PROJET
ELISA SPACE

ELISA SPACE : RAPPORT DU PROJET FUSEX FX-02 BELISAMA

VERSION N.1

RÉALISATION D'UNE FUSÉE EXPÉRIMENTALE DANS LE CADRE DU C'SPACE

SOMMAIRE :

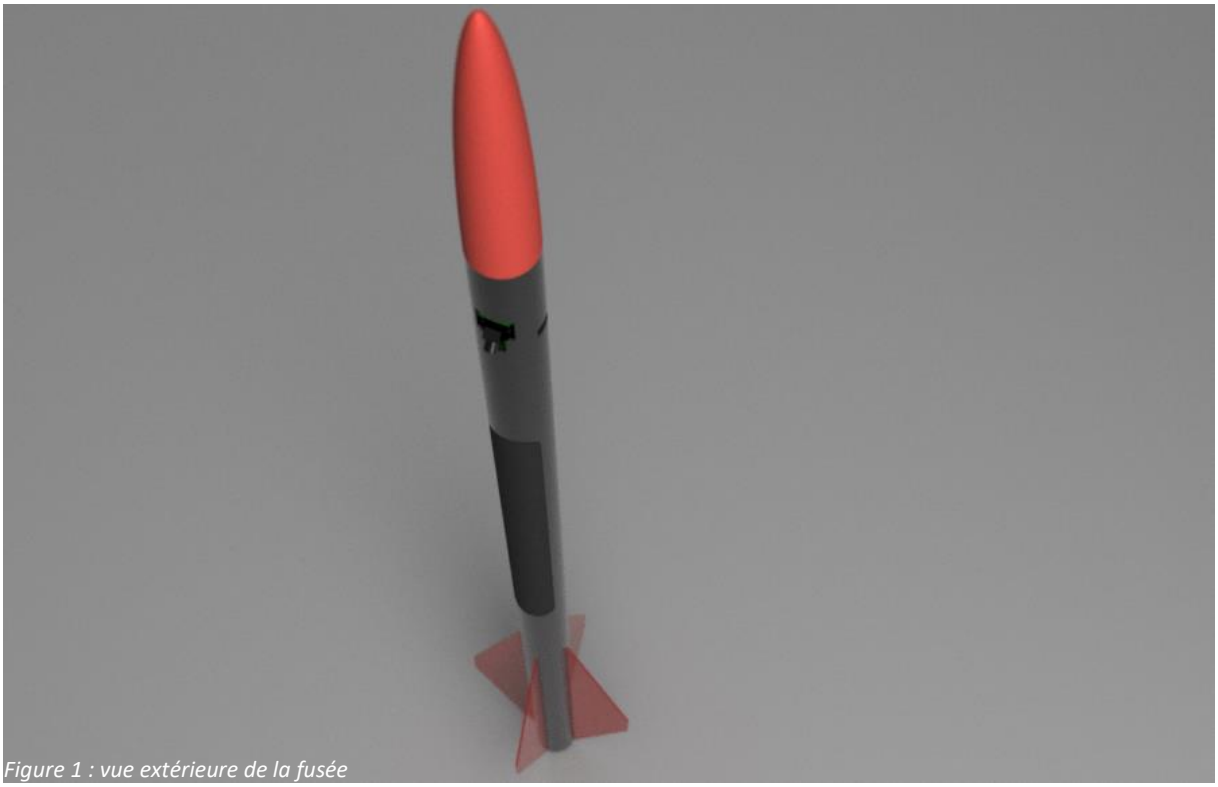


Figure 1 : vue extérieure de la fusée

LA MISSION

BELISAMA

L'équipe d'ELISA Space

Logo de la mission BELISAMA

L'histoire d'Elisa Space a commencé en novembre 2009 avec le rassemblement de plusieurs étudiants passionnés d'aérospatial. Très rapidement, un objectif c'est dessiné : participer à la réalisation d'une fusée expérimentale. Ses membres expérimentés ont participé à la création d'une fusée expérimentale nommée BELENOS. Cette fusée, très impressionnante, résume l'ambition de notre association. Composée de deux étages celle-ci fut utilisée pour tester le nouveau moteur du CNES (Le Centre National d'Etudes Spatiales) : le Pandora.

Dix ans plus tard, deux mini fusées, nommées TARANIS et SUBSONIC, ont été lancées au C'Space 2021. Elles ont toutes deux fait un vol nominal. Ce succès nous a donné l'assurance et les connaissances nécessaires pour nous lancer dans un projet fusée expérimentale !

BELISAMA, le surnom de Bélénos dans la mythologie celtique a une symbolique très forte dans l'histoire de notre association. Ce projet est celui que nous souhaitons faire briller lors de la campagne du C'Space organisée par le CNES et Planète Sciences en juillet 2023.

Voir la vidéo de Belisama sur la chaîne Taranis.

Voir le programme de télémétrie VisUI.

Plan FUSEX

Coiffe – Antennes & caméras

Coiffe fabriquée en PLA avec antennes et trois caméras embarquées pour filmer l'ensemble du vol.

Page 13

Système de récupération

Partie dédiée aux mécanismes pour le déploiement du parachute.

Page 10

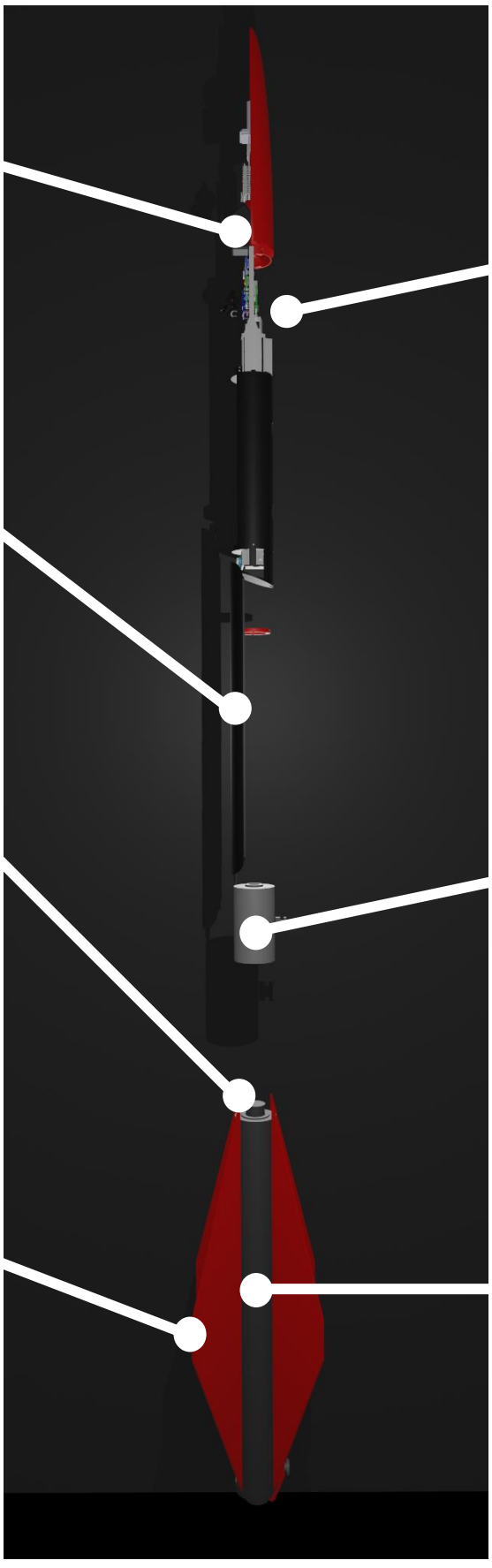
Bague d'ajout de masse

Bague permettant de maintenir le moteur en place et ajouter de la masse à la fusée.

Page 11

Ailerons

Empennage comprenant 4 ailerons profilés soudé directement au corps de la fusée. Soudés par l'entreprise **CTCIA**.



Rack électronique

Séquenceur assurant le déclenchement du système de récupération.

La charge utile permet l'envoi en temps réel des différents paramètres du vol de la fusée. **Page 5**

Système de contrôle du roulis

Masse en rotation rapide permettant le contrôle du roulis de la fusée. Usiné par **PFT Innovatech** .

Page 12

Moteurs

Propulseur à ergol solide (poudre) délivré et installé par les pyrotechniciens du CNES.

Modèle du moteur : Pro 54

Dimensionnement de BELISAMA

BELISAMA a pour but d'être un démonstrateur technologique dédié à l'étude d'un système de mesures et de télémesures tout en permettant la reconstitution du vol à l'aide des données GPS, gyroscopiques, inertielles et atmosphériques (altitude).

La trajectoire de notre lanceur nous permet d'approcher l'altitude maximale fixée par Planète Sciences qui est de **2.3 kilomètres**. Nous essayerons de contrôler l'attitude de la fusée en roulis et d'étudier l'impact que cela peut avoir dans le comportement en vol.

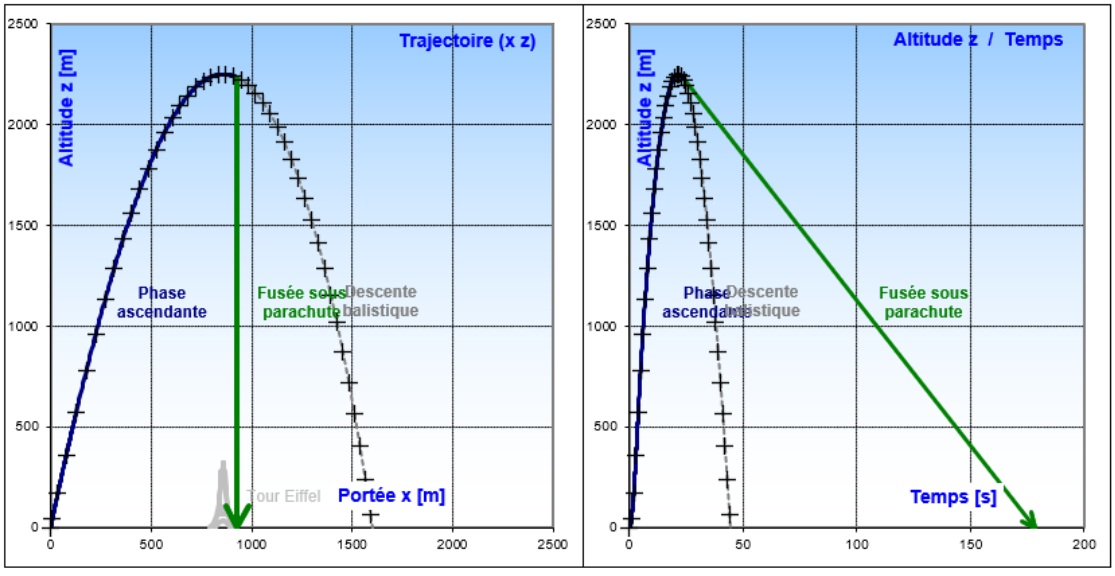
Nous avons dimensionné notre FUSEX à partir de ces critères. Ainsi, nous avons obtenu une fusée mesurant **1900 mm** de hauteur avec un diamètre de **60 mm** (55 mm int) pour une masse totale de **8085g** (avec masse moteur). Notre fusée sera équipée de 4 ailerons profilées en forme libre et mono-empennage. Pour correspondre aux attentes du vol, le moteur que nous souhaiterions utiliser est le **PRO-54-5G Barasinga**.

Dans cette configuration, nous estimons atteindre une apogée de **2360m** avec une vitesse maximale de **227 m/s** et une accélération maximale de **101 m/s².**

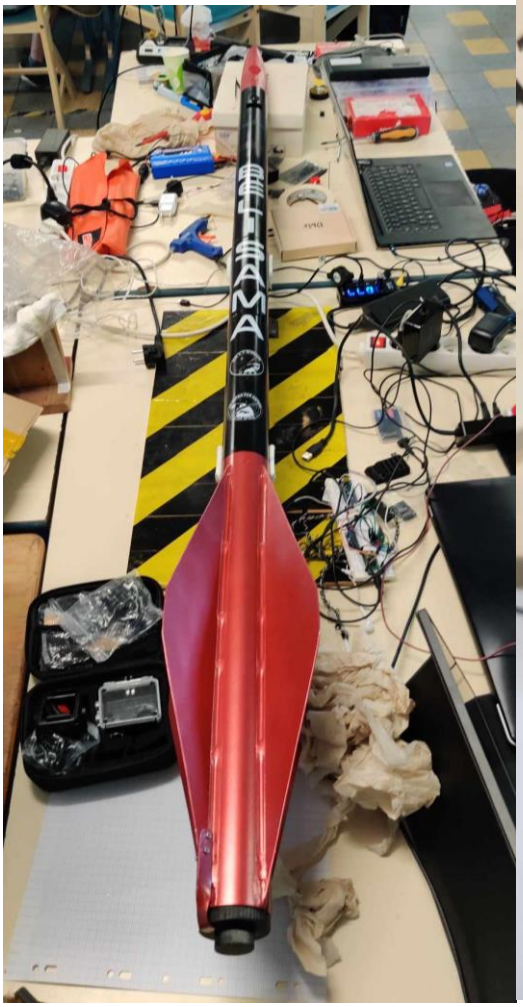
Pour obtenir ces dimensions et les critères de stabilité de la fusée, nous nous sommes basés sur trois logiciels. **Stabtraj** et **OpenRocket** permettant de réaliser des simulations à partir de la méthode de Barrowman et **CATIA** pour le design CAO.

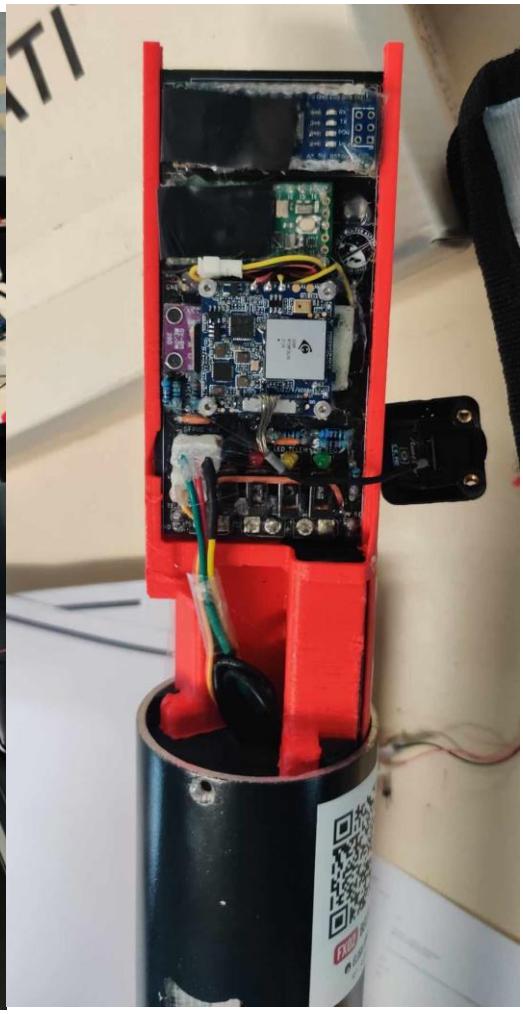
Concernant la disposition mécanique, notre fusée repose sur une structure externe très résistante composée d'un tube en aluminium de 2,5 mm d'épaisseur. La fusée se compose en quatre parties distinctes : un bloc moteur, un bloc télémesure avec roue d'inertie, la partie système de récupération et un rack électronique comprenant le séquenceur et la charge utile.

Les principaux éléments qui composent la fusée sont facilement usinables. Nous utiliserons de nombreuses pièces imprimées en **PLA** provenant d'impression 3D (additive manufacturing). Pour les pièces en métal, nous favorisons l'aluminium pour sa légèreté et sa facilité d'usinage.



01/06/2023	Temps	Altitude z	Portée x	Vitesse	Accélération	Efforts
Sortie de Rampe				27,8 m/s		
Vit max & Acc max				225 m/s	101 m/s²	
Culmination, Apogée	21,6 s	2251 m	857 m	37 m/s		
Ouverture parachute fusée	23,5 s	2233 m	928 m	41 m/s		569,5 N
Impact balistique	44,4 s	~0 m	1602 m	177,6 m/s		111268 J





Définition des expériences

Les expériences du projet Belisama ont été pensée suite au projet mini-fusée Taranis. Lors de ce projet, nous avions effectué une ouverture du parachute avec détection d'apogée à partir de la pression et la température. L'objectif est de retravailler cette expérience pour le projet Belisama en l'améliorant.

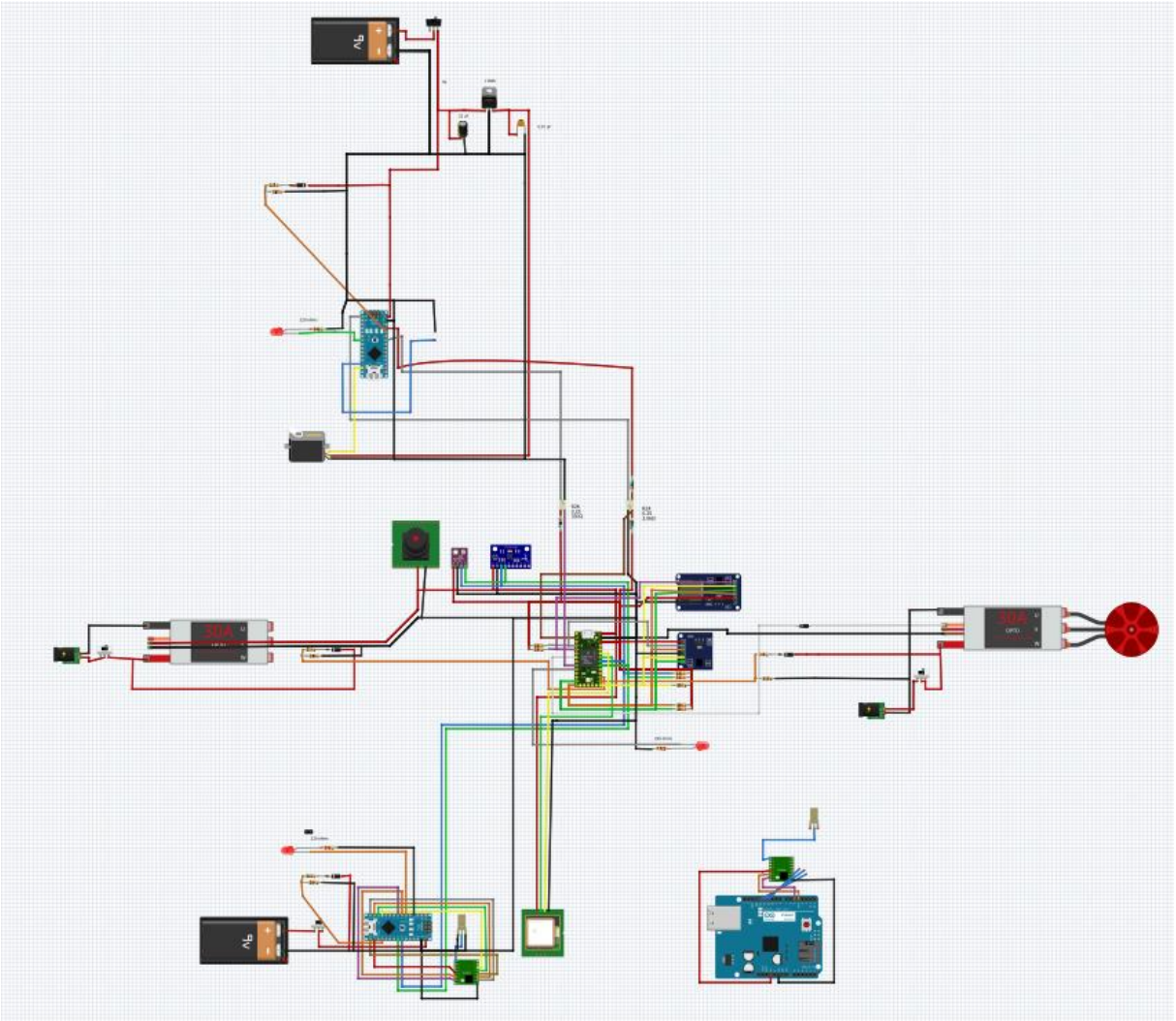
Ensuite, on souhaitait aussi obtenir une trajectographie 3D de la fusée après le vol. Cette expérience avait été tenté sur le projet Taranis mais le bruit des capteurs étant trop important, l'attitude de la fusée n'a pas pu être reproduite. Pour résoudre ce problème, on souhaite ajouter un magnétomètre au gyromètre et accéléromètre déjà pression.

Ces données seront corroborées par 3 caméras situé sur la fusée.

Ensuite, pour améliorer et valider la trajectographie et aider à la récupération de la fusée. Nous avions pour objectif de placer un GPS afin d'obtenir une position précise en 3D.

Afin de connaître cette position pour la récupération de la fusée, une télémesure longue portée a été installé. Les données seront visualisées avec VisUI logiciel développé par un membre pour le projet.

Enfin, à l'origine pensé comme une expérience secondaire, le contrôle de roulis est rapidement devenu une expérience phare pour le projet. Notamment car il permettrait aux caméras de filmer le vol sans trop de mouvement. Pour garantir l'activation du système, l'objectif n'est pas de supprimer le roulis, mais de le maintenir dans une certaine vitesse et direction.



Architecture électronique embarquée

L'architecture de notre système électronique est donc pensée autours de 3 systèmes indépendants :

- Le séquenceur
- L'expérience
- La télémétrie

Le séquenceur a pour but de gérer les étapes critiques de la fusée. Il est actionné à l'aide d'une prise jack et contrôle un servomoteur pour l'ouverture du parachute le tout alimenté avec une pile 9V.

Il permet de transmettre l'état de la fusée via des optocoupleur, afin d'isoler électriquement le système.

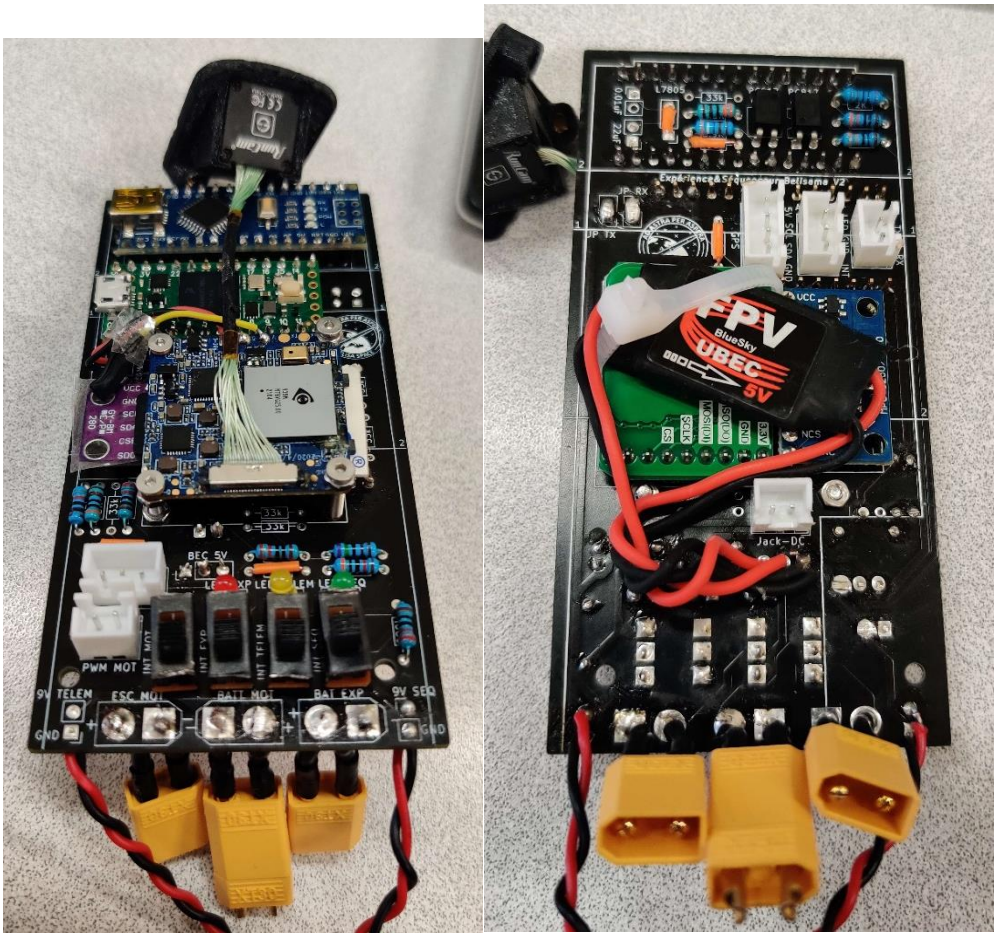
L'optocoupleur transmet l'état de la fusée à l'expérience qui vas alors suivre son programme.

L'expérience commande le système de contrôle de roulis à l'aide d'une liaison **PWM** via un l'**ESC** bidirectionnel. Ce système utilise une 2nd batterie indépendante de celle de l'expérience. Le circuit de l'expérience doit aussi alimenter la caméra 4k.

L'expérience stocke les données sur une carte SD reliée via un module communiquant en SPI.

Elle est reliée aux capteurs de pression, accélération, gyromètre et magnétomètre via une connexion I2C. De plus, elle transmet les données à la télémétrie via cette même connexion I2C.

La télémétrie reçoit et renvoie les informations via une puce lora en **968.5 Mhz**.



L’intégration de l’électronique

L’intégration de notre fusée se décompose en deux parties : l’ordinateur de vol (OBC) carte de télémétrie (CT).

<u>L’ordinateur de vol :</u>

L’ordinateur de vol est le centre de commande de notre fusée. C’est elle qui cadence et analyse les étapes clés lors du vol. Elle a aussi la charge de transmettre chaque événement du vol et de déclencher le système de récupération.

Pour réaliser ces analyses, la carte possède deux microordinateurs indépendants : Le séquenceur et l’expérience. Le séquenceur gère l’état de la fusée et l’ouverture du parachute alors que l’expérience récolte, stocke, transmet et interprète les données durant le vol. L’expérience utilise une **teensy 4.0** avec une horloge interne de **600 Mhz** et est alimenté par une batterie lithium 3S.

La carte séquenceur a sa propre carte d’alimentation avec une pile 9 Volts.

<u>La carte de télémesure :</u>

La carte de télémétrie est placée dans la coiffe, elle est composée de deux parties distinctes :

- **La carte télémesure** a la charge de la transmission des informations à la station sol. Elle réceptionne les informations de l’ordinateur de bord via le BUS I2C et gère la carte Lora pour la transmission à 686.5 Mhz.
- **Le GPS** a la charge d’obtenir la position précise de la fusée pour la récupération et le vol. Le modèle neo-6m est directement relié à l’OBC qui traite les informations et les retransmet à la **carte de télémesure**.

Le séquenceur

Le but du séquenceur est de gérer l'enchaînement de toutes les phases du vol, notamment du point de vue des normes de sécurité imposées par le cahier des charges de Planètes Sciences. Le programme du séquenceur doit prendre en compte et enclencher chacune des différentes étapes du vol.

Il doit compter le temps à partir du décollage et enclencher toutes les phases délimitées par un timer dont la plus crucial qui est l'ouverture du parachute. Pour des raisons de sécurité, à partir de l'instant théorique d'apogée, une fenêtre temporelle va être établie en dehors de laquelle le parachute ne pourra pas s'ouvrir. Pour cela on utilisera une Arduino nano du fait de son format compact et de sa simplicité.

Afin de communiquer à l'expérience de l'état du vol ainsi que de recevoir la commande de détection d'apogée, nous utilisons deux optocoupleurs dans le but d'isoler électriquement l'expérience du séquenceur.

On retrouve ci-dessous toutes les étapes de la fusée : Attente du décollage / Propulsion moteur / Activation volant d'inertie / Plage d'ouverture détection d'apogée / Ouverture parachute par algorithme / Ouverture parachute par temps / Arrêt du volant d'inertie / Arrêt de la télémétrie

Le signal de décollage est engendré en utilisant une prise jack. Le côté femelle est accroché à la fusée et le côté mâle à la rampe de lancement. Quand la fusée décolle, la rupture du contact électrique sera enregistrée par le microcontrôleur qui commenceront alors le timer.

La phase de propulsion consiste à la phase d'ascension de la fusée se vidant de la poudre du moteur. Lorsque celle-ci est vidée, avec une marge temps on active la gestion du volant d'inertie qui va stabiliser la fusée en roulis.

Vient ensuite l'ouverture de la plage de détection de l'apogée, on bloque la vitesse de la masse du volant d'inertie pour ne pas perturber et on active l'algorithme de détection de changement de pression. Si l'algorithme réussi, alors on ouvre le parachute, sinon à la fin de la plage (22.5s estimée par simulation) on ouvre le parachute par timer.

Lors de la phase de descente, on stop complétement le volant d'inertie. Pour cela on prend une marge de temps afin de ne pas perturber le déploiement du parachute. Enfin après atterrissage, on stop la transmission de télémétrie pour ne pas perturber le canal des Fusex suivantes. (**Voir programme en annexe**)

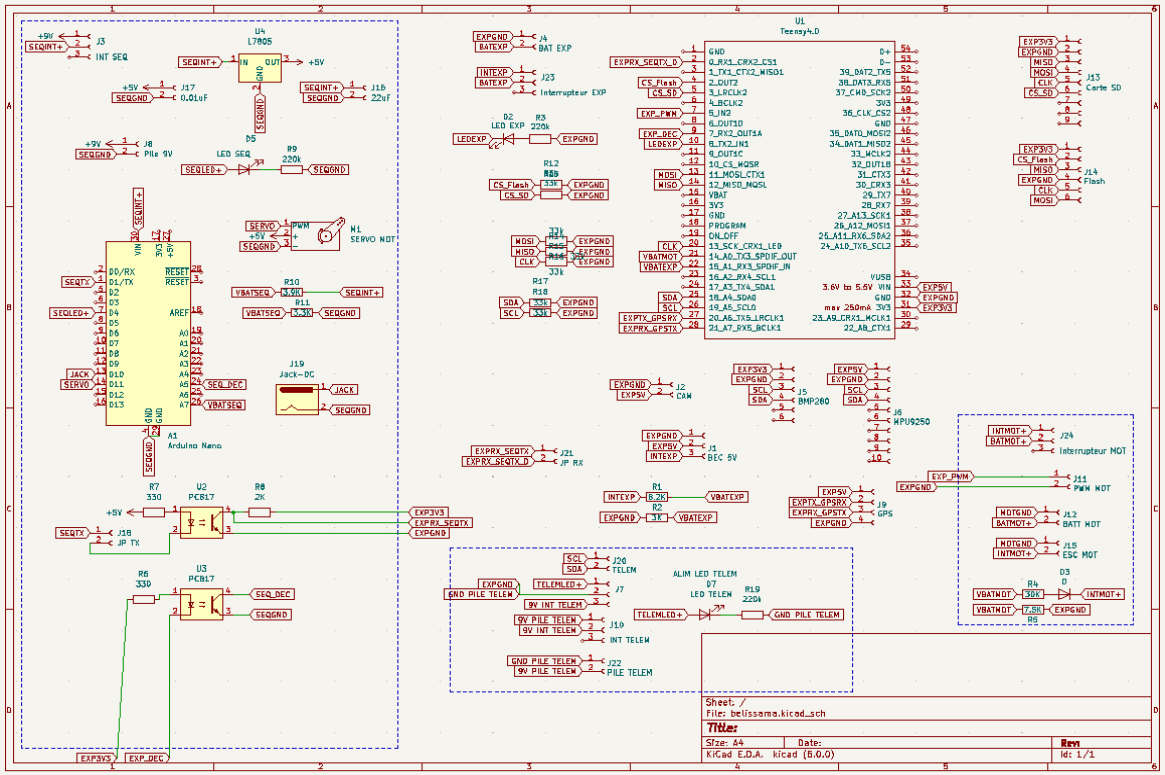


Figure 4 : Schéma de la carte séquenceur / expérience et la séparation physique des circuits

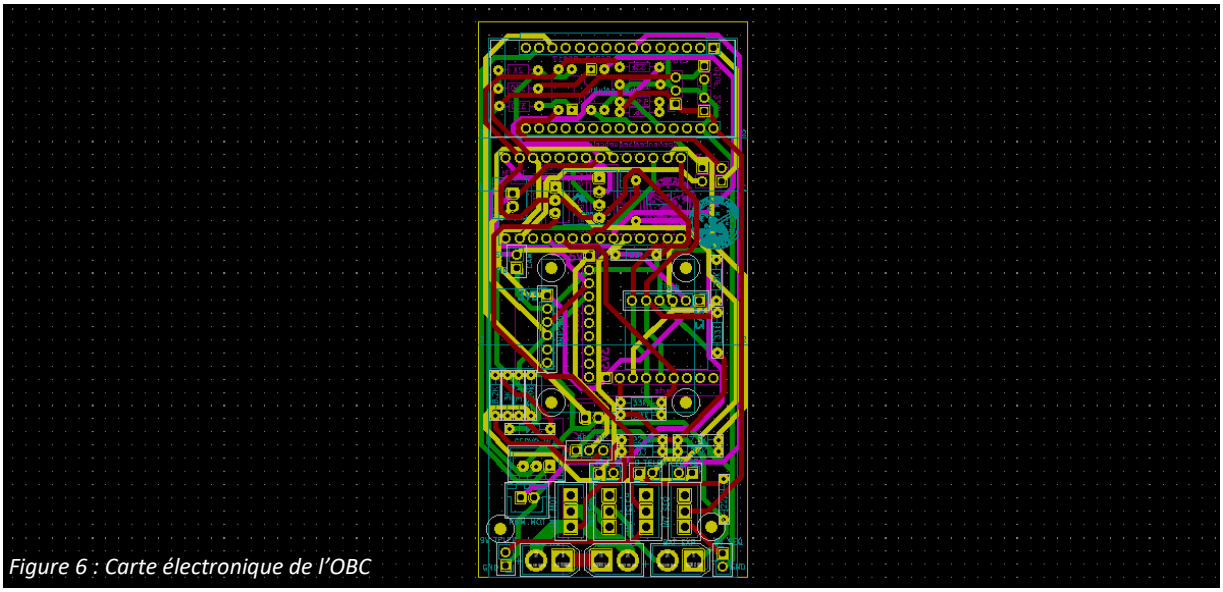


Figure 6 : Carte électronique de l'OBC

L'expérience

Lors de sa mission, la FUSEX récolte des informations grâce à des capteurs. Ces derniers récupèrent les données qui sont traitées puis enregistrées via l'intervention d'un microcontrôleur. Pour mettre en place ce système, nous avons décidé de réaliser une carte dédiée à ces tâches : la carte expérience.

La carte expérience est divisible en 3 parties :

1 : **acquisition** des données des capteurs ;

2 : **le traitement** des données et contrôle des systèmes ;

3 : **stockage et la transmission** des données.

Partie acquisition :

Cette partie est essentielle à la mission. Elle permet de récupérer les données de vol de la fusée afin de les stocker mais aussi de les envoyer en temps réel à la **carte télémétrie**.

Elle est composée de trois capteurs numériques dont deux communiquant sur un bus **I2C** cadencés à 400kHz et une communication **UART**.

Liste des capteurs :

a. **MPU9250** : centrale inertielle et magnétomètre 9 axes ;

b. **BME280** : Baromètre et thermomètre ;

c. **NEO6-M** : GNSS utilisant le GPS, Galileo etc…

Notre carte expérience est équipée d'un module GPS **NEO-6M**. C'est une puce de géolocalisation de la société u-blox nous permettant de localiser la fusée.

Partie traitement :

La partie traitement se fait à l'aide d'un microcontrôleur : l'**ATmega328** (Arduino Nano 3) de Microchip Technology. Il s'agit d'un mini-microcontrôleur 16 pins performant et capable de répondre à nos exigences.

Ce microcontrôleur possède aussi les 3 périphériques de communication qui nous sont nécessaires :

- I2C ;
- SPI ;
- UART/USART.

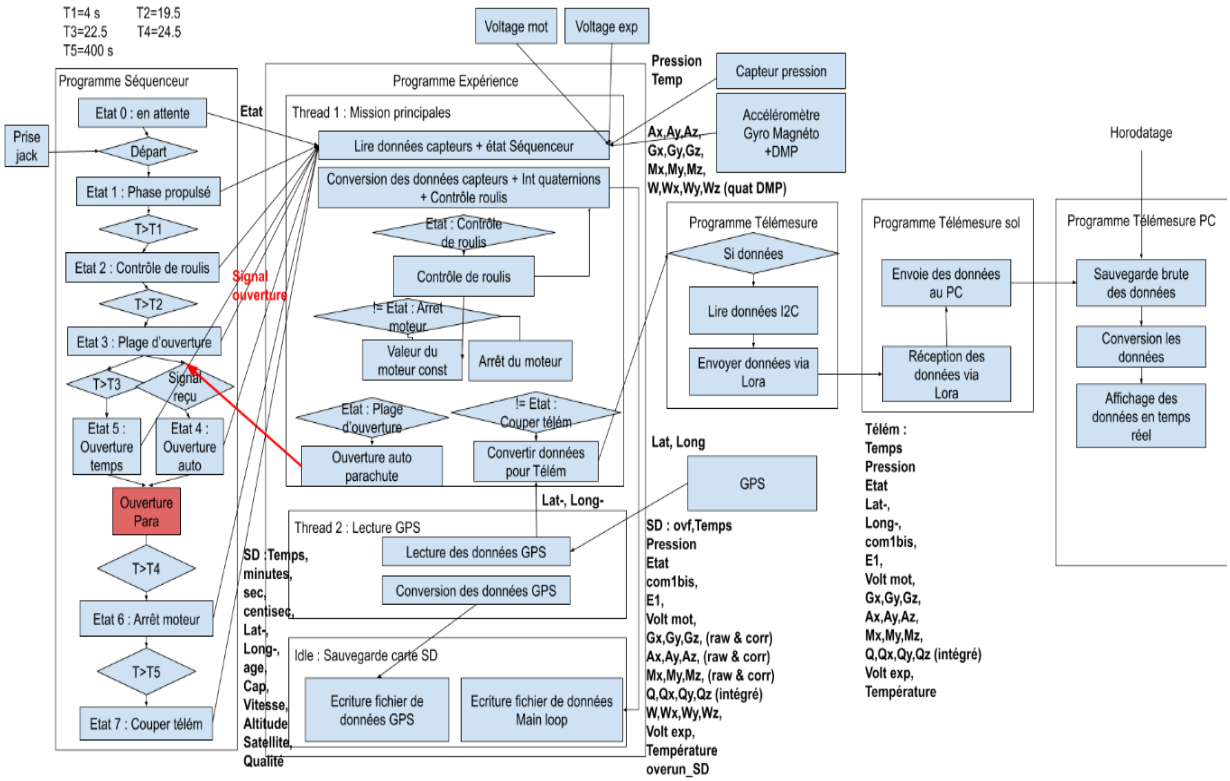
Partie stockage :

Afin d'enregistrer les données tout au long de la mission, deux data logger sont présents sur cette carte :

a. support micro-SD (data logger)

b. mémoire flash W25Q128

Cette double solution a pour avantage d'être plus apte à résister à un impact. En effet la mémoire flash est très résistante aux chocs. Les données



Le programme embarqué

Pour répondre à ces exigences, notre première étape a été de sélectionner un microcontrôleur offrant des performances en termes de vitesse d'exécution et de mémoire supérieures à celles d'un Arduino classique. C'est ainsi que nous avons porté notre choix sur la Teensy 4.0. Cette sélection s'est fondée sur sa fréquence d'horloge à 600MHz, qui nous permettra de gérer l'ensemble de nos opérations dans des délais restreints. De plus, l'espace disponible pour le code et les variables dépasse largement nos besoins.

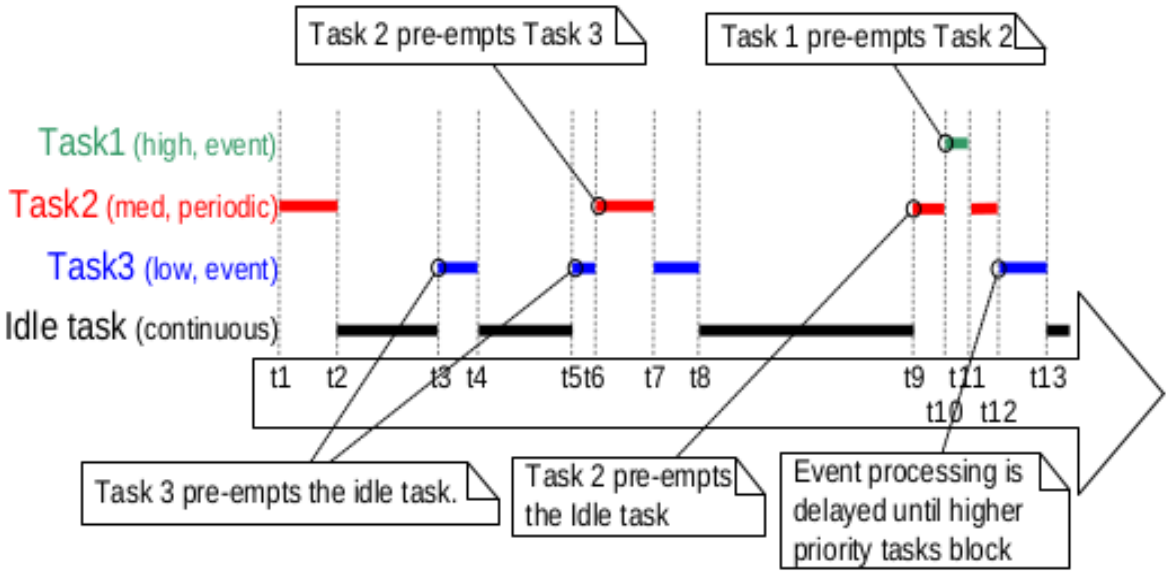
La Teensy 4.0 se distingue également par la présence des trois périphériques de communication indispensables à notre projet : I2C, SPI, UART/USART.

Grâce à ces caractéristiques, la Teensy 4.0 s'impose comme le choix idéal pour notre système, garantissant une exécution rapide et efficace de nos opérations tout en offrant les options de communication nécessaires à la réussite de notre projet.

Toutefois, pour garantir une gestion optimale des opérations entre ces trois composantes et assurer un fonctionnement prévisible du programme, nous avons décidé de mettre en œuvre un système d'exploitation temps réel (RTOS). Ce choix stratégique nous permettra de décomposer le code en plusieurs tâches distinctes, qui seront exécutées à des intervalles réguliers.

Grâce à ces caractéristiques, la Teensy 4.0 s'impose comme le choix idéal pour notre système, garantissant une exécution rapide et efficace de nos opérations tout en offrant les options de communication nécessaires à la réussite de notre projet.

Toutefois, pour garantir une gestion optimale des opérations entre ces trois composantes et assurer un fonctionnement prévisible du programme, nous avons décidé de mettre en œuvre un système d'exploitation temps réel (RTOS). Ce choix stratégique nous permettra de décomposer le code en plusieurs tâches distinctes, qui seront exécutées à des intervalles réguliers.



Le RTOS

L'introduction d'un RTOS offre plusieurs avantages significatifs. Tout d'abord, il favorise une allocation efficace des ressources du microcontrôleur, en permettant l'exécution parallèle de différentes tâches. Cela se traduit par une utilisation optimisée du processeur, évitant les retards inutiles et garantissant des temps de réponse constants pour chaque tâche.

En répartissant les opérations en tâches spécifiques et en les exécutant à intervalles réguliers, le RTOS permet de mieux contrôler la synchronisation entre les différentes parties du programme, simplifiant ainsi le développement et la gestion de l'ensemble du système.

Parmi les RTOS compatibles avec la Teensy 4.0, nous avons choisi d'utiliser ChibiRTOS qui est un RTOS léger tout en implémentant les fonctionnalités de base dont nous avons besoin.

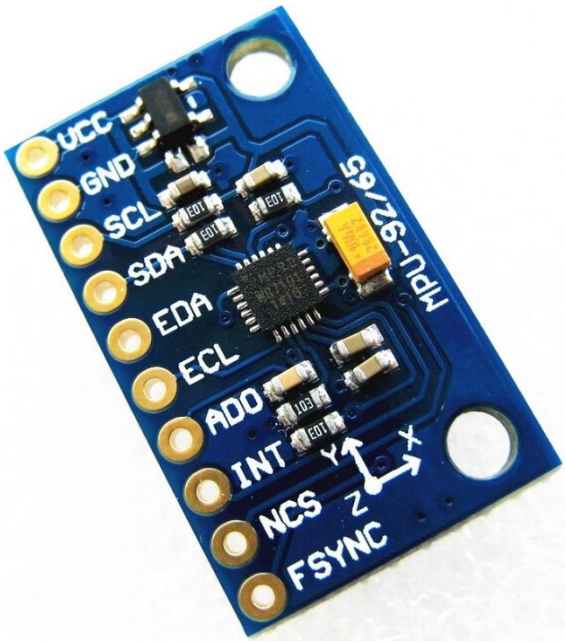
La première étape de notre démarche consiste à partitionner le programme en plusieurs threads distincts, chacun étant attribué à une priorité spécifique. Dans cette optique, nous avons opté pour la création de deux threads distincts, répondant à des objectifs spécifiques.

Le premier thread, ayant la priorité la plus importante, englobe plusieurs fonctions essentielles.

Il gère d'abord l'acquisition initiale des données provenant des capteurs. Ensuite, il prend en charge le traitement de ces données, ainsi que le contrôle du volant d'inertie et la détection de l'apogée. L'attribution de ces tâches au même thread assure une coordination cohérente entre les étapes de la mission, permettant une exécution sans problème.

Le second thread, ayant une priorité intermédiaire, est dédié à la gestion du GPS. Étant donné que le temps d'acquisition des données GPS est relativement lent, avec une fréquence de 500 millisecondes, il est judicieux de le traiter dans un thread séparé. Cette approche garantit que le système peut efficacement collecter et gérer les informations de localisation tout en maintenant une exécution fluide et en ne perturbant pas les autres tâches.

En parallèle, en tant que tâche de fond, nous mettons en œuvre la sauvegarde des données via l'Idle task. Un micro-contrôleur doit toujours avoir quelque chose à exécuter. En d'autres termes, il doit toujours y avoir une tâche en exécution. La plus petite priorité est attribuée à cette tâche. L'enregistrement est donc exécuté en arrière-plan, garantissant que les informations collectées sont enregistrées de manière fiable et continue tout au long de la mission, sans compromettre les autres opérations en cours.



L’acquisition des données

En ce qui concerne la collecte des données, nous avons mis en place une variété de méthodes en fonction du type de données à acquérir et du capteur associé. Voici comment nous procédons :

Gestion du Temps et Overflow : Pour surveiller le temps, nous utilisons la fonction micros(). Cependant, cette fonction renvoie une valeur non signée de 32 bits, ce qui provoque un débordement après 1 heure, 11 minutes et 34 secondes. Pour résoudre ce problème, nous utilisons un compteur d'overflow qui compare le temps actuel avec le dernier enregistré. Cela nous permet de gérer efficacement le temps de manière continue et éviter les erreurs causées par les débordements.

Transfert de Données GPS entre Threads : Afin de traiter les données GPS acquises en parallèle dans un autre thread, nous utilisons un Mutex (verrou mutuel). Le Mutex protège l'accès à ces variables partagées, empêchant toute interruption ou modification non autorisée lorsque les données sont transférées d'un thread à l'autre. Cela garantit l'intégrité des données et évite les conflits potentiels.

Acquisition de l'État de la Fusée : Nous collectons le dernier état de la fusée en lisant le buffer série de la carte de séquençage. Nous lisons la dernière entrée du buffer et le vidons pour nous assurer que la prochaine lecture fournira une valeur à jour. Cette méthode nous permet de récupérer l'état actuel de la fusée de manière précise.

Capteurs MPU9250 et BME280 via I2C : Les capteurs MPU9250 et BME280 sont connectés au bus I2C. Pour simplifier l'acquisition des données à partir de ces capteurs, nous utilisons les bibliothèques <SparkFunMPU9250-DMP.h> et <BMP280_DEV.h>. Ces bibliothèques fournissent une interface conviviale qui nous permet d'obtenir facilement les informations nécessaires sans avoir à décortiquer les datasheets des capteurs. Cela accélère le processus de collecte de données et simplifie la gestion de ces capteurs complexes.

En adoptant ces méthodes diverses et en utilisant des outils comme les Mutex et les bibliothèques, nous optimisons l'acquisition et la gestion des données, tout en simplifiant le processus global pour garantir des résultats précis et fiables.

```
static THD_WORKING_AREA(waThread2, 512);

static THD_FUNCTION(Thread2, arg) {
  (void)arg;

  systime_t wakeTime = chVTGetSystemTime();

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


  while (!chThdShouldTerminateX()) {
    // Sleep until next second.
    wakeTime += TIME_MS2I(GPSThreadPeriod);
    chThdSleepUntil(wakeTime);
```

Le GPS

Le deuxième thread est dédié au système GPS en raison de la latence d'acquisition et du risque de perte de signal pendant la phase de montée de la fusée. Le GPS présente des particularités qui nécessitent une attention spécifique. En effet, il faut du temps à une puce GPS pour acquérir le signal des satellites en orbite. De plus, pour des raisons de sécurité, cette puce ne fournit pas de données lorsqu'elle est soumise à de hautes vitesses, comme c'est le cas pendant la phase de montée rapide de la fusée.

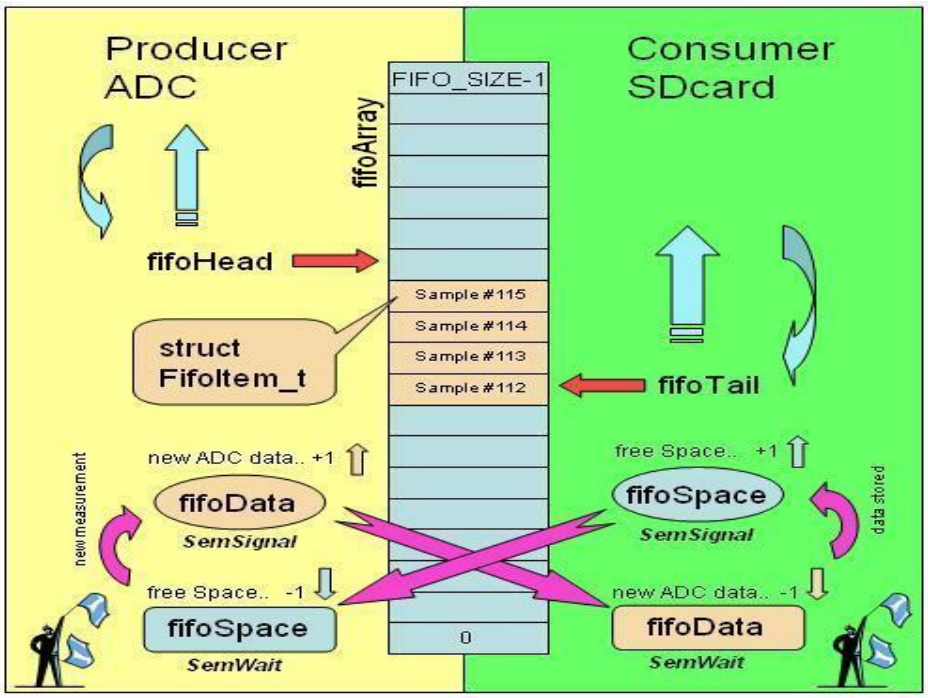
Afin de prendre en compte ces contraintes, nous avons configuré ce deuxième thread avec une période de 500 millisecondes. Cette période a été choisie de manière à ce que le thread GPS puisse fonctionner de manière autonome sans perturber les autres expériences en cours ni interférer avec la sauvegarde de données.

En définissant une période plus longue pour ce thread, nous permettons au GPS de prendre le temps nécessaire pour acquérir le signal des satellites de manière fiable, tout en évitant les problèmes de latence et de perte de signal pendant la phase critique de montée de la fusée. Cette approche garantit que les données GPS seront disponibles et précises, contribuant ainsi à l'obtention de mesures fiables pour le suivi et l'analyse de la trajectoire de la fusée.

Avant de commencer, nous établissons des offsets pour la latitude et la longitude, ce qui facilite le traitement ultérieur des données. Ces décalages ont été déterminés en examinant la position de la rampe de lancement sur Google Maps. Ce processus d'acquisition et de transmission/sauvegarde des données GPS est relativement simple et efficace.

En utilisant la bibliothèque <TinyGPSPlus.h>, nous sommes en mesure de déterminer si le GPS nous a fourni un point de localisation valide. Si tel est le cas, nous entamons alors le processus d'acquisition des autres données GPS. Si aucune donnée valide n'est reçue, nous attendons 500 millisecondes et réessayons ensuite. Cette approche permet de garantir que seules les données fiables et valides sont traitées.

Une fois que nous avons acquis les données GPS, nous les transmettons au processus de sauvegarde que nous aborderons sous peu. Il est important de noter que nous utilisons également un Mutex pour transmettre les valeurs de latitude et de longitude au premier thread. Cette utilisation du Mutex garantit que les données sont partagées de manière sécurisée entre les différents threads, prévenant tout conflit ou problème de synchronisation.



Sauvegarde des données

La phase finale est celle de la sauvegarde des données dans le module micro SD dédié. Cette étape s'est avérée complexe en raison des problèmes liés à l'enregistrement haute fréquence sur une carte SD.

Des sauts irréguliers entre les intervalles d'enregistrement ont été observés en raison de la latence inhérente à la carte SD. Cette irrégularité a été problématique avec l'utilisation d'un RTOS. Bien que le RTOS assure l'exécution précise des opérations, il exige que ces opérations se terminent à temps. Cependant, la latence variable de la carte SD, pouvant parfois atteindre plusieurs centaines de millisecondes, menaçait la stabilité du RTOS, provoquant des crashs et l'arrêt du programme.

Pour résoudre ce problème, nous avons créé une stratégie impliquant un buffer FIFO circulaire et l'enregistrement des données dans l'Idle Task du RTOS. Le principe est illustré ci-dessus.

Ce buffer FIFO circulaire permet de stocker temporairement les données avant de les transférer à la carte SD en blocs. La gestion de l'écriture sur la carte SD est gérée en arrière-plan par l'Idle Task du RTOS.

Tout d'abord, une structure "LogData" est conçue pour contenir les données, avec un tableau "DataFifo" utilisé comme buffer de stockage. Un index "fifoDataHead" suit où les nouvelles acquisitions sont stockées dans le buffer. Pour gérer efficacement ce buffer FIFO circulaire entre le thread principal et la tâche de fond, deux sémaphores sont utilisés : "fifoSpace" pour indiquer les emplacements disponibles dans le buffer avant que de nouvelles données n'écrasent les anciennes, et "fifoData" pour informer la tâche de fond du nombre de données dans le buffer.

Une fois les données dans le buffer circulaire, un index "fifoDataTail" suit l'emplacement de l'acquisition à enregistrer sur la carte SD. Le sémaphore est également utilisé pour interroger le buffer et déterminer s'il y a des données à enregistrer. Si c'est le cas, la tâche de fond écrit ces données sur la carte SD, puis libère de l'espace dans le buffer via le sémaphore.

Le même processus s'applique aux données GPS, gérées par un buffer distinct et enregistrées dans un fichier distinct sur la carte SD pour éviter les interférences entre les deux threads.

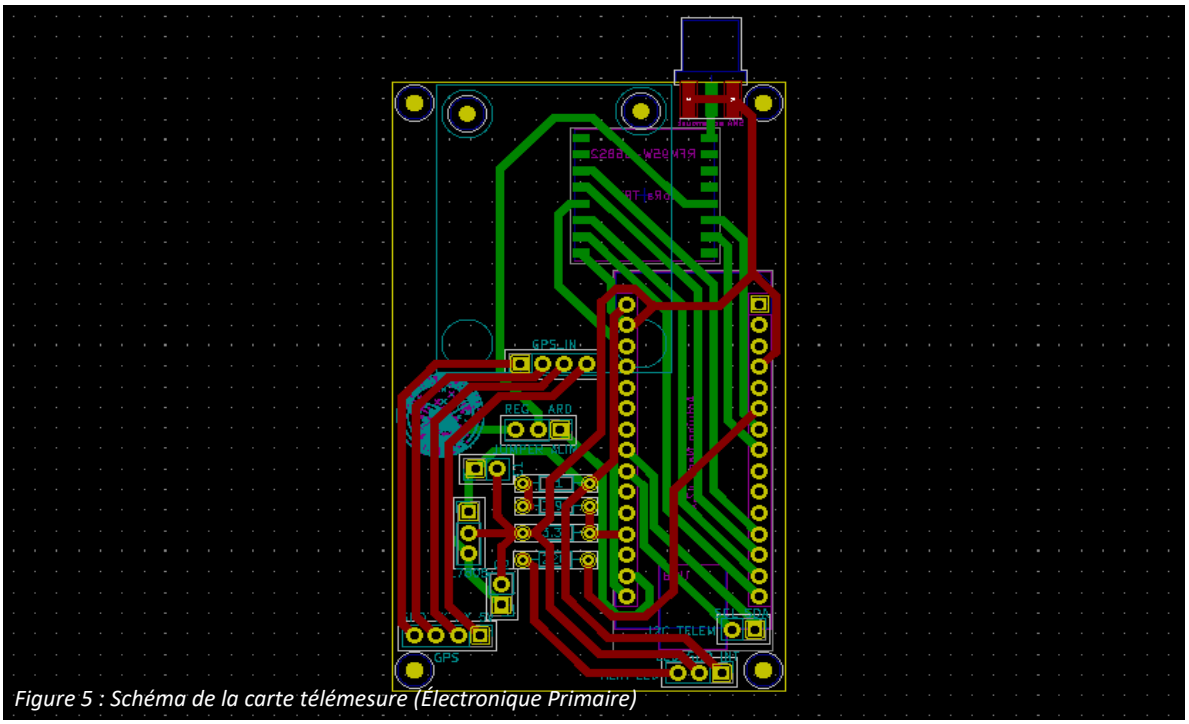


Figure 5 : Schéma de la carte télémesure (Électronique Primaire)

La carte de Télémesure

Contrairement à notre précédent système de télémétrie utilisé sur le projet "TARANIS", où nous transmettions un flux de données sous forme de chaîne de caractères avec des séparateurs, nous avons adopté un nouveau format de trame pour la télémétrie sur le projet "Belissama". Dans ce nouveau format, les données sont transmises sous forme de séquences de bytes, organisées en paquets. Chaque donnée est positionnée à un emplacement prédéterminé dans la trame, ce qui nous permet de récupérer sa valeur en consultant l'index approprié de bytes lors de la réception.

Cette approche présente plusieurs avantages, notamment en termes d'économie d'espace. En effet, ce procédé nous permet de transmettre beaucoup moins de bytes par rapport à la méthode précédente. Pour réaliser cela, nous devons stocker les bytes correspondant aux valeurs qui nous intéressent. Pour ce faire, nous avons mis en œuvre différentes fonctions en fonction du type de données que nous traitons. Un exemple important est l'utilisation d'un format de "float half-precision" (demi-précision) qui n'est pas pris en charge nativement sur les plateformes Arduino.

	type	bit de début	bit de fin
temps	uint32	8	40
pression	half float	40	56
etat	byte	56	64
latitude	half float	64	80
longitude	half float	80	96
com1bis	half float	96	112
E1	half float	112	128
volt moteur	uint16	128	144
gx	half float	144	160
gy	half float	160	176
gz	half float	176	192
ax	half float	192	208
ay	half float	208	224
az	half float	224	240
mx	half float	240	256
my	half float	256	272
mz	half float	272	288
qw	half float	288	304
qx	half float	304	320
qy	half float	320	336
qz	half float	336	352
volt exp	uint16	352	368
température	half float	368	384
RSSI	float	384	400

Figure 12. Liste des variables transmises

Notons que notre utilisation actuelle de la trame inclut uniquement les données brutes, facilitée par le protocole intégré de la puce LoRa qui encapsule les informations pour une communication à sens unique. Dans un contexte plus complexe, un en-tête enrichi pourrait améliorer la gestion de la communication. Cet en-tête potentiel inclurait : un signal "Start of Frame" pour le début de la trame, un identifiant unique pour différencier les types de données, un indicateur de taille pour préciser la longueur des données. De plus un code de redondance cyclique (CRC) après les données afin d'assurer l'intégrité post-transmission et un signal "End of Frame" pour marquer la fin de chaque ensemble de données pourrait être ajoutés.

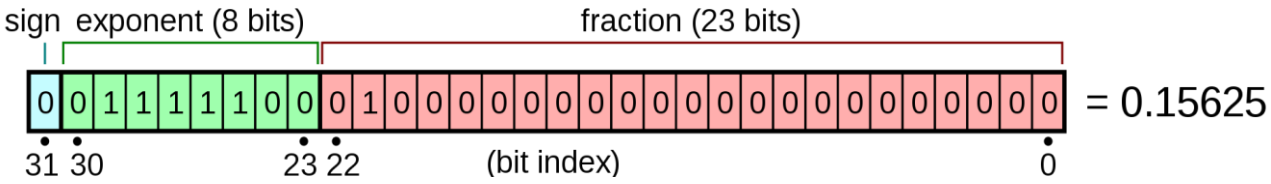


Figure 13. Représentation des nombres flottants selon la norme IEEE 754

La transmission des données

Pour faciliter la compression des données, nous avons implémenté plusieurs fonctions spécifiques. Le concept fondamental de ces fonctions est de récupérer les octets de chaque variable et de les stocker dans un tableau d'octets. Ce tableau d'octets est ensuite transmis à la carte de télémétrie, puis transmis via la technologie LoRa à la station sol.

Une caractéristique importante de notre méthode est que nous stockons les octets en respectant la norme d'encodage "Big-Endian". Cette convention implique que les valeurs sont enregistrées avec le byte le plus significatif en premier. En d'autres termes, le bit le plus élevé d'une valeur est stocké en premier, suivi du bit suivant et ainsi de suite. Cette norme est largement utilisée dans les systèmes informatiques et de communication.

En utilisant cette approche "Big-Endian", nous garantissons une cohérence et une interprétation correcte des données, tant du côté de l'émetteur que du récepteur. Cela permet également de simplifier le processus de récupération des données et de s'assurer que les valeurs sont correctement interprétées, quelle que soit la plateforme ou le système de réception utilisé.

Ainsi nous procédons à la conversion en byte de variable tels les unsigned int32, unsigned/signed int16 et unsigned int8.

En ce qui concerne les float, la fonction que nous avons développée offre une option permettant de choisir entre deux approches de compression : la première consiste à compacter les 4 octets qui composent le float tel quel, tandis que la seconde consiste à le convertir en "float half-precision".

Si nous choisissons de compacter le float sans conversion, l'approche est similaire à celle que nous avons utilisée pour les nombres entiers non signés de 32 bits, comme décrit précédemment. Cependant, si nous optons pour la conversion en "float half-precision", il est essentiel de comprendre comment un ordinateur stocke et interprète les valeurs de type float avant de pouvoir les réduire.

Un **float (voir si-dessus)** est composé de trois parties distinctes : une fraction qui représente un nombre décimal entre 0 et 1, une partie exponentielle qui permet d'ajuster la magnitude de la fraction, et enfin un signe pour indiquer la positivité ou la négativité de la valeur. En combinant ces trois parties comme illustré dans la formule ci-dessous, nous obtenons la valeur float :

$$(-1)^S \times 2^{E-127} \times 1.F$$

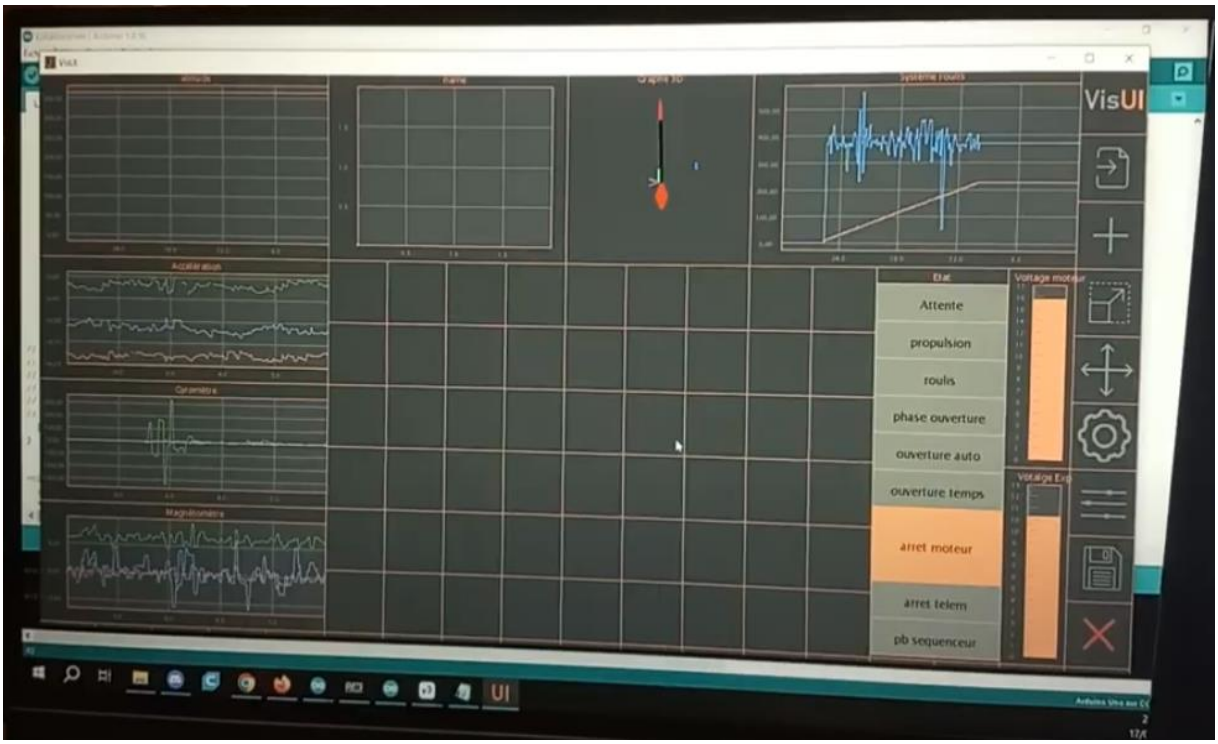
(Cette partie de vidéo explique ce qu'est la norme IEEE 754 qui régit les float 32bits https://youtu.be/p8u_k2LIZyo?t=258)

Il est important de noter que la partie fractionnaire est généralement la plus longue, ce qui implique que la conversion en "float half-precision" consiste à réduire la précision de cette partie à 10 bits et celle de la partie exponentielle à 5 bits. Cette réduction de précision permet de diviser par deux l'espace nécessaire pour stocker le float, tout en sacrifiant une partie de sa précision. Cependant, dans notre contexte, la précision réduite reste plus que suffisante, car les half float ont une précision au maximum de 1/1024, ce qui est tout à fait acceptable pour nos besoins.

Station sol

Après la transmission, la trame est reçue au sol par une carte Arduino qui est connectée à une puce LoRa configurée en mode réception. Une fois la trame reçue, les données sont initialement traitées par l'Arduino. Ensuite, l'Arduino envoie les données traitées via l'interface série (Serial) à un ordinateur hôte. Cette communication série peut être établie via un câble USB, connectant ainsi l'Arduino à l'ordinateur exécutant le logiciel de visualisation. Les données sont transmises à l'ordinateur sous forme de flux de caractères ou de bytes, en fonction de la manière dont elles ont été préparées sur l'Arduino.

Le code de l'Arduino utilise la bibliothèque <LoRa.h> pour mettre en attente la réception d'un paquet LoRa. Lorsqu'un paquet est intercepté, le code extrait les données qu'il contient. On ajoute ensuite des octets de début de trame ("start of frame") et de fin de trame ("end of frame") à ces données. Une fois que les octets de début et de fin de trame sont ajoutés, le paquet de données ainsi modifié est envoyé par l'intermédiaire du bus série USB, où il est prêt à être interprété par **l'interface graphique VisUI**.



Logiciel VisUI

L'application VisUI a été développée en Java avec l'IDE Processing dans le but de visualiser en temps réel les données de la télémétrie provenant d'une carte Arduino. Elle est modulaire, permettant d'ajouter des objets graphiques appelés "widgets" à l'écran. Ces widgets comprennent des graphiques, jauges, zones de texte, caméras et boussoles, tous personnalisables. Les configurations peuvent être sauvegardées et chargées pour éviter de reconfigurer l'interface à chaque utilisation.

Pendant le vol, l'application a bien fonctionné et s'est avérée utile pour vérifier des données telles que la réponse du système de roulis. Cette expérience a validé son utilité pour le projet, et suggère une utilité pour d'autres projets.

Dans cette section, l'attention est portée sur le fichier "LoadData", assurant la réception et le formatage des données provenant de l'Arduino. Il est important de noter que l'interface graphique complète est décrite en détail dans le github dédié : https://github.com/Sirkmix/VisUI

La réception des données dans "VisUI" utilise un thread distinct de celui qui gère l'affichage graphique.

Ce thread surveille le buffer série (Serial USB) de l'ordinateur afin de détecter toute nouvelle donnée entrante. Il vérifie les octets de fin de trame pour déterminer s'ils correspondent à la séquence définie comme l'"end of frame" dans le code Arduino.

Si une correspondance est trouvée avec les octets de fin de trame, le thread vérifie également le premier octet pour déterminer s'il correspond à notre marqueur de début de trame (start of frame), préalablement défini. Si toutefois l'un de ces éléments (les octets de début ou de fin de trame) est manquant, le thread enregistre les données lues en attente de la prochaine lecture. Cette procédure de stockage temporaire vise à garantir que seules les trames valides sont interprétées par le logiciel. Une fonction permet aussi de sauvegarder dans un fichier texte daté toute entrée de trame valide dans le logiciel.

Lorsque les widgets ont besoin d'utiliser les nouvelles données, ils invoquent la fonction "LoadData". Elle examine la dernière trame disponible, repère les bits demandés par le widget et effectue une conversion inverse par rapport aux opérations de conversion effectuées dans la fusée.

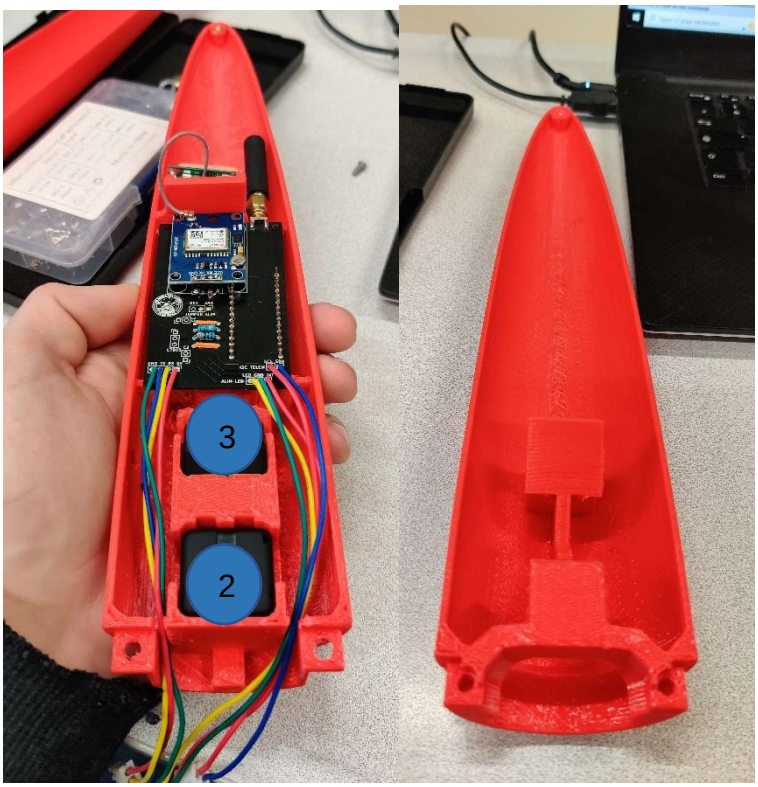


Figure 11 : Vue de la coiffe

La coiffe

La coiffe protège les antennes et la carte télémesure en permettant la transmission et réception des ondes radios. Deux des caméras utilisées pendant le vol y sont placées. Après le vol, les caméras embarquées nous permettront de vérifier le bon fonctionnement des différentes étapes du vol.

Deux caméras seront disposées dans la coiffe de la fusée (Images du dessus) :

- une **caméra 4K** (**1**) ;
- deux **caméra HD** (**2 et 3**).

Une caméra 4K :

La caméra 4K est une caméra **Runcam Split 4**. Il s'agit d'une caméra de drone qui filmera l'environnement extérieur et permettra de déduire le comportement de la fusée, ainsi que l'ouverture de la trappe avec la sortie du parachute.

À l'aide de son objectif fish-eye, elle aura un champ de vision large qui nous permettra d'en déduire l'orientation de la fusée.

Les deux caméra HD :

La caméra HD est une caméra Sq-11. Il s'agit d'une caméra de surveillance utilisée pour la mission TARANIS. Une d'elle sera située au bas de la coiffe et filmera l'horizon durant le vol.

L'autre sera située plus haut dans la coiffe et filmera le ciel, puis le parachute ouvert flottant au-dessus de la fusée.

Le stockage des images :

Les images récoltées seront stockées dans deux **cartes SD**. Celles-ci seront situées dans une « **boîte-noire** ». Nous utiliserons des **rallonges micro-SD** pour placer les enregistrements dans une boîte en PLA au centre de la partie électronique.

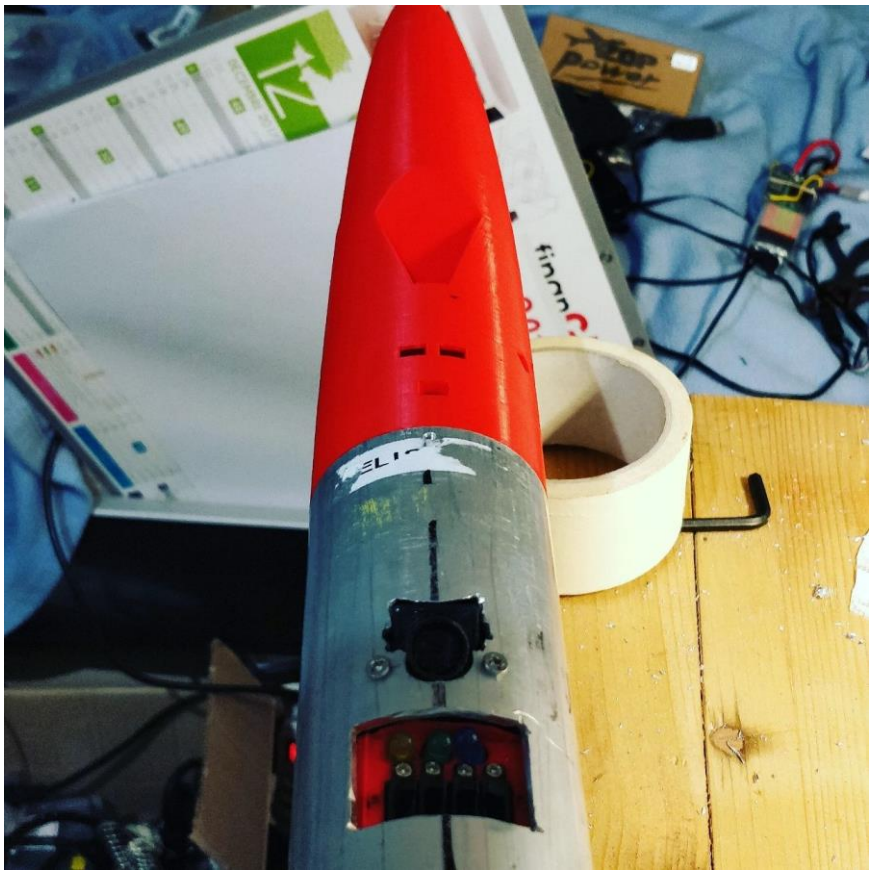


Figure 11 : Vue de la coiffe

Les caméras embarquées

Les caméras embarquées sont très utiles pour récolter de images impressionnantes durant le vol. Elles peuvent notamment servir comme puissant outil de communication pour une association.

Dans un second temps, les caméras situées à bord permettent de suivre les mouvements de la fusée. Etant fixé dans le référentiel de celle-ci, elles permettent d'estimer la rotation de la fusée de manière bien plus précise qu'une caméra au sol.

Ces deux utilisations des caméras ont des critères contradictoires. En effet, dans le cas de la communication, on cherchera à obtenir des caméras avec une très bonne qualité d'image. Alors que, dans le cas de l'analyse du mouvement, on cherchera à avoir un maximum d'image pas secondes capturé possibles au détriment de la qualité d'image.

Notre fusée est équipée de deux caméras HD à 30 FPS dans la coiffe, leur qualité n'est pas excellente. Mais ces caméras ont fait leur preuve sur la fusée Taranis.

Elle est ensuite équipée d'une caméra de drone **Runcam SPLIT 4** permettant de filmer en 4k à 30 FPS. Mais nous avons fait le choix de diminuer la qualité vidéo pour obtenir 60 FPS avec une qualité vidéo de 2k. Ce choix a été motivé pour garantir l'obtention d'image dans le cas où l'expérience de contrôle de roulis ne fonctionnerais pas et aggraverais le roulis de la fusée.

Ces 3 caméras sont disposées du côté opposé à la rampe affine de contrebalancer leur trainée par la présence des patins. Elles permettraient aussi de scanner le ciel au fur et à mesure que la fusée tourne sur elle-même durant son vol (de manière contrôler ou non).

Déploiement du parachute

Une fois l'apogée atteinte, la trappe de notre FuseX doit être éjectée pour permettre le déploiement du parachute. Pour cela, Bélisama est dotée d'un système double action :

- Serrage et verrouillage de la trappe plaquée contre le point d'étanchéité en composite.
- Verrouillage du ressors de catapultage de la trappe.

Légende du système (vue de la case parachute) :

1 : emplacement du parachute plié.

2 : trappe d'éjection (éjectée par le système).

3 : pignon de serrage de la trappe d'éjection.

4 : goupilles parachute.

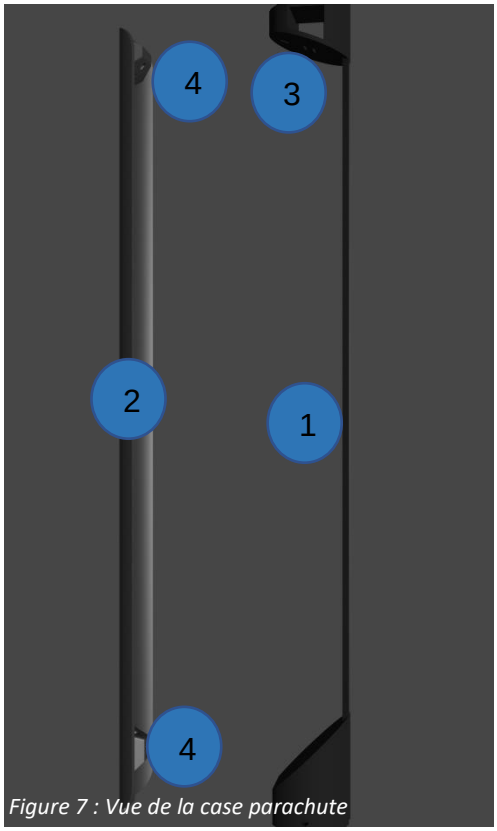


Figure 7 : Vue de la case parachute

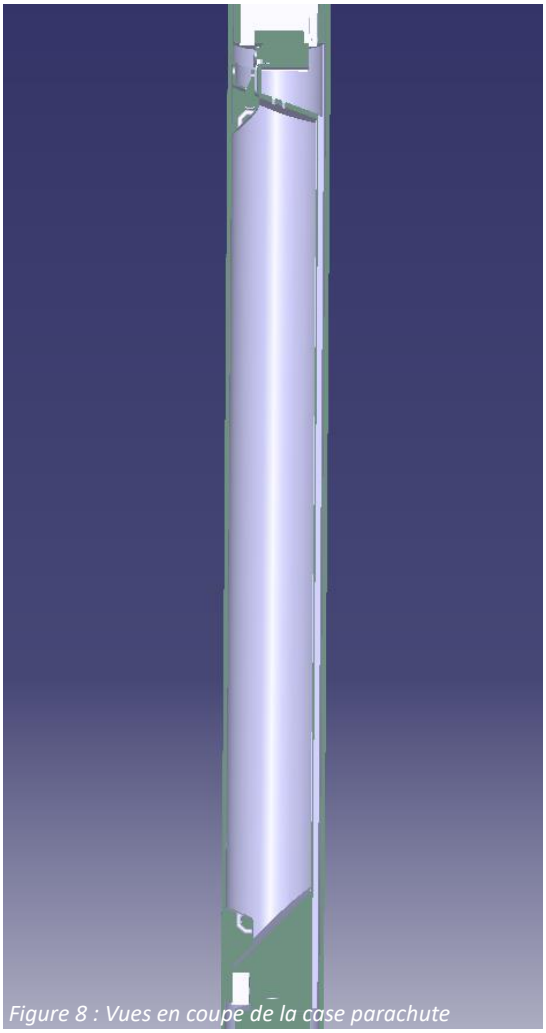


Figure 8 : Vues en coupe de la case parachute

<u>Fonctionnement :</u>

Pour armer le système, il suffit de placer le parachute au sein de son emplacement puis de positionner la trappe. Le servomoteur verrouille donc le système en serrant la trappe à l'aide d'une goupille. Le ressort d'éjection va alors se verrouiller.

Pour déclencher le système, le servomoteur tourne et libère la goupille. La trappe est alors littéralement catapultée à l'extérieur sous l'impulsion du ressort. La trappe, fixée à la moitié de la sangle du parachute est poussée par le flux d'air, elle extrait alors le parachute de son logement.

De cette façon la trappe et le parachute ne se croisent jamais et le risque de mise en torche est réduit.

Le système est capable de déployer le parachute en moins de trois secondes avec une grande fiabilité. Ce système est également très simple à utiliser puisqu'il suffit d'alimenter le servomoteur. Il permet d'avoir un suivi complet des différentes actions avec le retour numérique de position du servomoteur.

La première version a été testée sur le projet TARANIS, elle a permis de valider le design du système qui est maintenant à la phase d'optimisation.

Le parachute

Le parachute est un aspect extrêmement important du projet. C'est ce dispositif qui permet à la fusée de revenir en bon état aux membres qui l'on construit.

Ainsi, la résistance de celui-ci ne doit pas être pris à la légère. Nous avons donc fait le choix de prendre un parachute en croix composé de 2 rectangles en tissus cousue entre eux. Les suspentes du parachute sont intégrées dans les bords de coutures pour répartir la charge à l'ouverture et empêcher la propagation d'un déchirement.

Les suspentes sont composées des cordelettes d'escalades, elles sont recousues aux extrémités du parachute pour empêcher à celui-ci de glisser.

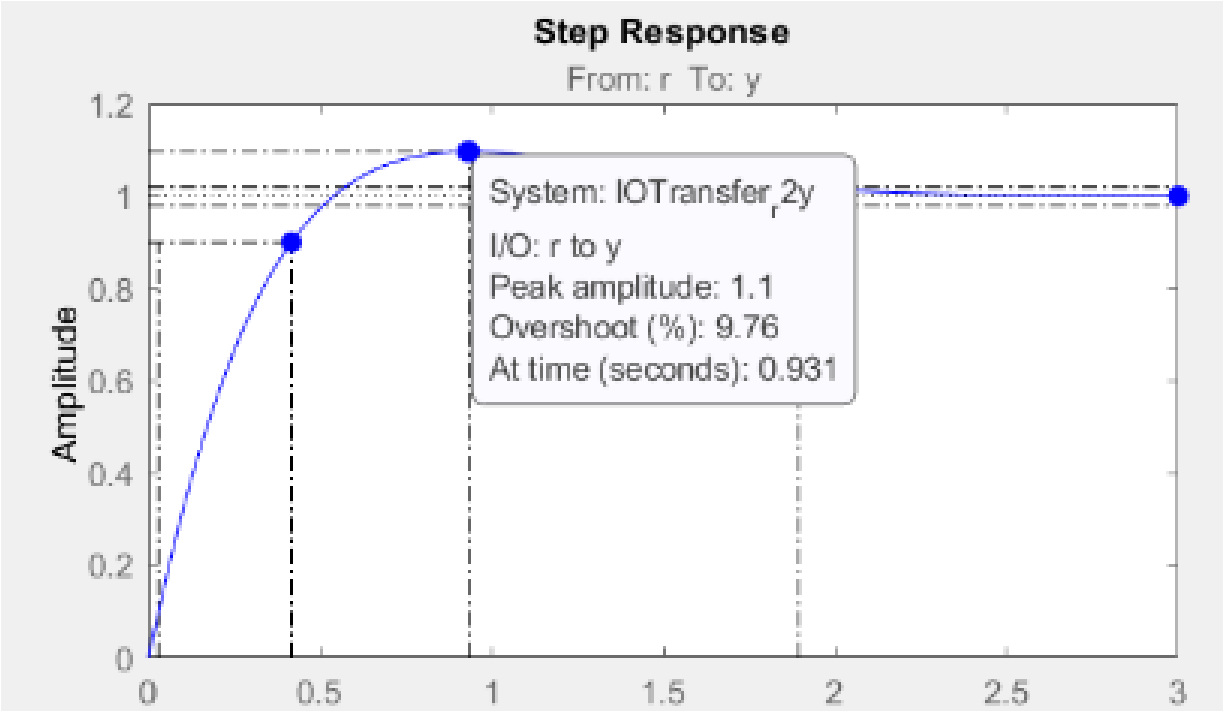
Les suspentes sont passé dans une bague anti-torche imprimée en 3D puis elles sont regroupées au niveau de l'émerillons à l'aide de nœuds.

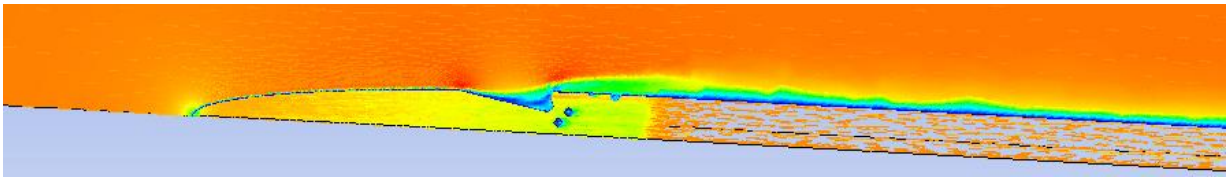
Une fois à la bonne taille, les extrémités des cordelettes sont brulé pour empêcher à celle-ci de glisser dans les nœuds et les déloger.

L'émérillons est ensuite reliée à l'aide d'une sangle cousue sur elle-même à la fusée. La sangle fait environ 3m soit 1.5 fois la fusée. Au milieu de la sangle, un tendeur pour vélo est cousu en parallèle de manière à reprendre une partie des efforts avant de se briser (la sangle en parallèle reprend les efforts).

Enfin, la sangle est reliée à un maillon rapide qui est reliée à un anneau renforcé dans le corps de la fusée.

La trappe est reliée en haut du parachute à l'aide de 2 fils, leur longueur permet à la trappe de bien prendre le vent afin d'éjecter franchement cette-ci.





Simulations

Pour éclaircir plusieurs points au cours de ce projet, nous avons eu recours à plusieurs méthodes de simulations.

Modélisation et simulation du système de contrôle du roulis

Le correcteur du système de roulis est basé sur un modèle créé à l'aide du modèle réel placé sur un stand de test et soumis à un échelon.

A partir de ce modèle, nous utilisons l'outil Sisotool pour créer un correcteur PID adapté à notre système réel et analyse ces performances.

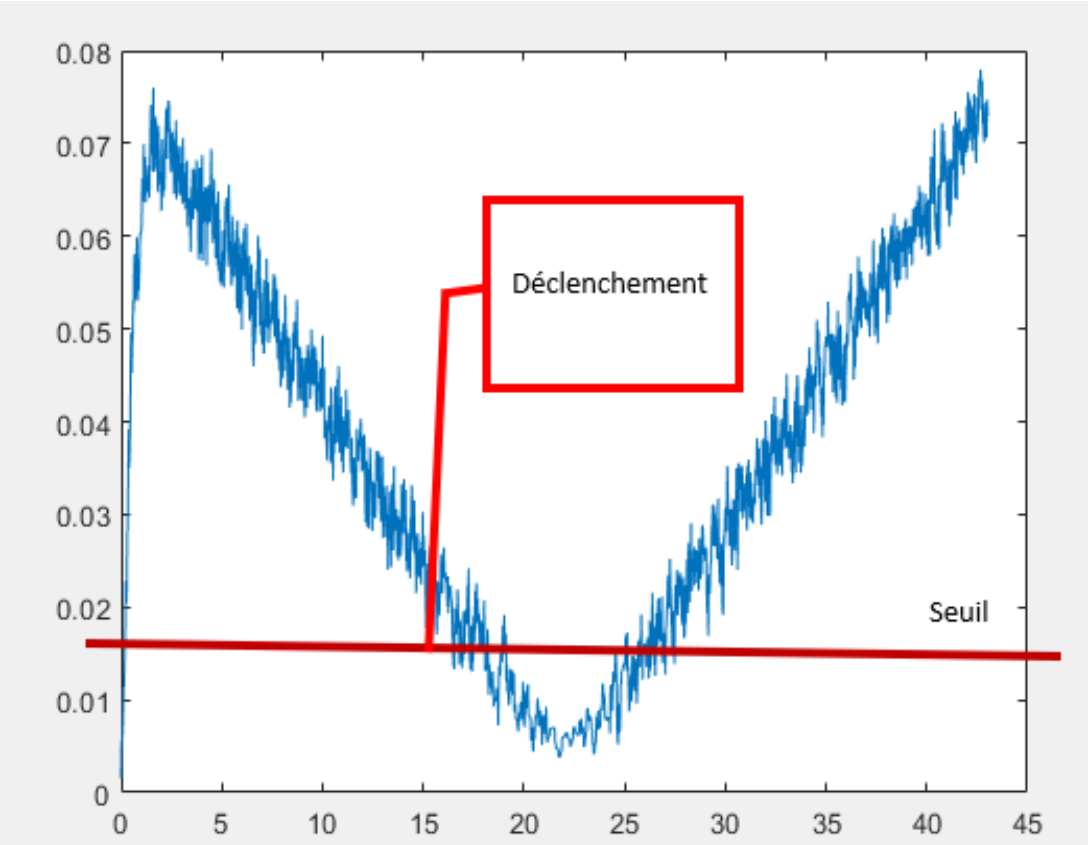
Ce système est ensuite converti dans l'OBC et testé sur le stand de test avant le vol.

Simulation du comportement gyroscopique de la fusée :

Les simulations MATLAB simplifié (sans frottement) montre que la fusée sera plus stable avec la masse de rotation car les perturbations effectuées sur une axe seront reportée sur l'axe à 90°. Cette stabilisation ne peut être supérieure à l'impulsion d'origine (conservation de l'énergie) donc la fusée restera stable dans toute circonstance

Simulation d'écoulement avec l'ouverture de la caméra :

A l'aide du logiciel de simulation **Fluent** de la suite **Ansys** nous avons effectué des simulations de moment générer par les trous des caméras située sur la coiffe. Ces simulations ont montré que l'écoulement est faiblement impacté et la différence de pression local est très faible.



Détection de l’apogée

La dernière étape thread expérience concerne l'algorithme de déclenchement de l'apogée. Pour cette phase, nous avons repris le concept central que nous avions utilisé avec succès dans notre mini-fusée précédente nommée "TARANIS". Cet algorithme se base sur la détection de l'apogée de la trajectoire en analysant la différence de pression par rapport à un seuil préalablement défini par simulation.

Le principe est le suivant : lorsque la fusée atteint l'apogée de sa trajectoire, la pression atmosphérique diminue de manière significative. Cette variation de pression est captée par notre capteur. Si cette diminution de pression dépasse un seuil prédéfini, cela indique que la fusée a atteint son apogée.

En détectant l'apogée de cette manière, nous pouvons déclencher le déploiement du parachute. Cette approche s'appuie sur des données mesurables et permet une réaction précise au moment où la fusée atteint son point le plus haut dans la trajectoire de vol.

Sur le projet "Belissama", nous avons choisi d'améliorer davantage ce concept en introduisant une amélioration significative : le lissage de la courbe de pression grâce à un calcul pondéré en temps réel. Cette approche nous permet d'éliminer les perturbations et les erreurs potentielles provenant du capteur de pression.

L'idée derrière ce lissage de courbe est de prendre en compte non seulement les valeurs brutes de la pression, mais également de les pondérer en fonction du temps. Plutôt que de réagir brusquement à chaque fluctuation de la pression, nous calculons une moyenne pondérée des valeurs de pression sur une période donnée. Cette moyenne pondérée est plus stable et moins susceptible d'être influencée par les bruits et les erreurs du capteur.

Figure 10 : Vue du volant d'inertie (usiner par PFT Innovaltech)

Volant d'inertie

Un volant d'inertie est un dispositif utilisé pour contrôler l'orientation et la stabilité de la fusée expérimentale. Lorsque la fusée se met à tourner autour de l'axe de roulis, le volant d'inertie produit un moment par accélération de sa masse dans direction opposée au roulis de la fusée. En conséquence, les effets d'inertie générée par le volant aident à contrer le mouvement de roulis non désiré.

Pour gérer le moteur dans toutes les phases de vol de la fusée, nous avons établi une distinction entre les différents modes de fonctionnement prévus. Au cours de la phase d'attente et d'ascension, le moteur reste désactivé et la commande est maintenue à zéro. Lorsque nous atteignons la phase de contrôle du roulis, nous relâchons la commande, permettant ainsi au moteur de s'aligner sur la rotation de la fusée et sur la consigne de vitesse de rotation souhaitée

C'est à ce stade que le correcteur entre en jeu, ajustant activement la commande pour assurer la stabilité souhaitée.

La phase de déclenchement du parachute représente une étape délicate. Durant cette période, nous évitons de manipuler la vitesse de rotation du moteur pour préserver la stabilité de la fusée pendant le déploiement du parachute, minimisant ainsi tout risque de perturbation.

Enfin, une fois que nous sommes certains que le parachute est ouvert en fonction de la durée ou de l'apogée, nous réduisons progressivement la vitesse de rotation du moteur jusqu'à l'arrêt complet.

Contrôle de roulis

Pour développer un modèle réaliste, nous avons mené un test d'échelon visant à étudier la réaction du processus face à un changement brusque. Ce test s'est déroulé dans des conditions spécifiques.

Pour mener à bien ce test, la fusée a été suspendue par sa coiffe à l'aide d'une corde liée à deux roulements à billes. Cette configuration a permis de laisser le mouvement de rotation libre autour de l'axe de roulis. Ensuite, nous avons mis en place un programme qui a commandé le moteur en deux phases distinctes. Dans la première phase, le moteur a été sollicité pour atteindre rapidement le milieu de sa puissance. Quelques secondes plus tard, lors de la deuxième phase, le moteur a été enclenché pour atteindre la limite maximale de sa vitesse. Ces deux phases successives ont généré deux réponses indicielles distinctes.

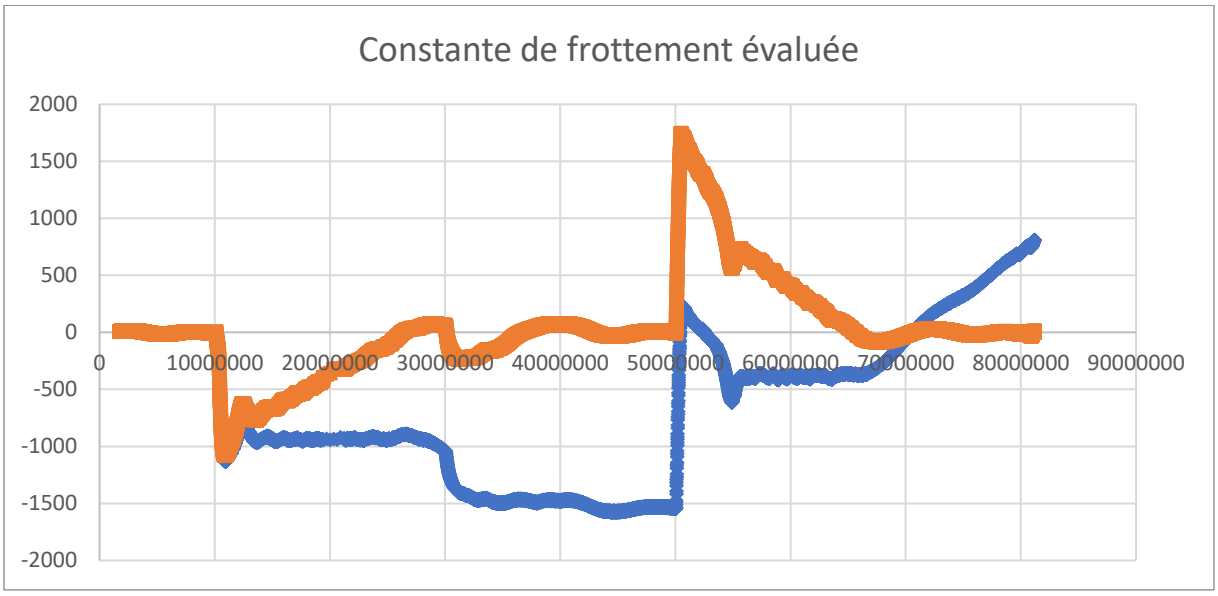
Le premier échelon a été générée alors que la fusée était immobile, suspendue dans cette configuration particulière. Le deuxième échelon a été générée alors que la fusée était en rotation autour de l'axe de roulis. En enregistrant les réponses du processus lors de ces deux échelons, nous avons obtenu des données pour construire un modèle réaliste du comportement de la fusée.

Les deux échelons induits par le moteur sont clairement observables. Néanmoins, deux phénomènes supplémentaires sont également apparents : les frottements des roulements à bille et la torsion de la corde sur elle-même.

Premièrement, les frottements induisent une perturbation dans les mesures, ce qui se traduit par une réduction du pic de roulis enregistré. La torsion agit comme une force de contrecoup, influençant la réponse de la fusée au mouvement de roulis. En conséquence, les valeurs mesurées peuvent ne pas refléter pleinement l'amplitude réelle du roulis généré par les échelons du moteur.

Deuxièmement, lorsque la fusée atteint son point d'équilibre en matière de roulis, les frottements freinent peu à peu le roulis désormais continu. Cette interaction contribue à expliquer pourquoi les mesures de roulis reviennent à zéro des deux échelons successifs.

Enfin la torsion de la corde induit plusieurs mini oscillations en période de stabilisation.



Création du modèle

Les phénomènes ajoutent de la complexité à l'analyse des données du test. Pour des résultats précis, il faut tenir compte de ces facteurs.

La pente entre le début supposé de la stabilisation de l'échelon et le retour du roulis à zéro est évaluée pour calculer une rampe de compensation. Après correction, les résultats deviennent cohérents et concordent avec un système réel, sans frottements ni perturbations. Les deux réponses indicielles montrent des caractéristiques d'une réponse d'ordre deux, ce qui guide la création du modèle.

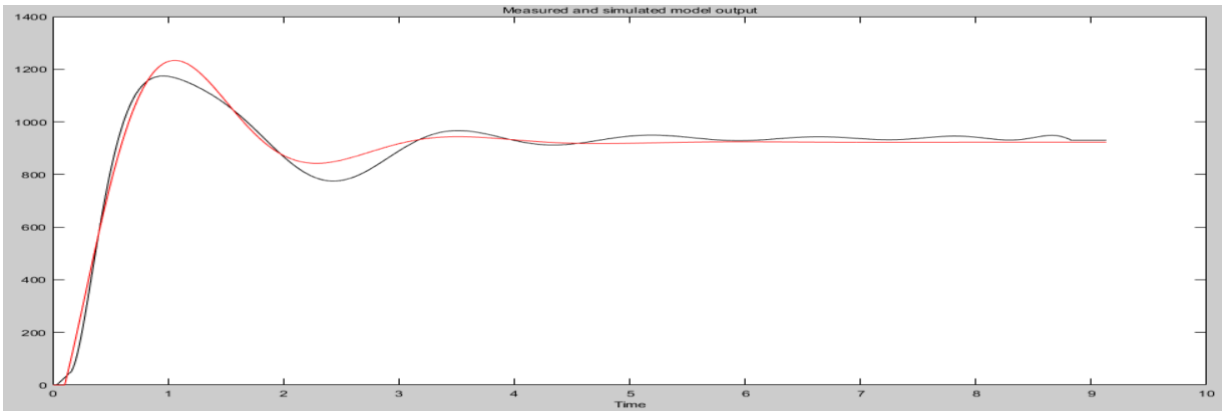
Pour créer ce modèle, nous nous appuierons sur MATLAB, en particulier sur la toolbox System Identification. Elle est utilisée pour la modélisation de systèmes dynamiques, l'analyse de séries temporelles et la prévision.

Nous commençons par importer les données corrigées (en mettant davantage l'accent sur le premier échelon) dans MATLAB. Pendant le test d'échelon, les données ont été enregistrées avec un intervalle de temps variable (enregistrement sur une carte SD en mode superloop). Par conséquent, nous allons d'abord créer une interpolation des données. Pour obtenir une série à intervalle de temps fixe. Dans notre expérience, nous souhaitons échantillonner toutes les 30 millisecondes, donc nous réalisons une interpolation à cet intervalle.

Nous importons ensuite l'interpolation créée dans la toolbox. Lors de l'importation, nous spécifions les caractéristiques de nos données, notamment notre échantillonnage discret. Nous générons diverses fonctions de transfert avec différentes caractéristiques telles que l'ordre et le retard. On obtiendra dans notre cas le résultat ci-dessous.

Les données réelles sont représentées en noir, tandis que la fonction de transfert modélisée est en rouge. Il est évident que le modèle n'est pas une réplique parfaite des données réelles à 100%. Cependant, il est important de noter que malgré cette légère différence, les temps de dépassement et de stabilisation à 2% entre le modèle et les données réelles sont quasiment identiques. Cette convergence est satisfaisante, d'autant plus que les perturbations ont introduit des déformations et des oscillations dans les amplitudes mesurées.

Les similitudes entre le modèle et les données réelles sont encourageantes. La capacité du modèle à reproduire les caractéristiques temporelles et de stabilisation du processus est un indicateur positif de sa fiabilité et de son utilité pour la modélisation du comportement de la fusée expérimentale.



Création d'un correcteur

Maintenant que nous avons développé notre modèle, nous allons concevoir un correcteur numérique approprié. Pour notre application, nous avons opté pour un correcteur Proportionnel Intégral (PI), en raison de la nature spécifique du processus que nous gérons. En effet, nous souhaitons obtenir une variation de l'accélération par un contrôle de la vitesse ce qui fait déjà un terme dérivé. Nous avons donc décidé d'exclure la composante dérivée du correcteur, car son inclusion pourrait rendre la réponse du système instable en ayant sur la fonction de transfert un ordre du numérateur supérieur à celui du dénominateur.

Dans le contexte de notre modèle de fusée expérimentale, l'absence du terme dérivé est justifiée par les caractéristiques particulières du système et la prévention des instabilités potentielles. Le correcteur PI est souvent choisi dans des situations où une réaction rapide aux erreurs est souhaitée, tout en évitant les effets indésirables de la composante dérivée, qui pourrait amplifier les variations brusques.

Le correcteur PI combine deux termes clés pour ajuster le comportement du système : le terme proportionnel et le terme intégral. Le terme proportionnel agit en proportion de l'erreur actuelle, tandis que le terme intégral agit sur la somme cumulée des erreurs passées. L'utilisation de ces deux termes vise à fournir une réponse de contrôle plus précise et stable.

Pour effectuer la synthèse de notre correcteur PI, nous faisons appel à la toolbox Control System Designer, qui propose des algorithmes et des applications pour l'analyse, la conception et le réglage de systèmes de contrôle linéaire.

Elle permet de définir des systèmes de contrôle sous différentes formes : fonctions de transfert, représentations d'état, modèles zéro-pôle-gain ou même des réponses fréquentielles.

L'utilisation de la toolbox Control System Designer simplifie grandement le processus de conception et de réglage du correcteur PI. Elle nous permet de manière intuitive de spécifier les paramètres du correcteur, tels que les gains proportionnel et intégral, en fonction de nos besoins spécifiques en matière de contrôle de l'orientation de la fusée. En utilisant les outils fournis, nous pouvons évaluer rapidement et efficacement les performances du système de contrôle, ajuster les paramètres du correcteur et obtenir une réponse stable et précise conformément à nos objectifs.

En choisissant une architecture classique de boucle de contrôle, nous avons réglé le correcteur de tel façon à avoir un temps à 2% le plus faible possible (entre 2.5 et 3), un dépassement inférieur à 30%, une marge de gain supérieure à 10dB et une marge de phase si possible vers 70°.

Après plusieurs tentatives nous avons obtenu le correcteur ci-dessous :

```
Tunable Block
Name: C
Sample Time: 0.03
Value:
  0.023667 (z-0.9526)
  -------------------
          (z-1)
```

```
float pwmmin = 1000;    //vitesse de commande max du moteur sens inverse
float pwmmax = 2000;   //vitesse de commande max du moteur sens normal

float constante_S1 = 1;
float constante_E = 0.023667;
float constante_E1 = -0.0225451842;


if (s == '2') {
  E1 = E;
  E = commande + corrGyrRawz;
  com1bis = com1bis + E * constante_E + E1 * (constante_E1 + wind_up * 0.00003);
  com1 = com1bis * 874.0 / Voltage_mot;
  if (com1 > 500) //saturateur haut
  {
    wind_up = 500 * Voltage_mot / 874 - com1bis;
    com1 = 500;
  }
  else if (com1 < -500) //saturateur bas
  {
    wind_up = -500 * Voltage_mot / 874 - com1bis;
    com1 = -500;
  }
  else
  {
    wind_up = 0;
  }
  com2 = 1500 + com1;
  if(com2 < 1580 & com2 > 1420)
  {
    com2 = 1500;
  }
}
```

Implémentation du correcteur

Intégrer le correcteur au code de l'expérience est une étape cruciale pour mettre en œuvre le contrôle du roulis. Pour réaliser cela, nous avons dû transcrire la fonction de transfert du correcteur en lignes de code compréhensible et exécutable par la Teensy. Cela a nécessité la conversion de la fonction de transfert en une équation récurrente.

Dans le contexte du contrôle de système, une équation récurrente peut représenter comment les entrées actuelles et passées influencent la sortie du système au fil du temps.

L'objectif de cette conversion est de créer une implémentation logicielle du correcteur qui puisse être exécutée en temps réel sur la Teensy, afin de réguler l'orientation de la fusée conformément aux spécifications du correcteur PI conçu

$$\frac{S}{E} = P + I * T_s \frac{1}{z-1}$$

$$S\,(z-1) = E * P * (z-1) + E * I * T_s$$

$$S\,(z-1) = E * P * z + E * (I * T_s - P)$$

$$S\,(1-z^{-1}) = E * P + E * z^{-1} * (I * T_s - P)$$

$$s_n - s_{n-1} = e_n * P + e_{n-1} * (I * T_s - P)$$

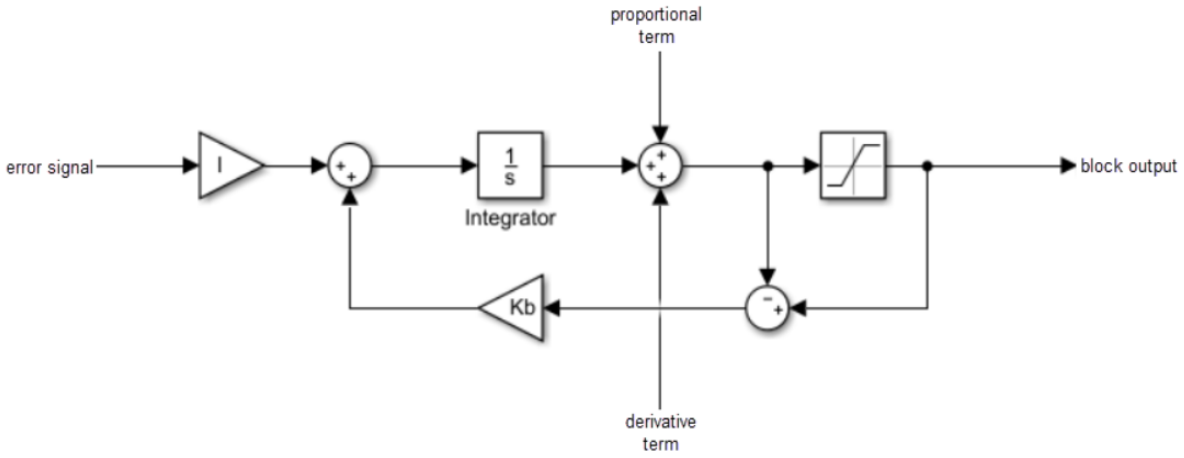
$$s_n = s_{n-1} + e_n * P + e_{n-1} * (I * T_s - P)$$

Par identification on reconnaît nos coefficients

$$P = 0.023667$$

et $I * T_s - P = 0.023667 * -0.9526 = -0.0225$

Cependant, lorsque le moteur atteint sa vitesse maximale mais que la commande continue de lui demander d'accélérer, l'erreur intégrée accumulée continue de croître, malgré l'incapacité du système à agir en conséquence. Ce phénomène est connu sous le nom de "Wind-Up".



Anti-Wind-Up

Pour résoudre ce problème de "Wind-Up", nous avons mis en place une mesure de protection. Cette mesure vise à prévenir l'accumulation excessive de l'erreur intégrée lorsque la saturation du moteur est atteinte. La technique que nous avons choisie pour remédier à cela est appelée "back-calculation". Cette approche implique un calcul rétroactif pour ajuster l'erreur intégrée en fonction des contraintes de saturation du moteur. En d'autres termes, lorsque le moteur est saturé et ne peut plus répondre à la commande, l'erreur intégrée est recalculée en fonction de la différence entre la sortie réelle du système et la valeur de consigne. On peut voir un schéma block de son implémentation sur le correcteur **ci-dessus.**

On peut donc actualiser notre équation récurrente en ajoutant la partie anti-Wind-Up :

$$\frac{S}{E} = P + (I + W * K_b) * T_s \frac{1}{z-1}$$

$$S\,(z-1) = E * P * (z-1) + E * (I + W * K_b) * T_s$$

$$S\,(z-1) = E * P * z + E * ((I + W * K_b) * T_s - P)$$

$$S\,(1 - z^{-1}) = E * P + E * z^{-1} * ((I + W * K_b) * T_s - P)$$

$$s_n - s_{n-1} = e_n * P + e_{n-1} * ((I + W * K_b) * T_s - P)$$

$$s_n = s_{n-1} + e_n * P + e_{n-1} * (I * T_s + W * K_b * T_s - P)$$

Le coefficient K_b est un coefficient de gain pour gérer la partie anti-Wind-Up, il n'est pas général à tous les systèmes et par itération nous avons fait le choix d'un gain à 0.001. On n'oubliera pas de mettre un saturateur correspondant aux limites du moteur après avoir calculé s_n.

A partir de cette équation récurrente, on peut écrire le code qui va gérer la commande du volant d'inertie.

On adapte également le code pour l'utilisation d'un ESC qui va contrôler le moteur par envoi d'un PWM, on a donc pour un ESC bidirectionnel une commande de PWM allant de 1000 à 2000 avec 1000 vitesse maximale en sens anti horaire, 1500 le point mort et 2000 vitesse maximale en sens horaire. Nous avons eu également l'effet d'une « dead zone » qui se traduit par une non rotation du moteur autour du point mort, nous avons donc mis une protection couvrant la zone 1420-1580.

Dans le cadre de ce correcteur, nous avons introduit une caractéristique additionnelle : l'ajustement de la commande en fonction de la tension de la batterie. Il est important de noter que la tension résiduelle de la batterie a un impact significatif sur la vitesse de rotation du moteur. Étant donné que l'ESC (contrôleur de vitesse électronique) agit comme un hacheur entre le microcontrôleur et le moteur, son fonctionnement est étroitement lié au niveau de la batterie. Pour compenser cette variation, nous avons normalisé la commande par rapport au niveau de batterie à pleine puissance. Ainsi, au fur et à mesure que la batterie se décharge, la commande envoyée au moteur est ajustée en conséquence pour maintenir une performance constante.

Fuselage

Le fuselage est composé d'un tube d'aluminium aéronautique extrudé de 160 cm. Il est composé d'une unique pièce de 2.5 mm d'épaisseur pour 6 cm de largeur. Cela donne à notre fusée un profil très réduit et permet de réduire le moment d'inertie.

Les découpes du fuselage ont été effectué à la Dremel et la lime. Pour aider dans cette démarche, des plans ont été créer sur Photoshop à partir des modèles 3D.

Ces plans ont été fixé sur le fuselage pour obtenir des découpes précisent. Il a fallu néanmoins faire attention aux marges sur les feuilles imprimées qui rendent les mesures incorrectes.

Le moteur fusé vient se reposer directement sur le bas du fuselage dont le diamètre interne est de 5.5cm il y'a donc un jeu non négligeable qui fut comblé à l'aide de **scotch aluminium, celui-ci a particulièrement bien résisté au vol**.

La peau étant porteuses, chaque pièce dans la fusée doit être mécaniquement attaché à celle-ci. Au vu du diamètre de notre fusée, l'utilisation de vis à tête classique ou plate causerais une trainée significative.

Ainsi, pour réduire les pertes aérodynamiques, et en prenant en compte la grande résistance mécanique de la fusée. Chaque trou de vis est chanfreiné pour accueillir des vis à tête fraisée.

Pour les trous des patins, il a été décidé de tarauder des trous droits puis de rajouter de l'époxy aluminium.

Malheureusement, les attaches des patins se sont révélé inefficace lors de la première mise en rampe et ont lâché sur la rampe fuseX.

Ainsi, nous recommandons fortement de maintenir les patins par contacts avec des boulons.

Ailerons

Les ailerons sont composés de plaques de 2 mm d'aluminium. Le diamètre de la fusée étant extrêmement petit et le moteur rentrant à peine dans celui-ci, il était impossible de fixer les ailerons depuis l'intérieur de la fusée.

La seule solution était donc de les souder. Nous avons fait appel à CTCIA pour souder nos ailerons, ce sont des professionnels de la fonderie.

Afin de garantir que les ailerons se placent bien dans la bonne position pour le soudage nous avons utilisé des plaques en bois découpé au laser via le fablab Artistik'lab.

Les soudeurs de CTCIA ont alors soudé la fusée en 4 points de chaque côté des ailerons. Suite à cela, nous avons utilisé de l'époxy pour remplir les cavités entre les trous et créer un congé aérodynamique autour des ailerons.

Bague d'ajout de masse

Après l'envoie des Stabtraj supposé définitif de la fusée, Planète science nous as informé que la portée de la fusée sur Stabraj, avec un angle de 45° et un coefficient de trainée de 0,6 devait être inférieur à 4000m, malgré l'absence d'information dans le cahier des charges et dans le document du terrain de Ger.

Il a donc été nécessaire d'ajouter de la masse dans la fusée pour répondre au critère de la sauvegarde.

Plusieurs solutions ont été envisagé, avant de se concentrer sur un ajout de masse au-dessus du moteur. Cela a pour avantage d'augmenter la stabilité de la fusée. L'acier a vite était sélectionnée pour ça résistance mécanique, ça facilité d'usinage et son poids.

Pour créer cette bague, deux pièces ont été utilisé :

- Une section de cylindre en acier avec une finition droite.
- Une bague en acier usiné via le site « Usineur.fr »

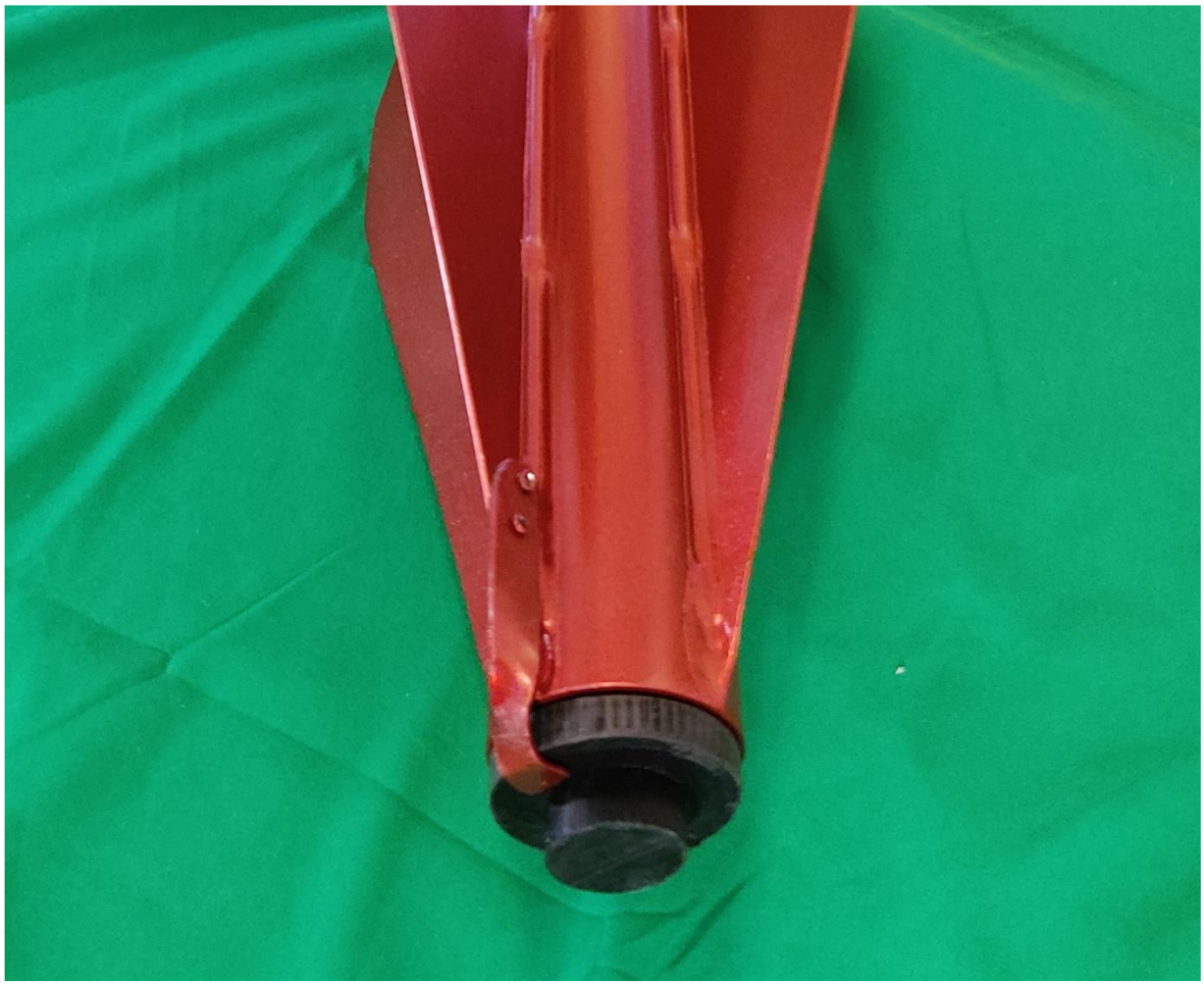
Les deux pièces sont soudées ensemble avec l'aide d'une pièce imprimée en PLA. Puis les trous sont taraudés (l'utilisation d'un taraud sur de l'acier a causé beaucoup d'imprévus).

Avant l'installation finale dans la fusée, la masse est pesée est ajustée.

Pour ajuster la masse, on choisit de souder un surplus sur la face supérieur du cylindre dans le cas où la masse est trop faible.

Dans le cas contraire, on perce la somme du cylindre pour enlever de la masse.

La masse ainsi créée pèse environ 1 Kg et représente environ 12% de la masse de la fusée et change notre apogée de 3 km (la limite haute du terrain de Tarbes) à 2.3 km.



Système de rétention moteur

Afin de supporter le moteur Pro54 avant le lancement, la fusée doit être équipée d'un système de rétention moteur.

Lorsque la fusée est positionnée à la verticale sur la rampe, le moteur doit rester plaqué contre le corps de la fusée.

Afin de garantir un système peu cher, nous avons choisi de créer notre bague à partir de chute d'aluminium utilisé pour les ailerons.

La pièce est pliée avec l'aide d'un patron imprimée en 3D pour obtenir la forme finale désirée.

Afin de faciliter la mise en place par le pyrotechnicien. On utilise deux vis, une première n'est pas démontable, et sert de pivot pour placer le moteur.

Ensuite, le pyrotechnicien place le Pro54 dans la fusée, avant de fermer le système de rétention et visser la seconde vis (avec l'aide d'un tournevis fournis par Elisa Space), cette vis maintient le système en position pour le décollage.

Ce système fonctionnera parfaitement pour le lancement et retiendra le moteur durant toute la phase de vol.

Autocollants et peintures

Avant de peindre notre fusée, nous avons commencé par protéger le sol et les alentours avec des sacs poubelles.

Le vent et le pollen nous ont empêcher de faire cela en extérieur, nous avons donc dû peindre dans un appartement en protégeant le plus de surface possible. Malheursement, la peinture des aérosols se dépose sur toutes les surfaces possibles et change la couleur de la pièce.

Avant de peindre, il faut poncer et nettoyer la surface de la fusée avec de l'acétone. Ensuite on pose une couche de primer pour l'adhérence des prochaines couches de peinture. Il est préférable d'utiliser des aérosols car ils permettent une application homogène de la peinture.

Après la mise en place du primer, nous avons passé plusieurs petites couches de peintures entrecoupées de longues pauses.

Enfin, avant que la peinture ne commence à sécher, nous avons enlevé les scotchs de démarcations pour la séparation des couleurs. Il est préférable de passer un coup de cutter avant d'enlever les scotchs pour éviter d'arracher la peinture.

Ensuite, on peut placer des scotchs et passer à la couleur suivante.

Enfin, on place les autocollants sur la fusée et la trappe (dans le bon sens).

Et on finit par l'application d'une couche de vernis.

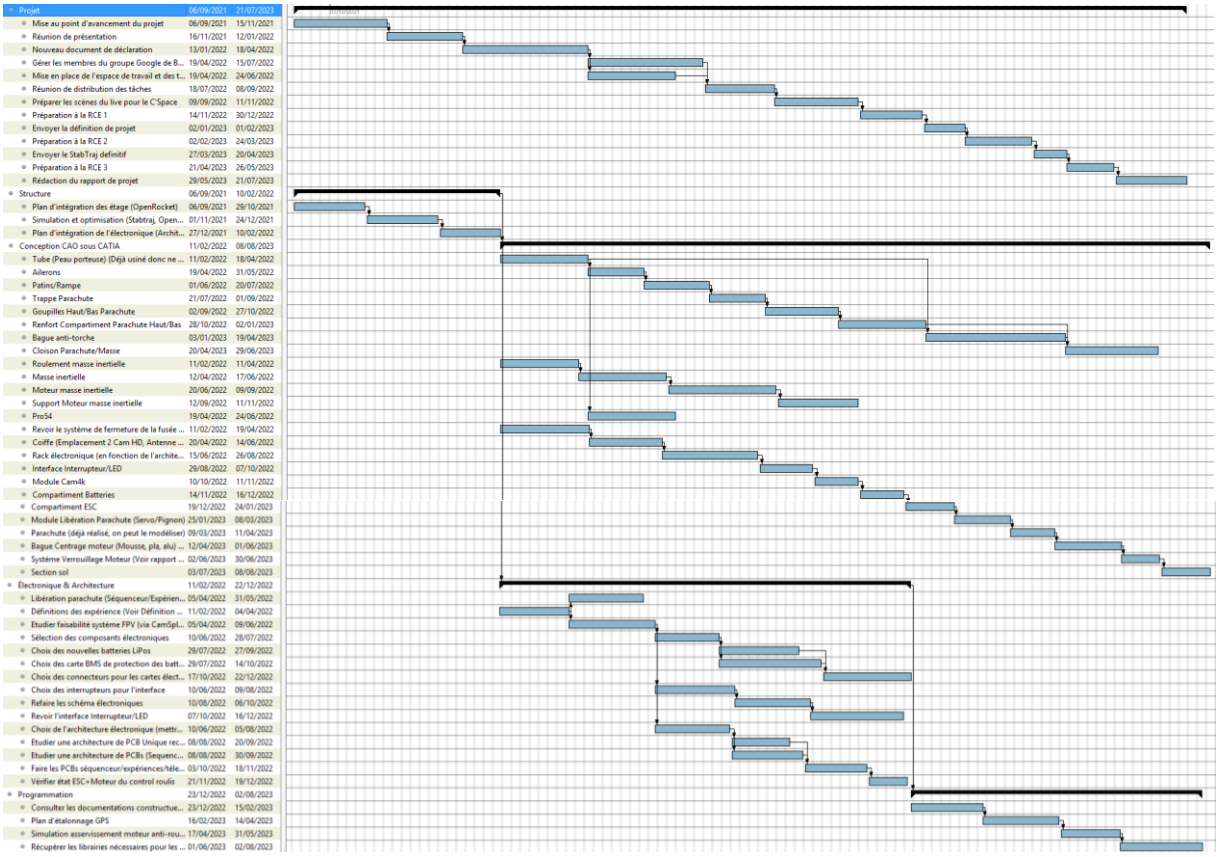
Chronologie

La chronologie représente la check-list des actions à effectuer avant le lancement de la fusée. Elle a nécessité **plusieurs jours à être mise au point**, au C'Space, et **doit être extrêmement précise**.

Durée	Lieu	Opération	Séquence
	Ateliers	Imprimer la chronologie 4 fois	
		Veille au soir	
30 min	Ateliers	Vider les cartes SD	I
3h	Ateliers	Recharger 2 batteries	I
2h	Ateliers	Recharger 2 caméras	I
3h	Ateliers	Rechargements ordinateurs + tous les téléphones	I
1h	Ateliers	Rechargement des caméras sols	I
5 min	Ateliers	Test logiciel télem/connexion internet	I
10 min	Ateliers	Préparer l'onduleur	I
		Pres déplacement	
15 min	Ateliers	Matériel : rallonges/scotch/tournevis/jack/onduleur/trépieds/serre joint/CAMx2/chargeur/antenne	II
15 min	Ateliers	Annonce d'un Stream et heure approx	II
5 min	Ateliers	Mise en place des piles	II
3 min	Ateliers	Mise en place des Batteries	II
10 min	Ateliers	Mise en place des cartes SD VIERGES données/ vierge caméra HD/1/2	II
5 min	Ateliers	Attention ! Mettre scotch sur carte SD FuseX	II
5 min	Ateliers	Mise en place des caméras 1/2	II
10 min	Ateliers	Scotch connecteur système anti roulis !	II
30 min	Ateliers	Remonter la fusée	II
10 min	Ateliers	Vérifier liste matérielle + Organisation sac	II
		En tente public	
10 min	Public	Revissage de toutes les vis	III
1 min	Public	Vérifier branchement et état des prises/cables dans la case	III
5 min	Public	Test allumage complet (séquenceur/moteur/données)	III
2 min	Public	Vérification ouverture parachute (avec serre joint)	III
4 min	Public	Rallumer le séquenceur pour fermer la trappe	III
2 min	Public	Couper tout	III
10 min	Public	Préparation déplacement fusée : câble jack + pince + tournevis x 2+scotch+cams	III
		Zone télémétrie Jupiter (Melvin)	
1 min	Jupiter	Installation pc	IV
2 min	Jupiter	Allumage PC	IV
5 min	Jupiter	Connexion wifi pour le PC	IV
1 min	Jupiter	Installation hub usb	IV
1 min	Jupiter	Installation webcam	IV
1 min	Jupiter	Installation Ecran	IV
3 min	Jupiter	Installation Sourie	IV
1 min	Jupiter	Préparer onduleur	IV
5 min	Jupiter	Connecter le chargeur au PC	IV
1 min	Jupiter	Connecter chargeur à la rallonge	IV
3 min	Jupiter	Connecter la rallonge au réseau électrique	IV
1 min	Jupiter	Installation trépied télém	IV
1 min	Jupiter	Installation antenne	IV

3 min	Jupiter	Câblage antenne-télém	IV
1 min	Jupiter	Câblage télém-PC	IV
1 min	Jupiter	Câblage HUB USB PC	IV
1 min	Jupiter	Câblage webcam HUB USB	IV
1 min	Jupiter	Câblage sourie HUB USB	IV
1 min	Jupiter	Câblage écran au PC câble USB	IV
2 min	Jupiter	Câblage écran au PC câble HDMI	IV
1 min	Jupiter	Lancer Obs	IV
3 min	Jupiter	Scène Belisama	IV
3 min	Jupiter	Lancer VisUI + config + save data	IV
2 min	Jupiter	Vérifier scène obs	IV
1 min	Jupiter	Préparer la page internet du live	IV
		En zone rampe	
1 min	Rampe	Enlever le serre joints	V
3 min	Rampe	Demander autorisation allumage télém 869.5	V
2 min	Rampe	Allumage télém+ expérience	V
2 min	Rampe	Confirmation réception telem	V
5 min	Rampe	Étalonnage du compas (faire des 8 avec la fusée) + cohérence données	V
1 min	Rampe	Cohérence des données	V
		Mise en rampe de la fusée	
3 min	En rampe	Mise en place caméra près de la rampe	VI
1 min	En rampe	Allumage caméra près de la rampe	VI
2 min	En rampe	Mise en place caméra plan large	VI
1 min	En rampe	Allumage caméra plan large	VI
1 min	En rampe	Noter l'heure	VI
3 min	En rampe	**Allumage caméra fusée 1**	VI
3 min	En rampe	**Allumage caméra fusée 2**	VI
1 min	Jupiter	Redémarrage VisUI	VI
1 min	Jupiter	Vérification télém	VI
1 min	Jupiter	Qualité donnée	VI
2 min	Jupiter	Recharger la config	VI
1 min	Jupiter	Check OBS	VI
1 min	Jupiter	Check Enregistrement données	VI
2 min	En rampe	Accroche de la prise jack sur le pas de tir	VI
2 min	En rampe	Branchement et maintien de la prise Jack	VI
1 min	En rampe	Allumage moteur + bruit	VI
1 min	En rampe	Allumage séquenceur LED continue	VI
1 min	En rampe	Vérifier trappe	VI
1 min	Jupiter	Vérification donnée télém Sérial	VI
1 min	En rampe	Vérification visuelle LED/Interrupteur/Cable Jack	VI
2 min	En rampe	Scotch tournevis pyro à la fusée	VI
		Orientation de la rampe	
1 min	Jupiter	Vérification état fusée télém	VII
		Equipe en tente pupitre PYRO	
5 min	Jupiter	OBS commencer streaming + enregistrement	VIII
2 min	Jupiter	Firefox passer au direct	VIII
1 min	Jupiter	Ultime vérification télémétrie/qualité donnée/Stream/batterie pc	VIII
1 min	Zone pyro	Vérification heure de décollage	
2 min	Zone pyro	Caméra / téléphones si possible Ok	VIII
3 min	Jupiter	Antenne pointée à la main	VIII
0 min		Lancement de la fusée	

Planning effectif



Projet	06/09/2021	21/07/2023
Mise au point d'avancement du projet	06/09/2021	15/11/2021
Réunion de présentation	16/11/2021	12/01/2022
Nouveau document de déclaration	13/01/2022	18/04/2022
Gérer les membres du groupe Google de B...	19/04/2022	15/07/2022
Mise en place de l'espace de travail et des t...	19/04/2022	24/06/2022
Réunion de distribution des tâches	18/07/2022	08/09/2022
Préparer les scènes du live pour le C'Space	09/09/2022	11/11/2022
Préparation à la RCE 1	14/11/2022	30/12/2022
Envoyer la définition de projet	02/01/2023	01/02/2023
Préparation à la RCE 2	02/02/2023	24/03/2023
Envoyer le StabTraj definitif	27/03/2023	20/04/2023
Préparation à la RCE 3	21/04/2023	26/05/2023
Rédaction du rapport de projet	29/05/2023	21/07/2023
Structure	06/09/2021	10/02/2022
Plan d'intégration des étage (OpenRocket)	06/09/2021	29/10/2021
Simulation et optimisation (Stabtraj, Open...	01/11/2021	24/12/2021
Plan d'intégration de l'électronique (Archit...	27/12/2021	10/02/2022
Conception CAO sous CATIA	11/02/2022	08/08/2023
Tube (Peau porteuse) (Déjà usiné donc ne ...	11/02/2022	18/04/2022
Ailerons	19/04/2022	31/05/2022
Patins/Rampe	01/06/2022	20/07/2022
Trappe Parachute	21/07/2022	01/09/2022
Goupilles Haut/Bas Parachute	02/09/2022	27/10/2022
Renfort Compartiment Parachute Haut/Bas	28/10/2022	02/01/2023
Bague anti-torche	03/01/2023	19/04/2023
Cloison Parachute/Masse	20/04/2023	29/06/2023
Roulement masse inertielle	11/02/2022	11/04/2022
Masse inertielle	12/04/2022	17/06/2022
Moteur masse inertielle	20/06/2022	09/09/2022
Support Moteur masse inertielle	12/09/2022	11/11/2022
Pro54	19/04/2022	24/06/2022
Revoir le système de fermeture de la fusée ...	11/02/2022	19/04/2022
Coiffe (Emplacement 2 Cam HD, Antenne ...	20/04/2022	14/06/2022
Rack électronique (en fonction de l'archite...	15/06/2022	26/08/2022
Interface Interrupteur/LED	29/08/2022	07/10/2022
Module Cam4k	10/10/2022	11/11/2022
Compartiment Batteries	14/11/2022	16/12/2022
Compartiment ESC	19/12/2022	24/01/2023
Module Libération Parachute (Servo/Pignon)	25/01/2023	08/03/2023
Parachute (déjà réalisé, on peut le modéliser)	09/03/2023	11/04/2023
Bague Centrage moteur (Mousse, pla, alu) ...	12/04/2023	01/06/2023
Système Verrouillage Moteur (Voir rapport ...	02/06/2023	30/06/2023
Section sol	03/07/2023	08/08/2023
Électronique & Architecture	11/02/2022	22/12/2022
Libération parachute (Séquenceur/Expérien...	05/04/2022	31/05/2022
Définitions des expérience (Voir Définition ...	11/02/2022	04/04/2022
Etudier faisabilité système FPV (via CamSpi...	05/04/2022	09/06/2022
Sélection des composants électroniques	10/06/2022	28/07/2022
Choix des nouvelles batteries LiPos	29/07/2022	27/09/2022
Choix des carte BMS de protection des batt...	29/07/2022	14/10/2022
Choix des connecteurs pour les cartes élect...	17/10/2022	22/12/2022
Choix des interrupteurs pour l'interface	10/06/2022	09/08/2022
Refaire les schéma électroniques	10/08/2022	06/10/2022
Revoir l'interface Interrupteur/LED	07/10/2022	16/12/2022
Choix de l'architecture électronique (mettr...	10/06/2022	05/08/2022
Etudier une architecture de PCB Unique rec...	08/08/2022	20/09/2022
Etudier une architecture de PCBs (Sequenc...	08/08/2022	30/09/2022
Faire les PCBs séquenceur/expériences/téle...	03/10/2022	18/11/2022
Vérifier état ESC+Moteur du control roulis	21/11/2022	19/12/2022
Programmation	23/12/2022	02/08/2023
Consulter les documentations constructue...	23/12/2022	15/02/2023
Plan d'étalonnage GPS	16/02/2023	14/04/2023
Simulation asservissement moteur anti-rou...	17/04/2023	31/05/2023
Récupérer les librairies nécessaires pour les ...	01/06/2023	02/08/2023

Les grandes étapes du projet ont été réparties et planifiées à l'avance afin d'avoir un suivi clair de l'avancement de projet. De plus, les différentes équipes sont réparties entre anciens et nouveaux membres dans le but de favoriser le partage d'expérience tout au long du projet et permettre ainsi aux différents membres d'être pleinement au courant des caractéristiques du projet.

L'équipe

LES PERSONNES DERRIÈRE BELISAMA

Florian LEVRAY Chef de projet, CAO et Intégration, Usinage, Électronique, Circuits imprimés & Télémesure.

Léo-Paul LAURENT Spécialiste Stabilisation, Électronique et Programmation.

Melvin SEAUVE Spécialiste Simulation et Logiciels.

Rémi VINCENT CAO et Intégration, Usinage, Électronique, Programmation, Circuits imprimés & Télémesure.

Marie BRUN Spécialiste système de récupération.

L'équipe de Bélisama est composée de 5 étudiants en formation ingénieur à l'École d'Ingénieurs des Sciences Aérospatiales de Saint-Quentin.

Campagne de lancement

La campagne de lancement du C'Space 2023 a été très éprouvante pour notre équipe.

Nous sommes arrivés le vendredi 14 juillet. Et nous avons commencé les qualifications dans les premiers. Dans un premier temps, nous avons eu des problèmes avec la comptabilité de nos patins à la rampe. Nous avons remplacé ceux-ci par de l'impression 3D qui a passé toutes les qualifications.

Ensuite, nous avons effectué le vol simulé plusieurs fois à cause de la mauvaise qualité de notre chronologie.

Finalement le mercredi, nous avons placé la fusée en rampe… et la fusée est tombé de la rampe.

Suite à cet incident, nous avons remplacé les patins en 3D par des patins en métal de l'équipe de **Mine Space** et nous les avons fixé avec des écrous freinés.

Malheureusement suite à cet incident, l'électronique faisait des siennes. Après beaucoup de recherche le problème a été identifié. Mais une erreur a grillé l'électronique.

Celle-ci a dû être intégralement ressouder sur un PCB neuf avec des composants neuf et un GPS ne fonctionnant plus. Entre le mercredi soir et le jeudi midi. Nous avons ensuite directement passé le vol simulé en mort subite et nous avons été requalifié.

Nous avons fini par lancer le vendredi à 11h55 pour une fermeture du C'Space à 12h. Suite à un long feu sur la fusée nous précédant, les pyrotechniciens ont décider de permettre le lancement de notre fusée puis de la fusée précédant un peu après.

Déroulement du vol

Initialement, le moteur a poussée de manière nominale. Cependant, quelques millisecondes après le démarrage du moteur, celui-ci a eu un dysfonctionnement. Le joint et le casing ont été percé sur un côté en bas du moteur.

Suite à cette fuite, la poussée du moteur c'est drastiquement réduit et la poussée latérale a commencé à déstabiliser la fusée.

Le manque de poussée a ainsi empêché la prise de vitesse de la fusée. Au fur et à mesure que celle-ci gagnait en altitude elle perdait de la vitesse.

Cette perte de vitesse réduit donc la capacité de la fusée à voler droit. Le moteur finira par complétement déstabiliser la fusée à l'apogée et effectuer 2 boucles sur elle-même.

Lors d'une de ces boucles on peut remarquer le patin en acier quitter la fusée dû à la chaleur du disfonctionnement moteur.

Après l'extinction du moteur, la fusée a effectué un balistique, s'écrasant avant la durée minimale d'ouverture du parachute.

Il convient de noter que même dans ces conditions dégradées. L'algorithme d'ouverture automatique du parachute aurait déclenché l'ouverture du parachute si le fenêtrage temporel ne lui avait pas empêcher l'ouverture.

Suite à l'impact, le recul du moteur dont la partie inférieure à totalement fondu à laisser un nuage de fumée très visible.

Heureusement, la télémesure a parfaitement fonctionné et nous as permis de récolter une partie des données.

Récupération

Dans l'après-midi suivant le vol de notre fuseX. Nous sommes allées la récupérer, malgré le vol balistique nous étions confiants pour cette récupération car la fuseX n'a pas dépasser les 300 m d'altitude.

Suite à plusieurs coordonnées GPS non valides, nous avons repéré la fusée dans une grande zone de hautes herbes. Après avoir aplanit la zone, nous avons attaqué la récupération à la pioche. Le profil de notre fusée, étant très fine, a favorisé la perforation de celle-ci dans le sol. Il a donc fallu un temps conséquent pour creuser jusqu'au trou de la trappe.

Notre fusée étant très rigide, nous avons tourné la fusée sur elle-même pour créer une carotte de terre contenant la plupart des systèmes avant de l'extraire. Nous avons ensuite placé les débris dans des sac poubelles.

Les pyrotechniciens ont demandé à examiner la fusée pour identifier le problème. Mais le moteur n'a pas pu être extrait.

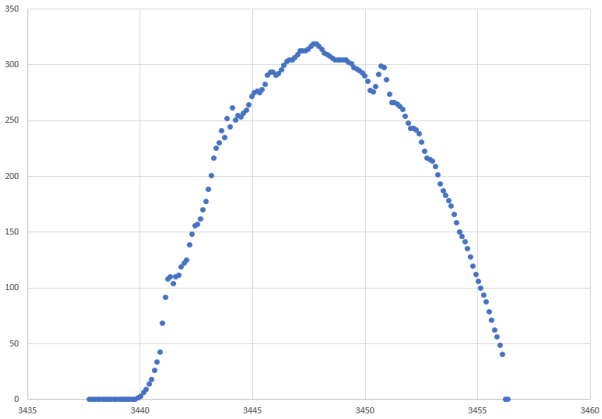
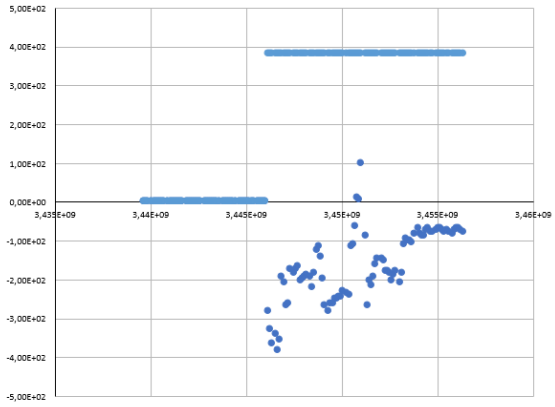
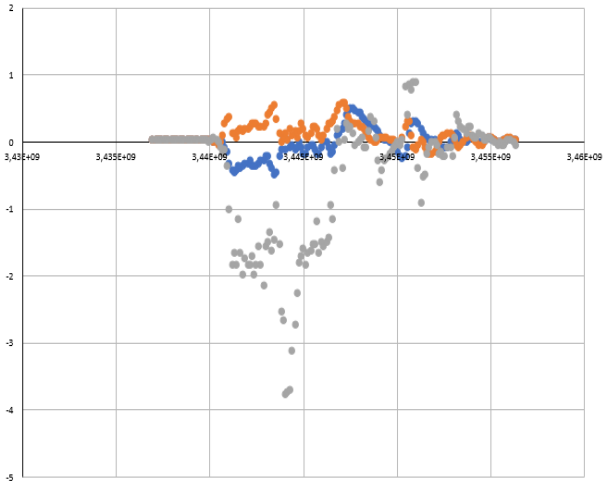
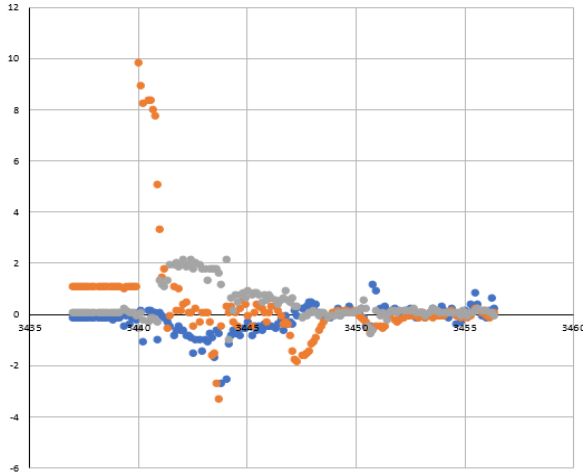
Après le C'Space, la fusée a été stocké dans une cave avant de déloger les batteries. Et de récupérer les cartes SD. Etonnamment le séquenceur montrait encore des signes de vie et s'est allumé lors de cette procédure.

Malheureusement, la totalité des cartes SD sont illisibles. Cependant la carte des données n'est pas craqué et a été pris en charge par une entreprise de récupération des données.

A ce jour les données n'ont pas encore été récupérés. Les données utilisables sont donc issues de la télémesure.

Après avoir délogé l'électronique et les batteries. Nous avons cherché à déloger le moteur. Le bas du tube et le moteur s'étant déformé, cela a nécessité des forces contraintes mécaniques qui ont endommagé le moteur et la fusée.

Suite à cela nous avons pu extraire la masse et le volant d'inertie qui ont été écrasé par le moteur au moment de l'impact.

Exploitation des résultats

Durant le vol, la télémesure a parfaitement jouée son rôle. Le logiciel VisUI a bien enregistré les trames qui nous ont permis de reconstituer le vol avec un pas de temps de 120 ms au lieu des 30 ms dans la fusée.

Nous nous sommes rendu compte suite au vol que le magnétomètre utilisé était une contrefaçon. Ainsi, nous n'avions pas de magnétomètre durant le vol, et cela a ajouté de la difficulté pour estimer l'attitude de la fusée et intégrer les quaternions. Ce problème nous empêche pour l'instant de réaliser une trajectographie 3D.

Malgré cela, les données reçues sont cohérentes, l'altitude atteinte suite au problème moteur était donc d'environ 320 m, ce qui est un record pour l'association (**graphe en haut à gauche**).

Le système de contrôle du roulis a joué son rôle et il se stabilise avant l'impact en 7.7 secondes après son activation mais seulement 2 secondes après l'extinction du moteur. Le système étant gêné par les mouvements de la fusée dû au moteur (**graphe en haut à droite**).

Le système d'ouverture du parachute n'était pas activé mais s'il l'avait été avec les données à bord il se serait activé d'après nos simulations effectué au sol avec l'algorithme de bord, le parachute se serait déployé si le système avait été actif.

Les courbes d'accélérations montrent l'anomalie de la poussée du moteur qui a effectué une forte poussée sur le côté de la fusée entrainant une accélération quasi constante de 2G sur l'axe X pendant 3 secondes (**graphe en bas à droite**).

Retours d'expériences

- **Toujours faire plusieurs électroniques en spare :**

Nous avions prévu des composants de remplacements mais nous n'avions pas assemblé d'électronique de secours. Il est plus utile de créer 2 électroniques fonctionnelles et d'utiliser la plus propre en vol (s'éviter un maximum de problème) tout en gardant l'autre en cas de secours.

- **Utiliser des systèmes de lissage / protection sur le PCB :**

L'utilisation de systèmes de sécurité (diode Zener et Photocoupleur) permettent de protéger le microcontrôleur. Ils peuvent rendre un système résiliant à certaines erreurs : Zener limite le courant, Photocoupleur isole le système. Des condensateurs peuvent être utilisé pour lisser le courant en entrée du système.

- **Ne jamais effectuer des actions sous tension même avec interrupteur fermé :**

La présence de pile ou batterie peut facilement détruire un système électronique. Il est nécessaire d'utiliser un système d'alimentation externe avec interrupteur et protéger des courts-circuits.

- **Intégration dans un tube trop petit :**

L'idéal serait d'obtenir un diamètre de 15 cm au détriment du moment d'inertie.

- **Vérifier la qualité des capteurs en Spare**
- **Vérifier les méthodes de calibrations**
- **Utiliser des patins en acier et/ou plastique du commerce/usiné et non en impression 3D fils ou résine :**

Les patins sont des éléments importants et doivent résister aux forces appliquées et à l'usure. Leur résistance mécanique garanti que la fusée n'aura pas de chute ou dommage.

- **Fixer de manière très résistante les patins au corps de la fusée :**

Les patins sont soumis à des contraintes non négligeables. Il est nécessaire de sécuriser leur pose à l'aide d'un écrou bloquant à l'intérieur du corps de la fusée.

- **Effectuer une chronologie complète avant le C'Space**

Pour éviter de perdre du temps au C'Space, effectuez un vol simulé avant le C'Space et créer une liste du matériel nécessaire au lancement.

- **Utiliser de la mousse expansive pour ouvrir le parachute :**

Afin de garantir une bonne ouverture sans utiliser de ressorts, de la mousse très expansive peut être utilisé. **Attention cependant à ce qu'elle ne gêne pas le système d'ouverture.**

- **Une séparation en acier entre le moteur et la fusée est une bonne chose :**

En cas de défaillance moteur, même en bas de celui-ci les flammes remontent le long du tube.

Conclusion

Malgré un développement et surtout, une campagne extrêmement sinueuse. Nous sommes parvenus à lancer la première fusée expérimentale de notre association, et de la même manière nous avons aussi effectué le premier balistique et la première récupération de fusée de l'association.

Malgré un échec moteur que nous ne pouvions pas anticiper, les deux tiers des expériences prévus ont joué leur rôle.

Le système de contrôle de roulis s'est activé, le système d'ouverture du parachute aurait été actionné s'il n'avait pas été inhibé avant la phase d'ouverture.

Concernant l'expérience de trajectographie 3D, elle n'a pas abouti pour plusieurs raisons. Notamment la perte de certains capteurs comme le magnétomètre et le GPS. Nous travaillons encore pour tenter d'exploiter les données de cette expérience. Et nous avons pour espoir que les données dans la carte SD soit un jour récupérées pour les exploiter.

Finalement, le retour d'expérience de ce projet et cette campagne permettra de guider et améliorer les futures fusées expérimentales de l'association.

Notamment la fusée expérimentale Elytra prévue qui devrait être lancée en 2024 au C'Space.

Qui contacter ?

Par mail

Adresse mail du chef de projet : florian.levray@free.fr

Adresse mail du club : spaceproject@elisa-aerospace.fr

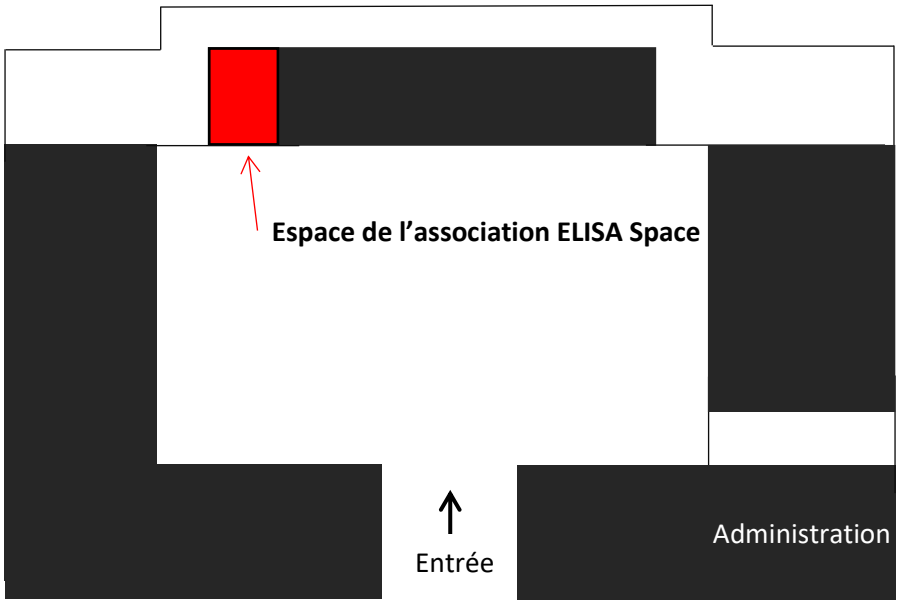
Site internet du club : https://www.elisa-space.fr

Par les réseaux sociaux :

Elisa Space | **Elisa_Space_** | **Elisa Space-Hdf** | **ELISA Space**

Où nous trouver ?

Notre quartier général se trouve actuellement à l'École d'Ingénieurs des Sciences Aérospatiales (ELISA-Aerospace) de Saint-Quentin (Hauts-de-France). C'est ici que l'on se retrouve pour organiser des réunions et travailler sur nos projets.



48 Rue Raspail, 02100, Saint-Quentin

Annexe :

Programme séquenceur :

```
/***************************************/
#include <Servo.h> //bibliothéque servo (déjà présente sur arduino)
Servo Servomoteur; //créer un objet servo
#define Pin_LED 4
#define Pin_Jack 10
#define Pin_Servo 11
#define Pin_Declenchement A5
const int Position_initiale=130; //65
const int Position_finale=90; //5

unsigned long dt1_millis=4000;    //temps avant plage controle roulis en millisec
unsigned long dt2_millis=19500;    //temps avant plage d'ouverture para en millisec
unsigned long dt3_millis=22500;    //temps ouverture para obligatoire en millise
unsigned long dt4_millis=24500;    //temps arrêt moteur en millisecc
unsigned long dt5_millis=500000;   //temps arrêt telem en millisec
unsigned long t1_millis;
unsigned long t2_millis;
unsigned long t3_millis;
unsigned long t4_millis;
unsigned long t5_millis;
unsigned long t0;

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

void setup() {
  Servomoteur.attach(Pin_Servo);   // Attache le servomoteur au PinServo
  Servomoteur.write(Position_initiale);          // Met le servomoteur en position
  Serial.begin(9600);
  Serial.print("test");
  pinMode(Pin_Jack,INPUT_PULLUP);              //set as INPUT
  pinMode(Pin_LED,OUTPUT);
  pinMode(Pin_Declenchement,INPUT_PULLUP);

  digitalWrite(Pin_LED, HIGH);
}

void loop() {
  if(HIGH==digitalRead(Pin_Jack)){
    t0 = millis();                             // prend T0 mission
    Serial.print('1');
    t1_millis = dt1_millis+t0;
    t2_millis = dt2_millis+t0;
    t3_millis = dt3_millis+t0;
    t4_millis = dt4_millis+t0;
    t5_millis = dt5_millis+t0;
    while(millis()<=t1_millis){             //attente fin de propulsion
      Serial.print('1');                    //Phase propulsé
      doblink();
    }
    while(millis()<=t2_millis){             //attente début phase d'ouverture
      Serial.print('2');                    //Plage roulis
      doblink();
    }
    while(millis()<=t3_millis){             //attente début fin d'ouverture
      Serial.print('3');                    //Attente ordre ouverture
      doblink();
      if(digitalRead(Pin_Declenchement)==0){            //Si ordre d'ouverture
        Serial.print('4');                              //Ouverture para auto
        Ouverture();
        doblink();
```

```
        while(millis()<=t4_millis){
          Serial.print('4');
          doblink();
        }
        after_openning();
      }
    }
    Serial.print('5');             //Si pas d'odre d'ouverture dans le temps impartis, ouverture temps
    Ouverture();
    doblink();
    while(millis()<=t4_millis){             //Attente ordre de  moteur
      Serial.print('5');                    //Ouverture temps
      doblink();
    }
    after_openning();             //Fonction de fin de vol
  }
  else
  {
    Serial.print('0');            //signal d'attente
  }
}

void after_openning(){
  while(millis()<=t5_millis){     //attente d'arrêt télém
    Serial.print('6');            //Arrêt du moteur
    doblink();
  }
  while(1){
    Serial.print('7');            //ordre de fin télém
    doblink();                    //Arrêt télém
  }
}

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

Commande du volant d'inertie :

```
  float pwmmin = 1000;    //vitesse de commande max du moteur sens inverse
  float pwmmax = 2000;   //vitesse de commande max du moteur sens normal

  float constante_S1 = 1;
  float constante_E = 0.023667;
  float constante_E1 = -0.0225451842;


  if (s == '2') {
    E1 = E;
    E = commande + corrGyrRawz;
    com1bis = com1bis + E * constante_E + E1 * (constante_E1 + wind_up * 0.00003);
    com1 = com1bis * 874.0 / Voltage_mot;
    if (com1 > 500) //saturateur haut
    {
      wind_up = 500 * Voltage_mot / 874 - com1bis;
      com1 = 500;
    }
    else if (com1 < -500) //saturateur bas
    {
      wind_up = -500 * Voltage_mot / 874 - com1bis;
```

```
        com1 = -500;
      }
      else
      {
        wind_up = 0;
      }
      com2 = 1500 + com1;
      if(com2 < 1580 & com2 > 1420)
      {
        com2 = 1500;
      }
    }
```

Gestion du volant d'inertie :

```
/-----------Algo_contrôle_de_roulis-------------


    //Serial.println(s);
    //Serial.println("Activation roulis");

//Serial.println((byte)s);
    commande = 380; //1 tour 15.5 s test
    digitalWrite(Pin_LED, HIGH);


    float pwmmin = 1000;    //vitesse de commande max du moteur sens inverse
    float pwmmax = 2000;   //vitesse de commande max du moteur sens normal

    float constante_S1 = 1;
    float constante_E = 0.023667;
    float constante_E1 = -0.0225451842;


    if (s == '2') {
      E1 = E;
      E = commande + corrGyrRawz;
      com1bis = com1bis + E * constante_E + E1 * (constante_E1 + wind_up * 0.00003);
      com1 = com1bis * 874.0 / Voltage_mot;
      if (com1 > 500)
      {
        wind_up = 500 * Voltage_mot / 874 - com1bis;
        com1 = 500;
      }
      else if (com1 < -500)
      {
        wind_up = -500 * Voltage_mot / 874 - com1bis;
        com1 = -500;
      }
      else
      {
        wind_up = 0;
      }
      com2 = 1500 + com1;
      if(com2 < 1580 & com2 > 1420)
      {
        com2 = 1500;
      }
    }
    else if(s == '3' | s == '4' | s == '5') {}
    else if(s == '6')
    {
       if( (com2 < 1540 & com2 > 1500) | (com2 > 1440 & com2 < 1500) )
       {
        com2 = 1500;
       }
       else if( com2 > 1500)
```

```
       {
        com2 = com2 - 1;
       }
       else if( com2 < 1500)
       {
        com2 = com2 + 1;
       }
    }
    else {
      com2 = 1500;
      com1 = 0;
      com1bis = 0;
    }


    Moteur.writeMicroseconds(com2);
```

Ouverture automatique du parachute :

```
//---------------Algo ouverture para-----------------

      //Serial.println("activation mode ouverture para");
      float seuil = 0.3;


      float average_pressure = p;
      for (int i = 4; i >= 0; i--)
      {
        average_pressure = average_pressure + pression_lisse[i];
      }
      average_pressure = average_pressure / 6 ;


      if (Lisse == 4)
      {
        Lisse = 0;
        pression_lisse[Lisse] = average_pressure;
      }
      else
      {
        Lisse++;
        pression_lisse[Lisse] = average_pressure;

      }


      if (Last == 9)
      {
        Last = 0;
        pression_memory[Last] = pression_lisse[Lisse];
      }
      else
      {
        Last++;
        pression_memory[Last] = pression_lisse[Lisse];;
      }


      float max1 = pression_memory[Last];
      for (int j = 0; j < 10; j++)
      {
        //Serial.println(pression_memory[j]);
        if (max1 < pression_memory[j])
        {
          max1 = pression_memory[j];
        }
      }
```

```
      float min1 = pression_memory[Last];
      for (int j = 0; j < 10; j++)
      {
        if (min1 > pression_memory[j])
        {
          min1 = pression_memory[j];
        }
      }


      if ( (max1 - min1 < seuil) and s == '3' ) {
        digitalWrite(PIN_Declenchement, LOW);
      }
```

Setup expérience:

```
void chSetup() {
  // Start analgRead() thread.

  chThdCreateStatic(waThread1, sizeof(waThread1),
                    NORMALPRIO + 3, Thread1, NULL);

  // Start print thread.
  chThdCreateStatic(waThread2, sizeof(waThread2),
                    NORMALPRIO + 1, Thread2, NULL);


}
```

Thread principal:

```
static THD_WORKING_AREA(waThread1, 2048);

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


  systime_t wakeTime = chVTGetSystemTime();

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

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

Gestion du temps et Overflow :

```
unsigned long t2;
    t2 = t;
    t = micros();

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

Threads GPS:

```
static THD_WORKING_AREA(waThread2, 512);

static THD_FUNCTION(Thread2, arg) {
  (void)arg;

  systime_t wakeTime = chVTGetSystemTime();

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

  while (!chThdShouldTerminateX()) {
    // Sleep until next second.
    wakeTime += TIME_MS2I(GPSThreadPeriod);
    chThdSleepUntil(wakeTime);
```

Données GPS:

```
//--------------GPS---------------------

float Offset_lat=43.2184361; //N+ 10^-7 ok précision
float Offset_lng=-0.04733333; //Ouest donc négatig
    while (Serial5.available() > 0) {
      if (gps.encode(Serial5.read())) {

        if (gps.location.isValid()) {

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

          LogGPS* GPS = &GPSFifo[fifoGPSHead];

          // get the byte data from the GPS
          GPS->teensy_time = millis();
          GPS->GPS_minute = gps.time.minute();
          GPS->GPS_second = gps.time.second();
          GPS->GPS_centisecond = gps.time.centisecond();
          GPS->GPS_lat = (gps.location.lat() - Offset_lat)*111120; //nombre de mètres dans 60
miles nautiques (1°)
          GPS->GPS_lng = (gps.location.lng() - Offset_lng)*111120; //nombre de mètres dans 60
miles nautiques (1°)
          GPS->GPS_age = gps.location.age();
          GPS->GPS_speed = gps.speed.kmph();
          GPS->GPS_speedValid = gps.speed.isValid();
          GPS->GPS_deg = gps.course.deg();
          GPS->GPS_degValid = gps.course.isValid();
          GPS->GPS_altitude = gps.altitude.meters();
          GPS->GPS_altitudeValid = gps.altitude.isValid();
          GPS->GPS_satellites = gps.satellites.value();
          GPS->GPS_satellitesValid = gps.satellites.isValid();
          GPS->GPS_hdop = gps.hdop.hdop();
          GPS->GPS_hdopValid = gps.hdop.isValid();

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


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

          chMtxLock(&dataMutex);

          vol_GPSlat = GPS->GPS_lat;
          vol_GPSlng = GPS->GPS_lng;

          // Unlock data access.
```

```
          chMtxUnlock(&dataMutex);
        }

      }

    }
```

Transfert de Données GPS entre Threads :

```
//--------------Declaration-----------------

    chMtxLock(&dataMutex);

    float GPS_lat = vol_GPSlat;
    float GPS_lng = vol_GPSlng;

    // Unlock data access.
    chMtxUnlock(&dataMutex);
```

Acquisition de l'État de la Fusée :

```
//----------Données Séq----------------
    char s =56;

    if (Serial1.available()) {
      s = Serial1.read();
      //Serial.println(s);
      Serial1.flush();
    }
```

Acquisition des capteurs MPU9250 et BME280 via I2C :

```
    //-----------------Données-----------------

    float p;

    float BME_temp;

    float altitude;

    bmp280.getMeasurements(BME_temp, p, altitude);

    imu.update(UPDATE_ACCEL | UPDATE_GYRO | UPDATE_COMPASS);

    int short Voltage_exp = analogRead(Vbatexp);
    int short Voltage_mot = analogRead(Vbatmot);
```

Conversion des données pour la télémétrie (dans l'expérience) :

```
//------------------------Envois télém-------------------
    if (LoRaCount == 0 && s != '7') {
      //temps
micro,Etat,Pression,GPS_lat,GPS_lng,Temp,gx,gy,gz,magx,magy,magz,ax,ay,az,q1,q2,q3,q1inty,intz,ro
dre_mot,sortie,sortie1,sortie2,entrée2,volt_mot,volt_exp
      //byte globalByteArray[54] = {start_of_frame};
      byte globalByteArray[60];
      int len = 0;

      byte end_of_frame1 = 0xD5;

      byte end_of_frame2 = 0xAE;

      byte end_of_frame3 = 0x77;
```

```
    byte end_of_frame4 = 0xBD;

    byte end_of_frame5 = 0x42;

    byte end_of_frame6 = 0xCD;

    convertToByteArray(t, &globalByteArray[len]);
    len += 4;

    convertToByteArray(p, &globalByteArray[len],1);
    len += 2;

    convertToByteArray((byte)s, &globalByteArray[len]);
    len += 1;

    convertToByteArray(GPS_lat, &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(GPS_lng, &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(com1bis, &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(E1, &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(Voltage_mot, &globalByteArray[len]);
    len += 2;

    convertToByteArray(corrGyrRawx, &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(corrGyrRawy, &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(corrGyrRawz, &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(Mxyz[0], &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(Mxyz[1], &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(Mxyz[2], &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(Axyz[0], &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(Axyz[1], &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(Axyz[2], &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(q[0], &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(q[1], &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(q[2], &globalByteArray[len], 1);
    len += 2;

    convertToByteArray(q[3], &globalByteArray[len], 1);
    len += 2;
```

```
      convertToByteArray(Voltage_exp, &globalByteArray[len]);
      len += 2;

      convertToByteArray(BME_temp, &globalByteArray[len],1);
      len += 2;

      convertToByteArray(end_of_frame1, &globalByteArray[len]);
      len += 1;

      convertToByteArray(end_of_frame2, &globalByteArray[len]);
      len += 1;

      convertToByteArray(end_of_frame3, &globalByteArray[len]);
      len += 1;

      convertToByteArray(end_of_frame4, &globalByteArray[len]);
      len += 1;

      convertToByteArray(end_of_frame5, &globalByteArray[len]);
      len += 1;

      convertToByteArray(end_of_frame6, &globalByteArray[len]);
      len += 1;



      Wire.beginTransmission(2); // transmit to device
      for(int k = 0; k < 32; k++){
      Wire.write(globalByteArray[k]);        // sends
      delayMicroseconds(1);
      }
      Wire.endTransmission();    // stop   transmitting
      Wire.beginTransmission(2); // transmit to device
      for(int k = 32; k < len; k++){
      Wire.write(globalByteArray[k]);        // sends
      delayMicroseconds(1);
      }
      Wire.endTransmission();    // stop   transmitting
    }
    LoRaCount = (LoRaCount+1)%2;
```

Setup de la carte télémesure :

```
void setup() {
  pinMode(PinLED, OUTPUT);

  Wire.begin(2);
  Wire.setClock(400000);
  Wire.onReceive(receiveEvent);

  Serial.begin(2000000); // Adjust baud rate as needed

  if (!LoRa.begin(869.5E6)) {
    while (1) {
      digitalWrite(PinLED, HIGH);
      delay(100);
      digitalWrite(PinLED, LOW);
      delay(100);
    }
  }

  LoRa.setFrequency(869.5E6);
  LoRa.setSpreadingFactor(SpreadingFactor);
  LoRa.setSignalBandwidth(250E3);
  LoRa.setCodingRate4(5);
  LoRa.setPreambleLength(8);
```

```
  LoRa.setTxPower(20, PA_OUTPUT_PA_BOOST_PIN);

  digitalWrite(PinLED, HIGH);
}
```

Programme de la télémesure :

```
void loop() {
  if (FrametoSend) {
    LoRa.beginPacket();
    LoRa.write(LoRaBuffer, LoraBufferSize);
    LoRa.endPacket();

    FrametoSend = false;
    dataLength = 0;
  }
}

void receiveEvent(int howMany) {
  for (int i = 0; i < howMany; i++) {
    byte1 = byte2;
    byte2 = byte3;
    byte3 = byte4;
    byte4 = byte5;
    byte5 = byte6;
    byte6 = Wire.read();
    data[dataLength] = byte6;
    dataLength++;

    if (byte6 == trame_end_of_frame6 && byte5 == trame_end_of_frame5 && byte4 ==
trame_end_of_frame4 && byte3 == trame_end_of_frame3 && byte2 == trame_end_of_frame2 && byte1 ==
trame_end_of_frame1) {
      for (int k = 0; k < dataLength-6; k++) {
        LoRaBuffer[k] = data[k];
    }
      LoraBufferSize = dataLength-6;
      FrametoSend = true;
      dataLength = 0;
    }
    else if (dataLength > 57) {
      dataLength = 0;
    }
  }
}
```