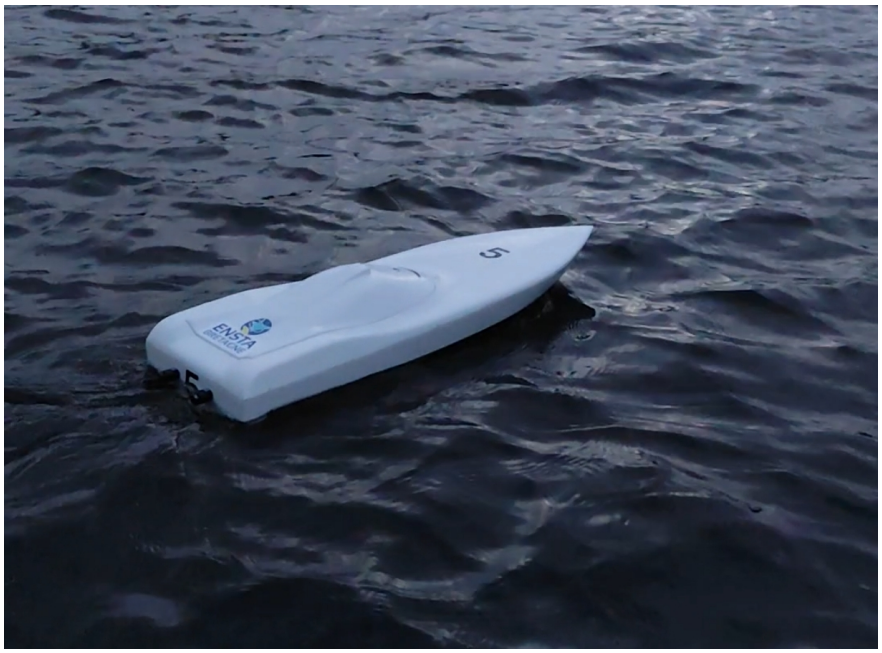
Compte Rendu

Année universitaire 2025-2026
Semestre 4

Guerlédan DD-BOAT 5



Groupe :
BIECHE Matys Adéas

Encadrant :
M.JAULIN Luc

Ensta Campus de Brest

Engagement de non plagiat

Je soussigné, BIECHE Matys Adéas,

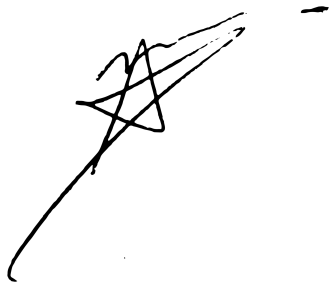
Déclare avoir pris connaissance de la charte de l'école et notamment du paragraphe spécifique au plagiat.

Je suis pleinement conscient(e) que la copie intégrale sans citation ni référence de documents ou d'une partie de document publiés sous quelques formes que ce soit (ouvrages, publications, rapports d'étudiant, internet etc...) est un plagiat et constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.

En conséquence, je m'engage à citer toutes les sources que j'ai utilisées pour produire et écrire ce document.

Fait le 15 Février 2026

Signatures



Résumé

Dans le cadre de la seconde année d'école d'ingénieurs à l'ENSTA campus de Brest, j'ai pu travailler sur la mise en oeuvre d'un bateau de surface robotisé, le DD-Boat, sur le lac de Guerlédan.

L'objectif principal est de réaliser chaque jour une mission (calibrage d'un capteur, implémentation du contrôle autonome...) avec/sur le DD-Boat afin de mieux appréhender la robotique et les concepts appris en cours.

Ce rapport a donc pour vocation de présenter mes travaux et mes choix d'implémentation, ainsi que les tests et résultats ayant pu être réalisés durant cette semaine.

De même, nous mettrons en lumière les compétences acquises, mais aussi les découvertes et recherches réalisées.

Sommaire

1 Introduction

Cette semaine à Guerlédan s'est articulée autour d'une mission principale pour le DD-Boat :

— Effectuer un suivi de polygone en essaim de DD-BOAT.

Afin d'effectuer ces missions, il était nécessaire de configurer le DD-Boat et de mettre oeuvre un certains nombres d'éléments :

— Calibrage du magnétomètre
— Suivi de segments par attraction potentielle
— Communication entre DD-BOAT
— Plan de suivi à plusieurs DD-BOAT

La calibration et le contrôle de notre bateau sont réalisés par le biais de scripts Python, transmis à la Raspberry Pi via SSH par le biais d'un réseau wifi accessible sur tout le site d'expérimentation.

Dans la suite, nous décrirons par jours les éléments de cours, ainsi que les tests et résultats qui nous ont permis d'effectuer divers objectifs sur toute la semaine, avec comme finalité la réalisation de la mission décrite précédemment.

Il est à noter que le DDBoat-5 que nous avons manipulé n'utilise pas son GPS principal, mais le second, ce qui explique des différenciations dans l'initialisation des drivers par rapport aux autres équipes.

L'ensemble des scripts, une explication plus détaillée du code, ainsi que quelques vidéos sont disponibles via les liens suivants :

Gitlab ENSTA Bretagne : https://gitlab.ensta-bretagne.fr/biechema/dd-boat-5-guerledan-2.git

2 Jours 1

2.1 Objectif 1 : Calibration / Autocalibration de l'IMU (magnétomètre)

Le premier objectif du jour était de supprimer les biais hard iron et les déformations soft iron du magnétomètre pour obtenir une norme de champ magnétique constante et un cap fiable. La démarche consiste à ajuster une ellipsoïde par moindres carrés sur les mesures

brutes (x, y, z), puis de la transformer de manière affine en la ramenant à une sphère unité.

2.2 Rappels théoriques

2.2.1 Interférences Magnétiques

Il existe deux types d'interférences magnétiques :

Le **hard iron** est dû à la présence d'éléments ferromagnétiques et aux aimantations permanentes couplées au champ magnétique terrestre

Le **soft iron** est simplement dû à la manière dont se dirige le champ magnétique de la Terre en un point donné et comment il oriente les éléments non magnétisés.

2.2.2 Méthode par moindres carrés

Du fait de ces interférences, les mesures ne sont plus une sphère centrée mais une ellipsoïde décalée. Cette ellipsoïde peut être modélisée par une formule quadratique dont on cherche les coefficients par méthode des moindres carrées.

On cherche donc à minimiser :

$$J(Q, b) = \sum_{n=1}^{N} \Big((x_n - b)^\top Q (x_n - b) - 1 \Big)^2,$$

avec Q définie positive. On paramètre Q et b via le vecteur $p \in \mathbb{R}^{10}$ tel que

$$M(x)^\top p = 0, \quad M(x) = \big[x_1^2,\, x_2^2,\, x_3^2,\, x_1x_2,\, x_1x_3,\, x_2x_3,\, x_1,\, x_2,\, x_3,\, 1\big]^\top,$$

où

$$Q = \begin{pmatrix} p_1 & \frac{p_4}{2} & \frac{p_5}{2} \\ \frac{p_4}{2} & p_2 & \frac{p_6}{2} \\ \frac{p_5}{2} & \frac{p_6}{2} & p_3 \end{pmatrix}, \qquad b = -\tfrac{1}{2}\, Q^{-1} \begin{bmatrix} p_7 \\ p_8 \\ p_9 \end{bmatrix}, \qquad p_{10} = b^\top Q b - 1.$$

La calibration qui ramène le nuage sur la sphère unité se trouve en remarquant que :

En cherchant une transformation du type :

$$y = A^{-1}\,(x + b)$$

On a :

$$(x-b)^\top Q (x-b) = (x-b)^\top Q^{\frac{1}{2}} Q^{\frac{1}{2}} (x-b) = y^\top y = \|y\|^2 \approx 1.$$

D'où :

$$y = Q^{\frac{1}{2}}\,(x - b)$$

Cette transformation assure la calibration de nos données brutes du magnétomètre.

2.3 Correction d'alignement absolu : détermination de $R \in SO(3)$

Après la calibration ellipsoïdale

$$y = Q^{1/2}(x - b),$$

le nuage est ramené sur une sphère unité.

Cependant, cette sphère reste exprimée dans le repère capteur. Il subsiste donc une rotation inconnue entre :
— le repère magnétomètre calibré $\mathcal{R}_m$,
— le repère navigation $\mathcal{R}_n$ (Nord–Ouest–Bas).

L'objectif est d'identifier une matrice de rotation $R \in SO(3)$ telle que

$$y_n = R\, y_m$$

avec

$$y_m = A_{\text{mag}}(x - b).$$

2.3.1 Hypothèse physique

Le champ magnétique terrestre local est constant et de norme unitaire après calibration :

$$\|y_m\| = 1.$$

Dans le repère navigation, il s'écrit :

$$h_n = \begin{pmatrix} \cos I \\ 0 \\ -\sin I \end{pmatrix},$$

où I est l'inclinaison magnétique locale (environ 64°).
Lorsque le robot change d'orientation, le champ mesuré est transformé uniquement par rotation.

2.3.2 Acquisition de deux poses indépendantes

On impose deux orientations connues :
— orientation Nord,

— orientation Ouest.

On mesure alors :

$$v_N = A_{\text{mag}}(x_N - b),$$
$$v_W = A_{\text{mag}}(x_W - b).$$

Ces vecteurs sont exprimés dans le repère magnétomètre.

2.3.3 Construction d'une base mesurée

On définit :

$$v_3 = v_N \times v_W.$$

Si les poses sont non colinéaires :

$$\|v_3\| > 0.$$

On forme la matrice :

$$V = \begin{pmatrix} v_N & v_W & v_3 \end{pmatrix}.$$

2.3.4 Base théorique dans le repère navigation

Dans le repère navigation, les directions attendues sont :
Orientation Nord :

$$y_N = \begin{pmatrix} \cos I \\ 0 \\ -\sin I \end{pmatrix},$$

Orientation Ouest :

$$y_W = \begin{pmatrix} 0 \\ -\cos I \\ -\sin I \end{pmatrix}.$$

On définit :

$$y_3 = y_N \times y_W,$$

et

$$Y = \begin{pmatrix} y_N & y_W & y_3 \end{pmatrix}.$$

2.3.5 Calcul de la rotation

On cherche R telle que : $RV = Y$ Ainsi, $R = YV^{-1}$.

Cette matrice n'est pas exactement orthogonale en raison du bruit et des erreurs numériques.

2.3.6 Projection sur $SO(3)$

On projette alors R_{raw} sur le groupe des rotations :

$$R = UV^\top,$$

où

$$R_{\text{raw}} = U\Sigma V^\top$$

est la décomposition en valeurs singulières (SVD).

On impose ensuite :

$$\det(R) = +1.$$

Cette étape garantit :

$$R^\top R = I, \qquad \det(R) = 1.$$

2.4 Calibration magnétique complète

La calibration complète devient :

$$y_n = R\, A_{\text{mag}}(x - b),$$

avec :
— b : correction hard iron,
— $A_{\text{mag}} = Q^{1/2}$: correction soft iron,
— R : alignement avec le repère navigation.
Le cap s'obtient alors par :

$$\psi = \text{atan2}(y_{n,y},\, y_{n,x}).$$

Cette méthode correspond à une identification directe d'une rotation 3D à partir de deux directions connues. Elle complète la calibration ellipsoïdale en supprimant l'erreur d'alignement absolu du cap.

2.5 Protocole d'acquisition

Le script `ReadIMU.py` permet d'acquérir les données :
— Environnement : éloignement des masses ferromagnétiques, moteurs coupés.
— Acquisition : magnétomètre, données bruts sur quelques minutes.
— Manœuvres : rotations du DDBoat afin de couvrir le maximum d'espace angulaire sur les trois axes.
— Export : fichier .txt avec trois colonnes $(x_{\text{mag}}, y_{\text{mag}}, z_{\text{mag}})$.

2.6 Algorithme du calcul de R

Le script `compute_R.py` permet de calculer la matrice de correction R vue juste avant :
— Environnement : éloignement des masses ferromagnétiques, moteurs coupés.
— Acquisition : magnétomètre, données bruts sur quelques minutes.
— Manœuvres : pointage du DDBoat vers le Nord et L'Ouest.
— Export : .csv avec le résultat du calcul.

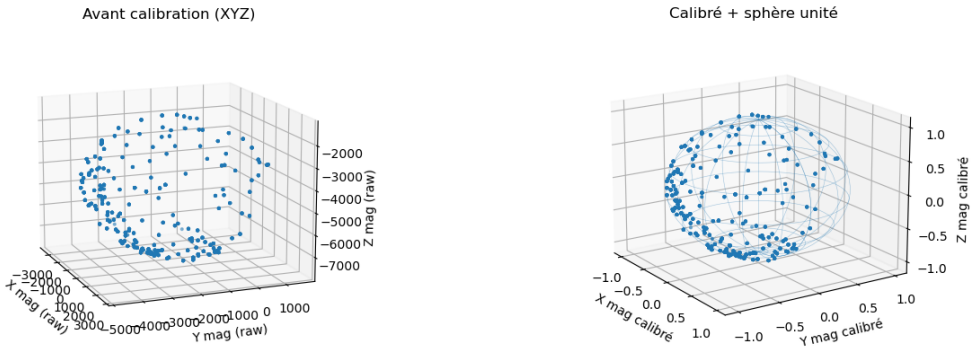
2.7 Algorithme d'autocalibration

Les données sont ensuite traitées dans notre script principal `Autonomous_capture.py`.

1. **Régression quadratique** par moindres carrés (SVD) pour estimer Q à partir des points (x_i, y_i, z_i).
2. **Restauration des paramètres géométriques** : Calcul du centre b, normalisation pour obtenir Q définie positive.
3. **Factorisation et calibration** point par point : $y = Q^{\frac{1}{2}}(x - b)$.
4. **Contrôles** : visualisation 3D avant/après via le script généré via ChatGPT pour le plot `plotIMUCalib.py`.

2.8 Validation

La figure figure 2.8 montre le résultat d'une calibration. On y remarque difficilement l'ellipsoïde excentrée (a), mais l'échelle permet de s'y convaincre. A droite (b), on obtient les points qui sont ajustés sur la sphère centrée.



(a) Nuage brut

(b) Nuage calibré + sphère unité

FIGURE 1 – Exemple de calibration magnétomètre par ajustement d'ellipsoïde et projection sur la sphère.

2.9 Objectif 2 : Suivi de segments d'un polygone

Le second objectif de la journée correspond au suivi de segments. Cette section décrit la loi de guidage et de commande utilisée pour assurer le suivi d'un segment par le DD-BOAT.

2.9.1 Définition du segment

On considère un segment défini par deux points :

$$a, \quad b.$$

On pose :

$$d = b - a.$$

L'angle directeur du segment est :

$$\rho = \text{atan2}(d_y, d_x).$$

2.9.2 Condition de validation du segment

Le passage au segment suivant est déclenché lorsque :

$$(b - a)^\top (b - m) < 0.$$

Cette condition correspond au franchissement du demi-plan perpendiculaire au segment en b.

2.9.3 Erreur transversale

On définit :

$$r = m - a.$$

L'erreur latérale signée par rapport à la droite support du segment est :

$$e = \frac{d_x r_y - d_y r_x}{\|d\|}.$$

Il s'agit du produit vectoriel 2D normalisé :

$$e = \frac{d \times r}{\|d\|}.$$

Propriétés :
— $e > 0$: robot à gauche du segment,
— $e < 0$: robot à droite,
— $e = 0$: robot sur la droite.

2.9.4 Loi de guidage en cap

Le cap désiré est défini par :

$$\theta_{\text{bar}} = \rho - \tanh\Big(\frac{e}{E}\Big)\,,$$

où $E > 0$ est un paramètre sur lequel jouer pour rallier plus ou moins proche le segment. (Par exemple si E = 10, on autorise d'être à 10m du segment)

2.9.5 Commande angulaire

$$\omega = k_1 \,\text{sawtooth}(\theta_{\text{bar}} - \theta).$$

La fonction sawtooth projette l'erreur angulaire dans $[-\pi, \pi]$.
On obtient une loi proportionnelle sur l'erreur de cap :

$$\omega = k_1(\theta_{\text{bar}} - \theta).$$

2.9.6 Commande en vitesse

La vitesse cible est v_{bar}.
On applique donc :

$$a = k_2 \tanh(v_{\text{bar}} - v),$$

2.9.7 Loi complète

La loi de commande s'écrit :

$$\begin{cases} \theta_{\text{bar}} = \rho - \tanh(e/E) \\ \omega = k_1(\theta_{\text{bar}} - \theta) \\ \dot{v} = k_2 \tanh(v_{\text{bar}} - v) \end{cases}$$

Cette structure comporte :

1. une étape géométrique (calcul de l'erreur latérale),
2. une loi de guidage non linéaire saturée,
3. une stabilisation proportionnelle du cap,
4. une régulation bornée de la vitesse.

2.10 Simulation

Le script `simu.py` permet de simuler ce principe. Il dépend de la bibliothèque roblib.

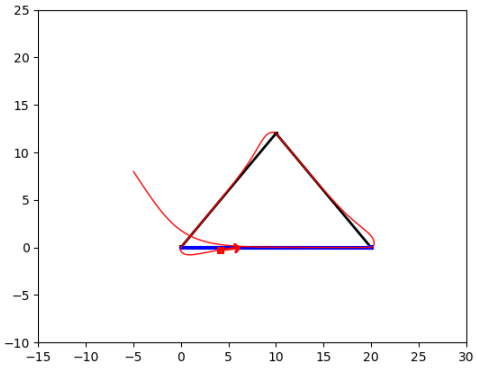


FIGURE 2 – Simulation du suivi de polygone

2.11 Validation de l'algorithme

Le module `path_following.py` implémente la brique géométrique de suivi de segment ou de polygone, utilisée par le script principal `DDBOAT5.py / DDBOAT10.py`.

Son rôle est de :

— Calculer les erreurs géométriques par rapport au segment courant.
— Générer un cap de référence ψ_{ref}.
— Gérer le passage automatique au segment suivant.

2.11.1 Géométrie du segment

À partir de deux points a et b :

— Calcul du vecteur directeur normalisé t.
— Calcul de l'abscisse curviligne projetée s.
— Calcul de l'erreur latérale signée e.

Ces grandeurs sont obtenues via la fonction :

$$(L, t, n, s, e) = \texttt{segment\_errors}(a, b, m)$$

où m est la position courante.

2.11.2 Génération du cap de référence

Deux stratégies sont disponibles :

— **Guidage par droite support :**

$$\psi_{\text{ref}} = \psi_{\text{ligne}} - \psi_{\max} \tanh\Big(\frac{e}{E}\Big)$$

— **Guidage par point de poursuite (lookahead) :**
  — Projection de m sur la tangente.
  — Ajout d'un *lookahead* le long du segment.
  — Cap vers le point cible.

Dans `DDBOAT5.py / DDBOAT10.py`, le mode utilisé est un mélange des deux. En effet, à la fin de cette première journée, seul le guidage par droite support a été réalisé. Nous reviendrons plus tard sur le lookahead et pourquoi il a été utilisé.

2.11.3 Changement de segment

Le passage au segment suivant est déclenché lorsque :

$$(b-a)^\top (b-m) \leq -\text{margin}\,\|b-a\|$$

Ce test correspond au franchissement du demi-plan perpendiculaire en b.

2.11.4 Fonction principale : `polygon_step`

Cette fonction :

1. Calcule le point cible sur le segment courant.
2. Détermine ψ_{ref}.
3. Teste si le segment est terminé.
4. Met à jour l'indice du segment si nécessaire.

Elle retourne :

$$(\psi_{\text{ref}}, \text{idx}, e, s, L, \text{target})$$

2.11.5 Rôle dans `DDBOAT5.py / DDBOAT10.py`

Dans le script principal :

1. Les waypoints GPS sont convertis en coordonnées locales métriques.
2. À chaque itération :
   — Lecture GPS $\rightarrow$ position m.
   — Appel à `polygon_step_geo`.
   — Obtention de ψ_{ref}.
3. Calcul de l'erreur de cap :

$$e_\psi = \text{sawtooth}(\psi_{\text{ref}} - \psi)$$

4. Commande différentielle :

$$\text{yaw\_cmd} = k_\psi e_\psi$$

5. Conversion en PWM moteurs.

2.11.6 Résultats

FIGURE 3 – Suivi du segment

Nous pouvons voir que le DD-BOAT suivait bien le segment.

FIGURE 4 – Suivi du triangle

Mais aussi plutôt bien le triangle.

2.12 Remarques

Mon code de calibration a servi de base à d'autres équipes qui n'y sont pas parvenus afin qu'il puisse continuer les challenges proposés.

3 Jour 2

3.1 Objectif : Suivi de de phase dans le triangle pour consensus

L'objectif du jour était d'implémenter le consensus. Pour ce faire, il fallait dans un premier temps réaliser un suivi de la phase. Puis la communication. Comme j'étais tout seul, j'ai préféré m'attarder sur la phase et l'amélioration de mon algorithme de suivi du triangle.

3.1.1 Algorithme de synchronisation de phase (`phase_tracking.py`)

Le module `phase_tracking.py` implémente la synchronisation temporelle de plusieurs robots le long d'un même polygone.

Son rôle est de :
— Transformer la position du robot sur le polygone en une phase angulaire $\phi_{\text{mes}} \in [0, 2\pi]$.
— Générer une phase de référence $\phi_{\text{ref}}(t)$.
— Calculer une erreur de phase.
— Moduler la vitesse pour assurer la synchronisation.

3.1.2 Partie géométrique : phase mesurée

On considère un polygone composé de segments successifs.

Longueur cumulée On calcule :
— Les longueurs de segments L_k.
— Les longueurs cumulées ℓ_k.
— La longueur totale L_{tot}.

Projection sur le segment courant À partir de la position m, on calcule l'abscisse projetée s sur le segment courant.

La longueur curviligne parcourue est :

$$\ell = \ell_k + s.$$

La phase mesurée est définie par :

$$\phi_{\text{mes}} = 2\pi \frac{\ell}{L_{\text{tot}}} \mod 2\pi.$$

Ainsi, un tour complet du polygone correspond à 2π radians.

3.1.3 Phase de référence

La phase de référence est définie par :

$$\phi_{\text{ref}}(t) = 2\pi \frac{t}{T} + 2\pi \frac{i}{N}.$$

où :
— T : période de mission,
— i : indice du robot,
— N : nombre total de robots.
Le terme $\frac{2\pi i}{N}$ impose un déphasage uniforme entre robots.

3.1.4 Erreur de phase

L'erreur de phase est :

$$d\phi = \text{wrap}(\phi_{\text{ref}} - \phi_{\text{mes}}) \in [-\pi, \pi].$$

Cette opération garantit une erreur minimale angulaire.

3.1.5 Loi de modulation de vitesse

La vitesse est modulée selon :

$$v_{\text{cmd}} = \text{sat}\left(v_{\text{nom}} + k_\phi d\phi\right).$$

avec saturation :

$$v_{\min} \leq v_{\text{cmd}} \leq v_{\max}.$$

Interprétation :
— Si le robot est en retard ($d\phi > 0$) $\rightarrow$ il accélère.
— S'il est en avance ($d\phi < 0$) $\rightarrow$ il ralentit.

3.1.6 Extension multi-robot

Il est possible d'appliquer une correction supplémentaire basée sur la phase d'un pair :

$$v_{\text{cmd}} = v_{\text{nom}} + k_\phi d\phi + (\phi_{\text{peer}} - \phi_{\text{mes}} - 2\pi/N).$$

Ce terme force l'espacement angulaire entre robots.

3.1.7 Rôle dans `DDBOAT5.py / DDBOAT10.py`

Dans `DDBOAT5.py / DDBOAT10.py` :

1. Calcul de ϕ_{mes} à chaque itération.
2. Génération de ϕ_{ref} en fonction du temps.
3. Calcul de $d\phi$.
4. Modulation de la vitesse de base.

Le module `phase_tracking` constitue donc la couche de coordination multi-robot, superposée au guidage géométrique fourni par `path_following`.

Le code a ensuite été validé via des logs affichés en temps réel pendant le parcours du polygone.

3.2 Amélioration de l'algorithme de suivi de segment

En gérant les différents gain, il m'était impossible d'obtenir un suivi de polygone convaincant.

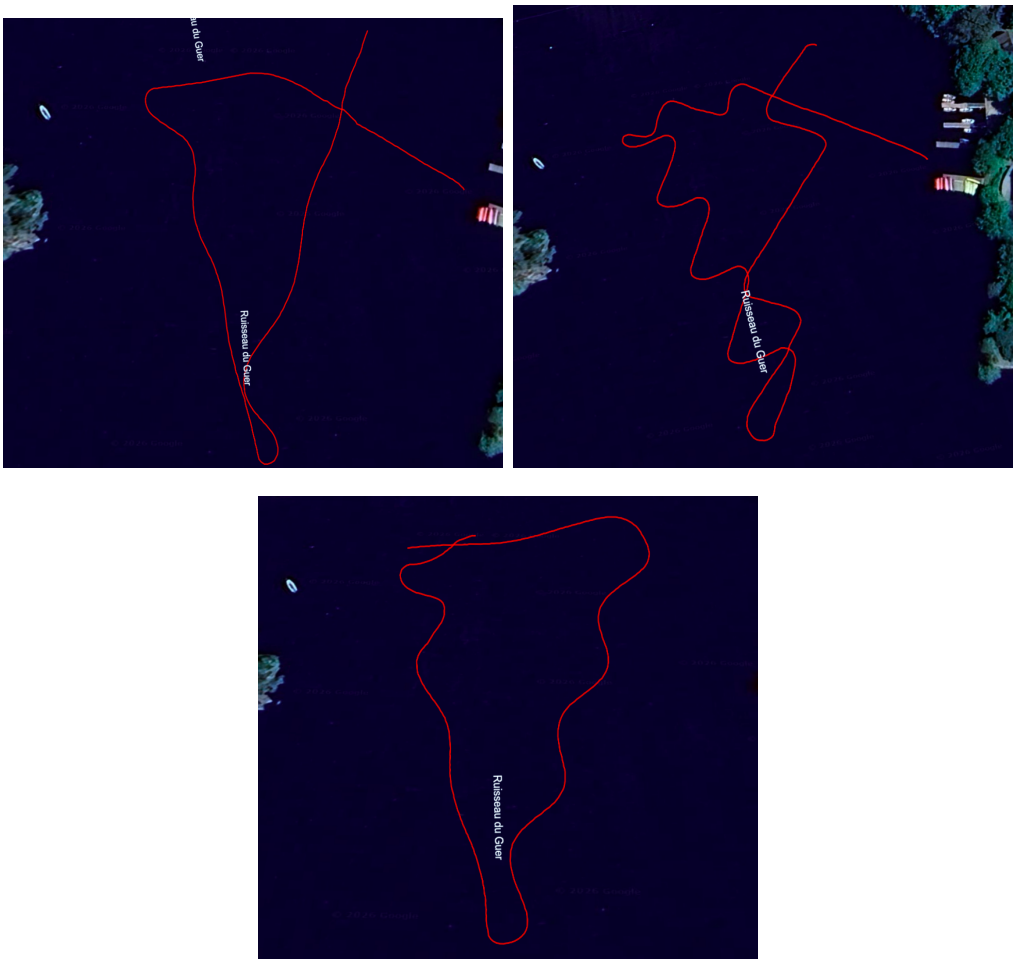


FIGURE 5 – Suivi du grand triangle avec variation des gains

De plus, en changeant l'échelle, le suivi devenait catastrophique :

FIGURE 6 – Suivi du petit triangle de tests

J'ai donc décidé d'ajouter un lookahead, sur conseils de Mr.JAULIN.

FIGURE 7 – Suivi du petit triangle de tests avec lookahead

Le DD-BOAT était ainsi capable de suivre parfaitement les polygones.

4 Jour 3, 4 et 5

4.1 Objectif : Consensus avec Communication

L'objectif de ces deux jours a été l'implémentation d'un algorithme de communication afin d'autoriser le consensus, mais aussi de continuer de fine-tuner les paramètres pour le suivi
Le but était de trouver un groupe avec lequel s'affilier pour réaliser ces tâches.

4.2 Communication

L'algorithme de communication a été réalisé par LUCAS Thibault. Je me suis d'ailleurs associé à son groupe (DD-BOAT 10) qui n'avait pas encore d'algorithme de suivi efficace. Nous avons donc pu échanger nos méthodes et ils ont pu reprendre mon algorithme de suivi afin de réaliser la mission.

4.3 Module de communication (`communication.py`)

Le fichier `communication.py` implémente une couche réseau minimale utilisée dans `DDBOAT5.py / DDBOAT10.py` pour :

— diffuser l'état courant du robot (au moins ϕ_{mes}),
— recevoir l'état d'un robot pair (principalement ϕ_{peer}),
— envoyer des données de log (position, segment) vers un serveur de supervision.

4.4 Architecture générale

La classe `Communication` lance **trois threads** indépendants :

1. **Serveur TCP** : le robot courant se comporte comme un serveur et **broadcast** périodiquement ses données aux clients qui se connectent.
2. **Client TCP** : le robot courant se connecte à un robot pair (DDBoat #XX) et **récupère ses données**.
3. **Client UDP de log** : le robot envoie périodiquement des messages vers un serveur de log.

La période de diffusion est fixée par `broadcast_period` (typiquement 1 Hz dans le code).

4.5 Données échangées

Le code maintient deux ensembles de variables :
— **TX (émission)** : `tx_x, tx_y, tx_phi_mes, tx_segment`
— **RX (réception)** : `rx_x, rx_y, rx_phi_mes, rx_time`
Elles sont mises à jour par l'API :
— `set_tx_data(...)` : mise à jour des données à diffuser/logguer.
— `set_rx_data(...)` : mise à jour des données reçues d'un pair.
— `get_rx_phi(max_age_s)` : retourne ϕ_{peer} si la donnée est récente, sinon `None`.

4.6 Serveur TCP : diffusion des données

Le serveur est démarré avec `run_server()` :
— ouverture d'un socket TCP sur `0.0.0.0:port`,
— attente de connexions (`accept()`),
— pour chaque client : création d'un thread `handle_client()`.
Dans `handle_client()` :

1. génération d'un message par `parse_send_data(tx_x, tx_y, tx_phi_mes)`,
2. envoi périodique au client (`send()`),
3. arrêt propre en fin de mission via l'envoi de `"fin_mission"`.

Remarque importante (format) Le message émis par `parse_send_data` est actuellement :

$$\texttt{"05|"} + \phi_{\text{mes}}$$

donc **le serveur n'envoie pas** (x, y) (malgré les signatures des fonctions).

4.7 Client TCP : récupération des données d'un pair

Le client est démarré avec `run_client(ddboat_number)` :
- construction de l'adresse du pair : `peer_ip = distant_server_ip + "XX"`,
- tentative de connexion TCP (jusqu'à 10 essais),
- boucle de réception :
  - `recv()` d'un message,
  - arrêt si `"fin_mission"`,
  - sinon décodage via `parse_received_data()`,
  - stockage via `set_rx_data()` (timestampé).

4.8 Client UDP de log

Le thread `run_log_client()` :
- ouvre un socket UDP,
- envoie périodiquement vers `log_server_ip:6969` un message formaté par `parse_log_data(tx_x, tx_y, tx_segment)` :

  `"05|x|y|segment"`
- s'arrête à la fin de la mission.

4.9 Intégration dans `DDBOAT5.py / DDBOAT10.py`

Dans `DDBOAT5.py / DDBOAT10.py`, l'intégration se fait en trois étapes :

4.9.1 1) Initialisation et lancement des threads

- Création : `comm = Communication(...)`.
- Lancement :
  - `thread_server = Thread(target=comm.run_server)`
  - `thread_log_client = Thread(target=comm.run_log_client)`
  - optionnel : `thread_client = Thread(target=comm.run_client, args=(peer_ddboat,))`

4.9.2 2) Alimentation des données TX depuis la boucle de navigation

À chaque itération de contrôle, `DDBOAT5.py / DDBOAT10.py` met à jour les données à diffuser/logguer :

`comm.set_tx_data(x=lat, y=lon, phi_mes=`ϕ_{mes}`, segment=seg_idx)`

Ces valeurs sont alors utilisées :
— par le serveur TCP pour diffuser l'état,
— par le client UDP pour logger la trace.

4.9.3 3) Utilisation des données RX dans l'asservissement

Toujours dans la boucle, `DDBOAT5.py / DDBOAT10.py` lit éventuellement la phase d'un pair :

$$\phi_{\text{peer}} = \texttt{comm.get\_rx\_phi(max\_age\_s=peer\_timeout)}.$$

— Si une phase récente est disponible : elle peut être utilisée pour le suivi/espacement.
— Sinon : `DDBOAT5.py / DDBOAT10.py` retombe sur une phase de secours (référence théorique).

4.10 Synthèse fonctionnelle

— `Communication` est un **service asynchrone** : la navigation n'attend jamais le réseau.
— Les échanges sont **périodiques** et **best-effort** (timeout + fallback côté contrôle).
— Le module fournit à `DDBOAT5.py / DDBOAT10.py` un canal pour :
  — partager ϕ_{mes} entre robots,
  — envoyer des logs de mission.

4.11 Résultat

Lors du Jour 4 puis du Jour 5, nous avons pu réaliser le consensus à deux DD-BOAT (le 5 et le 10) et suivre le polygone.

FIGURE 8 – Suivi du polygone en groupe

Conclusion

En conclusion, cette semaine d'experimentation nous a fait toucher du doigt des aspects essentiels du métier d'ingénieur robotique et présenté la réalité du terrain.

En effet, l'approche théorique et simplifiée ou parfois surdéveloppée permet dans une certaine mesure de se préparer aux missions à accomplir. Cependant une perturbation, ou encore une non linéarité des moteurs (plage morte ou irrégulière sur des moteurs, IMU mal placé dans le bateau...) sortent du cadre pédagogique usuel et nous poussent à établir des solutions alternatives pour arriver au résultat attendu.

En clair, cette semaine aura été instructive et confirme d'autant plus mon intérêt pour la robotique.

UE 3.2 - DDBOAT (Guerledan)
Promotion FISE 2027

ENSTA Campus de Brest
2 Rue François Verny
29200 Brest Cedex 9, France

Pedro ALVES DE ARAUJO SILVA
pedro.alves@ensta.fr
Luiz Felipe BARROS ALVES
luiz.barros@ensta.fr
Luíza DIESEL CHESANI
luiza.diesel_chesani@ensta.fr
Letícia Lie FURUTA
leticia.furuta@ensta.fr

Sommaire

Lien vers le dépôt : <https://gitlab.ensta-bretagne.fr/furutale/guerledan-2025>

1 Magnetometer Calibration

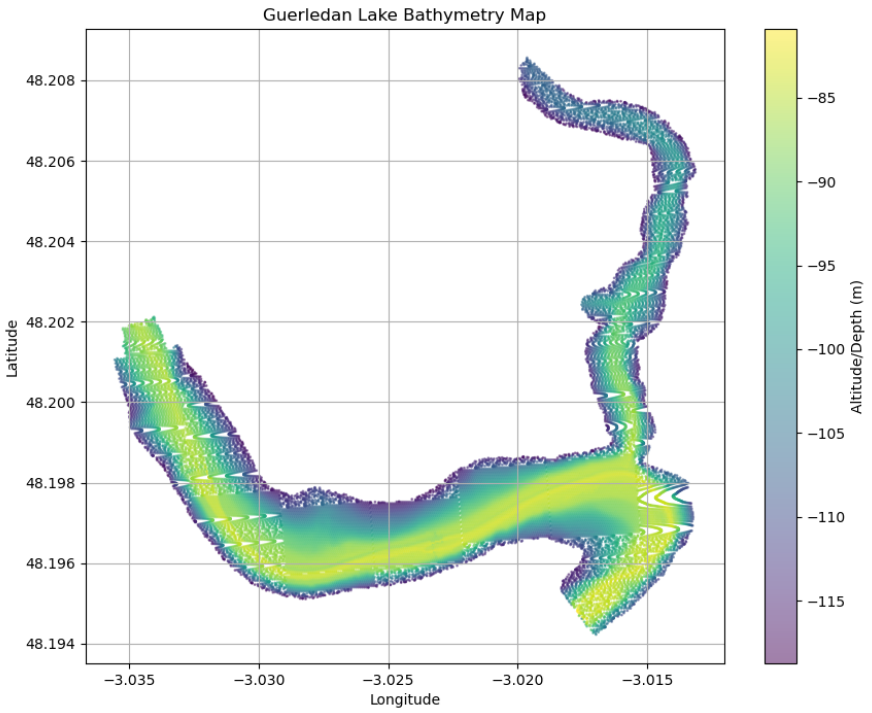
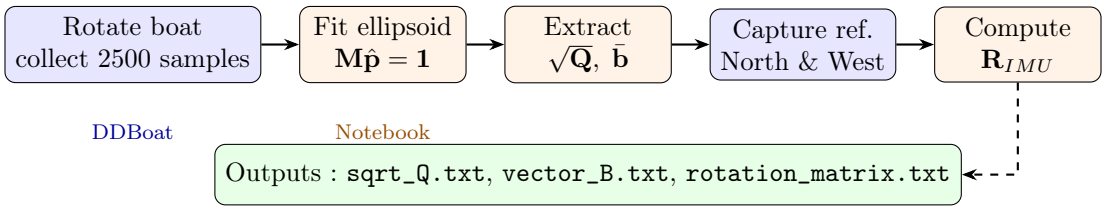


FIGURE 1 – Bathymetric map of Guerledan Lake — the environment used for the DDBoat challenges.

The magnetometer calibration transforms raw sensor data from an ellipsoidal distribution to a unit sphere centered at the origin. The process consists of five sequential steps, illustrated in the flowchart below.

1.1 Calibration Pipeline



Blue = DDBoat, Orange = Notebook post-processing.

1.2 Ellipsoid Fitting and Calibration

Raw magnetometer data $\mathbf{x} \in \mathbb{R}^3$ lies on an ellipsoid due to hard/soft iron distortions. We fit a general quadric (Eq. 1) via least squares, extract the symmetric matrix $\mathbf{Q}$ and

center $\bar{\mathbf{b}}$, then apply the calibration transformation :

$$(x - \bar{x})^T \mathbf{Q}(x - \bar{x}) = 1, \quad \bar{\mathbf{b}} = -\tfrac{1}{2}\mathbf{Q}^{-1}\mathbf{C} \tag{1}$$

$$\boxed{\hat{\mathbf{y}} = \sqrt{\mathbf{Q}} \cdot (\mathbf{x} - \bar{\mathbf{b}})} \tag{2}$$

where $\sqrt{\mathbf{Q}}$ is the matrix square root. This maps the ellipsoid to a unit sphere ($\hat{\mathbf{y}}^T\hat{\mathbf{y}} = 1$).

Implementation : `calibration/sphere_correction.py` (L.30–55) and `calibration/get_mag_data.py`.

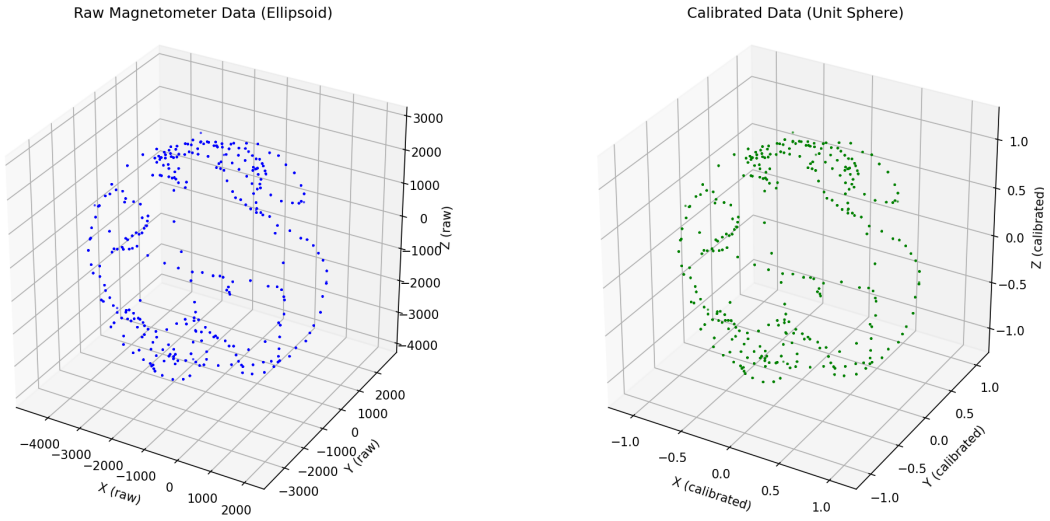


FIGURE 2 – Left : Raw magnetometer data (ellipsoid). Right : Calibrated data (unit sphere).

1.3 Rotation Matrix and Heading

With calibration parameters on the DDBoat, we capture calibrated readings pointing North ($\bar{\mathbf{y}}_N$) and West ($\bar{\mathbf{y}}_W$) using `calibration/capture_polar_data.py`. The Sensor-to-Body rotation matrix is :

$$\boxed{\mathbf{R}_{IMU} = \mathbf{V} \cdot \mathbf{Y}^{-1}} \tag{3}$$

where $\mathbf{V} = [\mathbf{V}_N \ \mathbf{V}_W \ \mathbf{V}_N \times \mathbf{V}_W]$ are the theoretical field vectors for France ($I \approx 64$) and $\mathbf{Y} = [\bar{\mathbf{y}}_N \ \bar{\mathbf{y}}_W \ \bar{\mathbf{y}}_N \times \bar{\mathbf{y}}_W]$ are the measured references.

Implementation : `calibration/compute_rotation_matrix.py`.

The complete heading pipeline applied in real-time is :

$$\mathbf{x} \xrightarrow{\sqrt{\mathbf{Q}}(\cdot - \bar{\mathbf{b}})} \hat{\mathbf{y}} \xrightarrow{\mathbf{R}_{IMU}} \mathbf{Y}_{cor} \xrightarrow{\text{project horiz.}} \boxed{\psi = -\arctan 2(R_{1,0},\, R_{0,0})}$$

Implementation : Function `get_psi_angle()` in `navigation/utils.py` (L.70–91) and `navigation/ddboat.py` (L.160–178).

2 Line Following and Heading Control

This section presents the navigation algorithms deployed on the DDBoat : heading control (magnetometer-only) and GPS-based line following. Both build upon the calibrated magnetometer from Section 1.

2.1 Heading Following (Suivi de Cap)

Given a target heading $\bar{\theta}$, the boat maintains course using the calibrated heading $\theta(t)$. The control law uses the angular correction :

$$\boxed{u_1 = K_1 \cdot \text{sawtooth}\big(\bar{\theta} - \theta\big)\,, \quad \text{sawtooth}(\alpha) = ((\alpha + \pi) \bmod 2\pi) - \pi} \tag{4}$$

Applied differentially to the motors :

$$\text{PWM}_L = V_{\text{base}} + u_1, \quad \text{PWM}_R = V_{\text{base}} - u_1, \quad u_1 = \text{clamp}(K_p \cdot \text{sawtooth}(\bar{\theta} - \theta),\ -80,\ 80) \tag{5}$$

with $K_p = 2.0$ and $V_{\text{base}} = 160$. A positive error produces a clockwise turn.

Implementation : `navigation/heading_control.py` (L.60–84). The script executes a two-phase mission : West (-90, 20s) then East ($+90$, 30s).

2.2 Line Following (Suivi de Ligne)

GPS coordinates are projected to UTM (zone 30N). Given waypoints $\mathbf{a}$ and $\mathbf{b}$, and boat position $\mathbf{m}$:

Cross-track error.

$$\boxed{e = \det\begin{pmatrix} \dfrac{\mathbf{b}-\mathbf{a}}{\|\mathbf{b}-\mathbf{a}\|} & \mathbf{m}-\mathbf{a} \end{pmatrix}} \tag{6}$$

Desired heading.

$$\boxed{\bar{\theta} = \varphi - \tanh\!\left(\frac{e}{\varepsilon}\right), \quad \varphi = \arctan 2((\mathbf{b}-\mathbf{a})_y,\ (\mathbf{b}-\mathbf{a})_x)} \tag{7}$$

In our implementation : $\bar{\theta} = \varphi - 0.1 \cdot \tanh(e/10)$, limiting the approach angle to ± 5.7 with transition distance $\varepsilon = 10$ m. The output is converted from mathematical to navigation convention : $\psi_{\text{ref}} = \pi/2 - \bar{\theta}$.

Implementation : `navigation/line_following.py` (L.101–111) and `navigation/ddboat.py` (L.231–240).

2.3 Waypoint Switching

The boat advances to the next segment when it passes beyond waypoint $\mathbf{b}$:

$$\boxed{(\mathbf{b}-\mathbf{a})^T \cdot (\mathbf{b}-\mathbf{m}) < 0} \tag{8}$$

Implementation : `stop_condition()` in `navigation/utils.py` (L.128–138).

2.4 Control Architecture

$$\underbrace{\text{GPS}}_{\mathbf{m}} \xrightarrow{\text{cross-track}} \underbrace{e}_{\text{error}} \xrightarrow{\tanh \text{ law}} \underbrace{\bar{\theta}}_{\text{desired}} \xrightarrow{u_1 = K_1 \text{ sawtooth}} \underbrace{u_1}_{\text{correction}} \xrightarrow{\text{diff. drive}} \underbrace{(\text{PWM}_L,\ \text{PWM}_R)}_{\text{motors}} \tag{9}$$

The outer loop (GPS, ∼10 Hz) computes $\bar{\theta}$; the inner loop (magnetometer, ∼100 Hz) tracks it via u_1. This cascaded guidance-and-control separation is classical in mobile robotics.

2.5 Results

As illustrated in Figure 3, the DDBoat successfully navigated the programmed waypoints. The experimental data demonstrates that the implemented control architecture allowed the boat to accurately follow the defined lines and correctly close the complete trajectory defined in the code.

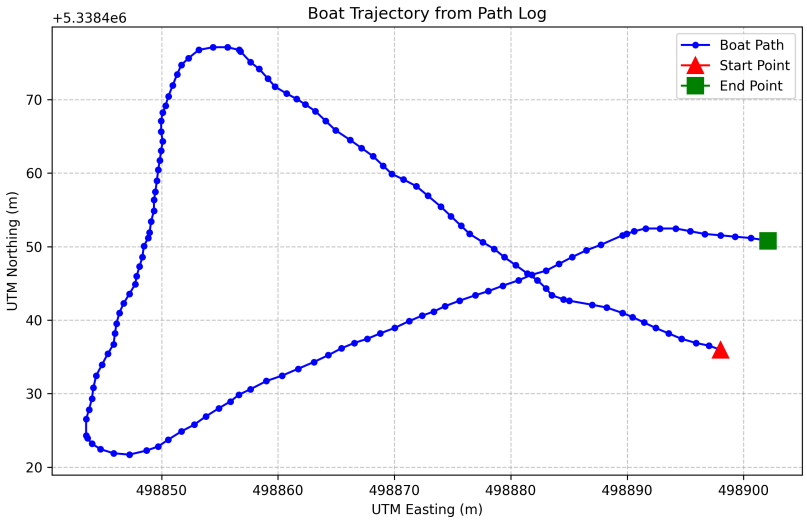


FIGURE 3 – Experimental results of the boat trajectory. The plot shows the successful execution of the line-following algorithm, accurately closing the predefined path.

3 Polygon Mission — Autonomous Waypoint Navigation

This section presents the integration of all components into a complete autonomous polygon mission. The script `navigation/polygon_mission_swarm.py` navigates the DDBoat along a closed polygon defined by GPS waypoints on Guerlédan Lake.

3.1 Mission Setup

All GPS positions are projected into UTM (zone 30N) and referenced to a local origin at the pier :

$$\mathbf{m}_{\text{local}} = \text{UTM}(\text{lat}, \text{lon}) - \text{UTM}(\text{lat}_0, \text{lon}_0) \tag{10}$$

where $(\text{lat}_0, \text{lon}_0) = (48.199240,\ -3.0145649)$.

Waypoint	Latitude	Longitude
Pier	48.199240	−3.014565
P_1	48.199147	−3.015209
P_2	48.199133	−3.015763
P_3	48.199250	−3.015703
P_4	48.199035	−3.015277

TABLE 1 – GPS waypoints for the polygon mission.

The boat navigates the sequence Pier $\to P_1 \to P_2 \to P_3 \to P_4 \to P_1 \to \cdots \to$ Pier, performing two complete laps of the polygon.

3.2 Control Loop

The main control loop runs at GPS update frequency (∼10 Hz). At each iteration :

1. **GPS Update** : Read position **m** via NMEA GLL messages.
2. **Waypoint Check** : Apply the dot product switching condition (Eq. 8).
3. **Heading** : Compute $\bar{\theta}$ via cross-track error law (Eq. 7).
4. **Motor Command** : Apply proportional control $u_1 = K_p \cdot \text{sawtooth}(\bar{\theta} - \theta)$ with $K_p = 2.0$.

All telemetry (position, heading, motor commands, segment index) is logged via the `DataLogger` class to timestamped CSV files.

Implementation : `navigation/polygon_mission_swarm.py` (main loop L.40–122), using utility functions from `navigation/utils.py` and `navigation/data_logger.py`.

3.3 Triangle Trajectory Analysis

In addition to the standard multi-point polygon, a specific triangular mission was executed to evaluate the control system's responsiveness to sharp waypoint transitions. Figure 4 presents a comprehensive telemetry dashboard captured during this phase.

The four subplots provide a detailed view of the cascaded control loop's behavior throughout the mission :

— **Path Trajectory (X vs Y) :** The boat successfully completes the triangular circuit. The plot demonstrates steady convergence to the desired path during straight segments and distinct turning maneuvers at the vertices.

— **Heading Evolution :** The target heading (dashed orange line) updates sharply when the switching condition (Eq. 8) is met at each waypoint (notably around $t \approx 60\,\text{s}$ and $t \approx 105\,\text{s}$). The actual heading (solid blue line) tracks these sudden step changes effectively.

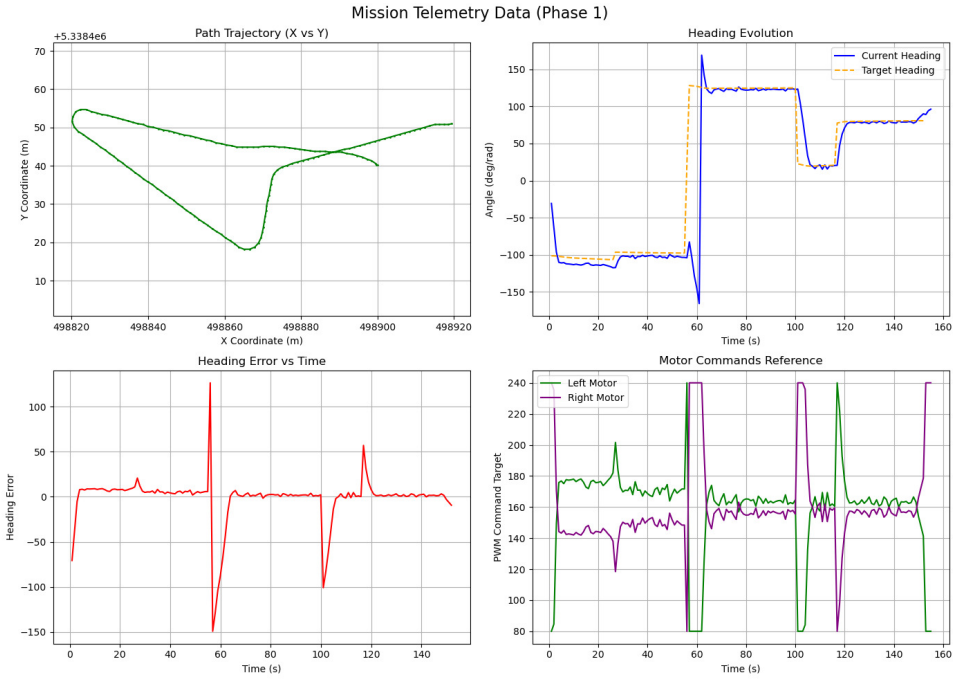


FIGURE 4 – Mission telemetry data for the triangular trajectory. Top-left : Spatial path trajectory (X vs Y). Top-right : Evolution of current heading versus target heading. Bottom-left : Heading error over time. Bottom-right : Left and right motor PWM commands over time.

— **Heading Error vs Time :** As expected, the heading error spikes significantly at the waypoint transitions due to the abrupt change in the target heading. However, the error rapidly converges back to near zero, validating the responsiveness of the proportional controller.

— **Motor Commands Reference :** The differential drive mechanism is clearly illustrated here. During straight-line navigation, both motors hover around the base PWM value. When a sharp turn is required (e.g., at $t \approx 60\,\text{s}$), the control law (Eq. 5) drives the motors in opposite directions—saturating at the lower bound of 80 and the upper bound of 240—to execute a rapid rotation before stabilizing again.

3.4 Swarm Challenge — Multi-Boat Communication

The final challenge extends the polygon mission to a **cooperative swarm scenario** : multiple DDBoats navigate the same polygon simultaneously, maintaining equal phase spacing around the perimeter. Each boat must follow the boat ahead of it, adjusting its speed based on the leader's position received over the network.

3.4.1 Communication Protocol

The `Communication` class (`navigation/comm_module.py`) implements a peer-to-peer architecture using two protocols :

— **TCP Server/Client** : Each DDBoat runs a TCP server (port 29200) broadcasting its position (x, y) and current phase ϕ to all connected peers at 1 Hz.

Simultaneously, it connects as a TCP client to every other DDBoat to receive their data. The data format is `x|y|phi`.

— **UDP Log** : Each boat sends its position and segment index to a central log server (`172.20.254.254:6969`) via UDP for mission-wide monitoring.

The communication runs in background threads (daemon), allowing the control loop to remain at ∼100 Hz without blocking.

Implementation : `navigation/comm_module.py` (L.1–278).

3.4.2 Phase-Based Speed Control

Each robot i is assigned a target phase offset $\phi_i = 2\pi \cdot i/N$, where N is the number of robots. The phase of each robot is computed from its distance traveled along the polygon :

$$\theta_{\text{actual}} = \frac{2\pi \cdot d_{\text{traveled}}}{\text{perimeter}} \tag{11}$$

The phase error is computed relative to a reference boat (DDBoat 16) :

$$\boxed{e_\phi = \theta_{\text{peer}} - \theta_{\text{actual}} - \frac{2\pi}{N}} \tag{12}$$

where θ_{peer} is obtained from the TCP data of the leader boat. The speed is then regulated with a proportional controller :

$$V_{\text{target}} = \text{clamp}(V_{\text{base}} + K_v \cdot e_\phi,\ 80,\ 160)\,, \quad K_v = 0.1 \tag{13}$$

This replaces the constant V_{base} in the motor command : $\text{PWM}_{L,R} = V_{\text{target}} \pm u_1$.

Implementation : `navigation/polygon_mission_swarm.py` (L.19–122).

4 Conclusion

This report presented the complete autonomous navigation pipeline developed for the DDBoat during the Guerlédan 2025 campaign. Three main contributions were achieved :

1. **Magnetometer Calibration** : A robust ellipsoid-to-sphere calibration was implemented, transforming raw magnetometer data into reliable heading estimates. The pipeline includes ellipsoid fitting, reference capture, and Sensor-to-Body rotation matrix computation.
2. **Line Following** : A cascaded control architecture combining a GPS-based cross-track error steering law (outer loop) with a magnetometer-based heading controller (inner loop) was deployed. This guidance-and-control separation proved effective for autonomous navigation between waypoints.
3. **Polygon Mission** : All components were integrated into a complete autonomous mission, where the DDBoat successfully navigated a closed polygon on the lake. Logged telemetry confirmed convergence of the cross-track error and stable heading tracking throughout the mission.

Rapport Projet Guerlédan

FISE27

22 février 2026

Florian Milin
florian.milin@ensta.fr

Guilhem Merlet
guilhem.merlet@ensta.fr

Juliette Lourdault
juliette.lourdault@ensta.fr

Table des matières

1 Introduction

Ce rapport fait suite à celui de la première semaine de Guerlédan en octobre 2026. Lors du précédent Guerlédan, nous avions calibré l'IMU afin d'obtenir la matrice d'Euler du bateau. Nous avions ensuite implémenté un algorithme de suivi d'un point GPS. L'objectif de ce second Guerlédan est de suivre un polygone et de communiquer avec les autres bateaux afin de maintenir un espacement optimal entre chacun d'eux sur ce polygone. La première étape à notre arrivée à Guerlédan consiste à recalibrer le bateau en utilisant les codes précédemment développés, car l'IMU a potentiellement été déplacée.

2 Le suivi de ligne

Le suivi de ligne demande comme au premier Guerlédan, de rejoindre une coordonée GPS mais en suivant le plus possible une droite comportant le point GPS objectif et un autre point.

2.1 Théorie

L'objectif est de construire un champ vectoriel générant une consigne de cap $\bar{\theta}$ permettant au bateau de converger vers une droite de référence définie par deux points a et b, où b représente l'objectif à atteindre. Ce champ est conçu de telle sorte que lorsque le bateau est éloigné de la droite, la consigne de cap tend vers un angle orthogonal à celle-ci, et que plus le bateau se rapproche de la droite, plus la consigne de cap s'aligne avec la direction de la ligne.

On note m la position GPS du bateau, exprimée dans un repère local bidimensionnel. On définit :

$$\begin{aligned} u &= \frac{b-a}{\|b-a\|}, \\ e &= \det(u, m-a), \\ \psi &= angle(b-a) \end{aligned}$$

La consigne de cap est alors donnée par :

$$\bar{\theta} = \psi - \tanh\Big(\frac{e}{k}\Big)$$

où k est un paramètre réglé expérimentalement.

Comme lors du précédent Guerlédant, le contrôle du bateau est assuré par deux commandes :

$$\begin{cases} u_1 = k_1 \,\text{sawtooth}(\bar{\theta} - \psi) \\ u_2 = k_2 \,\tanh(\bar{v} - v) \end{cases}$$

où u_1 est la commande angulaire, u_2 la commande en vitesse, ψ et v sont respectivement le cap et la vitesse du bateau, $\bar{v}$ la vitesse de consigne, et k_1, k_2 des constantes de gain.

2.2 Implémentation

Les essais réalisés ont montré un comportement conforme aux attentes théoriques. Le bateau converge de manière fluide vers la droite de référence, sans oscillations notables ni instabilités.

La transition entre phase de rapprochement et phase d'alignement s'effectue naturellement grâce au choix de la fonction hyperbolique tangente. Les gains sélectionnés ont permis d'obtenir un compromis satisfaisant entre rapidité de convergence et stabilité.

Globalement, l'implémentation s'est déroulée sans difficulté majeure et les performances observées en conditions réelles confirment la pertinence du champ vectoriel choisi.

3 Polygone

3.1 Théorie

Une fois le suivi de ligne implémenté, l'objectif est de permettre au bateau de suivre un polygone composé de n segments. Le principe consiste à appliquer successivement l'algorithme de suivi de ligne sur chaque segment reliant deux sommets consécutifs du polygone. Lorsque le bateau atteint la fin d'un segment, la référence est mise à jour afin de suivre le segment suivant.

La condition de fin de segment, exprimée avec les notations précédentes, est donnée par :

$$(b-a)^\top (b-m) < 0$$

Cette condition correspond à un critère de dépassement du point b : lorsque le produit scalaire devient négatif, le bateau a franchi l'extrémité du segment $[a, b]$, ce qui déclenche le passage au segment suivant du polygone.

3.2 Implémetation

Le premier jour, lorsque nous avons implémenté le triangle, tout semblait fonctionné à merveille. Cependant, le lendemain en comparant les trajectoires sur le lac avec les autres bateaux, nous avons remarqués que nous ne suivions pas le même triangle. En supperposants les log du bateau avec la trajectoire théorique, nous nous sommes rendu compte que le bateau ne suivait pas la bonne ligne. L'origine de l'erreur était liée au calcul de l'angle ψ : nous utilisions la fonction `atan2` pour déterminer l'orientation de la droite, mais cette fonction ne définit pas l'axe 0 radian dans la direction du nord.

Ainsi, l'angle de référence était exprimé dans un repère différent de celui utilisé par le système de navigation du bateau, ce qui introduisait un décalage systématique dans l'orientation des segments du polygone, et donc dans la trajectoire suivie.

Après correction de ce décalage angulaire afin d'aligner la référence 0 radian avec le nord, les trajectoires expérimentales ont correctement coïncidé avec les trajectoires théoriques.

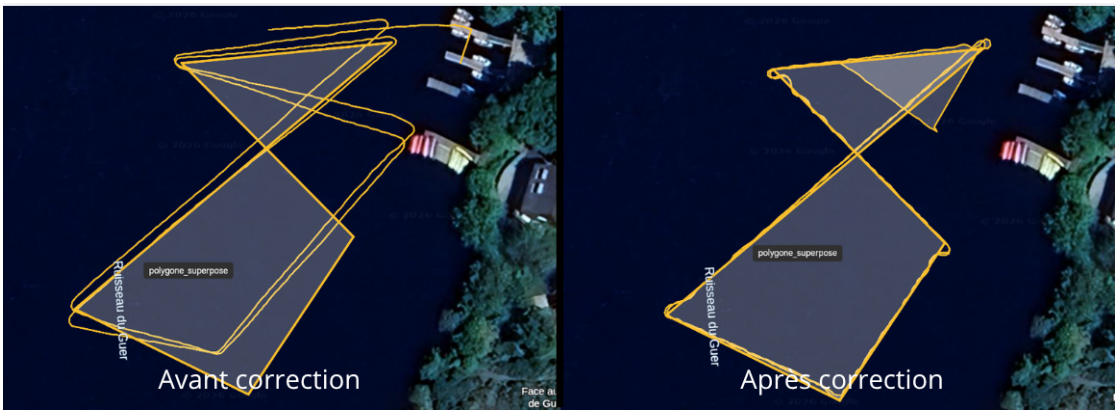
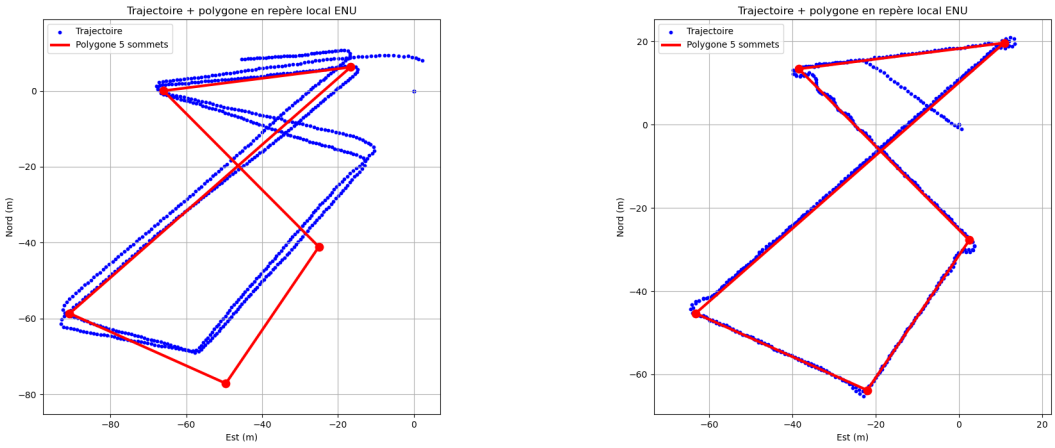


FIGURE 1 – Comparaison des résultats obtenues avant et après correction sur google earth



(a) Avant correction

(b) Après correction

FIGURE 2 – Comparaison des résultats obtenues avant et après correction sur python

4 Consensus

Le consensus consiste à faire suivre le polygone décrit dans la section précédente en coordination avec les autres bateaux. Pour ce faire, chaque bateau applique un contrôleur sur sa vitesse et communique sa position aux autres bateaux afin qu'ils puissent également ajuster leur vitesse pour rester correctement espacés le long du polygone.

4.1 Théorie

On considère un groupe de N bateaux numérotés de 0 à $N-1$. Chaque bateau i possède sa propre commande de vitesse nominale $\bar{u_{2i}}$, adaptée aux caractéristiques spécifiques de ses moteurs.

Pour gérer la progression sur le polygone, on définit une fonction bijective $\mathcal{L}$ qui transforme le polygone en un cercle trigonométrique : si l représente la distance parcourue depuis le premier sommet jusqu'au point projeté p sur le polygone, et L le périmètre total, alors

$$\phi_i = \mathcal{L}(p) = \frac{2\pi l}{L}.$$

Le point p correspond au projeté orthogonal de la position actuelle m du bateau sur le segment actif du polygone. Chaque bateau communique sa phase ϕ_i au bateau suivant, et reçoit la phase ϕ_{i+1} du bateau qui le précède.

La commande finale de vitesse est ajustée pour maintenir une répartition uniforme des bateaux sur le polygone :

$$u_{2i} = \bar{u_{2i}} + \alpha \,\mathrm{sawtooth}\Big(\phi_{i+1} - \phi_i - \frac{2\pi}{N}\Big),$$

où α est un gain permettant de réguler la vitesse de réaction du bateau, c'est-à-dire l'intensité de son ralentissement ou accélération pour rester à distance constante du bateau précédent.

4.2 Résultat

Les expérimentations menées lors du quatrième et cinquième jours ont permis de valider le fonctionnement du protocole de consensus sur le grand polygone. Nous avons réalisé les essais

avec trois autres bateaux, soit un total de quatre unités évoluant simultanément sur la trajectoire définie.

Au cours des différentes phases de test, les bateaux ont réussi à atteindre un consensus stable. Après une phase transitoire initiale, correspondant à l'ajustement des phases φ_i, les vitesses se sont progressivement régulées grâce au terme correctif proportionnel au décalage de phase. Les écarts angulaires ont convergé vers la valeur théorique $\frac{2\pi}{N}$, traduisant une répartition homogène des bateaux le long du polygone.

On observe que :

— Les vitesses se sont adaptées dynamiquement en fonction des positions relatives.
— Les accélérations et ralentissements sont restés progressifs, sans oscillations importantes.
— La stabilité du système a été maintenue même lors de légères perturbations (variations de trajectoire ou imprécisions de projection sur le segment actif).

Par ailleurs, l'ensemble des données expérimentales (positions, vitesses, phases et commandes appliquées) a été enregistré puis transmis au serveur.

Enfin, les essais de consensus ont été filmés à l'aide d'un drone, offrant une vue globale du comportement collectif. Les images confirment visuellement la bonne répartition spatiale des bateaux sur le polygone et la cohérence du mouvement d'ensemble.

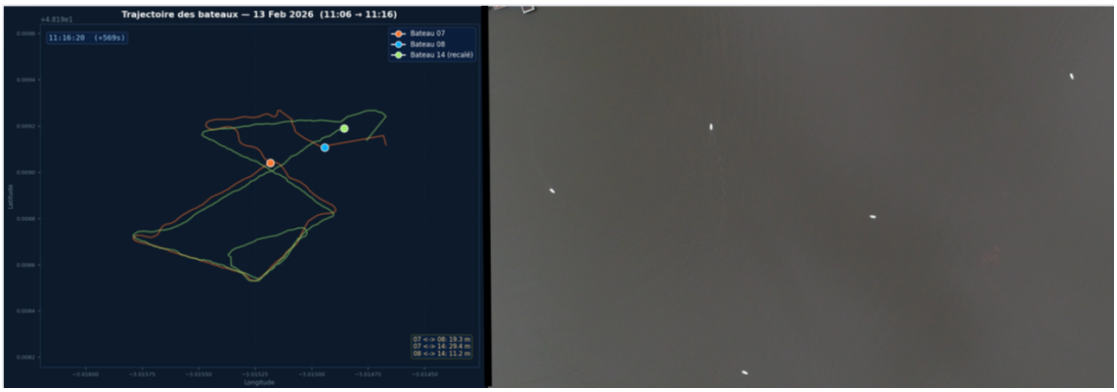


FIGURE 3 – Graphe et photo du consensus

Lien de la vidéo en drone

5 Conclusion

Ce projet Guerlédan nous a permis de mettre en pratique l'ensemble des notions de contrôle et de navigation acquises précédemment. Après un recalibrage initial de l'IMU, nous avons implémenté avec succès un suivi de ligne, puis étendu cette approche pour permettre le suivi d'un polygone complexe.

Les expérimentations ont mis en évidence l'importance de la cohérence des repères pour la navigation : un simple décalage angulaire pouvait entraîner des trajectoires déviées. La correction de ce décalage a permis d'obtenir un suivi précis et conforme aux trajectoires théoriques.

Enfin, la mise en œuvre du protocole de consensus a démontré que plusieurs bateaux pouvaient se coordonner efficacement sur un polygone, en maintenant un espacement régulier grâce à des ajustements de vitesse synchronisés. Les résultats expérimentaux et les observations par drone confirment la robustesse et la stabilité de ce contrôle collectif.

Ces travaux ouvrent la voie à des développements futurs, notamment l'extension à un plus grand nombre de bateaux, la prise en compte de perturbations environnementales plus importantes, ou encore l'intégration de stratégies adaptatives pour optimiser les trajectoires en temps réel.

Table des figures

DDBoat (Guerlédan)

Kadijatou DIALLO - **kadijatou.diallo@ensta.fr**

Maël MABILOTTE - **mael.mabilotte@ensta.fr**

Tom VOISINE - **tom.voisine@ensta.fr**

FISE27

Février 2025

Figure 1 – Image du Lac de Guerlédan

Table des Matières

1 Introduction

Du 9 au 14 février 2026 s'est tenue la deuxième session de travail sur les DDBoat au lac de Guerlédan. Cette semaine s'inscrivait dans la continuité directe de la session de novembre 2025, dont les principaux acquis — calibration de l'IMU, suivi de cap, projection de points GPS dans un repère cartésien local et navigation vers un waypoint — ont été repris, consolidés et étendus. L'objectif de cette nouvelle session était de dépasser les missions élémentaires pour mettre en œuvre des stratégies de navigation plus ambitieuses, impliquant le suivi de trajectoires géométriques complexes et la coordination entre plusieurs bateaux. La calibration du DDBoat a ainsi été réalisée de nouveau afin de garantir la fiabilité des mesures de cap. Nous avons ensuite implémenté une méthode de suivi de ligne définie par deux points GPS, première étape vers le suivi d'un polygone quelconque dont les sommets sont spécifiés en coordonnées géographiques. Une fois cette brique validée, nous avons introduit une régulation en vitesse basée sur le suivi d'une phase théorique définie sur le cercle trigonométrique. Cette phase était ensuite traduite en une position cible évoluant le long du polygone réel. Enfin, un mécanisme de communication inter-bateaux a été mis en place afin que chaque unité partage sa phase et qu'un consensus émerge naturellement sur la répartition des positions le long de la trajectoire. Cette approche distribuée constitue une première mise en œuvre d'un comportement de type swarm. Le présent rapport détaille les missions réalisées ainsi que les choix méthodologiques adoptés.

2 Calibration

Avant toute mission, une calibration du magnétomètre trois axes embarqué sur le DDBoat est indispensable. En effet, la présence de composants ferromagnétiques dans la structure du bateau perturbe les mesures du champ magnétique terrestre. Ce phénomène, appelé effet soft iron, déforme la sphère théorique des mesures en une ellipsoïde. Sans correction, l'estimation du cap est donc biaisée et instable. La procédure de calibration consiste à identifier cette ellipsoïde à partir d'un ensemble de mesures réalisées en faisant tourner le bateau dans toutes les orientations possibles. Une méthode d'estimation par moindres carrés permet de déterminer les paramètres décrivant cette déformation. Une fois ces paramètres identifiés, les mesures sont d'abord recentrées afin de corriger le biais (déplacement du centre de l'ellipsoïde), puis transformées par une opération linéaire permettant de « sphériser » les données. L'objectif est d'obtenir une distribution uniforme et centrée, équivalente à celle d'un capteur idéal non perturbé. Voici les données brutes (ellipsoïde) ainsi que les données sphérisées :

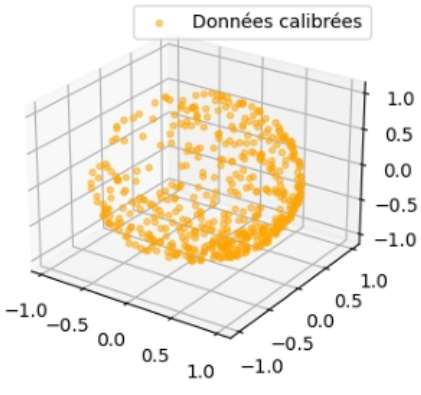


Figure 2 – Ellipse

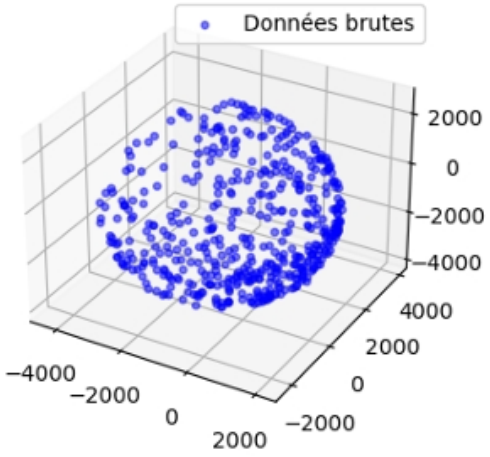


Figure 3 – Sphère

Enfin, une rotation supplémentaire tenant compte de l'inclinaison magnétique locale a été appliquée afin d'aligner les mesures avec le repère terrestre. Cette étape améliore la cohérence des angles estimés et garantit une utilisation fiable du cap dans les algorithmes de contrôle.

3 Suivi de Polygone

3.1 Suivi de ligne

L'objectif principal du guidage pour un robot DDBOAT consiste à asservir sa trajectoire sur un segment défini par deux points, a et b. Pour ce faire, nous définissons un champ de vecteurs attractif qui oriente le robot vers la ligne tout en maintenant une progression parallèlement à celle-ci. Le point m représente la position instantanée du robot dans le référentiel global.

La première étape consiste à quantifier l'écart latéral, noté e, entre le robot et la cible. Cette erreur de suivi est calculée à l'aide d'un déterminant normalisé, exprimé par la formule :

$$e = \det\left(\frac{b-a}{\|b-a\|}, m-a\right) \tag{1}$$

Cette valeur e représente la distance signée du robot par rapport à la droite (ab), permettant de déterminer si le DDBOAT se situe à gauche ou à droite du segment. Parallèlement, nous définissons φ comme l'angle d'orientation de la ligne à suivre (l'angle du vecteur $b-a$) par rapport à un axe de référence, tel que le Nord ou l'Ouest.

Afin de converger de manière fluide vers la trajectoire idéale sans provoquer de variations brusques de cap, nous utilisons une fonction de commande basée sur la tangente hyperbolique. Le cap de consigne, noté $\bar{\theta}$, que le bateau doit suivre est alors défini par l'équation suivante :

$$\bar{\theta} = \varphi - \tanh\left(\frac{e}{10}\right) \tag{2}$$

Ici, le facteur 10 agit comme un paramètre de lissage, limitant l'angle d'attaque pour assurer une approche asymptotique à environ ± 10 mètres de la ligne. Cette méthode garantit que plus l'erreur e est importante, plus le robot braque vers la ligne, tout en se stabilisant sur l'angle φ à mesure que e tend vers zéro.

3.2 Suivi de Polygone

Une fois le suivi de ligne défini par deux points GPS validé, nous avons étendu cette méthode au suivi d'un polygone quelconque. Le principe repose sur l'enchaînement successif de segments. Chaque segment est défini par deux points GPS consécutifs, préalablement projetés dans le repère cartésien local. Le bateau suit la droite orientée reliant le point A au point B à l'aide du correcteur de cap développé précédemment. La difficulté principale réside dans la détection du franchissement du waypoint, c'est-à-dire du passage d'un segment au suivant. Pour cela, on considère la position courante du bateau M dans le repère local et on évalue le signe du déterminant : det(AB, AM) ou, de manière équivalente, le produit vectoriel en dimension 2. Ce test permet de déterminer si le bateau a franchi le demi-plan orthogonal au segment en B. Lorsque le signe indique que le point B a été dépassé dans le sens de parcours, le segment suivant est activé. Les essais expérimentaux ont permis de valider cette approche, notamment lors du suivi d'un triangle.

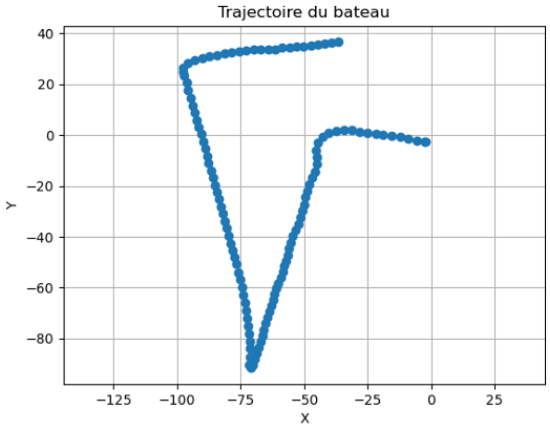


FIGURE 4 – Trajectoire du bateau sur le triangle

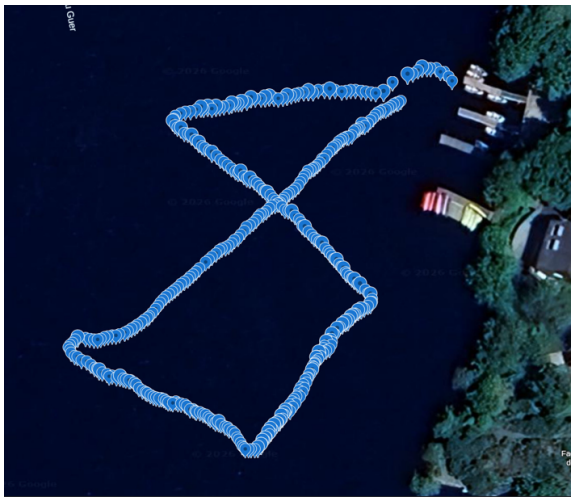


FIGURE 5 – Trajectoire Polygone

La transition entre segments s'est révélée robuste malgré la présence de vent et de courant. La logique mise en place est générique et s'applique à tout polygone défini par une liste ordonnée de points GPS.

4 SWARM

4.1 Suivi de phase et Régulation de vitesse

Après avoir réussi à faire suivre au DDBOAT un polygone à vitesse constante, nous avons cherché à réguler sa vitesse afin qu'il suive un point en translation sur le polygone. Il fallait pour cela que le DDBOAT soit capable d'accélérer pour rejoindre ce point puis maintenir une vitesse égale à la vitesse du point sur la suite du polygone malgré le vent et le courant. Dans un premier temps nous avons implémenté un correcteur proportionnel à la distance entre le bateau et le point à suivre. Cela fonctionnait tant que l'erreur était petite mais lorsque celle-ci devenait importante la correction atteignait les limites de vitesse des propulseurs et il n'était plus possible pour le bateau de tourner. En effet lorsque les deux propulseurs étaient à leur vitesse maximale pour tenter de s'approcher du point à suivre, le bateau ne pouvait plus utiliser le différentiel de vitesse entre les propulseurs pour tourner. Cela avait pour résultat de l'éloigner toujours plus de la cible. Pour remédier à cela nous avons borné le correcteur en vitesse pour qu'il n'écrase pas le correcteur angulaire.

4.2 Communication

Afin d'effectuer un test de consensus nous avons travaillé avec les groupes des bateaux 7, 9 et 14. Pour la communication nous avons choisi un protocole TCP en peer to peer. Nous nous sommes mis d'accord sur un ordre de "poursuite" sur le polygone et chaque bateau envoyait sa phase sur le cercle trigonométrique au bateau qui le suivait. Ainsi le bateau suiveur pouvait calculer la position à rejoindre en soustrayant 2*pi/N (avec N=4) à la phase reçue. C'était très simple et peu gourmand en énergie pour les bateaux et en bande passante pour le réseau. Chaque bateau avait besoin de deux threads, l'un pour envoyer sa position et l'autre pour écouter la position de celui de devant. Cela a fonctionné, nous avons pu réaliser à deux reprises un consensus sur un polygone.

Pour suivre la position de tous les bateaux simultanément nous avons aussi communiqué avec un serveur. Sur celui-ci tous les bateaux envoyaient en UDP leur position, latitude, longitude et numéro du

segment sur le polygone. L'UDP a permis de réduire la quantité d'informations échangées par rapport à la possibilité de le faire en TCP.

5 Conclusion

Cette seconde semaine au lac de Guerlédan a permis de consolider les acquis fondamentaux de la première session et de franchir une étape supplémentaire vers des comportements collectifs structurés. Nous sommes désormais capables non seulement de calibrer précisément les capteurs et de suivre des trajectoires géométriques complexes, mais également d'introduire une dimension dynamique via la régulation en vitesse et le suivi de phase. L'ajout d'une communication inter-bateaux a permis de mettre en œuvre une stratégie distribuée simple et efficace de type consensus, ouvrant la voie à des missions collaboratives plus avancées. Au-delà des résultats techniques, cette semaine a été particulièrement formatrice. Elle nous a confrontés aux contraintes réelles du milieu (vent, courant, limitations matérielles) et nous a amenés à adapter en permanence nos algorithmes aux conditions expérimentales. Cette interaction constante entre théorie, implémentation et validation sur l'eau constitue un apport essentiel dans notre formation en robotique autonome.

Guerlédan 2 : DDBoatthebest

FISE27

Février 2026

Elodie Conan elodie.conan@ensta.fr
Mahé Lafarge mahe.lafarge@ensta.fr
Oscar Verilhac oscar.verilhac@ensta.fr

Table des matières

1 Introduction

L'objectif de cette deuxième semaine sur le lac de Guerlédan était de contrôler les bateaux en essaim à l'aide d'une communication entre eux. La mission finale consistait à se déplacer sur un polygone avec pour objectif de suivre à une distance constante le bateau de devant qui lui-même suivait le bateau devant lui et ainsi de suite de sorte qu'il n'y ait aucun bateau leader. Enfin, l'ensemble des bateaux sur le polygone devaient être équirépartis sur le périmètre. Les codes que nous avons utilisés pendant cette semaine ainsi que les logs importants sont sur le GitHub suivant : GitHubDDBoatthebest.

2 Calibration

Comme les DDBoats ont été révisés avant cette semaine nous avons commencé notre travail par recalibrer son IMU. Les données utilisées pour la calibration sont dans le fichier *mag_raw2.txt* et la figure suivante a été tracée avec le programme *plot_mag_3D.py*.

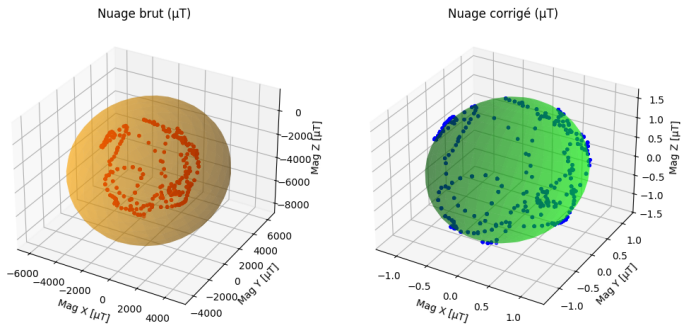


FIGURE 1 – À gauche les données du magnétomètre avant calibration, et à droite après calibration.

Ensuite, nous avons vérifié le bon fonctionnement de nos codes de la première semaine de Guerlédan, notamment le code permettant d'aller à une balise et de revenir au ponton.

3 Contrôle du bateau

Ces fonctions de contrôle permettent de déterminer les commandes moteurs u_1 et u_2.

3.1 Contrôle du cap

Pour le contrôle du cap nous avons choisi la fonction :

$$u_1 = k_{cap} \tanh\left(\frac{sawtooth(\bar{\theta} - \theta)}{k_e}\right)$$

Avec, $\bar{\theta}$ le cap objectif, θ le cap du bateau, k_{cap} le gain proportionnel et k_e un gain. La fonction *sawtooth* permet de prendre le plus petit arc de cercle entre ces 2 angles pour tourner du côté le plus court pour atteindre le cap objectif. Ensuite, la tangente hyperbolique permet d'adapter le comportement pour une petite ou une grande erreur de cap.

3.2 Contrôle de la vitesse

Pour le contrôle de la vitesse nous avons choisi la fonction :

$$u_2 = k_{vit} \tanh\left(\frac{\bar{\phi} - \phi}{k_v}\right)$$

Avec, $\bar{\phi}$ la phase objectif, ϕ la phase du bateau, k_{vit} le gain proportionnel et k_v un gain.

4 Commande des moteurs

À partir des deux commandes u_1 et u_2 calculées précédemment on peut déterminer les valeurs à appliquer au moteur droit et gauche ($\in [0; 255]$) pour diriger le bateau.

$$v_{gauche} = v_{base} + u_1 + u_2 - offset$$

$$v_{droite} = v_{base} - u_1 + u_2 + offset$$

La quantité $offset$ permet de compenser un déséquilibre de puissance entre le propulseur gauche et droit pour que le bateau puisse aller droit lorsque la même commande est donnée aux 2 moteurs.

5 Suivi de ligne

Notre première mission de la semaine a été d'implémenter un suivi de ligne. En novembre, nous avions fait du suivi de cap. On calculait le cap à suivre pour aller de notre position GPS au point où l'on voulait se rendre.

Pour le suivi de ligne, le but est de se rapprocher de la ligne puis de suivre son cap jusqu'à arriver à la fin du segment.

5.1 Théorie

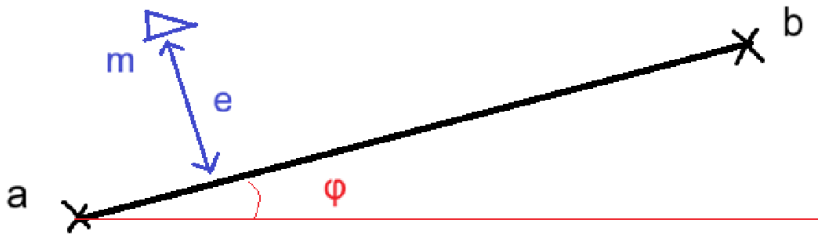


FIGURE 2 – Schéma du principe du suivi de ligne.

5.1.1 Définition du cap à suivre

L'erreur e entre la position du DDBoat et la ligne à suivre est :

$$e = \det\left(\frac{b-a}{\|b-a\|}, m-a\right)$$

Soit $b - a = \begin{pmatrix} x_{ab} \\ y_{ab} \end{pmatrix}$.

L'angle de la ligne par rapport à l'origine du cercle trigonométrique φ est donné par,

$$\varphi = \arctan\left(\frac{y_{ab}}{x_{ab}}\right)$$

Pour se rapprocher de la ligne tout en la suivant le robot doit se diriger selon l'angle donné par,

$$\bar{\theta} = \varphi - \tanh\left(\frac{e}{k_3}\right)$$

Avec k_3 un gain permettant de définir la largeur du couloir dans lequel la correction du cap est progressive. Au-delà de cette distance, la correction est saturée ce qui permet d'éviter de trop grandes corrections. Une fois que le bateau a rejoint sa ligne il reste dans un couloir de k_3 mètres autour de celle-ci. Nous avons choisi $k_3 = 5$.

Enfin, pour obtenir le cap que le bateau doit suivre, il faut convertir cet angle trigonométrique en cap géographique.

$$\bar{\theta}_{geo} = (90 - \bar{\theta})\%360$$

5.1.2 Validation du suivi de ligne

Le suivi de ligne se finit une fois que le bateau a dépassé le point b. C'est-à-dire lorsque le produit scalaire $\langle \overrightarrow{ab}, \overrightarrow{mb} \rangle$ est négatif.

$$(b-a)^{\mathsf{T}}(b-m) < 0$$

5.2 Parcourir un triangle

La première mission de cette semaine consistait à suivre un triangle constitué de 3 points GPS en utilisant le suivi de ligne. Nous avons utilisé le code *SuiviTriangle.py* pour parcourir le triangle et générer la trajectoire suivante.

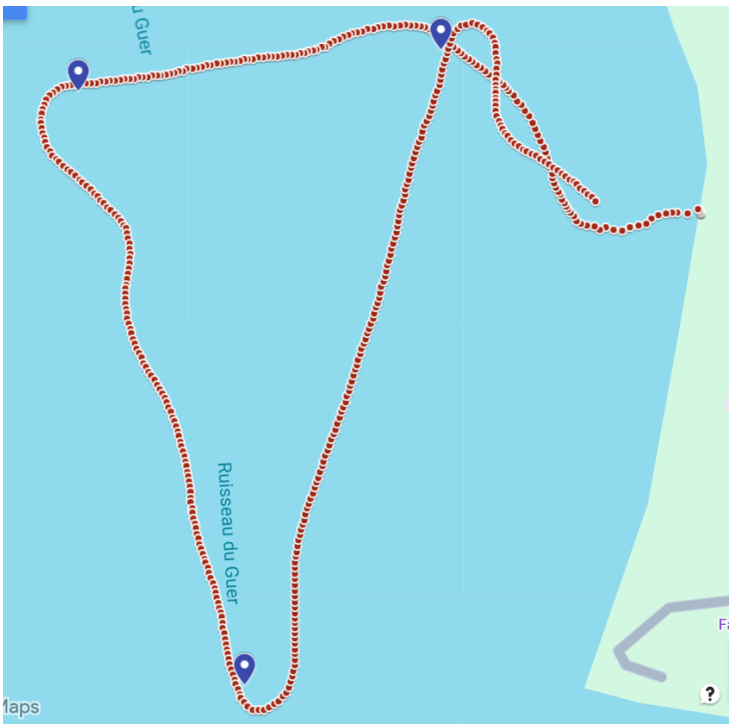


FIGURE 3 – Trajectoire du DDBoat lors de la mission triangle.

On peut remarquer qu'après les sommets il y a des oscillations et que la trajectoire du DDBoat finit bien par converger sur la ligne suivie.

Les logs sont stockés dans les fichiers *gps_ data_ triangle.kml* et *log_ mission_ triangle.txt*.

6 Suivi de phase

Le suivi de phase consiste à définir un point mouvant sur le polygone qu'il faut suivre en adaptant sa vitesse. Pour cela ce point aura pour référence un angle sur le cercle trigonométrique.

6.1 Théorie

Premièrement, il faut créer la bijection reliant les points du polygone et ceux du cercle trigonométrique.

Notons d le périmètre du polygone.
Alors cette bijection est de la forme,

$$\begin{array}{rcl} L : [0,d] & \longrightarrow & [0,2\pi] \\ \phi & \longmapsto & a\phi \end{array}$$

Avec,

$$a = \frac{d}{2\pi}$$

6.2 Réguler sa vitesse sur un polygone

Nous avons déterminé arbitrairement une phase qui évolue en fonction du temps selon la fonction :

$$\bar{\phi} = 2\pi \frac{(t - t_0)}{T}$$

Avec T la durée en secondes pour parcourir le polygone.

Ensuite, il fallait déterminer la phase actuelle de notre DDBoat pour déterminer s'il était en avance ou en retard et adapter notre vitesse.

Notons $\vec{n}$ le vecteur directeur du segment que le DDBoat suit, et d_p la somme des longueurs des segments déjà suivis sur le polygone.

Alors la phase actuelle du DDBoat est,

$$\phi = L^{-1}(\|\overrightarrow{am} - e\vec{n}^{\perp}\| + d_p)$$

Ainsi $\bar{\phi}$ et ϕ vont permettre de déterminer la commande $u2$.

Après plusieurs tests pour ajuster les valeurs des différents gains des contrôleurs de cap et de vitesse nous avons obtenu la trajectoire ci-dessous. Le temps de parcours est de 272 secondes pour une distance parcourue de 299m. Les gains appliqués pour cette figure sont :

$$k_{cap} = 60 \qquad k_{vit} = 30 \qquad k_e = 1.2 \qquad k_3 = 5$$

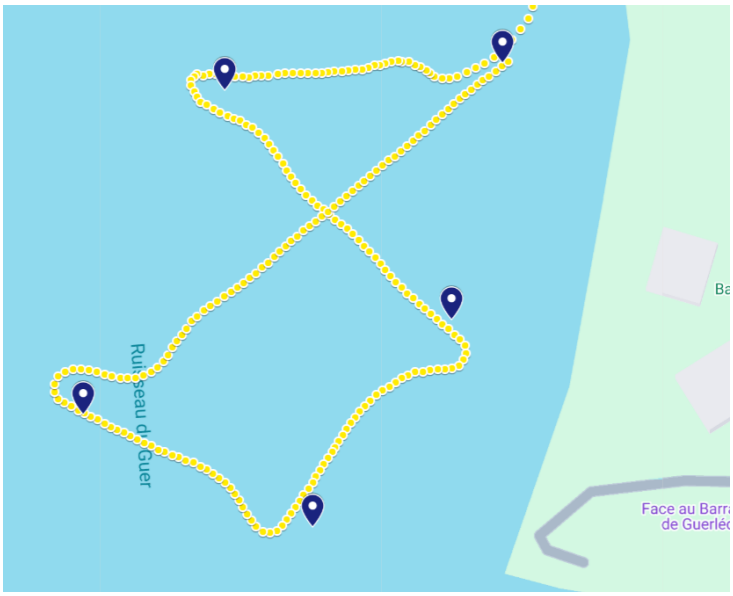


FIGURE 4 – Trajectoire du DDBoat lors de la mission polygone.

Cette mission a été réalisée avec le code *PolygonDanse.py* et les données utilisées pour tracer cette figure sont dans le fichier *gps_ data_ polygone_ 5.kml*.

7 Swarm

Le but final de la semaine est de suivre le polygone avec plusieurs DDBoats qui communiquent et se régulent entre eux pour converger vers une configuration où ils sont équirépartis sur le polygone.

7.1 Théorie

Chaque DDBoat reçoit la phase de celui de devant qu'il doit suivre et il envoie sa phase à celui de derrière qui le suit. Pour cela nous avons choisi de communiquer entre les DDBoats avec un protocole TCP.

Pour être équirépartis sur le polygone la phase objectif de chaque DDBoat doit être à une distance du DDBoat de devant dépendant du nombre N de DDBoat dans l'essaim. Notons ϕ_r la phase reçue, alors la phase objectif $\bar{\phi}$ est telle que,

$$\bar{\phi} = (\phi_r - \frac{2\pi}{N})\%2\pi$$

De plus des données de log sont envoyées au serveur avec un protocole UDP.

7.2 Consensus

Avec 3 autres groupes nous nous sommes accordés pour effectuer le consensus ensemble. Nous avons utilisé le même polygone que celui représenté sur la figure du suivi de phase. Aussi, nous avons choisi de communiquer notre phase en radian. Jeudi en fin d'après-midi, nous avons réussi à avoir 4 DDBoats autorégulés sur le polygone en utilisant le code *PolygonSwarm.py* et le code utilisé pour la communication est *Communication.py*.

Malheureusement, trop enthousiastes de notre réussite nous en avons oublié de récupérer les logs pour les joindre à ce rapport mais la mission a été filmée par le drone.

8 Conclusion

Cette semaine au lac de Guerlédan était très enrichissante. Elle nous a permis de découvrir la robotique en essaim ainsi que de perfectionner le contrôle de notre DDBoat.

Projet deuxième Guerlédan : S.W.A.R.M

ROB27

12 février 2026

Thibault LUCAS
Gaspard DELPIANO–MANFRINI
Matéo BRUN

Sommaire

1 Introduction

Il y a trois mois, nous sommes partis à Guerlédan une semaine avec pour but d'apprendre à calibrer un DDBoat, afin de lui faire suivre une direction cardinale puis enfin atteindre des coordonnées GPS données.

À la fin de cette semaine, nous avions réussi à obtenir une trajectoire "en spirale" qui simulait une capture de bateau, avec des phases données qui ne nécessitaient donc aucune communication entre les DDBoats.

Durant cette nouvelle semaine à Guerlédan, nous avons un nouvel objectif : faire du S.W.A.R.M, c'est-à-dire utiliser la communication entre les bateaux pour qu'ils se positionnent les uns par rapport aux autres à équidistance sur le parcours. De plus, il s'agit désormais de suivre une ligne et nous pas un point ce qui change également la façon de procéder en terme de contrôle.

Une fois de plus, plusieurs problèmes ont considérablement ralenti notre avancée sur le projet, et trois jours ont par exemple été nécessaires pour faire une simple trajectoire de triangle en suivi de ligne.

Lien vers notre gitlab : gitlab.
Lien vers la vidéo d'explication : Video YouTube.

FIGURE 1 – La fine équipe, avec les bateaux 10 et 5. Nous avons travaillé à 3 sur le bateau 10 toute la semaine et nous sommes joints à Mathys BIECHE pour faire les missions de S.W.A.R.M, notre travail collaboratif a beaucoup aidé dans l'avancée du projet

2 Ce qu'on réutilise du premier Guerlédan et calibration

Rappel :

L'équation d'une ellipsoïde s'écrit :

$f(x) = p_1x_1^2 + p_2x_2^2 + p_4x_1x_2 + p_5x_1x_3 + p_6x_2x_3 + p_7x_1 + p_8x_2 + p_9x_3$
$= 1$ (car on veut une sphère normalisée)

Le but est donc d'abord de trouver les paramètres de l'ellipse $\vec{p} = (p_1\ p_2\ p_3\ p_4\ p_5\ p_6\ p_7\ p_8\ p_9)^T$
tels que :

$$\begin{bmatrix} x_1^2(1) & x_2^2(1) & x_3^2(1) & x_1(1)x_2(1) & x_1(1)x_3(1) & x_2(1)x_3(1) & x_1(1) & x_2(1) & x_3(1) \\ x_1^2(2) & x_2^2(2) & x_3^2(2) & x_1(2)x_2(2) & x_1(2)x_3(2) & x_2(2)x_3(2) & x_1(2) & x_2(2) & x_3(2) \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ x_1^2(i) & x_2^2(i) & x_3^2(i) & x_1(i)x_2(i) & x_1(i)x_3(i) & x_2(i)x_3(i) & x_1(i) & x_2(i) & x_3(i) \end{bmatrix} \vec{p} = \begin{bmatrix} 1 \\ 1 \\ . \\ . \\ . \\ 1 \end{bmatrix}$$

$$\vec{p} = (X^TX)^{-1}.X^T. \begin{bmatrix} 1 \\ 1 \\ . \\ . \\ . \\ 1 \end{bmatrix}$$

2.1 Recalage en cap

On réutilise donc quasi totalement le code de calibration du premier Guerlédan, à l'exception de la calibration du Nord qu'on va simplifier légèrement en émettant l'hypothèse que le bateau ne tourne que selon l'axe de la vertical (on néglige les rotations due aux vagues).

En fonction de l'orientation de l'IMU au sein de la structure du bateau il peut y avoir un décalage constant entre le cap du bateau et le cap de l'IMU. Pour pallier ce problème, on cherche à calculer la matrice de rotation correspondant à la rotation de l'IMU dans le bateau. Par définition de R_{IMU} la rotation de l'IMU dans le bateau on déduit la relation suivante où Y sont les points sur la sphère associée au cap et Z sont ces mêmes points après recalage:

$$Z = R_{IMU} * Y$$

En posant :

$$Z_N = \begin{bmatrix} cos(I) \\ 0 \\ -sin(I) \end{bmatrix}$$

$$Z_W = \begin{bmatrix} 0 \\ -cos(I) \\ -sin(I) \end{bmatrix}$$

On extrapole la position vertical, cette extrapolation est permise par la faible amplitude des vagues comme pour l'accélération qu'on considère toujours vertical. $Z_U = Z_N \wedge Z_W$

$$Z_{NWU} = \begin{bmatrix} Z_N & | & Z_W & | & Z_U \end{bmatrix}$$

où I est l'angle entre le plan tangent à la surface de la terre et les lignes de champ magnétique. Au niveau de Guerlédan, on pourra prendre I = 64°
Z correspond dans ce cas aux valeurs théoriques pour un cap bien réglé des points associés au caps plein Nord puis plein Ouest et enfin vers le haut. Il ne nous reste plus qu'à mesurer Y_N, Y_W puis par la relation précédemment établie on en déduit :

$$R_{IMU} = Z_{NWU} * Y_{NWU}^{-1} \text{ où } Y_{NWU} = \begin{bmatrix} Y_N & | & Y_W & | & Y_U \end{bmatrix}$$

NB: chaque colonne de Z correspond au projeté du champ magnétique sur chacun des axes compte tenu de l'orientation du bateau.

Une amélioration qui a pu ensuite être apportée consiste à projeter la matrice R_{IMU} obtenue, qui dû à des imprécisions lors de l'acquisition des Y_N, Y_W, Y_U n'est pas complètement une matrice de rotation, sur SO(3). Pour cela nous avons utilisé la décomposition QR et son implémentation python. La matrice Q est la matrice de rotation voulue.

3 Suivi de ligne sur un triangle

La première différence entre les deux semaines à Guerlédan est donc qu'il ne faut pas simplement contrôler son cap vers un point mais qu'il faut suivre une droite, afin de réaliser des trajectoires en formes de polygones de façon plus précise.

3.1 Partie théorique

Contrairement au premier Guerlédan, nous n'allons pas réaliser la boucle de commande avec un PID, qui calcule l'erreur en direct et corrige la trajectoire petit à petit. À la place, nous utiliserosn le feed Forward Control, avec donc les commandes :

$$u_1 = k_1 * sawtooth(\bar{\theta} - \theta) \tag{1}$$

$$u_2 = k_2 * tanh(\bar{v} - v) \tag{2}$$

On note m la position du DDBoat, et phi l'angle entre la droite (a,b) suivie et l'horizontal. Ainsi, on obtient la distance perpendiculaire entre le bateau et la droite par :

$$e = det(\frac{b-a}{||b-a||}, m-a) \tag{3}$$

On obtient ensuite "thetabar" qui correspond au décalage ajouté au cape de la ligne, permettant de ce rapprocher de cette dernière par :

$$\phi = ang(b-a) \tag{4}$$

$$\bar{\theta} = \phi - tanh(\frac{e}{10}) \tag{5}$$

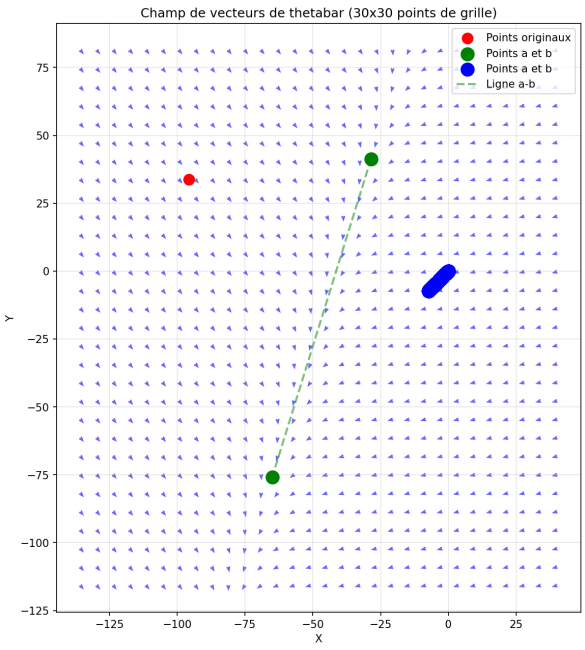


FIGURE 2 – Champ de vecteurs "thetabar"

Il faut aussi trouver une condition d'arrêt lorsque le DDBoat atteint la fin de la ligne. On applique ainsi la condition :

$$(b-a)^T(b-m) < 0 \tag{6}$$

3.2 Implémentation

3.3 Notre implémentation

Les 3 premiers jours ont donc été consacrés à implémenter ce FFC. Nous avons réutilisé la structure de la première semaine à Guerlédan, en simplifiant largement la machine d'états finis pour n'exécuter que cette mission (précédemment la FSM contenait toutes les missions demandées).

Dans les faits, nous avons essayé de comprendre pourquoi notre code de suivi de ligne ne fonctionnait pas, puisque lorsque l'on testait les fonctions individuellement, elles semblaient fonctionner parfaitement. Durant ce processus, nous sommes venus à créer une simulation qui reproduisait le comportement du DDBoat, dont le rendu semblait cohérent.

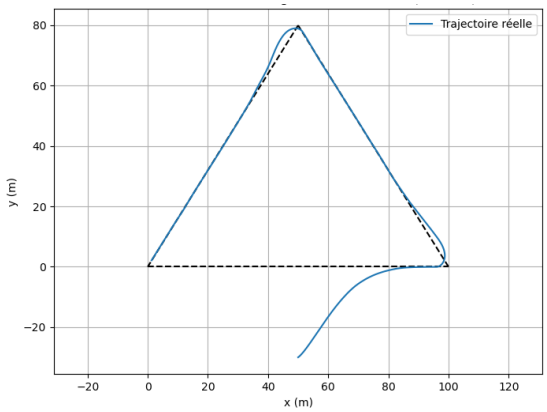


FIGURE 3 – Simulation d'une trajectoire suivant les lignes du triangle

Après débogage et test extensif nous avons réussi a compléter le premier objectif qui était de suivre des lignes composant un polygone, la trajectoire que nous avons obtenu est représentée sur la Figure 4.

FIGURE 4 – Enter Caption

4 Communication entre les bateaux

Deux protocoles de communication ont été utilisés :

— Le protocole TCP pour la communication entre les bateaux.

— Le protocole UDP pour la communication des bateaux vers le serveur.

Le programme que nous avons écrit met en place une classe Communication, avec plusieurs méthodes:

— Trois méthodes pour coder et décoder les données en transit dans le bon format.

— Une méthode qui lance un serveur TCP en sur le DDBoat et qui met chaque client sur un nouveau thread. Une autre méthode envoie alors à période régulière (ici 1 Hz) les données du DDBoat aux clients.

— Une méthode qui créé un client TCP pour se connecter à un autre DDBoat et récupérer ses données.

— Une méthode qui créé un client UDP pour envoyer les données au serveur de logs.

— Une méthode principale, qui ouvre le thread serveur, le thread client UDP, et parcourt toutes les adresses IP possibles pour les DDBoats et ouvre un thread client TCP pour chaque bateau qui répond.

Le principe que nous avons adopté est donc un réseau dans lequel chaque DDBoat est connecté à chaque autre DDBoat connecté. Cela permet de compter le nombre de connections actives et d'en déduire le nombre de DDBoats dans le swarm. Le numéro des bateaux présents étant communiqué, on peut alors les classer par ordre croissant. Chaque DDBoat en déduit alors sa phase.

Nous aurions aussi pu, plus simplement, utiliser un protocole UDP. Cependant, les conseils contradictoires des encadrants n'aidant pas, il nous fallait prendre une décision commune, ce que nous avons fait.

Les données du DDBoat sont utilisées et mises à jour dans l'objet communication à travers un attribut de classe liste modifiée en dehors de l'objet.

5 Conclusion

Ces cinq jours de travail intensif ont été particulièrement denses et exigeants. Certaines journées se sont étendues de 9 h à minuit, avec seulement deux pauses pour les repas : un rythme soutenu qui s'est révélé aussi éprouvant que formateur. Malgré la fatigue accumulée, nous gardons de cette expérience un souvenir très positif.

Tout au long du projet, nous avons rencontré de nombreuses difficultés : des erreurs dans la méthode d'origine, des détails techniques insignifiants qui nous ont fait perdre des heures, ou encore des ajustements imprévus à effectuer dans l'urgence. Ces moments de frustration ont néanmoins été essentiels à notre progression, car ils nous ont permis de mieux comprendre la logique du système et d'affiner nos méthodes de travail en équipe.

Finalement, nous avons atteint les objectifs fixés, et la satisfaction de voir notre DDBoat fonctionner de manière autonome a largement compensé les efforts fournis. Observer le bateau accomplir ses tâches avec succès — sans que nous ayons à sauter dans le kayak toutes les dix minutes — a été une véritable récompense. Cette aventure, bien que fatigante, a été à la fois formatrice, stimulante et très gratifiante.

Guerlédan — Deuxième semaine

Evann MICHEL
Noé CARRAYROU
Eliaz YAHIA

Lors de cette deuxième semaine de travail sur des DDboats à Guerlédan, le but a été de mettre les robots en communication pour leur permettre de réagir en fonction des actions des autres bateaux sur le plan d'eau.

Dans un premier temps, nous avons implémenté une nouvelle approche de suivi de trajectoire par suivi de ligne avant de nous pencher sur la communication entre les robots pour réaliser des missions communes.

La méthode de calibration du magnétomètre du DDboat est directement reprise de la première semaine de travail à Guerlédan, se référer au rapport correspondant pour en avoir le détail.

I Guidage par suivi de ligne (Line Following)

a) Théorie

La méthode de suivi de trajectoire que nous avons implémentée repose sur une loi de guidage géométrique de type "Vector Field Guidance". L'objectif est de contraindre le DDboat à converger de manière asymptotique vers un segment défini par deux points, A et B. La démarche s'articule autour de deux grandeurs fondamentales : l'erreur de distance latérale (e) et la consigne de cap (θ_c).

L'erreur latérale e représente la distance orthogonale entre la position actuelle du robot M et la droite (AB). Elle est calculée via le déterminant normalisé des vecteurs du segment et de la position :

e=det(AB/ // AB // ,AM)

Cette formulation algébrique est déterminante car son signe indique de quel côté du segment se situe l'embarcation, permettant une correction symétrique.

La consigne de cap transmise au pilote automatique est ensuite définie par la relation :

θ=ϕ−arctan(e/k)

où ϕ est l'angle d'inclinaison du segment par rapport au Nord (angle de route directe) et k est un paramètre de gain (ici fixé à 10). Cette fonction assure une transition fluide entre deux régimes :

1. Le ralliement : Lorsque l'erreur e est importante, le terme arctan(e/k) tend vers ±π/2 (90°), forçant le bateau à s'orienter perpendiculairement au segment pour le rejoindre au plus vite.
2. Le suivi : À mesure que le bateau s'approche de la ligne (e→0), la correction s'annule et le cap du robot converge vers l'angle ϕ du segment.

Cela permet d'obtenir deux consignes: une variation de cap et une variation de vitesse, qu'il faut ensuite intégrer à la loi de commande du robot.

Enfin, la gestion d'une suite de segments est assurée par une condition de franchissement de zone: lorsque M se retrouve de l'autre côté de la perpendiculaire à AB en B (ie $AB^T * MB < 0$) le robot considère qu'il doit passer à la ligne suivante.

En implémentant tous ces paramètres, on obtient un robot capable de suivre le tracé de n'importe quel polygone sur le lac, avec une précision correcte.

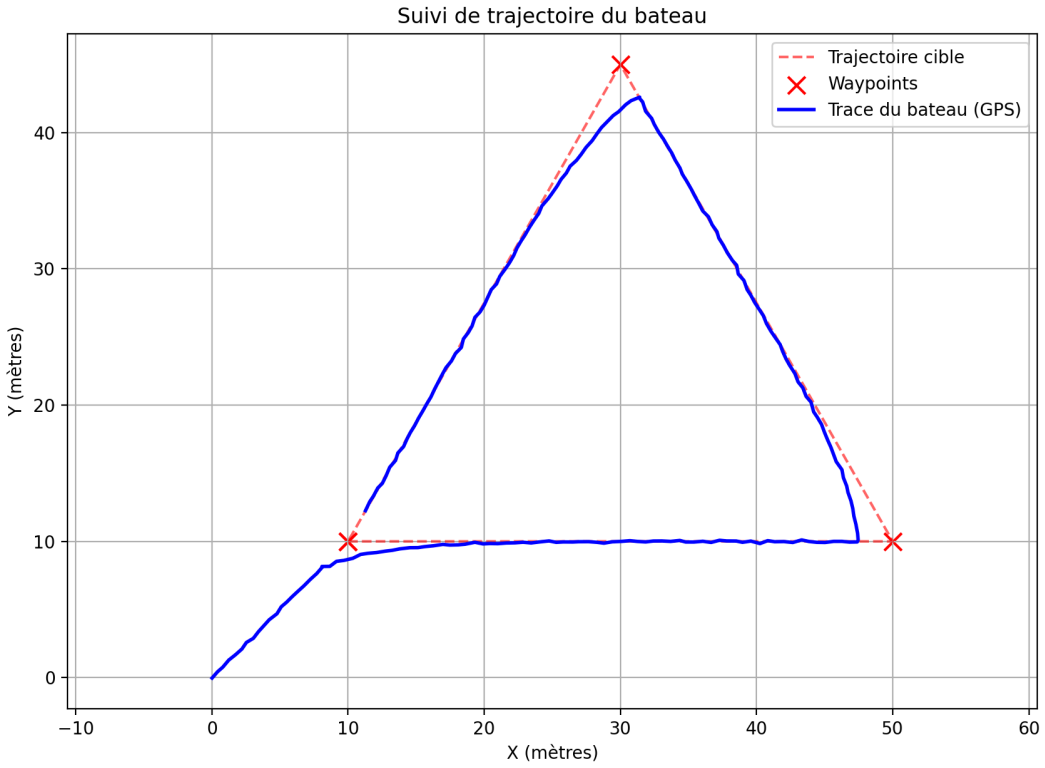
Pour permettre à terme au robot de converger vers une position précise sur le polygone, en accord avec les autres robots ,nous avons ajouté une phase que le robot devait suivre.

Pour cela, il a fallu déterminer une bijection entre une position sur le polygone et une phase dans le cercle trigonométrique. Ce qui se fait en déterminant la longueur parcourue le long du périmètre du polygone et en lui associant une phase unique, en s'accordant sur le fait que le premier point du polygone correspondait au 0 trigonométrique.

Ensuite, il s'agit de déterminer la phase objectif que notre robot doit suivre. Dans un premier temps, il s'agit d'une phase variant avec le temps $\phi(t) = sin(2\pi t/T_{phase} + P_i)$, mais ensuite, pour faire du swarm, il s'agira de la phase transmise par un des autres robots, plus un terme de déphasage. A partir de cette information, on détermine la position que doit avoir notre robot sur le polygone, ce qui nous permet de le faire converger vers la bonne ligne, en adaptant la vitesse de celui-ci par rapport à sa distance avec la position objectif.

b) Simulation

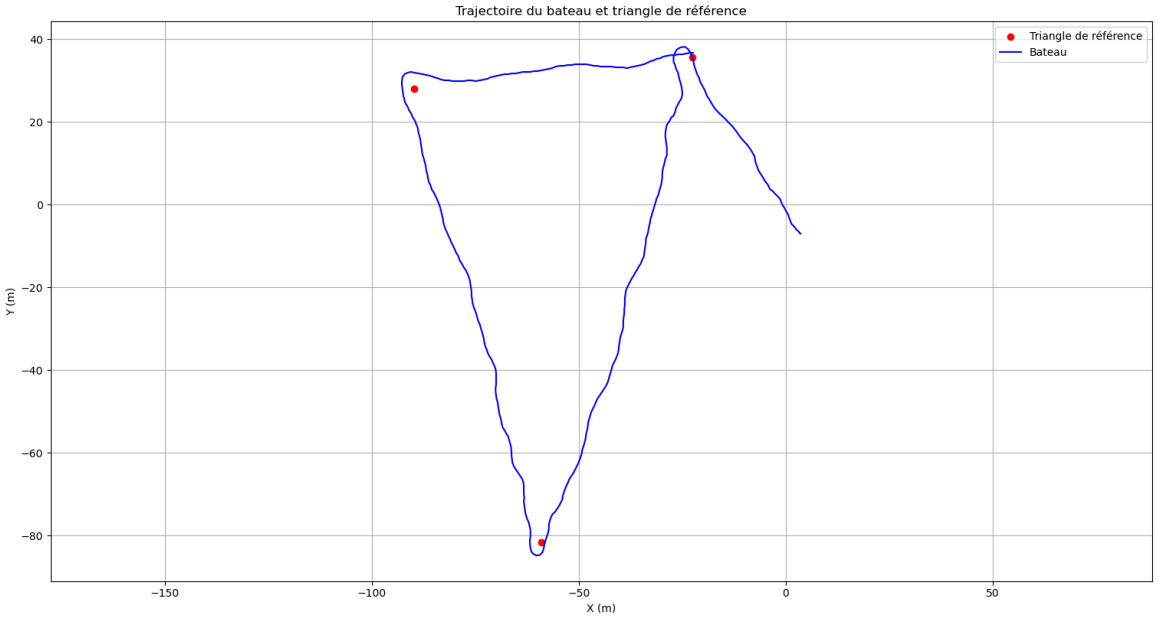
Voici les résultats d'une simulation de suivi de polygone :

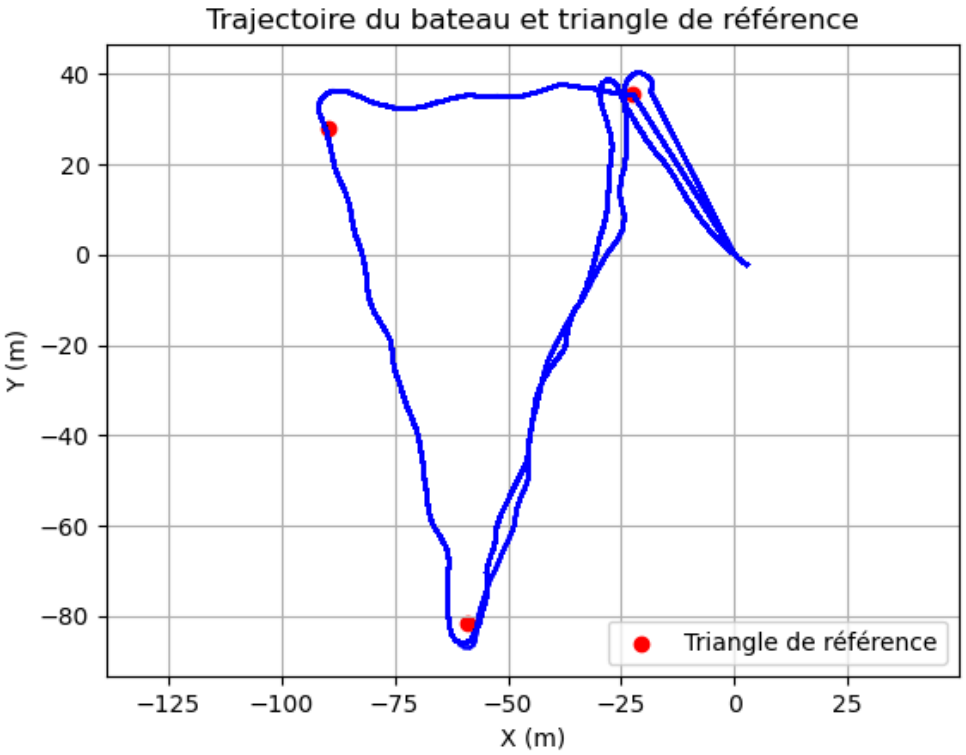


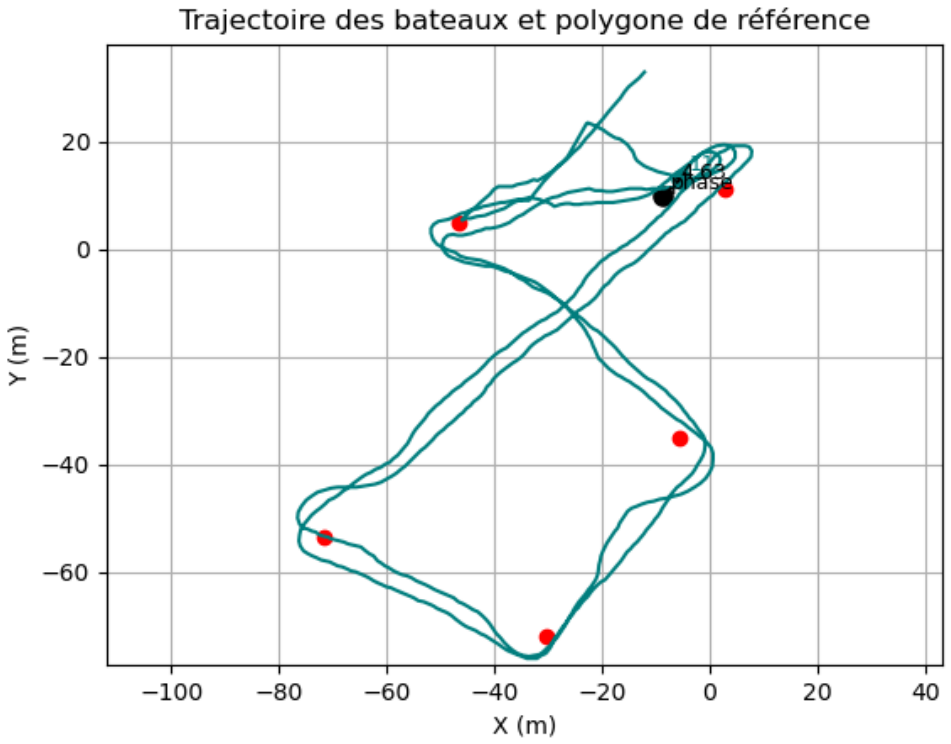
c) Résultats

Avec un robot seul, le suivi de polygone s'est très bien déroulé, avec plusieurs essais qui ont montré une bonne convergence du robot vers le polygone (ici: un triangle). On remarque que le robot dépasse toujours un peu le point qu'il doit atteindre, cela est dû à l'inertie du robot et au GPS (temps de réponse et précision). Pour pallier cela, nous avons par la suite, intégré un gain compensatoire, permettant au robot de ralentir à mesure qu'il s'approchait du point désiré.

Exemple de deux passages du DDBoat sur le triangle:







II Introduction à l'utilisation des sockets

Pour nous familiariser avec l'utilisation des sockets dans le contexte de l'utilisation des DDboats et avant de commencer le SWARM, nous avons réalisé différentes missions.

a) Affichage de la trajectoire en temps réel

La première mission a consisté en la mise en œuvre d'une architecture Client-Serveur distribuée sur le réseau DARTAP. Le dispositif s'articule autour de deux briques logicielles distinctes :

- L'application embarquée (Client) : Intégrée au système de navigation du DDBoat, elle assure l'encapsulation et l'envoi périodique des coordonnées GPS (longitude|latitude) dès leur acquisition par les capteurs.
- La station de contrôle (Serveur) : Hébergée sur un poste distant, cette application écoute en continu les flux de données entrants. Elle traite les paquets de télémesure pour générer une visualisation dynamique de la trajectoire.

Cette infrastructure de supervision en temps réel permet de valider la conformité de la trajectoire parcourue par rapport à la mission pré-programmée. Un protocole UDP est utilisé, au lieu d'un TCP, car il n'est pas nécessaire de recevoir toutes les positions (fréquence de 1Hz) et cela permet de moins saturer le réseau DARTAP.

Ce programme a par la suite été revisité pour enregistrer des logs des positions de tous les bateaux du SWARM dans le format "id du bateau"|"longitude"|"latitude"|"entier correspondant

à la ligne du polygone suivie". Cela permet d'enregistrer toutes les positions des bateaux et de reconstruire dans le futur leurs positions.

b) Stratégie de navigation multi-agents et régulation cinématique

L'objectif de cette mission est de coordonner la navigation d'une flottille de robots DDboats sur une trajectoire commune, tout en garantissant une répartition spatiale régulière et de mettre en place un échange de données inter-robots sans pour autant les traiter.

La démarche repose sur la dualité entre le suivi de trajectoire géométrique et la maîtrise de la dynamique temporelle :

- Guidage latéral : Le maintien des unités sur le tracé imposé est assuré par l'algorithme de suivi de ligne précédemment décrit. Ce dernier garantit que chaque robot minimise son erreur de dérive par rapport au segment de référence.
- Asservissement longitudinal (Régulation de vitesse) : Afin d'assurer un espacement constant entre les mobiles le long de la trajectoire, une loi de commande de vitesse est implémentée. Le système ne se contente plus de suivre un chemin, il doit atteindre une abscisse curviligne cible à un instant t précis.

En ajustant dynamiquement la vitesse de propulsion en fonction de l'avance ou du retard par rapport à une "position virtuelle" de référence obtenue via une fonction de calcul de phase, nous parvenons à synchroniser la progression de l'ensemble de la flottille. Cette fonction de phase permet de répartir régulièrement des points sur le cercle trigonométrique et d'y faire correspondre des positions sur la trajectoire à suivre, assurant la répartition des robots.

Pour cela, chaque robot communique aux autres sa phase calculée sur le cercle trigonométrique. En recevant les phase des autres, chaque robot détermine ensuite une phase objectif et corrige son comportement pour l'atteindre.

Pour le swarm multi-agent, nous avons utilisé deux DDBoats (le 11 et le 15). Dans un premier temps, nous voulions savoir si le suivi de phase transmis était fonctionnel et suffisamment fiable. Nous avons donc fait en sorte que le DDBoat 15 suive le tracé du polygone sans être influencé par le 11 (en suivant donc la phase temporelle). Le 15 devait ensuite transmettre sa phase calculée au 11, qui y ajoutait un déphasage de π et suivait cette phase là. Cela donnait donc une architecture de type master-follower

Dans la pratique, après un certain temps pour permettre aux DDboats de converger, les deux ont pu parcourir le polygone précisément, en étant chacun à l'opposé l'un de l'autre.

Dans un second temps, nous avons expérimenté le principe de consensus en swarm: sans leader particulier, chacun devait calculer sa phase objectif par rapport à celle des des autres. Cependant, avec seulement 2 DDBoats, nous avons été confrontés à plusieurs problèmes: s'ils ne commençaient pas dans les bonnes conditions, les DDBoats mettaient souvent beaucoup de temps à converger. Cela était souvent dû au fait que la

phase reçue donnait une autre ligne à suivre que celle suivie actuellement, ce qui changeait la phase calculée, et donc celle transmise à l'autre robot, et vice versa. Nous avons tout de même réussi par moment à avoir un consensus convergent.

Il était donc important d'augmenter le nombre de robots dans la flottille. Cependant, à cause de désaccords sur le protocole de communications entre robots entre les différentes équipes, nous n'avons pas eu le temps d'explorer plus en avant.

III Développement de fonctionnalités annexes en simulation

En plus du travail de développement de l'algorithme demandé pour la manipulation finale, nous avons simulé le comportement de bateau en déplacement sur le polygone en ajoutant une fonction d'évitement des bateaux passant à proximité tout à fait adaptée au SWARM. L'idée est de s'inspirer des intéractions entre particules chargées pour superposer à la position partagée par les bateaux sur le plan d'eau un champ local de vecteurs répulsifs superposé à un champ local en rotation pour permettre aux bateaux de s'éviter.

Nous avons aussi simulé une petite mission de défense d'un territoire qui correspond à un usage des communications TCP et UDP.

IV Analyse des résultats et traitement des données

a) Prétraitement et Stabilisation Vidéo

La première phase a consisté à préparer le support visuel pour l'extraction de mesures spatiales. Après une conversion de format pour optimiser la compatibilité logicielle, un algorithme de stabilisation d'image a été appliqué. En analysant le flux optique des éléments fixes du décor, nous avons généré un fichier de mouvements de caméra permettant de compenser les oscillations du drone. Cette étape est cruciale car elle transforme un repère mobile et instable en un plan fixe virtuel, condition *sine qua non* pour considérer les coordonnées pixels comme des mesures de trajectoires exploitables.

b) Extraction des Trajectoires par Tracking

Le suivi des mobiles a été réalisé via un algorithme de tracking par segmentation colorimétrique. En exploitant le contraste élevé entre la coque blanche des robots et le milieu aquatique, nous avons isolé les centroïdes des objets d'intérêt. L'association temporelle repose sur l'hypothèse de la proximité minimale : entre deux images, l'objet blanc le plus proche de la position précédente est identifié comme le robot. Cette méthode, bien que simplifiée, s'est avérée robuste pour traiter plusieurs milliers d'images et produire un jeu de données continu en coordonnées (x,y) pixels.

c) Synchronisation Spatio-Temporelle et Fusion

L’enjeu central de cette phase réside dans la fusion cohérente de deux sources hétérogènes : les logs GPS (données métriques discrètes) et le flux vidéo (données pixels continues). Le défi est double, nécessitant la résolution simultanée d’un décalage temporel (latence de déclenchement des enregistreurs) et d’un décalage spatial (homothétie et rotation du plan drone par rapport au Nord géographique).

L’analyse qualitative suggère que la comparaison des signatures de cap (orientation) et des morphologies de trajectoires offre une robustesse bien supérieure à celle des vitesses instantanées. Ces dernières, bien que physiquement pertinentes, se révèlent trop vulnérables au bruit de quantification du tracking et aux pertes de paquets chroniques observées dans les logs GPS, qui génèrent des vitesses nulles artificielles.

L'intérêt pédagogique et technique de cette démarche réside dans la réduction de dimensionnalité. Alors que nous travaillons initialement sur des signaux bidimensionnels (trajectoires XY), nous exploitons les outils classiques du traitement du signal — habituellement réservés au domaine unidimensionnel (audio, séries temporelles) — pour retrouver une sous-partie d'un enregistrement dans un flux global. En extrayant une grandeur scalaire telle que le cap ou la norme de la vitesse, nous transformons un problème de superposition géométrique complexe en un problème de corrélation croisée de signaux.

Enfin, il convient de souligner l'asymétrie d'information entre nos sources : la vidéo, par sa haute fréquence d'échantillonnage (30 Hz), offre une continuité et une richesse de détails bien supérieure aux logs (1 Hz).

d) Analyse des Écarts et Perspectives

Bien que les motifs globaux soient identifiables à l'œil en considérant les motifs ayant la plus forte ressemblance, des divergences subsistent entre les tracés GPS et vidéo. Ces écarts soulèvent des questions sur la possibilité d’exploiter les données en l’état dans un programme. Il faudrait filtrer et compléter les données pour les rendre directement utilisables pour des analyses numériques. La suite des travaux vise à affiner la matrice de passage par une corrélation croisée plus rigoureuse, afin de valider le consensus entre les capteurs embarqués et l'observation aérienne, garantissant ainsi la fiabilité de la vérité terrain.

ENSTA | INSTITUT POLYTECHNIQUE DE PARIS

Le DD-Boat à l'épreuve

Rapport du projet expérimental de Guerlédan

EXPÉDITION 2

Clément SARTRE — clement.sartre@ensta.fr
Simon KLÉCHA — simon.klecha@ensta.fr

18 février 2026

Table des matières

Résumé

Du 09 au 13 février 2026, la classe de ROB 27 est partie a Guerlédan pour faire du SWARM avec les DD-Boat. Le groupe constitué de Simon KLÉCHA et Clément SARTRE a rencontré de nombreux problèmes (et parfois leurs solutions) afin de tenter de répondre aux différentes missions imposées par Luc Jaulin. Ceux-ci font l'objet de ce rapport.

Mots clés

DD-Boat, calibration, cap, localisation, IMU, GPS, SWARM, UDP, polygone

Abstract

From February 9 to 13, 2026, the ROB 27 class traveled to Guerlédan to participate in SWARM with DD-Boats. The group, consisting of Simon KLÉCHA and Clément SARTRE, encountered numerous problems (and sometimes found solutions) while attempting to complete the various missions assigned by Luc Jaulin. These are the subject of this report.

Keywords

DD-Boat, calibration, heading, localization, IMU, GPS, SWARM, UDP, polygon

Remerciements

Luc Jaulin, Natacha Carouen et à tous les encadrants pour cette expérience si concluante. Nous exprimons un sentiment de reconnaissance particulier envers le personnel du self, à qui nous devons bien 90% de notre réussite, sans hésitation.

Liste des acronymes

ENSTA École Nationale Supérieure de Techniques Avancées
IMU Inertial Measurement Unit
GPS Global Positioning System
UDP User Datagram Protocol

1 Introduction

Ce séjour, comme le précédent, s'est organiser en objectifs évolutifs de la recalibration de notre bateau lors du premier jour jusqu'à réaliser l'individu d'un SWARM programmé pour s'équirépartir autour d'un polygone.

Les étapes de cette aventure sont détaillées par ordre chronologique dans la suite, en vue de répondre à la question suivante :avons-nous réussi le défi Guerlédan EXPÉDITION 2 ?

2 Configuration du DDBoat

2.1 Retrouvaille avec notre DDBoat

Nous avons retrouvé notre DDBoat 13 après une longue période durant laquelle il nous manquait. Nous avons programmer les formules nécessaires à la calibration décrite dans la partie suivante.

2.2 Recalibration

Un magnétomètre fournit des données brutes perturbées par les effets *hard-iron* et *soft-iron*. Ces perturbations introduisent une différence entre l'espace des données brutes $B(3)$ et le groupe $SO(3)$ des rotations dans l'espace tridimensionnel. On peut montrer qu'il existe une transformation affine qui réalise un difféomorphisme entre l'espace des données brutes $B(3)$ et l'espace des rotations $SO(3)$; autrement dit, $B(3)$ est une ellipsoïde. Intuitivement, l'effet *hard-iron* est dû à des objets produisant un champ magnétique propre, tandis que l'effet *soft-iron* se traduit par des déviations ou des altérations du champ magnétique existant.

2.2.1 Les mathématiques

En enregistrant les mesures de l'IMU pendant un certain temps, nous obtenons un ensemble de points $P \subset B(3)$. Puisque $B(3)$ est une ellipsoïde, il existe des coefficients $(p_i)_{i\in\{1,\dots,9\}} \in \mathbb{R}^9$ tels que, pour tout $x \in P$,

$$p_1x_1^2 + p_2x_2^2 + p_3x_3^2 + p_4x_1x_2 + p_5x_1x_3 + p_6x_2x_3 + p_7x_1 + p_8x_2 + p_9x_3 = 1. \tag{1}$$

Nous pouvons donc réaliser une méthode des moindres carrés. Notons n le nombre d'éléments de P et

$$P = \{x(i) \mid i \in \{1, \dots, n\}\}.$$

On cherche $\vec{p} \in \mathbb{M}(9,1)$ telle que la matrice

$$X = \big(x(i)_1^2, x(i)_2^2, x(i)_3^2, x(i)_1x(i)_2, x(i)_1x(i)_3, x(i)_2x(i)_3, x(i)_1, x(i)_2, x(i)_3\big)_{i\in\{1,\dots,n\}} \in \mathbb{M}(n,9)$$

vérifie $X\vec{p} = \mathbf{1}$, d'où

$$\vec{p} = (X^TX)^{-1}X^T\mathbf{1}.$$

La forme canonique d'une ellipsoïde s'écrit

$$\exists(\bar{x}, Q) \in \mathbb{R}^3 \times \mathbb{S}_3^+(\mathbb{R}), \quad (x-\bar{x})^TQ(x-\bar{x}) = 1. \tag{2}$$

On montre que $\bar{x} = -\frac{1}{2}P^{-1}C$ et

$$Q = \frac{P}{1 + \bar{x}^TP\bar{x}}$$

avec

$$P = \begin{bmatrix} p_1 & \frac{p_4}{2} & \frac{p_5}{2} \\ \frac{p_4}{2} & p_2 & \frac{p_6}{2} \\ \frac{p_5}{2} & \frac{p_6}{2} & p_3 \end{bmatrix}, \quad C = \begin{bmatrix} p_7 \\ p_8 \\ p_9 \end{bmatrix}.$$

En posant, pour $x \in B(3)$, $y(x) = \sqrt{Q}\,(x-\bar{x})$, on vérifie que $\forall x \in B(3)$, $y(x)^Ty(x) = 1$. L'image $y(B(3))$ est donc une sphère, et un point de cette sphère permet de représenter une rotation.

2.2.2 Le programme

Le programme fonctionne, et nous obtenons bien des points sur le cercle après calibration.

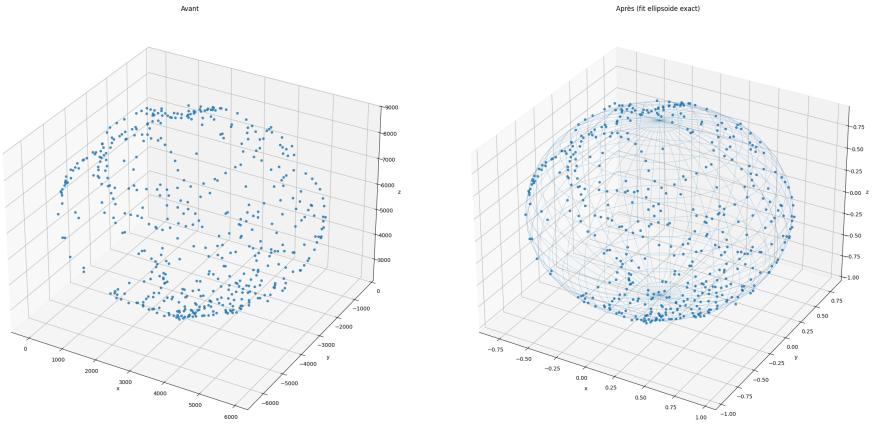


FIGURE 1 – Affichage des nuages de points avant et après calibration

2.3 Tests de mise à l'eau

Après la calibration, il reste à effectuer la réorientation fictive de l'IMU. Nous avons appliqué la méthode théorique décrite dans notre précédent rapport, en immobilisant le bateau dans deux poses et en effectuant un produit vectoriel sur nos deux premiers vecteurs afin d'obtenir la matrice de référence pour la rotation.

Après cette rectification, nous avons continué à observer des erreurs dont l'origine restait incertaine. Nous espérions une erreur quasi constante sur les caps, mais la précision de la boussole du téléphone laissait à désirer. Il était donc difficile d'obtenir des points de référence pour définir correctement R_{IMU}.

Pensant avoir trouvé des rectifications cohérentes, nous avons lancé une mission pour envoyer le DDBoat à la bouée puis le faire revenir. Comme on le voit sur la figure 2, il atteint la bouée, mais dès qu'il tourne pour retourner au ponton, on observe une forte différence entre le cap désiré (en vert) et le cap réel (en bleu) du bateau, sachant que ce dernier devrait être tangent à la trajectoire du bateau.

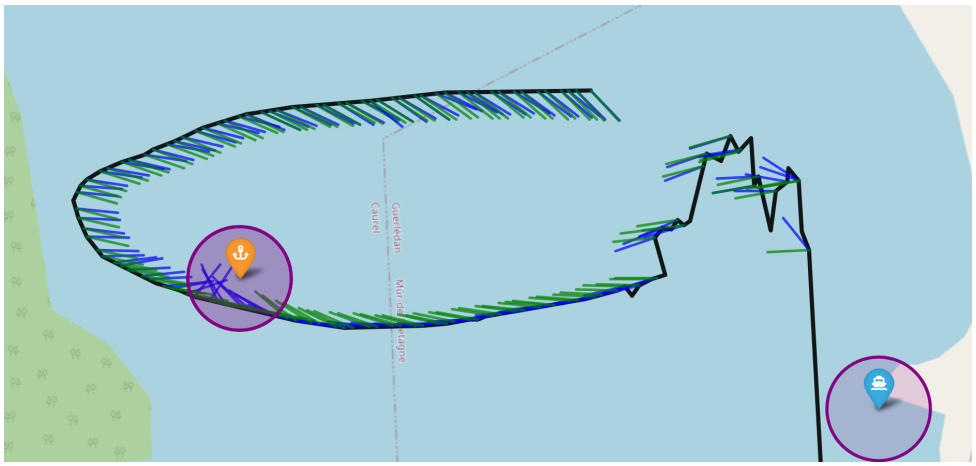


FIGURE 2 – Illustration erreur mission bouée 1

On observe que le cap mesuré s'éloigne de la tangeante de la trajectoire du bateau au fur et à mesure, il y a donc une erreur évolutive dans notrre calibration de l'IMU. Nous avons tenté à plusieurs reprises de relancer le programme, mais nous observions un comportement difficile à interpréter avec les logs. De plus, lorsque nous remesurions uniquement le cap reçu en pointant le bateau vers certaines directions, celui-ci restait incohérent avec le cap réel, alors que nous avions déjà effectué plusieurs rectifications pour obtenir quelque chose de cohérent. Après moulte tentatives, nous finissons par essayer la manipulation avec dles programmes de nos camarades. Mais, même avec une calibration effective sur la terre ferme, le comportement du bateau sur l'eau est chaotique :

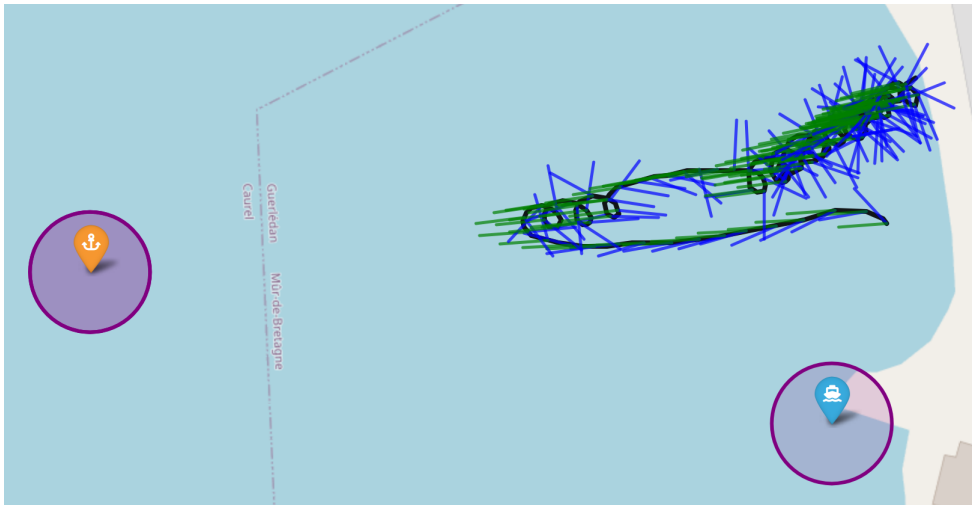


FIGURE 3 – Illustration erreur mission bouée 2

Le temps était assez capricieux et nous pensons que les vagues pouvaient avoir une influence sur le cap ressenti à cause d'une assiette différente de celle utilisée pour notre calibration. De plus les vagues ont pu imposer au bateau une rotation trop difficile à compenser avec sa boucle de contrôle, expliquant ces boucles répétitives.

2.4 Problèmes moteurs

Il nous a fallu du temps pour remarquer le comportement disymétrique des moteurs en rotation inverse. Cela nous a également permis de remarquer que notre propulseur droit

avait été monté à l'envers, ce qui est propbablement la cause de ce dysfonctionnement. Une vidéo complète du test permettant de mettre en évidence ce problème a été rendue avec ce présent rapport. Elle est également disponible ici.

3 De l'individuel au collectif

3.1 Suivi de ligne et triangle

Afin de réussir la mission pour aller à la bouée, nous calculons le cap que doit suivre le bateau pour aller à celle-ci à l'aide des coordonnées GPS et effectuons un suivi de cap. Nous nous servons de ce suivi de cap afin de réaliser un suivi ligne :

Notons $m \in \mathbb{R}^2$ la coordonnée du bateau et $(a, b) \in (\mathbb{R}^2)^2$ les coordonnées du segment que l'on cherche à suivre. On définit $\phi = angl(b-a)$ l'angle entre l'axe des abscisses et le segment (a, b). Notons $e = \det\left(\frac{b-a}{\|b-a\|}, m-a\right)$ la distance entre le bateau et la droite (ab). Nous controlons notre bateau en lui demandant de suivre le cap $\overline{\theta} = \phi - \tanh\left(\frac{e}{K_q}\right) K_3$ où K_q et K_3 sont deux constantes qu'on choisit afin que notre bateau soit stable et converge rapidement vers un état que l'on estime bien. Ci-dessous, on a tracé le champ de gradient de la fonction $m(t)$ de la position du bateau en fonction du bateau avec pour paramètre $K_q = 7$ et $K_3 = 2$

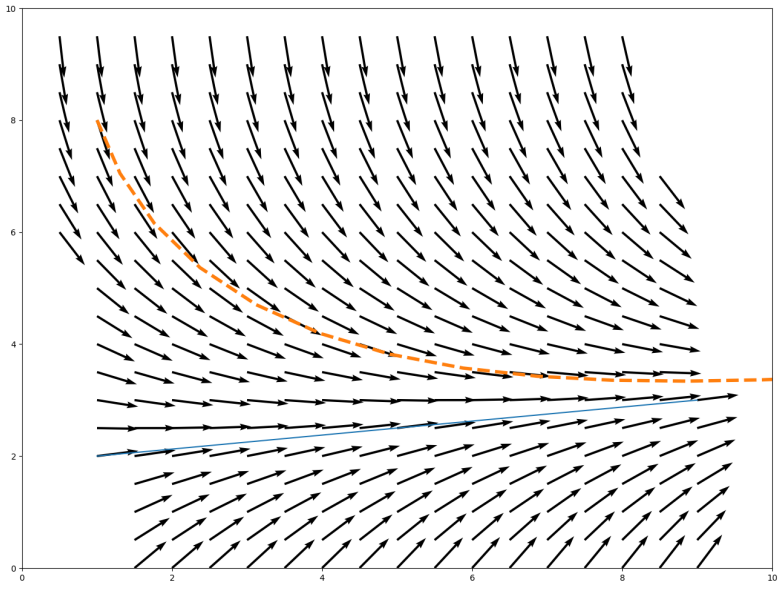


FIGURE 4 – Champ de gradient de la position du bateau

Nous indiquons à notre bateau d'arrêter de suivre la droite (ab) et de passer à sa mission suivante lorsqu'il "dépasse" le point b, c'est à dire lorsque $(b-a) \cdot (b-m) < 0$.

Désormais nous pouvons effectuer n'importe quel suivi de polygone, par exemple un triangle, comme sur la figure 5.

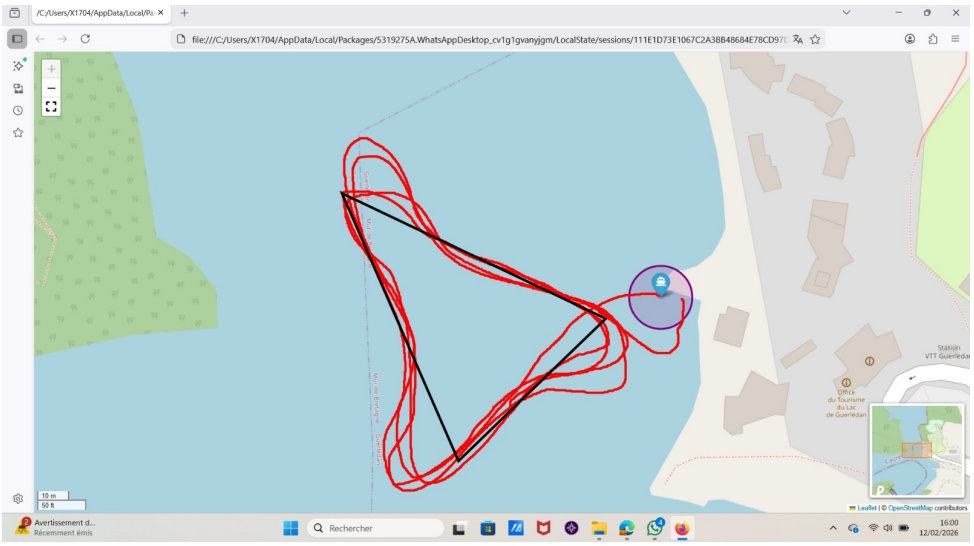


FIGURE 5 – Suivi du triangle

En revanche, nous ne pouvons pas encore le faire en un temps prédéfini, ce qui fait l'objet de la régulation suivante.

3.2 Suivi d'un polygone avec régulation du temps de trajet

Soit $n \geq 1$. On considère un polygone défini par ses sommets

$$(a_i)_{i \in \mathbb{Z}/n\mathbb{Z}} \in (\mathbb{R}^2)^n,$$

où les indices sont pris modulo n.

On note $\mathrm{dist}(p, q)$ la distance euclidienne entre deux points $p, q \in \mathbb{R}^2$.

Pour tout $i \in \mathbb{Z}/n\mathbb{Z}$, on définit la longueur de l'arête i par

$$l_i = \mathrm{dist}(a_i, a_{i+1}),$$

et on suppose que

$$l_i > 0 \quad \forall i \in \mathbb{Z}/n\mathbb{Z}.$$

On définit le périmètre total du polygone par

$$L = \sum_{i=0}^{n-1} l_i.$$

Afin de définir rigoureusement l'ensemble des points du polygone tout en conservant l'information de l'arête, on introduit l'espace abstrait

$$\tilde{P} = \bigsqcup_{i \in \mathbb{Z}/n\mathbb{Z}} ([0, l_i] \times \{i\}),$$

où $\bigsqcup$ désigne l'union disjointe.

On définit une relation d'équivalence $\sim$ sur $\tilde{P}$ par

$$(l_i, i) \sim (0, i+1) \quad \forall i \in \mathbb{Z}/n\mathbb{Z},$$

et aucune autre identification.

On définit alors

$$P = \tilde{P}/\sim .$$

Un élément de P est noté $[(s,i)]$, où $s \in [0, l_i]$ représente la distance au sommet a_i le long de l'arête i.

On définit l'application géométrique naturelle

$$\pi : P \to \mathbb{R}^2$$

par

$$\pi([(s,i)]) = a_i + \frac{s}{l_i}(a_{i+1} - a_i),$$

qui paramétrise le bord géométrique du polygone.

On définit maintenant l'application

$$\Phi : P \to \mathbb{U}$$

par

$$\forall [(s,i)] \in P, \quad \Phi([(s,i)]) = \exp\left(2\pi i \, \frac{\displaystyle\sum_{j=0}^{i-1} l_j + s}{L} \right).$$

L'application Φ est alors un homéomorphisme de P sur le cercle unité $\mathbb{U}$.

Nous souhaitons désormais que le bateau parcoure la trajectoire polygonale en une période fixée T. Pour cela, nous définissons une trajectoire de consigne sur le cercle unité $\mathbb{U}$ par

$$\phi(t) = \exp\left(2i\pi \frac{t}{T}\right).$$

Cette application décrit un parcours uniforme du cercle unité, avec une période T.

En utilisant l'homéomorphisme $\Phi : P \to \mathbb{U}$ défini précédemment, on en déduit le point de consigne sur le polygone

$$p(t) = \Phi^{-1}(\phi(t)).$$

Ainsi, le point $p(t)$ décrit le polygone à vitesse constante en longueur d'arc, et la trajectoire complète est parcourue en temps T.

Afin d'assurer que le bateau respecte cette contrainte temporelle, nous ajoutons une régulation de la vitesse fondée sur l'erreur longitudinale le long du segment courant. Plus précisément, nous considérons la projection orthogonale de la position du bateau sur la droite support du segment suivi, et mesurons la distance entre cette projection et le point de consigne $p(t)$. Cette erreur permet de quantifier l'avance ou le retard du bateau le long de la trajectoire, et est utilisée pour ajuster sa vitesse de manière à garantir la convergence vers la trajectoire de référence tout en respectant la progression temporelle imposée.

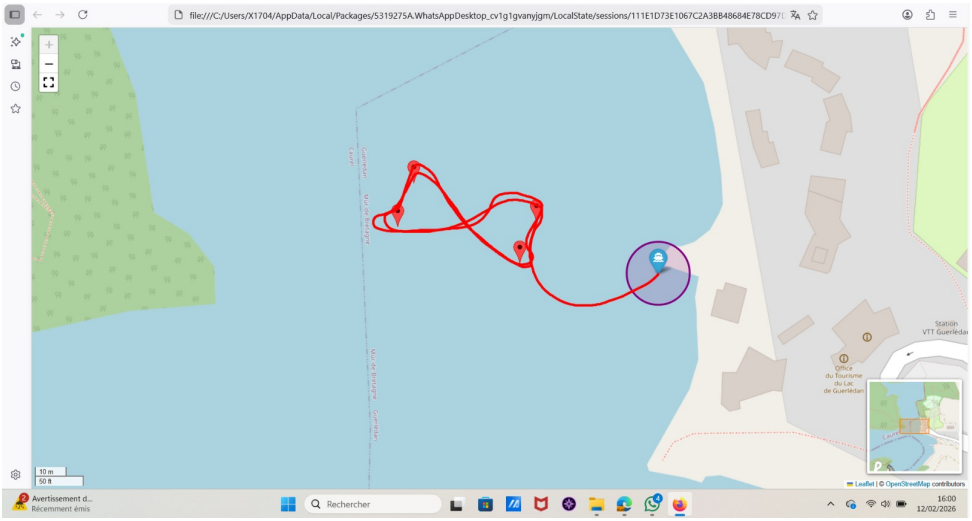


FIGURE 6 – Suivi du polygone

3.3 Explication théorique du SWARM

L'objectif de cette mission était de permettre à plusieurs DDBoats de se répartir uniformément le long du polygone, grâce à un mécanisme de consensus distribué reposant sur une communication entre bateaux. Chaque bateau jouait simultanément le rôle de client et de serveur au sein d'une architecture de communication en chaîne. Plus précisément, chaque bateau transmettait ses propres informations, notamment sa position le long du polygone, sa vitesse et son état, au bateau situé derrière lui, et recevait les informations du bateau situé devant lui. À partir de ces données, chaque bateau pouvait estimer sa position relative et ajuster sa vitesse afin de maintenir un espacement cohérent avec ses voisins, permettant ainsi une répartition progressive et autonome de l'ensemble des bateaux le long de la trajectoire. Ce fonctionnement reposait uniquement sur des échanges locaux entre bateaux adjacents, sans nécessiter de coordination centralisée pour le contrôle. Toutefois, une telle architecture implique que chaque bateau utilise la position de son voisin comme référence, ce qui peut entraîner une propagation des erreurs le long de la chaîne en cas de comportement incorrect d'un des agents. Dans une seconde phase, chaque bateau transmettait également ses informations à un serveur central. Les logs de ce serveur permettront de visualiser la trajectoire du DDBoat 13 dans cette chorégraphie SWARM.

4 Conclusion

La robotique est un foisonnement de difficultés redondantes, et nous ne attendions pas à devoir recommencer la calibration. Notons que notre bateau a été modifié, ce qui explique cette situation. Quoiqu'il en soit, nous avons su, à force de détermination, implémenter les algorithmes de suivi de ligne, puis de suivi de polygone, puis de SWARM, malgré les difficultés. La réussite réside dans la persévérance, voilà la vraie leçon de Guerlédan.

ENSTA | INSTITUT POLYTECHNIQUE DE PARIS

Rapport de Projet

Promotion 2027

DD-Boat Guerlédan

Auteurs :
Pedro BARBOSA NORI
Matheus BENIGNO DA SILVA
Daniel FÁRLEI SILVA SOUZA

Encadrant :
M. Luc JAULIN

Brest, le 21 février 2026

Table des matières

1 Introduction

Durant la semaine du 10 au 14 février 2026, les élèves l'ENSTA spécialisés en robotique sont allés au Lac de Guerlédan pour éffectuer des missions avec un ASV.

Les activités au Lac de Guerlédan se sont articulés autour de la thématique centrale de la composition d'un *swarm* (essaim) avec les ASV[1], aussi appelés DD-Boats. Notre groupe a utilisé le DD-Boat número 16 pour effectuer les missions suivantes, présentées par ordre de difficulté croissante :

— Calibration du magnetometre
— Suivre les lignes d'une trajectoire triangulaire données par 3 coordonnées GPS.
— Suivre un polygone à partir d'une position de phase de départ rélatif au trajet.
— Suivre un polygone en ajustant la position de phase rélatif au bateau suivant (*swarm*).

Compte tenu des codes crées lors de la précédente visite au lac en novembre 2025, les étapes concernant la communication avec les capteurs, la calibration du magnetométre et le contrôle des môteurs étaient déjà éffectués.

Pour le versionnement du code, nous avons utilisé le GitLab de l'ENSTA, disponible via le lien suivant `https://gitlab.ensta-bretagne.fr/barbospe/guerledan-2025`.

Tous les éléments théoriques présentés dans ce rapport s'appuient sur le document accessible à l'adresse suivante : `https://webperso.ensta.fr/jaulin/ddboats_2A.pdf`

1. ASV : *Autonomous Surface Vehicle*

2 Mission 1 - Calibration du magnetometre

2.1 Objectif

Calibrer le magnétomètre embarqué pour minimiser les effets de *hard-iron* et *soft-iron*. Ensuite, visualiser l'ellipsoïde des données brutes collectées, puis effectuer sa transformation en une sphère centrée.

2.2 Résultats

Les résultats de cette première mission sont concluants et ont permis de valider la chaîne d'acquisition du magnétomètre avant de réaliser les suivis de trajectoire. En utilisant les algorithmes déjà développé pendant la dernière visite à Guerlédan, nous avons recalculé les paràmetres de calibration. Cependant, contrairement à la méthode utilisée lors du séjour précédent, qui nécessitait trois mesures de pointage vers les directions du Nord (y_N), de l'Ouest (y_W) et de la verticale (y_H), nous avons simplifié l'algorithme pour éviter d'avoir à positionner le DD-Boat vers le haut. La direction y_H a été obtenue mathématiquement par le produit vectoriel de les deux vecteurs précédents.

Comme illustré sur la Figure 1, les données brutes initiales présentaient une distorsion caractéristique en ellipsoïde due aux effets de *hard-iron* et *soft-iron*. Après application du correcteur calculé par notre script, nous avons obtenu une sphère centrée à l'origine (Figure 2), garantissant ainsi une mesure plus précise du cap magnétique pour les missions suivantes.

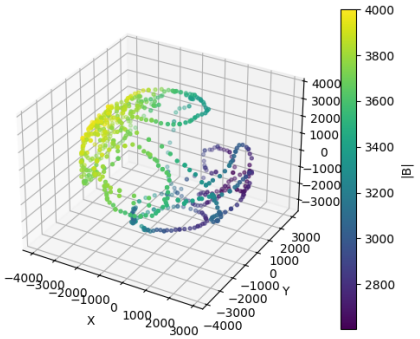


FIGURE 1 – Ellipsoïde des données brutes

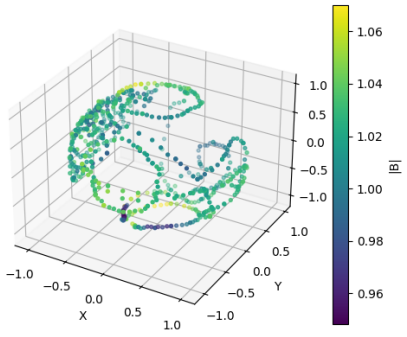


FIGURE 2 – Sphère avec les données normalisées

3 Mission 2 - Suivi de ligne

3.1 Objectif

L'objectif de cette mission est de réaliser le suivi d'une trajectoire triangulaire fermée, définie par trois points de passage exprimés en coordonnées GPS. Le robot doit converger vers chaque segment de la trajectoire et valider le franchissement d'un point pour passer automatiquement au segment suivant.

3.2 Éléments de théorie

Pour cette mission, nous avons implémenté un contrôleur sans modèle dissociant la commande de vitesse et la commande de cap. Les lois de commande utilisées sont les suivantes :

— **Commande de vitesse** (u_1) : régule la vitesse linéaire v vers la vitesse désirée v_d en utilisant une fonction de saturation :

$$u_1 = k_1 \cdot \tanh(v_d - v) \tag{1}$$

— **Commande de cap** (u_2) : ajuste la direction du robot en utilisant la fonction *sawtooth* pour éviter les sauts de phase de 2π :

$$u_2 = k_2 \cdot \text{sawtooth}(\theta_d - \theta) \tag{2}$$

Pour assurer le suivi de la ligne, le cap de référence θ_d est calculé en fonction de l'erreur latérale e par rapport au segment de trajectoire d'angle ϕ :

$$\theta_d = \phi - k_3 \cdot \tanh\left(\frac{e}{k_4}\right) \tag{3}$$

3.3 Résultats

Afin d'optimiser le temps d'expérimentation, nous avons sélectionné trois *waypoints* situés à proximité immédiate de la zone d'essai. La mission a été accomplie avec succès

sur le lac à la fin du première jour. Comme illustré par les relevés GPS (Figure 3), le DD-Boat converge efficacement vers les segments et gère correctement les transitions entre les lignes. Cette transition est déclenchée lorsque le robot entre dans le demi-plan de validation du waypoint suivant.

Les graphiques de la Figure 4 permettent d'analyser le comportement dynamique du système. On observe que le cap réel suit la consigne θ_d, bien que le suivi ne soit pas strictement superposé à la ligne théorique. Cet écart est dû au réglage des constantes k_3 et k_4, qui ont été configuré avec les valeurs 0.1 et 10, respectivement, afin d'assurer une navigation fluide, au prix d'une précision de suivi légèrement réduite.

Les signaux PWM montrent une activité cohérente avec les erreurs de cap mesurées. Malgré l'imprécision inhérente aux mesures GPS, les résultats globaux sont très satisfaisants et valident la robustesse du contrôleur.

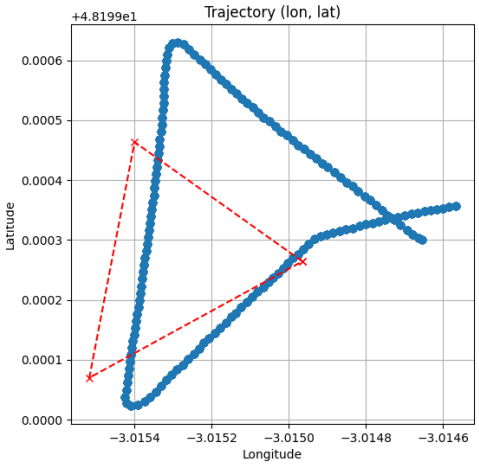


FIGURE 3 – Suivi de la trajectoire triangulaire

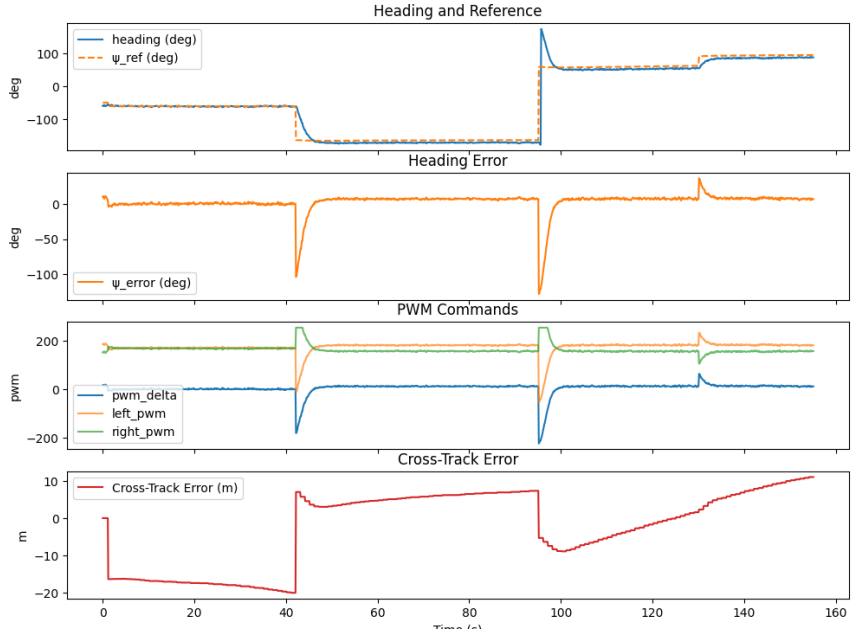


FIGURE 4 – Graphiques de comportement du robot

4 Mission 3 - Suivi de poylgone

4.1 Objectif

Dans la mission 3, le bateau doit parcourir un polygone à quatre côtés en respectant la contrainte de phase pour effectuer un tour complet.

Pour cela, le système conserve la logique de suivi de ligne sur chaque segment, mais ajoute une couche de synchronisation par phase : la position du bateau le long du parcours est convertie en une phase actuelle (4), et cette phase est comparée à une phase désirée (5) afin d'obtenir une erreur de phase (6), utilisée pour ajuster la vitesse.

$$\varphi_{\text{actuel}} = 2\pi \cdot \frac{d}{L_{\text{total}}} \tag{4}$$

$$\varphi_{\text{des}}(t) = 2\pi \cdot \frac{t}{T} + \bar{\varphi} \tag{5}$$

$$e_\varphi = \text{sawtooth}\left(\varphi_{\text{des}} - \left(\varphi_{\text{actuel}} + \frac{2\pi}{N}\right)\right) \tag{6}$$

Le temps a été défini à partir de la longueur totale du parcours, imposant une vitesse de base et corrigeant les écarts via l'erreur de phase, ce qui garantit que le bateau termine le polygone dans l'intervalle prévu.

4.2 Résultats

La mission a été menée à bien et le robot a suivi le polygone défini par les points GPS. Cependant, lors du premier test (Figure 5), on a constaté que la trajectoire du

robot s'écartait sensiblement de la trajectoire réelle. Afin d'améliorer ce comportement, le paramètre k_4 a été diminué de 10 à 4, ce qui a permis d'obtenir les résultats présentés sur la Figure 6.

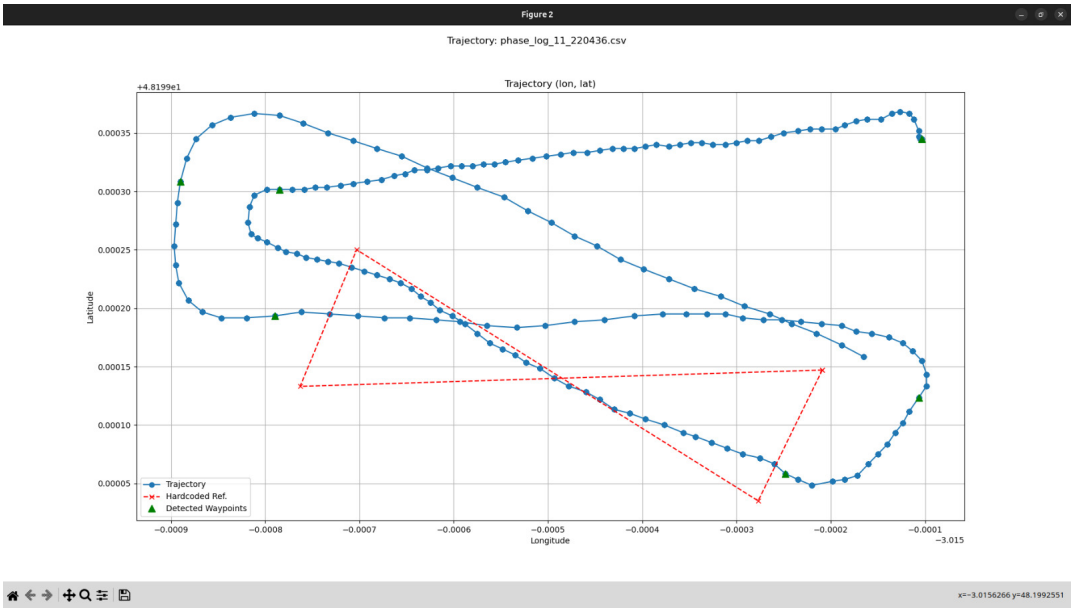


FIGURE 5 – Suivi du polygone avant modification des constantes

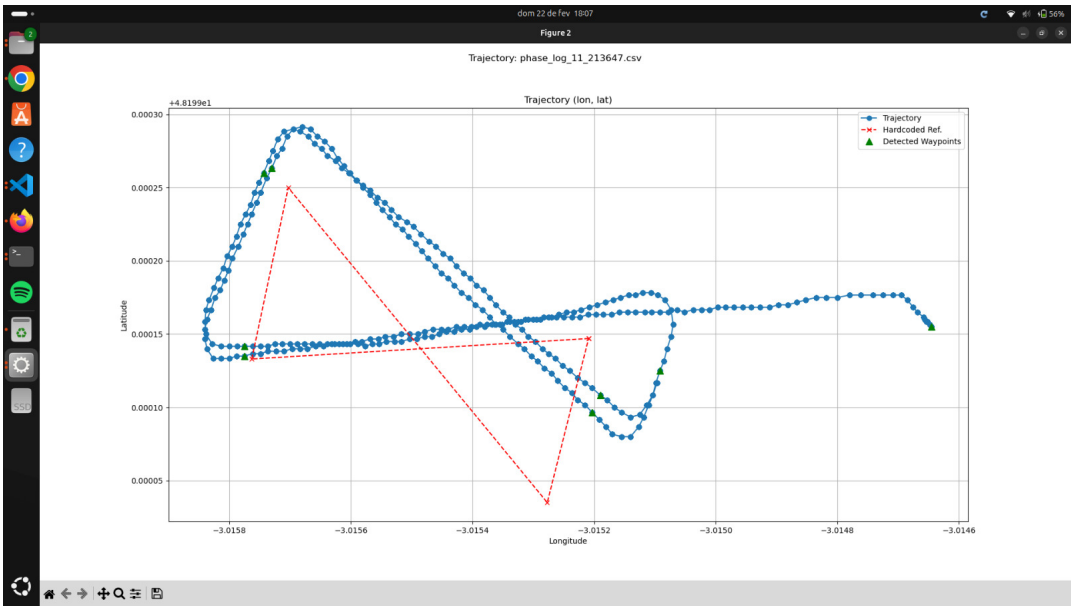


FIGURE 6 – Suivi du polygone après modification des constantes

5 Mission 4 - Suivi de poylgone (*swarm*)

5.1 Objectif

Pour la mission 4, il était nécessaire de parcourir le polygone, mais au lieu de suivre une phase déterminée par le temps, il fallait suivre un autre bateau naviguant sur la même trajectoire. Pour ce faire, une communication TCP entre les bateaux et une communication UDP entre le bateau et le serveur ont été préétablies. Le bateau 16 envoyait aux autres bateaux sa phase, sa latitude et sa longitude, tandis qu'il envoyait au serveur son identifiant (ID), sa longitude, sa latitude et le segment j actuel du trajet.

5.2 Éléments de théorie

La logique du code a été établie de la manière suivante : nous avons défini le nombre de bateaux présents dans l'essaim (swarm) — les meilleurs résultats pratiques ayant été obtenus avec 2 bateaux — et la distance que le bateau devait maintenir par rapport au leader. L'identification du bateau à suivre a été réalisée en configurant l'IP fixe du bateau voisin (dans ce cas, le bateau 06, IP 172.20.25.206) pour établir une connexion point à point via TCP.

Le calcul de la phase actuelle de notre propre bateau (ϕ_{actuelle}) a été implémenté en associant le pourcentage de la distance parcourue sur le polygone à un cycle complet de 0 à 2π :

$$\phi_{\text{actuelle}} = \left(\frac{d_{\text{parcourue}}}{d_{\text{totale}}} \right) 2\pi \tag{7}$$

Le déphasage de départ pour chaque robot est calculé en fonction de la variable i_{boat}, qui représente l'indice d'identification du bateau au sein de l'essaim (par exemple, 0 pour le leader, 1 pour le premier suiveur, etc.). Cette variable permet de définir la position relative nominale ($\bar{\phi}$) de chaque agent sur le polygone par rapport au nombre total de bateaux N :

$$\bar{\phi} = \frac{i_{\text{boat}}}{N} 2\pi \tag{8}$$

Lors de la réception de la phase du bateau leader (ϕ_{leader}) via la connexion TCP, il est nécessaire de calculer l'erreur de phase (e_ϕ) pour alimenter la commande du moteur. Pour garantir que le bateau suiveur maintienne la distance angulaire requise par rapport au bateau de devant, on soustrait le terme $\frac{2\pi}{N}$ à la différence des phases. Pour éviter les valeurs négatives et gérer les discontinuités de cycle, le calcul de l'erreur intègre une opération modulo (agissant comme la fonction sawtooth) :

$$e_\phi = \left(\phi_{\text{leader}} - \phi_{\text{actuelle}} - \frac{2\pi}{N} \right) \pmod{2\pi} \tag{9}$$

Contrairement aux missions précédentes, où la vitesse de base était strictement constante, dans la mission d'essaim (swarm), l'erreur de phase e_ϕ agit directement sur la loi de com-

mande de la vitesse linéaire, fonctionnant comme un contrôleur Proportionnel. La puissance de base envoyée aux moteurs (PWM) est ajustée dynamiquement par l'équation :

$$\text{PWM}_{\text{base}} = \text{PWM}_{\text{nominal}} + K_{p,\phi} \cdot e_{\phi} \tag{10}$$

Où $\text{PWM}_{\text{nominal}} = 170$ et le gain proportionnel $K_{p,\phi} = 50$. Le résultat numérique final subit une saturation (clipping) pour rester contenu dans l'intervalle de 100 à 255. Cette saturation garantit que le bateau ne s'arrête pas complètement (ce qui lui ferait perdre sa manœuvrabilité) s'il est trop en avance, et qu'il n'essaie pas de dépasser la limite physique de puissance des moteurs s'il est en retard.

Enfin, la commande de cap (heading), responsable du maintien physique du bateau sur la ligne du polygone, n'a subi aucune modification et est restée identique à celle des missions 2 et 3, utilisant l'approche des champs de vecteurs et la fonction sawtooth pour la convergence de la trajectoire.

5.3 Résultats

La figure 7 illustre le test réalisé en suivant le bateau numéro 6. Pour cette expérimentation, l'objectif était de suivre exactement le leader, c'est-à-dire sans déphasage (offset) de phase, afin de vérifier la capacité du contrôleur à maintenir la formation.

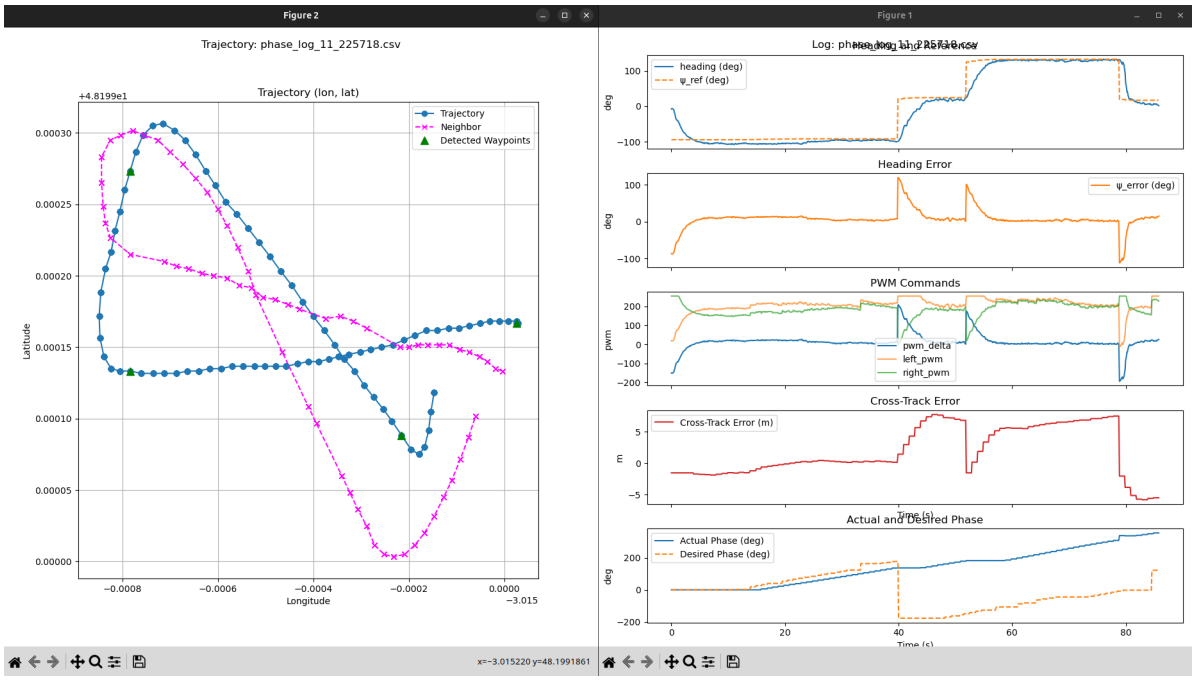


FIGURE 7 – Suivi du bateau 6 sans déphasage