



ICAM NANTES

Rapport Projet ROBONAV 2025 Phase 1

THED KAMGA
thed.kamga@2026.icam.fr
AGATHE DAUDENTHUN
agathe.daudenthun@2026.icam.fr
Tuteur : NICOLAS FERRY
Partenaire : JEAN FRUITET

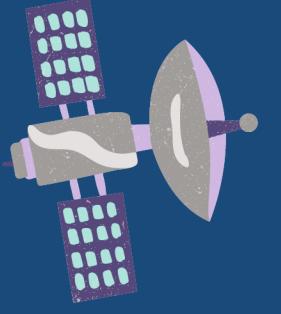


TABLE DES MATERIES

01	Présentation du projet
02	Nos objectifs
03	1er DGPS
04	DGPS M8N
05	DGPS LC29HDA
06	Résumé du projet & prochains objectifs

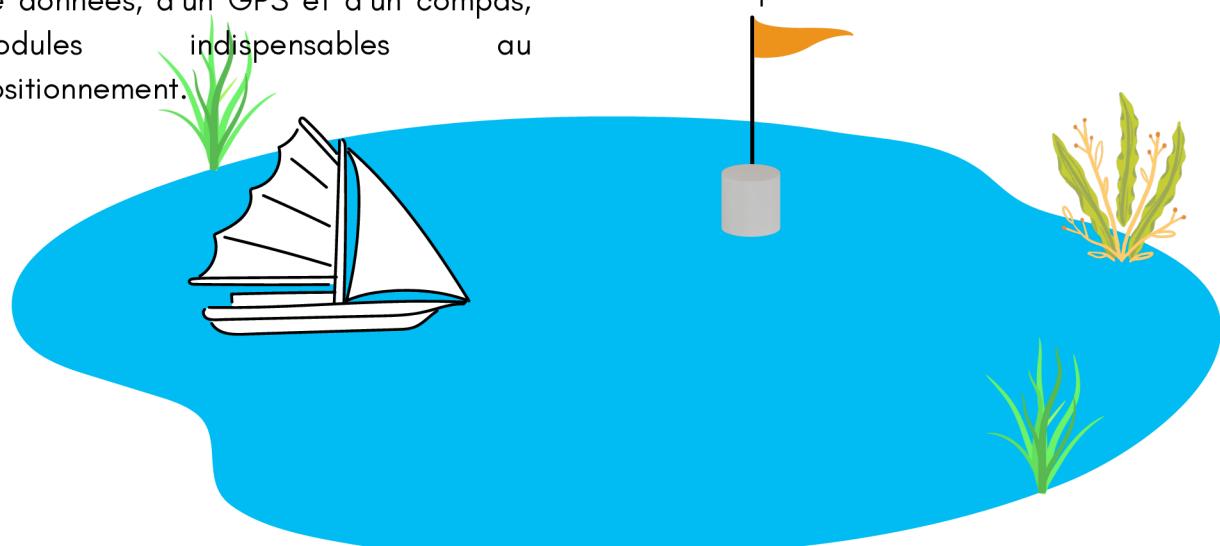


01 Présentation du projet

INTRODUCTION

Le projet RoBoNav a pour objectif de développer un système innovant de bouées de régate autonomes, pour la voile radiocommandée, capables de se positionner avec précision grâce à la technologie GPS. Ce dispositif vise à simplifier et fiabiliser la mise en place des parcours nautiques, en particulier dans un contexte de compétition où la précision de placement et la réactivité sont essentielles quand les conditions météo changent.

Cette année, nous entamons le 3ème année de travail sur ce projet. Aujourd’hui, la bouée est construite et opérationnelle. Elle est dirigée par une radiocommande 2.4 Mhz actionnant 2 turbines immergées permettant les déplacements sur 2 axes. De plus la bouée est dotée d’une carte contrôleur, d’une antenne WiFi pour la transmission de données, d’un GPS et d’un compas, modules indispensables au positionnement.

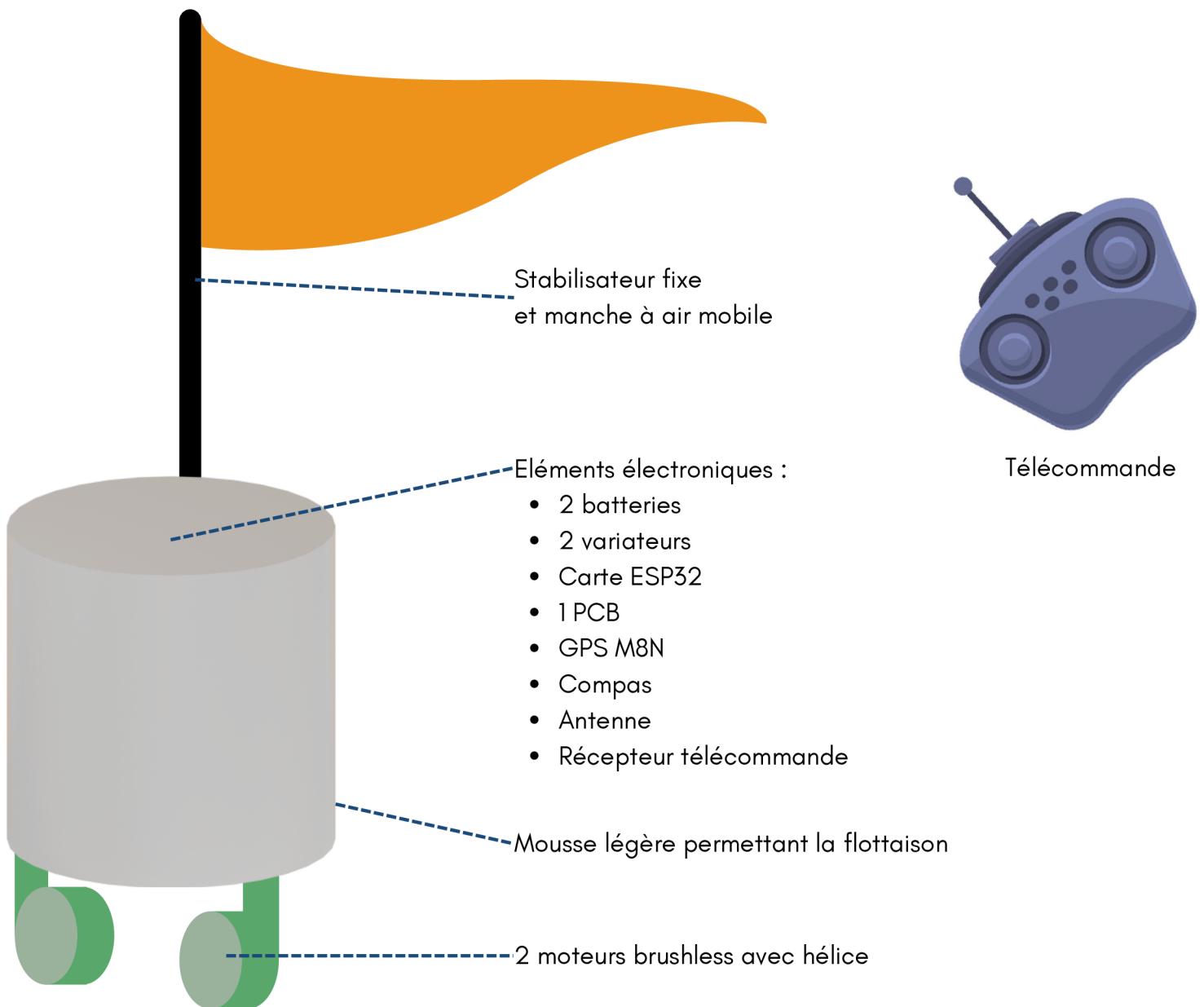


Ainsi, au moment où nous récupérons la bouée, cette dernière peut être positionnée à un point sur un lac et placée en mode amarrage virtuel (garder sa position malgré des forces extérieures tel que le vent ou le courant). Ce dernier point est très peu avancé. C'est à notre équipe (Thed et Agathe) de travailler sur cette partie détaillée les pages suivante.

Ce rapport présente l'état d'avancement du projet à ce jour (du 01/02/2025 au 24/04/2025), en mettant en lumière les étapes déjà réalisées, les défis rencontrés, ainsi que les prochaines phases prévues. Il s'inscrit dans une démarche de suivi régulier pour assurer une coordination efficace entre les différents acteurs impliqués et garantir le respect des objectifs techniques et calendaires fixés.

01 - Présentation du projet

PRÉSENTATION DE LA BOUÉE

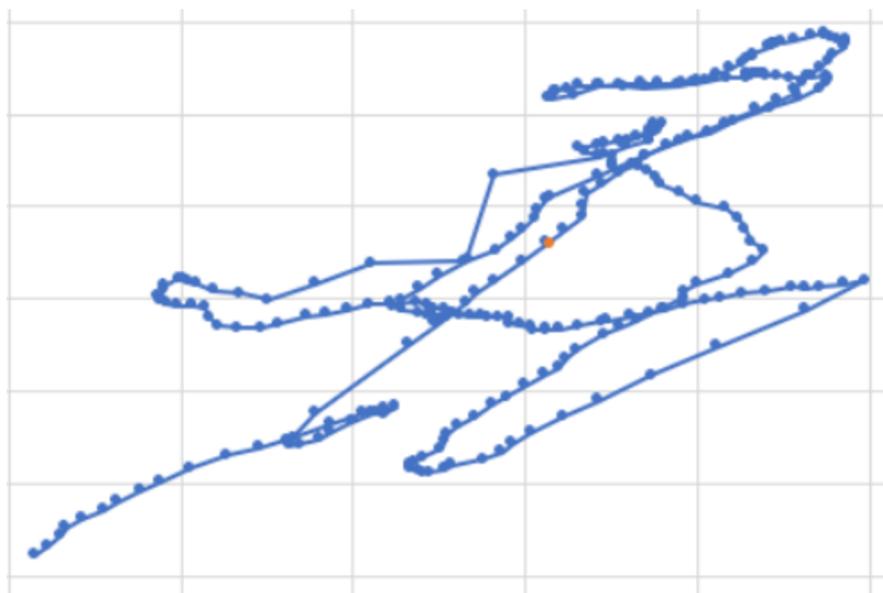


02 - Nos objectifs

Afin d'arriver à notre objectif : un amarrage virtuel de la bouée, il reste 2 grands points (détaillés ci-dessous) sur lesquels nous travaillons ce semestre.

1. Optimiser le positionnement GPS des bouées

En effet, vous observez ci-dessous le comportement des données fournies par un GPS sans correction (fonctionnement actuel). En bleu les positions reçus par le GPS et en Orange la position réelle du GPS. Ici le GPS est immobile, ce sont ses valeurs qui varient.



Cette dispersion, due à différents facteurs comme les interférences atmosphériques ou la géométrie des satellites, rendrait difficile un positionnement stable et fiable des bouées, notamment dans un contexte de régates où une grande précision est requise pour délimiter les parcours.

Notre objectif sera donc d'améliorer la précision des données GPS.

Comment améliorer un GPS ?

Il existe deux principaux moyens d'améliorer la position fournie par un GPS.

- En intégrant un système de correction différentielle (**DGPS / DGNSS**)
- En intégrant un système de cinématique en temps réel : **RTK** (Real-Time Kinematic)

02 - Nos objectifs

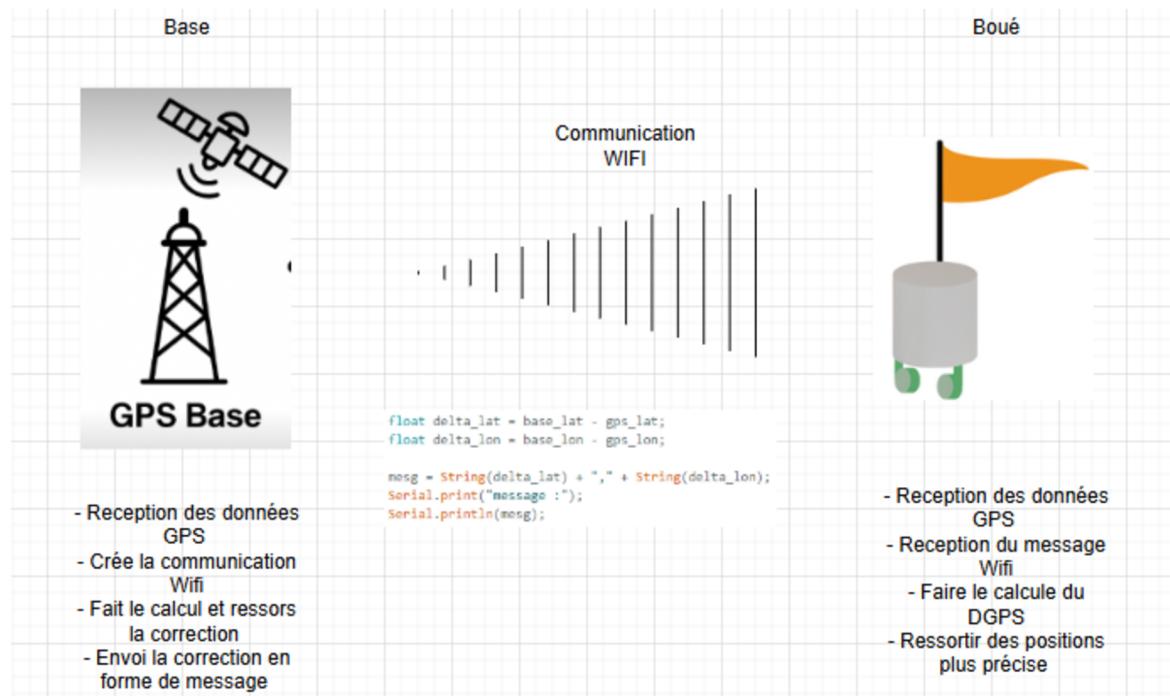
Qu'est ce qu'un DGPS ?

Le DGPS utilise une station de base fixe dont on connaît précisément la position. Cette station reçoit les mêmes signaux satellites que les autres GPS. Elle compare la position reçue avec sa vraie position (qu'elle connaît déjà). Elle calcule l'erreur et envoie une correction aux autres GPS (appelés rovers) de la zone, via un lien (Wi-Fi, radio, etc.). Les rovers (ici les bouées) appliquent cette correction et améliorent nettement leur précision (inférieure au mètre).

Qu'est ce qu'un RTK ?

Un système RTK est aussi un GPS différentiel ; il utilise des récepteurs GPS plus performants qui ne tirent pas leur position (longitude, latitude, altitude) des valeurs plus ou moins dégradées fournies par les constellations de satellites, mais la calculent directement par triangulation à partir des différences de phases entre les ondes porteuses des signaux reçus depuis une constellation de satellites... Avec cette technique il est possible d'arriver à un positionnement d'une précision de l'ordre du centimètre.

De plus un DGPS RTK peut accéder par GSM à des positions de références diffusées par un réseau de balises fixe connectées entre elles par Internet.



02 - Nos objectifs

2. Amélioration du contrôle des moteur

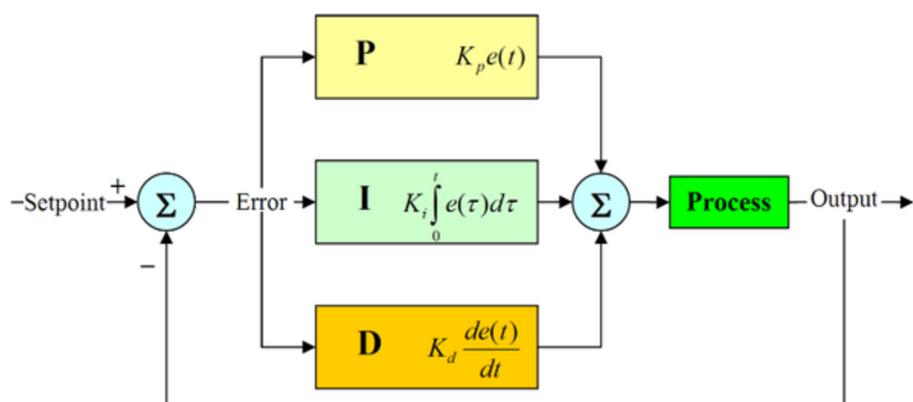
Nos prédecesseurs sur ce projet ont remarqué que les turbines ne sont pas parfaites et, à tension égale, ne tournent pas à la même vitesse. Cette légère différence de vitesse entre les deux moteurs, bien que minime, peut entraîner une déviation progressive de la trajectoire de la bouée.

Le calibrage des moteurs devrait permettre, à l'aide du compas embarqué et d'un PID programmé, de rectifier cette erreur en envoyant des tensions différentes à chaque moteur pour les faire tourner à la même vitesse et faire avancer la bouée en ligne droite.

Qu'est ce qu'un PID ?

Un régulateur PID (Proportionnel – Intégral – Dérivé) est un système de contrôle automatique largement utilisé pour maintenir une grandeur physique (comme la température, la pression ou la vitesse) à une valeur souhaitée, appelée consigne. Il agit, en comparant en permanence la valeur mesurée à la consigne, et en ajustant la commande, de minimiser leur écart.

La détermination des coefficients K est l'objet d'une approche expérimentale.



De plus, après l'amélioration du GPS, la fonctionnalité de maintien de position sera ajoutée sous forme d'asservissement : si la bouée détecte qu'elle s'est éloignée de sa position cible, elle ajustera automatiquement sa trajectoire pour y revenir.

03 - 1er DGPS



Les composants

Atom Lite

Pour faire un DGPS il nous faut donc 2 GPS et 2 cartes électroniques capables de lire et de calculer des données et dotées d'antennes WiFi. Au début du projet nous n'avions que c'est deux éléments : une carte Atom Lite et un module GPS M5Stack. Dans l'attente de recevoir d'autres modules GPS plus performants, nous avons créé notre premier DGPS avec ces composants. L'Atom Lite est doté d'une carte ESP32-PICO. Elle est ainsi programmable via l'application Arduino et dispose d'un module Wifi.

La programmation

Pour la programmation, nous y sommes allés par étapes :

- 1) La réception de données GPS
- 2) La communication Wifi
- 3) Le calcul d'erreur et de correction

1) La réception de données GPS

Le M5Stack est un GPS configuré pour recevoir des trames NMEA.

Qu'est ce qu'une trame NMEA ?

Une trame NMEA est une ligne de texte ASCII, généralement limitée à 82 caractères, et respecte une structure définie :

`$[ID][Type],[Données1],[Données2],...*[Valeur de contrôle]`

Par exemple, une trame \$GPGGA fournit des données essentielles de positionnement, telles que la latitude, la longitude, l'altitude, le nombre de satellites utilisés et la qualité de la position :

`$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47`

Ainsi, grâce à cette trame, nous obtenons les informations souhaitées :

Latitude : `48.07038`

Longitude : `11.31000`

03 - 1er DGPS

Pour une carte ESP32, il y a une bibliothèque pour décoder c'est trames : [TinyGPS++](#)
 Découvrir le code : ANNEXE 1

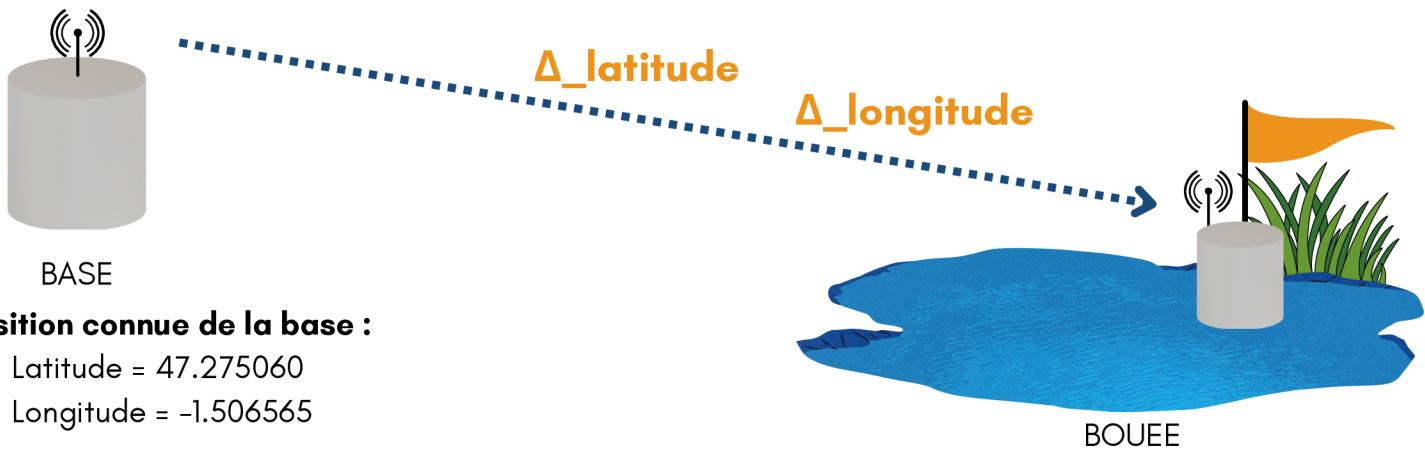
2) La communication Wifi

Pour créer la communication Wifi nous avons travailler avec un système Emetteur / Récepteur. La base (l'émetteur) enverra un message contenant les erreurs de position en latitude et en longitude à la bouée (le récepteur). Dans un premier temps, nous avons ciblé l'envoi de donnée à une bouée précise en utilisant son adresse mac. Dans la continuité du projet, nous allons créer un serveur où plusieurs bouées pourront se connecter et recevoir par "broadcast" les corrections à appliquer.

Découvrir le code : ANNEXE 2

3) Le calcul des erreurs et de la correction

Maintenant que nous pouvons récupérer les données GPS il nous reste à les utiliser pour améliorer les données GPS de la bouée (le récepteur). Ci-dessous un exemple de calcul de correction :



Position connue de la base :

- Latitude = 47.275060
- Longitude = -1.506565

Données GPS :

- Latitude_GPS = 47.275008
- Longitude_GPS = -1.506578

Calcul de l'erreur :

- $\Delta_{\text{latitude}} = \text{Latitude} - \text{latitude}_{\text{GPS}}$
 $= 47.275060 - 47.275008 = 0.000052$
- $\Delta_{\text{longitude}} = \text{Longitude} - \text{Longitude}_{\text{GPS}}$
 $= -1.506565 - (-1.506578) = 0.000013$

Données GPS :

- Latitude_GPS = 47.275328
- Longitude_GPS = -1.506987

Correction :

- Latitude corrigée = Latitude_GPS + Δ_{latitude}
 $= 47.275380$
- Longitude corrigée = Longitude_GPS - $\Delta_{\text{longitude}}$
 $= -1.506974$

03 - 1er DGPS

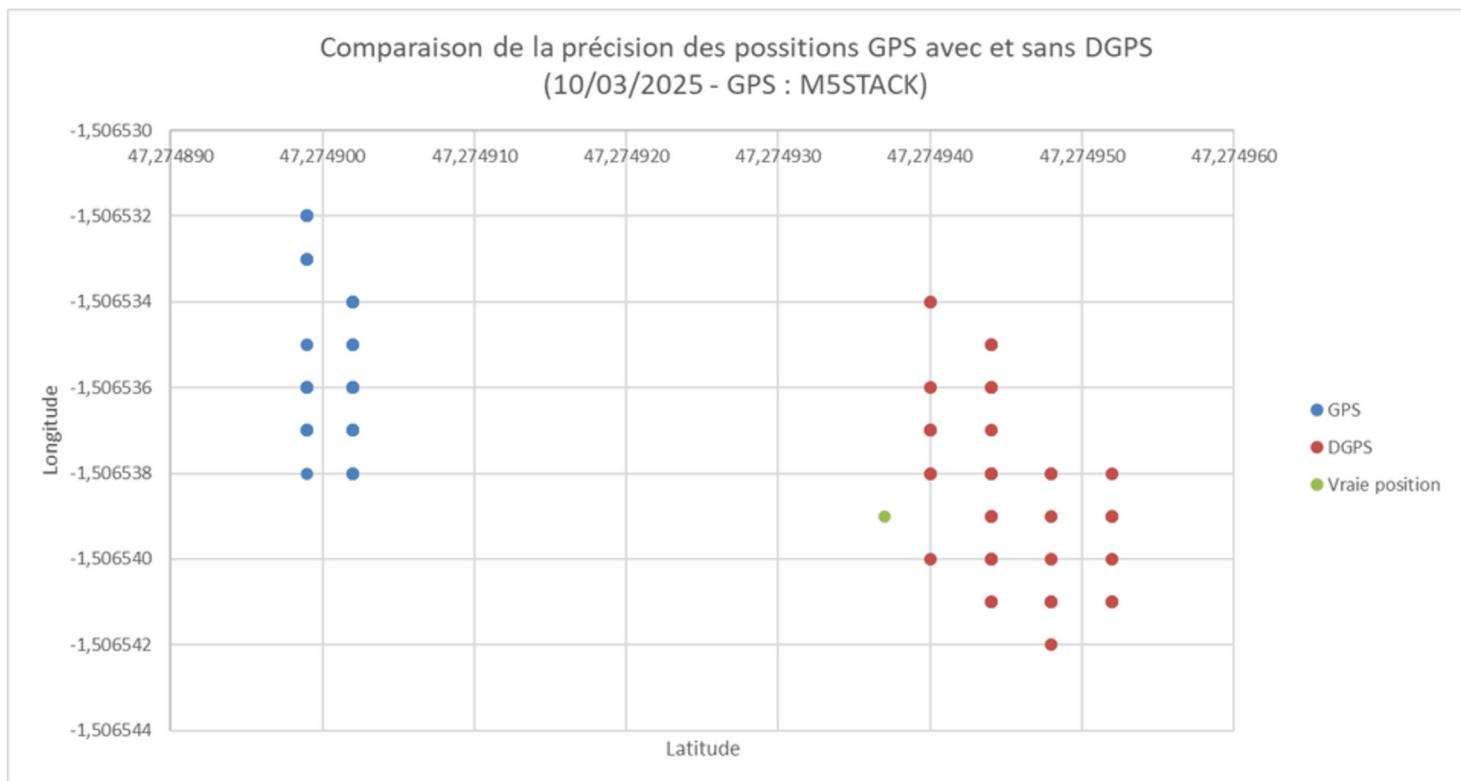
Schéma récapitulatif du système



03 - 1er DGPS

Les résultats

Nous avons fait tourner ce programme pendant 10 minutes. Nous avons donc pu récupérer et analyser des données comme ci-dessous :



Conditions de l'expérience :

- En extérieur
- Distance base-bouée : 3m (impossible de faire plus avec un seul ordinateur)

On observe très bien qu'avec un DGPS, les coordonnées sont plus proches de la vraie position de la bouée. Cependant, il aussi flagrant qu'il y a une plus grande dispersion.

Pour améliorer la qualité des données, il faudra être dans des conditions plus réalistes. A commencer par éloigner la base et la bouée d'au moins 10m. De plus, il nous faut une plus grande quantité de points. Durant notre test les GPS perdaient souvent la connexion satellite, car il ne sont pas de bonne qualité, et donc sur 10 min, beaucoup de points n'étaient pas traitables.

04 - DGPS UBlox M8N

Les composants



GPS UBlox M8N



ESP32 WROOM

Comme lors du premier test, la réalisation d'un système DGPS nécessite l'utilisation de deux modules GPS, cette fois des Ublox M8N, plus performants. Cependant, leur utilisation demande une étude approfondie des protocoles Ublox dont une bonne compréhension du protocole NMEA et du protocole binaire UBX propriétaire qui seul permet de paramétriser la communication des GPS. Chaque GPS est connecté à un ESP32, programmé via Arduino, et équipé d'un module Wi-Fi pour la communication.

La programmation

Pour la programmation, nous avons suivi les mêmes étapes :

- 1) La réception de données GPS
- 2) La communication Wifi
- 3) Le calcul d'erreur et de correction

1) La réception de données GPS

Le système utilise un module GPS U-Blox communiquant via une liaison série. Les données GPS sont reçues sous forme de trames binaires respectant le protocole UBX. Le microcontrôleur détecte l'entête des trames (**0xB5 0x62**) et extrait les informations utiles comme la latitude, la longitude, le nombre de satellites et la qualité de la position (**fixType**). Ces données sont stockées dans des structures (**NAV_POSLLH**, **NAV_PVT**, etc.) puis exploitées dans la boucle principale du programme.

ROBONAV

04 - DGPS M8N

1) La réception de données GPS

Configuration

```

const unsigned char UBLOX_CONFIG_PORT[] PROGMEM = {
// 0 - Config UBX-PORT - GPS COM-UART1
// 0xB5,0x62,0x06,0x00,0x14,0x00,0x01,0x00,0x00,0xD0,0x08,0x00,0x00,0x84,0x03,0x00,0x01,0x00,0x03,0x00,0x00,0x00,0x00,0x00,0x7E,0xB8
0xB5,0x62,0x06,0x00,0x14,0x00,0x01,0x00,0x00,0xD0,0x08,0x00,0x00,0xC2,0x01,0x00,0x01,0x00,0x03,0x00,0x00,0x00,0x00,0xBA,0x4E
//0xB5,0x62,0x06,0x00,0x14,0x00,0x01,0x00,0x00,0xD0,0x08,0x00,0x00,0xC2,0x01,0x00,0x01,0x00,0x23,0x00,0x00,0x00,0x00,0x00,0xDA,0x0E
//0xB5,0x62,0x06,0x00,0x14,0x00,0x01,0x00,0x00,0xD0,0x08,0x00,0x00,0xC2,0x01,0x00,0x23,0x00,0x01,0x00,0x00,0x00,0x00,0xDA,0x52
//0xB5,0x62,0x06,0x00,0x14,0x00,0x01,0x00,0x00,0xD0,0x08,0x00,0x00,0xC2,0x01,0x00,0x23,0x00,0x03,0x00,0x00,0x00,0x00,0xDC,0x5E
};

```

- 0xB5 0x62 : en-tête UBX (signature).
 - 0x06 0x00 : message CFG-PRT, qui configure un port (ici UART1).
 - La configuration utilisée ici : 115200 bauds, entrée : UBX, sortie : UBX + NMEA.

```

const unsigned char UBLOX_MSG_INIT[] PROGMEM = {
// 1 - Disable NMEA
0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x24, // GxGGA off
0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x01,0x00,0x00,0x00,0x00,0x01,0x01,0x2B, // GxGLL off
0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x02,0x00,0x00,0x00,0x00,0x01,0x02,0x32, // GxGSA off
0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x03,0x00,0x00,0x00,0x00,0x01,0x03,0x39, // GxGSV off
0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x04,0x00,0x00,0x00,0x00,0x01,0x04,0x40, // GxRMC off
0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x05,0x00,0x00,0x00,0x00,0x01,0x05,0x47, // GxVTG off

// 2 - Disable UBX
0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x17,0xDC, //NAV-PVT off
0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x12,0xB9, //NAV-POSLH off
0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xC0, //NAV-STATUS off

// 3 - Enable UBX
0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x07,0x00,0x01,0x00,0x00,0x00,0x00,0x18,0xE1, //NAV-PVT on
0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x02,0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x13,0xBE, //NAV-POSLH on
0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x03,0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x14,0xC5, //NAV-STATUS on

// 4 - Rate
//0xB5,0x62,0x06,0x08,0x06,0x00,0x64,0x00,0x01,0x00,0x01,0x00,0x7A,0x12, // (10Hz)
//0xB5,0x62,0x06,0x08,0x06,0x00,0xC8,0x00,0x01,0x00,0x01,0x00,0xDE,0x6A, // (5Hz)
0xB5,0x62,0x06,0x08,0x06,0x00,0x03,0x01,0x00,0x01,0x00,0x01,0x01,0x39, // (1Hz)

```

- Cela désactive les trames textuelles NMEA (ex : \$GPGGA), qui ne sont pas utiles ici car on utilise UBX (binaire, plus compact et précis).

NAV-PVT, NAV-POSLH, NAV-STATUS désactivés temporairement (pour éviter de les recevoir en double ou les réinitialiser)..

- Réactive NAV-PVT : donne toutes les infos principales (temps, position, vitesse, état fix).
 - Réactive NAV-POSLH : donne uniquement la position latitude/longitude/altitude.
 - Réactive NAV-STATUS : donne l'état du fix (2D, 3D, différentiel...)

```
const unsigned char UBX_HEADER[] = { 0xB5, 0x62 }; // Entête sur deux octets
const unsigned char NAV_POSLLH_HEADER[] = { 0x01, 0x02 }; // Message POSLLH
const unsigned char NAV_STATUS_HEADER[] = { 0x01, 0x03 }; // Message STATUS
const unsigned char NAV_PVT_HEADER[] = { 0x01, 0x07 }; // Message PVT
```

- Ces constantes permettent de reconnaître facilement les messages que tu reçois depuis le GPS. Chaque message UBX a un identifiant unique (classe + ID).

04 - DGPS M8N

1) La réception de données GPS

```

struct NAV_PVT {
    unsigned char cls;
    unsigned char id;
    unsigned short len;
    ...
    long lon; // Longitude (deg * 1e-7)
    long lat; // Latitude (deg * 1e-7)
    ...
};

union UBXMessage {
    NAV_POSLLH navPosllh;
    NAV_STATUS navStatus;
    NAV_PVT navPvt;
};

UBXMessage ubxMessage;

```

Ces structures décrivent le format des messages reçus du module GPS U-Blox (notamment les messages NAV-PVT, NAV-POSLLH, etc.).

```

if( config_gps_ok ) {
    int msgType = processGPS(); // Décode un message GPS reçu

    if ( msgType == MT_NAV_POSLLH ) {
        float gps_lat = ubxMessage.navPosllh.lat;
        float gps_lon = ubxMessage.navPosllh.lon;
        ...
    }
    else if ( msgType == MT_NAV_PVT ) {
        Serial.print("[PVT] SV: ");
        Serial.print(ubxMessage.navPvt.numSV);
        Serial.print(" fixType: ");
        Serial.print(ubxMessage.navPvt.fixType);
        ...
    }
}

```

C'est là que les messages sont identifiés et traités une fois reçus

04 - DGPS M8N

2) La communication Wifi

La communication Wi-Fi entre les deux GPS a été mise en place en configurant l'ESP32 de la base en point d'accès (SoftAP) avec un nom de réseau et un mot de passe. Ce module envoie ensuite les coordonnées corrigées en temps réel via UDP en broadcast sur le réseau local. Le second GPS, configuré comme rover, se connecte au point d'accès et reçoit ces données pour améliorer sa précision.

```
#include <WiFi.h>
#include <WiFiUdp.h>
...
```

Ces deux lignes incluent les bibliothèques nécessaires pour gérer la connexion Wi-Fi et la communication UDP sur un microcontrôleur compatible avec le langage Arduino, comme l'ESP32.

```
// --- Wi-Fi Access Point ---
const char* ssid = "BASE_GPS";
const char* password = "base_gps";

// --- UDP broadcast ---
WiFiUDP udp;
const int udpPort = 4210;
const char* udpAddress = "255.255.255.255";
```

Cette ligne définit un point d'accès Wi-Fi avec le SSID "BASE_GPS" et le mot de passe "base_gps", puis configure un objet UDP pour envoyer des données en broadcast sur le port 4210 à l'adresse "255.255.255.255".

```
// Lancer le point d'accès Wi-Fi
WiFi.softAP(ssid, password);
delay(1000); // petit délai pour laisser le temps de se lancer

Serial.println("Point d'accès créé !");
Serial.print("IP de l'ESP32 : ");
Serial.println(WiFi.softAPIP()); // souvent 192.168.4.1
```

Ce code lance un point d'accès Wi-Fi avec les informations fournies (SSID et mot de passe), attend une seconde pour que l'accès soit prêt, affiche l'adresse IP de l'ESP32, puis initialise une communication UDP sur le port 4210.

```
// Envoi en UDP broadcast
udp.beginPacket(udpAddress, udpPort);
udp.write((const uint8_t *)mesg.c_str(), mesg.length());
udp.endPacket();

// Vérification de l'état du Wi-Fi et des appareils connectés
int connectedDevices = WiFi.softAPgetStationNum(); // Nombre d'appareils connectés au SoftAP
Serial.print("Appareils connectés au point d'accès: ");
Serial.println(connectedDevices);
```

Ce code envoie un message en broadcast via UDP à l'adresse et au port spécifiés, puis vérifie et affiche le nombre d'appareils connectés au point d'accès Wi-Fi créé par l'ESP32.

04 - DGPS M8N

3) Le calcul d'erreur et de correction

Le DGPS consiste à utiliser un GPS fixe (base) pour calculer l'erreur de position en comparant sa position mesurée à sa position réelle, puis à transmettre cette correction au GPS mobile afin d'améliorer sa précision.

```
float gps_lat = ubxMessage.navPosllh.lat;
float gps_lon = ubxMessage.navPosllh.lon;
//Serial.print("[POSLLH] iTOW:"); Serial.print(ubxMessage.navPosllh.iTOW);
//Serial.print(" lat/lon: "); Serial.print(gps_lat); Serial.print(","); Serial.print(gps_lon);
//Serial.print(" hAcc: "); Serial.print(ubxMessage.navPosllh.hAcc/1000.0f);
//Serial.println();

float delta_lat = base_lat - gps_lat;
float delta_lon = base_lon - gps_lon;

mesg = String(delta_lat) + "," + String(delta_lon);
Serial.print("message :");
Serial.println(mesg);
```

Cette portion de code récupère la latitude et la longitude mesurées par le GPS, calcule l'écart entre ces coordonnées et celles d'une base fixe, puis crée un message contenant ces différences (delta_lat, delta_lon) à envoyer en correction différentielle (DGPS).

```
float delta_lat = part1.toInt();
float delta_lon = part2.toInt();

Serial.print("delta_lat : ");
Serial.println(delta_lat, 6);
Serial.print("lon : ");
Serial.println(delta_lon, 6);

loat true_lat = gps_lat + delta_lat;
loat true_lon = gps_lon + delta_lon;

Serial.print("lat_corrigé : ");
Serial.println(true_lat, 6);
Serial.print("lon_corrigé : ");
Serial.println(true_lon, 6);
```

Ce code convertit les corrections de latitude et longitude reçues (part1 et part2) en valeurs numériques, les ajoute aux coordonnées GPS mesurées localement pour obtenir la position corrigée (true_lat, true_lon), puis affiche cette position améliorée par le DGPS.

04 - DGPS M8N

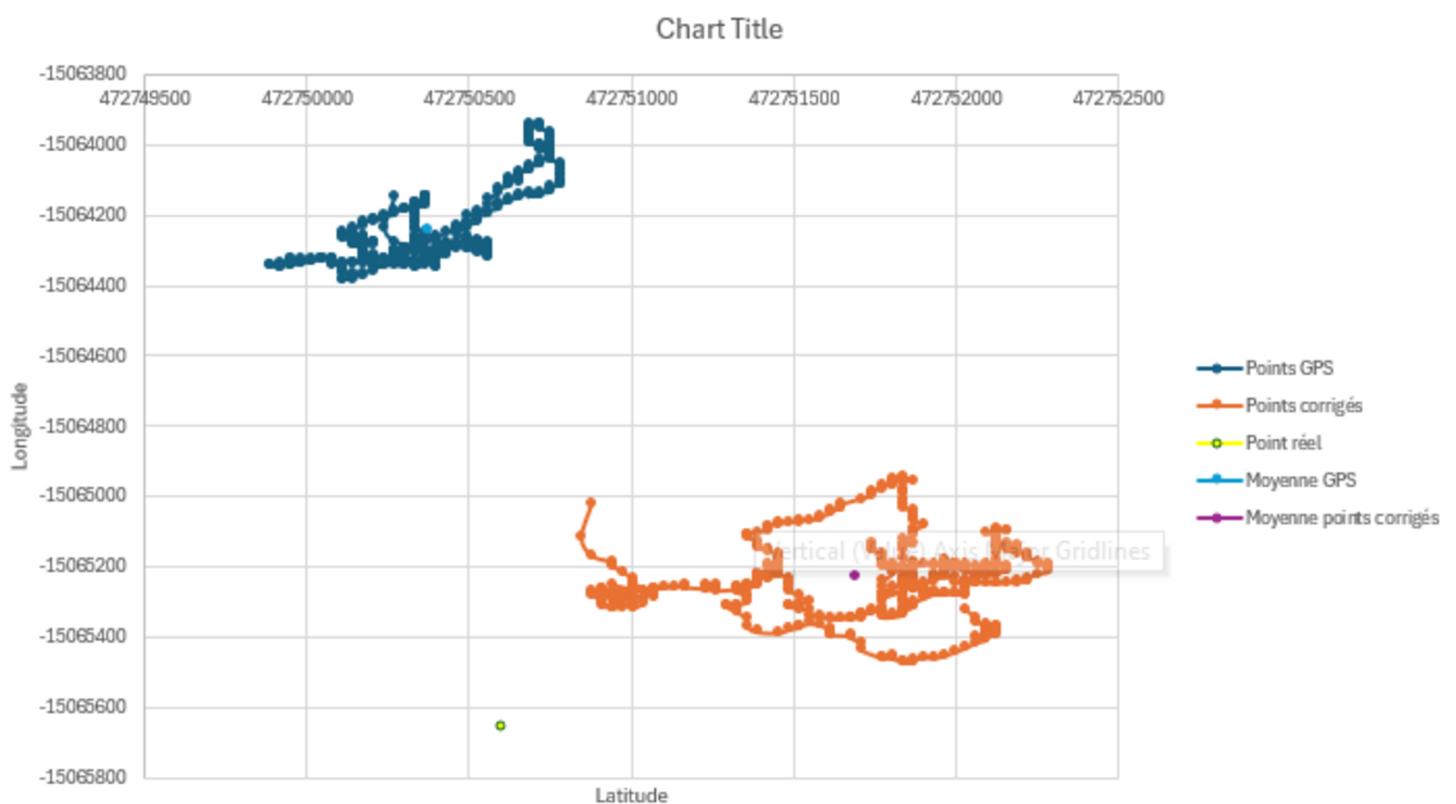
Schéma récapitulatif du système



04 - DGPS M8N

Les résultats

Nous avons laissé le programme tourner pendant une longue période, puis nous avons collecté toutes les positions GPS obtenues, que nous avons ensuite traitées sur Excel afin d'analyser le comportement de chaque point et vérifier s'ils se rapprochent ou non de la position réelle du GPS.



- La correction DGPS a augmenté la dispersion en latitude, ce qui peut être lié à une instabilité dans le calcul de correction ou à une mauvaise configuration du module en mode base.
- Même constat pour la longitude, la dispersion augmente légèrement après correction. Cela signifie que les points corrigés sont plus dispersés entre eux que les points bruts.
- La précision en latitude est moins bonne après correction. Le GPS brut était plus proche du point réel.
- En revanche, la correction a amélioré significativement la précision en longitude, réduisant l'erreur de plus de 70 %.
- Le "point réel" est sujet à caution car nous l'avons obtenu sur Google Map. Il faut améliorer cette mesure.

04 - DGPS M8N

Conclusion générale :

- Le système DGPS a amélioré la précision en longitude mais a dégradé la précision et la dispersion en latitude.
- Cela pourrait indiquer que la correction envoyée par la base est incorrectement calculée ou appliquée, notamment sur l'axe nord-sud (latitude).

Problèmes rencontrés

Comme problèmes rencontrés, nous avons d'abord travaillé avec un module GPS M8N sans obtenir de résultats, avant de nous rendre compte plus tard qu'il n'était pas adapté à notre besoin. De plus, le manque de connaissances sur les protocoles UBLOX et NMEA nous a également causé du retard dans l'avancement du projet.

05 - DGPS LC29HDA



Le module Quectel LC29HDA est un récepteur GNSS (Global Navigation Satellite System) avancé, conçu pour fournir une géolocalisation précise, notamment grâce à la technologie RTK (Real-Time Kinematic). Il est particulièrement adapté aux applications nécessitant une précision centimétrique, telles que les drones, les véhicules autonomes, la topographie et les systèmes de navigation avancés.

Module GPS RTK. Antenne hélicoïdale haute qualité



Caractéristiques

- Formats de sortie : Supporte les formats NMEA standard pour les données de positionnement et RTCM pour les corrections différentielles en mode RTK.
- Précision centimétrique avec RTK : En mode RTK, le module peut atteindre une précision de l'ordre du centimètre en quelques secondes.
- Mode RTK : Il est possible de configurer un module en station de base pour émettre des corrections RTCM. Ces données peuvent être transmises via des liaison série, radio ou Internet. Ainsi, la bouée recevra des trames et sera capable de corriger et améliorer la précision de positionnement.

Donc l'avantage de ce module est sa précision mais aussi la simplicité d'utilisation des trames en mode RTCM. Il n'est plus nécessaire de décoder les trames venant de la base.

Configuration

Le module LC29H est très bien conçu pour répondre au besoin de notre projet. Sa palette de configuration est très large.

Par contre il est nettement plus coûteux que le module UBlox M8N

Voici quelques commandes qui nous permettront d'obtenir une base RTCM et une bouée en trame NMEA auto corrigée.

ROBONAV

05 - DGPS LC29HDA

\$POTMRESTOREPAR*13 : cette commande permet de reset les configurations

POUR LA BASE

\$PAIR432,1*22: cette commande configure le module pour qu'il émette des messages RTCM3.x de type MSM7

\$PAIR434,1*24 : cette commande active l'émission du message RTCM de type 1005, qui contient les coordonnées de la station de base. Ce message est essentiel pour que les récepteurs RTK puissent calculer leur position relative avec précision.

\$PAIR436,1*26 : cette commande permet au module de transmettre les messages RTCM contenant les éphémérides des satellites GPS et GLONASS. Ces informations sont cruciales pour le calcul précis de la position en mode RTK.

\$POTMCFGRCVRMODE,W,2*29 : cette commande définit le module en tant que station de base, ce qui est nécessaire pour émettre des corrections différentielles vers les récepteurs mobiles.

\$POTMCFGVIN,W,2,0,0,X,Y,Z*CS : remplacez X, Y et Z par les coordonnées ECEF (Earth-Centered, Earth-Fixed) de votre station de base. Le *CS représente le checksum à calculer pour valider la commande. Cette configuration est essentielle pour que les corrections RTCM soient précises.

POUR LA BOUEE

\$PAIR062,0,01*0F : cette commande active l'émission du message NMEA GGA, qui fournit des informations sur la position, la qualité du signal et le nombre de satellites utilisés. Ce message est souvent requis par les logiciels de traitement GNSS.

\$POTMCFGNMEADP,W,3,8,3,3,3,3*39 : cette commande configure la précision des décimal des trames NMEA.

EN FIN DE CONFIGURATION

\$POTMSAVEPAR*5A : après avoir configuré le module, cette commande enregistre les paramètres dans la mémoire non volatile, assurant leur persistance après un redémarrage.

05 - DGPS LC29HDA

Programmation

Concernant à présent la programmation, comme nous utilisons également une carte EPS32, le code est similaire au M8N pour la partie communication Wifi.

Pour la partie configuration, il suffit d'initialiser en écrivant les commandes ci-dessus.

```
void setup() {  
    Serial2.begin(115200, SERIAL_8N1, rxPin, txPin);  
  
    Serial2.println("$PQTMRESTOREPAR*13");  
    ...  
    Serial2.println("$PQTMSAVEPAR*5A");  
}
```

A ce jour nous avons constaté qu'une de nos carte LC29HDA ne répondait pas correctement aux configuration que nous lui envoyons, notamment pour la configuration en mode base.

Cette difficulté a été résolue suite au changement de firmware.

Le programme est terminé, nous comptons maintenant passer à une batterie de tests.

05 - DGPS LC29HDA

Schéma récapitulatif du système



06 - Résumé du projet

Dans le cadre de ce projet, nous avons mis en place un système DGPS (Differential GPS) visant à améliorer la précision des relevés de position à l'aide de deux modules GPS : un en mode base (fixe) et l'autre en mode mobile (rover). Le principe repose sur l'envoi en temps réel des erreurs de position mesurées par le GPS fixe vers le GPS mobile, afin d'améliorer la précision de la position du rover (bouée) et éviter des errements parasites.

Pour la phase de test et de développement, nous avons d'abord utilisé un microcontrôleur ATOM Lite avec différents modules GPS. Ensuite nous avons travaillé avec un module GPS Ublox M8N, celui installé sur la bouée, mais nous n'avons pas obtenu de résultats assez précis.

Nous allons ensuite utiliser le module LC29HDA, plus récent et plus performant, mais nous avons rencontré un souci technique : l'un des modules ne répondait pas correctement aux configurations envoyées, en particulier pour l'activation du mode base. Suite à un échange avec le support technique ce dysfonctionnement matériel a été résolu par une mise à jour du firmware.

Le programme, quant à lui, est a priori achevé. Il inclut la mise en place d'un point d'accès Wi-Fi, la transmission des erreurs par trames RTCM, la récupération des erreurs et l'écriture dans le module LC29HDA et le décodage des trame NMEA du GPS mobile. Nous allons maintenant lancer les tests finaux afin de valider le fonctionnement global du système.

Quelle est la prochaine étape ?

La prochaine étape est de continuer les tests des DGPS afin de trouver la meilleure solution qui nous permette d'avoir une précision de l'ordre du mètre, et mieux si possible il se peut que le RTK ne soit pas nécessaire.

Une fois les tests DGPS validés et le système pleinement opérationnel, la prochaine phase du projet portera sur l'amélioration du contrôle des moteurs de la bouée. En effet, nous avons constaté une légère différence de vitesse entre les deux moteurs, ce qui peut provoquer une déviation progressive de la trajectoire. Pour corriger cela, un calibrage plus précis sera mis en place avec un KPI, affin d'ajuster dynamiquement la puissance envoyée à chaque moteur afin de maintenir un cap stable.

En parallèle, nous développerons une fonctionnalité de maintien de position : si la bouée dérive par rapport à sa position cible (définie par le GPS), elle corrigera automatiquement sa trajectoire pour y revenir. Cette fonctionnalité combinerà les données GPS corrigées par le DGPS et un algorithme d'asservissement de navigation basique pour ajuster le mouvement de la bouée de manière autonome.

ANNEXE 1

Récupération de données GPS module M5Stack

```
#include <TinyGPS++.h>          Bibliothèque permettant de décoder les trames NMEA
#include <HardwareSerial.h>       Bibliothèque permettant d'utiliser les port UART

static const int RXPin = 32, TXPin = 26;    Broches UART du GPS
static const uint32_t GPSBaud = 9600;         Vitesse de communication du GPS

TinyGPSPlus gps;  Déclaration d'une instance nommée "gps" de la classe TinyGPSPlus

HardwareSerial gpsSerial(1); Utilisation du second port série

// INITIALISATION
void setup() {
  gpsSerial.begin(GPSBaud, SERIAL_8N1, RXPin, TXPin);
}                                              Cette ligne initialise la communication série
                                                avec le module GPS. SERIAL_8N1 est le format
                                                de communication série (8 bits de données, pas
                                                de parité, 1 bit d'arrêt).

void loop() {
  while (gpsSerial.available()) {             Vérification si des données sont disponibles en lecture sur le port
                                                série connecté au module GPS.
    gps.encode(gpsSerial.read());            Lecture des données et envoie à la bibliothèque TinyGPS++ pour
                                            décoder la trame.

    if (gps.location.isUpdated()) {          Retourne true si de nouvelles données de localisation (latitude et
                                            longitude) ont été décodées depuis le dernier appel. Cela permet
                                            de s'assurer que les données utilisées sont récentes.

      GPS_lat = gps.location.lat();        Retourne la latitude et la longitude actuelle en degrés décimaux.
      GPS_long = gps.location.lng();
    }
}
```

ROBONAV

ANNEXE 2

Connection Wifi - cartes ESP32-PICO Atome Lite

// ----- EMMETEUR -----

```
#include <esp_now.h>      Ajout des bibliothèques nécessaires pour utiliser les fonctionnalités Wi-Fi et ESP-NOW de l'ESP32.
```

```
uint8_t broadcastAddress[] = {0x94, 0xB9, 0x7E, 0x9F, 0x98, 0x1C};
```

Définition de l'adresse MAC du récepteur, permettant à l'émetteur de savoir à quel appareil envoyer les données.

```
void sentCallback(const uint8_t *mac, esp_now_send_status_t status) {
    Serial.print("État de l'envoi : ");
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Succès" : "Échec");
}
```

Cette fonction est appelée après chaque envoi pour indiquer si l'envoi a été un succès ou un échec.

// INITIALISATION

```
void setup() {
    WiFi.mode(WIFI_STA);          Configuration de l'ESP32 en mode station (STA), nécessaire pour utiliser ESP-NOW.
```

```
if (esp_now_init() != ESP_OK) {
    Serial.println("ESP-NOW Init Failed");
    return;
}
```

Cette section initialise le protocole ESP-NOW. Si l'initialisation échoue, un message d'erreur est affiché.

```
esp_now_register_send_cb(sentCallback);
```

Cette ligne enregistre une fonction de rappel (sentCallback) qui sera appelée après chaque tentative d'envoi, indiquant si l'envoi a réussi ou échoué.

```
esp_now_peer_info_t peerInfo;
memset(&peerInfo, 0, sizeof(peerInfo));
memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.channel = 0;
peerInfo.encrypt = false;
```

Cette section configure les informations du pair (le récepteur) et l'ajoute à la liste des pairs ESP-NOW.

```
if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Échec d'ajout du récepteur !");
    return;
}
Serial.println("Récepteur ajouté avec succès !");
```

}

```
void loop() {
    esp_err_t result;
    result = esp_now_send(broadcastAddress, (uint8_t *)&myData, sizeof(myData));
```

Envie les données contenues dans myData au récepteur spécifié par broadcastAddress.

}

ANNEXE 2

Connection Wifi - cartes ESP32-PICO Atome Lite

```
// ----- RECEPTEUR -----
#include <WiFi.h>
#include <TinyGPS++.h>

void receiveCallback(const esp_now_recv_info_t *info, const uint8_t *incomingData, int len) {
    memcpy(&myData, incomingData, sizeof(myData));
    String message = myData.text;
}
```

Cette fonction est appelée automatiquement lorsqu'un message est reçu via ESP-NOW. Elle copie les données reçues dans la structure myData.

```
// INITIALISATION
void setup() {
    WiFi.mode(WIFI_STA);

    if (esp_now_init() != ESP_OK) {
        Serial.println("ESP-NOW Init Failed");
        return;
    }

    esp_now_register_recv_cb(receiveCallback);
}
```