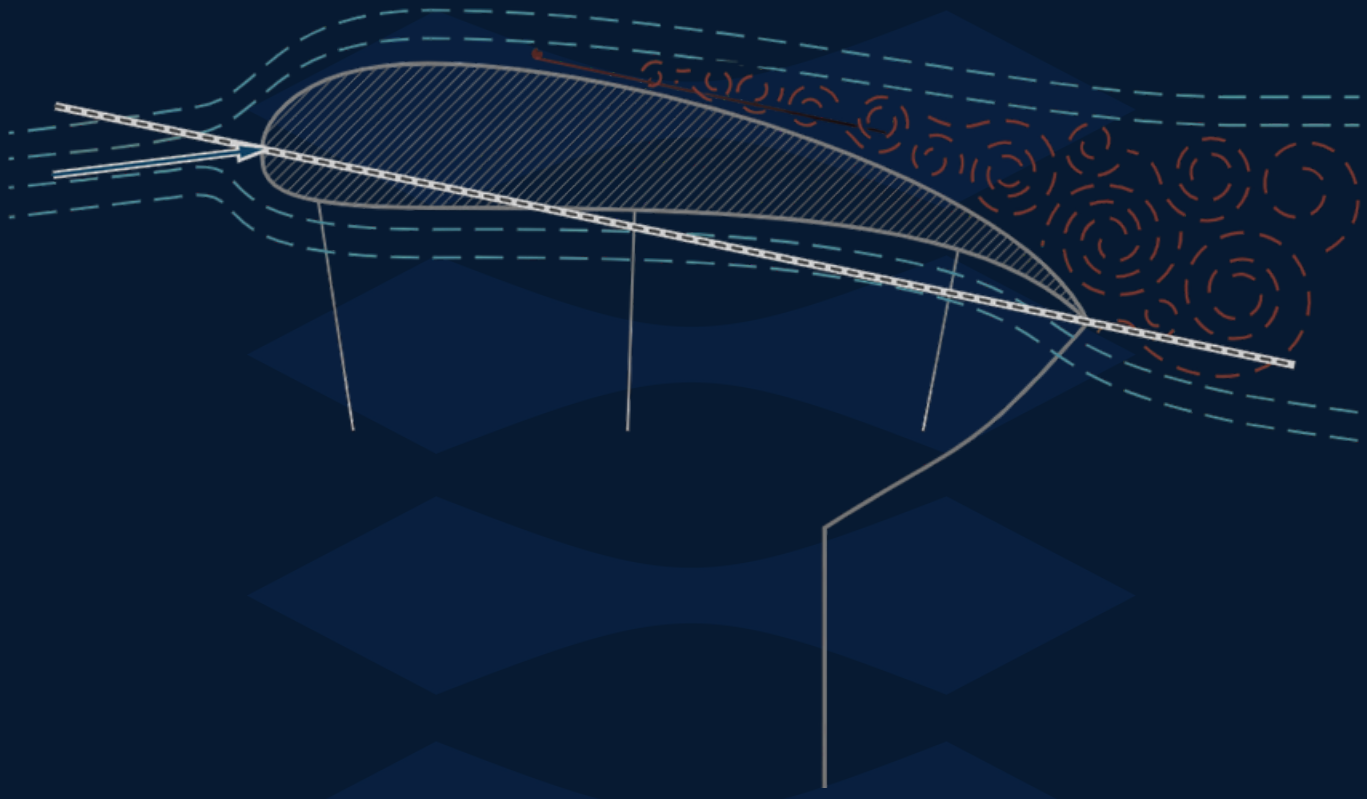
# Rapport de stage 2A

## Développement d'un paramoteur autonome et analyse de ses applications stratégiques

Hippolyte Lafrad



Stage effectué du 5 Mai 2025 au 5 Septembre 2025 dans le laboratoire Mars Lab au Danemark
Etablissement initial : Ecole Nationale Supérieure de Techniques Avancées - ENSTA
Etablissement d'accueil : Denmark's Technical University - DTU
Domaine : Robotique autonome, aéronautique
Elève : Hippolyte Lafrad
Tuteurs : Pr.Fumagalli, Georgios Stamatopoulos

# Table des matières

## Sources externes

Chaîne youtube SCOUT aviation
Paramotor Geometry Classroom
Paramotor theory of flight
Paramotor simulator Chris Leaver
https://discuss.px4.io/t/how-to-change-the-vehicle-shape-airframe-in-jmavsim/38915
PX4 Autopilot — Flying wing and TECS tuning
Contribution communautaire PX4 sur un modèle de paramotor par junwoo0914
QGroundControl User Guide
MAVROS Documentation
Flugschule-oberbayern.de

## Avant-propos

Ce rapport présente les travaux réalisés lors de mon stage consacré à l'autonomisation d'un paramoteur, menés en collaboration avec Georgios et sous la tutelle du professeur Fumagalli. L'objectif principal de ce stage était d'étudier, simuler et expérimenter un véhicule aérien autonome original, combinant la portance d'une voile de parapente et la propulsion d'un moteur léger, afin d'évaluer ses performances et ses limites, tant en matière de vol que de contrôle. Ces travaux m'ont permis d'appréhender la complexité du pilotage d'un aéronef dont la dynamique diffère significativement de celle des drones classiques à voilure fixe ou des quadrotors.

Au cours de ce stage, j'ai analysé ce nouveau type d'aéronef, dans ses dimensions physiques et logicielles. J'ai mobilisé mes compétences en Python pour modéliser et commander le drone, mis en place un environnement de simulation complet, créé un modèle de véhicule et paramétré ce modèle pour un autopilote. J'ai également renforcé mes aptitudes en modélisation de pièces 3D. Ce stage, mené de manière largement autonome, m'a offert une expérience riche en recherche de solutions, en résolution de problèmes, en réparation de matériel et en improvisation.

Ce rapport retrace l'ensemble de ces travaux, depuis l'étude initiale du paramoteur et la simulation de son comportement jusqu'au réglage de l'autopilote, aux essais en vol et aux adaptations matérielles, afin de proposer une vision complète du potentiel et des limites d'un paramoteur autonome.

Je tiens à remercier Georgios Stamatopoulos pour son encadrement et son accompagnement durant la première partie de ce stage. Ses conseils et son expertise m'ont permis de mener à bien l'étude, les simulations et les essais du paramoteur autonome.

# Liste des figures

# I. Etude

Dans cette partie, nous présentons le fonctionnement du paramoteur autonome. Nous décrivons d'abord ses composants et les forces aérodynamiques qui agissent sur lui, puis nous abordons la simulation dynamique développée pour modéliser son comportement en vol et optimiser son autonomie.

## I.A) Analyse du véhicule

Figure 1 : Image du Para-RC glider

Le paramoteur autonome étudié est constitué de plusieurs éléments principaux qui assurent son fonctionnement. À l'avant, une cabine abrite la batterie et sert de support aux différents systèmes embarqués. Deux servomoteurs actionnent les bras du dispositif, permettant de contrôler directement le freinage et les virages. La propulsion est assurée par un moteur à courant continu entraînant le rotor, tandis qu'une voile de type parapente fournit la portance nécessaire au vol. L'ensemble est complété par des capteurs et une électronique embarquée comprenant une liaison radio, un autopilote PX4, une centrale inertielle (IMU) ainsi qu'un GPS.


Le comportement du paramoteur en vol est régi par quatre forces principales. La portance, générée par la voile, s'oppose au poids du système. La poussée, produite par le moteur, permet de compenser la traînée aérodynamique qui tend à freiner l'avancée. L'équilibre est atteint lorsque la portance égale le poids et que la poussée est supérieure ou égale à la traînée.

Figure 2 : Schéma des forces principales s'appliquant au paramoteur

$$L = (1/2) \cdot \rho \cdot V^2 \cdot S \cdot C$$

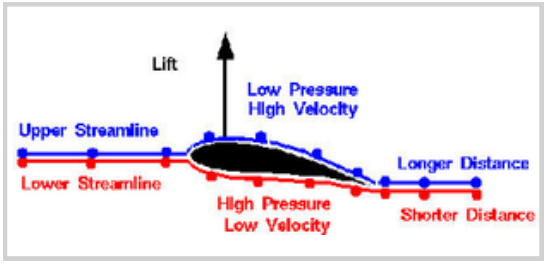
L : portance (N) ; ρ : densité de l'air (kg/m³) ; V : vitesse relative de l'air (m/s) ; S : surface alaire (m²) ; C : coefficient de portance (dépend du profil de l'aile et de l'angle d'attaque)

$$D = (1/2) \cdot \rho \cdot V^2 \cdot S \cdot C$$

D : traînée (N) ; C : coefficient de traînée

$$T \geq D$$

T : poussée




L'image ci-contre montre le profil d'une voile de paramoteur : l'air, plus rapide et à basse pression sur l'extrados (courbure supérieure), crée une portance. Sur l'intrados (face inférieure), l'air est plus lent et à haute pression. Cette différence de pression assure la sustentation de l'aile.

Figure 3 : Explication de la force de portance sur un profil de voile

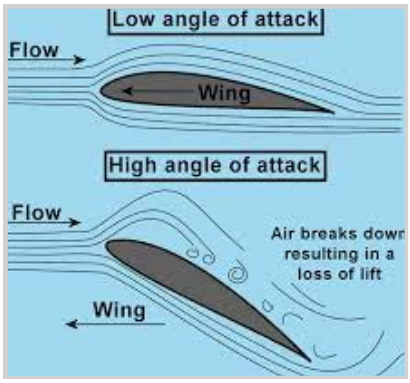



Figure 4 : Procédure de descente

Sur l'image ci-contre, on peut voir le procédé de descente du paramoteur qui, en changeant l'angle d'attaque, diminue sa portance, facilitant le travail de la force de pesanteur.
À ces forces s'ajoutent des effets dynamiques particuliers liés à la configuration du paramoteur. La rotation rapide du rotor engendre un effet gyroscopique qui contribue à stabiliser l'appareil, mais qui peut compliquer certaines manœuvres. Ces effets sont visibles sur l'image ci-dessous, les deux lignes rouges ne sont pas parallèles comme elles devraient l'être.

Figure 5 : Photo illustrant l'effet gyroscopique

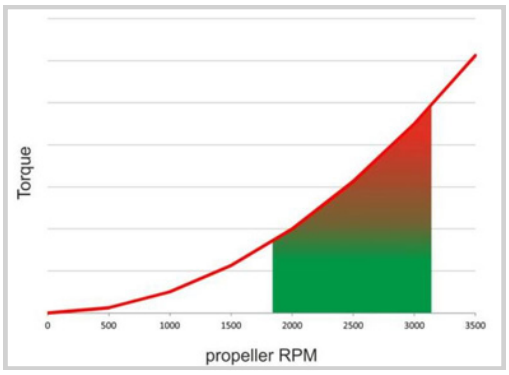



Figure 6 : Evolution du couple parasite selon les tours par minute

De plus, le rotor génère un couple parasite : selon la loi d'action-réaction, si celui-ci tourne dans le sens trigonométrique, la structure du paramoteur tendra à pivoter dans le sens horaire. Ces phénomènes doivent être pris en compte pour assurer la stabilité et le contrôle en vol. Sur le graphe ci-contre, on voit bien l'évolution du couple en fonction des tours par minute de l'hélice.

## I.B) Simulation dynamique

Afin de mieux comprendre le comportement du paramoteur, une simulation dynamique a été développée en Python, à l'aide des scripts *paramotor_simu.py* et *paramotor_simu_2.py*. Celle-ci permet de modéliser les forces en jeu et d'observer l'évolution de la trajectoire en fonction des paramètres aérodynamiques et des conditions de vol. Le graphe est disponible sur la page suivante.

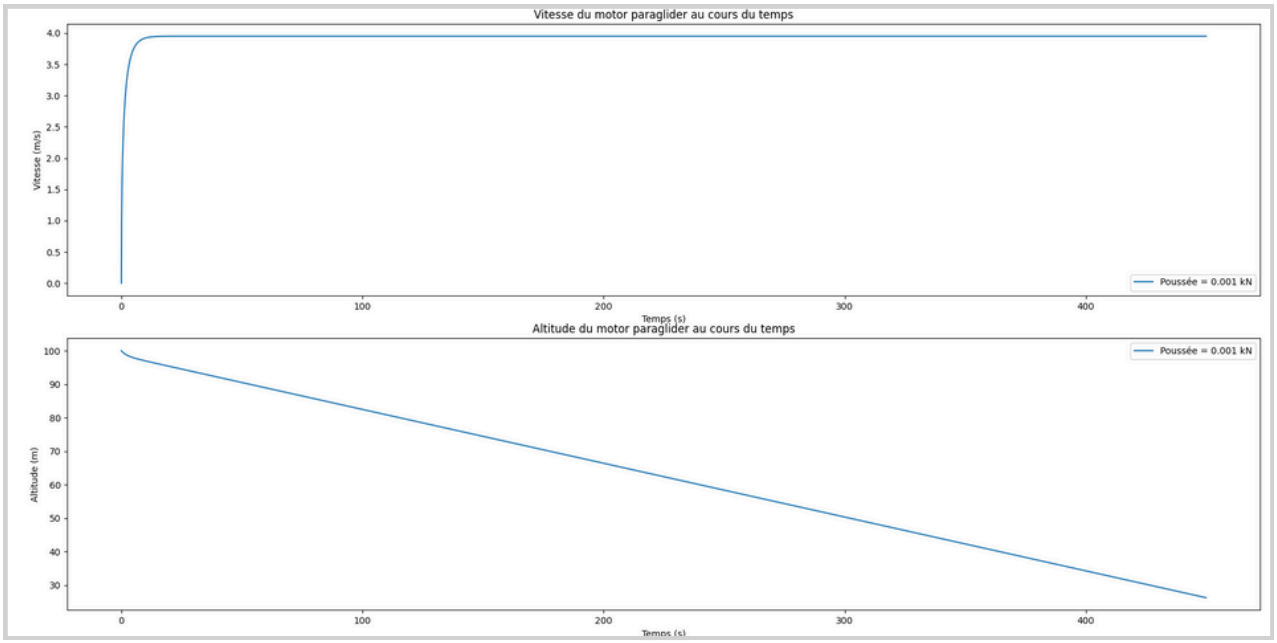



Figure 7 : Simulations dynamique de la vitesse (en haut), puis de l'altitude du paraglider en fonction du temps

Or l'endurance d'un paramoteur peut être augmentée en exploitant les phénomènes naturels qui prolongent la sustentation. Les ascendances thermiques, générées par des colonnes d'air chaud qui montent depuis le sol, et les ascendances dynamiques, produites par des obstacles tels que des falaises ou des montagnes, constituent des opportunités énergétiques intéressantes.

La méthodologie suivie s'appuie sur le travail de Chris Leaver, qui avait développé une simulation Python pour le vol libre en parapente. Cette base a été adaptée aux spécificités du paramoteur. Les fonctionnalités non pertinentes, telles que la modélisation des phases de descente ou certains modules audio, ont été supprimées. En revanche, un modèle de courants ascendants thermiques a été ajouté, afin de rendre le comportement en vol plus réaliste et de simuler différentes stratégies de pilotage. Cette adaptation permet ainsi de tester dans quelle mesure l'utilisation de ces ascendances peut prolonger l'autonomie et améliorer l'efficacité globale du système.

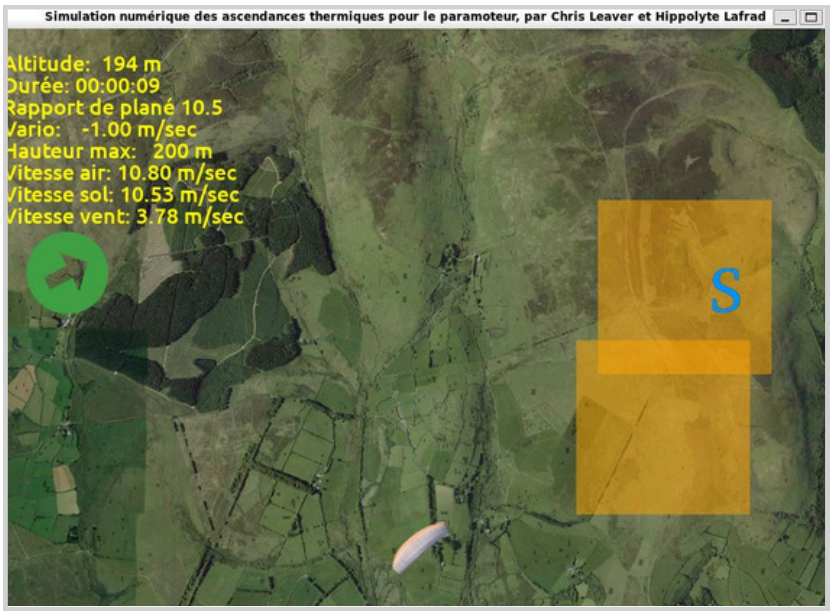



Figure 8 : Capture d'écran d'une simulation dynamique du paramoteur 2D simulant l'impact du vent et des ascendances thermiques

# II. Télé opération

Dans cette section, mettons en place un système de télé opération du drone. Nous procédons à la configuration du contrôle à distance du paramoteur réel à l'aide d'une télécommande, en vue de réaliser un premier vol.

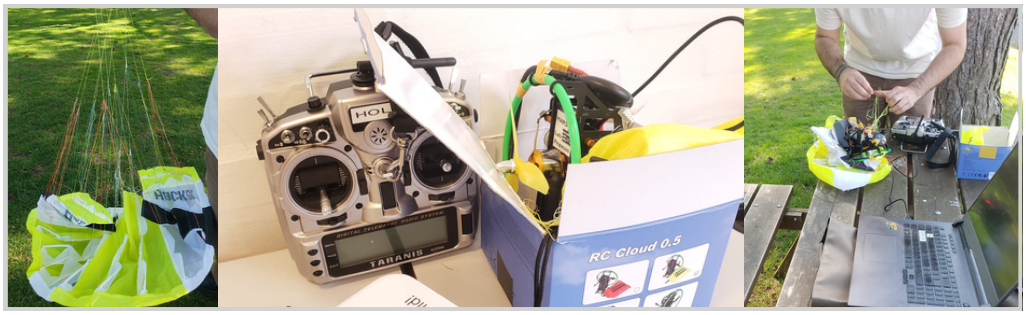



Figure 9 : photos prises lors de la télé opération

On commence par coupler le contrôleur de vol PX4 avec la télécommande. Pour cela, le récepteur de la télécommande, de type FrSky, est branché au contrôleur de vol en utilisant le protocole SBUS. Ce protocole est ensuite configuré dans QGroundControl via le menu « Radio Setup ». Une fois la radiocommande allumée, il est nécessaire de vérifier qu'elle est correctement appairée avec le récepteur, ce qui est indiqué par une petite diode. Les différentes fonctions sont alors assignées, de manière similaire à la configuration de touches dans un jeu vidéo.

Il est ensuite important de vérifier le fonctionnement du GPS intégré au PX4. La diode associée, située sur le dessus du drone, doit s'allumer en vert, et la position du drone doit apparaître correctement dans QGroundControl. Nous avons rencontré des problèmes de communication avec les satellites, qui ont nécessité de nous déplacer dans un espace dégagé orienté vers le sud pour obtenir un signal correct.

Le lancement du drone ne peut pas se faire n'importe où. Étant donné sa vitesse et sa sensibilité au vent, il existe un risque de perte de contrôle ou de blessure. Nous nous sommes donc déplacés dans un espace ouvert, sans personne autour. Avant le décollage, il faut également démêler soigneusement les fils de la voile, ce qui est une opération longue et fastidieuse.

Dans avons réalisé un premier vol sans encombre. Le drone répond correctement à la télécommande, mais il y a cependant des défauts dans la trajectoire suivie et des oscillations parasites du pitch. On reparlera de ce soucis plus tard. Lors du second lancement, le drone a décollé correctement, mais un coup de vent a fait basculer le parachute, entraînant une chute au sol et l'endommagement de l'un des bras. Cette expérience a mis en évidence la difficulté et la précision requise au lancement d'un paramoteur autonome en plus du temps nécessaire à sa mise en place.

# III. Architecture

Dans cette section, nous présentons et détaillons les étapes de création du modèle PX4 ainsi que la mise en place de la simulation du paramoteur. Nous aborderons ensuite les protocoles de communication établis permettant l'exécution de commandes et de programmes sur le drone. Ces graphes ont été réalisés par Georgios Stamatopoulos.

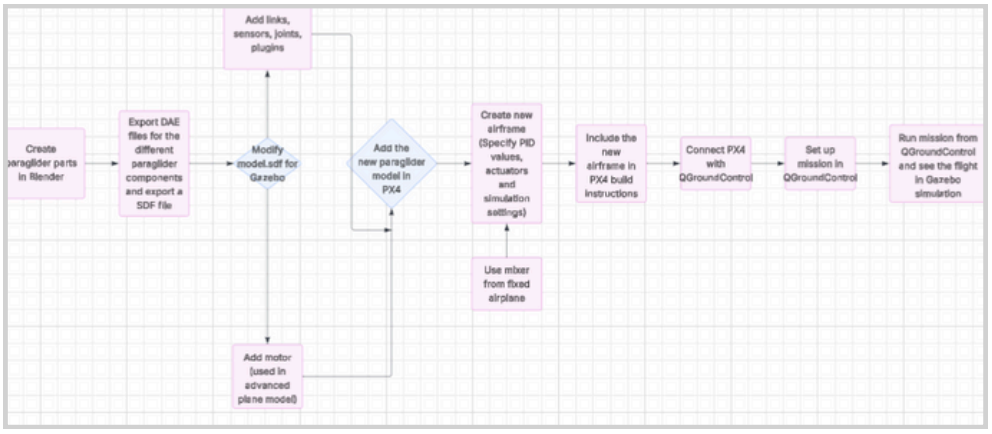



Figure 10 : Schéma organisant les étapes fondamentales pour la mise en place de notre environnement de simulation, par Georgios Stamatopoulos

Le schéma ci-dessus retrace les étapes à suivre pour créer un modèle de paramoteur, puis réaliser une première simulation via PX4 et QGC. Dans les grandes lignes, il faut d'abord modifier un modèle pour Gazebo, puis ajouter ce modèle à PX4. On créera ensuite un nouvel airframe qui sera utilisé dans un environnement complet, détaillé par la suite.

Une autre architecture qui nous intéresse et celle de la communication au sein de notre environnement, notamment grâce aux agent et client Micro XRCE-DDS. Ainsi, pour piloter le drone en lui implémentant des codes, on pourra les écrire en python sous ROS2 et ils seront ensuite communiqués au robot comme suit expliqué sur le schéma à la page suivante. Nos codes sont les *ROS2 nodes*.

Le Micro XRCE-DDS Client tourne sur un microcontrôleur (ou un PC) et échange des données via UDP, TCP ou série. Le Micro XRCE-DDS Agent, sur une machine plus puissante, sert de pont vers un système DDS standard comme Fast DDS. L'Agent attend la connexion d'un Client, qui déclare ses entités DDS et publie périodiquement des messages. L'Agent reçoit ces messages et les retransmet sur le réseau DDS.

Afin de permettre à notre paramoteur d'être pleinement autonome, il va ainsi nous falloir acquérir les outils présentés dans ces schémas et les articuler correctement selon ces architectures.

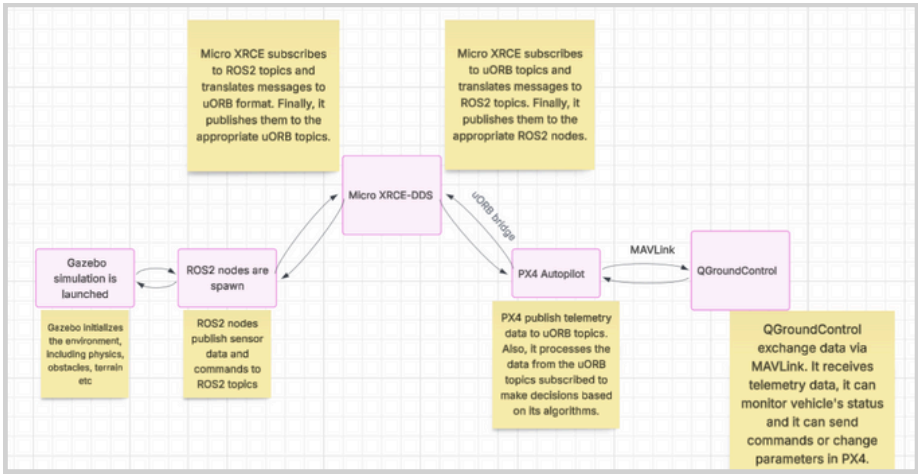



Figure 11 : Schéma résumant les communications entre les divers outils numériques, par Georgios Stamatopoulos

# IV. Modèle

Dans cette section, nous créons un modèle fidèle à notre drone. Nous décrivons notre tentative de création d'un modèle PX4 entièrement indépendant, avant de présenter la stratégie finale consistant à adapter un modèle préexistant. Nous détaillerons ensuite les différents paramètres définissant notre modèle final.

## IV.A) Création d'un nouveau modèle

Afin de simuler le paramoteur, nous avons tenté de créer un nouveau véhicule dans PX4. Pour cela, un nouvel airframe intitulé *3034_paramotor* a été ajouté dans le répertoire *PX4-Autopilot / ROMFS / px4fmu_common / init.d / airframes*. Parallèlement, un fichier de configuration spécifique, *paramotor.mix*, a été créé dans le dossier *PX4-Autopilot / ROMFS / px4fmu_common / mixers*.

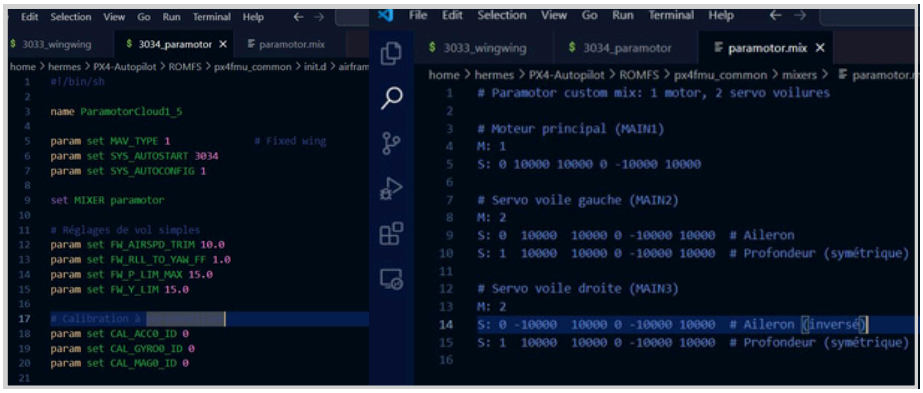



Figure 12 : Capture montrant le paramétrage du nouveau modèle (à gauche) et création du mixer associé (à droite)

Une vérification du paramètre *SYS_AUTOSTART* a ensuite été effectuée. Dans QGroundControl, l'indice du véhicule utilisé est bien affiché, confirmant la prise en compte du nouvel airframe. Cependant, un problème est rapidement apparu : d'après les échanges consultés sur le forum *discuss.px4*, le simulateur *jmavsim* ne prend en charge que les configurations de type quadrirotor. Cette limitation a été confirmée par des recherches complémentaires. Il est donc nécessaire de remplacer *jmavsim* par *Gazebo Classic*, qui propose un choix plus large de modèles de véhicules et permet ainsi d'intégrer un paramoteur personnalisé dans la simulation.

## IV.B) Modification d'un modèle préexistant

Plutôt que de créer un nouveau véhicule de A à Z, ce qui s'avère particulièrement complexe en termes de paramétrage, j'ai choisi de modifier un modèle déjà existant. Sur les conseils de mon tuteur de stage et d'après les échanges relevés sur le forum discuss.px4, le choix s'est porté sur un drone de type generic flying wing, équipé de deux élevons et d'un moteur.

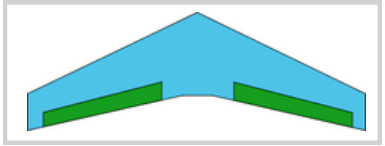


Figure 13 : Schéma QGC du modèle flying wing

Ce modèle de base doit toutefois être adapté pour correspondre aux caractéristiques d'un paramoteur. Pour cela, j'ai décidé d'utiliser le simulateur *Gazebo Harmonic*, en remplacement de *Gazebo Classic*, après avoir déjà testé les trois simulateurs courants disponibles. Mon maître de stage, Georgios, m'a fourni le modèle 3D de l'aéronef au format *.dae*, qu'il avait lui-même créé. Nous avons travaillé ensemble sur le fichier de paramètres du paramoteur, en prenant comme point de départ la configuration d'un aéronef de type *flying wing*.
Les paramètres modifiés sont regroupés dans le fichier *22001_gz_paraglider_[bêta]*. Le modèle *.dae* correspondant, intégré dans Gazebo Harmonic, est associé à l'indice 22001. Ce même indice apparaît bien dans QGroundControl, confirmant que le véhicule configuré, le *gz_paraglider*, est correctement reconnu.

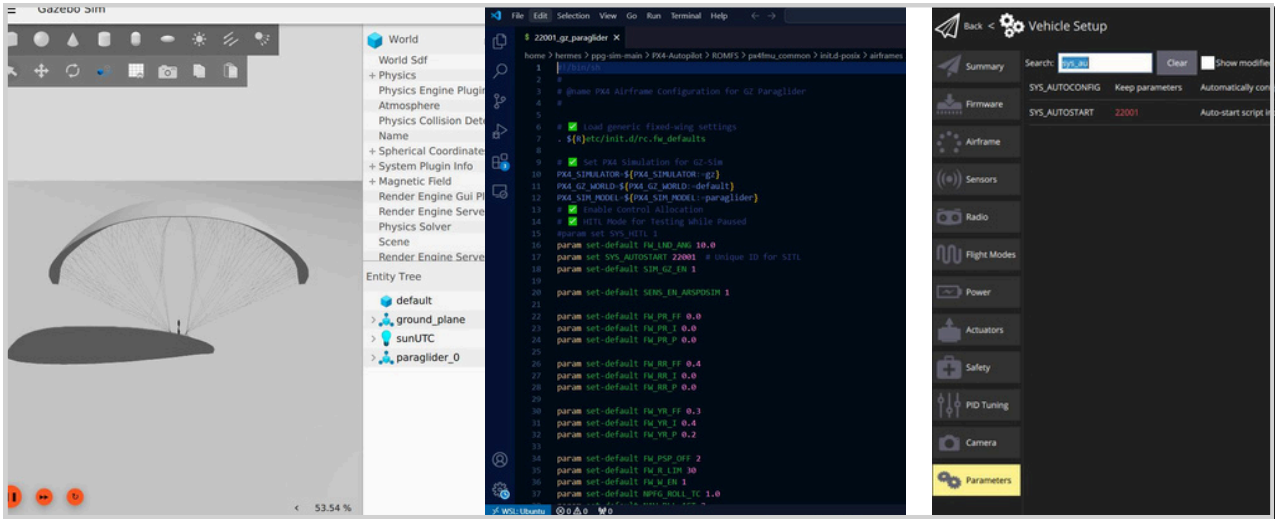



Figure 14 : Captures d'écran de Gazebo (gauche), du modèle 22001 créé (centre), du vehicle setup QGC (droite)

Grâce à cette configuration et à l'autopilote PX4, il est désormais possible de simuler le comportement du paramoteur directement dans QGroundControl.

## VII.C) Paramétrage du modèle

La solution envisagée consiste à modifier directement le modèle du flying wing dans ses paramètres afin de le rendre fidèle à sa nouvelle réalité physique de paramoteur. On précise que certains de ces paramètres ont été ajouté pendant des corrections de vol, et qu'ils n'étaient pas tous correctement définis au départ.
Par défaut, le contrôleur PX4 tente de réguler le roulis, le tangage et le lacet indépendamment. Or, sur un paramoteur, le contrôle latéral se fait uniquement par les freins gauche et droit : tirer à gauche entraîne simultanément un roulis et un lacet vers la gauche, et tirer à droite produit l'effet inverse. L'instabilité observée pourrait donc provenir du fait que PX4 tente de contrôler séparément le roulis et le lacet, ce qui ne correspond pas à la dynamique réelle de l'aéronef.
Pour stabiliser le système, nous avons choisi de supprimer le contrôle direct du lacet, de sorte que toute consigne en yaw soit convertie en commande de roulis. Ainsi, le lacet devient une conséquence naturelle du roulis et du virage induit. Nous nous sommes basés sur les paramètres d'un modèle de paramoteur pour PX4 proposé par junwoo0914 sur le site officiel.

Le réglage du contrôle de roulis a été modifié pour n'utiliser que le contrôle de type feed-forward : la commande appliquée aux freins est directement proportionnelle à l'erreur de vitesse de roulis, sans régulation PID classique.
Le terme intégral a été désactivé (FW_RR_I = 0) afin d'éviter l'accumulation d'erreurs pouvant générer une réponse lente ou instable. Le terme proportionnel a également été désactivé (FW_RR_P = 0) pour limiter les fortes oscillations liées à des erreurs importantes, compte tenu de la sensibilité du véhicule. Le feed-forward est activé avec un gain de 0,5 (FW_RR_FF = 0.5), ce qui permet une réponse plus rapide et prédictible, directement calculée à partir de l'erreur de vitesse de roulis.
Enfin, pour sécuriser le vol, l'angle de roulis en mode autonome a été limité à ±30° (FW_R_LIM = 30), garantissant ainsi des inclinaisons modérées et un vol plus stable pour le paramoteur. Le schéma ci-dessous illustre les nouveaux paramètres.

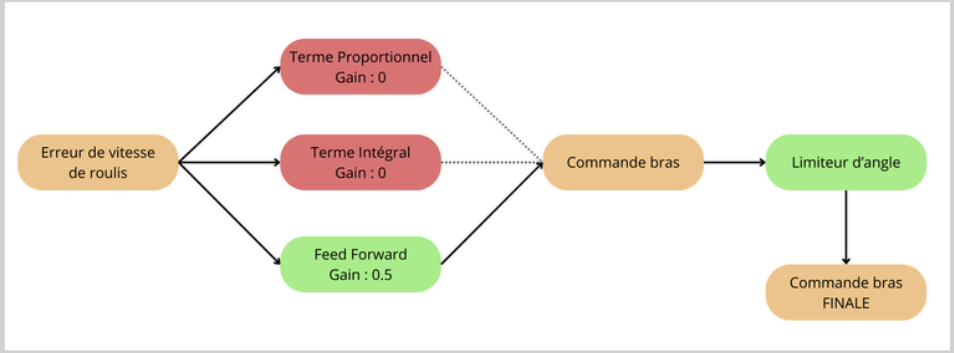



Figure 15 : Schéma illustrant le réglage du contrôle de roulis

Le paramoteur n'a pas pour priorité de contrôler son tangage. Pour cette raison, nous avons désactivé complètement le contrôle de l'axe de tangage en réglant les paramètres suivants : le feed-forward pour le tangage (FW_PR_FF) est mis à 0, de sorte qu'aucune commande prédictive n'est appliquée ; le terme intégral (FW_PR_I) est également à 0, empêchant l'utilisation de l'erreur accumulée pour corriger le mouvement ; enfin, le terme proportionnel (FW_PR_P) est désactivé, de sorte qu'aucune correction immédiate de l'écart de tangage ne soit effectuée. Les termes dérivés étant déjà à 0 par défaut dans PX4, il n'a pas été nécessaire de les modifier.
En résumé, le système ne tente plus de corriger automatiquement le tangage, laissant cette régulation au pilote ou aux forces naturelles du vent. Cette approche évite que le contrôle automatique rende le vol instable ou contre-intuitif, en particulier à basse vitesse. Néanmoins, le tangage continue d'influencer la direction de la poussée et les variations dues au vent, ce qui peut expliquer l'effet montagne russe observé lors des vols par la suite. Le schéma ci-dessous illustre les nouveaux paramètres.

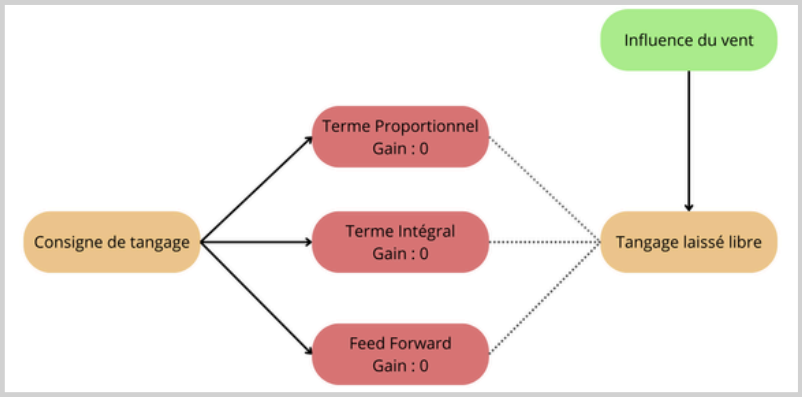



Figure 16 : Schéma illustrant le réglage du contrôle de tangage

# V. Simulation

Dans cette section, nous mettons en place une simulation réaliste et efficace du drone, permettant de tester ses performances sans contrainte de temps, conditions environnementales ou risque de dommages matériels. Nous avons ensuite analysé les simulations de vol afin d'étudier les logs générés. Enfin, nous détaillons l'implémentation d'un système de communication avec l'utilisateur.

## V. A) Mise en place de l'environnement

Avec Georgios, nous avons choisi d'utiliser PX4 couplé à QGroundControl. L'installation de PX4 sous WSL2 s'est révélée complexe, mais elle a pu être réalisée avec l'aide d'une documentation détaillée. Pour lancer PX4 en mode simulateur, l'outil jmavsim a été retenu pour sa simplicité..
QGroundControl a ensuite été installé et relié à PX4. Le logiciel s'exécute sous forme d'AppImage. Une fois l'installation terminée, l'interface de QGC affiche l'état « Ready to fly », confirmant la liaison avec PX4. Une première prise en main a permis d'établir un plan de vol et de réaliser une simulation avec un drone de type quadrirotor.
Une fois la connexion faite, un premier plan de vol a été réalisé, visible sur l'image ci-dessous.


Figure 17 : Capture d'écran du premier plan de vol réalisé une fois l'environnement de simulation mis en place.

L'étape suivante a consisté à installer Micro-XRCE-DDS-Agent et Micro-XRCE-DDS-Client. Le rôle de l'Agent est de servir de passerelle entre des systèmes embarqués très contraints (microcontrôleurs, capteurs ou périphériques à faibles ressources) et un réseau DDS classique utilisé par ROS 2. La compilation de ces modules est particulièrement longue (environ 30 minutes) et nécessite de bien vérifier la synchronisation avec le fichier .bashrc. L'Agent a été configuré pour fonctionner en UDP, sur le port 8888, tandis que le Client a été lancé en UDP pour communiquer avec l'Agent. Une modification du fichier CMakeLists.txt a été nécessaire pour ajouter l'exécutable et activer le support UDP du Client, ce qui a imposé de recompiler la bibliothèque Micro-XRCE-DDS-Client avec le profil UDP.

Afin de simplifier l'utilisation, l'installation de gnome-terminal a été effectuée. Ce terminal graphique facilite le lancement de commandes dans un environnement Linux avec interface visuelle. Un script .sh a ensuite été créé afin d'automatiser le démarrage simultané de QGroundControl, PX4 et Micro-XRCE-DDS-Agent. Grâce à ce script, l'ensemble des logiciels nécessaires peut être lancé en une seule étape, ce qui rend l'utilisation beaucoup plus pratique.
Nous avons ainsi obtenu un environnement complet, permettant de simuler le fonctionnement du paramoteur dans QGroundControl avec l'autopilote PX4 correctement configuré.
Georgios réalise un ReadMe pour de potentiels utilisateurs afin d'installer proprement l'environnement. Nous ajoutons aussi les erreurs rencontrées, spécifions certains téléchargements annexes à faire et indiquons des commandes de base dans un bloc-notes à part nommé commandes_utiles.

## V. B) Première simulation

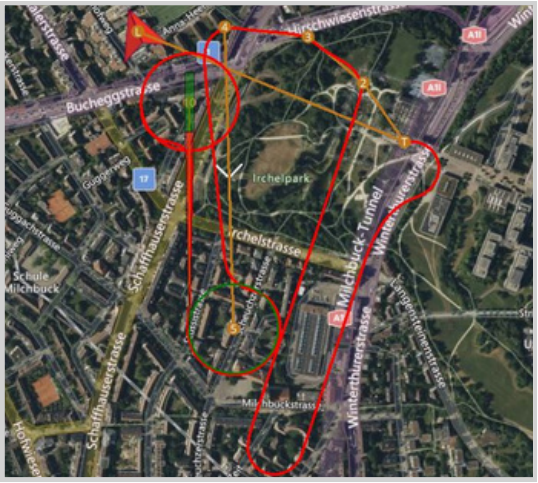
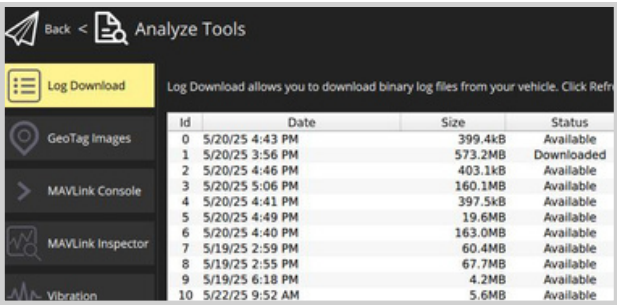



Figure 18 : Capture d'écran de la deuxième simulation initiée

Un premier plan de vol a été défini (en orange), puis le véhicule a été armé et la simulation lancée. La trajectoire effectivement suivie s'est affichée en rouge. Dans un premier temps, le drone est parti dans une direction incorrecte et a mis un certain temps avant de retrouver le cap prévu. Une fois stabilisé, il a cependant suivi correctement l'ensemble des waypoints planifiés.
Un second essai a été réalisé avec un nouveau plan de vol. Le résultat a été identique : le drone s'est d'abord engagé dans une mauvaise direction avant de corriger sa trajectoire et de suivre fidèlement le plan défini.

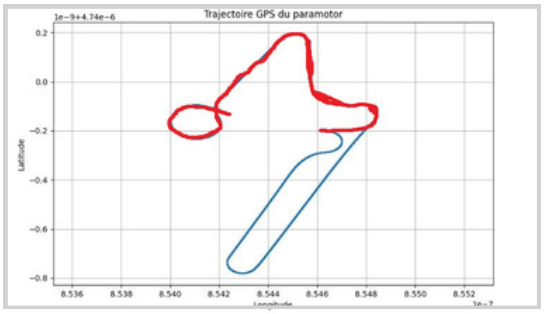
## V. C) Etude des logs

Pour comprendre le comportement observé, nous avons travaillé avec les logs de vol téléchargés depuis QGroundControl. Ces fichiers sont enregistrés au format .ulg. Afin de les lire et de les exploiter, un programme Python a été développé à l'aide de la bibliothèque pyulog. L'exécution est relativement longue en raison de la taille importante du fichier de log, qui atteint près de 500 Mo. Une fois le fichier chargé, la liste des messages disponibles à l'analyse est accessible.




| Id | Date | Size | Status |
|---|---|---|---|
| 0 | 5/20/25 4:43 PM | 399.4kB | Available |
| 1 | 5/20/25 3:56 PM | 573.2MB | Downloaded |
| 2 | 5/20/25 4:46 PM | 403.1kB | Available |
| 3 | 5/20/25 5:06 PM | 160.1MB | Available |
| 4 | 5/20/25 4:41 PM | 397.5kB | Available |
| 5 | 5/20/25 4:49 PM | 19.6MB | Available |
| 6 | 5/20/25 4:40 PM | 163.0MB | Available |
| 7 | 5/19/25 2:59 PM | 60.4MB | Available |
| 8 | 5/19/25 2:55 PM | 67.7MB | Available |
| 9 | 5/19/25 6:18 PM | 4.2MB | Available |
| 10 | 5/22/25 9:52 AM | 5.6MB | Available |

Figure 19 : Capture du téléchargement des logs depuis QGC, dans l'outil d'analyse




La trajectoire GPS a été tracée afin de vérifier sa cohérence avec celle affichée dans QGC. Les deux sont bien concordantes. Pour mieux visualiser l'écart observé lors des vols précédents, le plan prévu a été représenté en rouge, tandis que la boucle inutile réalisée par le drone avant de corriger son cap est tracée en bleu.

Figure 20 : Trajectoire mettant en valeur l'erreur de cap (en bleu par rapport au rouge)

L'analyse s'est poursuivie en comparant le cap estimé avec les consignes envoyées, à partir des messages vehicle_attitude et vehicle_attitude_setpoint. Le tracé des deux courbes a mis en évidence une erreur de cap initiale : le drone ne s'oriente pas dans la direction commandée au départ. Pour approfondir, le code a été adapté afin d'afficher directement l'erreur de cap. Comme attendu, les écarts observés en fin de courbe sont normaux : le paramoteur tourne plus lentement qu'un drone classique et décrit des virages larges. Globalement, la trajectoire est cohérente, mais l'erreur initiale reste très marquée et constante.

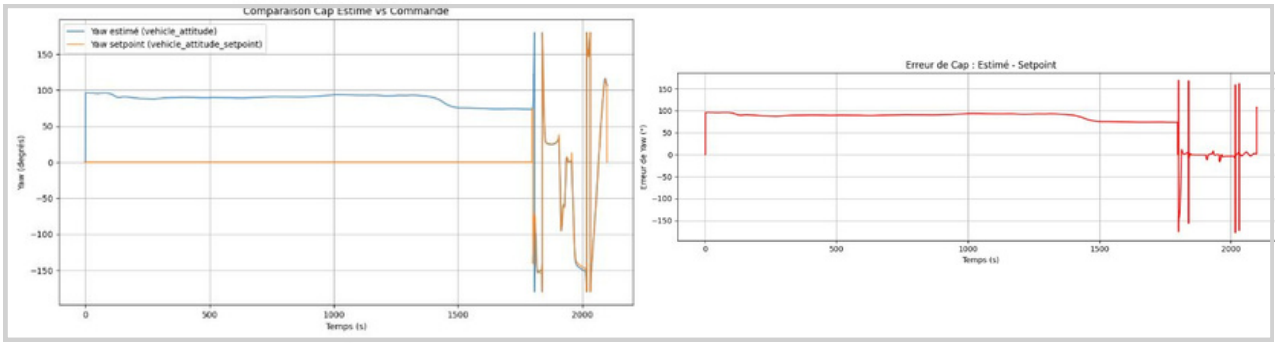



Figure 21 : A gauche, comparaison du cap estimé (bleu) et du cap commandé (orange), à droite, erreur de cap associée (rouge)

Cette observation correspond au moment où l'icône Mission dans QGC passe du jaune au rouge, signalant le passage de la phase d'initialisation à l'exécution effective de la mission. En jaune, le contrôleur de vol a reçu l'ordre mais attend de remplir certaines conditions (comme atteindre une altitude minimale ou rejoindre le premier waypoint). En rouge, la mission est considérée comme réellement lancée. Ce comportement peut s'expliquer par la nature même du paramoteur : contrairement à un quadrirotor qui décolle verticalement, il doit parcourir une certaine distance avant de réellement prendre son cap.

Pour vérifier cette hypothèse, l'altitude a été tracée en fonction du temps. Le changement brusque observé correspond précisément au moment où le drone corrige sa trajectoire, confirmant le lien entre la phase de décollage et l'erreur de cap initiale.

Afin d'améliorer ce comportement, il a été décidé de déclencher automatiquement la mission dès le décollage. Le paramètre d'altitude de take-off, initialement fixé à 82 ft, a été réduit à 10 ft afin que la mission débute plus rapidement. Un nouveau vol a ensuite été réalisé, les logs ont été récupérés et analysés pour vérifier l'efficacité de cette modification.

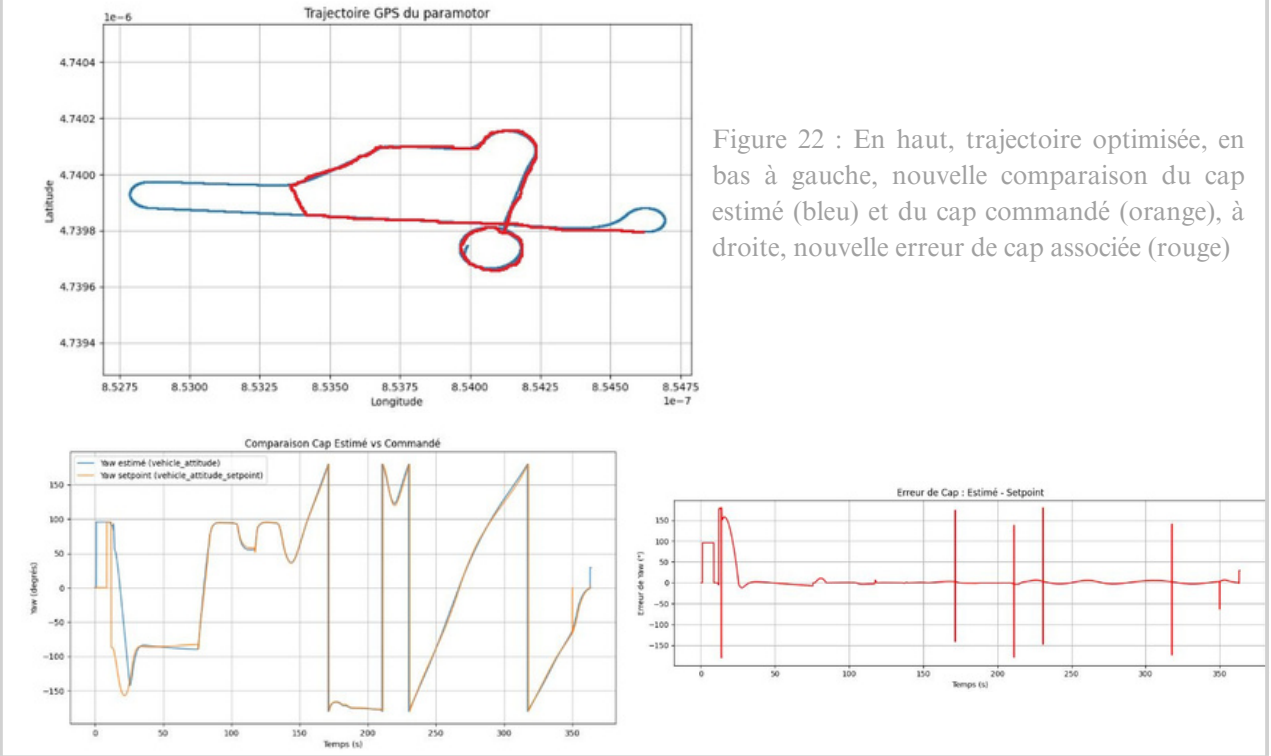



Figure 22 : En haut, trajectoire optimisée, en bas à gauche, nouvelle comparaison du cap estimé (bleu) et du cap commandé (orange), à droite, nouvelle erreur de cap associée (rouge)

La trajectoire obtenue après modification des paramètres est nettement plus cohérente. La comparaison entre le cap commandé et le cap réellement suivi montre une concordance bien meilleure que lors des essais précédents. L'erreur de cap est désormais pratiquement nulle. Les seuls écarts observés apparaissent lors des changements brusques de direction, correspondant à l'atteinte d'un waypoint. Ce comportement est attendu, car l'aéronef ne peut pas virer instantanément et doit décrire des courbes plus larges en raison de sa dynamique de vol.

## V. D) Topics et communications

L'installation de MAVROS2 a été réalisée afin de surveiller les topics en temps réel. Pour cela, les dépendances nécessaires ainsi que les fichiers de configuration d'UEC ont été installés, permettant une connexion correcte avec un autopilote tel que PX4. MAVROS joue le rôle de passerelle logicielle entre ROS (Robot Operating System) et les autopilotes utilisant le protocole MAVLink, comme PX4 ou ArduPilot. Il permet ainsi d'envoyer des commandes de vol et de recevoir en retour des données télémétriques, des positions GPS et d'autres informations de navigation.

Un package ROS2 a ensuite été créé et intégré au fichier *.bashrc* pour être correctement sourcé. Le workspace a été compilé à l'aide de la commande *colcon build*, en veillant à ce que les dépendances liées à *packaging* et *setuptools* soient mises à jour. Comme plusieurs nœuds étaient nécessaires, un fichier de lancement en Python a été rédigé. L'objectif est de pouvoir exécuter une mission commandée depuis QGroundControl en utilisant les librairies *px4_msgs* et *PX4_ros_com*.

La liaison entre le workspace ROS2 et PX4 (associé à QGC) a été vérifiée en lançant MAVROS et en contrôlant son activation via la commande ros2 topic list. Il est alors possible de piloter directement le drone depuis Ubuntu avec ROS2. Une première étape a consisté à armer l'appareil, puis à lancer une mission. L'objectif suivant est de développer un nœud ROS2 complet capable d'envoyer un plan de vol au drone.

L'architecture générale du code se décompose en quatre étapes : initialisation et connexion aux services MAVROS, récupération des données (position GPS, waypoints de décollage existants), génération de la mission, puis exécution via l'upload dans PX4 et l'activation du mode AUTO.MISSION. Un point important a été relevé : il est nécessaire de forcer l'index des waypoints à 0 après l'armement du drone.

Des difficultés ont toutefois été rencontrées : certains waypoints ne s'enchaînent pas correctement et sont transformés en loiter. Ce comportement s'explique par la nécessité pour le drone d'ajuster son altitude avant de poursuivre sa trajectoire, chaque waypoint étant associé à une condition d'altitude.

À ce stade, il est possible de contrôler le véhicule directement depuis l'invite de commande. Les liaisons avec PX4 et QGC sont correctement établies, et les codes Python peuvent être exécutés sous ROS2 sans difficulté, à condition de ne pas oublier de lancer MAVROS2 au préalable.

# VI. Vol en autonomie

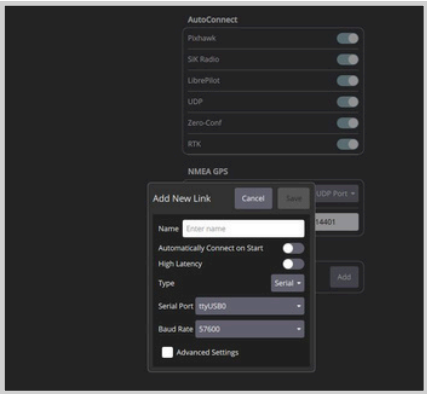
Dans cette section, nous nous intéressons à l'autonomisation du drone. nous présentons les conditions préalables au vol du drone, puis procédons à l'installation et à la configuration des logiciels nécessaires. Nous préparons ensuite le paramoteur et sa voile avant de réaliser le premier vol autonome.

## VI.A) Conditions de vol

Les conditions de vol pour l'utilisation d'un paramoteur autonome doivent être strictement respectées afin de garantir la sécurité et la fiabilité des essais. En premier lieu, il est indispensable d'évoluer dans un environnement dégagé, sans obstacles ni présence humaine à proximité, afin de limiter les risques en cas de perte de contrôle. Les conditions météorologiques jouent également un rôle critique : le vent doit être faible et régulier, car une rafale peut facilement déstabiliser la voile et provoquer un basculement du paramoteur. De même, il est préférable d'éviter les zones urbaines ou couvertes, car la réception GPS y est fortement perturbée. Un espace ouvert, orienté vers le sud et offrant une bonne visibilité satellite, est recommandé pour assurer une localisation précise et stable. Enfin, l'opérateur doit vérifier la qualité du signal radio et l'état de la batterie avant chaque vol, afin d'éviter toute interruption de communication ou panne en cours de mission.

## VI.B) Intégration logicielle

Pour établir la liaison entre l'ordinateur et le drone, nous avons branché un émetteur de télémétrie SiK Telemetry Radio V3 de Holybro. Après installation des pilotes, le port correspondant est renseigné dans QGroundControl.
Nous avons cependant rencontré un problème de détection de l'émetteur via USB. Celui-ci a finalement été reconnu sous Windows à l'aide du gestionnaire de périphériques, dans la section Ports (COM et LPT). Sous Ubuntu, plusieurs méthodes ont été testées. L'utilisation de socat a d'abord été envisagée, mais rapidement remplacée par usbipd, plus rapide et ne nécessitant pas de virtualisation. Après installation manuelle, nous avons identifié le périphérique via PowerShell en mode administrateur : l'émetteur SiK apparaissait alors en tant que USB Serial Converter.




La procédure a ensuite consisté à attacher ce périphérique à WSL2 afin de le rendre accessible depuis Ubuntu. Bien que plusieurs erreurs aient été rencontrées, l'installation de usbipd sous Ubuntu et Windows a permis de résoudre ces difficultés. Après avoir lié le périphérique dans PowerShell et démarré la distribution Ubuntu, l'émetteur SiK est devenu pleinement fonctionnel dans WSL2, ce qui a permis son exploitation dans notre environnement Linux.

Figure 23 : Capture de la création d'un nouveau Link sur QGC pour l'émetteur SiK

La configuration de QGroundControl a alors pu être réalisée en utilisant ce port série. Pour cela, il a été nécessaire de vérifier que l'utilisateur Ubuntu (Hermes) appartenait bien au groupe dialout, qui confère les droits nécessaires sur les ports série. Par précaution, des permissions globales (chmod 0666) ont temporairement été attribuées, garantissant l'accès universel au périphérique. Toutefois, afin d'éviter cette manipulation à chaque utilisation, une règle UDEV permanente a été créée. Les identifiants du périphérique (VID:PID = 0403:6015) ont été obtenus via usbipd list dans PowerShell. Dans Ubuntu, une règle UDEV a été définie pour donner automatiquement les permissions rw à tous les utilisateurs et rattacher le périphérique au groupe dialout.
Grâce à cette configuration, il est désormais possible de commander le drone directement depuis l'ordinateur et de lancer des missions complètes. Comme l'illustre l'image ci-dessous, le drone n'apparaît pas à la position exacte de sa trajectoire planifiée (cercle orange). Pour corriger ce problème, il a fallu sortir afin de capter correctement les satellites situés au sud.

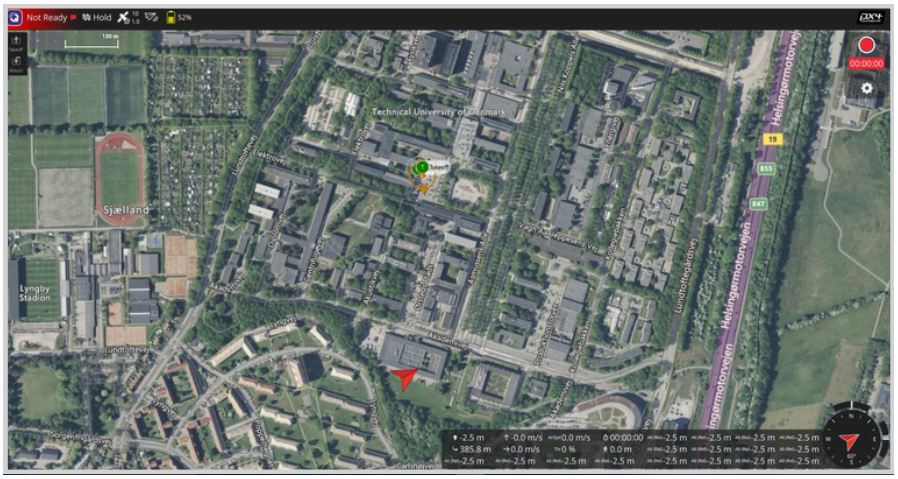



Figure 24 : Capture illustrant les problèmes de réception des données GPS

Pour éviter d'avoir à ouvrir PowerShell et exécuter la ligne de commande à chaque utilisation, il a été décidé de créer un script PowerShell automatisant cette opération. Cependant, un problème est survenu avec usbipd, des fichiers corrompus empêchant son fonctionnement correct. Plusieurs tentatives de réparation ont été effectuées : réinstallation depuis un autre site, redémarrage, installation directe via PowerShell, prise de contrôle manuelle des dossiers Program Files pour obtenir toutes les autorisations, forçage de l'installation dans un autre répertoire, exécution en mode TrustedInstaller et installation d'une version antérieure afin d'éviter les fichiers problématiques.
La dernière tentative a été fructueuse : la version 5.0.0 d'usbipd a été installée avec succès. Au laboratoire, un test a été réalisé avec l'émetteur SiK Telemetry Radio, qui a fonctionné correctement. Le robot a ensuite été relié au PC via la radio, et tous les indicateurs sont passés au vert. Un test moteur a été effectué pour vérifier le bon fonctionnement du système. L'ensemble des aspects logiciels est désormais prêt pour le lancement de la mission.

## VI.C) Préparation du paramoteur

Comme expliqué précédemment dans la partie consacrée à la télé opération, il est nécessaire de préparer soigneusement la voile avant chaque vol. Cette étape est longue et fastidieuse : il faut démêler les fils, couper et renouer ceux qui le nécessitent. Il s'agit d'une phase cruciale pour assurer le succès du vol et la sécurité du paramoteur.
Des tests de communication GPS ont ensuite été réalisés en extérieur, confirmant la bonne réception des signaux. Une fois ces vérifications effectuées, nous nous sommes rendus au parc sélectionné par Georgios, offrant un espace dégagé et sécurisé, sans présence de personnes sur le terrain d'essai.
Lors de la mise en place, nous avons appris que la voile doit être correctement orientée face au vent et bien gonflée avant le lancement du drone. Cette précaution permet d'assurer un décollage stable et efficace.

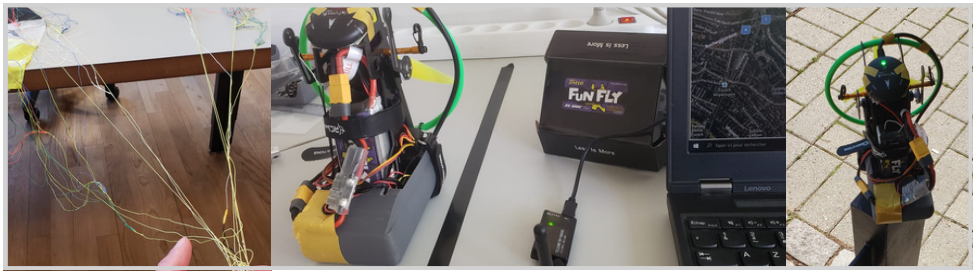



Figure 25 : Photos des fils à dénouer (gauche), de la connexion via SiK (centre) et de la réception GPS par le drone (droite)

## VI.D) Vol autonome

Lors d'un vol en conditions réelles, réalisé avec Georgios, le drone a été programmé pour exécuter une mission en loiter, c'est-à-dire une trajectoire circulaire. Les logs ont été récupérés et sont affichés ci-dessous. Le cercle obtenu n'est pas très propre et même si l'on ne peux pas négliger l'environnement, comme des pertes de données GPS temporaires ou le vent, il faut aussi prendre en compte des soucis au niveau de l'évolution du pitch du drone : les observations initiales ont révélé un mouvement irrégulier, qu'on qualifie d'effet montagne russe.

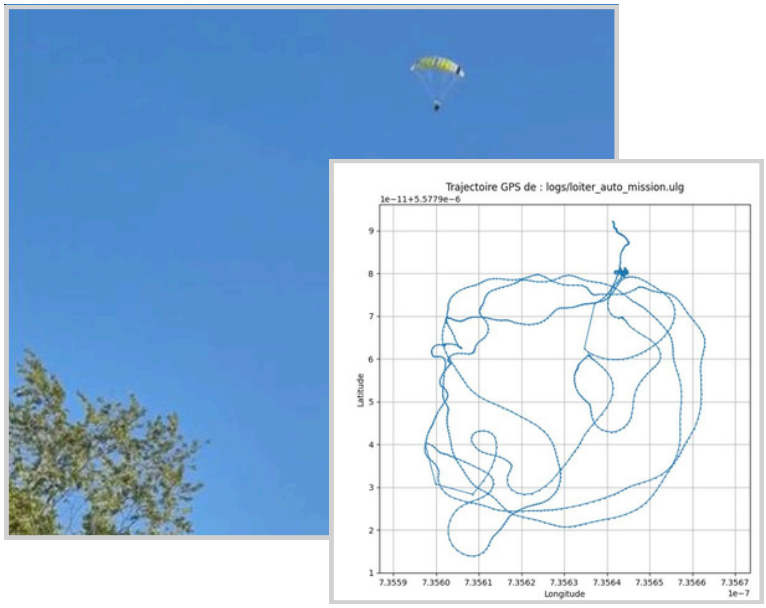



Figure 26 : Photo du paramoteur autonome en vol autonome (derrière), capture de la trajectoire GPS d'une mission loiter du drone


L'objectif était de relancer le drone dans un vol standard, sans correction, afin d'analyser le comportement du paramoteur. Le drone a été pris avec des batteries chargées à 60 %, qui se sont rapidement déchargées en début de vol. Heureusement, la chute qui en a résulté n'a causé aucun dégât matériel. Les batteries ont ensuite été mises en recharge sous la supervision du professeur Fumagalli, qui a rappelé les précautions nécessaires, notamment le bon positionnement des fils dans la broche du chargeur.

Figure 27 : Capture d'écran d'une mission annulée du fait de la chute soudaine de la batterie

La batterie tombée à plat a été considérée inutilisable et éliminée dans une poubelle spécialisée. Cette expérience a mis en évidence la nécessité d'une vigilance accrue lors de la préparation des batteries, car une charge inférieure à 50 % compromet rapidement la sécurité et la durée de vol.

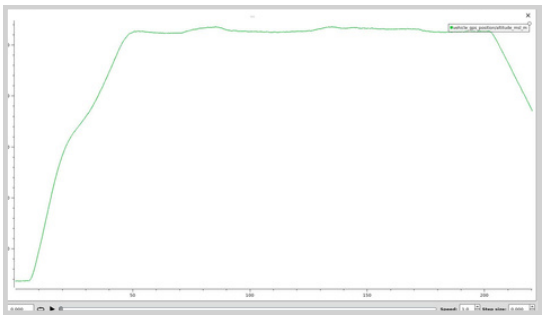
Une fois les conditions climatiques favorables — vent faible et absence de pluie — le drone a été relancé avec une batterie chargée à 90 %. Le second vol s'est déroulé de manière plus satisfaisante. Bien que la trajectoire ne fût pas parfaite, cela était attendu, car le test a été effectué sans modification de la matrice d'efficacité ni ajustement du modèle PX4. Le tracé obtenu avec Python a permis de visualiser la trajectoire. À l'atterrissage, le drone n'a pas arrêté sa course automatiquement, vraisemblablement en raison d'un bug du PX4 signalé par une notification d'erreur dans QGC. Il a donc été ramené manuellement avant d'être coupé.

L'analyse du comportement en vol montre que le paramoteur avance de manière imprécise, avec des déplacements par saccades. Ce phénomène est imputable à plusieurs facteurs : la présence de vent, les caractéristiques structurelles de l'aéronef limitant la précision des virages, et éventuellement un modèle fourni à l'autopilote insuffisamment représentatif de la réalité (notamment concernant le pilotage du pitch). Les mouvements discontinus observés, qualifiés d'effet montagne russe, pourraient également être liés à la matrice d'inertie ou au contrôleur utilisé, ce qui nécessitera une investigation plus approfondie.

# VII. Correction de vol

Dans cette section, nous commençons par identifier le problème rencontré lors du vol, puis nous tentons de le corriger en utilisant la matrice d'efficacité, suivie d'une légère modification du modèle. Il est précisé que toutes les corrections sont effectuées en simulation, afin d'éviter tout risque et de gagner du temps avant un vol autonome.

## VII.A) Identification du problème



Nous avons installé PlotJuggler afin de visualiser et d'analyser les logs de vol. Après avoir téléchargé des logs récents depuis QGroundControl, nous avons pris en main l'interface en affichant, à titre d'exemple, l'évolution de l'altitude.

Figure 28 : Courbe de l'altitude en fonction du temps tracée lors d'une simulation

Il est intéressant de constater que l'altitude simulée dans QGC apparaît relativement lisse. En revanche, lors de l'analyse des logs d'un vol réel du paramoteur, la courbe s'est révélée beaucoup plus instable, avec une allure sinusoïdale marquée lorsque l'appareil tente de maintenir une altitude cible. Cette différence traduit la nécessité d'améliorer la simulation pour qu'elle rende mieux compte des comportements imparfaits de l'appareil, et d'optimiser le contrôleur de vol afin de corriger ces instabilités. La première tâche est revenue à Georgios et la seconde à moi-même.

Afin de corriger le phénomène de soubresauts (non smooth) observé sur le paramoteur réel, nous avons directement exploré les fichiers du répertoire PX4-Autopilot/src/modules. L'objectif était d'identifier les contrôleurs pertinents et de les adapter aux spécificités de notre drone. Le module concerné est le TECS (Total Energy Control System), utilisé à l'origine pour les aéronefs à voilure fixe mais modifié ici pour le paramoteur. Ce contrôleur gère simultanément l'altitude et la vitesse de vol.

```
_update_timestamp = now;

// LAFRAD : On définit les TECS parameters pour un contrôle du pitch plus adapt
_tecs_params.pitch_damp = 0.3f;
_tecs_params.thr_damp = 0.4f;
```

Nous avons ajouté de nouvelles lignes de paramètres _tecs_params afin d'introduire un amortissement supplémentaire du tangage.

Figure 29 : Capture d'une modification des paramètre TECS

Plus précisément, la valeur de pitch_damp a été portée de 0,1 à 0,3. Ce choix, bien que provisoire et expérimental, vise à réduire l'amplitude des variations du tangage et à lisser le comportement en altitude, en ralentissant les oscillations trop rapides.

La modification d'un fichier source nécessitant une recompilation complète, nous avons relancé la construction du firmware. Toutefois, un problème est survenu : l'un des véhicules n'était pas reconnu dans la liste des airframes, ce qui bloquait le processus. Pour contourner cette difficulté, nous avons créé un leurre permettant d'achever la compilation. Enfin, afin d'éviter que ce problème ne se reproduise pour de futurs utilisateurs, nous avons documenté la solution adoptée dans le fichier ReadMe.

## VII.B) Modification de la matrice d'efficacité

Dans l'architecture de PX4, la matrice d'efficacité (Effectiveness Matrix) permet d'ajuster la manière dont les commandes de contrôle sont interprétées et appliquées par le véhicule. Elle définit notamment la réponse des surfaces de contrôle — ailerons, gouvernes de profondeur ou, dans notre cas, les bras du paramoteur — en fonction des entrées transmises par le pilote automatique. À l'inverse, la matrice d'allocation (Allocation Matrix) est utilisée pour répartir ces commandes entre les différents actuateurs disponibles. Elle dépend principalement de la configuration physique du drone et de ses capacités mécaniques, et reste donc moins directement modifiable par l'utilisateur.

Afin d'adapter le comportement du paramoteur, nous avons choisi de modifier la matrice d'efficacité. Pour ce faire, nous avons travaillé dans le fichier ActuatorEffectivenessFlyingWing.cpp. Afin de vérifier que les modifications étaient bien prises en compte, nous avons utilisé l'instruction PX4_INFO, permettant d'afficher directement dans le terminal associé à PX4 les valeurs mises à jour de la matrice d'efficacité.

Malgré nos modifications, le message inséré via PX4_INFO ne s'affiche pas parmi les informations de type "INFO". Nous avons donc entrepris une vérification dans les fichiers de log au format .ulg, en utilisant notre script Python pour analyser leur contenu. Cependant, le message attendu n'apparaît pas non plus dans ces enregistrements. Bien que la recompilation ait été correctement effectuée, il est probable que le code modifié ne soit pas appelé par PX4 pour notre configuration spécifique d'airframe, ce qui signifie que nous avons travaillé sur un fichier non sollicité par le système.

Pour pallier ce problème, nous avons choisi de créer directement un nouveau module dans un répertoire distinct, afin de forcer son appel par PX4. Nous avons donc ajouté le fichier ActuatorEffectivenessParaglider.cpp dans notre dossier src (soit hors de PX4). Afin que ce nouveau contrôleur soit pris en compte, le fichier CMakeLists.txt a été modifié en conséquence pour intégrer la compilation de ce module. La génération du code doit désormais se faire dans le projet ppg, incluant explicitement le contrôleur d'airframe correspondant au paramoteur.

Après avoir compilé dans le projet ppg, nous avons enfin réussi à modifier la matrice d'efficacité, comme montré dans l'image ci-dessous.

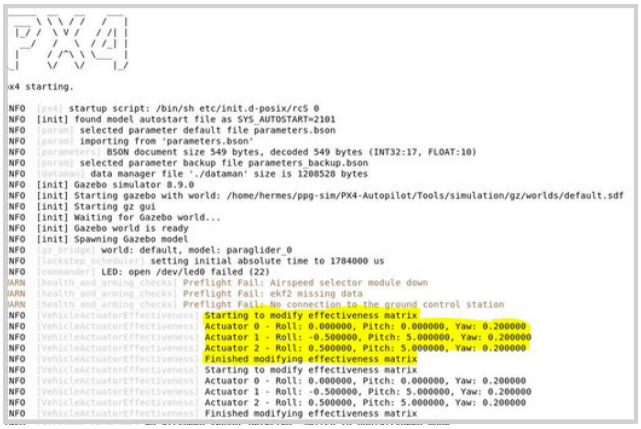



Figure 30 : Fenêtre PX4 ouverte à l'exécution de l'environnement affichant les modifications de la matrice d'efficacité

Un premier test dans QGroundControl a confirmé que ces changements avaient un impact direct sur la trajectoire du paramoteur. Pour en apporter la preuve, nous avons volontairement mis à zéro tous les coefficients liés aux angles d'Euler. Dans ce cas, le drone était incapable de tourner (voir image de gauche ci-dessous). Il visait bien le waypoint suivant mais ne pouvait pas corriger sa trajectoire, se contentant de ralentir et d'attendre un virage qui ne venait jamais. Cette expérience a montré que nous étions désormais en mesure de contrôler efficacement le comportement du véhicule en jouant sur cette matrice. Un second essai sur la matrice d'efficacité à réduit amplement les manœuvres du paramoteur qui a réalisé un circuit aberrant décrit dans l'image de droite ci-dessous).


Figure 31 : Capture de deux simulations avec des modifications drastiques de la matrice d'efficacité

Nous avons ensuite cherché à affiner la commande en nous concentrant sur l'axe de tangage, paramètre essentiel pour corriger l'effet de soubresauts observé lors des vols précédents. Avant d'intervenir de nouveau dans le code, nous avons tenté une approche plus simple via l'interface utilisateur de QGroundControl. En accédant à la section PID Tuning, il est possible de modifier les gains proportionnels, intégrateurs et dérivateurs de chaque axe afin d'optimiser la stabilité et la réactivité. Cependant, cette méthode a montré ses limites : PX4 considère encore le paramoteur comme un aéronef de type flying wing, ce qui fausse l'efficacité du réglage et empêche une correction fine.

Nous avons donc poursuivi avec une étude ciblée sur la modification du pitch dans la matrice d'efficacité. Trois vols d'essai ont été réalisés, avec des coefficients de tangage fixés successivement à 0,5, puis 5, et enfin 0,05. Ces expérimentations, consignées dans les fichiers de log correspondants, permettront d'analyser plus précisément l'influence de ce paramètre sur la dynamique du vol et d'orienter les prochains ajustements.

L'analyse des différents essais a été réalisée avec l'outil PlotJuggler, en visualisant l'évolution du pitch sur les logs de vol. Pour interpréter correctement ces résultats, nous avons choisi de suivre l'altitude de l'aéronef, directement influencée par l'angle de tangage. Ainsi, chaque log a été examiné séparément afin de comparer la réponse en altitude pour les différentes valeurs de pitch appliquées.

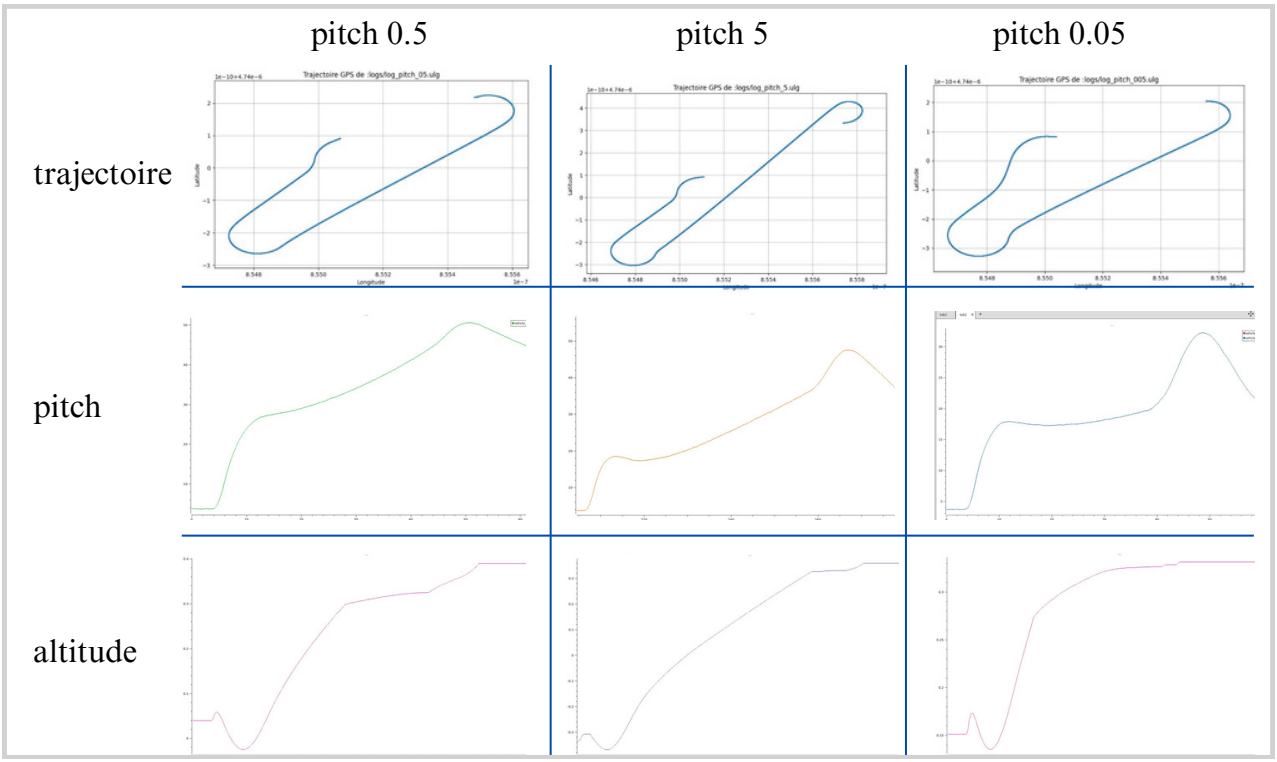



Figure 32 : Tableau récapitulatif des effets de la matrice d'efficacité sur la trajectoire, le pitch et l'altitude

L'analyse des résultats reste peu concluante, la courbe la plus stable correspondant au premier essai. Il serait nécessaire de tester sur le drone réel pour évaluer plus précisément l'impact des modifications de la matrice d'efficacité. Cependant, ces essais sont longs et présentent des risques importants pour la sécurité des personnes et du matériel, notamment lors de vols avec des paramètres non éprouvés.

## VII.C) Modification du modèle

Une nouvelle tentative a été effectuée pour corriger l'effet montagne russe. L'idée initiale de laisser le contrôle du pitch complètement libre, c'est-à-dire de permettre au paramoteur d'osciller au gré du vent, s'est révélée problématique, notamment dans des conditions venteuses comme au Danemark. Ces variations traduisent un effet de rétroaction naturelle de la poussée : lorsque le nez du drone monte ou descend légèrement, la direction de la poussée change, transformant une partie de la poussée verticale en composante horizontale ou inversement. Cela accentue le tangage au lieu de le stabiliser.

Pour atténuer ce phénomène, un léger contrôle sur le pitch a été réintroduit, sans le verrouiller complètement, afin d'amortir la dérive naturelle : les paramètres ont été réglés sur FW_PR_FF = 0.1, FW_PR_I = 0.1 et FW_PR_P = 0. Cette configuration permet de limiter les oscillations tout en conservant une certaine liberté de tangage pour que le comportement du paramoteur reste naturel et adapté aux variations du vent.

# VIII. Réparation

Dans cette section, nous nous concentrons sur les différentes réparations effectuées au cours du projet, d'abord du bras, ensuite du crochet puis de la voile.

## VIII.A) Réparation du bras

Lors d'un essai de vol en téléopération, un choc a endommagé une pièce ronde du bras du robot. Pour réparer cette pièce, Georgios a reproduit le modèle existant en 3D et l'a imprimé. Grâce à cette pièce de remplacement, il a été possible de restaurer complètement le bras du robot et de rendre le système opérationnel à nouveau.

## VIII.B) Réparation du crochet

Lors du troisième lancement, après modification de la matrice d'efficacité, le lancement manuel s'est déroulé correctement au début. Cependant, un fil s'est coincé dans l'hélice en raison d'un mouvement de roulis induit par le décollage. Toutes les cordes du côté droit de la voile ont été entraînées par le moteur, rendant le démêlage très compliqué et nécessitant une réparation complète de la voile.

Figure 33 : Photo de la voile prise dans l'hélice du paramoteur

Ce constat confirme que le lancement manuel reste une opération délicate, exigeant technique et timing précis, avec un risque réel de coincer les fils dans l'hélice si la procédure n'est pas parfaitement maîtrisée. Lors de l'intervention, nous avons constaté qu'une pièce avait été éjectée lors de la violente rotation. Malgré des recherches sur le lieu, ce crochet n'a pas été retrouvée, nécessitant son remplacement par une pièce imprimée en 3D.
La première étape a été de remplacer le crochet manquant. Pour cela, nous avons utilisé un trombone et une pièce modélisée en 3D. On a d'abord schématisé sur papier la pièce. L'outil Autodesk Inventor Professional a ensuite été utilisé pour modéliser la pièce de remplacement, en respectant les dimensions de l'originale. La pièce pourrait être trop petite pour l'impression, mais elle conserve les proportions essentielles. L'impression sera réalisée sur l'une des machines disponibles, pilotée via CrealityPrint, et un trombone servira de support mécanique complémentaire.

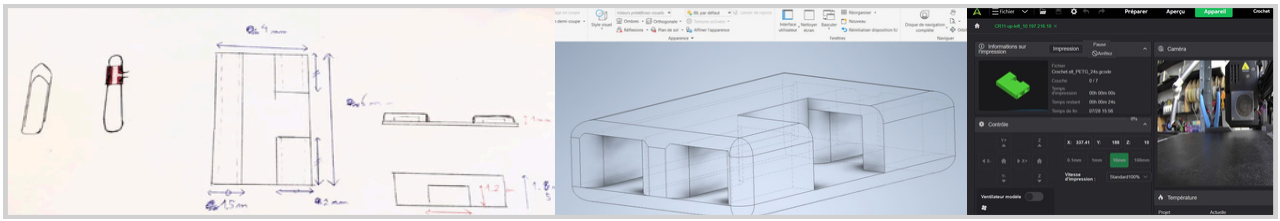


Figure 34 : Captures des différentes étapes de création de la pièce de rechange, schématisation (gauche), modélisation (centre), paramétrage d'impression 3D (droite)

La pièce modélisée initialement risquait d'être trop petite pour une impression correcte, bien qu'elle respecte les dimensions de l'originale. L'impression a été réalisée sur l'une des machines disponibles, pilotée via CrealityPrint, mais elle a échoué rapidement en raison de sa taille insuffisante. Il a donc été décidé de réaliser une pièce plus grande.

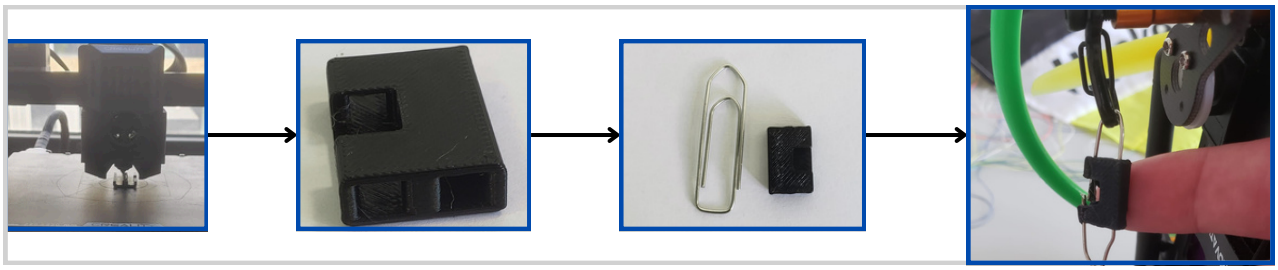


Figure 35 : Photos des différentes étapes d'assemblage de la pièce de rechange

## VIII.C) Réparation de la voile

Lors de la rotation violente de la voile, deux fils, l'un jaune et l'autre bleu, se sont arrachés de la voile elle-même. L'accroche étant endommagée, il a été nécessaire de trouver un moyen durable pour les refixer.

La réparation de la voile a ensuite été entreprise. Un nouveau trou précis et propre a été réalisé, et des pinces très fines ont été utilisées pour travailler avec précision. Les fils ont été renoués et du fil supplémentaire a été ajouté afin de consolider les attaches. Un test a été effectué en maintenant le drone par la voile dans le vide, vérifiant la solidité des réparations.

Ces interventions ont permis de restaurer l'intégrité de la voile, et le drone devrait désormais être prêt à voler à nouveau.

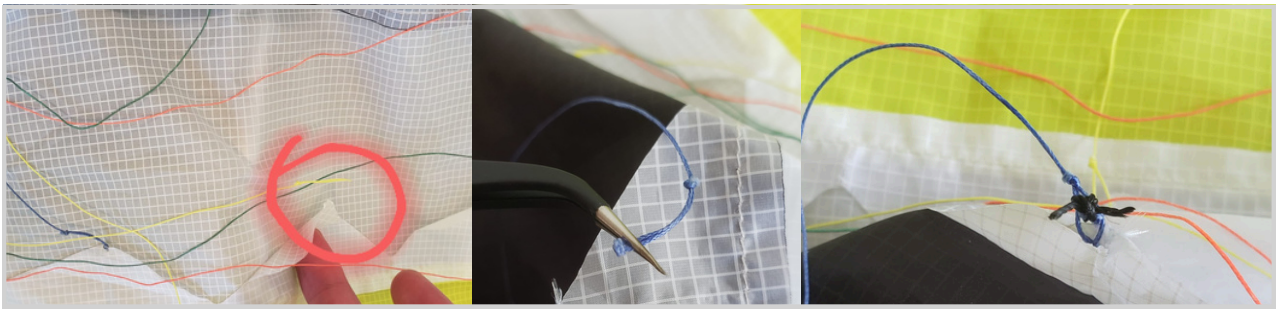


Figure 36 : Photos de la voile endommagée (gauche) et de la réparation manuelle (centre et droite)

# VI. Potentielle insertion

Dans cette section, nous répondons à la question des applications stratégiques du paramoteur autonome énoncées par le sujet. D'abord nous explorons ses capacités en le comparant à ses homologues, le drone à voilure-fixe et le quadrirotor. Ensuite nous conclurons quant aux applications militaires potentielles du paramoteur.

## VI.A) Limites et comparaison

On précise en premier lieu que beaucoup des avantages présentés sont conditionnels à la charge, au vent et à l'expérience de l'opérateur, surtout pour le paramoteur.

- **Endurance et autonomie**

Le paramoteur est capable de planer sur de longues distances tout en consommant peu d'énergie, grâce à la portance générée par sa voile. Son autonomie peut dépasser celle d'un quadrirotor, qui dépense beaucoup d'énergie pour maintenir un vol stationnaire. La voilure fixe, quant à elle, est très efficace en croisière et peut parcourir de grandes distances, mais nécessite une vitesse minimale pour rester en vol et présente une manœuvrabilité limitée dans les espaces confinés. Le quadrirotor offre un vol stationnaire facile, mais son autonomie reste très limitée.

- **Evolution en zones exiguës**

Si le quadrirotor excelle dans ces déplacements dans un espace restreint, le paramoteur se rapproche plus du voilure-fixe dans ce contexte et a besoin d'un espace plus ou moins grands, selon sa vitesse, pour effectuer un virage.

- **Silence et discrétion**

La voile du paramoteur réduit le besoin de grandes hélices rapides, contrairement au quadrirotor. Le bruit généré est donc moins perceptible, surtout à distance. Malgré tout, il n'est pas non plus silencieux. Cela va surtout dépendre du modèle et de l'environnement qui exigera plus ou moins de performances du moteur.

- **Décollage et atterrissage en terrain difficile**

Le paramoteur peut décoller et atterrir dans un espace assez restreint, même sur un terrain accidenté, à condition d'être précis. À l'inverse, la voilure fixe nécessite généralement une piste ou une catapulte pour le décollage. Le quadrirotor excelle également dans ce domaine, se ce n'est plus. Le lancement du paramoteur, comme expliqué et démontré précédemment est pénible et bien plus chronophage et risqué que celui du paramoteur.

- **Capacité de charge**

Le paramoteur peut transporter des charges plus lourdes sur de longues distances, telles que caméras, capteurs, équipements légers, voire du petit matériel tactique. La voilure fixe peut également transporter des charges, mais elle reste moins maniable à basse altitude.

- **Stabilité en vol lent**

Le paramoteur est capable de maintenir un vol très lent avec stabilité, contrairement à la voilure fixe qui nécessite une vitesse minimale pour rester en vol. Cela permet des observations précises, ou une réduction des nuisances sonores, depuis un point quasi-stationnaire, offrant un compromis intéressant entre quadrirotor et drone à voilure fixe.

- **Contrepoints notables**

Le paramoteur est plus sensible au vent et aux conditions météorologiques que ses homologues. Il est moins rapide et moins manœuvrable qu'un quadrirotor dans les espaces très confinés. Sa taille plus importante le rend également moins discret visuellement et ses lancements sont exigeants.

## VI.B) Applications militaires potentielles

Le paramoteur autonome présente un intérêt militaire notable grâce à son endurance et son autonomie, supérieures à celles d'un quadrirotor, lui permettant de parcourir de longues distances avec une consommation énergétique faible. Sa voile réduit le bruit, offrant une certaine discrétion, tandis que sa capacité à décoller et atterrir sur des terrains difficiles permet des opérations dans des environnements variés. Il peut transporter des charges légères ou tactiques sur de longues distances et maintenir un vol lent et stable, idéal pour la surveillance ou la reconnaissance.

Cependant, le paramoteur reste sensible au vent et aux conditions météorologiques, moins maniable dans des espaces confinés que le quadrirotor, et son lancement exige technique et précautions. Sa taille plus importante limite aussi sa discrétion visuelle. Ces contraintes doivent être prises en compte pour une utilisation militaire efficace.

# Conclusion

Le paramoteur autonome développé n'est pas encore pleinement opérationnel en vol, en raison de sa forte sensibilité à l'environnement et des limites de son modèle actuel. Plusieurs pistes d'amélioration ont été explorées, notamment l'ajustement de la matrice d'efficacité, l'utilisation de la stratégie TECS et le paramétrage du modèle. Cependant, les résultats obtenus restent insuffisants : les essais réalisés sont perturbés par les conditions climatiques (notamment le vent), qui faussent les mesures, et la simulation disponible ne reproduit pas avec assez de fidélité la réalité pour servir de banc de test fiable.

Ainsi, une première étape essentielle serait d'améliorer la simulation afin de la rendre encore plus réaliste, bien que la prise en compte des aléas climatiques demeure un défi complexe. Sur cette base, une optimisation des paramétrages, suivie d'une adaptation appropriée de la matrice d'efficacité, devrait permettre d'obtenir des résultats plus robustes et convaincants.

DTU

MARS LAB
Mechatronic and Advanced Robotic Systems Laboratory

ENSTA
IP PARIS

# Fin du rapport de stage 2A

Développement d'un paramoteur autonome et analyse de ses applications stratégiques