Rapport de vol

Mini-Fusée : MF57 TARANIS

Juillet 2021

INTRODUCTION

Le projet MF57 TARANIS de l'année 2019-2021, est un projet de mini-fusée expérimentale d'ELISA SPACE, une association d'ELISA AEROSPACE, école d'ingénieurs spécialisée dans le domaine de l'aéronautique et du spatial. L'association, ayant pour but de promouvoir le domaine spatial au sein de l'École, propose différents projets : micro-fusées, mini-fusées...

Ce projet a été créé en novembre 2019 par 6 membres et depuis, nous n'avons cessé de travailler sur ce projet durant ces deux dernières années. Notre mini-fusée a été lancée lors de la campagne de lancement du C'Space 2021 et a embarqué un ensemble de deux cartes électroniques programmées, lui permettant de déployer un parachute et d'effectuer des mesures en vol. Avant de pouvoir être lancée, toute fusée doit prouver qu'elle réalisera un vol nominal (récupération de la fusée intacte) et qu'elle embarquera des expériences d'intérêt scientifique.

N'ayant pas encore d'expérience sur les projets mini-fusées, notre groupe a décidé de récolter le maximum d'informations pendant le vol en embarquant plusieurs capteurs de mesures (accéléromètre, gyromètre, capteur de pression / température) afin, par la suite, de reconstituer le vol et de permettre de concevoir des projets plus aboutis.

La suite de ce rapport présente, tout d'abord, en détail le contexte de notre projet de l'année 2019-2021, puis le déroulé des tâches et enfin la réalisation de la mini-fusée en décrivant les problèmes rencontrés et les résultats obtenus.

Table des matières

Contexte du Projet TARANIS

Encadrement du Projet

Planète Sciences

Les membres du projet sont en contact avec l'association Planète Sciences qui s'occupe chaque année du suivi des projets étudiants semblables à travers la France. Ils sont également chargés de l'organisation du C'Space, campagne nationale de lancement des projets étudiants dans le domaine de l'espace, qui a pour objectif d'encourager l'intérêt et la pratique scientifique des jeunes. Le C'Space n'est que l'une des activités que Planètes Sciences propose. En effet, cette association est également impliquée dans l'organisation de la Nuit des étoiles ou des Trophées de Robotique. Au cours de ces deux ans, Planète Sciences a accompagné notre équipe en organisant 5 Rencontres de Clubs Espaces (RCE) afin de nous guider et de nous fournir les informations essentielles en vue de la campagne de lancement.

Figure 1 : Logo de l'association Planète Sciences

La campagne de lancement C'Space

Dans le but d'expérimenter nos travaux sur TARANIS, le projet a été représenté à une campagne de lancement nationale : le C'Space. Celui-ci s'est déroulé du 15 au 23 juillet 2021 à Tarbes. Organisé par Planète Sciences en partenariat avec le CNES, le C'Space constitue l'aboutissement de notre projet. TARANIS a été lancée et a effectué un vol nominal, composé d'un décollage durant lequel la fusée est propulsée, suivi du déploiement du parachute, au cours duquel la mini-fusée est freinée jusqu'à 36 km/h en attendant de retomber au sol.

Le Centre National d'Etudes Spatiales (CNES)

Le CNES est coorganisateur du C'Space avec Planètes Sciences. Agence spatiale l'une des plus importantes de l'Union européenne par l'ampleur de ses financements, cet établissement public est chargé du programme spatial français. Dans notre cas, il fournit les propulseurs à poudre nécessaires au lancement des fusées pour les étudiants, restant très dangereuses à manipuler.

Figure 2 : Logo du CNES

Equipe du Projet

Le projet mené par six étudiants d'ELISA AEROSPACE au cours des deux années scolaires, a été découpé en trois domaines techniques afin que ses membres puissent se répartir le travail. Après avoir nommé Alexandre LELEUX responsable de projet, les six participants se sont répartis dans les différents pôles comme suit :

- Programmation : Léo-Paul LAURENT, Melvin SEAUVE
- Electronique : Florian LEVRAY
- Mécanique : Marie BRUN, Florian LEVRAY
- Rédaction : Corentin COPET

Objectifs du Projet

Position du problème

Outre sa conception, nous devons faire en sorte que la mini-fusée effectue un vol nominal, c'est-à-dire un vol qui se termine par la récupération intacte de la fusée, par opposition à un vol balistique. Le vol de la fusée présente différentes phases :

- **La propulsion :** elle débute à partir de la mise à feu du propulseur lorsque la fusée se trouve encore sur la rampe de lancement jusqu'au moment où tout le carburant a été consommé
- **La phase balistique :** au cours de cette étape, qui succède immédiatement la phase précédente, la fusée s'apparente à un corps libre soumis uniquement à son poids et aux forces de frottement

- **La récupération :** après avoir atteint l'apogée, le parachute se déploie pour faire atterrir en douceur la mini-fusée. Sa vitesse diminue alors fortement

La figure 3 représente les différentes phases du vol de la mini-fusée.

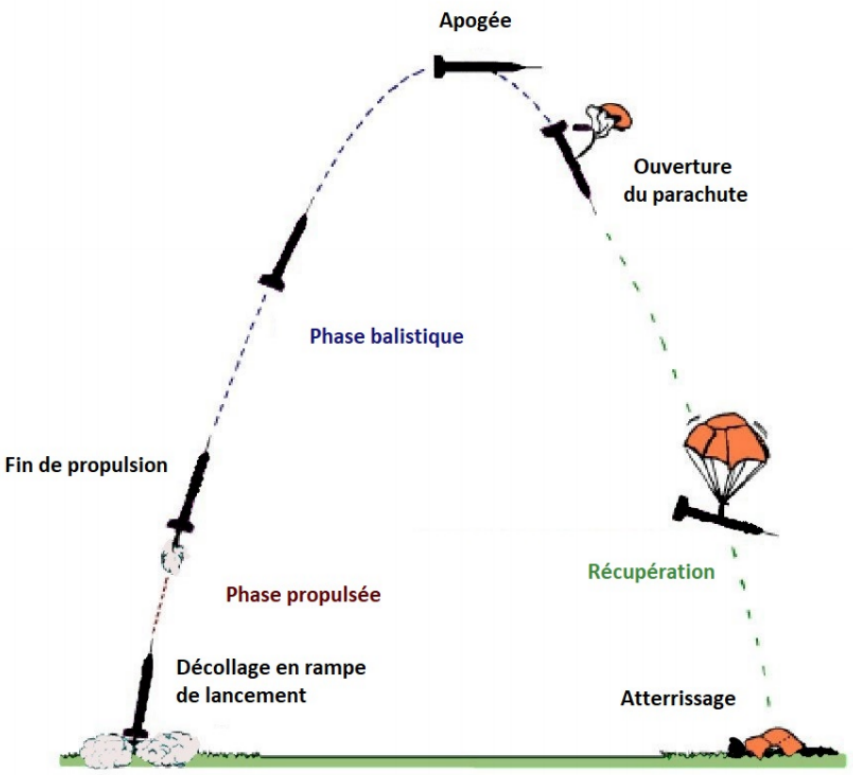


Figure 3 : Les différentes phases de vol de la mini-fusée

Expériences

Le projet TARANIS, étant qualifié de « mini-fusée », doit au moins permettre de réaliser une expérience à bord. Il a été décidé de réaliser plusieurs expériences pour le projet, qui feront office de charge utile pour le vol :

- Connecter la carte expérience et la carte séquenceur via des optocoupleurs
- Enregistrer les accélérations et la vitesse angulaire que subira la fusée durant son vol ainsi que la pression extérieure et la température
- Ouvrir le parachute automatiquement à l'aide du capteur de pression ou d'un minuteur si un certain délai est dépassé
- Connaître l'état de la fusée en tout temps (avant décollage, ascension, début de plage d'ouverture, ouverture parachute par détection d'apogée, ouverture parachute par minuterie) et enregistrer celui-ci
- Filmer le vol à l'aide de deux caméras embarquées : l'une regardant au loin et vers l'horizon, l'autre regardant le sol et la trappe d'ouverture du parachute
- Enregistrer tout cela dans une carte SD embarqué qui servira de boîte noire
- Transmettre ces données en direct vers le sol à l'aide d'une télémétrie longue portée
- Faire un live C'Space et expliquer les données exploitées de la télémétrie

L'idée derrière ces expériences est, une fois leur fonctionnement prouvé et validé, de les installer de manière récurrente sur les futurs projets plus évolués d'ELISA Space, afin d'augmenter chaque année le niveau de complexité mais également les performances des projets mis en œuvre.

Planification des tâches

Gestion de projet

La gestion de projet doit être un effort continu sur toute la durée du projet. En effet, la répartition de notre équipe en pôles a pour conséquence directe une perte de visibilité sur l'avancement réel du projet. De plus, les pôles sont interdépendants, ce qui implique que tout retard sur un d'entre eux entraîne un retard cumulable sur les autres. Ainsi, dès le début de notre projet, il a été soigneusement identifié quelles tâches devaient être réalisées prioritairement. Le déroulement du projet a été programmé et les objectifs à court et long termes fixés.

Rencontres Club Espace (RCE)

Le projet a participé aux deux RCE de 2019-2020 puis aux trois RCE de 2020-2021 organisées par Planète Sciences.

La RCE 1 et la RCE 2 de 2019-2020 a permis une prise de contact entre le projet et les bénévoles de Planète Sciences; des idées ont été proposées.

La RCE 1 de 2020-2021 fut une revue de tout le design et la RCE 2, un bilan sur la validation du projet. La RCE 3 fut la pré-qualification (validation de la stabilité du projet et test du système de récupération). Il nous a été demandé de détacher la trappe en haut du parachute pour la mettre sur l'émerillon, situé entre la sangle et les suspentes. Le but étant d'éviter que la trappe tombe sur le parachute et ne s'emmêle dedans. Un interrupteur pour la télémétrie a été ajouté, ainsi qu'un changement de puissance du transmetteur afin d'avoir moins de 25 mW (14 dB) de puissance.

StabTraj et la propulsion

La stabilité de la mini-fusée lors de la phase de vol et notamment du décollage est calculée grâce à une page Excel, nommée StabTraj, fournie par le CNES. En entrant dans la page l'ensemble des dimensions de la fusée ainsi que les coordonnées de son centre de masse, la feuille de calcul confirme si la fusée est stable ou non. La stabilité de la fusée peut être contrôlée par la coiffe, les ailerons et le centre de masse. Elle peut être ajustée en jouant sur les dimensions des quatre ailerons. Des captures du Stabtraj sont présentées à l'Annexe 1.

La propulsion de la fusée est assurée par le propulseur à poudre de type Pandora (Pro-24) fourni par le CNES pour la campagne du C'Space, ses dimensions ont été normalisées. On retrouve les dimensions suivantes :

- 0.1599 kg de masse totale
- 228 mm de longueur totale
- 24 mm de diamètre nominal
- 142.4 N.s d'impulsion totale

En matière de performance, on peut retrouver la courbe de poussée sur le site du CNES.

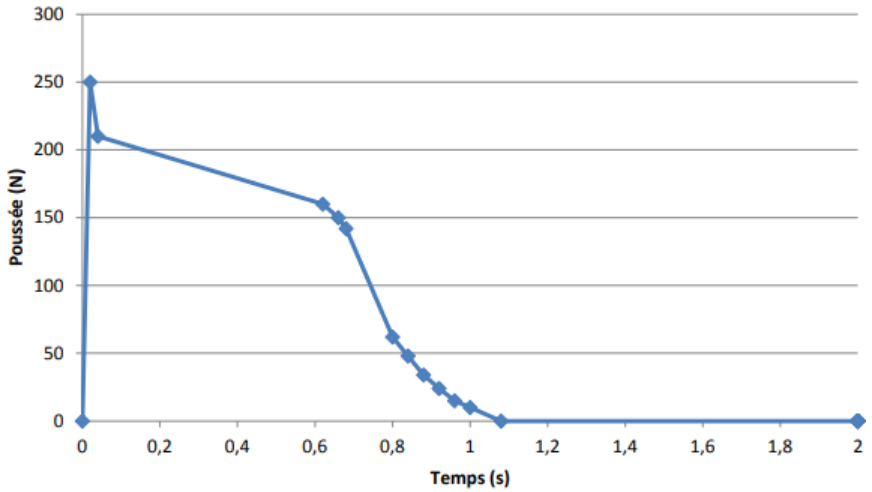


Figure 4 : Courbe de poussée du propulseur Pandora

On remarque une poussée maximum de 250 N et une poussée moyenne d'environ 100 N pendant plus d'une seconde, ce qui est un bon compromis pour un propulseur de cette taille.

Réalisation du projet

Conception de l'électronique

Afin de déclencher le lancement du parachute et d'assurer diverses fonctions de la mini-fusée, à savoir communiquer avec le sol et effectuer des mesures en vol, il est nécessaire de concevoir et réaliser des cartes électroniques. Mais lorsque nous avons commencé ce projet, nous n'avions jamais touché à une Arduino et encore moins fait clignoter une LED. Ainsi, la première année s'est résumée à essayer de faire fonctionner de manière sûre tout le système électronique de la fusée.

Très vite, nous sommes passés sur l'Arduino mini car c'est la seule carte qui restait dans notre matériel et qui pouvait entrer dans la case d'équipement. En février 2020, une ébauche des premiers programmes a été réalisée. La figure 5 représente le câblage complet réalisé sur Fritzing. Le montage comporte une carte séquenceur, qui détecte le départ et ouvre le parachute et une carte expérience qui enregistre les données, les transmet via la télémétrie et communique avec le séquenceur pour ouvrir le parachute au bon moment. Ces cartes sont toutes deux alimentées par deux piles de 9V.

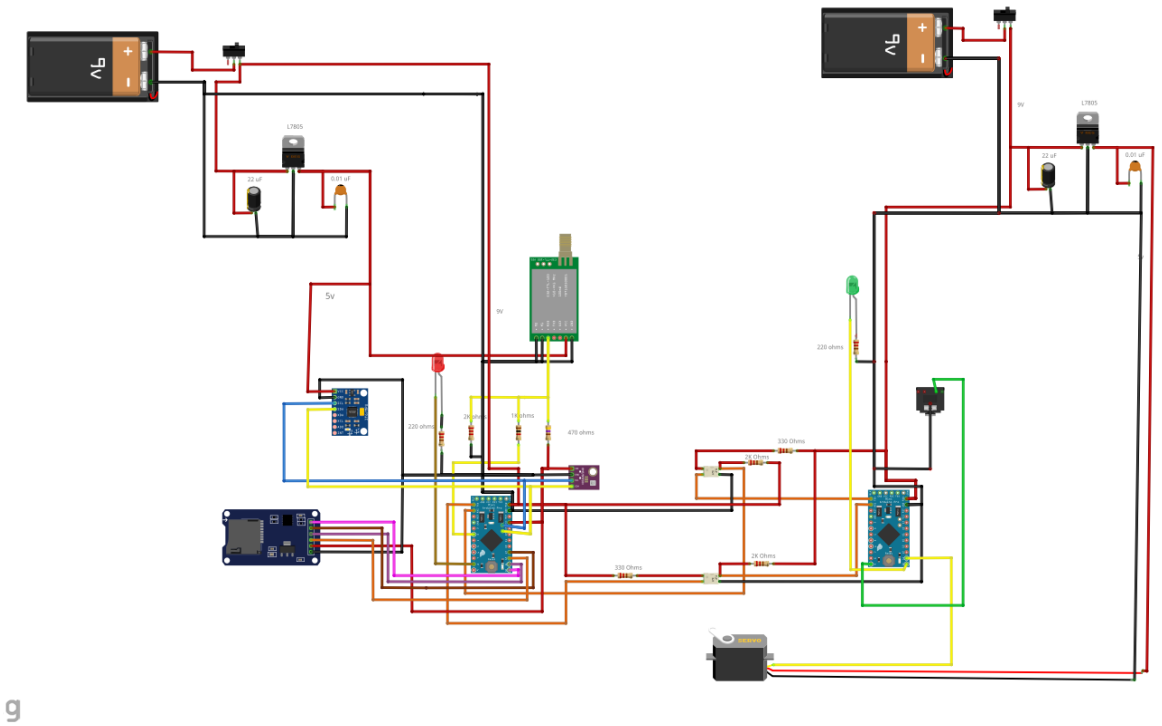


Figure 5 : Plan de câblage sur Fritzing

La carte expérience

Carte télémétrie émettrice

La télémesure est un aspect important de l'expérience, menée à bord de la fusée, puisque nous voulons suivre la trajectoire de la fusée en temps réel. Pour gérer la communication à longue portée, une carte émettrice (Lora Ebyte E32 868Mhz avec une puissance maximale de 20 dBm, Figure 6) dont l'objectif est d'envoyer toutes les données à une carte réceptrice positionnée. Les cartes réceptrices au sol doivent toutes les deux être programmées. La maîtrise de la communication longue portée est d'autant plus importante que l'enregistrement des données durant la phase balistique. Si ce n'est pas le cas, aucune reconstitution de la trajectoire de la fusée ne pourrait être possible si cette partie de la programmation n'est pas fonctionnelle.

La communication des mesures utilise la technologie LoRa qui transmet à une fréquence de 869 MHz avec une portée de 1.2 km. L'émetteur est relié au microprocesseur via une communication Serial et les fonctionnalités de la librairie C++ (voir Annexe 2 librairie) sont utilisées. La carte dispose d'une longue antenne enfermée dans la coiffe de la fusée pour pouvoir émettre convenablement.

En effet l'aluminium ne permet pas le passage des ondes radios (tout comme le carbone) donc l'emplacement de l'antenne devait être fait dans le sommet de la fusée (la coiffe étant imprimée en PLA).

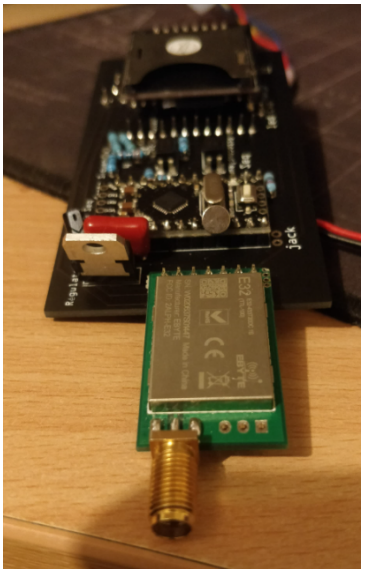


Figure 6 : Carte émettrice

Carte télémétrie réceptrice

La carte réceptrice (Figure 7) reste au sol, attachée à un ordinateur portable et couplée à un logiciel adapté. Les données sont récupérées en Serial via le logiciel CoolTerm. Son objectif est de saisir toutes les données envoyées par la carte de télémétrie émettrice et de les communiquer à l'ordinateur. La carte réceptrice doit être configurée pour avoir la même

fréquence que la carte émettrice. On lui donne l'adresse dans lequel on est censé lui répondre. Elle convertit les données en caractère qui sont ensuite transmises au module. Ce dernier reçoit la transmission, la stocke et l'affiche via la console Serial. Le câblage du module a été très difficile à configurer mais possède une portée importante. Son code est représenté en Annexe 5.

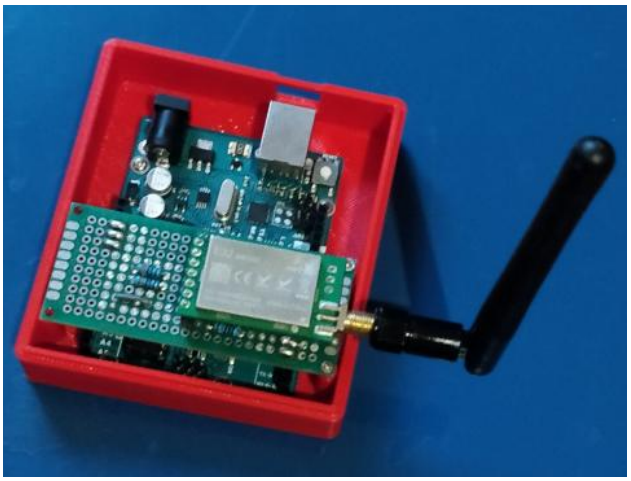


Figure 7 : Carte réceptrice

Le gyromètre

Le MPU-6050 (Figure 8), contient à la fois un gyromètre trois axes et un accéléromètre permettant des mesures indépendamment l'un de l'autre, mais toutes basées sur les mêmes axes. La configuration de tous les offset doit être réalisée à partir d'un programme de calibration spécial à l'horizontale dans un module imprimé en 3D.

Figure 8 : Optocoupleur

Dans notre projet, un MPU-6050 a été utilisé. Celui-ci possède une plage de 16 G d'accélération dans tous les sens et d'une plage de 2000 deg/s pour les vitesses angulaires.

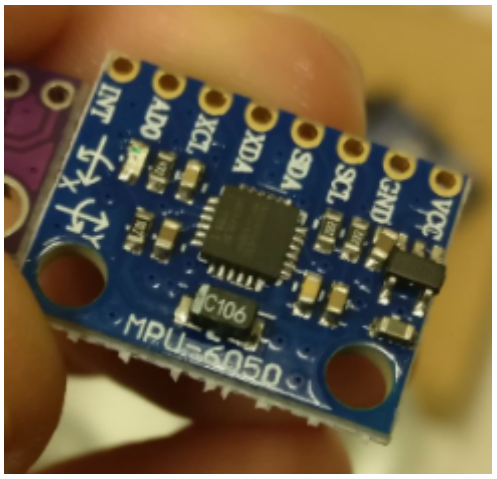


Figure 9 : MPU-6050

La communication entre la carte Arduino et le MPU-6050 se fait en protocole I2C, utilisable par le port Arduino sur les ports SCL (Serial Clock Line) et SDA (Serial Data Line) grâce à la bibliothèque I2C dev (Inter-Integrated Circuit MPU-6050). On accède au registre de réglages du MPU avec la bibliothèque Wire. On récupère, à chaque instant, dans les registres les informations suivantes :

- Vitesses angulaires [-32768, +32767]
- Accélérations [-32768, +32767]

Les données reçues sont sous forme d'entiers sur 16 bits, qu'il faut convertir en $m.s^{-2}$ pour les accélérations et en °. s^{-1} pour les vitesses angulaires.

On récupère le facteur par lequel il faut diviser nos valeurs obtenues dans la datasheet, marquées en LSB/unité. Ces valeurs dépendent des réglages de sensibilités du gyroscope et de l'accéléromètre que l'on a envoyé dans les registres.

La carte SD

Il s'agit d'un système de stockage de données flash, c'est-à-dire qu'elle est composée de mémoire NAND qui a l'avantage d'être très rapide en écriture. Nous souhaitons récupérer nos données par une carte Arduino qui sera stockée sur une carte SD (Figure 10). Dès l'allumage de la carte, les données doivent comprendre l'état de la fusée (non initialisée, actionneur para, en attente décollage, vol actif para…).

Figure 10 : Carte SD

Algorithme de déclenchement du parachute à l'apogée

La conception de la mini-fusée Taranis demandant une ouverture de la trappe du parachute au plus proche de l'apogée, il était nécessaire de concevoir un algorithme pouvant détecter celle-ci. L'ouverture n'étant pas instantanée, il était d'autant plus intéressant de prédire l'apogée pour ouvrir la trappe en amont. De plus, il était impératif que l'ouverture se fasse entre 80% (T1) et 120% (T2) du T0, T0 étant l'apogée théorique déterminer AVANT le vol.

Les données qui seront utiles au programme sont :

- Le temps au décollage (en seconde)
- Chaque donnée de l'altimètre (en Pascal)
- Le temps à chaque données de pression (en seconde)

Une des inconnues que nous avons déterminées est la précision de l'altimètre.

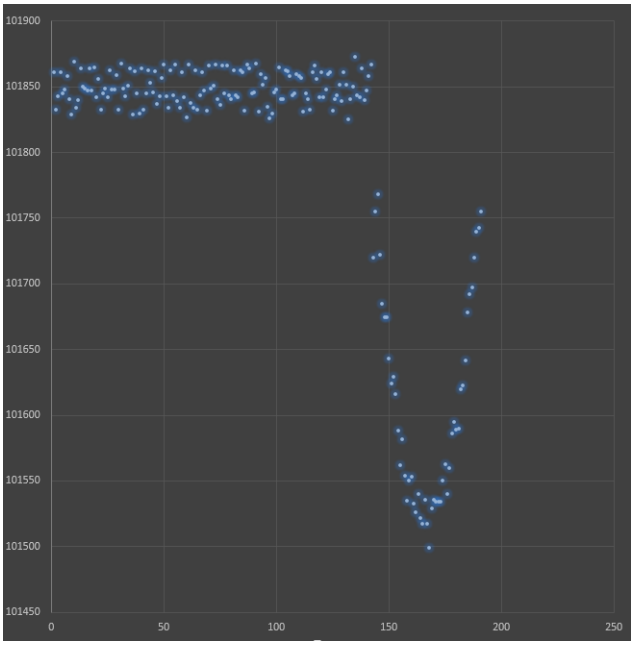


Figure 11 : Simulation de la pression statique en fonction du temps

Ce qui montre une précision d'environ 2.76 mètres.
En simulant sur MATLAB le vol de la fusée, on voit que cette erreur prend peu d'importance.

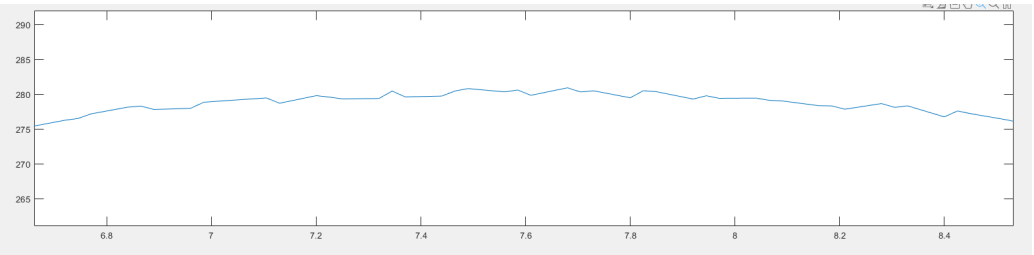


Figure 12 : Simulation du vol de la fusée

Le cerveau de notre fusée était une carte arduino, or celle-ci a une capacité de calcul relativement limitée ce qui rend impossible l'implémentation d'algorithmes lourds pour l'ouverture de la trappe.

La solution mise en place est la suivante : à chaque donnée, on stocke celle-ci dans un tableau de 5 éléments, ce tableau est une sauvegarde des 5 dernières données de pression. On calcule la moyenne, ce qui permet de quasiment effacer l'erreur du capteur, la courbe des points moyenne étant bien plus proche d'une parabole que celle des données brut (surtout à l'apogée).

Pour savoir quand ouvrir, on regarde à chaque calcul de moyenne si celui-ci est inférieur à un seuil. Ce seuil, déterminé avant le vol, est calibré pour être suffisamment grand pour que notre moyenne passe à un moment sous ce seuil et assez petit pour que l'on ne déclenche pas trop tôt.

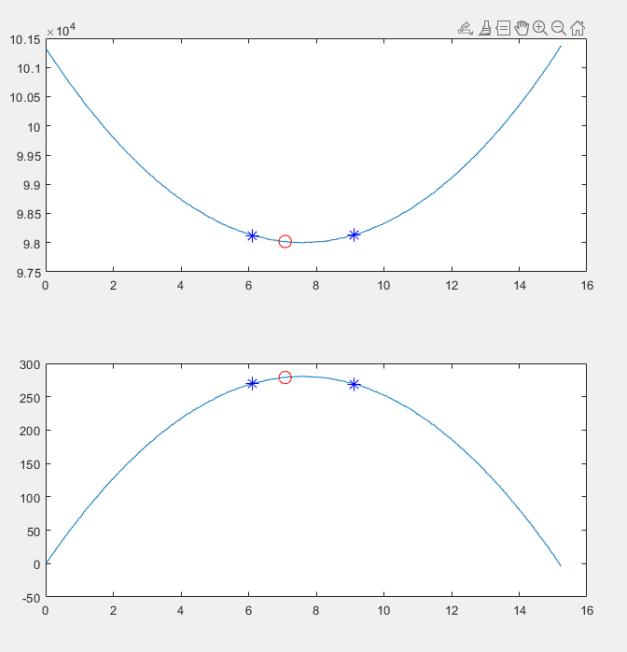


Figure 13 : Graphique du déclenchement de parachute

On voit ici en bleu les bornes T1 et T2 entre lesquelles on doit ouvrir la trappe, le point rouge est l'ouverture par l'algorithme avec un seuil de 10.

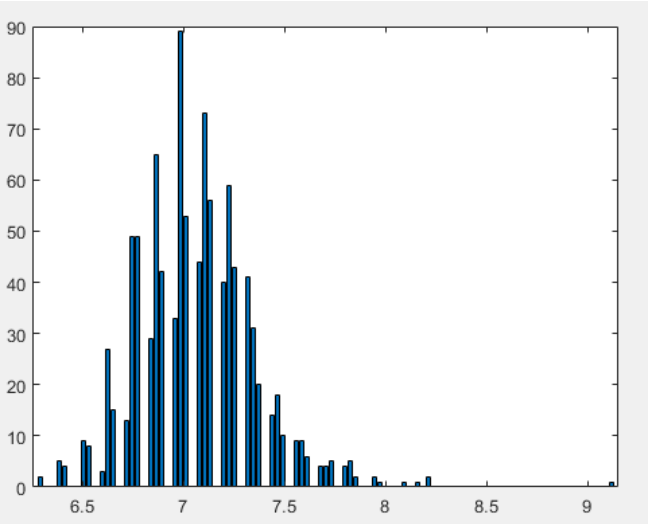


Figure 14 : Diagramme du temps de déclenchement

On voit ici le temps de déclenchement de 1000 vol simulé toujours avec un seuil de 10. Ce seuil est celui qui a été choisi pour le vol de Taranis.

Si l'on souhaite améliorer ce programme pour un nouveau vol de mini fusée ou un vol de Fusex, il y a plusieurs possibilités :

- Affiner le programme actuel avec un seuil plus précis ou modifier la moyenne, soit en y ajoutant des valeurs soit en utilisant une moyenne pondérée. Utile si vous disposez de peu de puissance de calcul.
- Dans le cas d'une trajectoire parabolique, déterminer pendant la montée de la fusée, les paramètres a, b et c d'une parabole, si possible avoir un algorithme qui se corrige à chaque nouvelle donnée.
- Similaire au cas précèdent, utiliser une régression polynomiale, permettant de faire correspondre un nuage de point avec une courbe (ici une courbe du second degré). Il y a deux étapes : stocker toutes les données de temps et de pression jusqu'au moment où l'on veut prédire la courbe, ensuite appliquer à ces données un algorithme de régression polynomiale. Attention, cet algorithme est lourd mais ne nécessite qu'une seule exécution, il faut donc avoir un peu de puissance de calcul (faire des tests pour voir si votre carte peut le supporter), et faire ce calcul suffisamment tôt pour que le calcul soit fini avant l'apogée et assez tard pour avoir assez de données (encore une fois, faites des tests). Il est d'ailleurs possible d'optimiser en ne prenant qu'une valeur sur deux lors de la montée. En effet, le nombre de données influe sur la durée de calcul et l'écart entre la première valeur et la dernière influe sur la précision. Cette dernière solution semble être la meilleure, l'image qui suit est une simulation de 1000 vols avec cette méthode en ne prenant les données que sur le premier 5ème du trajet entre le décollage et l'apogée.

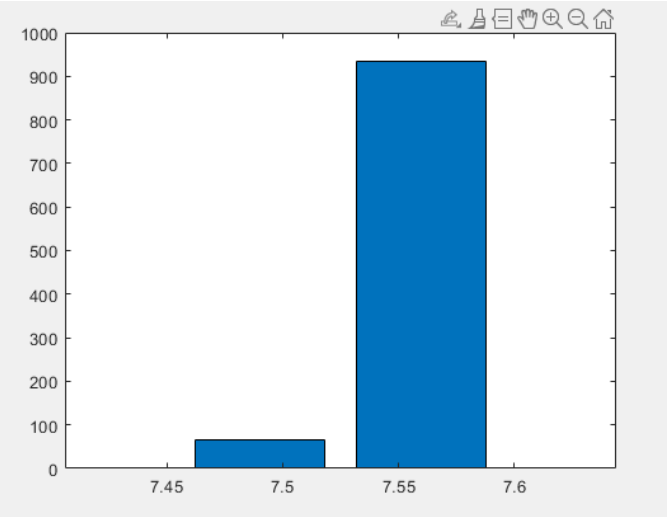


Figure 15 : Simulation de 1000 vols

Interprétation des données

Les données du gyromètre, de l'accéléromètre et de la température seront affichées en données brutes. Un programme a été créé pour convertir ces données brutes en données exploitables qui seront lues sur un fichier Excel. La mesure de la température et de la pression a été mesurée à partir d'un capteur BME280 (Figure 16). Son code est représenté en Annexe 4. Le BME est disposé dans une case en dessous de l'électronique, dans lequel il sera en contact avec l'atmosphère.

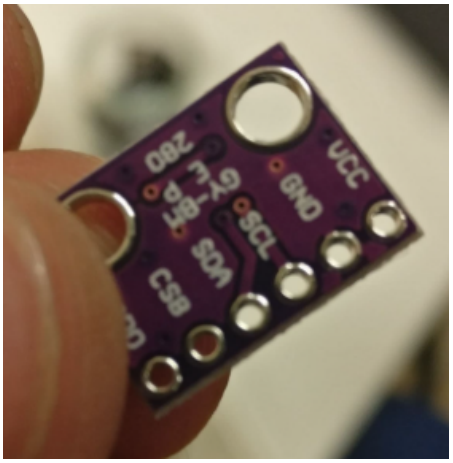


Figure 16 : Capteur BME 280

La carte séquenceur

Le but de la carte 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 de la carte séquenceur doit servir à prendre en compte et enclencher chacune des différentes étapes du vol. On y retrouve notamment la commande qui ouvre le parachute et envoie le signal de bonne réception à la carte expérience. Son code est représenté en Annexe 3.

Le microcontrôleur a une fonction principale de séquenceur. Il doit compter le temps à partir du décollage et déclencher l'ouverture du parachute au bon moment. 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.

Deux cas peuvent apparaître :

- Soit l'apogée est détectée grâce à notre expérience, à un instant T et que l'instant T + 1s se trouve dans la fenêtre temporelle : alors le parachute s'ouvrira une seconde après réception du signal d'apogée
- Soit ce n'est pas le cas et le parachute s'ouvrira automatiquement à la fin de la fenêtre temporelle

Gestion de la sécurité du vol

Commande des servomoteurs

L'ouverture du parachute passe par l'action d'un servomoteur, situé au niveau de la case parachute et son action va permettre l'ouverture de la porte de cette case contenant le parachute. Le signal de contrôle des servomoteurs est un signal PWM, géré par les microcontrôleurs.

Un servomoteur a été choisi d'être utilisé plutôt que des moteurs à courant continu. En effet, avec un servomoteur, le signal de commande indique immédiatement l'angle que le servomoteur va tenter d'atteindre, alors que pour obtenir le même résultat avec des moteurs à courant continu, il aurait fallu déterminer le temps nécessaire au moteur pour effectuer la rotation voulue et l'alimenter pendant cette durée. En cas de blocage, la rotation risquerait de s'arrêter avant d'avoir atteint l'angle voulu. A l'inverse, le servomoteur va continuer à travailler tant que l'angle correspondant à la commande n'est pas obtenu.

Prise jack de déclenchement

Le signal de décollage est engendré en utilisant une prise jack (Figure 17). 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 les deux microcontrôleurs séquenceurs qui commenceront alors leur comptage. Ce signal de déclenchement sera réceptionné par le microcontrôleur mère. Le système jack de déclenchement est situé en bas de la case électronique.

Figure 17 : Prise jack mâle

La détection du départ se fait à l'aide d'un pull-up sur un des pin de la carte séquenceur. Lorsque le jack mâle est branché le pull-up reste chargé à 5V et dès que le jack mâle s'en va le système se referme et le pull-up se décharge dans le ground. C'est à l'aide de cette variation d'état du pull-up que l'on détecte le départ.

Figure 18 : Système du pull-up

Isolation électrique du séquenceur parachute : utilisation d'optocoupleurs

D'après le cahier des charges, le séquenceur parachute doit être isolé électriquement du reste de la fusée, au cas où il y aurait un problème dans le reste de l'électronique, afin que le parachute se déploie quand même. Pour ce faire, des optocoupleurs / photocoupleurs (Figure 19) sont utilisés, composants électroniques qui permettent de communiquer une tension entre l'entrée et la sortie sans qu'il y ait de contact électrique. Un optocoupleur combine une diode électroluminescente, qui émet simplement de la lumière infrarouge, et un phototransistor qui est une variante d'un transistor. Ce dernier ne laisse passer du courant que s'il est éclairé par la diode. Ainsi, il permet de transmettre un signal sans aucune liaison électrique. Nous en mettons sur toutes les pistes servant à la communication Serial entre la carte séquenceur et la carte parachute. Après plusieurs essais, la vitesse de transmission maximale qu'on a réussi à atteindre avec un optocoupleur est de 19200 bauds.

Figure 19 : Optocoupleur

La connexion Serial

La communication entre la carte séquenceur et la carte expérience est assurée par la connexion Serial à l'aide de phototransistors, cela veut aussi dire que chaque système est indépendant et utilise sa propre batterie tout en étant capable de communiquer. Il est également possible de communiquer entre deux Arduinos, en connectant leur pin TX et RX entre eux. Le pin TX permet de transmettre le signal et le pin RX reçoit le signal. On n'a pas besoin de connecter tous les ports entre eux. En effet, le signal doit être envoyé par le séquenceur à la carte expérience et non l'inverse.

Réalisation des circuits

Un circuit imprimé (ou PCB en anglais), composé de plusieurs couches de cuivre recouvertes d'un matériau isolant, est un circuit électrique dont les composants électroniques sont intégrés à une structure mécanique. Les circuits électroniques ont été tout d'abord intégrés sur une perfboard, mais la difficulté de souder et la nécessité d'avoir un système plus ouvert nous a fait changer d'avis. Nous avons donc conçu le circuit sur Fritzing (Figure 20 et 21), exporté le dessin du circuit et de ses perçages sur un fichier Gerber, puis préparé l'usinage avec FlatCam. L'utilisation de tulipes aurait pu être envisagée pour pouvoir débrancher et rebrancher plus facilement. La vue du dessus du PCB est représentée en figure 22.

Les ports SDA et SCL ont été accidentellement inversés, donc nous avons ajouté des fils pour résoudre le problème.

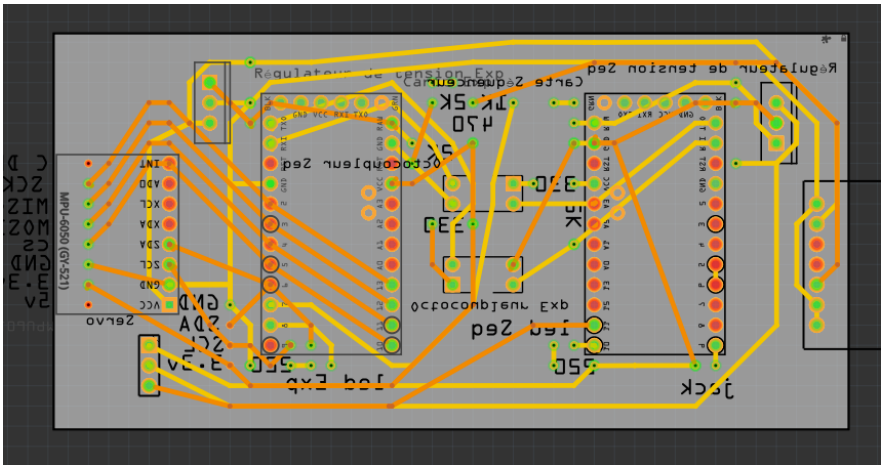


Figure 20 : Modélisation du PCB sur Fritzing (vue du dessous)

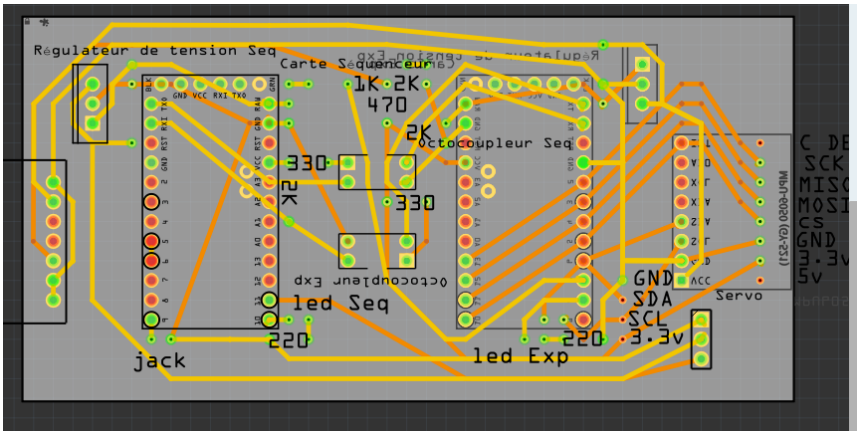


Figure 21 : Modélisation du PCB sur Fritzing (vue du dessus)

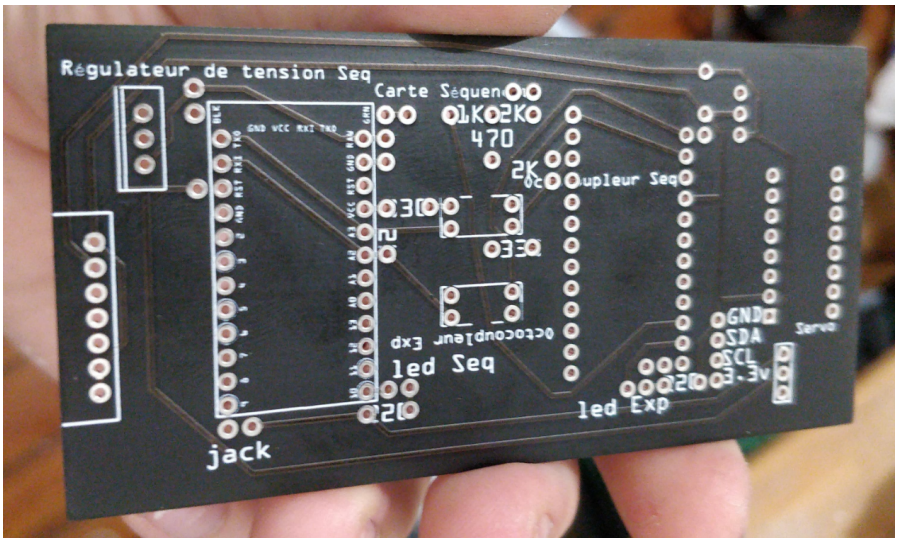


Figure 22 : PCB vue du dessus

Les caméras

Deux caméras ont été utilisées pour pouvoir filmer l'ouverture du parachute et l'horizon (vérification des données de l'accéléromètre). Elles ont été positionnées en haut du corps de la fusée, à l'intérieur de la coiffe. Pour effectuer ces prises de vue, les caméras utilisées sont des DV SQ11 avec micro intégré, d'une autonomie de 1H environ. Chacune d'entre elles a une carte micro SD, cela nous a permis de visualiser les données après le vol. Nous avons fait en sorte qu'en cas de crash, si la carte SD se déloge du système d'enregistrement, celui-ci ne corrompait pas le système vidéo. Etant donné que ces caméras enregistrent par période de 10 min, il va donc falloir chronométrer le lancement pour pouvoir tirer au milieu de ces plages.

Les cartes SD sont collées aux caméras et sont très difficiles à démonter : six cartes ont été cassées.

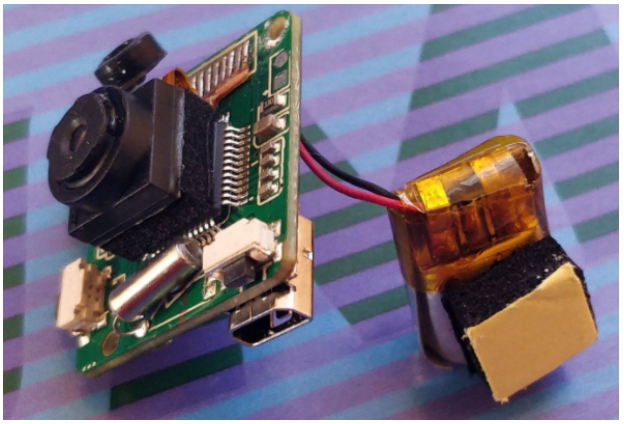


Figure 23 : Caméra DV SQ11

Intégration de l'électronique

Le module cartes

L'expérience embarquée n'est ni plus ni moins la raison d'être de la fusée. Lancer une fusée vide n'ayant pas d'expérience n'a aucune utilité et ne justifie pas l'investissement qui a été fait pour la réaliser. L'expérience embarquée dans notre fusée consistait à mesurer de différentes manières les paramètres de vol, à savoir les accélérations et la vitesse angulaire que subira la fusée durant son vol ainsi que la pression extérieure et la température.
La carte électronique (Figure 24), par souci de simplicité, a été placée verticalement dans la fusée. La fixation, en plus de devoir être fiable dans les conditions du vol (forte accélération, vibrations), devait être facile et rapide à monter et à démonter, afin de pouvoir facilement intervenir sur la carte, pour la reprogrammer par exemple.

Figure 24 : Module électronique

La coiffe

La coiffe de la fusée, en forme d'ogive, a été réalisée en PLA, de diamètre extérieur 6 cm et de diamètre intérieur 5.5 cm. Comme on peut le voir dans la figure 25 et 26, les caméras sont intégrées dans le module coiffe.

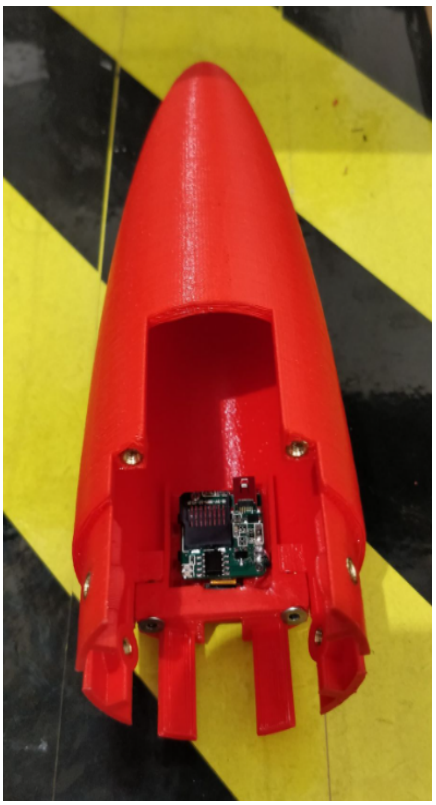


Figure 25: Coiffe + Emplacement d'une caméra

Figure 26 : Coiffe complète

Partie mécanique

La structure de la fusée doit être conçue de façon à atteindre un double objectif : accueillir et permettre la mise en œuvre des expériences choisies et respecter les conditions du cahier des charges de Planètes Sciences pour la sécurité des participants à la campagne. La conception et la fabrication de la structure sont explicitées par la suite. Les différentes parties de la fusée ont été réalisées sur CATIA. La partie construction a commencé très tard (mars 2021) car l'idée était d'avoir l'électronique avant d'avoir tout le reste. L'impression en 3D a été envisagée car les pièces imprimées sont solides et plus pratiques à assembler.

Structure générale

La fusée est divisée en trois sous-modules :

- Le module propulseur (au niveau des ailerons)
- Le corps de la fusée
- Le module parachute

Le module propulseur

Le module propulseur (Figure 27) a nécessité beaucoup d'impressions pour rendre compatible l'installation du moteur. Son intégration à l'intérieur de la mini-fusée se fait à l'aide d'une bague de poussée qui tient la base du moteur et les ailerons par l'arrière ; et de deux bagues de centrage qui d'une part fixe le haut des ailerons et de l'autre part, cale le moteur.

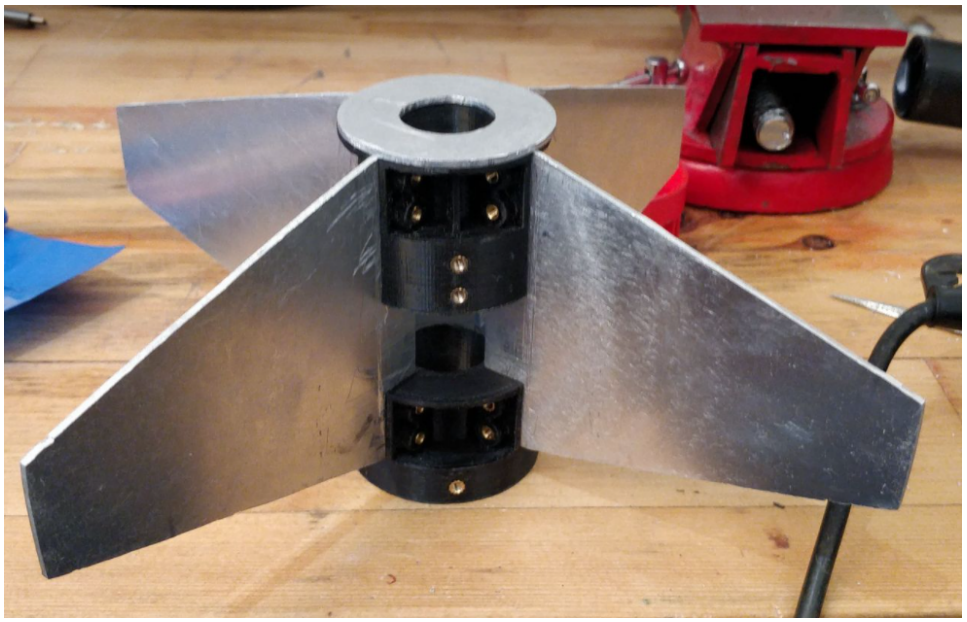


Figure 27 : Module propulseur avec les bagues de rétention

Ailerons

Le rôle des ailerons (Figure 28) est de stabiliser la fusée au cours de son vol, afin de l'empêcher de tourner sur elle-même, ce qui réduirait ses performances et de dévier de sa trajectoire, ce qui pourrait être dangereux. Des ailerons bien dimensionnés permettent d'assurer le retour à la position d'équilibre de la fusée en cas de déviation. Dans le cas contraire, deux types de comportement peuvent survenir : une fusée instable (ailerons trop petits) se mettra en rotation, réalisant des « loopings » avant de s'écraser. Une fusée sur stable (ailerons trop grands) oscille en permanence autour de sa position d'équilibre. Une fois les ailerons réalisés en aluminium, nous les avons poncés, afin de les rendre le plus lisse possible pour une meilleure intégration dans l'air et donc une meilleure qualité de vol.

Figure 28 : Ailerons

Le corps de la fusée

Le corps représente la plus grande surface en contact avec l'extérieur de la mini-fusée. Elle remplit deux rôles, elle protège les éléments intérieurs de la fusée, et dans le même temps elle remplit le rôle de squelette externe. L'usinage du tube principal est composé d'aluminium léger de 60 mm d'épaisseur extérieur et de 25 mm d'épaisseur intérieur. Le tube a été usiné à la main à l'aide d'une Dremel de lime et de beaucoup de patience. Des trous dans le tube ont été réalisés avec une perceuse. Des vis M3 ont été choisies car elles ont un bon compromis entre la taille et la résistance. L'utilisation de chanfrein à servi à cacher ces vis à l'intérieur du corps de la fusée. L'avantage d'avoir un tube aussi épais c'est qu'il n'y a aucun problème de flexion.

Figure 29 : Aperçu général de la fusée

Le parachute

Le parachute (Figure 30) constitue le système de récupération de notre fusée. Il est éjecté avant 7 m de l'apogée (fixé d'après StabTraj à 293 m), puis assure une descente contrôlée d'environ 17 m/s jusqu'à l'atterrissage de la fusée. Nous avons fabriqué un parachute cruciforme formé de 5 carrés pour ralentir la chute de la fusée. Le tissu utilisé est du nylon (tissu dont la résistance au choc est très élevée). Des bagues coulissant le long du parachute ont été imprimées pour empêcher les câbles de s'entremêler.

Figure 30 : Parachute déplié

La conception de ce parachute est relativement simple, il s'agit de deux tissus rectangulaires superposés qu'on a au préalable fermé le contour avec une bande rouge pour assurer une meilleure solidité et éviter que le tissu principal s'effiloche. C'est également dans cette bande que les câbles du parachute passent.

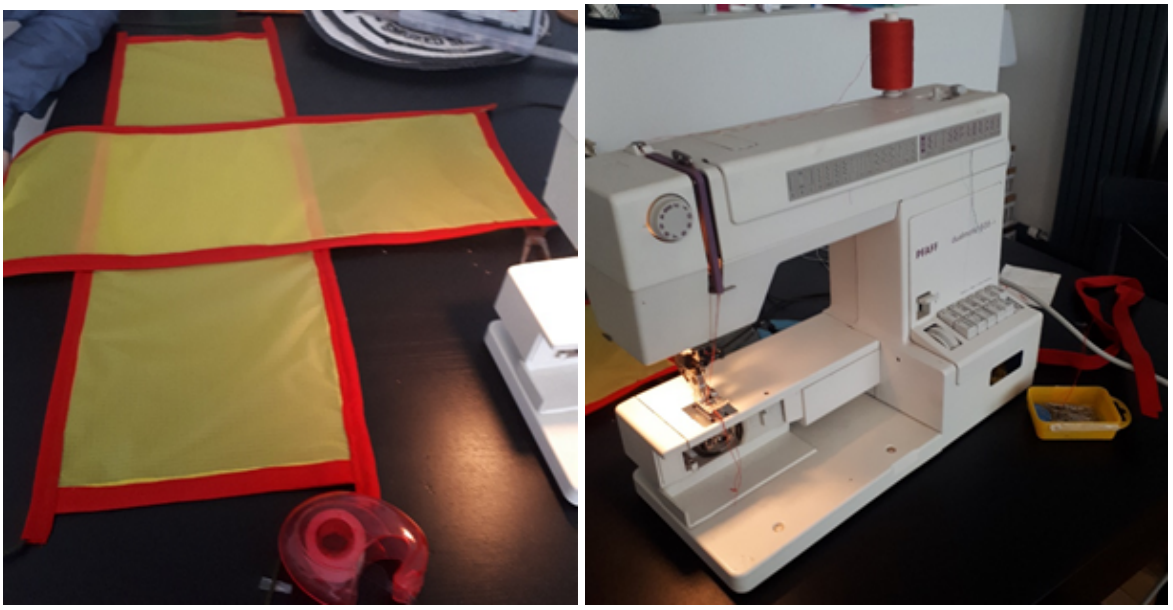


Figure 31-32 : Conception du parachute

Une fois les tissus positionnés, on les coud en utilisant le point zigzag de la machine à coudre. On choisit ce point ci parce que c'est celui qui présente le plus de résistance.

La peinture

La fusée porte le nom TARANIS, dieu gaulois du ciel et des éclairs. La peinture a été choisie volontairement colorée (rouge pour la coiffe, blanc pour le nom) afin de la voir plus facilement dans le ciel et au sol dans l'herbe. Le style graphique est basé sur la fusée expérimentale BELENOS, projet lancé par des étudiants de notre école en 2007.

Figure 33 : Utilisation de bombes pour inscrire le nom de la fusée

Figure 34 : Boîtier de la carte réceptrice

Campagne de lancement

Afin de respecter les consignes sanitaires demandées par Planètes Sciences, seulement trois personnes de l'équipe (Marie BRUN, Léo-Paul LAURENT et Florian LEVRAY) ont été invitées à venir qualifier et lancer le projet.

Le jour J

Pour avoir une autorisation de vol, la mini-fusée doit passer toute une série de tests répertoriés dans un cahier des charges, rédigé par les bénévoles de Planète Sciences. Ces tests se font en plusieurs étapes et doivent démontrer, entre autres, la stabilité de la fusée avec sa géométrie finale, le bon fonctionnement du séquenceur qui contrôle l'ouverture du parachute, le bon fonctionnement de l'expérience, la cohérence de la chronologie de vol spécifique à notre projet, etc. Pendant ces tests, l'intégralité de la fusée a donc été inspectée le soir du 24 juillet.

Déroulement du vol

La mini-fusée a décollé de la rampe de lancement le 25 juillet à 14H. Après une ascension d'environ 10 s dans un ciel nuageux avec un plafond à 200 ft, l'ouverture du parachute s'est déclenchée à l'apogée de sa trajectoire, soit une altitude de 290 m au-dessus du sol. La descente s'est faite à une vitesse constante de 36 km/h, ce qui était prévu. La mini-fusée s'est ensuite posée sans encombre sur l'herbe au bout de 40 s de vol nominal. A l'arrivée, notre fusée était tout à fait intacte. Les caméras et données nous ont indiquées que nous avons fait une ouverture au meilleur moment et la détection d'apogée a parfaitement fonctionné ! La fusée était censée prendre 14 G au décollage mais n'a pris que 10 G. Cela est dû à la rampe de la cage qui l'a fortement ralentie.

Préparation du Live

Malgré quelques soucis de connexion internet et de batteries, nous avons pu tout de même enregistrer, en live sur Youtube (chaîne Taranis), le lancement de la mini-fusée. Cet événement a permis à notre audience de pouvoir réagir et poser des questions instantanément pendant le déroulé du vol.

Par ailleurs, Léo-Paul a réalisé une application sur MATLAB (Figure 35), permettant d'interpréter en direct les données de la télémétrie. C'était donc l'occasion de l'afficher durant le live.

L'application Matlab a été conçue avec l'aide de l'app designer inclus dans MATLAB R2020b avec la compatibilité de l'arduino. Le principe du programme (voir Annexe 6) repose sur un callback appelé dès que MATLAB reçoit une information serial de l'arduino. Lorsque ce callback est appelé, on fait en sorte de lire les dernières informations reçues afin d'assurer une certaine fluidité quelque soit la puissance de l'ordinateur. Il y a également quelques protections empêchant d'afficher des informations non complètes qui sont cependant à renforcer au vu des problèmes mineurs ayant eu lieu pendant le live.

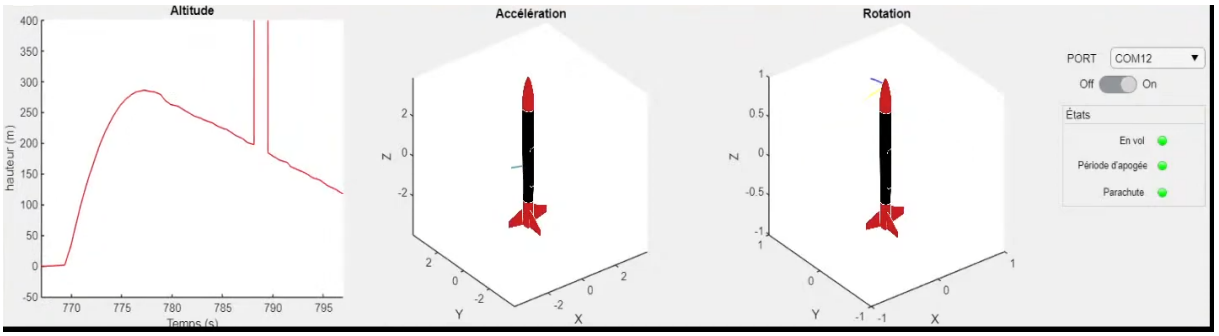


Figure 35 : Application pour exploiter la télémétrie

Exploitation des données de vol

Les données de vols reçues par télémesure en temps réel ont été stockées sur l'ordinateur à l'état brut dans un fichier texte, puis importées dans un tableur Excel, afin de calculer tous les paramètres de vol, du décollage jusqu'au contact avec le sol. Ces données de vol sont présentées sous forme de graphiques dont les plus importants sont interprétés ci-dessous. Ces courbes permettent de confirmer les prédictions des simulations réalisées avant le vol ou de déceler certains écarts avec la théorie.

Pression statique

Le capteur BME280 a parfaitement rempli son rôle et nous a permis d'obtenir la courbe de pression que nous attendions (Figure 36).

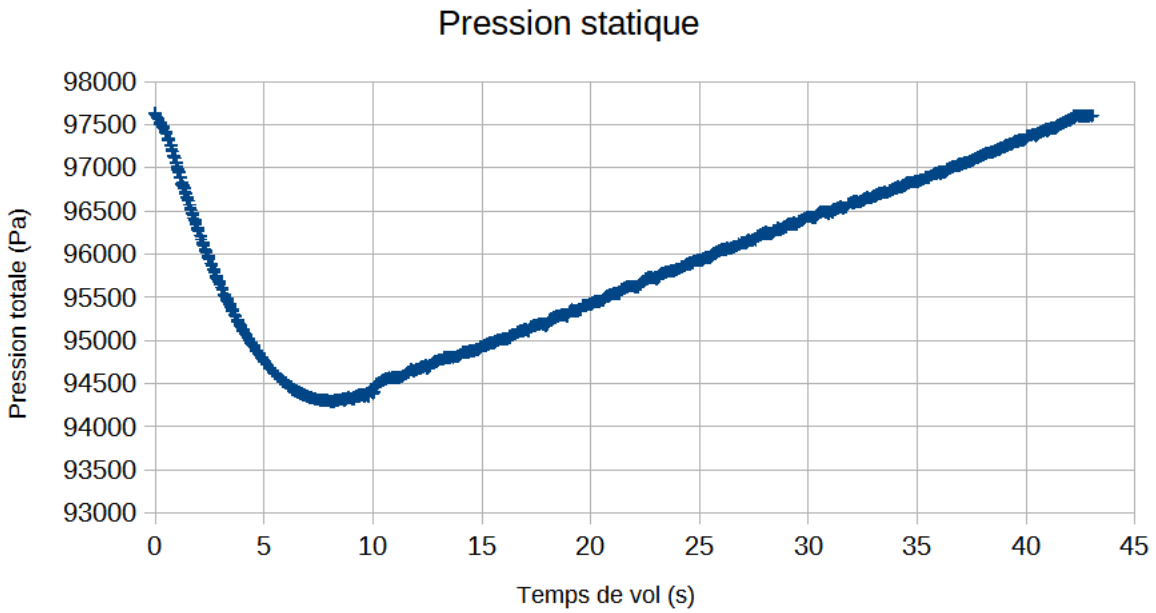


Figure 36 : Pression statique mesurée par le capteur BME280

Comme nous pouvons le voir, la pression statique diminue d'autant que l'altitude augmente pour atteindre un minimum de 942.94 hPa. Lors de la décélération complète, après ouverture du parachute, la pression statique remonte pour atteindre la pression atmosphérique au sol du

départ. Pour obtenir le temps de vol exact, il a fallu soustraire le temps enregistré par le capteur BME par le temps au moment du décollage, puis de tout diviser par 1000.

Altitude

De la mesure de pression statique, nous avons pu déterminer l'altitude de vol de la fusée à tout instant :

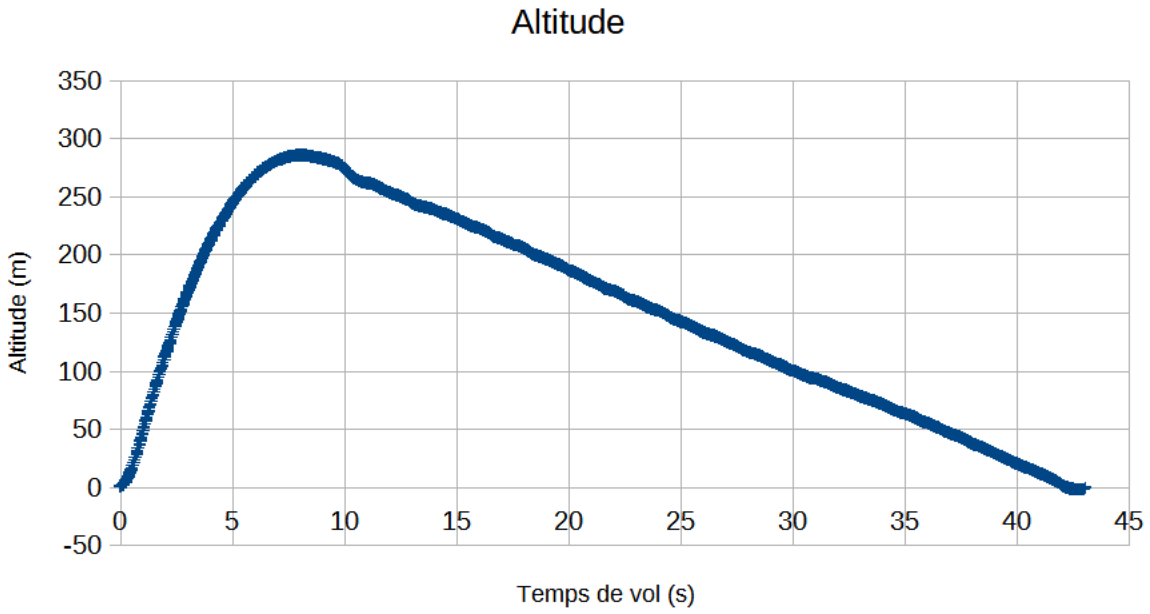


Figure 37 : Courbe d'altitude calculée à partir de la pression statique

Sur ce graphique est représenté l'évolution de l'altitude pendant toute la durée du vol (phase propulsive, phase balistique et retombée sous parachute). La mini-fusée a culminé à une altitude de 290 m, ce qui est 9 m de plus que les prévisions faites à l'aide du logiciel StabTraj. Cela prouve ainsi la bonne qualité de l'ensemble de la chaîne de télémesure (émetteur, antennes, récepteur).

Accélérations

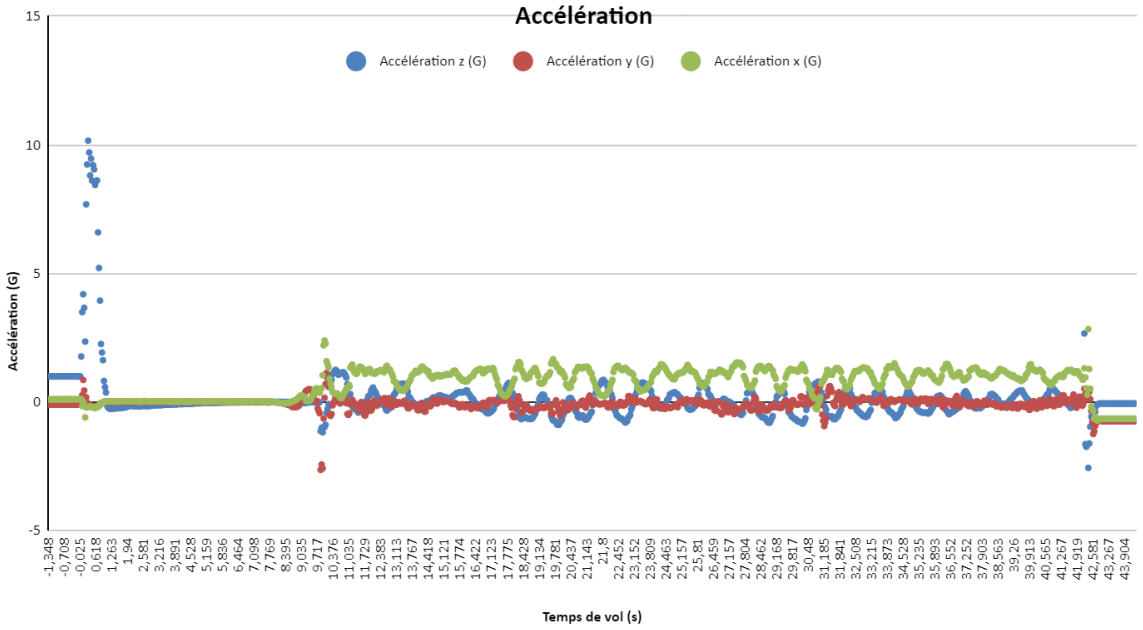


Figure 38 : Accélérations autour des axes x,y et z

Sur cet autre graphique sont représentées toutes les accélérations sur les trois axes x,y et z subies par la fusée pendant toute la phase ascendante. Les accélérations transversales sont quasiment négligeables par rapport à l'accélération longitudinale. Concernant cette dernière, avant décollage, on constate une valeur de 1 G correspondant à l'attraction terrestre détectée par le capteur. Au top du décollage, l'accélération fournie par le moteur est maximale, un pic de 10 G est enregistré pour rapidement se diriger sur une pente descendante. Au départ, la courbe dépend de la force propulsive générée par le moteur Pandora. Plus la vitesse augmente, plus la traînée aérodynamique va contrer cette force et plus la fusée prend de la hauteur, plus l'atmosphère se raréfie et la traînée aérodynamique diminue.

Vitesses angulaires

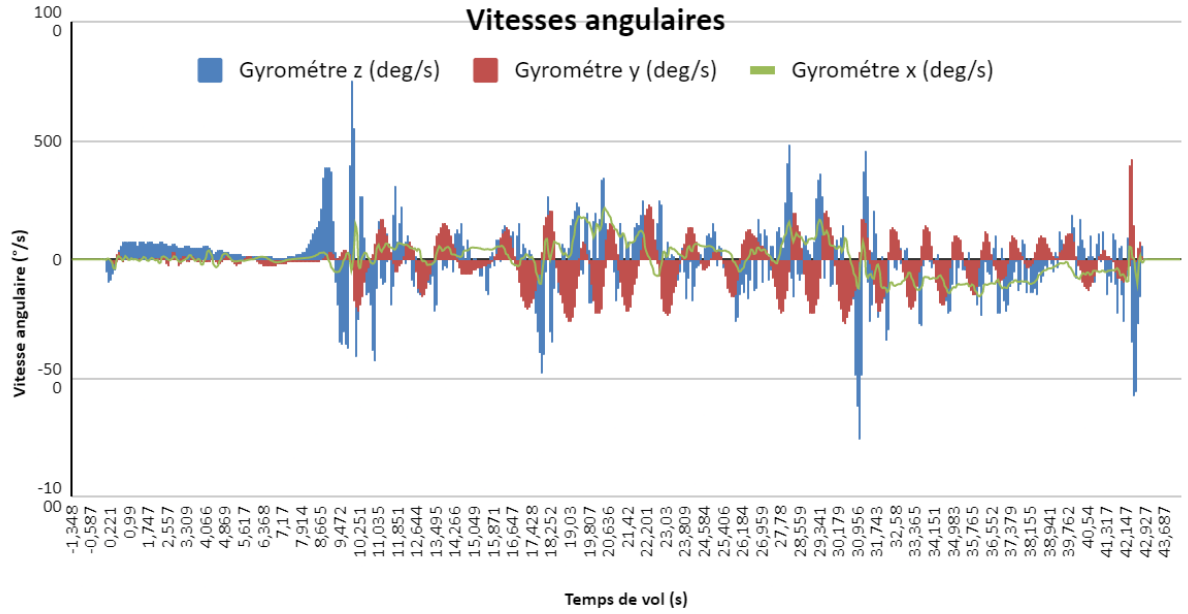


Figure 39 : Vitesses angulaires autour des axes x,y et z

Voici l'évolution des vitesses angulaires autour des axes x,y et z de la fusée mesuré par le gyroscope pendant le vol. On remarque clairement que les trois vitesses angulaires sont restées à peu près constantes tout au long de la phase ascensionnelle. On notera les fortes variations en fin de phase de propulsion, exactement comme celles observées sur le graphique des accélérations. Au moment de l'ouverture du parachute, la valeur des vitesses angulaires "s'affolent" puisque la fusée n'est plus sur une trajectoire stabilisée mais réalise des mouvements erratiques.

Erreurs à ne pas refaire

Après le vol expérimental et l'analyse des différents paramètres physiques mesurés à bord de la fusée, voici les points sur lesquels la prochaine équipe devrait éviter de refaire :

- Préférer une perf-board à un PCB:

Les perf boards sont très difficiles à utiliser pour les débutants et trop lourds pour le vol d'une fusée. Il faut utiliser des PCB fait maison ou commander sur internet à l'aide de sites comme JLC pcb.

- Ne pas utiliser d'insert pour les vis dans la structure imprimée en 3D :

Les inserts augmentent drastiquement la résistance de la structure de la fusée pour un poids négligeable. L'utilisation d'insert est préférable car leur position peut être légèrement ajustée à l'aide d'un fer à souder pour prendre en compte un mauvais alignement.

- Mal définir les butées du servo moteur du parachute :

Suite à une erreur d'estimation, le servomoteur ne pouvait pas atteindre la position demandée lorsque la trappe était fermée. Celui-ci va s'user beaucoup plus vite et peut se casser s' il cherche à maintenir une position que ne lui permet pas le mécanisme.

- Utiliser une perceuse colonne plutôt qu'une perceuse à main (plus précise) :

La perceuse colonne est moins pratique et précise qu'une perceuse à main avec un opérateur habitué.

- Ne pas ajouter de jeux dans la modélisation des pièces :

La précision de l' usinage n'est pas absolue, pour que 2 objets s'emboîtent il faut laisser un jeu. Pour que deux pièces imprimées en 3D s'emboîtent, il faut laisser une marge d'au moins un demi-millimètre entre celles-ci.

- Ne pas gluer les fils de la carte PCB

Les fils soumis à de fortes accélérations peuvent être arrachés, soit par leur propre poids soit par un composant qui se serait libérée lors de l'accélération. Pour les empêcher de se détacher et empêcher les courts circuits il faut les sécuriser sur le PCB, il existe des produits spéciaux mais le pistolet à colle et la glue marchent parfaitement dans la plupart des cas.

- Ne pas utiliser de mousqueton pour fixer la ligne parachute :

La chaîne parachute est très complexe et doit pouvoir être remplacée facilement. Pour cela on peut utiliser un mousqueton qui se fixe sur la fusée. Néanmoins, celui-ci doit être très résistant.

- Utiliser une trappe imprimée en 3D avec un parachute très comprimé :

La trappe en PLA effectue une flexion importante qui est aggravée par la pression que le parachute exerce sur cette porte. Il faut améliorer ce point sur les futures fusées en utilisant par exemple des baguettes en métal ou imprimer en 3D, en concevant une trappe qui prend en compte ces efforts ou en limitant la compression du parachute.

- Utiliser de l'aluminium pour le tube :

L'aluminium est très résistant et léger mais très difficile à usiner. A l'aide d'une dremel et d'une ponceuse il a fallu 3 mois pour faire les 80 cm du tube de TARANIS. Un tube en fibre de verre aurait été plus léger et beaucoup plus facile à usiner qu'un tube en aluminium.

- Démonter une caméra trop fragile

Les caméras sont trop fragiles pour être démonter comme nous l'avons fait, celle-ci cassait trop facilement. Il faut changer de caméra à l'avenir et opter pour des modèles FPV.

- Utilisation des carte Arduino mini pro

Les cartes Arduino nano sont plus facilement programmables et offrent plus de possibilités de branchements tout en permettant la conversion de courant.

- Utiliser le détecteur de branchement du Jack

Le jack possède un pin qui détecte si une prise est branché dans le port. Mais si la prise se détache et reste coincée dans le port lors du décollage, la fusée s'écrasera. Pour cela on peut simplement relier les deux bornes de la prise jack mâle (qui sera arrachée) avec du fil électrique. Ainsi, si le fil se rompt, la fusée déclenchera le séquenceur malgré le fait que la prise jack est restée branchée sur la fusée.

Conclusion

Taranis est une mini-fusée expérimentale réalisée au cours des années scolaires 2019-2020 et 2020-2021 par des membres de l'association ELISA SPACE. Le projet a rempli ses objectifs et a effectué un vol nominal, ce fut un véritable succès. C'était également l'occasion pour l'équipe de mener à bien un projet ambitieux et de collaborer avec différents partenaires, ce qui est particulièrement adapté aux formations de l'école d'Ingénieurs au sein duquel a eu lieu ce projet.

Ce type de projet nous a, par ailleurs, permis d'appliquer les cours appris à l'école et d'améliorer nos compétences techniques. En effet, il était nécessaire d'avoir des connaissances à la fois en traitement du signal et en électronique, deux matières enseignées au sein de notre établissement. Un tel projet fut très formateur pour l'équipe, cela a permis de transmettre les connaissances aux élèves qui nous suivent, afin d'éviter certaines erreurs et permettre de réaliser des projets toujours plus aboutis.

Il est très important de transmettre nos connaissances vers des projets similaires futurs. En effet, d'autres personnes voulant construire une fusée pourraient bénéficier de notre expérience et d'une certaine hérédité qui accroîtrait au fil des années. C'est un avantage dont il ne faut pas passer à côté, il est important de l'utiliser pour progresser. Les projets sur les mini-fusées sont conséquents, il n'est pas possible de repartir chaque année à zéro, d'autant plus que ceci aurait peu d'intérêt. C'est la raison pour laquelle j'ai essayé d'expliquer en détail toute la démarche du projet. De plus, nous transmettrons aux prochaines équipes, travaillant sur des projets de mini-fusées, tous nos documents afin de les aider dans leurs travaux à partir des expériences que nous avons pu avoir.

ANNEXE

Annexe 1 : StabTraj

STABILITO

Stabilité de fusée à ailerons

Remplir les cases jaunes

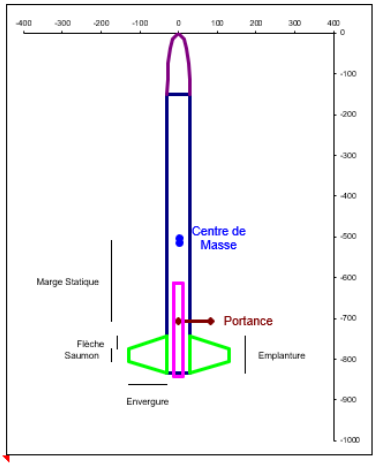
Fusée	
Nom	Taranis
Club	Elisa Space
Type	Minifusée
Masse	1478,7 g sans propu
Centre de Masse	490 mm sans propu
Longueur totale	834,664 mm

Propulseur	
Type	Pandora
Position du bas	841,014 mm

Coiffe	
Forme	Parabolique (arrondie)
Hauteur	150 mm
Diamètre	60 mm

Ailerons	Mono-empennage
Emplanture 'm'	90 mm
Saumon 'n'	30 mm
Flèche 'p'	30 mm
Envergure 'E'	100 mm
Epaisseur 'ep'	2 mm
Nombre	4
Position du bas	834,664 mm

Commentaire libre :



17/06/2021	Min	Résultats		Max
Finesse	10	13,9		20
Portance	15	20,6	20,6	30
Marge Stat.	1,5 D	3,21 D	3,38 D	6 D
Couple	30	66,0	69,6	100
XCp		706 mm	706 mm	
MS /L		23% L	24% L	

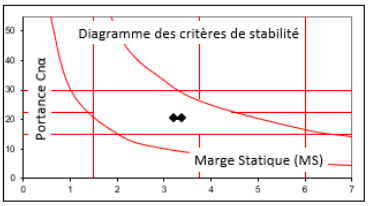
STABLE

Language/Langue : Français

Fusée mono-diamètre,

	Propu plein	Propu vide	Sans propu
Masse propu	0,16 kg	0,084 kg	-
CdM propu	114 mm	114 mm	-
Masse fusée	1,639 kg	1,563 kg	1,479 kg
CdM fusée	513 mm	503 mm	490 mm

	XCp	Cnα
Coiffe	75 mm	2,0
Ailerons	773 mm	18,6



Checksum : propu OK v3.4

Vérification de la stabilité

TRAJECTO

Trajectographie de fusée

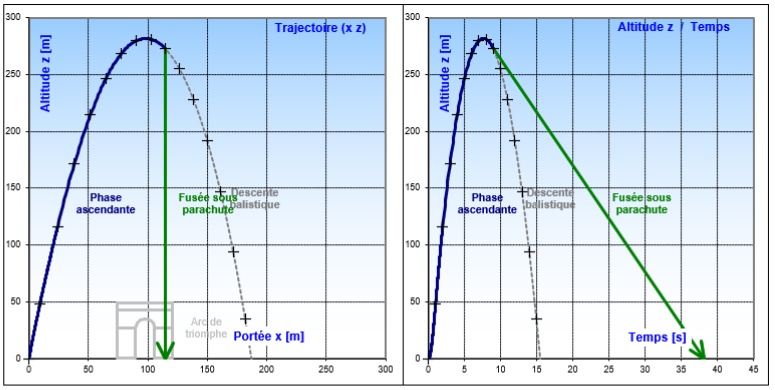
Remplir les cases jaunes

Fusée	
Nom	Taranis
Club	Elisa Space
Masse totale	1,6386 kg
Propulseur	Pandora

Traînée Aérdynamique	
Surface Réf.	0,003627 m²
Cx	0,6

Rampe de Lancement	
Longueur	2,5 m
Élévation	80 °
Altitude	0 m

DescenteSousParachute	Fusée	0 satellite
Masse	1,563 kg	
Dépotage	autre	
Ouverture para	9 s	
Surface para	0,29 m²	
Cx parachute	1	
Vitesse du vent	5 m/s	
Vitesse descente	9,3 m/s	
Durée descente	29 s	
Durée du vol	38 s	
Déport latéral	± 146 m	



17/06/2021	Temps	Altitude z	Portée x	Vitesse	Accélération	Efforts
Sortie de Rampe				23,5 m/s		
Vit max & Acc max				77 m/s	137 m/s²	
Culmination, Apogée	7,6 s	281 m	98 m	12 m/s		
Ouverture parachute fusée	9,0 s	273 m	115 m	18 m/s		56,5 N
Impact balistique	15,6 s	~0 m	188 m	67,3 m/s		3538 J

	Pour localiser la fusée
Couleur fuselage/coiffe	Orange
Couleur parachute fusée	Rouge

propu OK v3.4

Commentaire libre :

Estimation de la trajectoire et dimensions du parachute

Annexe 2 : Liens des librairies

MPU6050_6Axis_MotionApps20.h :
https://github.com/jrowberg/i2cdevlib/blob/master/Arduino/MPU6050/MPU6050_6Axis_MotionApps20.h

SparkFunBME280.h :
https://github.com/sparkfun/SparkFun_BME280_Arduino_Library/blob/master/src/SparkFunBME280.h

LoRa_E32.h : https://github.com/xreef/LoRa_E32_Series_Library/blob/master/LoRa_E32.h

Annexe 3 : Séquenceur

```
#include <Servo.h> //bibliothèque servo (déjà présente sur arduino)
Servo Servomoteur; //créer un objet servo
const int Pin_LED = 6;
const int Pin_Jack = 9;
const int Pin_Servo = 11;
const int Position_initiale = 90;
const int Position_finale = 180;

unsigned long dt1 = 10; //temps avant plage d'ouverture para en sec
unsigned long dt2 = 20; //temps ouverture para obligatoire en sec
unsigned long dt1_millis;
unsigned long dt2_millis;
unsigned long t1_millis;
unsigned long t2_millis;
unsigned long t0;

void Ouverture(){
 Servomoteur.write(Position_finale);
 digitalWrite(Pin_LED, HIGH);
}

void setup() {
 Servomoteur.attach(Pin_Servo); // Attache le servomoteur au PinServo
 Servomoteur.write(Position_initiale); // Met le servomoteur en position
 dt1_millis = dt1*1000;
 dt2_millis = dt2*1000;
 Serial.begin(19200);
 pinMode(Pin_Jack,INPUT_PULLUP); //set as INPUT
 pinMode(Pin_LED,OUTPUT);
 digitalWrite(Pin_LED, HIGH);
}

void loop() {
 if(LOW==digitalRead(Pin_Jack)){
 t0 = millis(); // prend T0
 Serial.print(1);
 digitalWrite(Pin_LED, LOW);
 t1_millis = dt1_millis+t0;
 t2_millis = dt2_millis+t0;
 while(millis()<=t1_millis && LOW==digitalRead(Pin_Jack)){
 Serial.print(1);
 }
 while(millis()<=t2_millis && LOW==digitalRead(Pin_Jack)){
 Serial.print(2);
 if(Serial.available()){
 Serial.print(3);
 Ouverture();
 while(1){
 Serial.print(3);
```

```
}
}
}
Serial.print(4);
Ouverture();
while(1){
Serial.print(4);
}
}
else
{
Serial.print(0);
}
}
```

```
const int Pin_LED = 8;
//------------------SD--------------------
#include <SPI.h>
#include <SD.h>
File donnees;
const int Pin_CS = 10;
//----------------MPU+I2C-----------------
#include <Wire.h>
#include <I2Cdev.h>
#include "MPU6050_6Axis_MotionApps20.h"
MPU6050 mpu;
int16_t ax, ay, az, gx, gy, gz;
//------------------BME--------------------
#include "SparkFunBME280.h"
BME280 bme;
//-----------------TELEM-------------------
#include "LoRa_E32.h"
LoRa_E32 e32ttl100(14, 7,15); // e32 TX e32 RX

float presmin;
const double facteurDeDescente = 1.000005;

void setup() {
 pinMode(Pin_LED, OUTPUT);
 //----------------------Serial----------------------
 Serial.begin(19200);
 if (!Serial) {
 while (1) {
 digitalWrite(Pin_LED, HIGH);
 delay(100);
 digitalWrite(Pin_LED, LOW);
 delay(100);
 }
 }
 //-------------------------BME--------------------
 Wire.begin();
bme.setI2CAddress(0x76); //adresse du capteur de pression BMP280
bme.beginI2C();
if (!bme.begin()) {
Serial.println("Echec connextion BME");
while (1) {
digitalWrite(Pin_LED, HIGH);
delay(100);
digitalWrite(Pin_LED, LOW);
delay(100);
}
}
Serial.println("2");
//-------------------------SD-----------------------
if (!SD.begin(Pin_CS)) {
Serial.println("Echec connextion SD");
while (1) {
digitalWrite(Pin_LED, HIGH);
delay(100);
digitalWrite(Pin_LED, LOW);
delay(100);
}
```

```
}
//------------------------MPU+I2C--------------------
Wire.begin();

mpu.initialize();
mpu.dmpInitialize(); //initialise la connexion au calculateur du MPU6050
mpu.setFullScaleAccelRange(MPU6050_ACCEL_FS_16); //Configure le capteur à 16g
mpu.setFullScaleGyroRange(MPU6050_GYRO_FS_2000); //Configure le capteur à 250 deg/sec
mpu.setDLPFMode(6);
mpu.setDHPFMode(1);

if (!mpu.testConnection()) {
Serial.println("Echec connexion MPU");
while (1) {
digitalWrite(Pin_LED, HIGH);
delay(100);
digitalWrite(Pin_LED, LOW);
delay(100);
}
}
mpu.setXAccelOffset(-5132); //mettre les offset déterminer à partir du programme "Calibration_offset"
mpu.setYAccelOffset(-2092);
mpu.setZAccelOffset(1788);
mpu.setXGyroOffset(-98);
mpu.setYGyroOffset(129);
mpu.setZGyroOffset(69);
//----------------------TELEM--------------------
pinMode(16,OUTPUT);
digitalWrite(16,HIGH);
e32ttl100.begin();

digitalWrite(Pin_LED, HIGH);
}
void loop() {
byte s;
unsigned long t = millis();
int16_t p = bme.readFloatPressure() - 100000;
mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
while(Serial.available()){
s = Serial.read();
}
byte c = bme.readTempC();
char frame[59];
snprintf(frame, sizeof(frame), "t%lup%ds%dg%dz%dh%dy%dj%dx%dc%d\n", t, p, s, gx, ax, -gy, -ay, -gz, -az,
c);
// temps pression serial gory z acc z gyro y acc y gyro x acc x température
donnees = SD.open("donnees.txt", FILE_WRITE);
donnees.println(frame);
donnees.close();
e32ttl100.sendMessage(frame);
if (p <= presmin) {
presmin = p;
}
else if (s == '2' && p>=presmin*facteurDeDescente) {
Serial.print('1');
}
}
```

Annexe 5 : Télémétrie

```
/*
 * LoRa E32-TTL-100
 * Write on serial to transfer a message to other device
 * https://www.mischianti.org
 *
 * E32-TTL-100----- Arduino UNO
 * M0 ----- GND
 * M1 ----- GND
 * TX ----- PIN 2 (PullUP)
 * RX ----- PIN 3 (PullUP & Voltage divider)
 * AUX ----- Not connected
 * VCC ----- 3.3v/5v
 * GND ----- GND
 *
 */
#include "Arduino.h"
#include "LoRa_E32.h"

LoRa_E32 e32ttl100(2, 3); // e32 TX e32 RX

void setup() {
 Serial.begin(19200);
 // Startup all pins and UART
 e32ttl100.begin();
}

void loop() {
 ResponseContainer rc = e32ttl100.receiveMessageUntil('\n');
 Serial.println(rc.data);
}
```

Annexe 6 : Application MATLAB de réception

```
classdef streaming < matlab.apps.AppBase
    % Properties that correspond to app components
    properties (Access = public)
        UIFigure              matlab.ui.Figure
        Switch                matlab.ui.control.Switch
        PORTDropDownLabel     matlab.ui.control.Label
        PORTDropDown          matlab.ui.control.DropDown
        tatsPanel             matlab.ui.container.Panel
        EnvolLampLabel        matlab.ui.control.Label
        EnvolLamp             matlab.ui.control.Lamp
        PriodedapogeLampLabel matlab.ui.control.Label
        PriodedapogeLamp      matlab.ui.control.Lamp
        ParachuteLampLabel    matlab.ui.control.Label
        ParachuteLamp         matlab.ui.control.Lamp
        UIAxes                matlab.ui.control.UIAxes
        UIAxes2               matlab.ui.control.UIAxes
        UIAxes3               matlab.ui.control.UIAxes
    end

    properties (Access = private)
        COM;
        data;
        arduinoObj;
        n;
        R = 1;
        C=0.0005;

        z1 = linspace( 0,0, 100);

        x2 = linspace( 0,0, 100);

        y3 = linspace( 0,0, 100);

        h_0;
    end

    methods (Access = public)
        function read(app,src, ~)

            table = 'tpsgzhyjxc';
            % Read the ASCII data from the serialport object.

            while src.NumBytesAvailable ~= 0 || app.data == "\n" || strlength(app.data) < 39
              app.data = readline(src);

            end

            src.UserData.Count = src.UserData.Count + 1;
```

```
zer = app.data;
expression = '[a-z][-]?[\d]+';
matchStr = regexp(zer,expression,'match');

m = max(size(matchStr));
u = zeros(length(table),1);

for i = 1:m

    for j = 1:length(table)

      if matchStr{i}(1) == table(j)

       u(j) = str2double(matchStr{i}(2:end));

      end

    end

 end


if u(2) ~= 0
   u(2) = u(2) + 100000;
end


h = 44330.8*(1-(u(2)/101325)^0.190289);

if src.UserData.Count == 1
app.n = animatedline(app.UIAxes,(1),h,'Color','r');
xlim(app.UIAxes,[0 30]);
ylim(app.UIAxes,[-50 400]);

app.h_0 = h;

end

if u(1) > 30000
  xlim(app.UIAxes,[u(1)/1000-30 u(1)/1000])
end

addpoints(app.n,u(1)/1000,h - app.h_0);

drawnow;


if abs(u(9)/2077) > 2 || abs(u(7)/2044) > 2 || abs(u(5)/2043) > 2
    accelmax = max([abs(u(9)/2077),abs(u(7)/2044),abs(u(5)/2043)]);
    xlim(app.UIAxes2,[-(accelmax) (accelmax)]);
    ylim(app.UIAxes2,[-(accelmax) (accelmax)]);
```

```
        zlim(app.UIAxes2,[-(accelmax) (accelmax)]);
    end

    quiver3(app.UIAxes2,0,0,0,u(9)/2077,u(7)/2044,u(5)/2043) %%ici inverser si besoin

    if(u(8)>0)
        th1 = linspace( 0, -(u(4)-13), 100);
    else
        th1 = linspace( -(u(4)-13), 0, 100);
    end
    x1 = app.R*cos(th1*app.C);
    y1 = app.R*sin(th1*app.C);

    if(u(6)>0)
        th2 = linspace( 0, u(8), 100);
    else
        th2 = linspace( u(8), 0, 100);
    end
    z2 = app.R*cos(th2*app.C);
    y2 = app.R*sin(th2*app.C);

    if(u(4)>0)
        th3 = linspace( 0, -(u(6)-5), 100);
    else
        th3 = linspace( -(u(6)-5), 0, 100);
    end
    z3 = app.R*cos(th3*app.C);
    x3 = app.R*sin(th3*app.C);

    plot3(app.UIAxes3,x1,y1,app.z1,"color",'red');
    hold(app.UIAxes3,"on")
    plot3(app.UIAxes3,app.x2,y2,z2,"color",'blue');
    plot3(app.UIAxes3,x3,app.y3,z3,"color",'yellow');
    hold(app.UIAxes3,"off")

    if u(3) == 48
        app.EnvolLamp.Color =  'red';
    end

    if u(3) == 49
        app.EnvolLamp.Color =  'green';
    end
    if u(3) == 50
        app.PriodedapogeLamp.Color =  'green';
    end
    if u(3) == 51
        app.ParachuteLamp.Color =  'green';
    end
    if u(3) == 52
        app.ParachuteLamp.Color =  'blue';
    end
end
```

```
    end

    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
            app.UIFigure.Name = "TARANIS flight observer";

            app.PORTDropDown.Items = serialportlist("available");
            app.COM = app.PORTDropDown.Value;
            app.EnvolLamp.Color =  'black';
            app.PriodedapogeLamp.Color =  'black';
            app.ParachuteLamp.Color =  'black';

            axis(app.UIAxes2,"equal");
            xlim(app.UIAxes2,[-2 2]);
            ylim(app.UIAxes2,[-2 2]);
            zlim(app.UIAxes2,[-2 2]);

            axis(app.UIAxes3,'equal');
            xlim(app.UIAxes3,[-1 1]);
            ylim(app.UIAxes3,[-1 1]);
            zlim(app.UIAxes3,[-1 1]);
        end

        % Value changed function: Switch
        function SwitchValueChanged(app, event)
            if app.Switch.Value == "On"
                app.arduinoObj = serialport(app.COM,115200);

                configureTerminator(app.arduinoObj,"CR/LF");

                flush(app.arduinoObj);

                app.arduinoObj.UserData = struct("Count",0);

                configureCallback(app.arduinoObj,"terminator",@app.read);
            end
            if app.Switch.Value == "Off"
                delete(app.arduinoObj);
                cla(app.UIAxes);
                cla(app.UIAxes2);
                cla(app.UIAxes3);
                pause(2);

                cla(app.UIAxes);
                cla(app.UIAxes2);
                cla(app.UIAxes3);
            end

        end
```

```
        % Value changed function: PORTDropDown
        function PORTDropDownValueChanged(app, event)
            app.COM = app.PORTDropDown.Value;

        end

        % Drop down opening function: PORTDropDown
        function PORTDropDownOpening(app, event)
            app.PORTDropDown.Items = serialportlist("available");
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 827 208];
            app.UIFigure.Name = 'MATLAB App';

            % Create Switch
            app.Switch = uiswitch(app.UIFigure, 'slider');
            app.Switch.ValueChangedFcn = createCallbackFcn(app, @SwitchValueChanged, true);
            app.Switch.Position = [747 117 44 19];

            % Create PORTDropDownLabel
            app.PORTDropDownLabel = uilabel(app.UIFigure);
            app.PORTDropDownLabel.HorizontalAlignment = 'right';
            app.PORTDropDownLabel.Position = [706 143 39 22];
            app.PORTDropDownLabel.Text = 'PORT';

            % Create PORTDropDown
            app.PORTDropDown = uidropdown(app.UIFigure);
            app.PORTDropDown.DropDownOpeningFcn = createCallbackFcn(app, @PORTDropDownOpening,
true);
            app.PORTDropDown.ValueChangedFcn = createCallbackFcn(app, @PORTDropDownValueChanged,
true);
            app.PORTDropDown.Position = [760 143 63 22];

            % Create tatsPanel
            app.tatsPanel = uipanel(app.UIFigure);
            app.tatsPanel.Title = 'États';
            app.tatsPanel.Position = [706 0 117 105];

            % Create EnvolLampLabel
            app.EnvolLampLabel = uilabel(app.tatsPanel);
            app.EnvolLampLabel.HorizontalAlignment = 'right';
            app.EnvolLampLabel.FontSize = 10;
```

```
app.EnvolLampLabel.Position = [59 57 33 22];
app.EnvolLampLabel.Text = 'En vol';

% Create EnvolLamp
app.EnvolLamp = uilamp(app.tatsPanel);
app.EnvolLamp.Position = [107 63 10 10];

% Create PriodedapogeLampLabel
app.PriodedapogeLampLabel = uilabel(app.tatsPanel);
app.PriodedapogeLampLabel.HorizontalAlignment = 'right';
app.PriodedapogeLampLabel.FontSize = 10;
app.PriodedapogeLampLabel.Position = [17 31 78 22];
app.PriodedapogeLampLabel.Text = 'Période d''apogée';

% Create PriodedapogeLamp
app.PriodedapogeLamp = uilamp(app.tatsPanel);
app.PriodedapogeLamp.Position = [107 37 10 10];

% Create ParachuteLampLabel
app.ParachuteLampLabel = uilabel(app.tatsPanel);
app.ParachuteLampLabel.HorizontalAlignment = 'right';
app.ParachuteLampLabel.FontSize = 10;
app.ParachuteLampLabel.Position = [41 3 51 22];
app.ParachuteLampLabel.Text = 'Parachute';

% Create ParachuteLamp
app.ParachuteLamp = uilamp(app.tatsPanel);
app.ParachuteLamp.Position = [107 9 10 10];

% Create UIAxes
app.UIAxes = uiaxes(app.UIFigure);
title(app.UIAxes, 'Altitude')
xlabel(app.UIAxes, 'Temps (s)')
ylabel(app.UIAxes, 'hauteur (m)')
zlabel(app.UIAxes, 'Z')
app.UIAxes.PlotBoxAspectRatio = [1.18134715025907 1 1];
app.UIAxes.Position = [1 0 230 209];

% Create UIAxes2
app.UIAxes2 = uiaxes(app.UIFigure);
title(app.UIAxes2, 'Accélération')
xlabel(app.UIAxes2, 'X')
ylabel(app.UIAxes2, 'Y')
zlabel(app.UIAxes2, 'Z')
app.UIAxes2.PlotBoxAspectRatio = [1.2962962962963 1 1];
app.UIAxes2.Position = [230 3 244 206];

% Create UIAxes3
app.UIAxes3 = uiaxes(app.UIFigure);
title(app.UIAxes3, 'Rotation')
xlabel(app.UIAxes3, 'X')
ylabel(app.UIAxes3, 'Y')
```

```
            zlabel(app.UIAxes3, 'Z')
            app.UIAxes3.PlotBoxAspectRatio = [1.21164021164021 1 1];
            app.UIAxes3.Position = [473 3 231 206];

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = streaming

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```