Projet ELYTRA

2022-2024
Association ELISA SPACE

Equipe

Aurélien ATTARD, chef du projet
Arthur JUNG
Arthur DEWASTE
Emilie PEREZ
Baptiste SCHWENDIMANN
Lana FREYMANN – DAMIEN

Vue d’ensemble

Le projet Elytra est une fusée expérimentale dotée d'un squelette interne en aluminium et PLA, ainsi que d'une peau externe en fibre de carbone. Elle mesure 2000 mm de hauteur et pèse 8,3 kg. Elle embarque diverses expériences, telles que la télémesure, qui renvoie des données de vol comme la vitesse dans l’air, les accélérations, un GPS, etc. La fusée est également équipée de quatre caméras, dont une stabilisée en roulis, ainsi que d'un tube Pitot à son sommet.

Figure 1 : ELYTRA au décollage

Table des matières

Table des matières

I. Expériences

1. Caméra stabilisée en roulis

Un roulis de la fusée est attendu lors de la phase d'ascension en raison de l'alignement imparfait des ailerons par rapport à l'axe de la fusée. Un roulis élevé n'est pas agréable pour le visionnage des caméras qui filment le voyage. Les fusées du C'Space sont généralement stabilisées en roulis grâce à une masse mise en rotation inverse au roulis induit. Dans notre cas, l'expérience a été de stabiliser uniquement une caméra au roulis. C'est-à-dire que la fusée pouvait tourner sur elle-même tandis qu'une caméra à l'intérieur tournait à la même vitesse dans le sens inverse pour compenser la rotation sur les images.

A. Mécanique et intégration

L'ensemble des équipements fonctionne indépendamment du reste de la fusée. En effet il s'agit d'un bloc composé de la caméra avec le hardware qui est en rotation lorsque la fusée tourne sur elle-même :

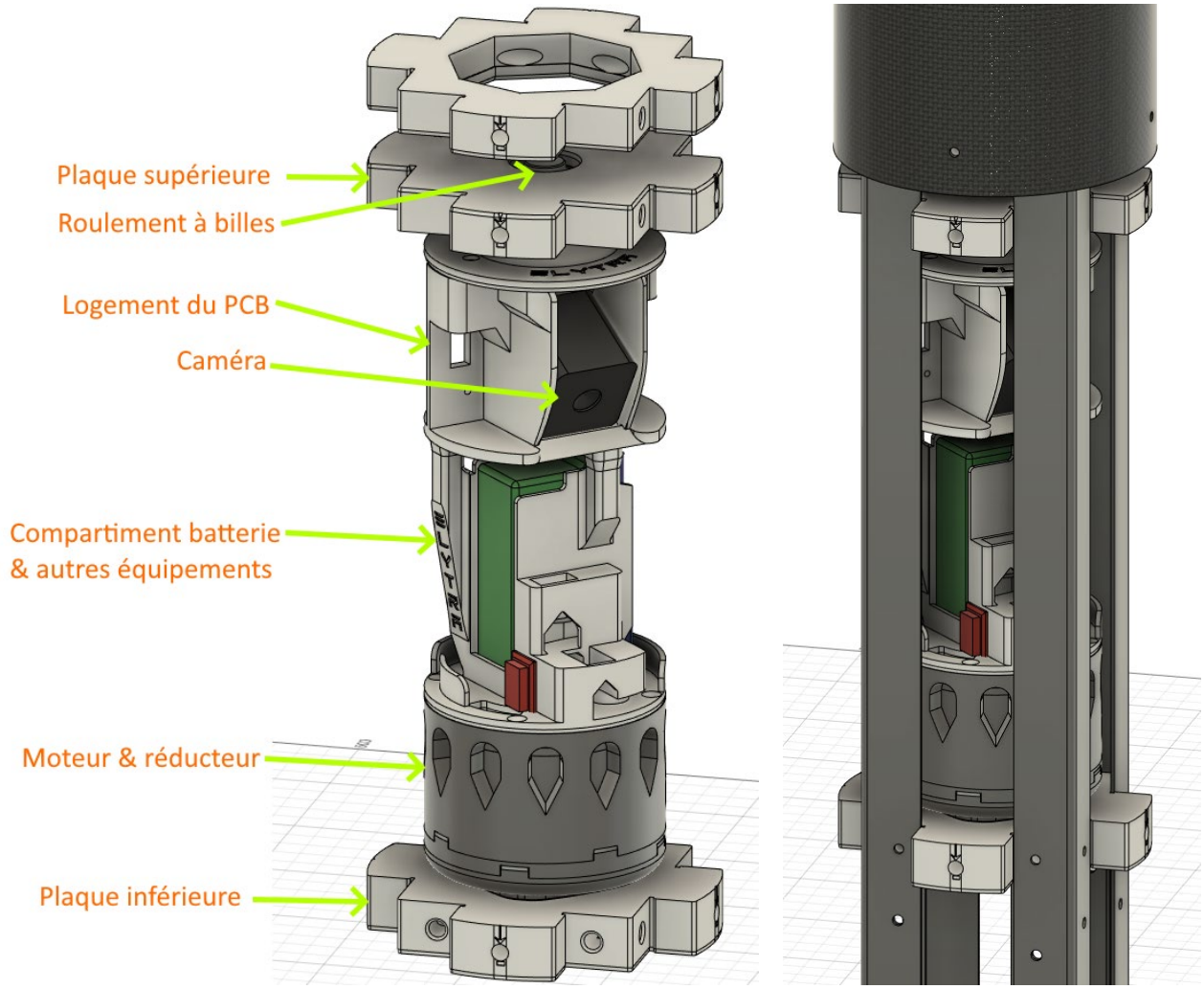
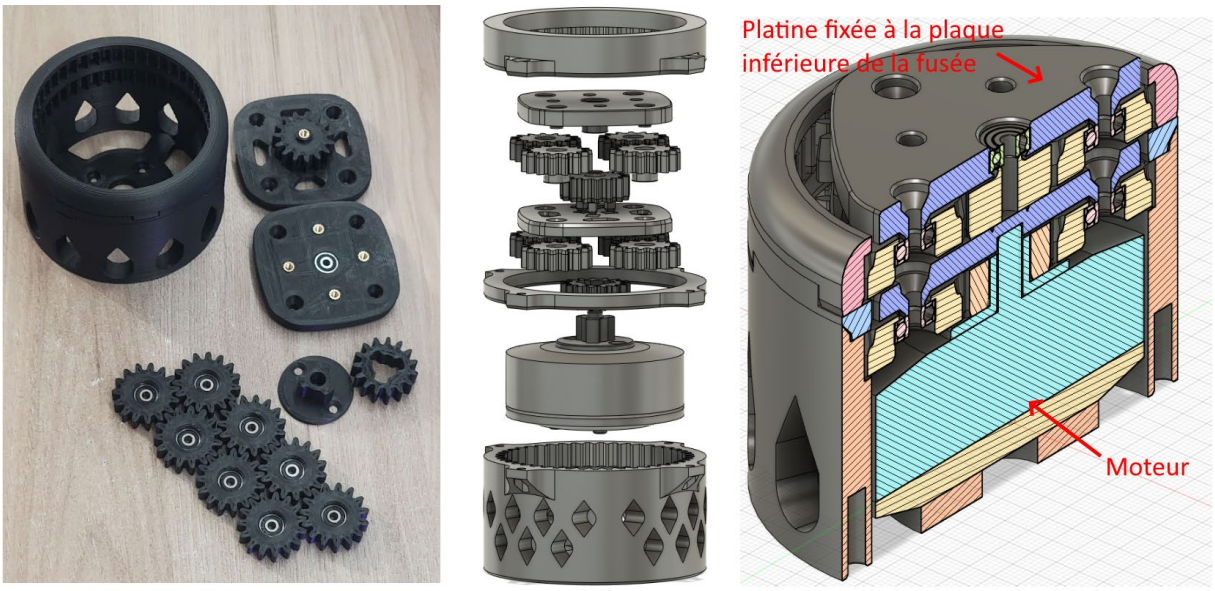


Figure 2 : Module du stabilisateur logé entre deux plaques de fixation

Les plaques permettent la fixation du module à la fusée. La plaque supérieure possède un roulement à billes pour maintenir le module en rotation. Sur la plaque inférieure, est fixée une platine liée au bloc moteur. Le rotor de ce dernier est fixe par rapport à la structure de la fusée. C'est le stator qui pivote autour du rotor fixé à la plaque. L'ensemble du module pivotable est vissé sur le stator du moteur.
La largeur du module est limitée par la place qu'occupe les cornières du squelette. Il a fallu condenser tout le système dans un espace étroit. Le défi de l'intégration était particulièrement lié au choix du moteur. C'est un moteur sans balais de 130 KV alimenté en 12V soit une vitesse de rotation de 130*12 = 1560 tr/min dans les deux sens. Après avoir étudié les rapports de fuseX d'autres clubs, il se trouvait que le roulis n'allait jamais au-delà de 100 tr/min. Afin de réduire la vitesse du moteur, il a fallu réaliser un réducteur de vitesse. Le choix était de faire un réducteur de type planétaire car il est compact. Le réducteur est composé de deux étages réduisant ¼ de la vitesse chacun. Le rapport de réduction en sortie est donc de 1/16. Le module avait donc une vitesse maximale théorique de plus ou moins 97,5 tr/min ce qui répondait aux exigences souhaitées.

Le réducteur est imprimé en PLA carbone et les roues dentées sont dotées de petits roulements à billes. L'ensemble des pièces frottantes ont été lubrifiées avec du lubrifiant sec au PTFE.

Le développement du réducteur a nécessité à lui seul 15 itérations de prototypage. Après de nombreux essais, le réducteur en PLA résistait étonnamment bien aux frottements aux contactes des dents. Aucune poussière de PLA ne se produisait à l'intérieur du réducteur. Les autres éléments du module de stabilisation sont également imprimés en PLA. Le compartiment à batterie et le compartiment de la caméra sont assemblés à l'aide de vis et d'inserts. L'ensemble du module avait une masse approximant les 900g. La batterie et le moteur sont les éléments les plus lourds.



Pièces du réducteur | Réducteur en vue éclatée | Vue en coupe du réducteur assemblé

Figure 3 : Assemblage du réducteur

L'intégration du module dans la fusée a nécessité de découper des fenêtres dans la peau en fibre de carbone. Concernant les 4 cornières autour, elles figureront dans le champ de vision de la caméra. Les trous dans la peau n'ont pas trop impactés la rigidité de la fusée grâce au squelette porteur.

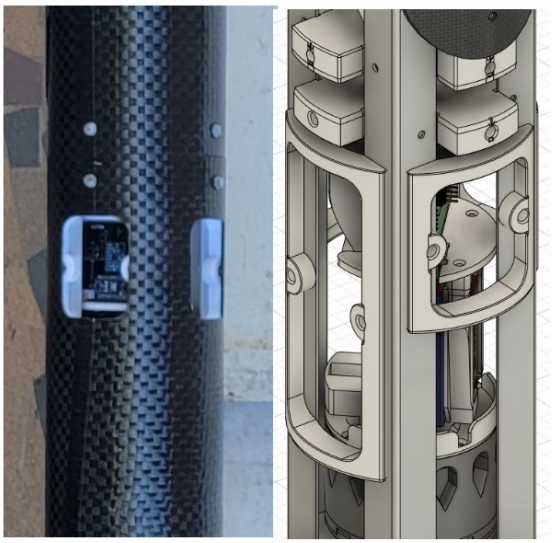


Figure 4 : Fenêtres pour la caméra stabilisée en roulis

La caméra stabilisée était exposée à l'air libre à travers les quatre fenêtres. Une fenêtre est plus grande que les autres permettant d'accéder facilement à la batterie située plus bas.

B. Hardware

Les équipements électroniques qui ont été nécessaires sont les suivants :

- Une caméra 1080p 30fps autonome avec sa propre batterie et micro SD
- Un moteur de drone 130KV
- Un ESC bidirectionnel
- Une batterie LiPo 3S
- Un BMS

Sur le PCB :

- Arduino Nano
- Un gyromètre MPU6050
- Un module d'écriture sur carte micro SD
- Une LED bleue
- Un écran oled + module Bluetooth + 3 boutons servant au débogage

L'ordinateur qui pilote la rotation du module était réalisé sur un PCB logé derrière la caméra :

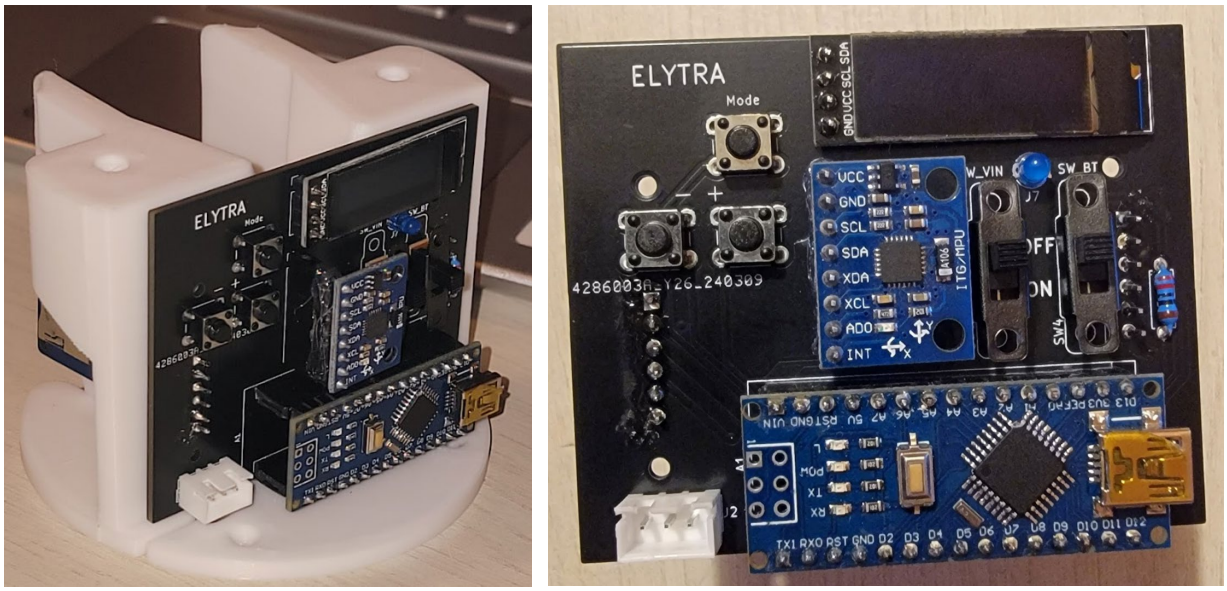


Figure 5 : PCB du stabilisateur

En raison d'un manque d'espace, le module d'écriture de carte micro SD ainsi que le module Bluetooth HC05 se situent à l'arrière du PCB.

Alimentation : :

Le module de stabilisation est alimenté par une batterie LiPo 3S 35C 2200mAh. L'ajout d'un BMS est nécessaire pour respecter le cahier des charges des FuseX. La sortie d'alimentation passe par l'ESC pour commander le moteur. L'ESC possède une sortie 5V qui sert à alimenter les composants du PCB. Il y a un interrupteur général au niveau de la batterie. Deux autres switchs sur le PCB pour l'alimentation en 5V et l'alimentation seule du module Bluetooth. Ce dernier ne servait qu'aux tests lors du développement et n'était pas utilisé lors du vol.

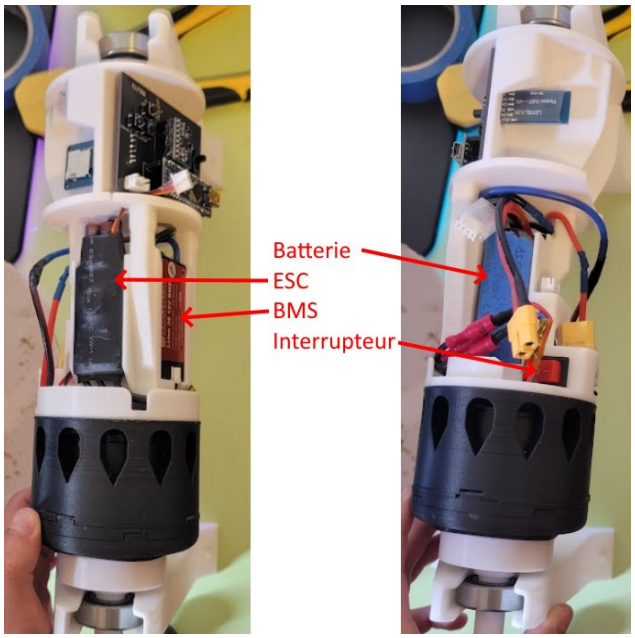


Figure 6 : Module de stabilisation sur banc d'essai

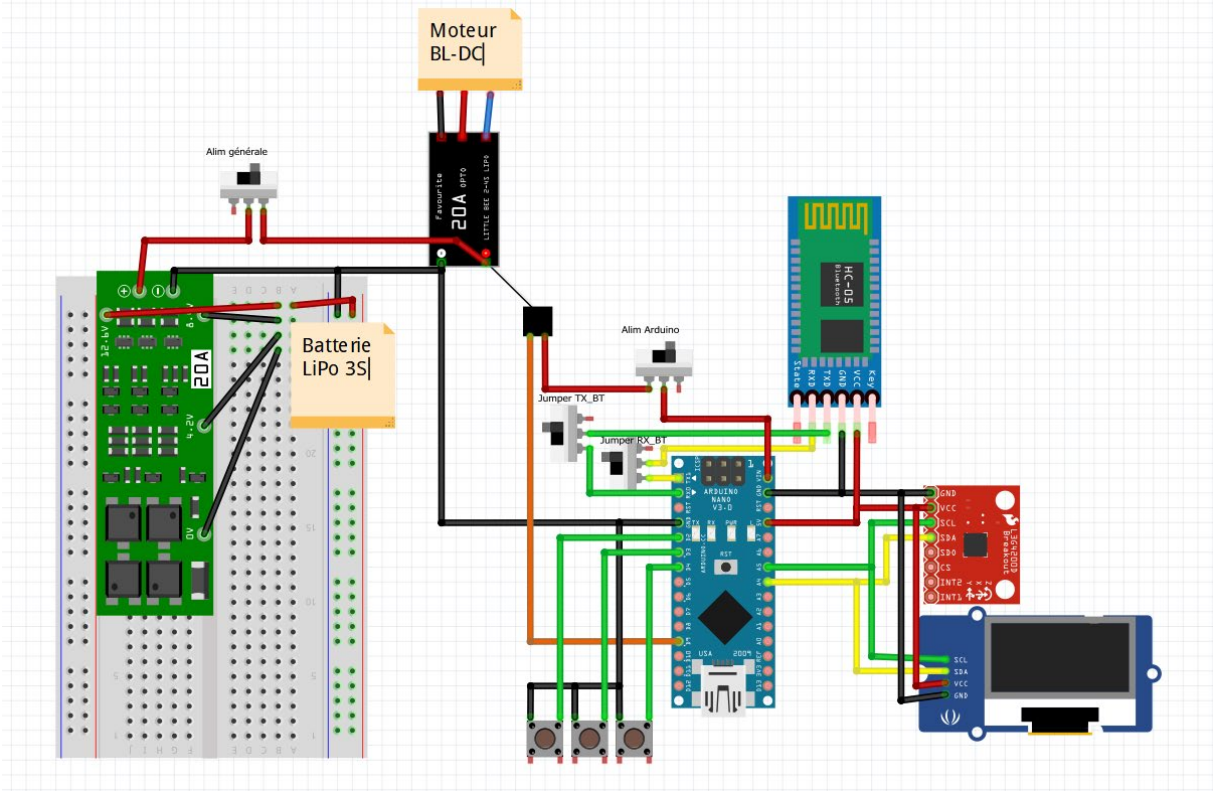


Figure 7 : Schéma électrique du module

Mise à jour du schéma : Les switchs "TX_BT" et "RX_BT" ont été remplacés par un switch sur le VCC du module HC05.

C. Software

Le programme du stabilisateur est composé d'une boucle d'asservissement pour annuler le roulis de la fusée. Le modèle du système dynamique envoie une commande en PWM à l'ESC du moteur. Cette commande varie entre 1000 et 2000 où 1500 est le point mort car nous utilisons un ESC bidirectionnel. La consigne souhaitée est une vitesse angulaire nulle (0°/s). Lorsque la fusée tourne, le gyromètre MPU6050 mesure l'erreur de cette vitesse entre le module de stabilisation et le roulis de la fusée. Pour atteindre la consigne de 0°/s en commandant le moteur, un correcteur PI (proportionnel, intégral) suffisait. Le caractère de dérivation dans le correcteur n'apportait pas de correction plus dynamique dans notre système, on s'est donc passé de faire un PID.
La détermination des coefficients du filtre PI a été faite sur Matlab Simulink. Premièrement, il a fallu identifier le modèle physique par une fonction de transfert. L'outil "System Identification" sur Simulink nous a permis d'identifier le modèle à partir d'une commande échelon envoyée au moteur. Une commande échelon est une simple commande fixe envoyée au moteur (en PWM) pour observer sa réponse (vitesse de rotation) :

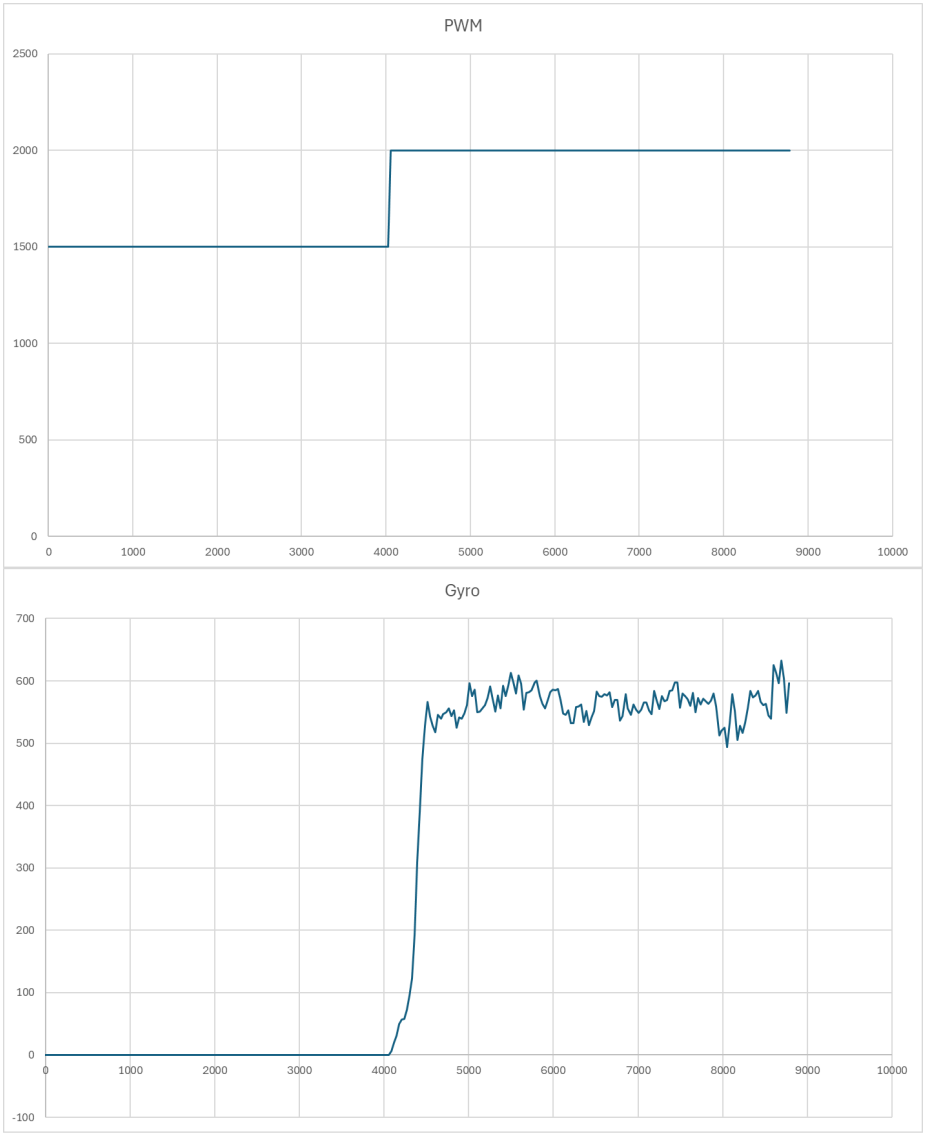


Figure 8 : Réponse du système en fonction de la commande échelon, commande (PWM) et vitesse angulaire du module (Gyro) en fonction du temps.

Le système avait un temps de réponse élevé de 360 millisecondes. Une moyenne glissante est appliquée sur le gyromètre pour lisser les mesures. Le programme fonctionne sur un pas de temps fixe de 30 ms.

La fonction de transfert déterminée à partir de l'outil sur Simulink est la suivante :

$$z^{(-12)} * \frac{0.2342\ z^{-1}}{1 - 0.1694\ z^{-1}}$$

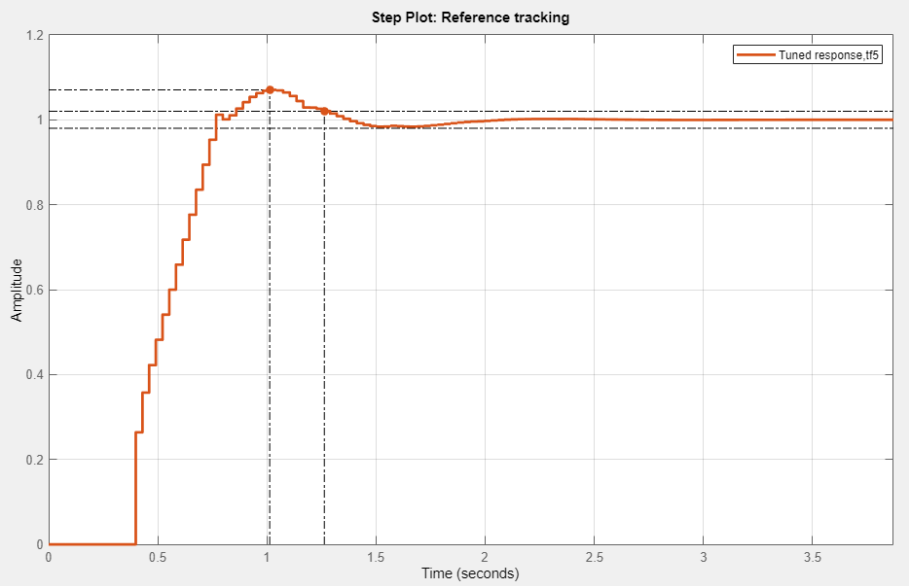


Figure 9 : Réponse du filtre PI calibré

Le système est discret avec un pas de temps fixe, Ts = 30 ms, retard : 12*Ts (=360 ms). La fonction de transfert a permis de déterminer un modèle du correcteur PI à l'aide de Matlab, en utilisant l'outil "PID Tuner". Le correcteur PI a été réglé pour obtenir cette réponse :

La réponse du système n'est pas très rapide mais suffit pour corriger le roulis induit de manière stable. En revanche, il n'est pas assez dynamique pour corriger un mouvement du type balancement, oscillation de la fusée lorsqu'elle est sous parachute par exemple.

La stabilisation de la caméra devait se déclencher après la phase de propulsion du moteur (qui était inférieure à 2 secondes). Tout d'abord, le programme attend le décollage. Ce dernier peut être détecté par une accélération élevée mesurée par l'accéléromètre MPU6050. Le module étant réfugié à l'intérieur de la fusée et pouvant tourner, il n'était pas envisageable d'utiliser une prise jack de déclenchement. Après la détection du décollage, 2 secondes sont attendues avant de pouvoir actionner la stabilisation. En parallèle, l'accélération de la fusée, la vitesse angulaire du module et la commande moteur sont enregistrées dans une micro SD. L'écriture sur carte SD a tendance à ralentir les performances du programme. Le microcontrôleur ATmega328p de l'Arduino nano est assez limité pour exécuter l'asservissement avec l'enregistrement sur carte SD. Une solution était d'utiliser la librairie ChibiOS (https://github.com/greiman/ChRt) afin de séparer la partie asservissement du moteur et l'enregistrement sur la carte SD sur deux threads. Les threads sont exécutés en protocole FIFO (first in first out). Ceci permet de garantir un pas de temps fixe (ici 30 ms) et de garder la priorité sur la boucle d'asservissement si la séquence d'écriture sur la carte micro SD devait dépasser le lapse de 30 ms imposé. Le programme est situé en annexe.

2. Tube Pitot

Figure 10 : Coiffe et Pitot de la fusée ELYTRA

La fusée expérimentale ELYTRA est équipée d'un tube Pitot, il permet de mesurer une pression dynamique P_{dyn} à partir d'une pression totale P_{tot} et statique P_{stat} :

$$P_{dyn} = P_{tot} - P_{stat}$$

Cette pression dynamique nous permet ensuite d'en déduire la vitesse de la fusée dans la masse d'air.

Voici, en Figure 11, un schéma simplifié du tube de Pitot :

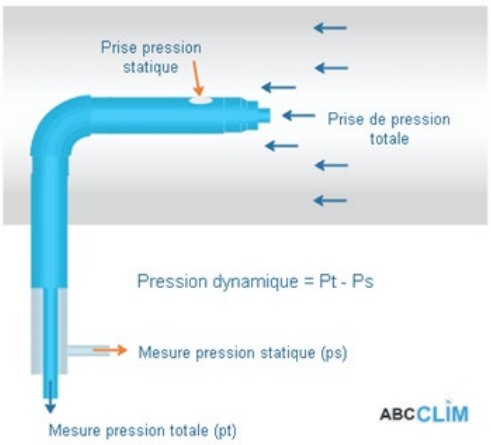


Figure 11 : Schéma simplifié du tube de Pitot.

Sur ELYTRA, le Pitot a été placé à la pointe de la fusée (Figure 10) pour profiter du meilleur flux d'air possible, un placement sur la peau de la fusée aurait pu créer une dissymétrie, jouant donc négativement sur la stabilité de la fusée.

Il aurait été cependant possible de réaliser un montage de Pitot sur la peau de la fusée (surélevé de quelques centimètres pour éviter les perturbations de la couche limite) en installant deux Pitot diamétralement opposés. Cela éviterait les pertes de stabilité grâce à la symétrie du système tout en garantissant un principe de redondance.

Figure 12 : Tube Pitot utilisé sur ELYTRA

Le tube Pitot utilisé pour ce projet a été acheté en ligne (Figure 12). Il comporte deux orifices : le premier, situé au centre, mesure la pression dynamique, tandis que le second, légèrement excentré, mesure la pression statique.

De façon à mesurer les différentes pressions du tube de Pitot, nous avons utilisé un capteur de pression différentielle MPX5050DP visible en Figure 13.

Figure 13 : capteur MPX5050DP

Ce capteur possède 2 entrées, la première, pour la prise de pression statique (dite VACUUM) et la seconde, pour la prise de pression dynamique (dite POSITIVE PRESSURE).

Le capteur MPX5050DP est un transducteur de pression piézorésistif intégré, il fonctionne en mesurant la pression à l'aide d'une jauge de contrainte en silicium, qui se déforme sous l'effet de la pression appliquée, modifiant ainsi sa résistance électrique. Cette variation de résistance est convertie en un signal électrique, amplifié et conditionné pour fournir une sortie

analogique proportionnelle à la pression. Le capteur est pré-calibré et compensé en température pour assurer une précision optimale.

En sortie de ce capteur, nous obtenons donc une tension analogique proportionnelle à la pression appliquée. Cette tension varie linéairement en fonction de la pression mesurée, avec une plage de sortie typique de 0,2 V à 4,7 V correspondant à une pression de 0 à 50 kPa.

Cette tension est mesurée à l'aide d'un des ports analogiques de la carte expérience LILYGO basé sur un ESP 32. Du fait du fonctionnement de la carte en 3,3 V, il était nécessaire de réduire la plage de sorties du MPX5050DP à l'aide d'un pont diviseur de tension, directement présent sur le PCB expérience d'ELYTRA (visible sur la Figure 14).

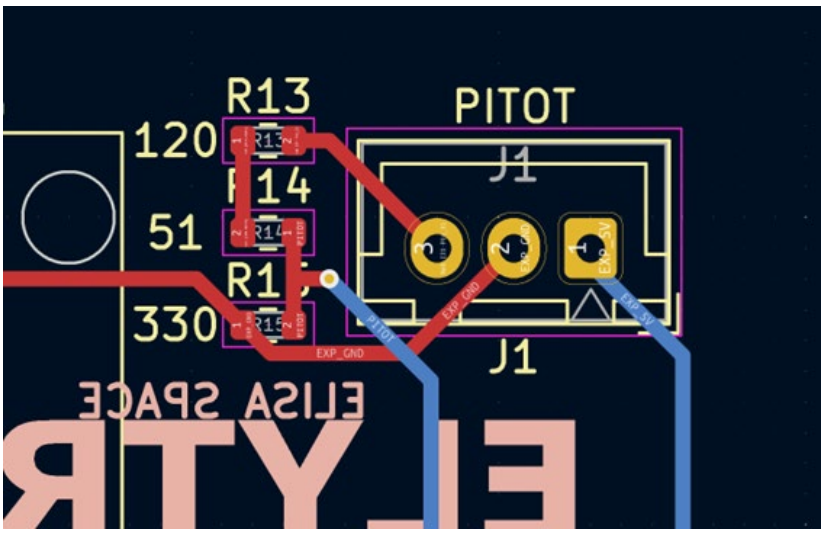


Figure 14 : pont diviseur de tension présent sur le PCB expérience d'ELYTRA

D'après les estimations, la vitesse maximale d'ELYTRA allait être supérieure à $600\ km.h^{-1}$, à ces vitesses, il n'est plus possible de négliger la compressibilité de l'air. Nous ne pouvons donc pas utiliser l'équation de Bernoulli.

Il est cependant possible d'utiliser l'équation de Saint Venant :

$$v = \sqrt{\frac{2\Delta P}{\rho\left(1+\frac{M^2}{4}\right)}}.$$

Avec :

v : la vitesse de la fusée dans la masse d'air;

ΔP : le différentiel de pression donné par le capteur MPX5050DP;

ρ : la masse volumique de l'air;

M : le nombre de Mach de la fusée.

Nous pouvons remarquer que lorsque M est suffisamment faible, cette équation est équivalente à celle de Bernoulli.

De manière à exploiter cette équation pour le calcul de vitesse, nous avons rédigé un programme Arduino, visible en annexe 1, pour exploiter le MPX5050DP. Ce programme est conçu pour mesurer la pression et calculer la vitesse de la fusée en temps réel à l'aide d'un capteur de pression différentielle MPX5050DP connecté à un microcontrôleur ESP32.

Le programme commence par une phase de calibrage, où l'offset du capteur est ajusté pour s'assurer que la pression mesurée à l'état stable est proche de zéro. Ensuite, dans la boucle principale, le programme lit la valeur analogique du capteur, la lisse en utilisant une moyenne glissante sur 10 lectures, puis la convertit en tension et en pression (en kPa). La vitesse de la fusée est ensuite calculée en appliquant l'équation de Barré de Saint-Venant, tenant compte des effets de compressibilité à haute vitesse. La vitesse, la pression et la tension sont ensuite transférées, permettant un suivi en temps réel des performances de la fusée. Le programme met également à jour la valeur de Mach à chaque itération en fonction de la vitesse mesurée.

De manière à valider le fonctionnement de notre programme, nous avons utilisé la soufflerie de notre école, ELISA Aerospace Saint-Quentin. Cette soufflerie est capable de simuler un vol à près de 100km.h^(-1)(opérations visibles en Figure 15).

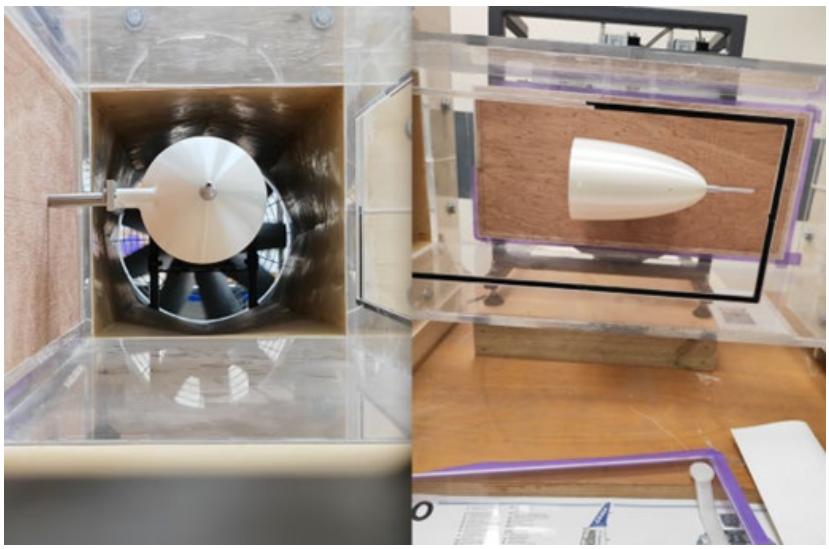


Figure 15 : développement du programme dans la soufflerie de l'école.

Pendant les premières secondes de vol, le tube Pitot remplit parfaitement mission. Les données de pression ont été correctement captées par le capteur et transmises à la carte LILYGO. Ces données ont ensuite été envoyées avec succès via le système de télémesure, permettant une analyse en temps réel de la vitesse de la fusée. Les mesures étaient cohérentes avec les prévisions.

Cependant, au moment où la fusée a atteint une accélération maximale, un incident imprévu s'est produit. La forte accélération a provoqué l'arrachement de la carte LILYGO, qui était fixée à l'intérieur de la fusée. Cet incident a immédiatement interrompu la connexion entre le capteur et la carte, entraînant la fin brutale de la transmission des données.

En conséquence, les données de vol n'ont pu être collectées que durant les premières secondes, limitant ainsi l'analyse complète des performances en vol. Cet incident souligne l'importance cruciale d'une fixation robuste et d'une meilleure gestion des contraintes mécaniques sur les composants électroniques lors de vols à haute vitesse. Des améliorations devront être apportées à la structure de fixation de la carte et à la conception globale pour éviter de telles défaillances lors des prochains projets de fusées expérimentales.

II. Ordinateurs de bord et télémesure

1. Introduction

Le présent rapport porte sur la télémétrie réalisée pour la FuseX Elytra, un projet ambitieux mené dans le cadre de l'association Elisa Space de l'école ELISA Aerospace. L'objectif principal de cette télémétrie était de récupérer et de visualiser en temps réel les données expérimentales provenant de la coiffe de la fusée, sous diverses formes, incluant un modèle 3D, ainsi que des graphiques relatifs à l'altitude, la vitesse, les données GPS, et les mesures des capteurs accéléromètriques et gyroscopiques.

La FuseX Elytra est une fusée de deux mètres de hauteur, intégrant trois expériences principales : un tube de Pitot, un système de télémesure, et un stabilisateur de caméra conçu pour contrer le roulis pendant le vol. La réalisation de cette télémétrie s'inscrit dans un projet universitaire, avec pour finalité la participation au C'Space 2024, une compétition organisée par Planète Sciences et le CNES, qui offre aux étudiants l'opportunité de lancer leurs propres fusées. Cette initiative a d'ailleurs été récompensée, témoignant de la qualité et de l'originalité du travail accompli.

L'intérêt pour la télémétrie est né d'une curiosité personnelle pour un domaine peu exploré mais intrinsèquement fascinant : celui de la visualisation en temps réel de données expérimentales issues d'un système en mouvement. Travailler en collaboration avec Melvin, un ancien membre de l'association, a permis d'optimiser un logiciel existant en fonction des exigences spécifiques du projet, rendant l'expérience non seulement technique, mais aussi extrêmement enrichissante. Et travailler avec Arthur, pour la partie hardware de la télémétrie, qui a été un fondement dans la recherche des bons composants nécessaires à bonne réalisation de l'expérience.

2. Matériel et Méthodes

Cette section décrit en détail l'architecture matérielle et logicielle utilisée pour la télémétrie de la FuseX Elytra, en couvrant les éléments suivants : les capteurs et la collecte des données, la transmission de ces données au système de réception, les outils logiciels utilisés pour la visualisation en temps réel, et enfin le système de sauvegarde des données.

A. Système d'Acquisition de Données (Sender)

La partie Sender peut être divisée en deux sous-sections : la création des données par les capteurs et la transmission de ces données.

Capteurs :
Qui dit envoi de données, dit création de données au travers de ces capteurs ! Dans la coiffe de la FuseX Elytra, un PCB (Printed Circuit Board) a été intégré, portant une série de capteurs essentiels :

- MPU 6050 : Un capteur combinant un accéléromètre et un gyroscope, offrant des mesures précises de l'accélération et de la rotation dans trois axes. Il est souvent utilisé pour déterminer l'orientation et les mouvements angulaires de la fusée.
- MPU 9250 : En plus des fonctionnalités du MPU 6050, ce capteur inclut un magnétomètre, ce qui permet théoriquement de mesurer l'orientation par rapport au champ magnétique terrestre, bien que cette fonctionnalité n'ait pas été exploitée dans notre cas en raison de problèmes techniques.
- BMP280 : Capteur de pression barométrique utilisé pour déterminer l'altitude de la fusée en vol. Ce capteur fournit également des données de température.
- GPS NEO 6M : Un module GPS permettant d'obtenir des données de positionnement géographique précises, essentielles pour suivre la trajectoire de la fusée.
- Tube Pitot : Bien que non intégré directement sur le PCB, le tube Pitot est utilisé pour mesurer la vitesse de la fusée en fonction de la pression dynamique de l'air.

Transmission des Données :

Après avoir collecté les données via les capteurs, celles-ci sont transmises vers le récepteur. Divers tests ont été effectués pour sélectionner la meilleure carte pour cette tâche, et le choix final s'est porté sur une LILYGO T3 V1.6.1. Cette carte intègre une LoRa 32 de 868/915 MHz, avec une antenne intégrée.

Figure 16 : Carte LILYGO LoRa TTGO

Le choix de la fréquence 868 MHz s'explique par sa conformité avec les réglementations françaises pour ce type d'application. Cette fréquence est largement utilisée pour les communications longue distance à faible consommation d'énergie, rendant le système robuste et fiable. L'ensemble de l'électronique est soigneusement intégré dans la coiffe de la fusée pour minimiser l'encombrement et maximiser l'efficacité.

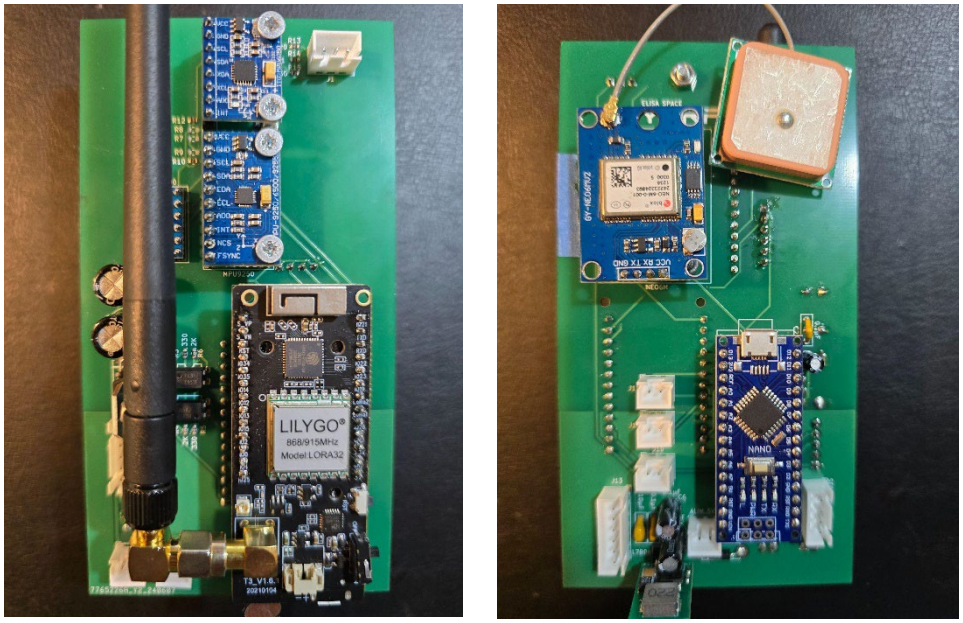


Figure 17 : Vue d'ensemble du PCB et des capteurs avec la LILYGO et le séquenceur

L'Arduino Nano sur le PCB est le séquenceur qui permet l'ouverture de la trappe parachute à l'apogée si la carte expérience Lilygo échoue.

B. Système de Réception des Données (Receiver)

Le Receiver est également divisé en deux parties : l'antenne de réception et le traitement des données reçues.

Antenne de Réception :

Pour capter le signal transmis par la fusée, une antenne Yagi de 12 dB a été utilisée. Cette antenne, précédemment employée dans un autre projet de l'association, "Belissama", est idéale pour maintenir une connexion stable avec la fusée pendant son vol. La puissance de 12 dB a été choisie pour garantir une réception optimale, même en cas de conditions suboptimales ou d'éloignement important.

L'antenne Yagi, également connue sous le nom d'antenne Yagi-Uda, est une antenne directionnelle à éléments parasites, utilisée pour sa capacité à focaliser les signaux dans une direction spécifique. Cette caractéristique est particulièrement utile pour suivre la fusée pendant son vol, car elle permet d'obtenir un gain de signal important dans la direction visée, tout en minimisant les interférences provenant d'autres directions.

Le gain de 12 dB signifie que l'antenne peut augmenter la puissance du signal reçu de manière significative, comparée à une antenne isotrope (théorique). Ce gain est obtenu grâce à la configuration particulière de l'antenne Yagi, composée d'un élément alimenté (dipôle) et de plusieurs éléments non alimentés (parasites), comme les directeurs et les réflecteurs. Ces

éléments parasites réémettent les signaux reçus en les renforçant dans une direction spécifique, optimisant ainsi la réception.

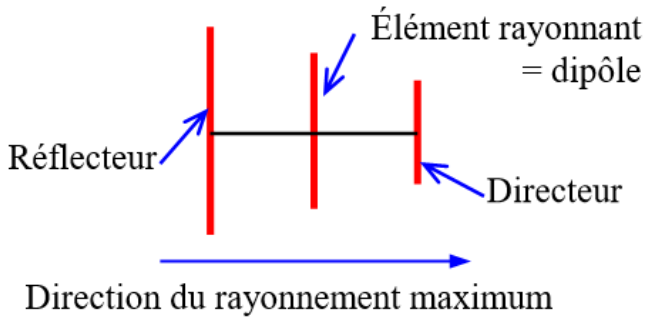


Figure 18 : Schéma simplifié du fonctionnement d'une antenne YAGI

Cependant, ce gain élevé nécessite une grande précision dans l'orientation de l'antenne, car celle-ci doit être pointée directement vers la fusée pour capter le signal efficacement. La directivité de l'antenne Yagi, qui est à la fois un avantage et un inconvénient, demande une attention particulière pour maintenir la connexion durant tout le vol, surtout lorsque la fusée atteint des altitudes et des distances significatives.

Enfin, l'antenne Yagi présente un diagramme de rayonnement étroit, ce qui signifie que le signal est concentré dans une seule direction, augmentant ainsi la portée effective de l'antenne. Ce type d'antenne est couramment utilisé dans les applications où la précision et la distance sont critiques, comme les liaisons point à point ou les communications radio longue distance, du fait de l'utiliser pour de la télémétrie de fusée amateur.

Figure 19 : Antenne YAGI 12 dB

Traitement des données :

Pour ce faire, nous avons utilisé une nouvelle fois une carte LILYGO, identique à celle employée pour l'émetteur (sender). Cette LILYGO se charge de traiter les données reçues avant de les transmettre directement au logiciel utilisé pour leur visualisation. Cela constitue une transition idéale pour aborder la description du logiciel lui-même.

C. Logiciel de Visualisation (ViSU)

Le logiciel utilisé pour la visualisation en temps réel des données est appelé ViSU. Ce logiciel, initialement développé par Melvin, un ancien membre de l'association, et a été ensuite optimisé pour répondre aux besoins spécifiques de ce projet.

Problèmes et Optimisations :

L'un des principaux défis rencontrés dans l'utilisation du logiciel ViSU pour la visualisation en temps réel des données a été la gestion des interférences. Ces interférences, ou "bruit", se manifestent par l'apparition de valeurs aberrantes sur les graphiques. Ces valeurs anormales peuvent parfois se présenter sous forme de barres verticales visibles sur les graphes, comme vous pouvez le constater dans l'image ci-dessous. Ce bruit peut provenir de diverses sources, telles que des perturbations électromagnétiques ou des erreurs de transmission de données.

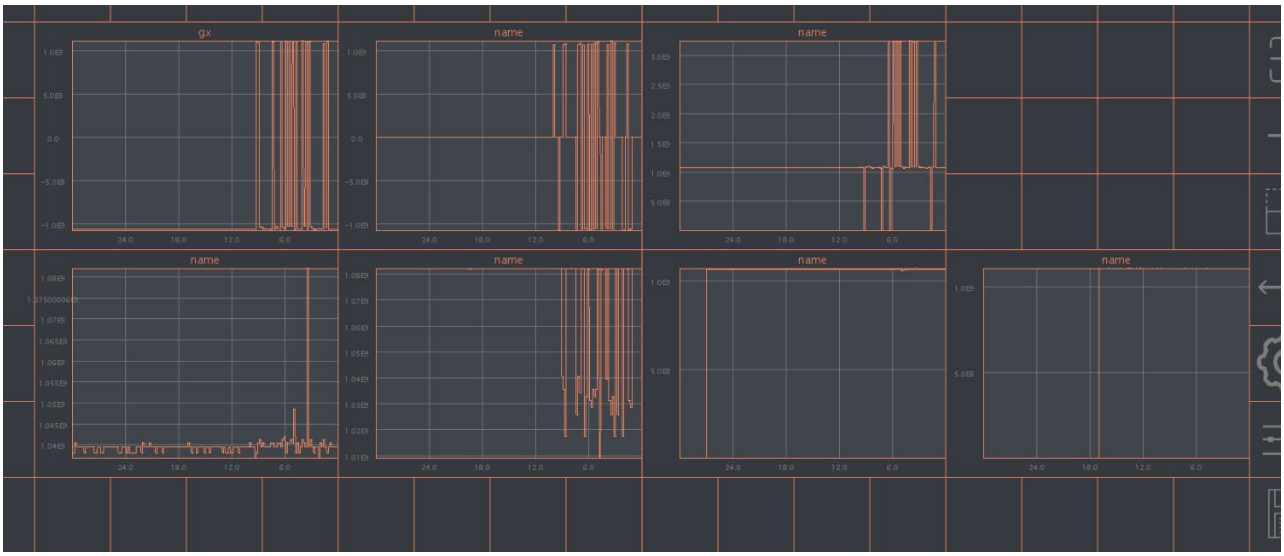


Figure 20 : Bruit impliquant des valeurs illisibles

Pour tenter de résoudre ce problème, nous avons expérimenté l'application de différents filtres, tels que le filtre de Kalman, afin de limiter l'impact du bruit sur les données. Cependant, il s'est avéré complexe de combattre efficacement ces interférences uniquement avec des filtres. Les erreurs persistantes ont continué d'affecter la qualité des données affichées.

Pour contourner ce problème et garantir des données lisibles, nous avons finalement intégré un filtre directement dans le logiciel ViSU. Ce filtre ne tente pas de supprimer le bruit, mais

plutôt de cadrer les graphiques en imposant des valeurs limites. Ainsi, même si quelques valeurs aberrantes persistent, elles ne vont pas écraser les autres données, car elles sont plafonnées par ces limites. Cette approche permet de préserver la lisibilité et la cohérence des visualisations, en évitant que les anomalies ne perturbent la lecture des données principales.

Fonctionnalités :

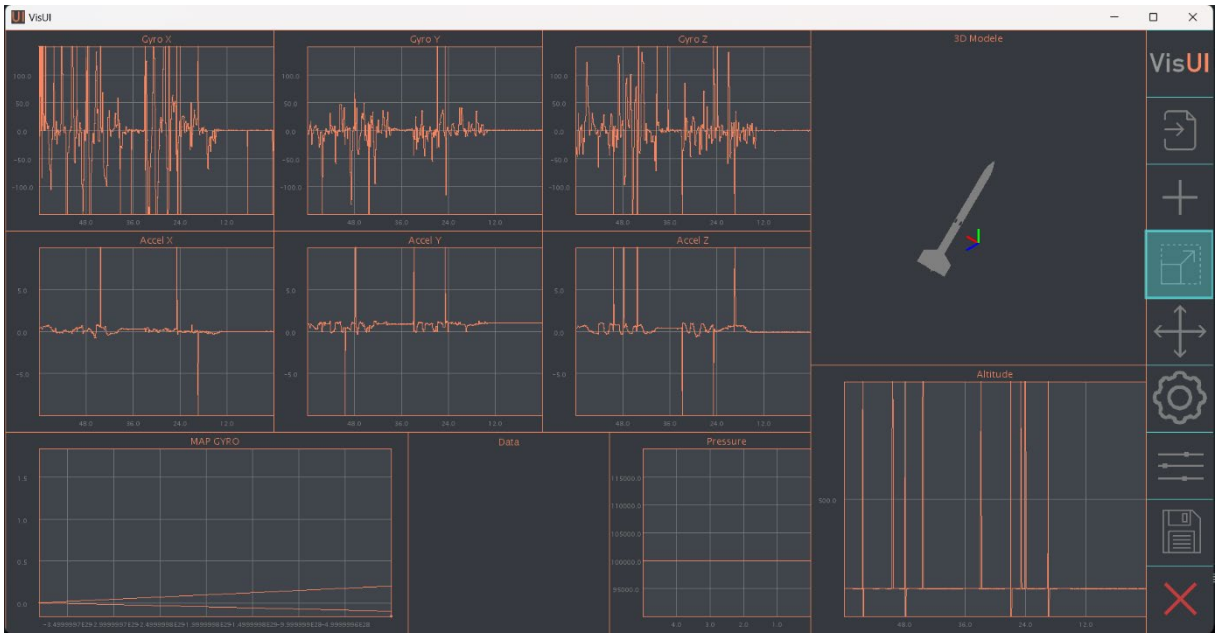
Concernant les fonctionnalités, le logiciel ViSU a été conçu pour offrir une interface complète et intuitive pour la visualisation des données de la fusée en temps réel. Ses fonctionnalités sont spécifiquement développées pour répondre aux besoins des projets spatiaux amateurs, en mettant l'accent sur la précision des données et la clarté des visualisations.

- Modèle 3D : Permet de visualiser l'orientation de la fusée en temps réel. Les données gyroscopiques et de l'accéléromètre sont utilisées pour animer ce modèle. Le magnétomètre, qui aurait pu ajouter de la précision, n'a pas été exploité en raison de dysfonctionnements techniques.
- Graphes des Données : Des graphiques ont été mis en place pour afficher les données des accéléromètres et gyroscopes selon les axes X, Y, et Z. D'autres graphes montrent l'altitude, calculée grâce à une formule aérodynamique basée sur la pression barométrique mesurée par le BMP280, tel que :

▷ Pressure at altitude Z can be computed from temperature evolution:

$$p(Z) = p_0\left(1 + \frac{T_z}{T_0}Z\right)^{-\frac{g}{RT_z}} \qquad g = 9.81\ \mathrm{m/s^2}$$

- Vitesse et GPS : Une section est dédiée à l'affichage de la vitesse de la fusée et des données GPS, essentielles pour, respectivement, récolter des données de vitesse pour déduire l'apogée et retrouver facilement la fusée après son lancement.



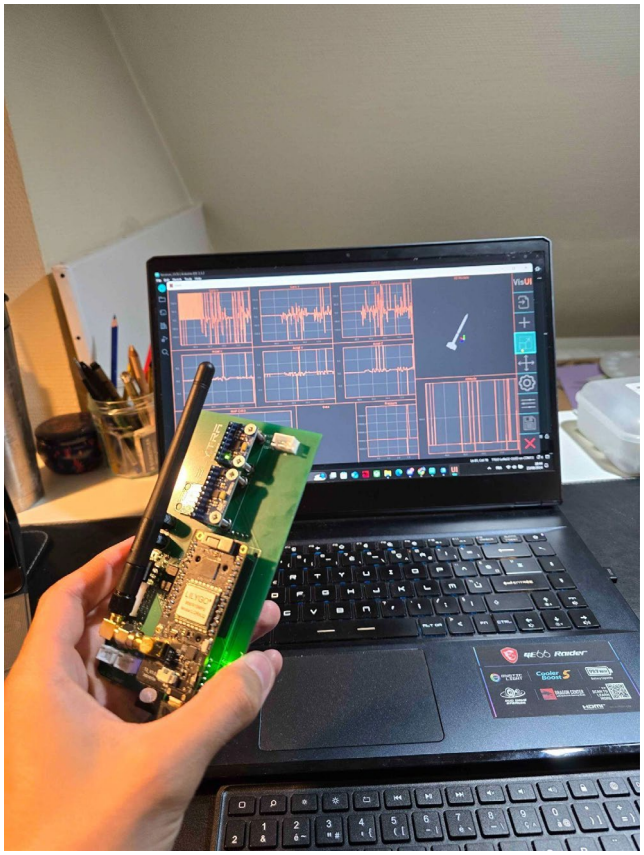


Figure 21 : Vue d'ensemble sur les fonctionnalités de ViSU

D. Système de Sauvegarde des Données (Backup)

Le système de sauvegarde des données est activable directement via le logiciel ViSU et fonctionne automatiquement une fois activé. Les données sont enregistrées sous forme de fichiers texte en binaire, représentant les valeurs brutes reçues par le logiciel.

Sécurité et Intégrité des Données :

Pour garantir l'intégrité des données, un protocole de sauvegarde fragmentée a été mis en place. Chaque fichier de données est limité à 300 ko, après quoi le fichier est fermé et un nouveau fichier est créé. Cette méthode permet d'assurer que, même en cas de problème durant le vol ou la transmission, plusieurs fichiers partiels sont déjà enregistrés et sécurisés sur le PC.

Résultats

Cette section présente les résultats obtenus lors des tests préliminaires de la télémétrie de la FuseX Elytra, ainsi que les événements survenus lors du lancement effectif.

1. Test Pré-Lancement

Avant le lancement de la FuseX Elytra, plusieurs tests de télémétrie ont été réalisés pour vérifier la fiabilité du système. Ces tests ont montré que la télémétrie fonctionnait correctement, même dans des conditions exigeantes.

Test de Longue Distance :

Le test le plus significatif a été effectué à une distance de 3 km entre le système d'acquisition des données (Sender) et le récepteur (Receiver). Ce test a été réalisé dans un environnement complexe, avec une forêt dense et des habitations entre les deux points. Malgré ces obstacles, la transmission des données s'est déroulée sans interruption. Les informations provenant des capteurs ont été reçues avec précision, notamment les données GPS et celles permettant de visualiser le modèle 3D en temps réel du système. Ce succès a confirmé la robustesse de la transmission en utilisant la technologie LoRa à 868 MHz et a validé l'efficacité du système ViSU pour la visualisation en temps réel.

2. Lancement et Incident

Cependant, lors du lancement effectif de la fusée, un incident critique est survenu, entraînant la perte totale des données de télémétrie une seconde après le décollage.

Détachement de la Carte LILYGO :

Le problème a été causé par une défaillance mécanique dans la coiffe de la fusée. La carte LILYGO, qui jouait un rôle crucial en tant que "carte mère" pour la gestion des capteurs et la transmission des données, était fixée sur un PCB monté verticalement à l'intérieur de la coiffe. Bien que ce montage ait été jugé adéquat lors des tests statiques, la poussée intense générée lors du décollage a provoqué un léger mais brusque recul du PCB. En raison d'un jeu dans le rail de maintien du PCB, la carte LILYGO a été projetée contre une partie de la structure interne de la fusée, ce qui l'a détachée de son support.

Cette déconnexion a immédiatement interrompu la transmission des données vers le Receiver, entraînant une perte totale du signal et de toutes les données collectées au cours du vol. Étant donné que la LILYGO gérait également les expériences situées dans la coiffe, cet incident a non seulement compromis la télémétrie, mais aussi l'ensemble des mesures expérimentales prévues.

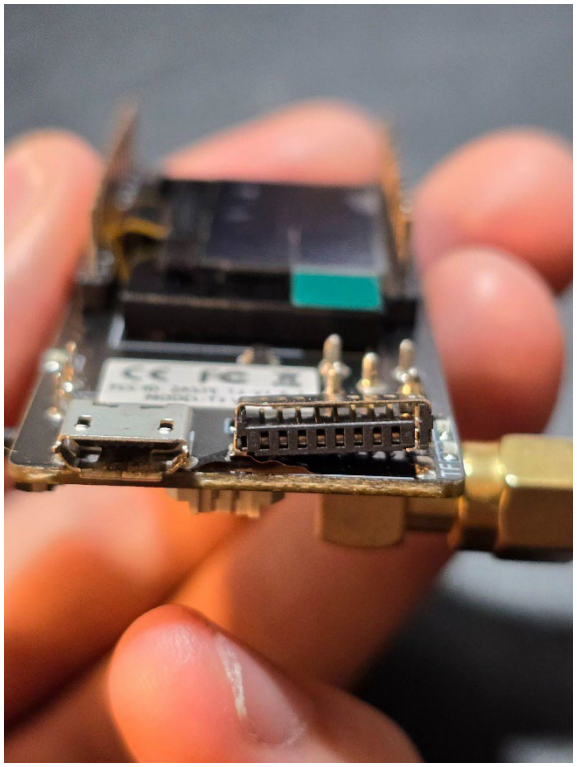


Figure 22 : Impact sur la carte LILYGO

3. Analyse et Leçons Apprises

Bien que cet incident ait empêché la collecte des données critiques lors du lancement, il a néanmoins fourni des enseignements précieux pour l'amélioration future du système :

Robustesse Mécanique :

Il est crucial d'améliorer le montage du PCB et de la carte LILYGO pour résister aux forces dynamiques générées lors du décollage. Un système de fixation plus robuste, possiblement avec un verrouillage supplémentaire ou un amortissement des chocs, pourrait prévenir ce type de défaillance.

Redondance des Systèmes : La centralisation des fonctions critiques sur un seul composant (la carte LILYGO) a créé un point de défaillance unique. Dans les futures itérations, il serait judicieux d'introduire des redondances pour certaines fonctions critiques, comme la gestion des expériences et la transmission des données.

Tests Dynamiques :

Les tests pré-lancement ont démontré la fiabilité du système en conditions statiques ou de faible mouvement, mais les conditions dynamiques spécifiques au décollage n'ont pas été suffisamment simulées. La mise en place de tests dynamiques plus rigoureux, reproduisant les forces d'accélération et de vibration du décollage, pourrait aider à identifier et corriger ces faiblesses avant le vol.

En conclusion, bien que la télémétrie n'ait pas fonctionné comme prévu lors du vol effectif de la FuseX Elytra, les tests préalables ont confirmé le potentiel du système en termes de transmission et de visualisation des données. Les leçons tirées de cet incident seront essentielles pour renforcer la conception et la résilience des futurs systèmes de télémétrie.

Discussion :

Analyse Technique de la Transmission de Données et Conversion en Byte

1. Introduction à la Transmission de Données via LoRa

La communication entre un émetteur (sender) et un récepteur (receiver) repose sur la technologie LoRa (Long Range), réputée pour sa capacité à effectuer des transmissions longue distance tout en minimisant la consommation d'énergie. Le projet en question implique la collecte de données provenant de capteurs variés, dont un accéléromètre/gyroscope (MPU9250), un capteur de pression barométrique (BMP280), un capteur GPS, et un tube de Pitot, avant leur transmission vers le récepteur via LoRa. Cette transmission est essentielle pour le suivi en temps réel, notamment dans des applications telles que la télémétrie de fusées ou de drones, où la robustesse et l'efficacité énergétique sont cruciales.

2. Les Éléments Techniques Clés

2.1. LoRa (Long Range)

LoRa fonctionne dans la bande de fréquences ISM (Industrial, Scientific, and Medical), typiquement autour de 868 MHz en Europe. C'est une technologie de modulation basée sur l'étalement de spectre à large bande (chirp spread spectrum, CSS), permettant des communications longue portée avec une robustesse accrue face aux interférences. Voici quelques paramètres clés utilisés dans ce projet :

Facteur d'étalement (Spreading Factor, SF) : Défini entre 6 et 12, le facteur d'étalement est directement lié à la durée de transmission d'un paquet. Plus le SF est élevé, plus la portée est grande, mais avec une vitesse de transmission réduite. Dans le code, un SF de 7 est utilisé, offrant un compromis équilibré entre portée et débit.

$$Durée\ du\ symbole\ = \frac{2^{SF}}{Bande\ passante\ (BW)}$$

Puissance d'émission (Tx Power) : Configurée à 14 dBm, cette puissance détermine l'énergie émise par le module. En augmentant la puissance, la portée augmente également, mais au détriment de la consommation énergétique. La puissance maximale admissible est de 17 dBm pour la bande ISM en Europe.

Largeur de bande du signal (Signal Bandwidth, BW) : Fixée à 125 kHz, la largeur de bande affecte la vitesse de transmission et la sensibilité du récepteur. Une largeur de bande plus étroite améliore la portée et la sensibilité, mais réduit la vitesse de transmission.

$$Vitesse\ de\ transmission\ = \frac{Bande\ passante\ (BW)}{2^{SF}}$$

Taux de codage (Coding Rate, CR) : Un taux de codage de 4/5 est généralement utilisé, permettant de corriger des erreurs sur la base des données redondantes ajoutées. Cela améliore la robustesse des transmissions aux dépens de la capacité utile.

2.2. Sérialisation et Conversion en Byte

La sérialisation est le processus de conversion des données complexes en un format linéaire (généralement une chaîne de caractères) pour les transmettre via des réseaux comme LoRa. Dans ce projet, les données des capteurs sont d'abord converties en une chaîne de caractères où chaque valeur est séparée par une virgule. Cette chaîne est ensuite convertie en bytes avant d'être envoyée.

Concaténation des données : Les valeurs des capteurs sont concaténées sous la forme d'une chaîne unique, facilitant leur transmission sous forme de paquet LoRa. Par exemple, les valeurs d'accélération en 'g' et de rotation en '°/s' sont mises en forme en tant que chaîne comme suit :

$$Données\ sérialisées\ = String(corr\ ax)\ +\ ","\ +\ String(corr\ ay)\ +\ ","+ \ldots$$

3. Conversion des Données en Byte

3.1. Conversion et Optimisation de l'Usage des Données

Les données des capteurs, comme les coordonnées GPS en format flottant (float), sont optimisées avant leur transmission. Par exemple, les valeurs de latitude et longitude sont multipliées par 10^6 et converties en entiers. Cette approche réduit la taille des données tout en maintenant une précision suffisante.

Conversion des coordonnées GPS :

$$Latitude\ =\ latitude\ x\ 10^6$$

$$Longitude\ =\ longitude\ x\ 10^6$$

Cela transforme une valeur flottante en un entier long, réduisant la taille des données à transmettre tout en évitant les pertes de précision.

- Méthode LoRa.print() :

Le code utilise la méthode LoRa.print(serializedData) pour envoyer la chaîne sérialisée. Cette méthode convertit implicitement chaque caractère en byte, permettant une transmission efficace.

4. Traitement des Données au Niveau du Récepteur

4.1. Dé-sérialisation et Reconstitution des Données

Une fois les données reçues sous forme de bytes au niveau du récepteur, elles doivent être reconverties en un format compréhensible. Cette étape critique assure que les informations des capteurs sont correctement interprétées.

- Dé-sérialisation : Lors de la réception, les données arrivent sous forme de chaînes de caractères compactées en bytes. Le processus de dé-sérialisation consiste à lire ces données byte par byte pour reconstituer la chaîne d'origine. Ensuite, cette chaîne est divisée en sous-chaînes basées sur des séparateurs prédéfinis (par exemple, des virgules), permettant de retrouver les valeurs numériques initiales. Ces valeurs sont ensuite associées aux différentes mesures des capteurs, telles que l'accélération, la température, la pression, ainsi que la position géographique (latitude et longitude).

$$Données\ reçues -> Chaîne\ dé-sérialisée -> Valeurs\ individuelles$$

- Utilisation des données : Une fois les données dé-sérialisées, elles sont converties en leur type numérique d'origine (par exemple, en float pour des mesures comme l'accélération ou la température). Cette conversion est cruciale pour permettre leur exploitation ultérieure, que ce soit pour des calculs, des analyses ou leur affichage sur un logiciel externe. Cette phase assure que les données reçues sont prêtes à être utilisées de manière efficace et précise.

En résumé, l'efficacité de la transmission de données via LoRa dépend de la configuration optimale des paramètres de transmission, ainsi que de la sérialisation et dé-sérialisation des données. Le choix de ces paramètres a un impact direct sur la portée, la vitesse, et la robustesse des communications, ce qui est crucial pour des applications exigeantes telles que la télémétrie en temps réel.

5. Performance et Fiabilité du Système

Facteurs d'Étalement et Robustesse :

- Le facteur d'étalement choisi (Spreading Factor) impacte la robustesse de la communication. Un facteur plus élevé augmente la portée mais réduit la vitesse de transmission.
- La puissance d'émission et la largeur de bande sont configurées pour équilibrer entre la portée nécessaire et la consommation d'énergie, un compromis crucial pour les systèmes embarqués où l'énergie est limitée.

Optimisation des Ressources :

- Le code inclut des mécanismes pour calibrer les capteurs et compenser les erreurs de lecture, comme le capteur de pression différentielle (Pitot), ce qui assure que les données transmises sont précises et fiables.
- L'utilisation d'une moyenne glissante pour les données de pression améliore la stabilité des lectures, réduisant ainsi le bruit dans les données transmises.

6. Conclusion

La transmission de données par LoRa dans ce projet est une application typique de l'IoT (Internet of Things), où l'efficacité de la communication, la robustesse et la faible consommation d'énergie sont prioritaires. L'implémentation technique, incluant la sérialisation des données et l'optimisation des transmissions, démontre une compréhension approfondie des défis associés à la transmission de données dans des environnements contraints. Ce système est non seulement conçu pour être efficace mais aussi pour maximiser la précision des données reçues, ce qui est essentiel dans des applications critiques telles que la télémétrie en temps réel pour des fusées.

Conclusion Générale Télémétrie

Le projet de télémétrie pour la FuseX Elytra a permis de démontrer l'efficacité et la complexité de la collecte et de la transmission de données en temps réel dans un environnement dynamique et exigeant. Malgré un incident critique lors du lancement effectif, ce projet a été une réussite significative en termes de conception, d'optimisation, et d'implémentation des systèmes de télémétrie.

Les tests préliminaires ont prouvé la robustesse du système, notamment la transmission des données sur de longues distances, même dans des environnements complexes. Le système de télémétrie, intégrant des capteurs avancés et un logiciel de visualisation en temps réel, a montré son potentiel à offrir une représentation précise et détaillée des données expérimentales, essentielles pour le suivi et l'analyse des performances de la fusée.

L'incident survenu lors du lancement a néanmoins souligné des points d'amélioration cruciaux, particulièrement en ce qui concerne la robustesse mécanique des composants critiques et la nécessité d'une redondance pour éviter les points de défaillance unique. Les leçons tirées de cette expérience, notamment l'importance des tests dynamiques et de la conception résiliente, seront déterminantes pour le succès des futures missions.

En conclusion, bien que la télémétrie de la FuseX Elytra n'ait pas fonctionné comme prévu lors du vol, le projet a permis d'acquérir une expérience précieuse et a jeté les bases pour des améliorations futures. Ces efforts seront essentiels pour garantir que les systèmes de télémétrie de prochaine génération soient encore plus fiables, robustes et capables de surmonter les défis des environnements extrêmes.

III. Structure

1. La peau en fibre de carbone

La construction du tube en fibre de carbone a été une étape clé du projet. Le processus a débuté par l'utilisation d'un tube en PVC (diamètre 125mm) qui a servi de mandrin pour former la base du futur tube en fibre de carbone. Ce mandrin a permis de définir le diamètre intérieur du tube tout en offrant un support rigide pendant la fabrication.

Figure 23 : Réalisation du tube

Nous avons enroulé un tissu de fibre de carbone tressée (300g/m²) autour de ce mandrin, en appliquant simultanément de la colle époxy. Cette méthode permettait à la fibre de carbone d'être imbibée directement de résine époxy au fur et à mesure de l'enroulement. Cela assurait une adhésion optimale des fibres, garantissant une solidité accrue après durcissement. Il était essentiel de maintenir un enroulement précis et régulier tout au long de cette étape afin d'éviter les irrégularités dans la structure.

Figure 24 : Application de la résine époxy sur le tissu de carbone

Après avoir entièrement recouvert le tube de PVC avec la fibre de carbone imprégnée d'époxy, nous avons laissé la résine durcir pendant 72h. Une fois la solidification complète, le mandrin en PVC a été retiré après quelques difficultés.

Ce tube constitue la peau extérieure de la fusée et ajoute une rigidité supplémentaire au squelette interne.

Figure 25 : Tube terminé et laissé en séchage

2. Les ailerons renforcés en composites

La construction des ailerons de la fusée a été réalisée en utilisant des matériaux composites pour garantir légèreté et solidité. Ces ailerons jouent un rôle crucial dans la stabilité lors du vol de la fusée.

La première étape de la fabrication a consisté à coller des plaques de fibre de G10 de 5mm d'épaisseur (plaque de fibre de verre compressée) directement au tube en fibre de carbone. L'assemblage a été réalisé à l'aide de colle époxy à durcissement rapide (5 min) pour maintenir les ailerons en position avant de les consolider.

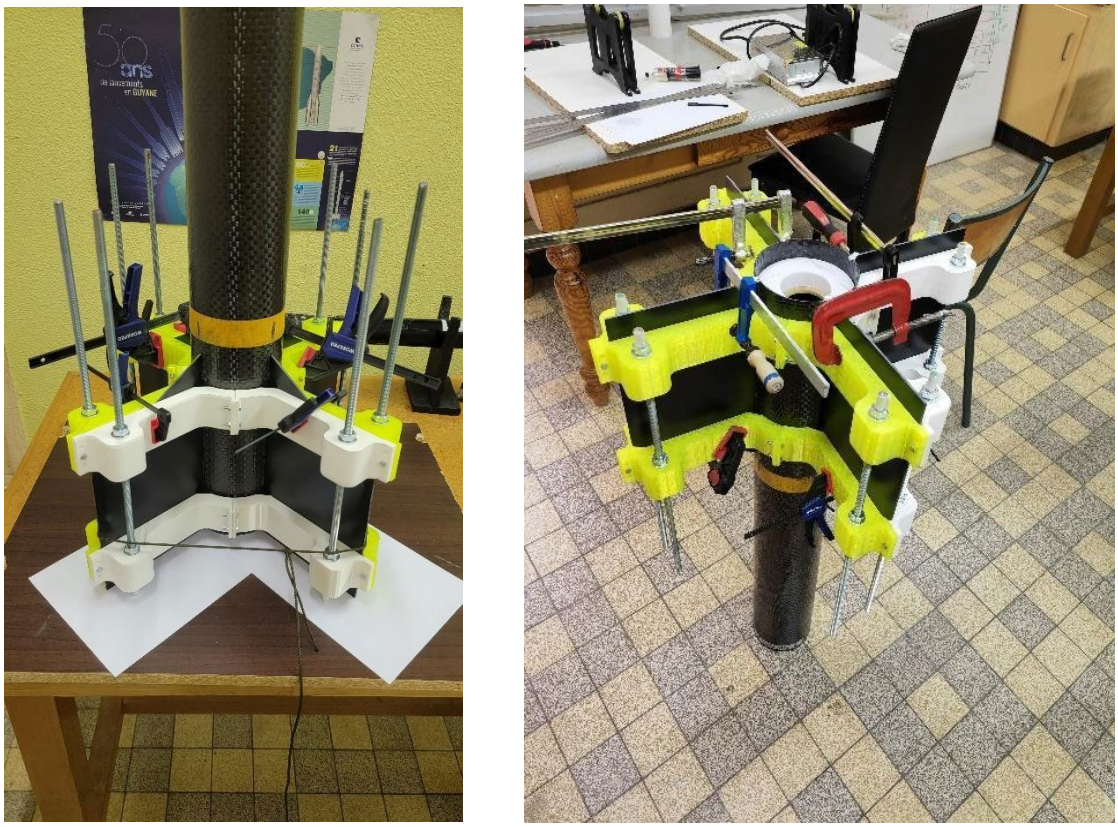


Figure 26 : Collage des ailerons

Une fois les plaques en place, elles ont été minutieusement poncées à plusieurs reprises pour assurer une bonne liaison chimique avec l'époxy. De la colle époxy supplémentaire a été ajoutée pour les congés de raccordement. Après séchage, 6 couches de fibre de verre tissée (200g/m²) ont été positionnées en orientation 0°-45°-0°-45°-0°-45° pour renforcer la solidité des ailerons.

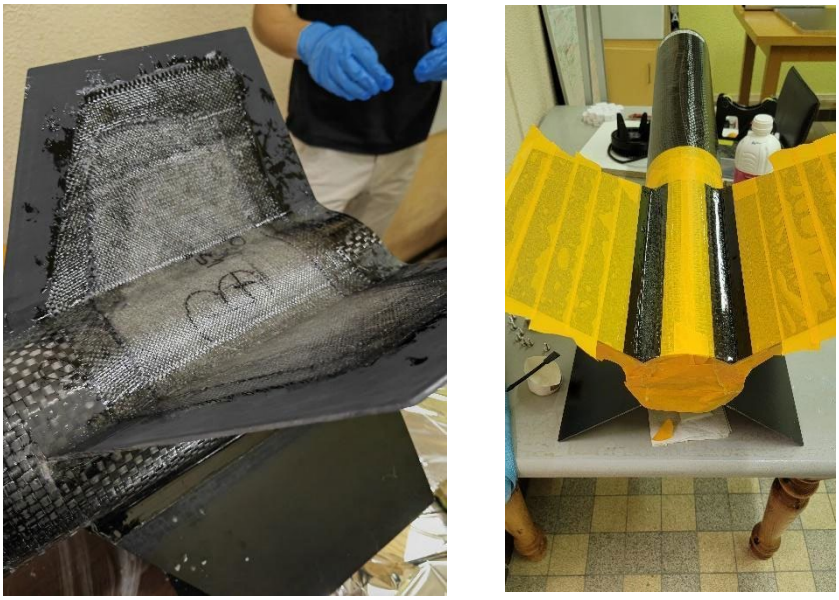


Figure 28 : Renforcement des ailerons

Enfin, pour l'esthétique, les ailerons ont été recouverts d'une couche de fibre de carbone (celle utilisée pour le tube). Un tissu Peel Ply est posé par-dessus pour absorber l'excédent d'époxy. Cette finition extérieure confère aux ailerons un aspect lisse au carbone accordé avec le tube.

Figure 27 : Séchage de la résine époxy

Grâce à cette combinaison de matériaux, les ailerons offrent à la fois légèreté, robustesse et une résistance suffisante pour maintenir la stabilité de la fusée pendant le vol.

3. Trappe du parachute

La trappe de la fusée a été fabriquée en utilisant une méthode similaire à celle employée pour le tube principal, avec des adaptations spécifiques pour répondre aux exigences de cette partie essentielle. La trappe devait s'intégrer parfaitement au tube tout en étant suffisamment solide pour résister aux contraintes du vol.

Pour commencer, nous avons utilisé un moule en PLA, imprimé en 3D, qui reproduit exactement le diamètre du tube principal. Le PLA a été choisi pour sa précision dimensionnelle et sa capacité à être facilement formé en formes complexes. Ce moule a servi de base pour la fabrication de la trappe.

Ensuite, nous avons appliqué trois couches de fibre de carbone sur ce moule. L'orientation des fibres a été soigneusement planifiée pour maximiser la résistance et la rigidité de la trappe. La première et la troisième couche ont été disposées avec une orientation de 0°, parallèlement à l'axe de la trappe, pour assurer une excellente résistance longitudinale. La couche intermédiaire a été placée avec une orientation de 45°, afin d'améliorer la résistance aux torsions et d'assurer une répartition homogène des contraintes.

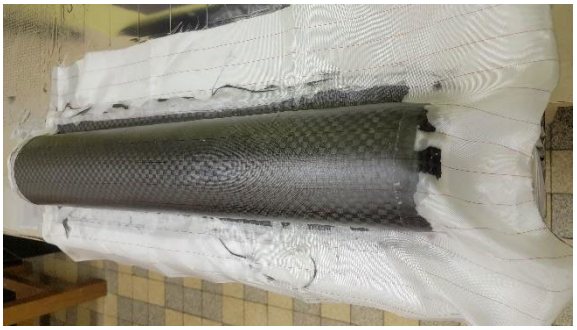


Figure 29 : Fabrication de la trappe

Chaque couche de fibre de carbone a été imprégnée de colle époxy au fur et à mesure de l'application, suivant le même procédé que pour le tube principal. Pour garantir une finition propre et retirer l'excédent d'époxy, nous avons utilisé un Peel Ply. Ce tissu spécial, appliqué sur la dernière couche de fibre de carbone, permet de drainer l'excès de résine époxy et de créer une surface texturée. Après le durcissement de l'époxy, le Peel Ply a été retiré, emportant avec lui l'excédent de résine et laissant une surface propre, prête pour d'éventuelles finitions.

Figure 30 : Fabrication de la trappe

4. La coiffe en fibre de verre

La coiffe de la fusée a été fabriquée en fibre de verre tressée, un matériau choisi non seulement pour sa légèreté et sa résistance, mais aussi pour ses propriétés spécifiques par rapport à la télémétrie embarquée. Contrairement à la fibre de carbone, la fibre de verre permet le passage des ondes électromagnétiques, ce qui est crucial pour les équipements de télémétrie logés à l'intérieur de la coiffe.

Tout comme la trappe, pour fabriquer la coiffe, un moule en PLA a été utilisé. La coiffe a été constituée de six couches principales de fibre de verre tressée, appliquées selon un schéma d'orientation spécifique pour maximiser sa résistance et sa durabilité. Les couches ont été disposées dans l'ordre suivant : 0°/45°/0°/45°/0°/45°. Cette alternance des orientations assure une excellente résistance dans toutes les directions, permettant à la coiffe de supporter les forces aérodynamiques et les éventuelles contraintes mécaniques subies en vol.

Figure 31 : Préparation d'une couche en fibre de verre

Chaque couche de fibre de verre a été divisée en deux demi-coiffes pour mieux épouser la forme complexe de la coiffe. Cela a permis d'éviter les plis et les déformations, tout en garantissant une application uniforme sur toute la surface du moule. Pendant l'application, chaque couche a été soigneusement imprégnée de colle époxy, assurant une bonne adhérence entre les fibres et une solidité optimale une fois durcies.

Figure 32 : Fabrication de la coiffe

Pour finir, une septième couche très fine de fibre de verre a été appliquée. Cette couche finale avait pour but d'offrir une surface plus lisse et homogène, améliorant l'aérodynamique de la coiffe tout en lui donnant un aspect fini.

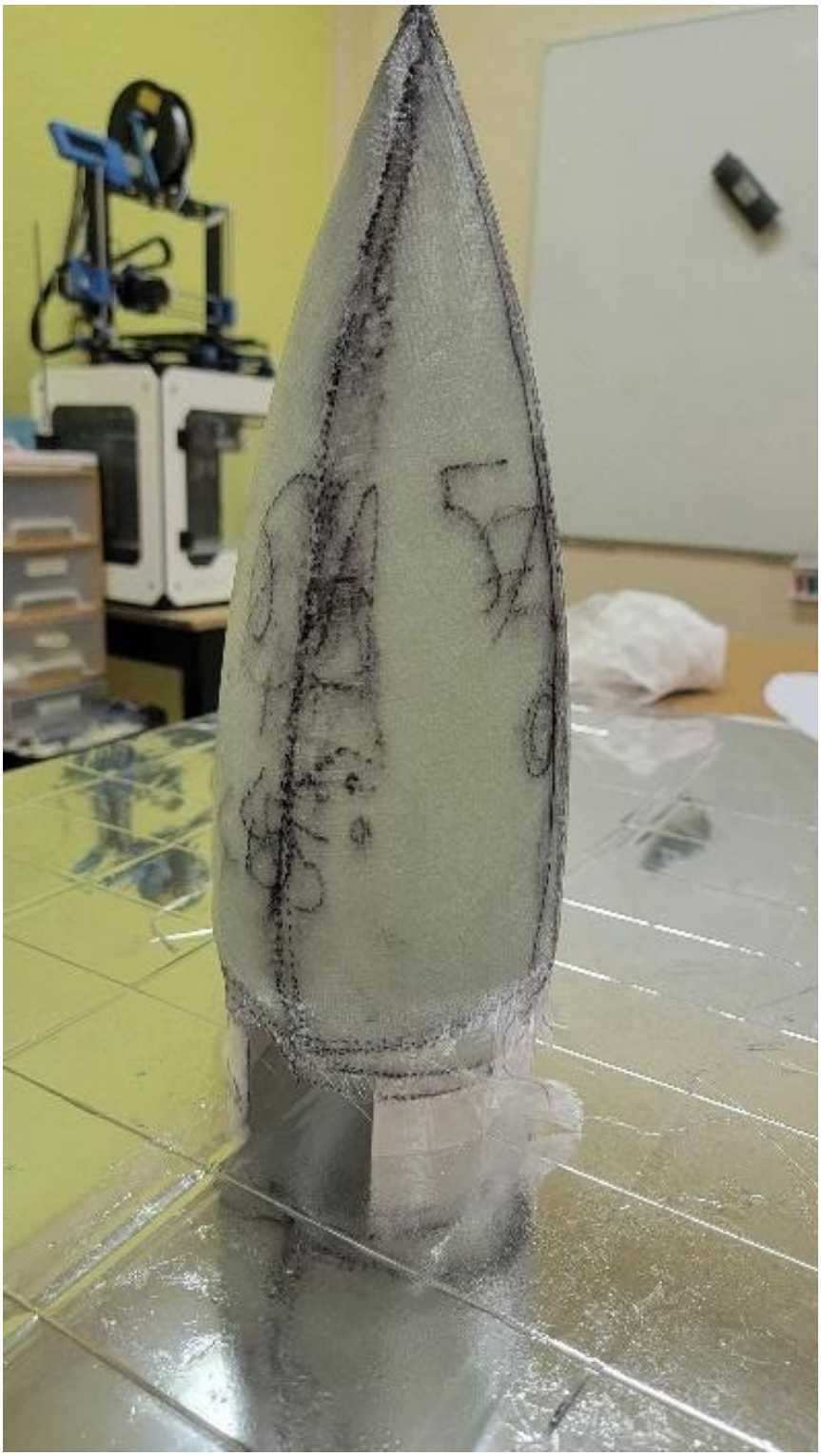


Figure 33 : Coiffe en fibre de verre avec toutes ses couches

5. Le squelette porteur

Dans les lancements de fusées de notre club, nous n'avions jamais utilisé de matériaux composites. Le choix d'un squelette porteur pour Elytra nous a servi de point de départ serein sur la rigidité de l'ensemble. La fabrication d'une peau en fibre de carbone était une nouvelle expérience pour l'association. Nous n'avons donc pas pris le risque de fabriquer une peau porteuse, surtout au niveau de la caméra stabilisée qui a besoin d'une ouverture à 360 degrés.

Le squelette est composé de 4 cornières en aluminium et de plusieurs bagues en PLA tout au long de la fusée pour maintenir le tout. Le squelette part du bas des ailerons à la base de la coiffe.

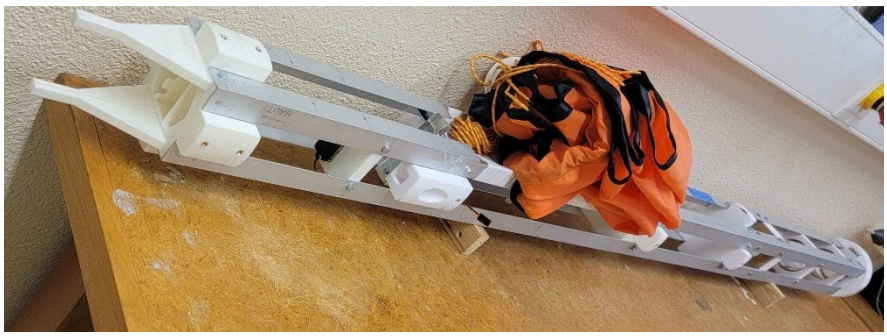


Figure 34 : Squelette porteur

Les bagues de fixation du moteur PRO4 sont faites en PLA et en bois pour résister à la chaleur.

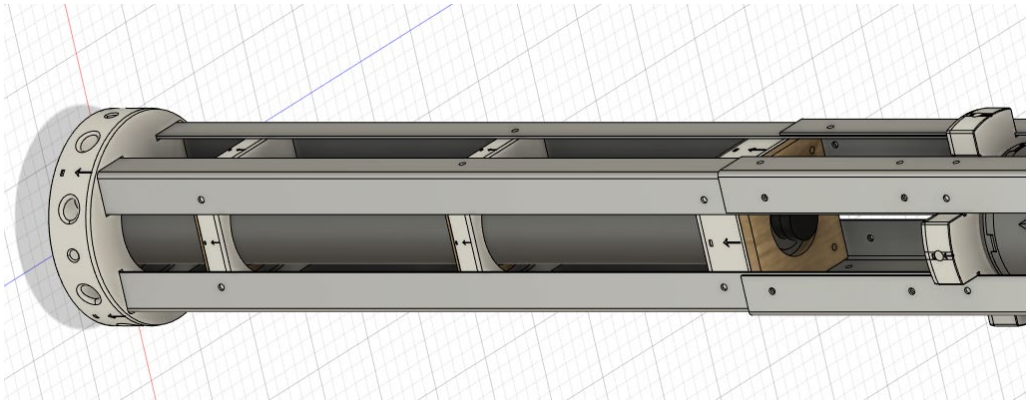


Figure 35 : CAO de la fixation du moteur au squelette

La bague à la base des ailerons reprend la poussée du moteur. Des pièces en aluminium font le contact avec le moteur. La transmission des forces se fait du moteur à la bague et de la bague aux cornières.

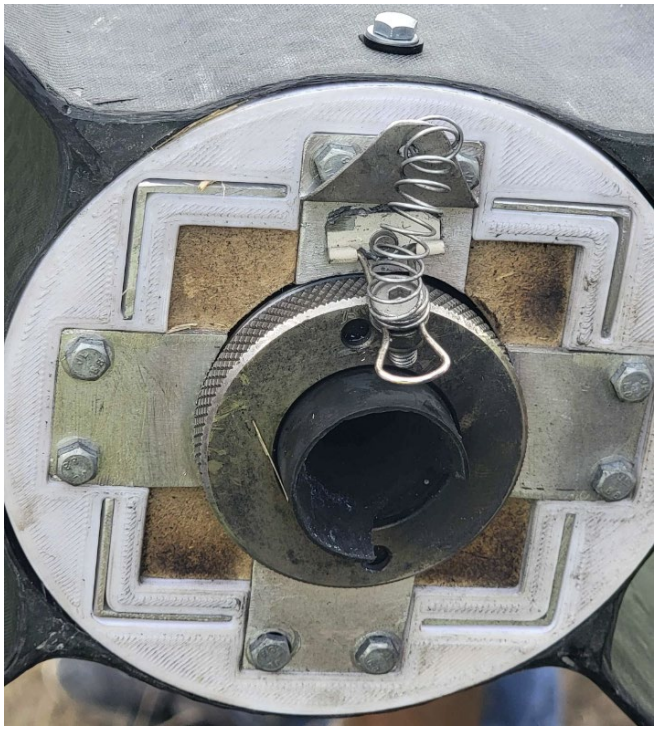


Figure 36 : Bague de reprise de poussée du moteur, photo prise après le vol

6. Parachute

Le parachute de la fusée Elytra est un composant essentiel, conçu pour assurer une descente sécurisée après le vol. Il est fabriqué en toile de polyester, renforcée avec des côtés en satin pour améliorer sa résistance et sa durabilité. Le parachute adopte une forme de croix, avec un diamètre de 180 cm et une largeur de 60 cm, afin de garantir une vitesse de descente inférieure à 15 m/s, ce qui est crucial pour la sécurité et l'intégrité de la fusée lors de l'atterrissage.

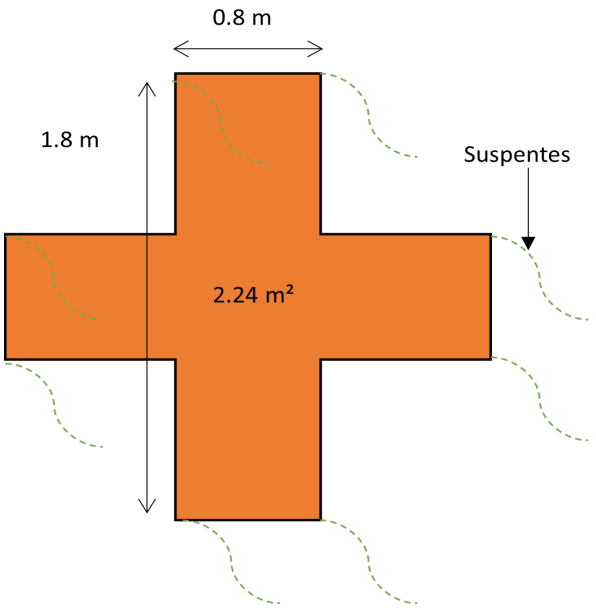


Figure 37 : Dimensions du parachute

Ce parachute a une dimension humaine et particulière puisqu'il a été confectionné par la grand-mère de l'un des membres de l'équipe. Son expertise en couture a été mise à profit pour s'assurer que chaque couture et chaque renforcement soient parfaits, offrant ainsi une solidité optimale au parachute.

Les suspentes, qui jouent un rôle crucial dans le déploiement et la stabilisation du parachute, sont des cordes d'escalade robustes, chacune capable de supporter une charge de 20 kg. Chaque côté du parachute est équipé de 270 cm de suspentes, avec un total de 8 suspentes assurant une répartition équilibrée des forces. Ces suspentes sont solidement attachées à un émerillon, lequel est fixé à une sangle pour assurer une connexion fiable et stable.

Une fois le parachute terminé, il est soigneusement placé dans un compartiment dédié à bord de la fusée. Ce compartiment est équipé d'un tissu en lycra, un matériau élastique qui permet d'accueillir le parachute et facilite son déploiement une fois que la fusée est en vol. Le tissu en lycra est fixé à deux cornières, assurant ainsi une sortie fluide et sans accroc du parachute au moment opportun.

Figure 38 : Photo du parachute au C'Space

IV. Retours d'expérience et conclusion

1. Analyse du vol et des données

Ce projet a été une véritable aventure, marquée par de nombreux défis que nous avons dû surmonter pour mener à bien notre mission. Lors du vol simulé, la caméra stabilisée a été mal placée, compromettant ainsi toute l'expérience. Nous avons dû reconstruire l'ensemble en une nuit pour repasser les contrôles le lendemain. Afin de prévenir ce type d'incident à l'avenir, nous avons pris la décision de fixer la caméra à l'avance dans la fusée et de renforcer les vérifications lors de la chronologie sur la rampe de lancement.

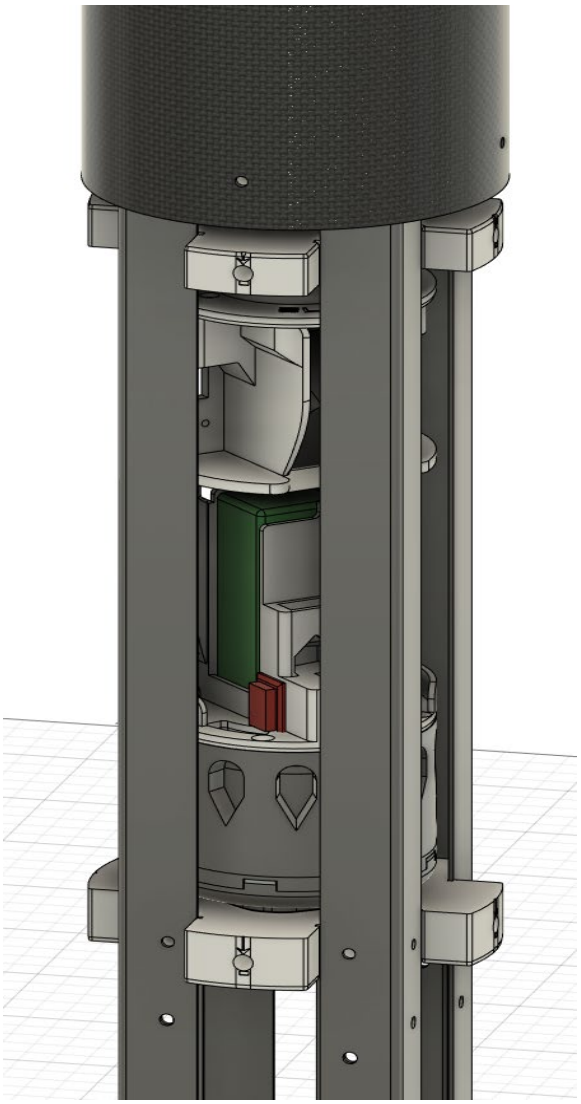
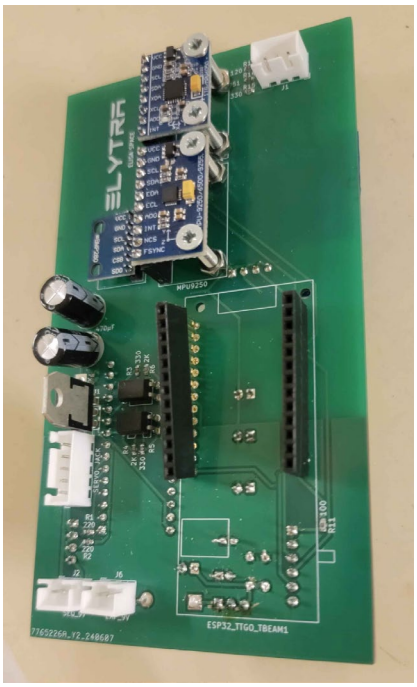
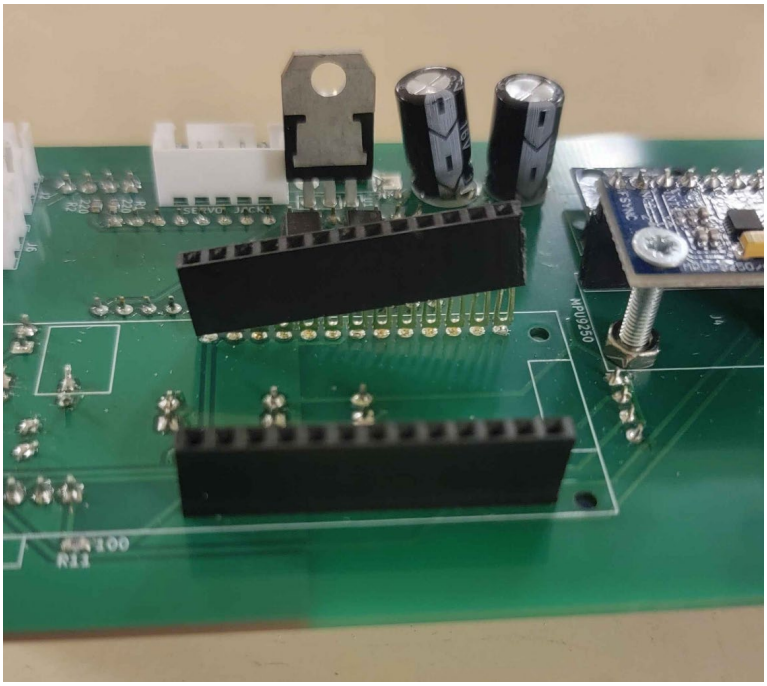


Figure 39 : CAO du stabilisateur de la caméra dans la fusée

Lors du vol, de nouvelles complications sont apparues. Sous l'effet de l'accélération, la carte LilyGO a été arrachée, empêchant toute transmission de télémétrie. De plus, le PCB a été enfoncé, ce qui a endommagé la carte SD et rendu son utilisation impossible.





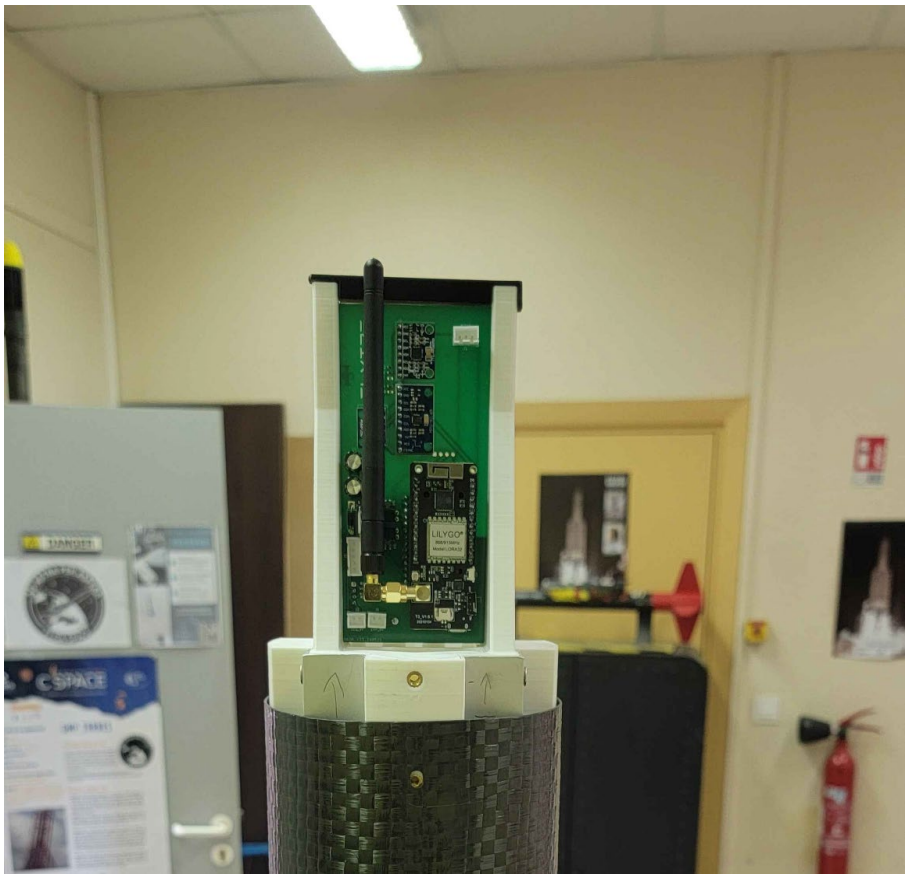


Figure 40 : Intégration du PCB (cartes expérience et séquenceur) dans la coiffe

D'autres imprévus ont également affecté nos expériences. L'ESC (contrôleur de vitesse électronique) s'est mis en veille, ce qui a empêché le fonctionnement de la stabilisation de la caméra. En conséquence, la caméra a filmé une cornière tout au long du vol, sans capturer les images prévues. Par ailleurs, l'autonomie limitée des petites caméras très low cost n'a pas permis d'enregistrer de vidéos du vol.

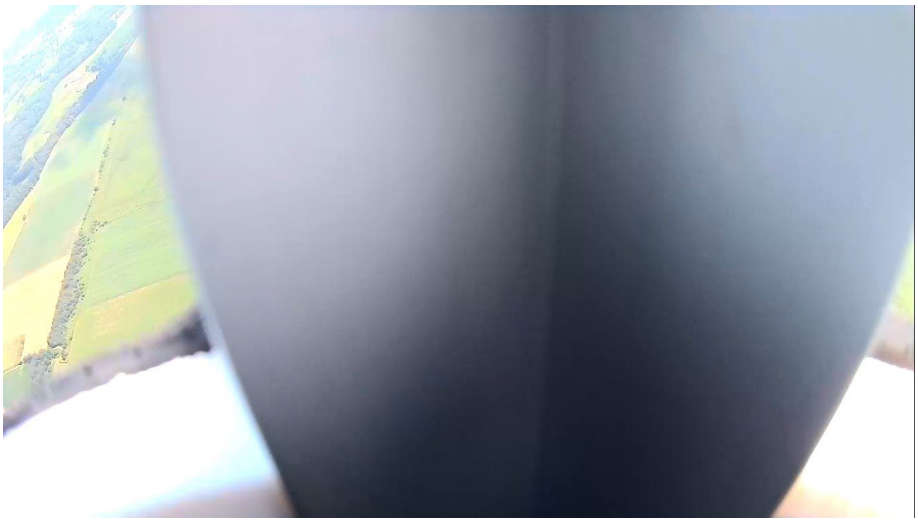


Figure 41 : Cornière filmée tout au long du vol

Malgré ces nombreux obstacles, notre équipe a persévéré et a su démontrer une grande capacité d’adaptation. Le vol, bien que marqué par des difficultés, a été nominal, et nos efforts ont été reconnus et récompensés par le prix du coup de cœur du jury, toutes catégories confondues. Ce prix représente une reconnaissance précieuse pour le travail accompli et témoigne de notre engagement et de notre détermination à surmonter les défis rencontrés.

Figure 42 : Récompenses du C'Space 2024

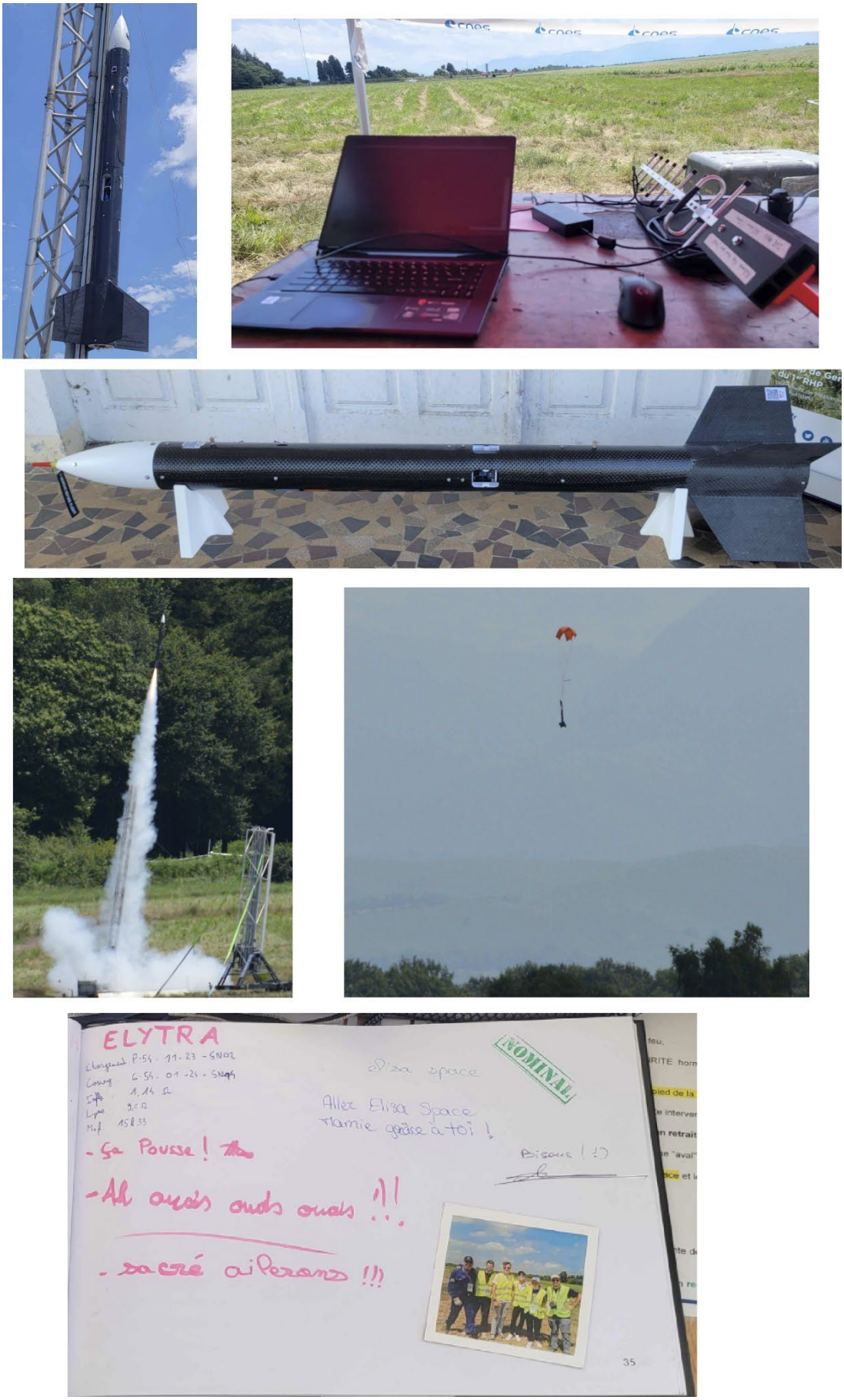


Figure 43 : Photos du C'Space

2. Remerciements

Nous souhaitons exprimer notre sincère gratitude à tous ceux qui ont contribué au succès de ce projet au sein d'ELISA Space. Nous remercions chaleureusement tous les membres de l'association qui, de près ou de loin, ont apporté leur aide précieuse tout au long de cette aventure. Leur dévouement, leur expertise et leur passion ont été des éléments clés pour surmonter les défis que nous avons rencontrés.

Nous tenons également à adresser nos remerciements les plus sincères à nos partenaires, sans qui ce projet n'aurait pu être mené à bien. Nos remerciements vont aux entreprises ProTeam, au garage LAAS Renault et au groupe Blondel, dont le soutien financier a été essentiel pour faire avancer l'association et concrétiser notre vision. Leur confiance et leur engagement envers notre projet ont été une source d'inspiration et de motivation pour nous tous.

Annexes

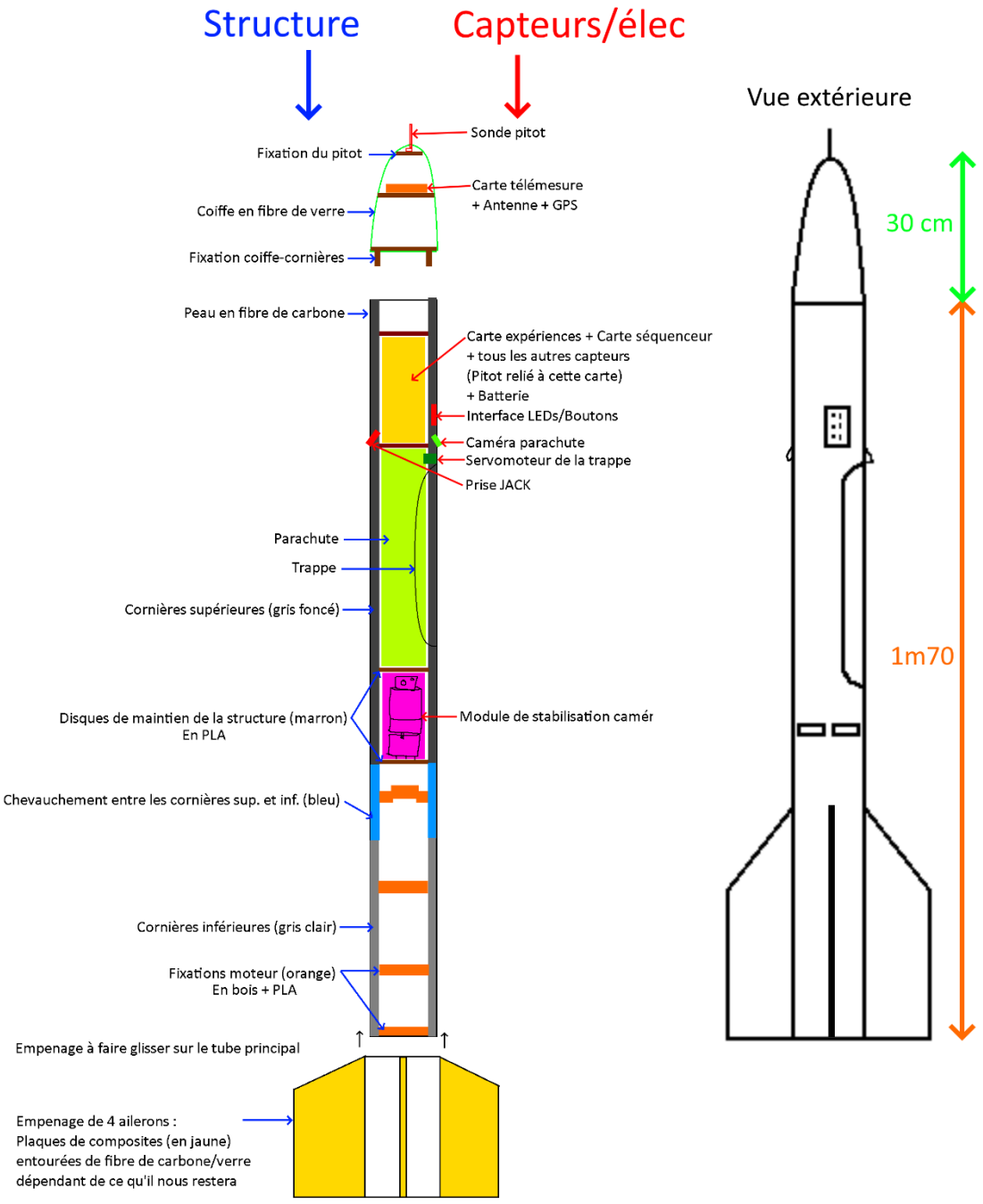


Figure 44 : Schéma de la fusée

Programme du séquenceur :

```
/**************************************/
#include <Servo.h> //bibliothéque servo (déjà présente sur arduino)
Servo Servomoteur; //créer un objet servo
#define LED_POWER 5
#define LED_STATE 4 // 4
#define JACK 2
#define PIN_SERVO 3
#define RX 6
#define TX 7

const int Position_FERME = 125; //1
const int Position_OUVERT = 1; //125
const int t_apogee = 14400; //temps de l'apogée en millisec

unsigned long dt1_millis = t_apogee - 2000;    //temps avant plage d'ouverture
para en millisec
unsigned long dt2_millis = t_apogee + 2000;    //temps ouverture para
obligatoire en millisec

unsigned long t1_millis;
unsigned long t2_millis;
unsigned long t0;

void setup() {
  Servomoteur.attach(PIN_SERVO);   // Attache le servomoteur au PinServo
  Servomoteur.write(Position_FERME);          // Met le servomoteur en
position

  pinMode(JACK, INPUT_PULLUP);              //set as INPUT
  pinMode(LED_POWER, OUTPUT);
  pinMode(LED_STATE, OUTPUT);
  pinMode(RX, INPUT);
  pinMode(TX, OUTPUT);

  while(digitalRead(JACK) == HIGH){
    doblinkall();
  }

  digitalWrite(LED_POWER, HIGH);
  digitalWrite(LED_STATE, LOW);
}

void loop() {
  if(digitalRead(JACK) == HIGH){
    t0 = millis();                          // prend T0 mission
    t1_millis = dt1_millis+t0;
    t2_millis = dt2_millis+t0;
```

```
    digitalWrite(TX, HIGH);

    while(millis()<=t1_millis){             //attente début phase d'ouverture
      doblink();
    }

    while(millis()<=t2_millis){
      digitalWrite(LED_STATE, LOW);//attente début fin d'ouverture Attente
ordre ouverture
      digitalWrite(TX, LOW);
      if(digitalRead(RX) == HIGH){             //Si ordre d'ouverture
        digitalWrite(TX, HIGH);                                //Ouverture para
auto
        Ouverture();
        while(1){doblinkall();}
      }
    }
    //Si pas d'odre d'ouverture dans le temps impartis, ouverture temps
    digitalWrite(TX, LOW);
    Ouverture();
    digitalWrite(LED_STATE, HIGH);
    while(1){}
  }
}

void Ouverture(){
  Servomoteur.write(Position_OUVERT);
}

void doblink(){
  if(millis() % 150<75){
    digitalWrite(LED_STATE, LOW);
  }
  else{
    digitalWrite(LED_STATE, HIGH);
  }
}

void doblinkall(){
  if(millis() % 150<75){
    digitalWrite(LED_STATE, LOW);
    digitalWrite(LED_POWER, LOW);
  }
  else{
    digitalWrite(LED_STATE, HIGH);
    digitalWrite(LED_POWER, HIGH);
  }
}
```

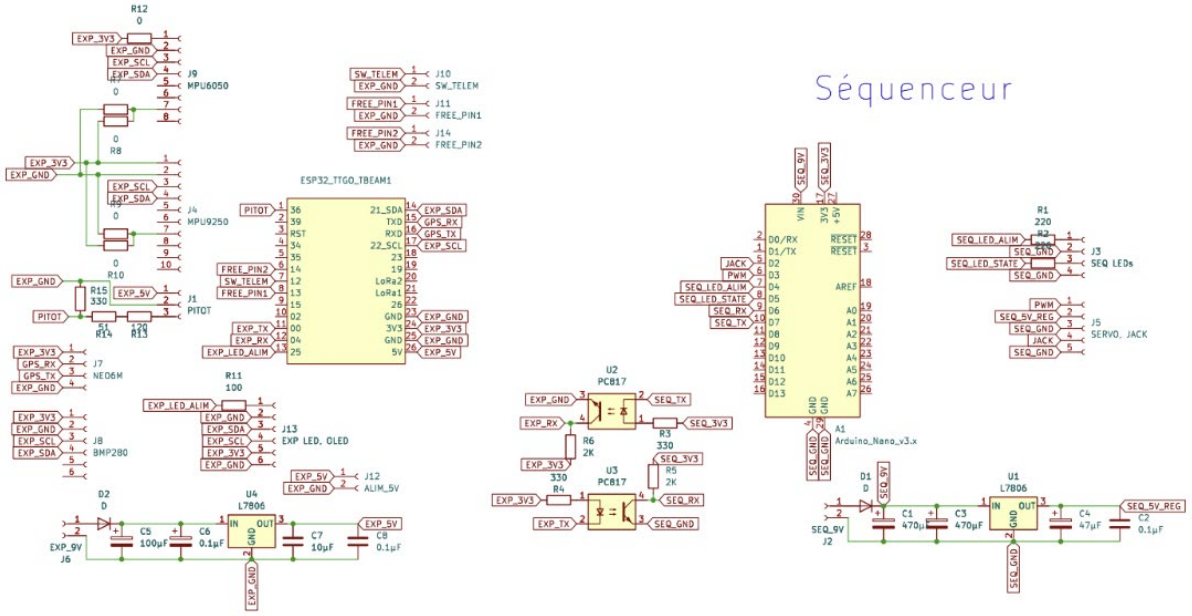


Figure 45 : Schéma électrique du PCB contenant le séquenceur et la carte expérience

Programme de la carte expérience Lilygo (avec télémesure) :

```
#include <Wire.h>
#include <SPI.h>
#include <LoRa.h>
#include <MPU9250.h>
//#include <Adafruit_MPU6050.h>
#include <Adafruit_BMP280.h>
#include <Adafruit_Sensor.h>

#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#include <TinyGPSPlus.h>
#include <mySD.h>

#define SS_PIN 18
#define RST_PIN 23
#define DIO0_PIN 26
// If LILYGO 18, 23, 26
// 5, 14, 2
#define SCL 22
#define SDA 21

File SDdata;

// Define the I2C addresses for the sensors
#define MPU6050_ADDR 0x68  // MPU-6050
#define MPU9250_ADDR 0x69  // MPU-9250
```

```
//#define AK8963_ADDRESS 0x0C // AK8963 (magnetometer) I2C address
#define BMP_ADDR 0x76   // BMP280

// Telem variables
const long FREQUENCY = 868E6;
const int DELAY_BETWEEN_TRANSMISSIONS =50;
int SPREADING_FACTOR = 7;
int TX_POWER = 2;

// MPU variables
float corr_ax, corr_ay, corr_az, corr_gx, corr_gy, corr_gz;
float corr_az_16g;
// Magneto variables
//float mx, my, mz;

// Define the scale factors for the MPU sensors
float accelScaleFactor = 16384.0; // For +/- 2g
float gyroScaleFactor = 32.8; // Scale factor: 32.8 LSB/°/s for 1000
degrees/sec

// Initialize the BMP280 sensor
Adafruit_BMP280 bmp;

// BMP variables
float temperature;
float pressure, corr_pressure, last_pressure;
#define N 10 // Moyenne glissante sur N valeurs
float HistoriqueSignal[N];
float TotalBufferCirculaireSignal=0; // Stockage du total de toutes les
valeurs du buffer.
byte Plus_ancienSignal=0;

// Pitot
const float R = 8.314; // Constante universelle de gaz parfaits
const float rho = 1.204; // Masse volumique de l'air
const float temp = 293.15; // Température ambiante
const float voltageMax = 3.3; // voltage maximal (correspondant à la plage du
pin analogique de l'ESP32)
const float kpaRangeTopVoltage = 3.33; // valeur maximale renvoyée par le
capteur, par l'intermédiaire d'un pont diviseur de tension.

const int numReadings = 10; // Nombre de lectures pour la moyenne glissante
int readings[numReadings] = {0}; // Tableau pour stocker les lectures des
capteurs
int currentIndex = 0; // Index actuel pour le tableau des lectures
int total = 0; // Somme des lectures pour le calcul de la moyenne
int sensorPin = 36; // Pin analogique où le capteur est connecté
```

```
int sensorValue = 0, sensorMax = 4096, sensorOffset = 0; // Variables pour la valeur du capteur, valeur max du capteur, et l'offset
float voltage = 0, kpa = 0, speed = 0, lastspeed = 0, mach = 0; // Variables pour la tension, pression en kPa, vitesse, dernière vitesse et nombre de Mach
// bool calibrationDone = false; // Booléen pour vérifier si le calibrage est effectué

// GPS
static const uint32_t GPSBaud = 9600;
double lat, lng;
TinyGPSPlus gps;

// General
#define LED 25 // LED Bleue
#define SIGNAL 0 // Signal parachute
#define S_SEQ 4 // Signal Séquenceur
#define SW_TELEM 12 // Switch télem
bool signal_seq_recu;
bool apogee_sent;
uint16_t t0;

// Structuration télémetrie
struct TelemetryData {
  float corr_ax;
  float corr_ay;
  float corr_az;
  float corr_gx;
  float corr_gy;
  float corr_gz;
  /*float mx;
  float my;
  float mz;*/
  float lat;
  float lng;
  float temperature;
  float corr_pressure;
};

// ================================================ SETUP

void setup() {

  ////////////////////////////////////////////////
  // TELEM INITIALIZATION [BEGIN]
  Serial.begin(115200);
  while (!Serial);

  // Separate SPI configurations for LoRa
  SPI.begin(5, 19, 27, 18); //SCK, MISO, MOSI, CS
```

```
  //5, 19, 27, 18
  //18, 19, 23, 5
  Serial.println("LoRa Sender");
  LoRa.setPins(SS_PIN, RST_PIN, DIO0_PIN);

  while (!LoRa.begin(FREQUENCY)) {
    Serial.println("LoRa initialization failed. Retrying...");
    delay(500);
  }

  LoRa.setSyncWord(0xF3);
  SPREADING_FACTOR = constrain(SPREADING_FACTOR, 7, 12); // Limits of
SPREADING_FACTOR
  TX_POWER = constrain(TX_POWER, -3, 15); // Limmits of TX_POWER

  LoRa.setSpreadingFactor(SPREADING_FACTOR);
  LoRa.setTxPower(TX_POWER);
  LoRa.setSignalBandwidth(125E3);           // 500 Hz = 0.0005 MHz Plus étaler
= plus de vitesse de transmission et moins de portée (car plus de bruit).
7.8E3, 10.4E3, 15.6E3, 20.8E3, 31.25E3, 41.7E3, 62.5E3, 125E3, 250E3, and
500E3
  LoRa.setCodingRate4(5);                   // entre 5 et 8 plus la valeurs
est elevée plus les erreurs sont faible (portée augmente) débit est réduit 4/5
4/8
  LoRa.setPreambleLength(8);            //Taille des données anti erreurs plus
le chiffre est grand plus la portée augmente (et la vitesse diminue)
  Serial.println("LoRa Initializing OK!");
  ////////////////////////////////////////////////////

  // Pins setup
  pinMode(LED, OUTPUT);
  pinMode(SIGNAL, OUTPUT);
  pinMode(S_SEQ, INPUT);
  digitalWrite(SIGNAL, LOW);
  pinMode(SW_TELEM, INPUT_PULLUP);
  signal_seq_recu = false;
  apogee_sent = false;

  // Start the serial communication and the I2C bus
  Serial.begin(GPSBaud);
  Wire.begin();

  // Initialize the MPU sensors
  initializeMPU(MPU6050_ADDR);
  initializeMPU(MPU9250_ADDR);
  //initializeMAGNETO(AK8963_ADDRESS); // MAGNETO NE SORT PAS DE VALEURS

  // Initialize the BMP280 sensor
  if (!bmp.begin(BMP_ADDR)) {
```

```
    Serial.println("Could not find a valid BMP280 sensor, check wiring!");
    while (1){blinkLED();};
  }

  // Calibration Pitot
  sensorOffset = adjustOffset(sensorPin, sensorOffset);

  // Initialize SD library with SPI pins
  if (!SD.begin(13,15,2,14)) {            //T1:13,15,2,14  T2: 23,5,19,18 M5 :
4,23,19,18 uint8_t csPin, int8_t mosi, int8_t miso, int8_t sck
    Serial.println("initialization failed!");
    while (1) {
      digitalWrite(LED, HIGH);
      delay(1000);
      digitalWrite(LED, LOW);
      delay(1000);
    }
  }

  SDdata = SD.open("DATA.txt", O_CREAT | O_WRITE | O_APPEND);
  SDdata.println("start :");
  SDdata.close();
  Serial.println("SD Ok");

  delay(3000);
  digitalWrite(LED, HIGH);
  Serial.println("Setup done!");
  t0 = millis();

}

// ================================================ LOOP

void loop() {

  ////////////////////////////////////////////////////
  SPREADING_FACTOR = constrain(SPREADING_FACTOR, 7, 12);
  TX_POWER = constrain(TX_POWER, -3, 15);

  TelemetryData data;
  ////////////////////////////////////////////////////

  int signal_seq = digitalRead(S_SEQ);
  // Read the data from the MPU sensors and the BMP280 sensors
  readMPU9250Data(MPU9250_ADDR);
  readMPU6050Data(MPU6050_ADDR);
  //readMAGNETO(AK8963_ADDRESS);
  sensorValue = analogRead(sensorPin) - sensorOffset; // Calcul de la valeur
renvoyée par le capteur après calibrage
```

```
  updateReadings(sensorValue); // Mise à jour des lectures du capteur
  float smoothedSensorValue = getSmoothedSensorValue(); // Obtention de la
valeur lissée du capteur
  calculateAndPrintValues(smoothedSensorValue); // Calcul et affichage des
valeurs
  last_pressure = corr_pressure;
  readBMPData();

  // Détection de descente
  if ((last_pressure + 0.5) < corr_pressure){
    digitalWrite(SIGNAL, HIGH); // Signal envoyé au séquenceur
    //digitalWrite(LED, HIGH);
    apogee_sent = true;
  } else {
     digitalWrite(SIGNAL, LOW);
     //digitalWrite(LED, LOW);
     apogee_sent = false;
  }

  // Réception signal du séquenceur
  if (signal_seq == HIGH) {
    signal_seq_recu = true;
  } else {
    signal_seq_recu = false;
  }

  // Read Data from the GPS
  readGPS();

  ////////////////////////////////////////////////
  // Serialize data into a string
  String serializedData = String(corr_ax) + "," + String(corr_ay) + "," +
String(corr_az) + ","
                        + String(corr_gx) + "," + String(corr_gy) + "," +
String(corr_gz) + ","
                        + String(lat*1000000) + "," + String(lng*1000000) +
","
                        + String(temperature) + "," + String(corr_pressure);

  // Send data via LoRa
  LoRa.beginPacket();
  LoRa.print(serializedData);
  LoRa.endPacket();
  ////////////////////////////////////////////////

  SDdata = SD.open("DATA.txt", O_WRITE | O_APPEND);;
  if(SDdata) {
    SDdata.print(millis()-t0);
    SDdata.print(',');
```

```
  SDdata.print(signal_seq_recu);
  SDdata.print(',');
  SDdata.print(apogee_sent);
  SDdata.print(',');
  SDdata.print(corr_ax);
  SDdata.print(',');
  SDdata.print(corr_ay);
  SDdata.print(',');
  SDdata.print(corr_az);
  SDdata.print(',');
  SDdata.print(corr_gx);
  SDdata.print(',');
  SDdata.print(corr_gy);
  SDdata.print(',');
  SDdata.print(corr_gz);
  SDdata.print(',');
  SDdata.print(corr_az_16g);
  SDdata.print(',');
  SDdata.print(temperature);
  SDdata.print(',');
  SDdata.print(corr_pressure);
  SDdata.print(',');
  SDdata.print(lat, 10);
  SDdata.print(',');
  SDdata.println(lng, 10);
  SDdata.close();
}


// VARIABLES SD :
// millis()
// corr_ax, corr_ay, corr_az, corr_gx, corr_gy, corr_gz, mx, my, mz;
// corr_az_16g;
// lat, lng
// signal_seq_recu, apogee_sent
// temperature, corr_pressure
// voltage, kpa, speed

// VARIABLES TELEM :
// millis()
// corr_ax, corr_ay, corr_az, corr_gx, corr_gy, corr_gz, mx, my, mz;
// lat, lng
// corr_pressure
// speed

////////////////////////////////////////////////
// Important !
//delay(DELAY_BETWEEN_TRANSMISSIONS);
////////////////////////////////////////////////
```

```
}

// ================================================== FONCTIONS

void initializeMPU(uint8_t address) {
  // Wake up the MPU sensor
  Wire.beginTransmission(address);
  Wire.write(0x6B);  // PWR_MGMT_1 register
  Wire.write(0);     // Set to zero (wakes up the MPU6050)
  Wire.endTransmission(true);

  if (address == MPU9250_ADDR) {
  // Set the gyro configuration register
  Wire.beginTransmission(address);
  Wire.write(0x1B);  // Gyro configuration register
  Wire.write(2 << 3); // Set the range to 1000 degrees/sec: 0b10 (2) shifted
to bits 4 and 3 = 2 << 3 = 00010000
  Wire.endTransmission(true);
  } else {
    // Set the accelerometer configuration register
    Wire.beginTransmission(address);
    Wire.write(0x1C);  // ACCEL_CONFIG register
    Wire.write(3 << 3); // Set the range to ±16g: 0b11 (3) shifted left 3 bits
    Wire.endTransmission(true);
    }
}

/*void initializeMAGNETO(uint8_t address) {
  // Configure the magnetometer (AK8963)
  Wire.beginTransmission(address);
  Wire.write(0x0A); // Control register 1
  Wire.write(0x16); // Set to 16-bit output and 100Hz continuous measurement
mode
  Wire.endTransmission(true);
}*/

void readMPU9250Data(uint8_t address) {
  // Read the raw data from the MPU sensor
  int16_t ax, ay, az, gx, gy, gz;

  Wire.beginTransmission(address);
  Wire.write(0x3B);  // Starting with ACCEL_XOUT_H register
  Wire.endTransmission(false);
  Wire.requestFrom(address, 14, true);  // Request a total of 14 registers

  ax = Wire.read() << 8 | Wire.read();  // ACCEL_XOUT_H and ACCEL_XOUT_L
  ay = Wire.read() << 8 | Wire.read();  // ACCEL_YOUT_H and ACCEL_YOUT_L
  az = Wire.read() << 8 | Wire.read();  // ACCEL_ZOUT_H and ACCEL_ZOUT_L
  Wire.read();  // Skip TEMP_OUT_H
```

```
  Wire.read();  // Skip TEMP_OUT_L
  gx = Wire.read() << 8 | Wire.read();  // GYRO_XOUT_H and GYRO_XOUT_L
  gy = Wire.read() << 8 | Wire.read();  // GYRO_YOUT_H and GYRO_YOUT_L
  gz = Wire.read() << 8 | Wire.read();  // GYRO_ZOUT_H and GYRO_ZOUT_L

  corr_ax = ax / accelScaleFactor;
  corr_ay = az / accelScaleFactor;
  corr_az = ay / accelScaleFactor;
  corr_gx = gx / gyroScaleFactor-2.40;
  corr_gy = gz / gyroScaleFactor+2.40;
  corr_gz = gy / gyroScaleFactor+1.00; // Invert Z and Y axis

  // Print the acceleration and gyro data for the MPU sensor
  /*Serial.print("MPU9250 - Acceleration: ");
  Serial.print("X = "); Serial.print(ax / accelScaleFactor);
  Serial.print(", Y = "); Serial.print(az / accelScaleFactor);
  Serial.print(", Z = "); Serial.println(ay / accelScaleFactor);

  Serial.print("MPU9250 - Gyro: ");
  Serial.print("X = "); Serial.print(gx / gyroScaleFactor-2.40);
  Serial.print(", Y = "); Serial.print(gz / gyroScaleFactor+2.40);
  Serial.print(", Z = "); Serial.println(gy / gyroScaleFactor+1.00);*/
}

void readMPU6050Data(uint8_t address) {
  // Read the raw data from the MPU sensor
  int16_t ay_16g;

  Wire.beginTransmission(address);
  Wire.write(0x3B);  // Starting with ACCEL_XOUT_H register
  Wire.endTransmission(false);
  Wire.requestFrom(address, 4, true);  // Request a total of 4 registers

  Wire.read();  // Skip ax_OUT_H
  Wire.read();  // Skip ax_OUT_L
  ay_16g = Wire.read() << 8 | Wire.read();  // ACCEL_YOUT_H and ACCEL_YOUT_L

  corr_az_16g = ay_16g / 2048.0;

  //Serial.print("MPU6050 - Acceleration: ");
  //Serial.print("Z_16g = "); Serial.println(ay_16g / 2048.0); // 2048.0 scale
factor for 16g

}

/*void readMAGNETO(uint8_t address) {
  // Read magnetometer data
  Wire.beginTransmission(address);
  Wire.write(0x03); // Starting register for magnetometer data
```

```
  Wire.endTransmission(false);
  Wire.requestFrom(address, 7, true); // Request 7 registers

  int16_t mx_raw, my_raw, mz_raw;
  mx_raw = Wire.read() | Wire.read() << 8; // MAG_XOUT_L and MAG_XOUT_H
  my_raw = Wire.read() | Wire.read() << 8; // MAG_YOUT_L and MAG_YOUT_H
  mz_raw = Wire.read() | Wire.read() << 8; // MAG_ZOUT_L and MAG_ZOUT_H

  Wire.read(); // Skip ST2 register

  Serial.print(" | Mag raw: ");
  Serial.print(mx_raw);
  Serial.print(", ");
  Serial.print(my_raw);
  Serial.print(", ");
  Serial.print(mz_raw);
  Serial.println();
}*/

void readBMPData() {
  // Read the temperature and pressure from the BMP280 sensor
  temperature = bmp.readTemperature();
  pressure = bmp.readPressure();  // Pressure in Pa

  // Moyenne glissante
  TotalBufferCirculaireSignal = TotalBufferCirculaireSignal + pressure -
HistoriqueSignal[Plus_ancienSignal];
  HistoriqueSignal[Plus_ancienSignal] = pressure;
  Plus_ancienSignal ++;
  if(Plus_ancienSignal == N) Plus_ancienSignal = 0;
  corr_pressure = (TotalBufferCirculaireSignal/N);

  // Print the temperature and pressure data for the BMP280 sensor
  /*Serial.print("BMP280 - Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C, Pressure: ");
  Serial.print(corr_pressure);
  Serial.println(" Pa");*/
  //Serial.println(corr_pressure);
}

void blinkLED() {
  if(millis() % 150<75){
    digitalWrite(LED, LOW);
  }
  else{
    digitalWrite(LED, HIGH);
  }
}
```

```
// Fonctions Pitot
int adjustOffset(int sensorPin, int sensorOffset) { // Fonction
"adjustOffset", permettant de calibrer le capteur pitot
  Serial.println("Commencement de l'etalonnage"); // Affichage pour le
développement
  while (true) {
    sensorValue = analogRead(sensorPin) - sensorOffset; // Lecture de la
valeur du capteur, ajustée par l'offset
    voltage = sensorValue * (voltageMax / sensorMax); // Convertion de la
valeur analogique en tension
    kpa = ((voltage / kpaRangeTopVoltage) - 0.04) / 0.018; // Convertir la
tension en pression (kPa) selon le datasheet du capteur MPX5050DP :
https://www.nxp.com/docs/en/data-sheet/MPX5050.pdf

    if (kpa > 0.05) { // si la pression est supérieure à 0.05 kPa...
      sensorOffset++; // ...Augmentation de l'offset
    } else if (kpa < -0.05) { // si la pression est inférieure à -0.05 kPa...
      sensorOffset--; // ... Diminution de l'offset
    } else {
      Serial.println("Etalonnage termine"); // Affichage d'un message de fin
d'étalonnage
      break;
    }
    Serial.println("Etalonnage en cours..."); // Affichage d'un message
pendant l'étalonnage
  }
  return sensorOffset; // Retourner la nouvelle valeur d'offset ajustée
}

void updateReadings(int newValue) { // Fonction pour mettre à jour les
lectures du capteur
  total -= readings[currentIndex]; // Soustraction de l'ancienne valeur à la
somme totale
  readings[currentIndex] = newValue; // Mise à jour de la valeur actuelle dans
le tableau
  total += readings[currentIndex]; // Ajout de la nouvelle valeur à la somme
totale
  currentIndex = (currentIndex + 1) % numReadings; // Mise à jour de l'index
actuel en bouclant si nécessaire
}

float getSmoothedSensorValue() { // Fonction pour obtenir la valeur moyenne
lissée
  return total / numReadings; // Calcul de la moyenne des lectures
}

void calculateAndPrintValues(float smoothedSensorValue) { // Fonction pour
calculer et afficher les valeurs
```

```
  voltage = smoothedSensorValue * (voltageMax / sensorMax); // Conversion de
la valeur du capteur en tension
  kpa = ((voltage / kpaRangeTopVoltage) - 0.04) / 0.018; // Conversion de la
tension en pression en kPa

  mach = lastspeed / (20.05 * sqrt(temp)); // Calcul du nombre de Mach
  speed = sqrt((2 * kpa * pow(10,3)) / (rho * (1 + (pow(mach, 2) / 4)))); //
Calcul de la vitesse
  lastspeed = speed; // Mise à jour de la dernière vitesse

  Serial.print("Voltage: "); // Affichage de la tension
  Serial.println(voltage, 3); // Affichage de la tension avec 3 décimales

  Serial.print("Kpa: "); // Affichage de la pression en kPa
  Serial.println(kpa, 1); // Affichage de la pression avec 1 décimale

  Serial.print("vitesse :"); // Affichage de la vitesse
  Serial.println(speed); // Affichage de la vitesse
}

void readGPS(){
  if (Serial.available() > 0) {
    gps.encode(Serial.read());
    if (gps.location.isValid()) {
      lat = gps.location.lat();
      lng = gps.location.lng();

      Serial.print(F("Latitude: "));
      Serial.print(gps.location.lat(), 6); // Latitude in degrees (6 decimal
places)
      Serial.print(F(" Longitude: "));
      Serial.println(gps.location.lng(), 6); // Longitude in degrees (6
decimal places)*/
    }
  }

}
```

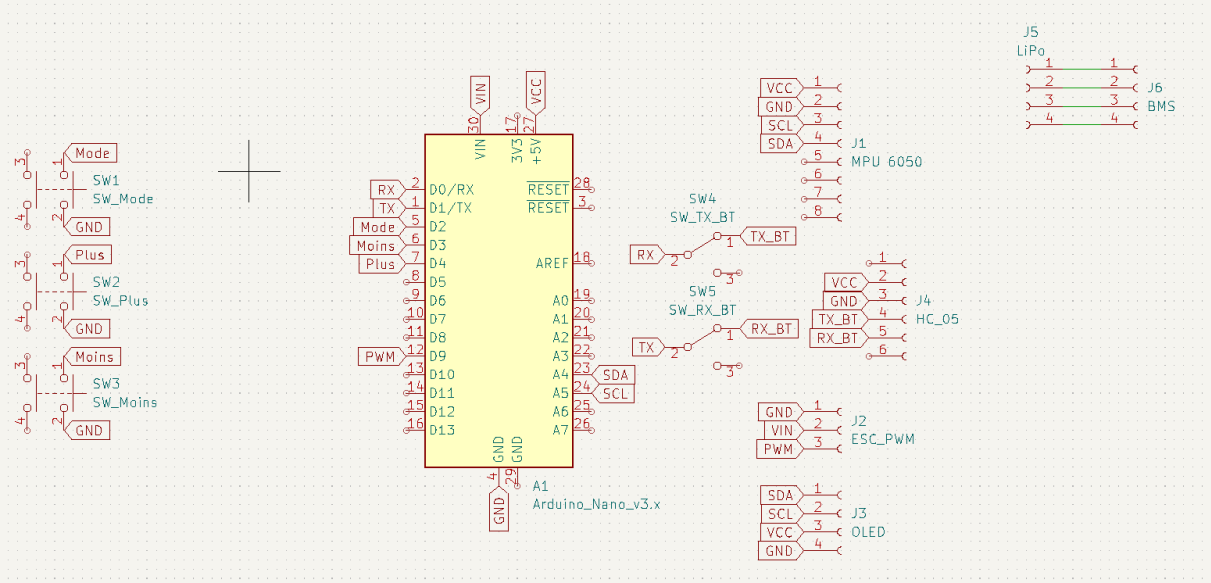


Figure 46 : Schéma électrique du PCB de la caméra stabilisée

Programme du stabilisateur de la caméra :

```
//----------------Cablage-----------------
#define LED 5
//#define ARRET_SW 2
#define PWM 9
#define CS_SD 8

//----------------MPU+I2C-----------------
#include <Wire.h>
#include <MPU6050.h>
MPU6050 mpu;

#define N 5 // Moyenne glissante sur N valeurs
float HistoriqueSignal[N];
float TotalBufferCirculaireSignal=0; // Stockage du total de toutes les
valeurs du buffer.
byte Plus_ancienSignal=0;

//-------------------Roulis--------------------
#include <Servo.h>
Servo Moteur;
bool ARRET = false;
bool DEPART = false;
bool LIFTOFF = false;
bool FLOP_EXPE_RUN = false;

//---------------------SD--------------------
#include "SdFat.h"
SdFat sd;
SdFile data;
struct LogData {
```

```
  unsigned long t;
  byte time_overflow;
  float E1;
  float gyroY;
  float accY;
  int16_t PWM_moteur;
  uint16_t overRun;
};
uint16_t overRun = 0;

//---------------------RTOS--------------------
#include "ChRt.h"

// Shared data, use volatile to insure correct access.

// Mutex for atomic access to data.
MUTEX_DECL(dataMutex);

volatile unsigned long t;
volatile unsigned long t0;
volatile unsigned long t_depart;
byte time_overflow;
volatile float E;
volatile float E1;
volatile float S;
volatile int16_t PWM_moteur;
volatile uint16_t PWM_init = 1500;
// volatile unsigned long timePrev;
// volatile unsigned long elapsedTime;

// Fifo definitions.

// Size of fifo in records.
const size_t FIFO_SIZE = 8;

// Count of data records in fifo.
SEMAPHORE_DECL(fifoData, 0);

// Count of free buffers in fifo.
SEMAPHORE_DECL(fifoSpace, FIFO_SIZE);

// Array of fifo items.
LogData DataFifo[FIFO_SIZE];

#define ExpThreadPeriod 30


//------------------------------------------------------------------------------
--
```

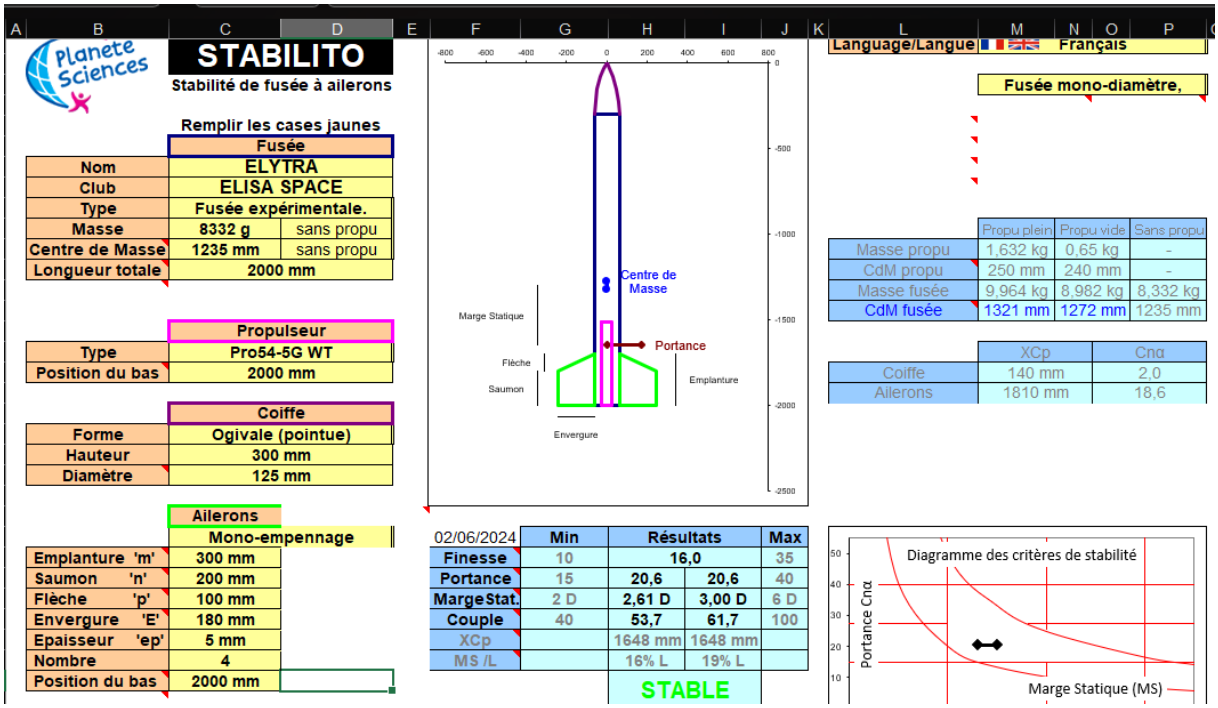
```
// Thread 1, high priority to read sensor.
// 64 byte stack beyond task switch and interrupt needs.
static THD_WORKING_AREA(waThread1, 48);

static THD_FUNCTION(Thread1, arg) {
  (void)arg;

  systime_t wakeTime = chVTGetSystemTime();

  // Index of record to be filled.
  size_t fifoDataHead = 0;

  while (!chThdShouldTerminateX() && ARRET == false) {
    // Sleep until next second.
    wakeTime += TIME_MS2I(ExpThreadPeriod);
    chThdSleepUntil(wakeTime);

    unsigned long t2;
    static unsigned long flop_expe_timer = 0;
    t2 = t;
    t = millis();

    if (t < t2){time_overflow += 1;}

    //-----------------Données-----------------

    float gyroY;
    float gyroYraw;
    float accY;

    gyroYraw = mpu.getRotationY();
    gyroY = gyroYraw / 32.8; // 32.8 LSB/°/s for ±1000 degrees/s range
    accY = mpu.getAccelerationY();
    accY = accY / 2048.0; // 2048.0 LSB/g for ±16g range

    // Moyenne glissante
    TotalBufferCirculaireSignal = TotalBufferCirculaireSignal + gyroY -
HistoriqueSignal[Plus_ancienSignal];
    HistoriqueSignal[Plus_ancienSignal] = gyroY;
    Plus_ancienSignal ++;
    if(Plus_ancienSignal == N) Plus_ancienSignal = 0;
    gyroY = (TotalBufferCirculaireSignal/N);

    //--------------sauvegarde des données-------------------

    if (chSemWaitTimeout(&fifoSpace, TIME_IMMEDIATE) != MSG_OK) overRun++; //
Fifo full, indicate missed point.

    LogData* record = &DataFifo[fifoDataHead];
```

```
    record->time_overflow = time_overflow;
    record->t = t;
    record->E1 = E1;
    record->PWM_moteur = PWM_moteur;
    record->gyroY = gyroY;
    record->accY = accY;
    record->overRun = overRun;

    // Signal new data.
    chSemSignal(&fifoData);

    //-----------Stabilisation Caméra-------------
    if(accY > 2.0 && LIFTOFF == false) {
      flop_expe_timer = millis();
      LIFTOFF = true;
      FLOP_EXPE_RUN = false;
      digitalWrite(LED, HIGH);
    }

    if(LIFTOFF == true && FLOP_EXPE_RUN == false && (millis() -
flop_expe_timer > 2000)){
      FLOP_EXPE_RUN = true;
    }

    //digitalWrite(LED, HIGH);
    static float kp = 0.9;  //1.0128
    static float ki = 0.77;  //0.8067
    PWM_moteur = 1500; //initial value of throttle to the motors

    // elapsedTime = (t - timePrev) / 1000;

    if(FLOP_EXPE_RUN == true) {
      E1 = E;
      E = gyroY;
      S = S + E*kp - E1*kp*ki;
      // S = S + E*kp - E1*(kp*ki + windup*0.0000307);
      PWM_moteur = PWM_init - S;
    }

    if (PWM_moteur < 1000){PWM_moteur = 1000;}
    else if (PWM_moteur > 2000){PWM_moteur = 2000;}
    else if (PWM_moteur >= (PWM_init - 60) && PWM_moteur <= (PWM_init +
60)){PWM_moteur = PWM_init;}

    Moteur.writeMicroseconds(PWM_moteur);
/*
    if (t % 6000 < 3000) {
      digitalWrite(LED, HIGH);
```

```
      PWM_moteur = 1600;
      Moteur.writeMicroseconds(PWM_moteur);
      }
    else {
      digitalWrite(LED, LOW);
      PWM_moteur = 1400;
      Moteur.writeMicroseconds(PWM_moteur);
      }*/

    if(t - flop_expe_timer >= 200000 && FLOP_EXPE_RUN == true){
      ARRET = true;
      digitalWrite(LED, LOW);
      Moteur.writeMicroseconds(1500);
      while(1);
      }
    // timePrev = t;

    // Advance FIFO index.
    fifoDataHead = fifoDataHead < (FIFO_SIZE - 1) ? fifoDataHead + 1 : 0;
  }
}

//------------------------------------------------------------------------SETUP-
----------------------------------------------------------------------

void chSetup() {
  // Start thread
  chThdCreateStatic(waThread1, sizeof(waThread1),
                    NORMALPRIO + 2, Thread1, NULL);
}

void setup() {
  Serial.begin(9600);
  Wire.begin();
  Moteur.attach(PWM);
  pinMode(LED, OUTPUT);
  //pinMode(ARRET_SW, INPUT_PULLUP);

  //-------------------------SD-----------------------

  // Initialize at the highest speed supported by the board that is
  // not over 50 MHz. Try a lower speed if SPI errors occur.
  if (!sd.begin(CS_SD, SD_SCK_MHZ(10))) {
    //Serial.println("Echec connextion SD");
    while (1) {
      digitalWrite(LED, HIGH);
      delay(200);
      digitalWrite(LED, LOW);
      delay(200);
```

```
    }
  }

  //Sauvegarde des données
  data.open("data.txt", O_CREAT | O_WRITE | O_APPEND);
  data.println("Start :");
  data.close();

  //------------------------MPU+I2C--------------------

  byte error;
  Wire.beginTransmission(0x68);
  error = Wire.endTransmission();

  mpu.initialize();
  // Set accelerometer range to ±16g
  mpu.setFullScaleAccelRange(3);
  // Set gyro range to ±1000 degrees/s
  mpu.setFullScaleGyroRange(2);

  // Check if MPU6050 is connected
  // mpu.testConnection()
  if (error == 0) {
      //Serial.println("MPU6050 connection successful");
      digitalWrite(LED, HIGH);
      delay(500);
      digitalWrite(LED, LOW);
  } else {
      //Serial.println("MPU6050 connection failed");
      while (1){
        // halt if connection failed
        digitalWrite(LED, HIGH);
        delay(500);
        digitalWrite(LED, LOW);
        delay(500);
      }
  }

  //-----------------------------------------

  Moteur.writeMicroseconds(PWM_init);
  delay(3000);
  t0 = millis();
  chBegin(chSetup);
}

//--------------------------------------------------------------------LOOP
IDLE----------------------------------------------------------------
```

```
// FIFO index for record to be written.
size_t fifoDataTail = 0;

void loop() {
  chSemWait(&fifoData);

  LogData* ExpData = &DataFifo[fifoDataTail];
  if (fifoDataTail >= FIFO_SIZE) fifoDataTail = 0;

  data.open("data.txt", O_WRITE | O_APPEND);
  data.print(ExpData->t);
  data.print(',');
  data.print(ExpData->time_overflow);
  data.print(',');
  data.print(ExpData->E1);
  data.print(',');
  data.print(ExpData->PWM_moteur);
  data.print(',');
  data.print(ExpData->gyroY);
  data.print(',');
  data.print(ExpData->accY);
  data.print(',');
  data.println(ExpData->overRun);
  data.close();

  // Release record.
  chSemSignal(&fifoSpace);

  // Advance FIFO index.
  fifoDataTail = fifoDataTail < (FIFO_SIZE - 1) ? fifoDataTail + 1 : 0;
}
```



STABILITO

Stabilité de fusée à ailerons

Remplir les cases jaunes

Fusée		
Nom	ELYTRA	
Club	ELISA SPACE	
Type	Fusée expérimentale.	
Masse	8332 g	sans propu
Centre de Masse	1235 mm	sans propu
Longueur totale	2000 mm	

Propulseur	
Type	Pro54-5G WT
Position du bas	2000 mm

Coiffe	
Forme	Ogivale (pointue)
Hauteur	300 mm
Diamètre	125 mm

Ailerons	
	Mono-empennage
Emplanture 'm'	300 mm
Saumon 'n'	200 mm
Flèche 'p'	100 mm
Envergure 'E'	180 mm
Epaisseur 'ep'	5 mm
Nombre	4
Position du bas	2000 mm

02/06/2024	Min	Résultats		Max
Finesse	10	16,0		35
Portance	15	20,6	20,6	40
MargeStat.	2 D	2,61 D	3,00 D	6 D
Couple	40	53,7	61,7	100
XCp		1648 mm	1648 mm	
MS /L		16% L	19% L	
		STABLE		

Language/Langue : Français

Fusée mono-diamètre,

	Propu plein	Propu vide	Sans propu
Masse propu	1,632 kg	0,65 kg	-
CdM propu	250 mm	240 mm	-
Masse fusée	9,964 kg	8,982 kg	8,332 kg
CdM fusée	1321 mm	1272 mm	1235 mm

	XCp	Cnα
Coiffe	140 mm	2,0
Ailerons	1810 mm	18,6

Figure 47 : StabTraj