879,  
        "longitude" : -1.54864  
    },  
    "IMU" : {  
        "degNord" : 31.000  
    },  
    "GIROUETTE": {  
        "degVent": 357.000000  
    }  
}
```

A partir de ce fichier, le programme va extraire les données en fonction de l'état des capteurs. Si un des capteurs est dans un état KO ('0'), le programme va garder la dernière valeur validée.

L'actualisation de l'interface graphique se fait dès la réception d'une nouvelle donnée.

V. Crédits

- Louis GUIBERT
- Théo GABOREAU
- Valentin BOUDEVIN
- Thomas DEBEIRE
- Nolann BORJON