Ensma Space Project

Documentation sur la réalisation d'une fusée expérimentale

VOLOKNA

Etude des déformations de la structure pendant la phase propulsée

Auteurs :

M. Clément LINGOIS

M. Alexis COLLET

M. Antoine CARTON

M. Anthony BARTHELEMY

M. Zacharie BAEUMLIN

Encadrants :

Mme. Laurence CHOCINSKI

M. Jean-Marie PETIT

Version du 30 août 2021

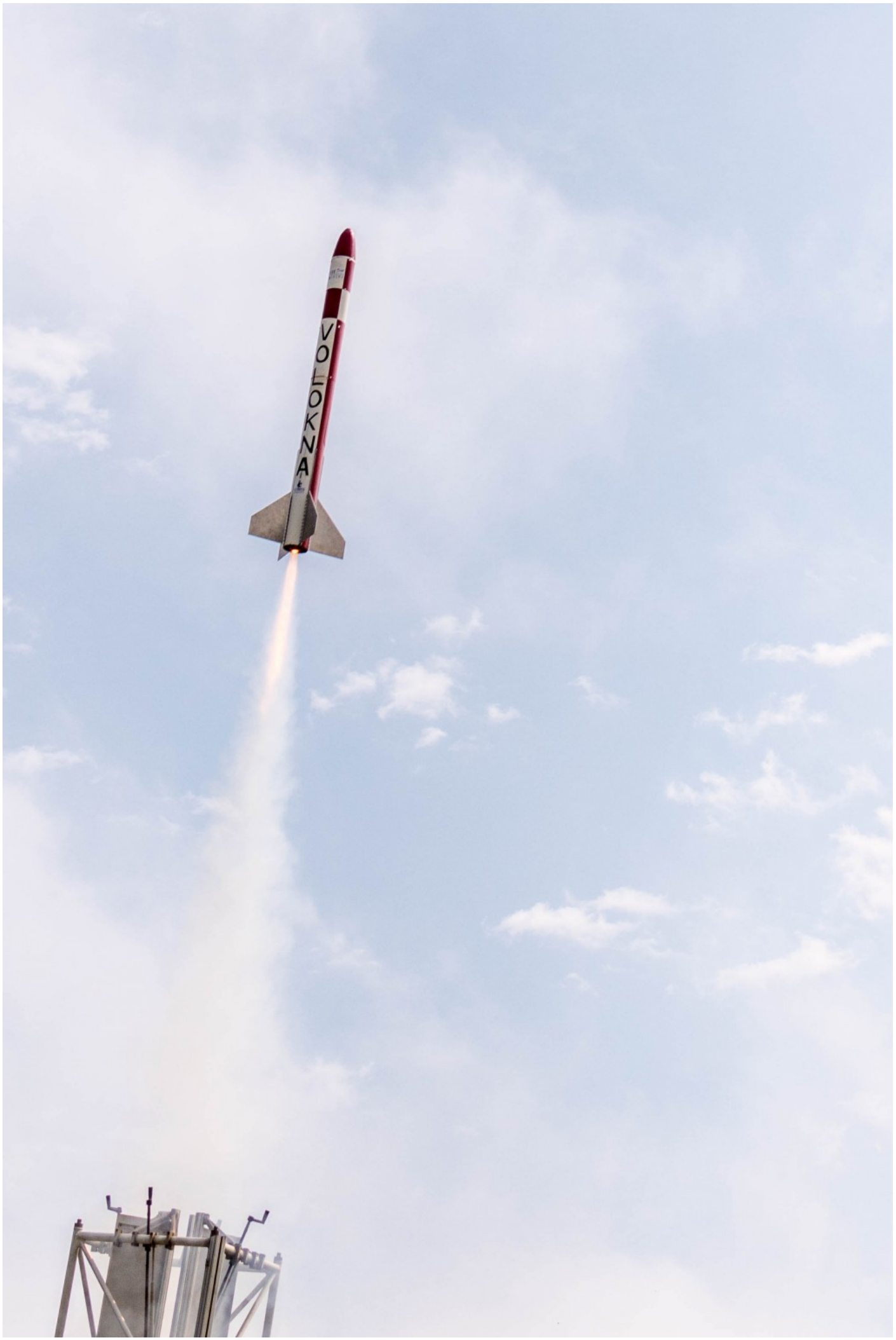

VOLOKNA

Table des matières

1 Avant-Propos

L'ESP (Ensma Space Project) est un club de l'ENSMA dont le but est de permettre aux étudiants de réaliser des projets dans le domaine spatial. L'événement principal est le C'Space, une campagne de lancement de fusée de différentes tailles organisée par Planète Science et le CNES.

Ainsi depuis plusieurs années l'ESP envoie des fusées au C'Space. Toutefois les fusées envoyées sont pour la grande majorité des mini-fusées, c'est-à-dire des fusées de petites tailles dont la conception en 1 an reste relativement facile. C'est pourquoi nous avons décidé de réaliser cette année une fusée expérimentale avec une structure simple pour réaliser une documentation poussée. Le but étant que pour les prochaines années les étudiants pourront réaliser de superbes fusées en 1 an.

2 Introduction

Depuis quelques années l'ESP (Ensma Space Project) construit et lance des fusées avec l'aide du CNES (Centre National d'Etudes Spatiales) et de Planète Sciences. La construction se fait tout au long de l'année.

Le CNES fournit le propulseur et gère sa mise en œuvre dans les conditions de sécurité adéquates. Planète Sciences gère l'organisation du lancement et s'occupe du suivi technique du club tout au long de l'année. En contrepartie, le club s'engage à respecter un cahier des charges (contenant essentiellement des règles de sécurité)

La campagne de lancement se déroule sur un terrain militaire. Pendant une semaine, il s'agit pour tous les clubs « spatiaux » de France (et parfois de l'étranger) de qualifier sa fusée et de la lancer. Planète Sciences gère ces qualifications.

Planète Sciences est une association nationale loi 1901 qui encadre aussi d'autres secteurs comme l'astronomie, l'environnement ou la robotique. C'est d'ailleurs Planète Sciences qui organise la coupe E=M6. Cette année nous allons travailler sur une fusée expérimentale nommée "Volokna" ce qui signifie fibre en russe.

L'équipe est composée d'ensmatiques issues des trois promotions 2021 2022 2023. L'équipe est détaillée en annexe.

3 Phase d'avant projet

L'idée de faire une fusée expérimentale nous est venue pendant les vacances d'été ainsi il a fallu réfléchir à une expérience pas trop complexe à concevoir que l'on pourrait embarquée dans notre fusée.

Une des grosses interrogations que nous avons eu concernait la faisabilité d'une fusée expérimentale en une année scolaire (soit 9 mois de cours). En plus de cela il y a le départ des deuxième année à prendre en compte car c'est eux qui lancent le projet et qui initient les premières années à la conception de fusée.

Ensuite nous nous sommes penchés sur la question de la conception et de la fabrication de la fusée. Allions nous avoir le temps nécessaire pour réaliser la conception sur catia (logiciel de modélisation 3D), la réalisation de la carte électronique et du système de récupération ?

Il faut garder en tête le fait que la conception doit être finie avant fin janvier car sinon les délais de fabrication ne seront pas respectés. De plus, les pièces réalisées sur catia doivent être facilement réalisables par le technicien de l'atelier.

Enfin nous nous sommes demandé s'il y allait avoir assez de personnes au C'space pour travailler sur notre fusex.

4 Phase de projet

4.1 Pôle experience

Notre expérience consiste en la mesure des contraintes et déformations subies par la structure de notre fusée expérimentale à l'aide de jauges de déformation. Afin de savoir où les placer et quelle sensibilité elles doivent avoir, nous réalisons une étude théorique.

4.1.1 Structure de la fusée

Nous détaillons ici la structure de notre fusée. La partie basse de la fusée sera faite de bagues en aluminium, la plus basse reprenant la poussée du moteur. Elles sont reliées par quatre poutres en U. La peau (un tube de PVC) est fixée aux bagues grâce à des vis. La partie haute a une structure similaire, à la différence qu'il n'y a plus que trois poutres longitudinales.

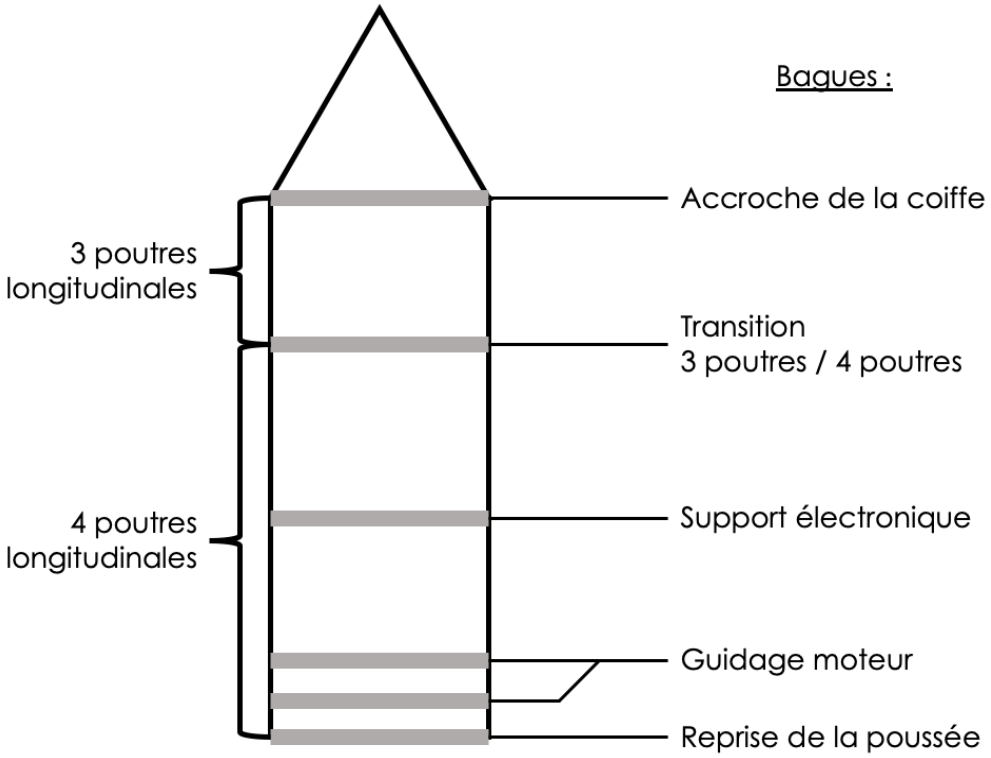


Figure 1.1 – Schéma de la répartition des bagues dans la fusée

4.1.2 Modélisation du chargement

Afin de pouvoir utiliser les outils de théorie des poutres, nous devons ramener les phénomènes dynamiques subis par la fusée à un chargement statique équivalent. De plus, d'après les données du stabtraj de Planète Sciences notre fusée sera quasiment verticale pendant la phase propulsée, puisqu'il y aura un angle de 10 à 15 degrés entre la verticale et l'axe longitudinal de la fusée. On se ramène donc au cas d'une fusée verticale et dont l'accélération est selon son axe longitudinal.

On a alors le schéma suivant :

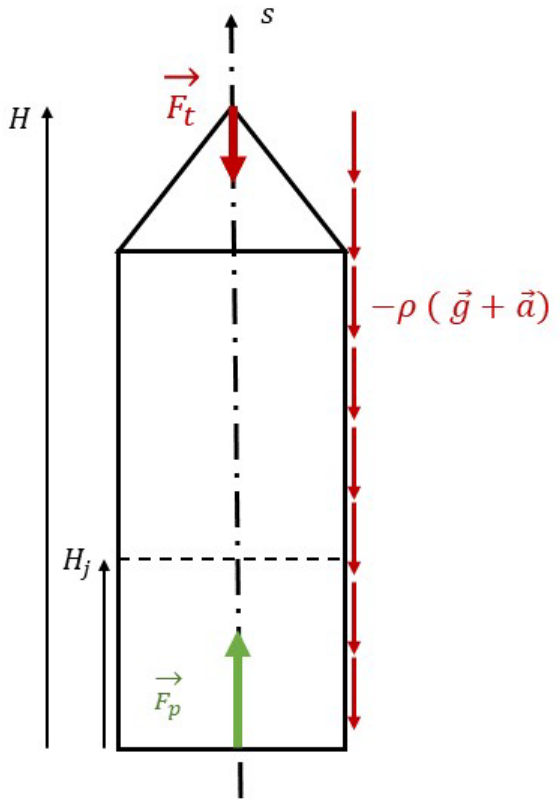


Figure 2.1 – Schéma de la fusée et introduction des notations

On relie alors la tension mesurée dans un pont de Wheatstone avec la déformation explicitée ci-dessus. On définit :

- s : l'abscisse curviligne le long de la fusée
- $\vec{F}_t$: la traînée
- $\vec{F}_p$: la poussée du moteur
- $\vec{g}$: le champ de pesanteur terrestre supposé constant
- H : la hauteur de la fusée
- H_j hauteur à laquelle se situent nos jauges de déformation
- $\vec{a}$ l'accélération de la fusée

On constate ici que nos jauges ne devraient fonctionner qu'en compression.

Afin de pouvoir appliquer la méthode de la coupure comme sur un chargement statique, on va considérer une force volumique associée à l'accélération : $-\ \rho\vec{a}$

On coupe alors la fusée, considérée comme une poutre, au niveau des jauges de déformation pour exprimer l'effort normal :

$$N = F_t + \int_{H_j}^{H} -\rho\,(a+g)\ dV \quad = \quad -\left(1/2\,\rho_{air}\,AV^2C_x + m_{sup}(a+g)\right)$$

Mais aussi :

$$N = \quad -F_p - \int_{0}^{H_j} -\rho\,(a+g)\ dV = \quad -F_p + m_{inf}(a+g)$$

Avec :

- m_{sup}: la masse de la fusée au dessus des jauges
- m_{inf}: la masse de la fusée sous les jauges. Elle varie dans le temps du fait du moteur.
- C_x: le coefficient de traînée de la fusée donné dans le stabtraj
- A: le maître couple de la fusée
- V: la vitesse de la fusée

On remonte ensuite à la contrainte associée à cet effort :

$$\sigma = \frac{N}{S} = -\frac{1/2\,\rho_{air}\,S\,V^2C_x + m_{sup}(a+g)}{S} = \frac{-F_p + m_{inf}(a+g)}{S}$$

Où S est la section des 4 poutres en U, puisque nous supposons que ce sont elles qui reprennent tous les efforts.

On remonte ensuite à la déformation :

$$\epsilon = \frac{\sigma}{E} = -\frac{1/2\,\rho_{air}\,S\,V^2C_x + m_{sup}(a+g)}{SE} = \frac{-F_p + m_{inf}(a+g)}{SE}$$

En prenant le module d'Young de nos poutres qui sera déterminé expérimentalement sur un banc d'essai. Voir annexe.

On en déduit alors le déplacement :

$$U = \epsilon H_j = - H_j \frac{1/2\, \rho_{air} S V^2 C_x + m_{sup}(a+g)}{SE} = H_j \frac{-F_p + m_{inf}(a+g)}{SE}$$

On relie alors la tension mesurée dans un pont de Wheatstone avec la déformation explicitée ci-dessus.

De plus, grâce aux deux expressions provenant des deux côtés de la coupure, il sera possible de recalculer expérimentalement le Cx de la fusée pour le comparer à celui fourni dans le Stabtraj.

La formule permettant de calculer le Cx est :

$$C_x = - 2\frac{\varepsilon ES + m_{sup}(a+g)}{\rho S V^2}$$

4.1.3 Application des calculs

On prend les valeurs suivantes pour ces grandeurs :

- $E = 56\, GPa$ mesurée par PLUME (voir annexe)
- $S = 2,52\, 10^{-4}\, m^2$
- $m_{sup} = 4,82\, kg$ mesurée
- $m_{inf} = 4,87\, kg$ avec propulseur plein. L'évolution de cette masse est donnée par le Stabtraj.
- $g = 9,81\, m^2 s^{-1}$

En se basant sur la coupure à gauche (expression de N avec la poussée) on obtient alors la courbe prévisionnelle suivante pour la déformation :

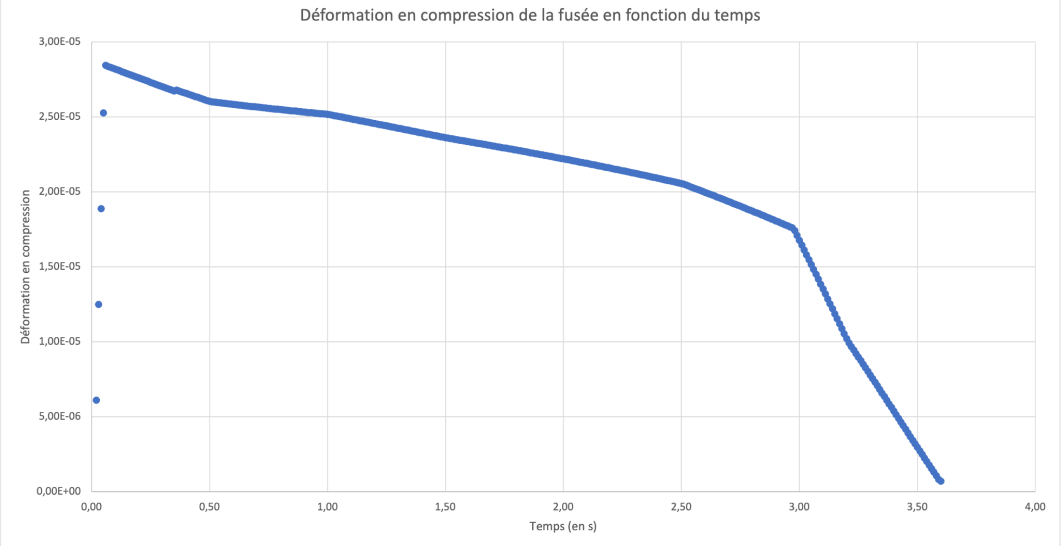


Figure 3.1 – Schéma de la fusée et introduction des notations

4.1.4 Critique du modèle

On a supposé ici que la fusée était une poutre ne travaillant qu'en compression. Cependant, la fusée ne vole pas verticalement, ainsi l'assiette de la fusée fait apparaître de la flexion par le biais du poids. De plus, la vitesse et l'accélération de la fusée ne sont pas exactement selon son axe longitudinal, ce qui là encore crée des moments de flexion. Enfin notre fusée n'est pas une poutre mais un assemblage d'une multitude de pièces, ainsi pour affiner les résultats, il serait judicieux de passer par un calcul éléments finis. Cependant ce type de calcul est très compliqué à mettre en place.

De plus, cette étude ne tient pas compte des modes de résonance en traction-compression de la fusée qui pourraient venir parasiter nos mesures. Une approche MEF semble nécessaire pour calculer ces grandeurs.

4.1.5 Utilisation des données

Nous allons comparer les mesures et les prévisions de déformation, pour nous permettre de valider notre modèle et nos hypothèses. Ceci nous permettra de confirmer notre capacité à mesurer une telle déformation et notre capacité à surmonter les difficultés perçues :

- Fréquences de résonance de la poutre : la fréquence d'échantillonnage a donc été choisie pour ne pas se faire brouiller (dans le pire des cas, une analyse spectrale nous permettra de supprimer le mode de résonance)

- Dilatation thermique des poutres : une seconde jauge de déformation perpendiculaire à la première est posée et intégrée au pont de Wheatstone pour annuler les dilatations thermiques

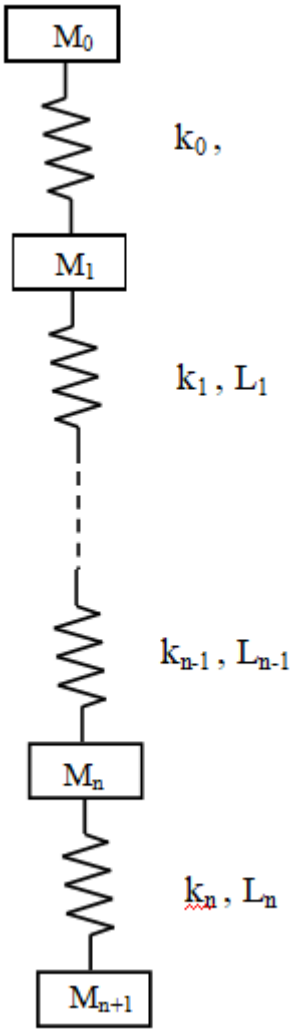
Pour finir, un logiciel a été préparé pour faciliter le post-traitement, il permet de préremplir un tableur Excel avec les données enregistrées par la fusée et les calculs que nous avons à réaliser (ceci permet un accès facile aux données et calculs en cas de besoin). Ce logiciel effectue de plus une double intégration des données d'accélération à l'aide de celles du gyroscope et du magnétomètre pour nous permettre de connaître la vitesse et la trajectoire de la fusée.

Ce logiciel est réalisé en Java, avec intégration d'un script Python pour la double intégration.

4.1.6 Calcul des fréquences propres de la structure

En supposant que la structure peut-être modélisée par une suite de masses reliées par des ressorts, N masses induisent N modes de vibrations. Ce résultat obtenu en 1D doit permettre de critiquer les résultats issus de l'acquisition électronique. Les ressorts

représentent les poutres et les masses représentent les bagues ainsi que les équipements fixés dessus. La nature de ces équipements n'est donc pas prise en compte dans les calculs. L'idée reste d'avoir des ordres de grandeurs pour savoir dans quel mode la fusée risque de résonner longitudinalement. Les modes de résonance en flexion et en torsion ne sont pas étudiés.



Chaque nœud possède une inertie et l'on peut donc appliquer le principe fondamentale sur chaque nœud. Les forces qui s'appliquent sur un nœud dépendent des deux nœuds adjacents. Pour un noeud j :

$$M_j \frac{d^2 x_j}{dt^2} = k_{j-1}(x_j - x_{j-1}) + k_j(x_j - x_{j+1})$$

Ce système de N masses possède N inconnues (leurs positions), il est possible d'écrire ce système linéaire sous forme matricielle. Les raideurs des ressorts sont calculés à partir de la théorie des poutres $k_j = SE/L_j$.S est la section de 4 ou 3 poutres selon j. E est le module d'Young de la poutre et Lj la distance entre la masse j et la masse j+1.

$$M\frac{d^2X}{dt^2} = AX$$

$$X = \left[x_0, x_1 \ldots x_N\right]^T$$

$$M = \begin{bmatrix} M_0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & M_N \end{bmatrix}$$

On en déduit que les pulsations de résonance de la structure globales sont les racines inversent des valeurs propres de la matrice $M^{-1}A$.

Mode	fréquence de vibration (Hz)
1	0
2	646
3	1008
4	1711
5	2312
6	3479
7	3779
8	5768
9	8428

Une configuration stable est bien entendue au repos. Les autres fréquences de résonances se situent au-delà de 500 Hz ce qui fixe bien la limite du système d'acquisition. Le calcul des valeurs propres est réalisé sur matlab en résolvant les “eigenvalues”.

4.1.7 Logiciel d’interprétation des données

Pour faciliter le traitement des données acquises par la fusée, il a été décidé de préparer un logiciel interprétant les données écrites par la carte embarquée à bord de la fusée.

Pour faciliter l'interprétation des mesures et permettre de modifier les formules de calcul rapidement et sans avoir besoin d'une grande connaissance du programme et/ou du langage utilisé, il a été décidé que les données seront exportées sur un fichier Excel.

Ce logiciel permettra donc de partir d'un fichier texte contenant les données de la fusée et de les importer dans un classeur Excel. Le point fort de ce logiciel est qu'il va permettre d'intégrer au classeur tous les calculs que nous avons envie de réaliser, ainsi que d'importer les différentes modélisations qui ont été réalisées par le pôle expérience.

Ce rapport détaillera premièrement le fonctionnement du logiciel ainsi que le format dans lequel les données doivent se trouver pour pouvoir être traitées. Il sera ensuite détaillé le fonctionnement interne du programme et la logique derrière sa conception.

Pour la création du fichier Excel ce programme utilise la bibliothèque 'Apache POI 5.0'. Finalement il est à noter que le logiciel est codé en Java et qu'une version de Java 8 est nécessaire pour l'exécuter.

Finalement, ce rapport détaillera les grandes lignes du projet de calcul de la trajectoire, les progrès réalisés, les problèmes rencontrés et la raison de son abandon.

4.1.7.1 Logiciel de post traitement

4.1.7.1.1 Utilisation

4.1.7.1.1.1 Utilisation du logiciel

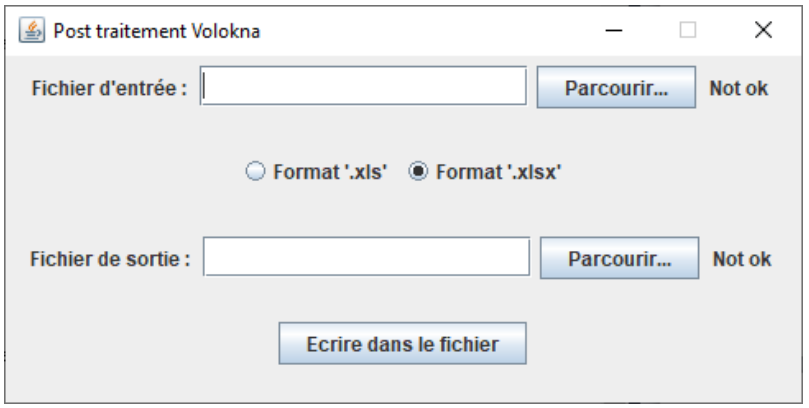


Figure 1 – Interface du logiciel

L'interface permet de renseigner :

- Le chemin pour le fichier exporté par la fusée : 'Fichier d'entrée'
- Le format souhaité de classeur Excel
- Le chemin où le classeur Excel sera écrit

Pour renseigner les chemins de fichier il y a deux options : rentrer/modifier le chemin manuellement dans la barre de texte ou utiliser le bouton 'Parcourir...'. Ce bouton ouvre une fenêtre de navigation qui permet de sélectionner un fichier (ou un dossier pour le fichier de sortie).

Les messages de texte à droite des boutons 'Parcourir...' indiquent si le logiciel reconnaît les fichiers pointés comme utilisables (s'il ne reconnaît pas un des chemins, les boutons en bas sont désactivés).

L'utilisateur peut choisir s'il veut exporter les classeurs Excel dans l'ancien format ou non. Il est recommandé d'utiliser le nouveau format, mais si un logiciel possédant d'une ancienne version d'Excel doit être utilisé il y a possibilité d'export selon l'ancienne norme. Il est à noter que la feuille de graphiques n'est créée que si le format '.xlsx' est choisi.

Le bouton 'Écrire dans le fichier' permet de calculer et d'écrire dans le fichier choisi. Dans le cas où le fichier de sortie existe déjà, une fenêtre apparaîtra au moment d'écrire et proposant trois choix :

Yes : Écrase le fichier
No : Ajoute des ’’’ à la fin du nom pour créer un nouveau fichier
Cancel : Annule l’écriture du nouveau fichier

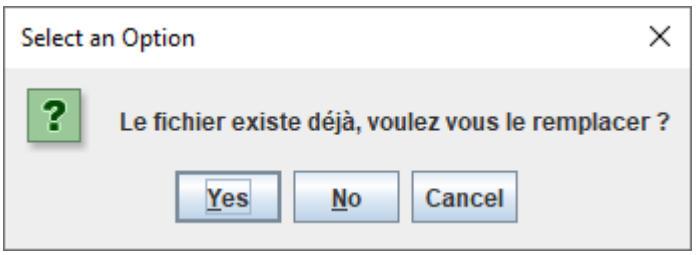


Figure 2 – Fenêtre apparaissant si le fichier de sortie existe déjà

4.1.7.1.1.2 Format des données en entrée

Le programme est capable de lire les données se trouvant dans un fichier texte qui doit respecter certaines règles de mise en forme :

- Toute ligne vide ou commençant par ’#’ est ignorée
- La première ligne contient le nombre de valeur prises par champ (nombre de pas de temps)
- Tout champ de donnée commence par une ligne avec son nom puis une ligne par valeur
- Le nom des différents champs sont : **`temps`**, **`jauge0`**, **`jauge1`**, **`jauge2`**, **`jauge3`**, **`jauge4`**, **`acc0`**, **`acc1`**, **`acc2`**, **`gyro0`**, **`gyro1`**, **`gyro2`**, **`mag0`**, **`mag1`**, **`mag2`** et **`alt`**
- La jauge 4 est la jauge supplémentaire (pas prise en compte pour calculer la déformation de la structure)
- Le séparateur décimal est ’.’ et il n’y a pas d’espaces dans les nombres
- Il n’y a pas d’ordre à respecter pour les champs

4.1.7.1.2 Classeur excel

4.1.7.1.2.1 Format général

Le classeur de sortie est composé de trois feuilles :

- Une première feuille contenant les constantes modifiables du programme et un récapitulatif des coefficients des polynômes utilisés dans la modélisation
- Une seconde page contenant les données importées du fichier de la fusée ainsi que les calculs réalisés (théoriques et expérimentaux)
- Une page avec des graphiques de comparaison entre les données expérimentales et théoriques

4.1.7.1.2.2 Feuille ’Constantes’

Les deux premières colonnes de cette feuille donnent les constantes modifiables du programme. Les noms des constantes donnent aussi l’unité dans laquelle la valeur est demandée, qui est toujours l’unité SI.

Les constantes modifiables sont :

- La constante de pesanteur
- La section d’une poutre en U
- Le module d’Young des poutres en U
- La masse au décollage de la fusée
- La masse au dessus des jauges dans la fusée
- La constante K des jauges **multiplié par** 10^6
- Le gain de chaque jauge (jauges 0 à 4)
- La tension d’entrée
- La tension de référence

Vient ensuite le récapitulatif des coefficients des polynômes utilisés pour les modélisations. Ils ne sont présents qu’à titre indicatif et modifier ces valeurs n’impacte pas les feuilles suivantes. Pour modifier les calculs de modélisation il faut modifier les formules dans les trois colonnes marquées de **Mo** dans la section suivante.

Modelisation de la poussee (a)	(b)	(c)	(d)	(e)	(f)
	0	8,69586E-06	897,084	58069600	-2,03E+10
	1	-2512,574	-193,977	-109913000	2,854E+10
	2	1670358	38,3436	86559500	1,606E+10
	3	-62924310	-9,14712	-36303000	4,52E+09
	4	1103789000		8551850	-6,36E+08
	5	-7221759000		-1072870	35790598
	6	-1466050000		56002,5	

Figure 3 – Lecture des coefficients d’une modélisation. En (a) le nom de la modélisation, en (b) les degrés des coefficients et en (c), (d), (e) et (f) les différents polynômes utilisés.

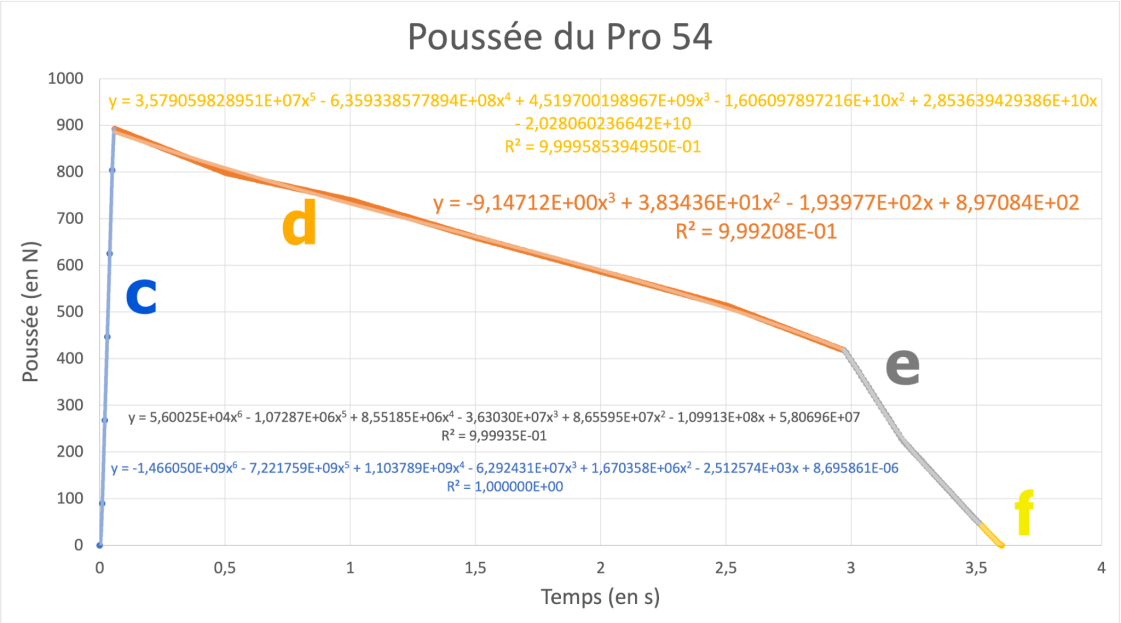


Figure 4 – Les données de poussé du Pro54 issues stab traj, chaque couleur représente un polynôme et les lettres correspondent

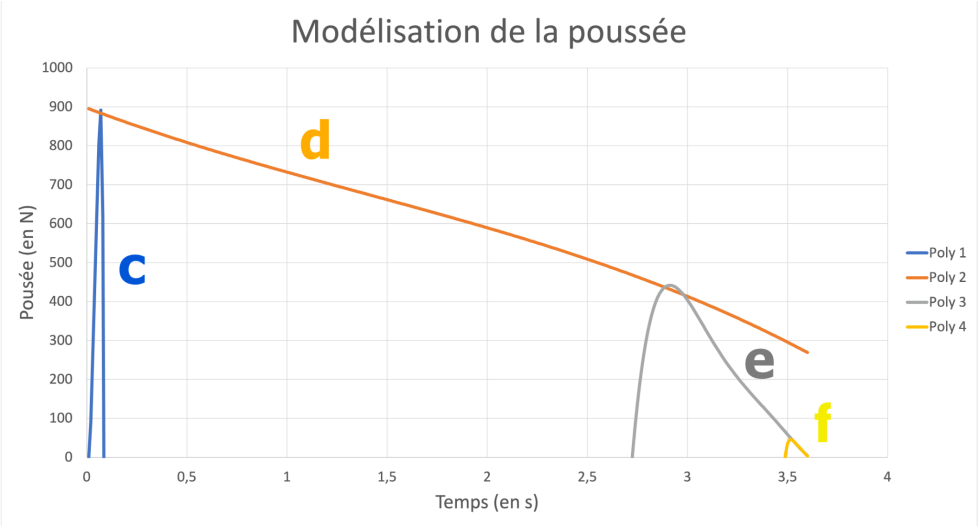


Figure 5 – Les différents polynômes de la modélisation de la poussée, chaque couleur représente un polynôme et les lettres correspondent

4.1.7.1.2.3 Feuille ' Données'

Cette feuille regroupe toutes les données expérimentales (mesures et exploitation) et théoriques (modélisations et prévisions).

La liste suivante récapitule les colonnes présentes et les formules utilisées. **Me** indique que la donnée est une mesure, **E** indique que la donnée est calculée depuis les mesures expérimentales, **Mo** indique que la donnée est issue des modélisations faites depuis le StabTraj et **P** indique que la donnée est une prévision théorique.

Me Temps *T* : instants de mesures donnés par la fusée
E Pas de tempsΔt
$\Delta t = Ti - Ti-1$
Me Tension aux bornes de la jauge 0 V_0
Me Tension aux bornes de la jauge 1 V_1
Me Tension aux bornes de la jauge 2 V_2
Me Tension aux bornes de la jauge 3 V_3
Me Tension aux bornes de la jauge 4 V_4
Me Accélération selon l'axe x a_x
Me Accélération selon l'axe y a_y
Me Accélération selon l'axe z a_z
Me Vitesse rationnelle selon l'axe x θ_x
Me Vitesse rationnelle selon l'axe y θ_y
Me Vitesse rationnelle selon l'axe z θ_z

Me Champ magnétique selon l'axe x μ_x

Me Champ magnétique selon l'axe y μ_y

Me Champ magnétique selon l'axe z μ_z

Me Altitude du baromètre *h*

P/Mo Masse *M*

$$M = M_{init} - \int \frac{dm_{propu}}{dt} dt = M_{init} - M_{perdue}^{modélisation}$$

P Masse en dessous des jauges M_{inf}

$$M_{inf} = M - M_{sup}$$

Mo Poussée moteur F_{propu}

Mo Angle par rapport à l'horizon θ

E/P Effort normal théorique N_{th}

$$N_{th} = F_{propu} - M_{inf}(a_x - g * sin(\theta))$$

E/P Contrainte théorique σ_{th}

$$\sigma_{th} = \frac{N_{th}}{Su}$$

E/P Déformation théorique ε_{th}

$$\varepsilon_{th} = \frac{\sigma_{th}}{E}$$

E Déformation vue par la jauge 0 ε_0

$$\varepsilon_0 = 4\frac{V_0 - V_{ref}}{K_0^{gain} V_{in} K * 10^6}$$

E Déformation vue par la jauge 1 ε_1

$$\varepsilon_1 = 4\frac{V_0 - V_{ref}}{K_0^{gain} V_{in} K * 10^6}$$

E Déformation vue par la jauge 2 ε_2

$$\varepsilon_2 = 4\frac{V_0 - V_{ref}}{K_0^{gain} V_{in} K * 10^6}$$

E Déformation vue par la jauge 3 ε_3

$$\varepsilon_3 = 4\frac{V_0 - V_{ref}}{K_0^{gain} V_{in} K * 10^6}$$

E Déformation vue par la jauge 4 ε_4

$$\varepsilon_4 = 4\frac{V_0 - V_{ref}}{K_0^{gain} V_{in} K * 10^6}$$

E Déformation moyenne ε_{exp}

$$\varepsilon_{exp} = \frac{\varepsilon_0 + \varepsilon_1 + \varepsilon_2 + \varepsilon_3}{4}$$

E/P Écart relatif avec les prévisions théoriques *err*

$$err = \frac{|\varepsilon_{exp} - \varepsilon_{th}|}{\varepsilon_{exp}}$$

4.1.7.1.2.4 Feuille 'Graphiques'

Cette feuille regroupe des graphiques de base. La bibliothèque utilisée ne permet de créer un graphique directement utilisable, il faut donc utiliser un des styles de base Excel pour les rendre rapidement exploitables.

Cette feuille contient trois graphiques, le premier comparant les tensions mesurées aux bornes des différentes jauges, le second comparant les déformations théoriques et expérimentales et le dernier traçant l'écart relatif en fonction du temps.

4.1.7.2 Essai de calcul de trajectoire

Lors des calculs théoriques nous nous sommes rendu comptes que nos mesures nous permettraient d'obtenir la vitesse et de calculer un coefficient C_x avec la formule suivante :

$$C_x = -2\frac{\varepsilon ES + m_{sup}(a+g)}{\rho S V^2}$$

Ce calcul impliquait cependant d'être capable de calculer la trajectoire à partir des mesures d'accélération, de gyroscope et de champ magnétique.

4.1.7.2.1 Algorithme AHRS

Dorian Dosjoub avait trouvé sur internet un algorithme permettant de calculer la rotation globale d'un objet grâce aux mesures que nous prenions.

La page GitHub dédiée donnait le code en C, avec un fichier d'exemple montrant l'utilisation du projet.

Nous avons donc codé un programme en C capable de lire le fichier et d'importer les données de la fusée à partir du même fichier, ce qui nous aurait normalement permis d'utiliser AHRS pour retracer la trajectoire avec une double intégration et donc d'avoir la vitesse en tout point.

N'ayant pour l'instant pas de données pour tester l'algorithme, il a été décidé d'essayer d'en créer avec des simulations. Les différents programmes et scripts créés ensuite ne permettaient pas de faire fonctionner l'algorithme.

Ci dessous se trouve le détail de ce qui a été réalisé et en conclusion le récapitulatif des avancement fait et des obstacles non surmontés.

4.1.7.2.2 Scripts Python sur Blender

Le second intérêt est de calculer la trajectoire et de pouvoir ensuite la visualiser en 3D. Blender étant capable d'exécuter des scripts python, il a été choisi.

Il ne restait plus qu'à coder deux scripts :

- Un permettant d'importer des positions et rotations et de les appliquer à un objet dans un environnement 3D pour visualiser la trajectoire
- Un permettant de partir d'une modélisation 3D et d'exporter les données mesurées par les capteurs de la fusée.

4.1.7.2.2.1 Export de simulation

Blender propose plusieurs systèmes pour décrire la rotation dans l'espace (Euler, quaternions et rotation autour d'un vecteur) et nous avons choisi d'utiliser la rotation Euler ZYX pour les calculs de projection.

Les trois angles donnés correspondent à, dans l'ordre : la rotation autour de l'axe X, la rotation autour du nouvel axe Y (noté Y') et la rotation autour du dernier axe Z (noté Z''). Ce système peut poser problème dans le cas où des axes deviendraient confondus, mais ce n'a pas posé de problème dans notre cas.

Nous arrivons finalement aux projections suivantes :

$$\begin{array}{rl} (x_f, y_f, z_f) & \text{Les coordonnées projetées} \\ (x, y, z) & \text{Les coordonnées globales} \\ \alpha & \text{La rotation autour de Z''} \\ \beta & \text{La rotation autour de Y'} \\ \gamma & \text{La rotation aoutour de X} \end{array}$$

$$\left\{ \begin{array}{rcrcrcr} x_f & = & \cos\alpha\cos\beta\, x & + & (-\cos\alpha\sin\beta\sin\gamma + \sin\alpha\cos\gamma)\, y & + & (\sin\alpha\sin\gamma - \cos alpha\sin\beta\cos\gamma)\, z \\ y_f & = & -\sin\alpha\cos\beta\, x & + & (-\sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma)\, y & + & (\sin\alpha\sin\beta\cos\gamma + \cos\alpha\sin\gamma)\, z \\ z_f & = & \sin\beta\, x & - & \cos\beta\sin\gamma\, y & + & \cos\beta\cos\gamma\, z \end{array} \right.$$

Un script Python a donc été implémenté, permettant de récupérer l'orientation et la position d'un objet 3D. Je dérivais ensuite la position pour obtenir l'accélération et la rotation pour obtenir la vitesse angulaire.

Nous projetions ensuite les données pour obtenir les rotations et accélérations mesurées par la fusée avant de les écrire dans un fichier.

Pour ce qui est de la simulation 3D, nous avions premièrement animé un objet pour lui faire suivre une trajectoire simple mais sans sens physique (car AHRS devait être capable d'en retrouver la trajectoire) avant d'implémenter une trajectoire réaliste (calculs de mécanique pour une phase propulsée et une chute libre) pour être capable de vérifier les valeurs aux différentes étapes de nos tests.

4.1.7.2.2.2 Importation des données

Un second script a été écrit permettant d'importer les données de position et de rotation globale dans une visualisation 3D.

Ce programme est plus simple et plus facile à tester (script précédent d'exportation sans les calculs de dérivation et de projection). Il ne posa donc pas de problème.

4.1.7.2.3 Programme en C

Le programme en C codé utilise le fichier **`ExampleAhrs.c`** trouvé sur le GitHub. Nous avons simplement implémenté une structure de données et des méthodes de lecture de fichier pour pouvoir charger les données depuis un fichier.

Nous avons aussi modifié l'utilisation de l'algorithme pour l'implémenter dans une boucle et avons ajouté des fonctions de projection et d'intégration pour pouvoir traiter les données.

La documentation indique que les angles d'Euler récupérés suivent la norme NWU (North-West-Up), ce qui a donc été pris en compte pour l'importation dans le script Blender.

La principale difficulté rencontrée a été le manque de compréhension de la méthode de fonctionnement de l'algorithme et du contexte d'utilisation prévu. Ceci nous a donc poussé à expérimenter à l'aveuglette pour essayer de comprendre son fonctionnement et obtenir des résultats cohérents. Ces tests ne nous ont cependant pas permis d'obtenir un ensemble cohérent.

4.1.7.2.2.4 Conclusion

Malgré tous ces problèmes, il reste quelques points positifs : certains programmes et scripts écrits fonctionnent comme prévu et peuvent être réutilisés dans l'avenir :

- Le script d'importation des données dans Blender. Il permet d'importer les données et de créer une animation du vol en plus de créer une parabole qui permet de visualiser rapidement la trajectoire.
- Les routines d'importation et d'exportation des scripts Python. Ces sous-scripts permettent de lire un fichier texte du même format que décrit dans la première partie.
- La structure de données et l'importation/exportation de données dans le programme C. De même, les méthodes permettant d'utiliser cette structure de données fonctionnent sans souci.
- La partie intégration du programme C. En effet, lors d'un de mes tests j'ai oublié d'implémenter la rotation dans la simulation. L'algorithme AHRS n'a donc pas modifié les données et l'intégration m'a donné une trajectoire très proche à celle exportée (à un facteur près). L'altitude étant mesurée dans la fusée j'ai pu rétablir une trajectoire correcte.

Les parties que je soupçonne de causer problème dans l'ensemble simulation/algorithme AHRS sont :

- Les vitesses angulaires. Les vitesses angulaires exportées sont les rotations d'Euler dérivées puis projetées, je ne suis pas sûr que cela corresponde réellement aux vitesses angulaires mesurées par le gyroscope.
- La dérivation dans le script de simulation sur Blender. En effet, nous avons utilisé des schéma centrés pour la majeure partie des données mais nous avons été obligé de calculer les première et dernière valeur de chaque avec des schémas excentrés. Les calculs de dérivés première ne posent pas de problèmes mais les dérivées secondes arrière et avant semblent diverger.
- L'utilisation précise de l'algorithme AHRS. Le fichier exemple propose l'utilisation d'une correction de l'erreur sur le gyroscope (qui permet de contrecarrer la dérive) et une phase de calibration (avec un gain variant) dont nous ne comprenons pas le fonctionnement (il faut peut être prendre des données sur la rampe avant le lancement). Finalement l'algorithme semble utiliser les données de l'accéléromètre pour trouver la pesanteur, il faut peut-être priver l'algorithme des données d'accélération pendant les temps de forte accélération.

4.2 Pôle électronique

4.2.1 Mise en situation

Nous rappelons que Volokna est une fusée expérimentale dont l'expérience est l'étude de jauges de déformations posées sur sa structure.

Le cahier des charges de planète science est très stricte en ce qui concerne la gestion de l'électronique dans la fusée. Chaque fusée doit comporter une carte électronique de séquençage disjointe de la carte expérience.

Les cartes ne doivent en aucun cas être en contact électriquement. En conséquence, leurs alimentations doivent être disjointes. Les communications se font donc par le biais d'optoélectronique. Le principe est de faire passer du courant dans une diode, qui active un phototransistor qui lui même fait passer du courant dans un deuxième circuit.

Nous détaillerons le fonctionnement de chaque carte de manière séparée.

4.2.2 Le Séquenceur

C'est le centre de la fusée : il doit détecter le décollage, allumer les différents systèmes en conséquence, s'assurer de la fenêtre idéale pour activer la récupération, effectuer, vérifier la bonne évolution du vol.

Les contraintes :

- Le séquenceur doit pouvoir rester allumé 2h en étant opérationnel - Doit activer la récupération dans un interval de temps [T1, T2]

- Disposer d'avertisseur lumineux témoignant de l'état du séquenceur Notre séquenceur pour des raisons pratiques, s'articule autour d'une carte programmable Arduino Mini.

La détection du décollage se fait par le biais d'une prise jack, qui rompt le contact entre deux fil en s'arrachant de la rampe de lancement. On utilise un pin Analogique pour lire l'état de la prise. La pin analogique est en pull up, et le jack est relié à la masse. Un buzzer jouant le rôle d'alarme est implémenté pour biper au cas où la batterie passe en dessous d'une certaine tension.

Un interrupteur positionné contre la trappe du parachute permet de vérifier que la trappe est bien vérouillée. Le système de détection est le même que pour la prise jack. Nous avons à l'heure qu'il est encore le choix entre deux moyens pour libérer la trappe parachute : par solénoïde ou par servo.

Pour l'instant seul la connectique pour un servo connecté en PWM est ajouté. On peut quand même piloter un solénoïde avec cette sortie si on fait passer le signal dans un transistor sans oublier la diode de roue libre (cf Rapport de Récupération).

Pour des raisons de modularité, il est prévu que le séquenceur puisse communiquer avec la carte expérience et un carte de télémétrie potentielle. La communication se ferait grâce aux ports RX TX de l'Arduino. Une sortie particulière est utilisée pour démarrer l'expérience.

Sans trop d'importance mais dans l'idée de faire les choses bien, le contrôleur de la RCE2 nous à proposé de mettre un petit condensateur à l'entrée de l'alimentation de la carte Arduino pour éviter les quelconques fluctuations de tension qui pourraient limiter son fonctionnement (à grande échelle).

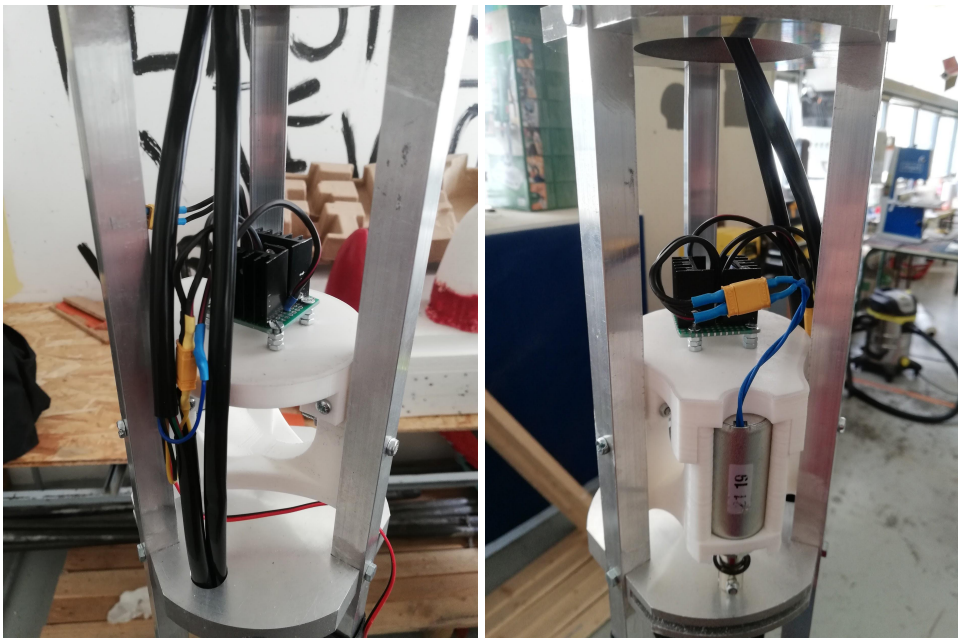
Nous avons aussi ajouté un écran 16X2 pour afficher la chronologie et les évènements importants. Il sera particulièrement utilise lors des séances de débugage.

A la date du 21/03/2020, deux séquenceurs ont été soudés, avec des plaques PCB commandées sur JLCPCB. La finition est impeccable, si l'un des séquenceur grille, l'équipe

en aura un de rechange. Ce séquenceur étant prévu pour être modulable, il devrait être utilisable pour les années à venir.

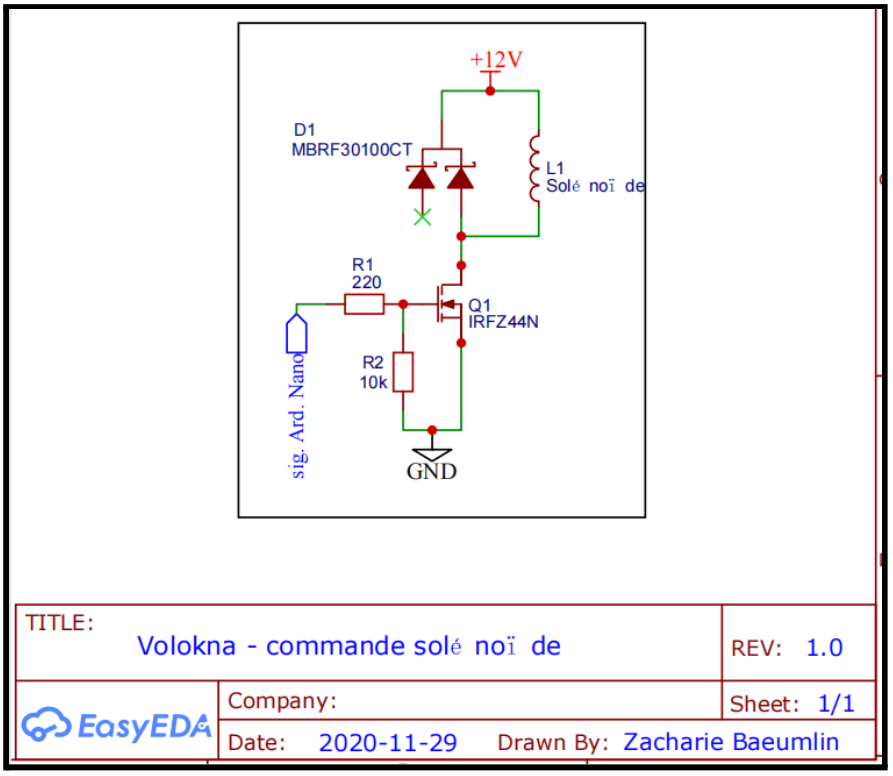
4.2.2.1 Le solénoide et son support

La carte de commande du solénoïde a pour but d'alimenter en puissance le solénoïde à partir d'un signal logique (5V ou 0V) venant d'une carte Arduino Nano.

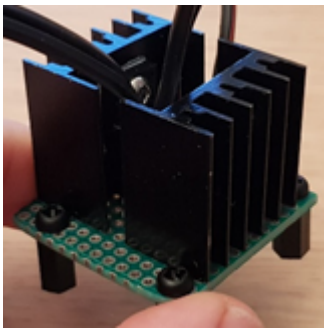


Un **transistor de puissance** est utilisé pour commuter l'alimentation du solénoïde. Il agit comme un interrupteur électronique qui va commuter un courant important à partir d'un signal logique.

Le circuit requiert un transistor MOSFET de type N, celui choisi a pour référence IRFZ44N. Son intensité maximale de drain est de 49A, bien supérieure à l'intensité traversant le solénoïde qui est de l'ordre de 1A. La tension maximale admissible entre le drain et la source est de 55V, ce qui est suffisant car la tension d'alimentation est de 12V (on décidera de l'augmenter à 24V car la force de rétraction n'était pas suffisante, ce qui reste dans la plage d'utilisation du transistor).



De plus, les éventuelles surtensions générées par induction par le solénoïde sont évitées grâce à la **diode de roue libre** de type Schottky de référence MBRF30100CT placé en parallèle du solénoïde. **Cette diode est indispensable pour éviter de générer de très hautes tensions lors de l'ouverture du circuit, elle est placée en parallèle de la charge inductive** (moteur, solénoïde, électro aimant, relais électromagnétique etc…). Sur le schéma on aperçoit 2 diodes car le composant utilisé en possède 2, ce qui est courant pour les diodes de puissance. Nous en utilisons qu'une seule.



Le transistor choisi est surdimensionné pour ce système, ce qui permet une faible dissipation de chaleur et de garantir une grande durée de vie de la carte. Néanmoins la diode et le transistor sont placés sur des **dissipateurs de chaleur** qui assurent une meilleure résistance mécanique et donc un plus faible risque de détérioration des soudures.

La résistance R1 permet de limiter l'intensité fournit par la carte Arduino lors d'une commutation. 5V/220 = 22.7mA < 40mA (intensité max sur une sortie de l'Arduino Nano). **Cette résistance est placée par prévention, si la carte arduino détecte une trop forte demande de courant, elle va s'éteindre ce qui peut avoir de graves conséquences sur le reste du système**, dans ce cas le séquenceur.

La résistance R2 est une **résistance de pull-down** qui permet d'assurer que le transistor est bloqué s'il n'y a pas de signal venant de « sig. Ard. Nano », ainsi que de décharger la grille du transistor lors d'un front descendant, cette résistance est indispensable.

4.2.3 La carte expérience

La carte expérience doit remplir toutes les caractéristiques imposées par la théorie de l'expérience. En effet la fusée est le vecteur de l'expérience, qui doit s'accorder à la théorie.

Dans le cadre de la métrologie, la fusée doit prendre des mesures suffisamment précises pour être interprétée. L'équipe de planète science insiste très lourdement, sur le

traitement de l'expérience post lancement, d'où l'intérêt d'avoir un système permettant de restituer fidèlement les informations du vol.

On va donc segmenter la carte expérience autour de plusieurs parties.

4.2.3.1 La partie alimentation

L'alimentation se fait avec une **batterie LiPo** de 3 cellules, ce qui donne un tension moyenne de 11,1V et une tension de fin de charge de 12,6V (une cellule Lipo a une tension moyenne de 3,7V et une tension maximum de 4,2V lorsqu'elle est chargée). Il y a une batterie pour le séquenceur et une pour l'expérience. Elles sont équipées d'un système de protection appelé **BMS**.

Nous avons décidé de faire fonctionner la carte expérience autour d'une Teensy 3.5 pour la multitude de fonctionnalités qu'elle peut proposer et sa capacité de calcul (c'est une Arduino plus puissante en quelque sorte). Il faut commencer par réguler la tension de la carte à 5V, tension utilisée pour tout alimenter. La partie amplification nécessite une alimentation très précise pour faire des mesures de qualité. Le contrôleur de la RCE2 nous a donc suggéré d'utiliser une référence de tension 5V pour obtenir une grande précision de tension sur un bus dédié. Comme pour le séquenceur, un condensateur permet de lisser la tension en entrée de la Teensy 3.5. Cette fois c'est un peu plus important pour perturber au minimum le processus d'acquisition.

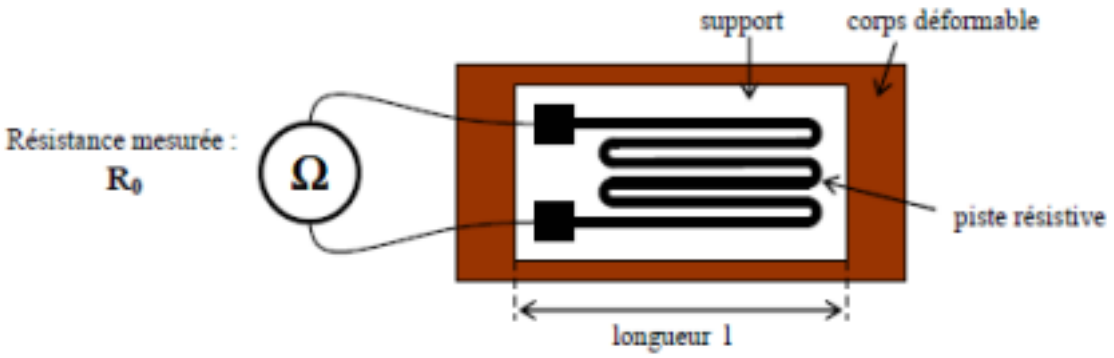
4.2.3.2 La partie acquisition

La partie acquisition se compose de plusieurs parties mais qui forment un tout :

$$\Delta e \rightarrow \Delta R \rightarrow \Delta V \rightarrow K(\Delta V + C)$$

L'acquisition doit permettre de remonter à la déformation à partir de la valeur de la tension lue par la carte Teensy.

Le passage de la déformation à une variation de résistance s'effectue à l'aide d'une jauge de déformation. Son étirement provoque l'allongement des fils qui la compose, ce qui induit une variation de résistance du tout.

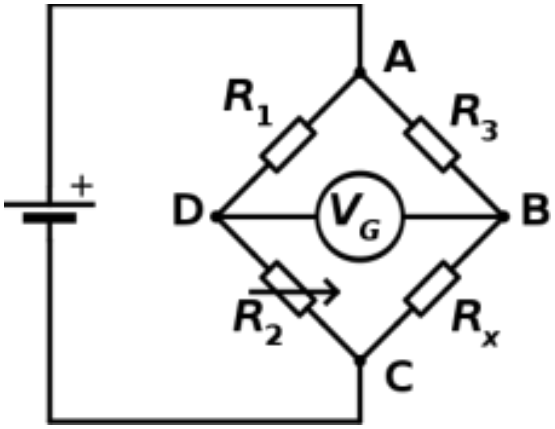


Nous avons la relation

$$\epsilon = \frac{\Delta R}{kR} * 10^{-6}$$

En sachant k la constante de jauge, égale à 2 dans notre cas. La résistance à vide de nos jauges est de 350Ω.

Nous pouvons remonter à un variation de tension en utilisant un pont de wheatstone. C'est un montage simple avec 4 résistances, normalement identiques. Le montage permet de créer une différence de potentiel fonction de la variation de résistance d'une des résistances.



Dans notre cas le pont est alimenté avec une tension de 5.000V (référence de tension suivi par un amplificateur opérationnel). Ce qui nous intéresse est la tension Vg. Si deux résistances en parallèle ont la même variation de résistance, les variations de tension se compensent. On peut alors rendre notre système indépendant fasse aux dilatations thermiques.

En effet, en plus de mesurer une variation de résistance dûe à la traction / compres sion on mesure

$$\varepsilon = \frac{\sigma}{E} + \alpha\Delta T$$

Avec la variation de résistance thermique associée.

On place donc une seconde jauge en parallèle de la première sur le pont de wheatstone, mais collée sur le même matériau (non sollicité), proche de la zone d'étude. Nous n'avons plus de problèmes de dilatation thermique, nous pouvons donc écrire :

$$\Delta V = V_{in} \frac{\Delta R}{4R}$$

Les autres résistances ne sont pas parfaites, on nous a conseillé de remplacer une des deux autres résistances par un potentiomètre. Celui-ci permettra de régler la tension Vg à zéro en l'absence de déformation, c'est la calibration des ponts.

A ce stade, une diminution de résistance (compression à priori) entrainera une tension négative sur la différence de potentiel.

$$\Delta V = 10^6 V_{in} \frac{k\epsilon}{4}$$

Les vibrations de la structure entrainent des phénomènes de déformations à haute fréquence de la structure, qui ne nous intéresse pas. On sait que notre fréquence d'échantillonnage sera de l'ordre de 1kHz, d'après le théorème de Shanon, il faut que notre signal ait une fréquence de 500Hz maximum. Il faut alors placer des filtre passe bas avec un fréquence de coupure de 500Hz max. On utilise pour cela un couple résistance 20kΩ / condensateur 100µF.

La difference de potentiel est mesurée avec un amplificateur instrumental. On développe ce point dans la partie suivante.

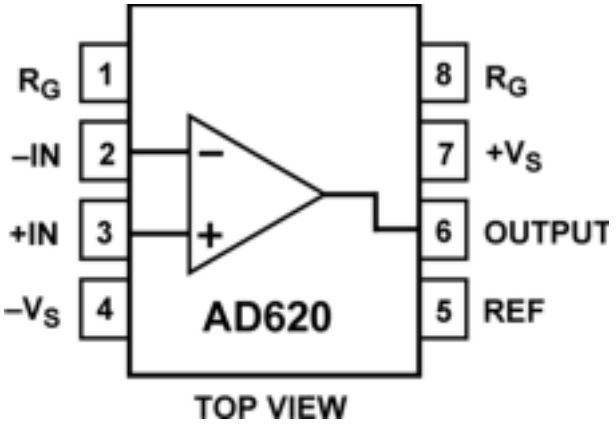
4.2.3.3 La partie amplification

L'amplification consiste à rendre le signal sortant des ponts avec assez d'amplitude pour être correctement échantillonné par la teensy.

Correctement échantillonné, cela signifie :

- à la bonne fréquence (pas au dessus de 500Hz)
- des tensions positives
- des tensions non supérieures à 5V

Notre outil est l'amplificateur AD620.



Les deux branches dont on veut mesurer la différence de potentiel y sont connectées. Le composant est alimenté en 0-5V.

La référence joue un rôle important car c'est autour de celle-ci va évoluer la tension de sortie. Une différence de tension nulle entraîne une sortie égale à la référence et une différence non nulle est amplifiée et rajoutée à la référence.

La constante d'amplification est obtenue par une résistance variable placée sur la carte. Chaque jauge peut alors être amplifiée de manière indépendante pour pouvoir mesurer des niveaux de déformation de diverses amplitudes. Par exemple on pourra peut amplifier une jauge pour avoir la réponse globale de la structure mais beaucoup amplifier les autres pour avoir une réponse plus particulière et précise au début et à la fin de la déformation.

La partie amplifiée est directement rentrée sur les pin analogiques de la teensy. On peut alors écrire l'équation de la tension lue par la carte :

$$V_{lue} = G\Delta V = G(10^{6} V_{in} \frac{k\epsilon}{4})$$

En pratique, Vref est précisément fixée à 2.500V.

4.2.3.4 La partie sauvegarde

La carte expérience ne fait actuellement pas appel à de la télémétrie, elle doit donc sauvegarder toutes les données du vol d'une manière sûre. On choisira de multiplier les supports de stockage. D'une part, on peut utiliser la carte SD de la teensy, d'autre part, on utilise des EEPROM pour les différentes jauges et données à enregistrer.

Pour le stockage il faut enregistrer :

- les déformations de chaque jauge
- les accélérations dans les 3 axes
- les vitesses angulaires dans les 3 axes
- les instants auxquels sont prises les mesures

La communication entre tous les composants (capteurs d'accélérations / gyro) s'effectue avec le protocole I2C. C'est un protocole qui nécessite seulement deux pin pour communiquer avec $2^3 = 8$ composants. Il y a un pin de clock et un pin data. C'est un protocole maître/ esclave qu'il est intéressant d'approfondir (mais pas ici).

4.2.3.5 La partie communication

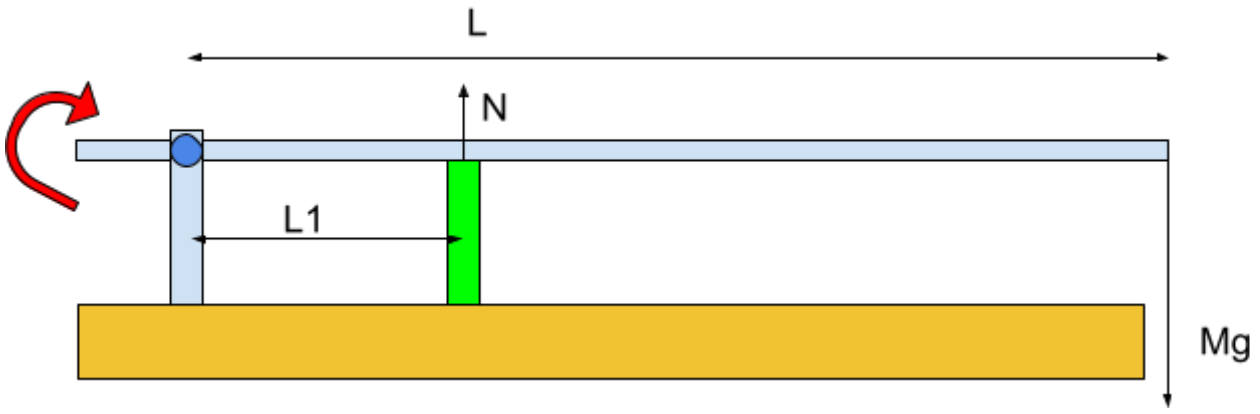
La carte expérience a la possibilité de communiquer avec le séquenceur par le biais de la communication RX / TX. Il n'y a pas grand chose d'autre a dire. L'allumage de la carte se fait par un interrupteur déporté sur un panneau de contrôle. Diverses led permettent de voir l'état de la carte :

— Power On
— Erreur
— Saving
— Recording

Cependant pour des raisons de sécurité, la carte experience n'envoie pas de données vers le séquenceur. Nous ne voulions pas risquer de bloquer la procédure de récupération pendant le vol. D'autre part, une quelconque communication entre les deux cartes doit être clairement motivée.

4.2.3.6 Etalonnage de la carte d'aquisition

Un banc d'essai à été conçu pour calibrer les jauges de la carte experience. Il s'agit d'appliquer un effort normal sur un morceau de poutre identique. Une jauge identique au autre a été collée dessus. L'idée est de réaliser un bras de levier pour imposer une large gamme d'efforts normaux.

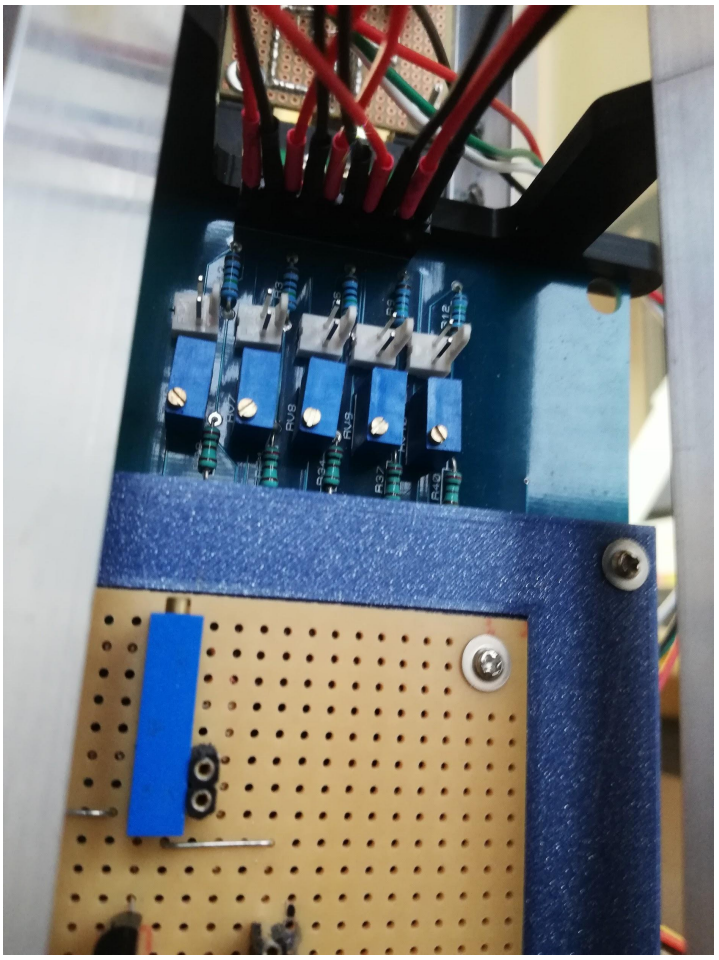


L’équilibre des moments sur le bras de levier donne

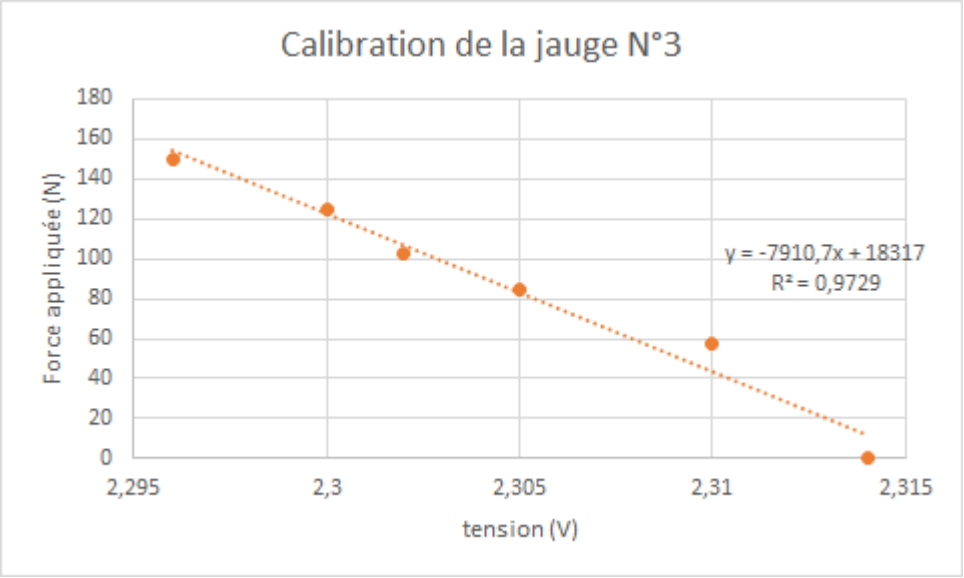
$$NL1 = MgL$$

D’où $N = \frac{MgL}{L1}$

Pour un réglage de gain donné, plusieurs efforts sont appliqués sur la poutre. La tension est mesurée avec un multimètre de précision aux bornes analogique de la carte teensy. Pour calibrer chaque voie une à une la jauge du banc d’essai est connectée aux autres voies d'acquisition. Pour faire cela il faut déconnecter les jauges de déformation de la carte experience.



Pour des raison développées dans les problèmes liés au déroulement du C'Space, la chaine d'acquisition n'est pas très bonne. L'amplification est assez capricieuse et le gain ne se règle pas bien. D'autre pas seule la jauge numéro 3 fonctionne. C'est pourquoi uniquement cette dernière est calibrée.



Nous remarquons plusieurs choses interessantes. D'abord pour un poutre au repos, une tension de référence est mesurée à 2.3154V. Cette valeur dépend du pin REF de l'AD620 car le pont de wheatstone a été annulé à 1 mV près avant la calibration.

D'autre part, la pente de la coubre de calibration peut servir dans le cas ou la référence change légèrement car elle n'en n'est pas dépendante. Il sera d'ailleurs possible de recaler cette référence avec la phase d'apesanteur pendant laquelle la fusée n'est soumise à aucune force volumique.

4.2.4 Intégration des cartes électroniques dans la fusée

4.2.4.1 Choix des batteries

Le cahier des charges impose l'utilisation de deux batteries différentes dans la fusée. Caractéristiques attendues des deux batteries qui seront identiques :

— Tension d'alimentation de 12V

— Délivrer un courant moyen de 100 mA pendant 1h

N.B.: le choix de la tension d'alimentation doit être imposé par le circuit de puissance, dans notre cas le solénoïde qui, au départ, devait s'alimenter en 12V. Ce choix implique de mettre, si besoin, des régulateurs sur les circuits de faible puissance, mais permet d'éviter de placer des convertisseurs de puissance qui sont encombrants, plus onéreux et qui entraînent des pertes de puissance non négligeables.

Nous choisissons des accumulateurs **LiPo** (Lithium-ion Polymères) qui offrent une **forte densité d'énergie électrique**. Elles ont aussi l'avantage d'être stables ce qui permet de les **conserver longtemps** sans qu'elles ne se dégradent (donc on peut les conserver sans problème de l'achat jusqu'au lancement), contrairement aux piles 9V. Pour le **stockage**, l'idéal est de les laisser à mi-charge, d'ailleurs le chargeur intègre une fonction "storage" pour cela.

On peut donc se permettre de surcalibrer les batteries sans qu'elles ne soient trop encombrantes ni trop lourdes. On choisit une **capacité de 1Ah** par batterie. Cette capacité offre au séquenceur une autonomie d'environ 18h et à l'expérience environ 8h. Ces autonomie sont calculées à partir d'une mesure de consommation lors de simulations de vol. C'est bien supérieur à ce qu'il faudrait pour le lancement, prévoir 2h d'autonomie minimum en cas d'attente sur le pas de tir.

Ces batteries restent dangereuses lors du chargement et de leurs opérations. Une déformation mécanique peut percer leurs membranes et mettre le feu à la fusée. L'autorisation de l'utilisation de ces batteries s'est prononcée en décembre 2020.

Il reste alors quelques conditions d'utilisation :

— Déconnexion facile des batteries

— Bon état physique et d'origine

— Stockée dans un sac ignifugé

— Attache propre dans la fusée

Chaque cellule LiPo délivre une tension moyenne de 3,7V. Il faut donc un accumulateur composé de 3 accus LiPo pour obtenir une tension moyenne de 11,1V. La tension de fin de charge sera de 4,2V*3 = 12,6V. **Tous les composants alimentés directement par la batterie devront bien fonctionner sur cette plage de tension** : 9,6V à 12,6V. Si ce n'est

pas le cas, il faut ajouter des convertisseurs pour adapter la tension, dans notre cas des régulateurs linéaires 9805 permettent de fournir du 5V sur les cartes **(à noter que les régulateurs linéaires sont adaptés pour les applications de faible puissance !)**.

Un **BMS** (Battery Management System) est indispensable pour protéger chaque cellule contre la **décharge profonde** (tension inférieure à 2,90V), la **surcharge** (tension supérieure à 4,225V), et les **courts-circuits**. Il faut un BMS 3S (car chaque batterie possède 3 cellules) et qui puisse supporter une intensité suffisante. La consommation est inférieure à 1A pendant la majeure partie du fonctionnement de la fusée, elle augmente de quelques ampères lors de la libération du parachute, où le solénoïde est alimenté.

Les composants choisis :

- 2 accumulateurs LiPo 3S (3 cellules), 1Ah, 70C (peut délivrer en théorie 70A maximum)
- 1 accumulateur LiPo de secours 3S, 1Ah, 75C
- 3 BMS pour accumulateur LiPo 3S, délivrant 20A maximum
- Un chargeur de 80W (6A max) pour accumulateur LiPo, permettant de recharger de manière « balancée »
- Un sac ignifugé pour le transport et la conservation des accumulateurs
- Des connecteurs de type XT30 pour assurer un raccordement électrique fiable et robuste

Photo d'un BMS et des fils gainés reliés aux connecteurs

Lors de la réception des batteries, on leur fait subir une **phase de test** pour vérifier qu'elles fonctionnent correctement. Malheureusement, les vendeurs chinois envoient parfois des accumulateurs défectueux…

Les tests que l'on peut faire sont :

- vérifier la tension de chaque cellule (avant d'essayer de recharger), si la tension descend en dessous de 2,9V pour un élément, alors l'élément est défectueux.
- recharger les accus (charge balancée) et voir si en fin de charge chaque cellule a atteint environ 4,2V.
- décharger la batterie (fonction "discharge" du chargeur), si un seul élément s'est déchargé (tension bien inférieure aux autres), alors il est défectueux. Et vérifier la capacité donnée par le chargeur, elle doit être proche de celle donnée par le fabricant.

Résultat des tests :

Batterie 1 : tensions ok → charge balancée complète à 1A → tensions ok → décharge complète jusqu'à 8.85V → charge de stockage à 1.8A jusqu'à 11.40V
La décharge a donné 978 mAh, ce qui est proche des 1000 Ah donnés par le vendeur.

Batterie 2 : tensions ok → charge balancée complète à 1A → tensions ok → décharge complète à 0,5A jusqu'à 8.92V → charge de stockage à 1A jusqu'à 11.40V
La décharge a donné 952mAh, ce qui est proche des 1000 Ah donnés par le vendeur.

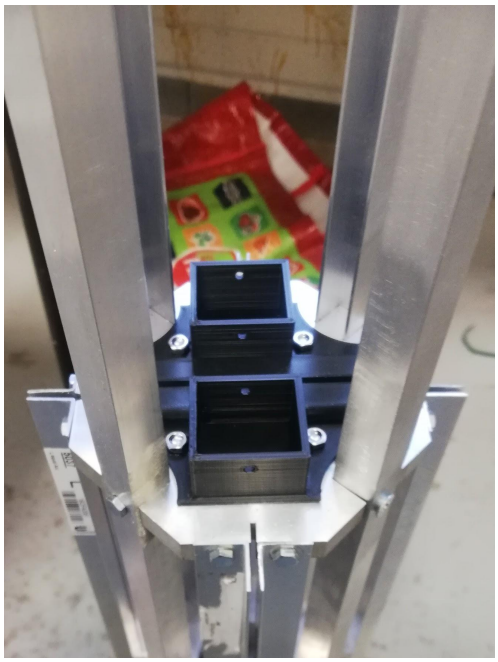
Batterie 3 (batterie de secours) : tensions ok → charge balancée complète à 1A → tensions ok → décharge complète à 0,5A jusqu'à 8.92V → charge de stockage à 1A jusqu'à 11.40V
La décharge a donné 874mAh, ce qui est proche des 1000 Ah donnés par le vendeur.
Cette batterie s'est révélée défectueuse après quelques utilisations durant des tests de fonctionnement où le BMS a coupé l'alimentation car un élément était trop déchargé. Les mesures de tensions ont révélé que seul cet élément s'est déchargé et pas les autres. Ce problème a permis de vérifier le bon fonctionnement du BMS.

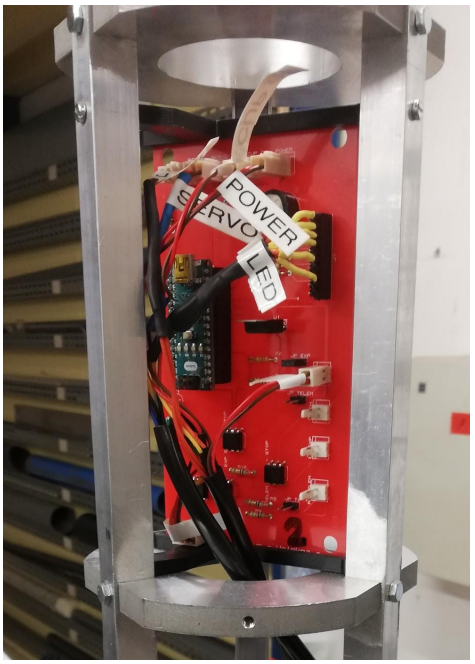
Un accumulateur déclaré défectueux par ces tests n'est pas inutilisable, cela veut dire que sa capacité a diminué et qu'il peut fournir moins de puissance.

4.2.4.2 Fixation des cartes sur la structure

Pour des raisons de simplicité et pour avoir un grand accès aux deux cotés des cartes il a été décidé de ne pas faire de rack d'éléctronique. C'est une compétence que le club devrait développer puisque finalement il est plus simple de travailler sur des cartes hors de la fusée lorsque c'est possible.

Dans notre cas les cartes sont prises en sandwich entre deux pièces imprimées en 3D. Une fente sécurise les cartes et empêche tout mouvement.

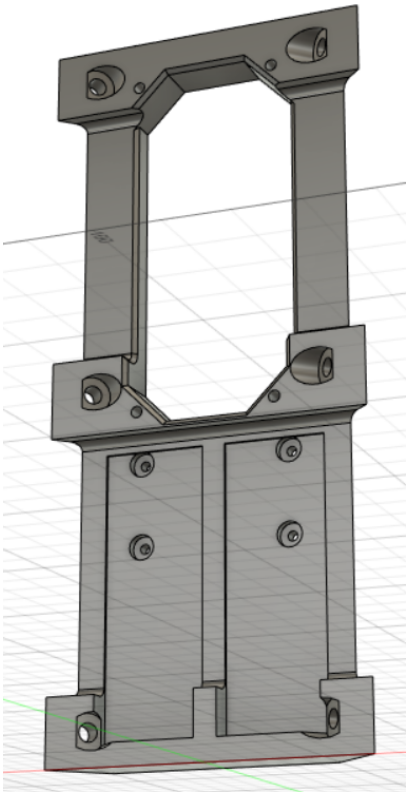


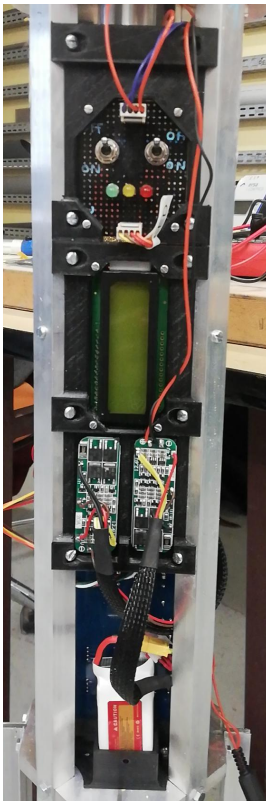


Pour la carte experience le support du bas contient aussi deux emplacement pour les batteries.

4.2.4.3 Intégration de la connectique et des interfaces utilisateur

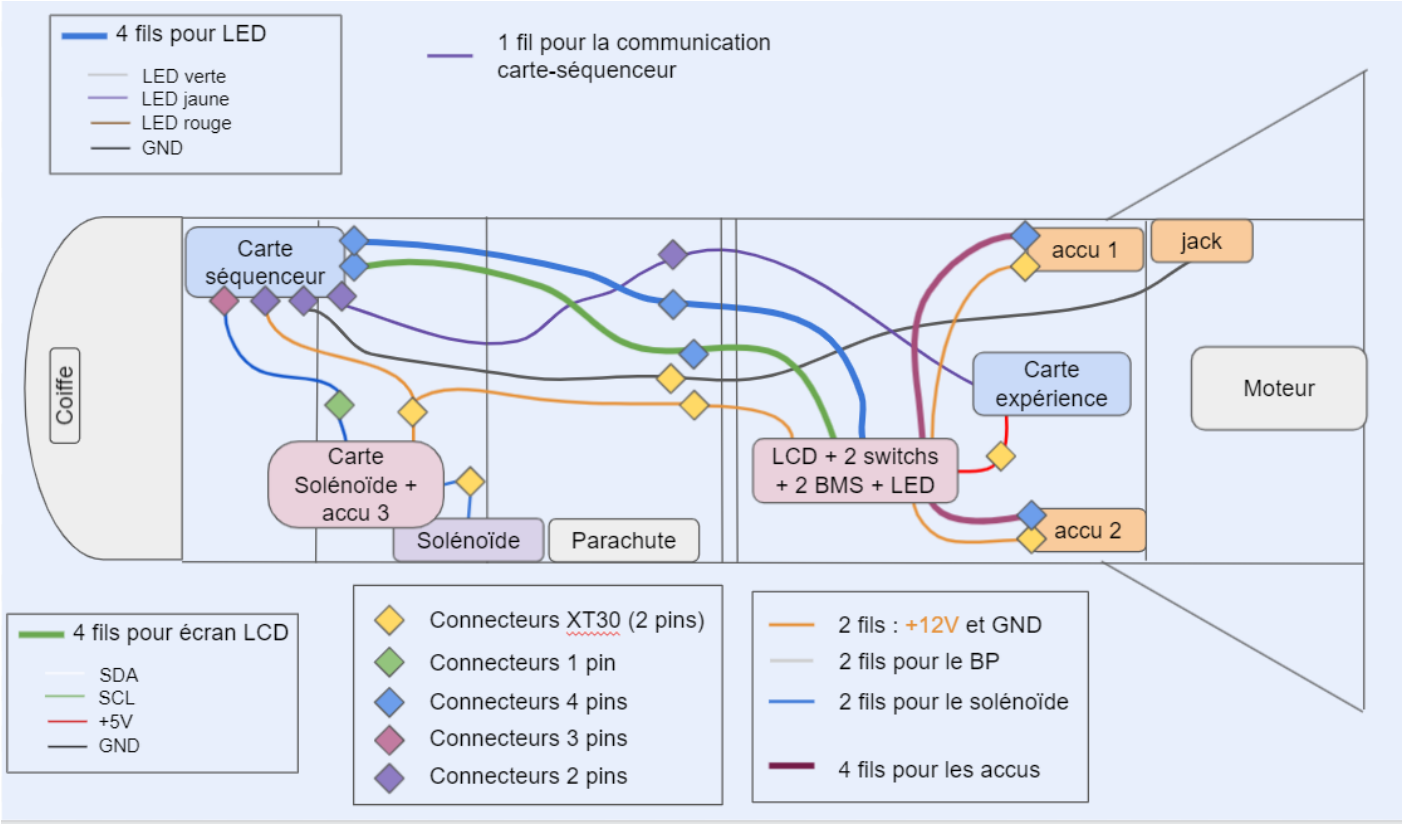
Le panneau de contrôle situé sur la partie basse de Volokna est le seul moyen de communiquer avec la fusée. Il contient les deux interrupteurs (expérience et séquenceur), le panneau LCD du séquenceur et les BMS. Sur son dos, la centrale inertielle y est attachée.



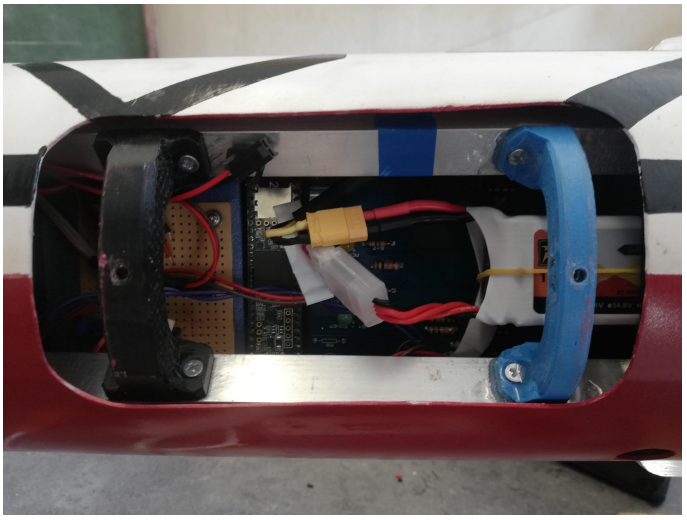


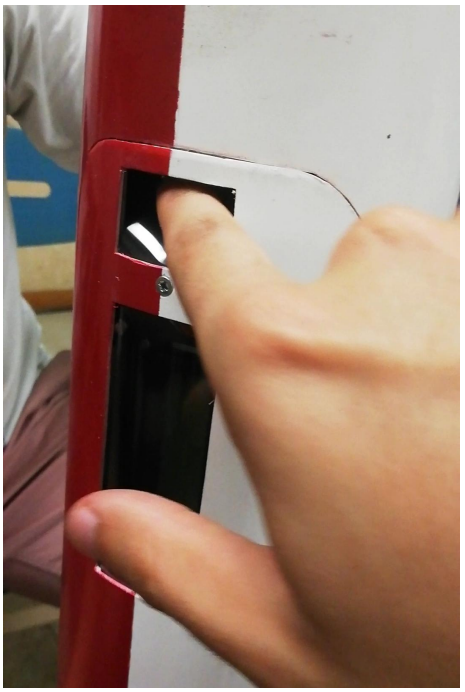
Ces supports sont vissés dans les poutres ce qui confère une très grande rigidité au panneau de contrôle. D'autre part, la forme du panneau ne peut pas gêner l'insertion du tube de peau.

Les liaisons électriques entre les différents modules impliquent un câblage assez conséquent (voir schéma ci-dessous). Les fils sont protégés par des gaines supplémentaires. Les connecteurs XT30 offrent un maintien mécanique robuste et un contact électrique très fiable (plaquage or).



La fermeture des compartiments pour accéder aux batteries, à l'écran LCD et aux switchs se fait par des trappes découpées dans le PVC à la dremel, qui sont vissées sur des pièces imprimées en PLA. Ces pièces sont vissées et collées sur les U en alu.



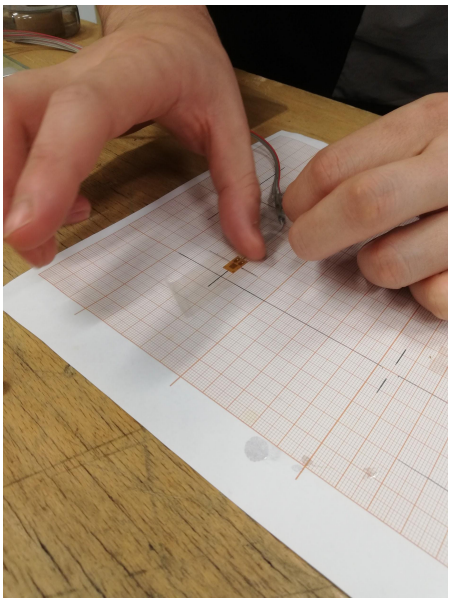


On a finalement ajouté une vitre à la trappe LCD pour visualiser l'état de la fusée une fois les trappes fermées, et une autre vitre à moitié attachée permettant d'accéder aux switchs (cf photo ci-dessus).

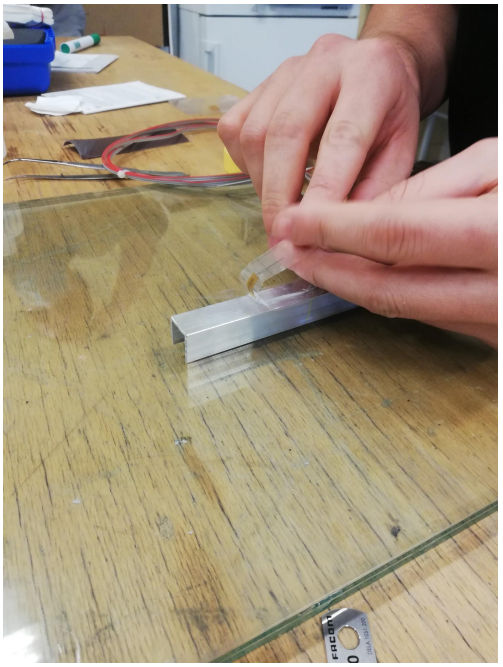
4.2.5 Collage des jauges de déformation

Les jauges de déformations sont collées à même le métal. Toute une procédure permet l'adhérence et la tenue mécanique de la jauge. Tout commence par un nettoyage de la surface avec de l'alcool et un papier abrasif. Il y a deux types de produit, un au début et un pour la finition du nettoyage.

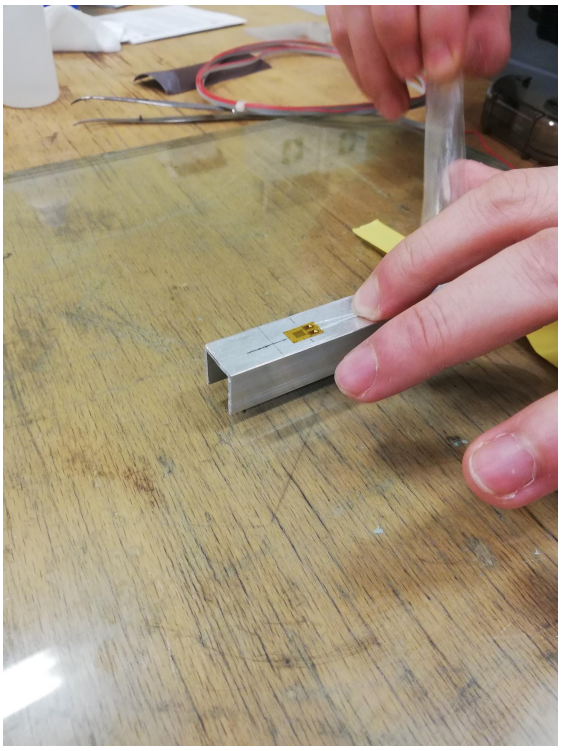
Ensuite la jauge est marquée et placée sur du scotch. Les marques permettent un alignement de la jauge avec la poutre.



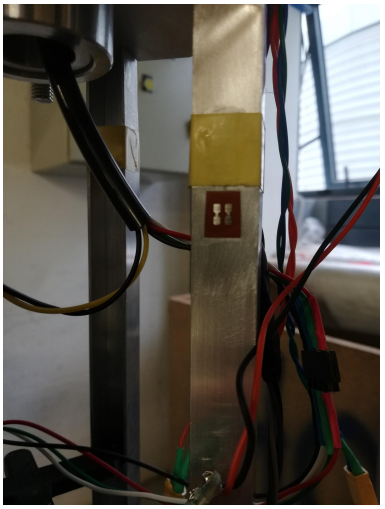
De la colle spéciale est déposée sur la poutre du côté de la base de la jauge et celle-ci est pressée contre la poutre à l'aide du scotch pour y chasser toutes les bulles. Il faut maintenir le doigt très fort pendant quelques instants (30-60s) pour que la colle se polymérise sous la chaleur du pouce.



Une fois cette étape réalisée, il faut patienter 10 min puis retirer le scotch en le tirant vers l'opposé du coin pour ne pas risquer le décollement de la jauge.



Pour les cosses la même procédure est appliquée. Les jauges thermiques sont traitées de la même manière que les jauges mécaniques. Les cosses servent à protéger la jauge au cas où un câble s'arrache.



4.3 Pôle programmation

4.3.1 Introduction

Ce rapport a pour but d'expliquer le cheminement suivi afin de partir d'un code blanc pour arriver à un résultat final satisfaisant et répondant au cahier des charges que nous nous somes fixés.

Sur le long chemin jusqu'au résultat il est tout à fait normal de rencontrer des phénomènes inattendu et parfois décevant, aussi appelé *bug*. Pour les générations futures, il est intéressant de revoir ensemble les difficultés rencontrées et surtout la manière de les résoudre. Bien sûr, c'est en communiquant avec les différents acteurs du projet que des problèmes simples peuvent être évités. Globalement il n'y a pas de solution miracle, beaucoup de temps et de patience arriveront au bout de beaucoup de désordres binaires.

4.3.2 Choix sur la structure de la carte et conséquences

4.3.2.1 Teensy

Nous avons choisi d'utiliser une Teensy pour ce projet pour des questions de rapidité d'exécution et taille réduite et de la nombreuse rangée de pin qu'elle possède. Cependant nous ne sommes pas familier avec cette carte à l'ESP. Cette dernière peut être assimilée à une super arduino. Le code arduino peut être compilé dessus la plupart du temps (cf les bugs...).

On ne peut pas directement connecter une teensy sur l'IDE arduino, il faut installer Teensyduino. C'est une sorte de complément à l'IDE arduino, qui comprend la spécification des différentes teensy et quelques exemples d'utilisation. Après avoir installé ce logiciel, tout se passe comme si la teensy était une arduino. On retrouvera un setup et une loop exécuté à l'infini. Nous avons une teensy 3.5, celle-ci possède un port USB ce qui facilite l'importation d'un programme. Dans le cas contraire, il faudrait utiliser un bus SPI, je n'en ai jamais utilisé mais le protocole consiste à passer la programme avec un plus bas niveau. En gros, une autre carte de type arduino sert de programmateur, et est connectée d'un côté en USB et de l'autre en SPI.

4.3.2.2 Interface et connectique

Le lien entre l'électronique et la programmation se fait par le biais des interfaces pour connecter les différents composants : on doit savoir quel pin correspond à quel branchement. Un des défauts majeurs de cette carte est qu'elle ne respecte pas les standards que l'on pourrait attendre ce genre de carte. En effet les prises I2C (en l'unique prise) présentes les pins SDA, SCL, Vin, GND dans un sens aléatoire, et sans l'avoir inscrit sur le PCB, c'est embêtant pour savoir comment connecter les fils sans faire de bétises. Il en va de même pour l'alimentation, mais ce n'est plus le problème du développeur. Un autre défaut que l'on pourrait reprocher serait qu'il n'y a qu'un seul port de communication avec la teensy en plus des ports RX, TX. C'est dommage au cas où nous voudrions connecter plus d'appareils. Notre carte fournit beaucoup d'entrées sorties, même si toutes ne vont pas être utilisées, les placer en attente avec des connecteurs est une bonne chose.

4.3.2.3 Bus de communication

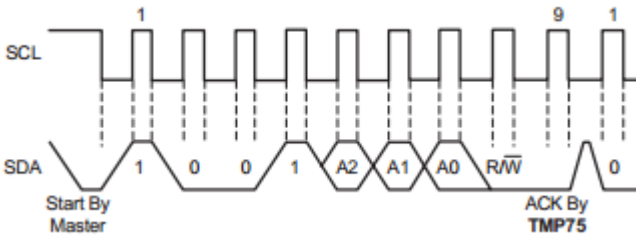
Comme énoncé dans le rapport d'électronique, nous utilisons de manière préférentielle la communication I2C, aussi appelée TWI (Two Wire Interface). Comme son nom l'indique, seulement deux fils en plus de Vin et GND suffisent pour communiquer avec beaucoup de composants. La plupart des accessoires fonctionnent avec ce protocole de communication, afin de ne pas se limiter dans les choix de ces derniers, il convient de choisir le protocole qui va avec. Comme vous avez pu l'avoir vu en Système Embarqué, il existe d'autres protocoles qui serait intéressant d'implémenter dans une fusée comme le CAN, mais la difficulté semble aussi plus importante.

Le principe du protocole I2C est assez simple dans la pratique. Dans notre cas, il faut s'imaginer la présence d'un maître (la Teensy) et des esclaves (les EEPROMS, la centrale . . .).

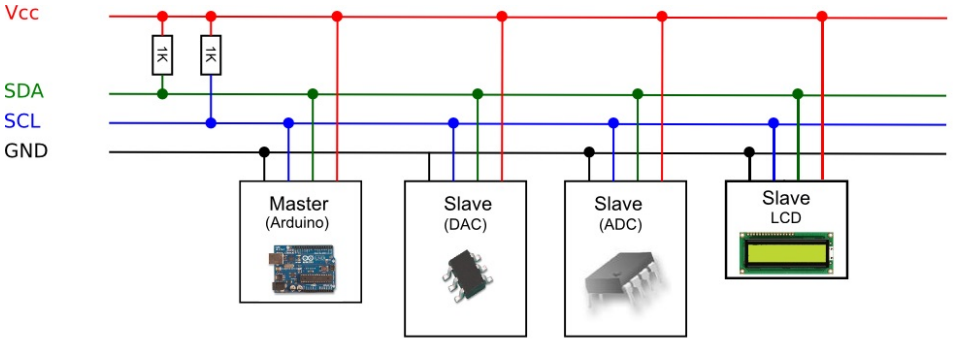
- Les esclaves ne peuvent pas prendre la parole sur le bus quand ils veulent. — Seul le maître parle quand il veut

- Chaque esclave possède une unique adresse codée sur 8 bits. (Souvent 7 bits définitifs et 1 bit pour savoir si on lit ou écrit)

— Tous les esclaves sont relié sur la même paire de fils SDA, SCL



Les signaux sur SDA et SCL sont des signaux logiques, SDA est cadencé par l'horloge SCL, qui fixe donc le débit de données sur le bus.



Exemple de bus I2C

Avoir 2 fil pour transmettre des données c'est très pratique, surtout pour celui qui conçoit le PCB. D'autre part, côté programmation, les librairies sont plutôt simples d'utilisation quand tout fonctionne bien.

On peut observer un petit bout de code sur la communication I2C :

```
#include <Wire.h>
void setup()
{
  Wire.begin(); // join i2c bus
(address optional for master) }
4.3 Pôle programmation 25


byte val = 0;

void loop()
{
  Wire.beginTransmission(44); // transmit to device
                              #44 (0x2c) //
                              device address
                              is specified in
                              datasheet
  Wire.write(byte(0x00)); // sends
  instruction byte Wire.write(val); //
  sends potentiometer value byte
  Wire.endTransmission(); // stop
  transmitting
```

On distingue bien toutes les phases du protocole :

1. On initialise le bus avec begin

2. On se concentre sur l'objet désiré en lui envoyant son adresse (0x2c en hexa) 3. On lui envoie les données nécessaires, par octets !

4. On termine la transmission!

Il y a quand même quelques défauts notables, mais qui ne sont pas directement liés au protocole mais plutôt à l'expérience d'un utilisateur débutant. La teensy possède plusieurs ports I2C, noté de 0 à 2. On imagine bien que sur chaque port, nous pouvons adresser 2^7 composants. Il reste en théorie à savoir comment changer de port rapidement.

D'autre part, certains composants, c'est le cas des EEPROMS, ont leurs 4 premiers bits réservés, il ne reste donc que trois bits d'adressage, souvent noté A0, A1, A2. Cela donne au maximum 8 EEPROMS sur un bus. Il faut y penser si le projet nécessite plus de mémoire.

Je n'ai pas eu ce problème mais il faut aussi veiller à utiliser des composants compatibles ! Il est impossible d'avoir deux esclaves avec la même adresse, et souvent il est impossible ou très difficile de modifier cette adresse. C'est la force et le défaut du protocol I^2C.

4.3.3 Programmation du système embarqué

4.3.3.1 Problématique de la mémoire

Notre fusée doit enregistrer beaucoup de données pendant son vol, mais pas n'importe comment. Les données les plus intéressantes correspondent sûrement à la déformation des jauges installées. Il faut de base comprendre que le problème de la place se pose si l'on veut une grande fréquence d'échantillonnage pendant un long lapse de temps.

Pour l'instant la fusée sauvegarde :

— Les valeurs des 5 jauges (int de 0 à 1023)

— Les trois composantes du gyroscope (float)

— Les trois composantes de l'accéléromètre (float)

— Les trois composantes du magnétomètre (float)

— L’altitude pression (float)

— Le temps à la mesure (int)

Ces données sont des flottants et entiers. Un flottant est représenté sous 32 bits, 16 bits pour un entier int.

Chaque instant sauvegardé tient une place.

M = 16 x (5 + 1) + (3 + 3 + 3 + 1) ∗ 32 = 416 bits = 52 octets

On souhaite sauvegarder environ 3,2 s du vol, temps pendant lequel le moteur est allumé.

Concentrons nous sur la mémoire disponible. Il faut se pencher sur la mémoire utilisée par les variables. Au passage nous avons aussi déclaré des constantes, notamment pour les pins.

Les constantes sont des valeurs qui ne changeront pas pendant l’exécution du programme. Ce sont des valeurs fixes stockées dans la mémoire de donnée de l’Arduino. Ces valeurs ne surchargent pas le programme et leur stockage est optimisé par le compilateur de l’IDE Arduino. Donc si vous avez une constante à déclarer, ne la déclarez pas en variable.

Il existe 3 moyens différents de déclarer des valeurs constantes :

— Avec le mot-clé const. C’est la manière la plus recommandée pour déclarer une constante dans un programme.

— Avec le define. La constante sera ici prise en charge par le préprocesseur. — Avec le mot-clé enum. Ici enum est très pratique pour déclarer des constantes de noms et non de valeurs.

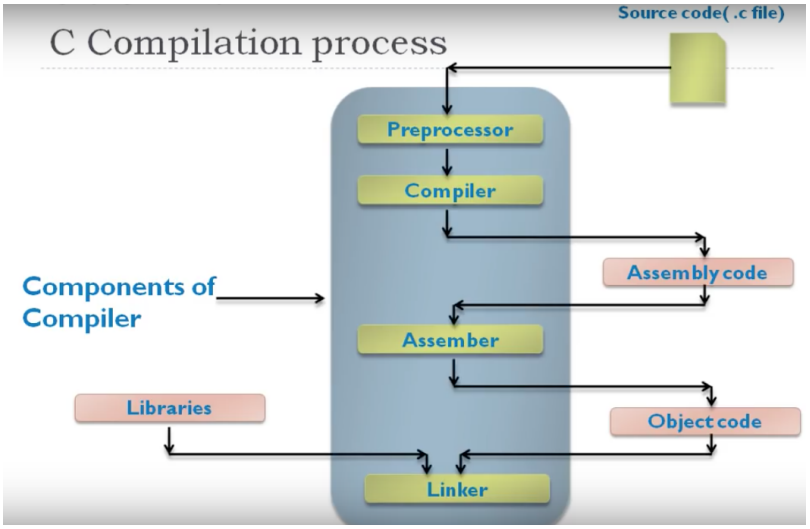
Plusieurs types de mémoires sont disponibles sur la teensy comme en atteste le tableau de caractéristiques suivant.

Microcontrôleur	
Nom:	Teensy 3.5
Marque:	PJRC
Caractéristiques	
CPU:	ARM Cortex-M4
Tension d’alimentation :	3.6-6V
Tension logique:	5V
E/S digitales:	64
Entrées analogiques:	27
Flash:	512kB
SRAM:	256kB
EEPROM:	4kB
Fréquence d’horloge:	120 MHz
Wifi:	No
Bluetooth:	No
SD card:	Yes
Touch:	No
UART/SPI/I2C/I2S:	Yes/Yes/Yes/Yes

Les variables dynamiques sont stockées en RAM, notre teensy 3.5 dispose de 256 ko. On réserve 2 ko pour les autres variables. On peut alors en utiliser 254 ko. Avec 52 octets par instant on peut sauver 254 000 / 52 = 4884 instants, pendant 3,2 s cela correspond à une fréquence d'échantillonnage de 1,5 kHz à ne pas dépasser. Pour des raisons physiques (cf le théorème de Shannon en électronique) nous ne pouvons pas dépasser 1 kHz. Il faut vérifier si cette fréquence ne pose pas problèmes concernant le temps d'accès à la centrale inertielle. En réalité, le protocole I2C ne permet pas d'accéder à la centrale inertielle à plus de 100 Hz, cela sera la fréquence limitante. Le temps d'acquisition peut alors être augmenté.

4.3.3.2 Multifichier

Avant de commencer la programmation nous avions le choix de séparer le fichier principal en plusieurs fichiers pour les utiliser comme des librairies. Ceci rend le code plus clair et il est plus facile de s'y retrouver. Personnellement je n'ai pas senti ce besoin car le programme tient sur moins de 500 lignes de code. Mais pour des plus gros besoins je recommande l'utilisation de plusieurs fichiers. Pour rappel, une librairie c'est un header qui présente les fonctions disponibles (entre autres), et un fichier source. Ces deux fichiers sont inclus dans le code source principal, souvent appelé main. Le compilateur "link" le tout et forme un seul fichier exécutable par l'arduino ou la teensy.



Organisation de la compilation en C

4.3.3.3 Le setup

Le setup doit en gros initialiser ce qui il y a à initialiser, préparer les entrées sortie. Globalement on y met ce qui doit être exécuté qu'une seule fois au début de notre code.

```
void setup() {
Serial.begin(9600);
delay(1000);
Serial.println("BEGIN SETUP");
```

```
Wire2.begin();
if (!gyro.begin())
{
Serial.println("Failed to autodetect gyro type!");
}
else
{
Serial.println("Gyro Connected !");
}
gyro.enableAutoRange(true);
if (! accel.begin()) {
Serial.println("LSM303 Couldnt start");
}
4.3 Pôle programmation 28
Serial.println("LSM303 found !");
if(!mag.begin())
{
Serial.println("Ooops, no LSM303 Mag detected ... Check your wiring!"); }
else{
Serial.println("Mag connected !");
}
if(!bmp.begin())
{
/* There was a problem detecting the BMP085 ... check your connections */
Serial.print("Ooops, no BMP085 detected ... Check your wiring or I2C ADDR!");
}
else{
Serial.println("BMP085 Connected !");
}
pinMode(pin_battery, INPUT);
pinMode(pin_buzzer, OUTPUT);
pinMode(pin_button, INPUT);
pinMode(pin_led_erreur, OUTPUT);
pinMode(pin_button, INPUT);
pinMode(pin_WP, OUTPUT);
pinMode(pin_led_save, OUTPUT);
pinMode(pin_led_on, OUTPUT);
digitalWrite(pin_led_on, HIGH);
pinMode(pin_wheatstone, OUTPUT);
pinMode(pin_dem_exp, INPUT);
int i = 0;
for (i = 0; i < 5; i++)
{
pinMode(pin_capteurs[i], INPUT);
```

```
}
ready_ = true;
delay(500);
scan_i2c();
Serial.println("\n");
Serial.println("FIN SETUP");
}
```

Comme c'est là que les composants de la centrale sont trouvés et initialisés, il est intéressant de l'annoncer via le Serial, afin de faciliter le débogage en cas de problème.

En général, quand un composant n'est pas trouvé c'est qu'aucun ne l'est. On remarque aussi que la non utilisation d'un capteur n'est PAS un un procédé bloquant. C'est-à-dire que notre centrale peut être déconnectée, on pourra quand même finir le setup. Cela à deux conséquences majeures :

— Si la centrale est déconnectée, la fusée peut quand même enregistrer les jauges (à condition de ne pas écouter la centrale ensuite).

— La centrale n'est pas obligatoire pour le vol en mode dégradé.

4.3.3.4 La partie loop

4.3.3.4.1 Suppression et délais et alarme

C'est la fonction bien célèbre qui s'exécute à l'infinie tant que la teensy est sous tension, à la manière d'un while(true). On va se servir de cette boucle pour scruter le plus rapidement le décollage de la fusée et exécuter ce qu'il en suit. Cela a pour conséquence de proscrire l'utilisation des delays pendant la phase d'attente! En effet, si j'utilise un delay pour faire clignoter une led par exemple et que la fusée décolle au début du delay, il sera trop tard quand je m'en rendrais compte car le delay met en "pause" la teensy.

Pour faire clignoter une led ou un buzzer une technique simple existe, l'utilisation des dates :

— On commence par prendre une date de référence (initialisé à 0 c'est important) — On compare millis() avec. millis() est la date de la teensy

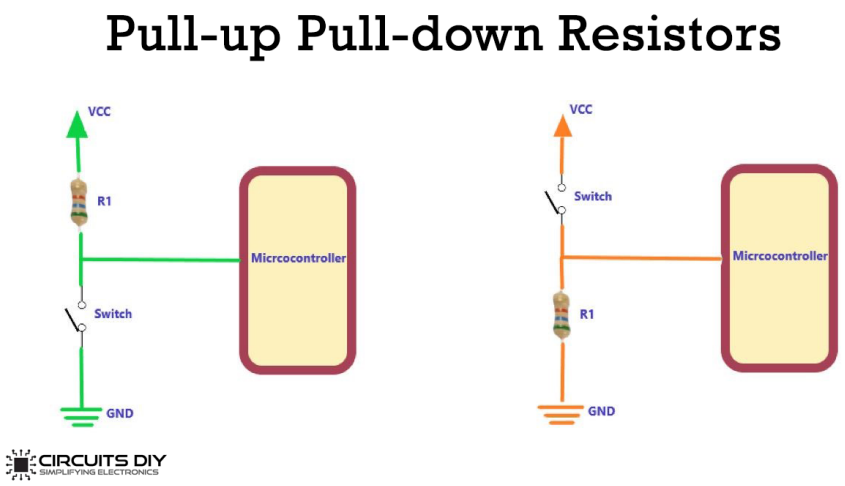
— Si la différence est plus grande que le delay de clignotement, on inverse l'état de la led ou du buzzer. Sinon rien

Vous remarquez qu'aucun délai n'est utilisé! Dans le code une telle implémentation ressemble à ça (ici l'alarme du buzzer).

```
if (alarm_buzzer){
if(millis() - date_buzzer > signed(delay_bip_alarm) ) { // bip en alternance
si letat_buzzer = not etat_buzzer;
date_buzzer = millis();
digitalWrite(pin_buzzer, etat_buzzer);
}}
else{
digitalWrite(pin_buzzer, LOW);
}
```

4.3.3.4.2 Démarrage de l'expérience

D'après le schéma électrique, le séquence informe que la fusée a décollé en faisant le contacte entre deux pin, dont l'un est relié à une entrée analogique en PULL DOWN et l'autre au 5v.



L'impédance (supposée infinie) des pins analogiques permet de dire qu'en pull down, l'arduino reçoit 5v seulement et seulement si le contacte est effectué. Cela permet d'utiliser une mesure digitale fiable (pas de fils relié à rien avec une signal aléatoire).

```
if (digitalRead(pin_dem_exp) == HIGH) { //demarage exp
on_record = true;
ready_ = false;
Serial.println("Demarrage Exp...");
record();
on_record = false;
SDsave();
}
```

Nous sommes en pull down, si on lit un HIGH, c'est que l'expérience a débuté, on commence l'enregistrement avec record(). A partir du moment où l'expérience a commencé la fusée ne sera plus prête à décoller (ready_ = false). Le clignotement de la led qui indique que la fusée est prête se fige. Pour une plus grande rapidité, les communication séries doivent être enlevées pour le vrai décollage. on_record doit permettre d'afficher l'état de la carte avec les leds, ce n'est pas implémenté sur cette version du code.

Une fois que l'enregistrement est terminé, on peut prendre du temps avec SDsave() pour basculer les données sur la carte SD.

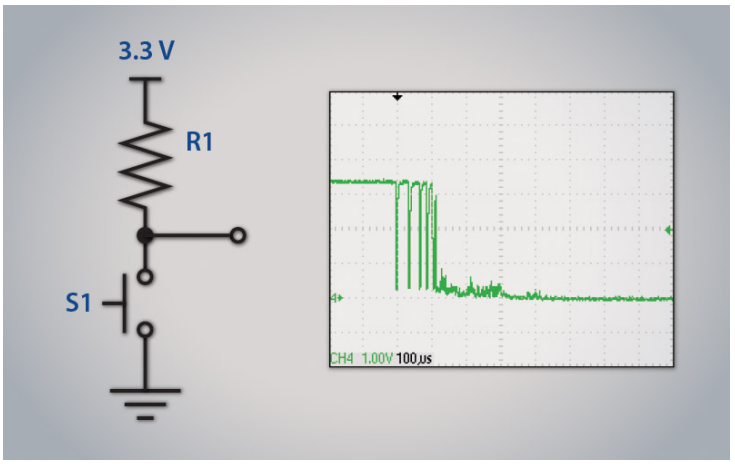
4.3.3.4.3 Gestion du bouton

Un bouton est présent sur la carte expérience et il a deux rôles :

— Faire une mise à zéro des jauges juste avant le décollage sans contrainte — Basculer les données sur les EEPROM manuellement (fonction de débogage principalement)

La bibliothèque bounce présente sur teensy permet de contrer les défauts mécaniques des boutons poussoir.

En effet lors du relâchement du bouton, le contact électrique n'est pas rompu de manière propre, en témoigne la figure suivante :



Même si le bouton n'est activé qu'une seule fois ici, 3 appuis sont détectés, ce qui peut mettre à mal le code qui s'ensuit. La librairie bounce permet de gérer les boutons comme un capteur qu'il faut surveiller. On va ainsi détecter un front montant (un appuis) et un front descendant (un relâchement). Le temps entre ces deux événements module la fonction que l'on exécute. Un appuis court correspond à une remise à zéro des jauges, un appuis long à un basculement des données.

```
push_button.update();
if (push_button.risingEdge() and not on_record) { // remise à zéro offset
jauges adate_rising_edge = millis();
4.3 Pôle programmation 32
}
if (push_button.fallingEdge() and not on_record){
date_falling_edge = millis();
if (date_falling_edge - date_rising_edge > 1000) { // appuis long -> save
(1000msdigitalWrite(pin_led_save, HIGH);
Serial.println("Sauvegarde sur EEPROM");
save_eeprom();
digitalWrite(pin_led_save, LOW);
}
else { //appui court -> offset zero
digitalWrite(pin_led_erreur, HIGH);
Serial.println("Offset jauges ...");
set_offset();
delay(100);
digitalWrite(pin_led_erreur, LOW);
delay(100);
digitalWrite(pin_led_erreur, HIGH);
delay(100);
digitalWrite(pin_led_erreur, LOW);
}
}
```

Dans le cas d'un offset, on indique qu'il est bien effectué par un clignotement de la led erreur. J'utilise des délais ici puisque si ce code est exécuté, c'est que le bouton est appuyé, et que par conséquent, la fusée n'est pas prête de décoller.

4.3.3.4.4 Alarme de batterie faible

La fusée doit nous avertir si sa batterie est faible par un signal sonor, une alarme. Inutile de vérifier la tension de la batterie 5000 fois par seconde, on perdrais de la précision sur la détection du décollage.

```
if (millis() % 10000 <= 25) { // toute les 10s
alarm_buzzer = check_battery(); //on vérifie le niveau de batterie }
```

Ce morceau de code n'est pas parfait mais fait l'affaire. Il permet de vérifier l'état de la batterie dans un intervalle de temps de 25 ms toute les 10 s. Si vous remarquez, rien n'interdit la fusée d'exécuter check_battery() plusieurs fois pendant ces 25 ms! C'est un défaut de cette implémentation. check_battery() renvoie un booléen selon que la batterie est déchargée ou non.

4.3.3.5 Les autres fonctions

4.3.3.5.1 SDsave

L'utilisation d'une carte SD est très simple sur la teensy 3.5 puisque qu'un support SD est directement disponible. Concernant son utilisation, un mot clef spécial permet de démarrer le support présent. La suite est très simple, on utilise print et println avec en préfixe le fichier.

On vérifie que le fichier est bien ouvert et on poursuit ! Une grosse modification sur la fonction est apportée pendant le C'Space, les explications sont développées dans la partie sur le déroulement du C'Space.

```
Serial.print("Initializing SD card...");
if (!SD.begin(BUILTIN_SDCARD)) {
Serial.println("initialization failed!");
return;
}
Serial.println("initialization done.");
// open the file.
File myFile;
const char nom[] = "save.txt";
myFile = SD.open(nom, FILE_WRITE);
// if the file opened okay, write to it:
if (myFile) {
Serial.println("Enregistrement sur la carte ...");
for(int i=0; i<5; i++){
myFile.print("Jauge Vlue en V:");
myFile.println(i);
for(int j = 0; j < n_donnees; j++){
myFile.println(map(float(data_strain[i][j]), 0.0, 1023.0, 0.0, 5.0)); }
```

```
}
```

Il est important de ne pas oublier #include <SD.h> au début de votre projet. Pour tout enregistrer, on parcourt tout simplement tous les tableaux de données en prenant soin d'annoter les unités et les coordonnées.

4.3.3.5.2 Record

C'est peut être la fonction la plus importante de toute puisque c'est elle qui va tout échantillonner et stocker. Elle n'est pas pour autant très compliquée. Tout s'articule autour d'une boucle while sur le temps passer à enregistrer.

```
void record() {//enregistrement du vol jusqu'a la fin
float t = 0.0;
float T = 1 / float(freq_echant); //en seconde
int j,i = 0;
j = 0;
sensors_event_t event;
float temperature;
float seaLevelPressure;
int t_ref = millis();
while (j < n_donnees) {
t_ref = millis();
digitalWrite(pin_wheatstone, HIGH);
for (i = 0; i < 5; i++)
{
data_strain[i][j] = lecture_jauge(i);
Serial.print("Jauge ");
Serial.print(i);
Serial.print(" : ");
Serial.println(data_strain[i][j]);
}
digitalWrite(pin_wheatstone, LOW);
```

La lecture des jauges est vraiment simple. Il ne faut pas pour autant oublier d'alimenter les ponts de wheatstone avant et de les éteindre après pour économiser de l'énergie.

Je ne peux pas assurer un intervalle de temps exactement constant entre deux instants de mesure. Cet intervalle dépend du temps d'accès au bus I2C, aux valeurs qui transitent. . . Cela veut dire qu'il ne faut pas attendre le même temps entre deux mesures, ce temps dépend de la période d'échantillonnage et du temps que l'on à mis pour faire les dernières mesures. Bien sûr, si le temps d'une période est dépassé, c'est qu'on est en retard, on attend pas. C'est exactement la raison pour laquelle la date de chaque mesure est aussi enregistrée.

```
if (millis() - t_ref < T * 1000) { //millis()-t_ref est le temps de
l'operation delay( T * 1000 - millis() + t_ref); // on attend le temps restant
}
```

L'extraction des données de la central se fait en faisant une référence à une variable du type sensors_t_event qui permet transformer les données reçu par I^2C.

```
accel.getEvent(&event);
data_acc[0][j] = event.acceleration.x;
data_acc[1][j] = event.acceleration.y;
data_acc[2][j] = event.acceleration.z;
gyro.getEvent(&event);
data_gyro[0][j] = event.gyro.x;
data_gyro[1][j] = event.gyro.y;
data_gyro[2][j] = event.gyro.z;
mag.getEvent(&event);
data_mag[0][j] = event.magnetic.x;
data_mag[1][j] = event.magnetic.y;
data_mag[2][j] = event.magnetic.z;
```

En ce qui concerne le calcul de l'altitude, le baromètre utilise simplement la différence de pression entre le niveau de la mer et la pression actuelle. Comme la pression au niveau de la mer est une constante dans le capteur mais pas en réalité, l'altitude réelle peut être fausse! Par contre, cela ne pose pas de problème car nous voulons connaître l'altitude de la fusée relativement au sol : les erreurs s'annulent. La librairie du baromètre fournit directement le calcul de l'altitude.

```
bmp.getEvent(&event);
bmp.getTemperature(&temperature);
seaLevelPressure = SENSORS_PRESSURE_SEALEVELHPA;
data_alt[j] = bmp.pressureToAltitude(seaLevelPressure, event.pressure,
temperature
```

4.3.3.5.7 lecture_jauge

Pas grand chose à dire, c'est une lecture analogique, à laquelle il faut retirer la déformation résiduelle au repos.

```
int lecture_jauge(int i) {
return analogRead(pin_capteurs[i]) - off_set_jauges[i];
}
```

Je souligne le retour d'un entier. On ne convertit pas directement la tension en V puisque les EEPROM sauvegardent plus simplement des entiers transformés en bytes.

4.3.3.5.3 set_offset

Au repos, on fait une première mesure des jauges, qui permet de sauvegarder le décalage à prendre en compte.

```
void set_offset() {
int i = 0;
digitalWrite(pin_wheatstone, HIGH);
```

```
for (i = 0; i < 4; i++) {
off_set_jauges[i] = analogRead(pin_capteurs[i]);
}
digitalWrite(pin_wheatstone, LOW);
}
```

4.3.3.5.4 save_eeprom

Afin de sauvegarder les données des jauges dans les EEPROM il faut mettre en place une manière d'organiser la base de données dans 8 EEPROMS de 256 ko. On choisit d'attribuer une EEPROM pour une jauge, une pour l'accéléromètre, une pour le gyroscope et une dernière pour le temps. On arrive bien à 8 EEPROMS utilisées. Pour les données en trois dimensions, on écrit les trois coordonnées à la suite. L'adresse de chaque eeprom est stockée dans un tableau hexadécimal nommé ADRESS_EEPROM.

```
void save_eeprom() {
int i = 0;
int rank = 0;
int max_addr = pow(2, 7);
int index_EE_acc = 5;
int index_EE_gyro = 6;
int index_EE_time = 7;
byte addr[2] = {0x0, 0x0};
for (i = 0; i < 5; i++)
{
for (rank = 0; rank < n_donnees; rank++)
{
addr[0] = rank % max_addr;
addr[1] = (rank - addr[0]) / max_addr;
eepromWrite(i, addr[1], addr[0], byte(data_strain[i][rank])); }
}
```

La difficulté est portée sur la fonction eepromWrite, qui prend en argument l'index de l'eeprom dans laquelle écrire, l'adresse de haut niveau et celle de bas niveau ainsi que l'octet à écrire

4.3.3.5.5 check_battery

C'est une fonction booléenne qui retourne vrai si la batterie atteint un seuil prédéfini. Comme la tension de la batterie ne peut pas être fournie, on utilise un pont diviseur de tension. Qu'il faut bien ajuster au passage. Pour convertir la tension analogique en numérique on utilise la fonction map qui permet de remonter à la tension réelle sur le port. En utilisant des arguments flottant on s'assure de ne pas tronquer le résultat.

```
bool check_battery() {
int niveau = analogRead(pin_battery);
float tension_lue = map(float(niveau), 0.0, 1023.0, 0.0, 5.0);
```

```
Serial.println("tension batterie (V):");
Serial.println(tension_lue / rapport_pont_diviseur_batterie); return
tension_lue / rapport_pont_diviseur_batterie <= seuil_limit_bat; }
```

4.3.3.5.6 eepromRead

Pour construire la lecture je me suis basé sur l'exemple fournie, en y ajoutant l'index de l'eeprom cible. C'est une utilisation typique de la librairie Wire. Cette fonction renvoie l'octet présent dans cette case mémoire.

```
byte eepromRead(int index, byte highAddress, byte lowAddress) {
Wire.beginTransmission(ADDRESS_EEPROM[index]);
Wire.write(highAddress);
Wire.write(lowAddress);
Wire.endTransmission();
Wire.requestFrom(ADDRESS_EEPROM[index], 1); // one byte requested
while (!Wire.available())
{
delay(10);
}
return Wire.read();
}
```

4.3.3.5.7 eepromWrite

L'écriture ne change absolument pas des fichiers d'exemple de la librairie Wire.

```
void eepromWrite(int index, byte highAddress, byte lowAddress, byte data) {
Wire.beginTransmission(ADDRESS_EEPROM[index]);
Wire.write(highAddress);
Wire.write(lowAddress);
Wire.write(data);
Wire.endTransmission();
}
```

4.3.4 Et le code du séquenceur ?

Le code du séquenceur n'est pas complexe en soi. Il est basé autour du même principe d'utilisation de la boucle loop pour la détection de l'alarme batterie et du décollage de la fusée par le biais du câble jack. L'écran LCD affiche les principaux états de la fusée. Il est de toute façon donné en annexe.

4.3.5 Problèmes rencontré et solutions associées

4.3.5.1 Quid des gros numéros de pin ?

Pour les différentes entrées analogiques nous utilisons les entrées A1, A2. . . Et notamment A22 et A21. Il se trouve que ces pin ne sont pas utilisés par les Arduino courantes. On ne pouvait donc pas compiler sous peine de recevoir le message “A21” is not defined. Le problème est que pour les première compilation nous n’avions pas teensyduino, qui fait comprendre à l’IDE que les pin A21 et A22 existent bien, mais pour la teensy.

4.3.5.2 Ma centrale inertielle ne se connecte pas

Les fichiers d’exemples permettent de tester les divers composants à tester. D’une manière générale, les tester sur une arduino permet de savoir si le problème vient du fait que ce soit une teensy ou pas. D’autre part, souvent vérifier qu’une centrale fonctionne se fait par le biais du port série. Il faut que celui-ci soit à la bonne fréquence (baud rate). Pour nous, la centrale fonctionnait sous arduino en série, le code compilait sur teensy mais rien ne s’affichait. La teensy dispose de 3 interfaces I2C notées respectivement (SDA0 ;SCL0, SDA1 ;SCL1, SDA2 ;SCL2). De base, la teensy se connecte avec la librairie Wire.h sur le port 0. Ici nous avions connecté la centrale sur le port n^{o} 2, et la carte ne pouvait donc pas trouver la centrale. En déplaçant sur une plaque d’essai, la connectique sur le port 0, on retrouve le fonctionnement recherché. Tout finit bien, oui et non : la carte est déjà imprimée et la sortie prévue pour la centrale est branché définitivement sur SDA2, SCL2, il faut impérativement trouver un moyen de changer le bus I2C pour la com avec la centrale.

4.3.5.3 Comment changer les numéros des bus I^2C

Une librairie nommée i2c_t3 doit en principe permettre de modifier les pin SDA et SCL pour choisir où le protocole i2c est utilisé en lui indiquant les pin SDA, SCL. Le problème est que cette librairie remplace Wire.h. C’est une librairie plus avancée mais qui suit quand même les fonctions de base. Or la centrale inertielle ainsi que tous les composants à sa surface sont faits pour Arduino et utilisent Wire.h en conséquence. Si on essaye de compiler ces deux libraires ensemble, on obtient un joli message de définition conflictuel. Il faut alors utiliser i2c_t3 à CHAQUE FOIS qu’il est utilisé ailleurs. C’est assez pénible et pas sécurisant de modifier directement le code dans une librairie. Une autre méthode trouvée sur internet consiste à utiliser Platform IO et lui indiquer de toujours ignorer Wire.h.

Pour d’autres raisons, Virtual Studio Code avec l’extension PlatformIO n’est pas utilisé. Mais cette solution est élégante.

L’autre solution présentée ici
(https://forum.pjrc.com/threads/21680-New-I2C-library-for-Teensy3
?p=79995viewfull=1post79995) consiste à déguiser Wire.h en i2c_t3.h :

1) Create a Wire directory in <sketchbook>/libraries:

2) Create a Wire.h file in it with the following contents:

Code:

```
#ifndef _WIRE_SPOOF_I2C_T3_
#define _WIRE_SPOOF_I2C_T3_

#include <i2c_t3.h>

typedef i2c_t3 TwoWire;

#endif
```

The idea being that I don't need to modify your i2c_t3 library as well.

Extrait de la solution internet

Il faut donc commencer par créer une fausse librairie dans le dossier avec les autres librairies (le mettre dans le sketchbook n'a pas suffit pour moi, il faut donc enlever la vrai librairie Wire.h), y coller le morceau de code ci dessus* et utiliser Wire.h normalement (i.e. en rajoutant <i2c_t3.h> sur les projets).

*Il est mieux d'utiliser le code suivant pour remplir Wire.h :

```
#ifndef _WIRE_SPOOF_I2C_T3_
#define _WIRE_SPOOF_I2C_T3_
#include <i2c_t3.h>
typedef i2c_t3 TwoWire;
#endif
```

On évite les définitions en boucle par le compilateur.

Cette méthode m'a permis de compiler correctement mais l'ancien bus est utilisé quand même. . . La seule méthode qui a bien fonctionné au final, a été d'aller modifier dans la librairie des composants, le port I2C utilisé, c'est à dire remplacer Wire.xxxx par Wire2.xxxx sauf pour include <Wire.h> bien entendu. Ce qui a été fait précédemment ne semble pas changer le problème. Cette dernière m'a été inspiré par un thread qui a été posté sur PJRC (Teensy forum) pour l'occasion, le lien du thread :

https ://forum.pjrc.com/threads/61221-Adafruit-IMU-not-recognized-on-Wire2 ?p=242151posted=1

4.3.5.4 Je n'arrive pas à communiquer avec les EEPROM

Faire rentrer un float , utiliser le bon Wire1.

4.3.5.5 Mon bouton n'est pas bien reconnue

Utiliser la librairie bounce, utiliser des niveaux logiques.

4.3.5.6 Mon buzzer d'alerte fait un bruit bizarre

Pont diviseur mal défini -> sous alimentation de la carte.

4.3.6 Conclusion

Le choix est-il justifié de prendre une Teensy ?

Carte SD, bon fonctionnement, rapidité, faible consommation, présence de beaucoup de port i2c,mémoire de stockage très importante.

En contrepartie non négligeable, la teensy nécessite une prise en main un peu plus délicate que les cartes Arduino habituelles. L'utilisation du Wire 2 ne semble pas encore très bien conçue et nécessite des déviations peu conventionnelles. Si le besoin d'une teensy n'est pas mieux justifié (par rapport à une arduino classique), je ne recommande pas personnellement son utilisation.

4.3.7 Le code du banc d'essai

Rien à dire de particulier, les données analogiques sont envoyées sur le serial pour faire des points de mesure. Le code se trouve en annexe.

4.4 Pôle récupération

La partie récupération s'occupe de déterminer le moyen par lequel la fusée sera ramenée au sol. Dans cette partie nous traiterons essentiellement de la partie mécanique mais il est à noter que les systèmes doivent être mis en place avec les électroniciens (dans notre cas l'interlocuteur est Alexis Collet) car leur système de commande doit être adapté à la mécanique.

4.4.1 Système d'éjection

Nous avons tout d'abord déterminé quel système d'éjection nous souhaitions. Nous avions le choix entre trois principales méthodes : par la coiffe, en séparant la fusée et par une trappe latérale. Plusieurs d'entre nous ayant essayé une mini fusée bi-étage sans succès l'année précédente, nous avons décidé de rester sur la solution de trappe latérale qui nous semblait la plus simple.

Il faut alors déterminer comment ouvrir cette trappe. Pour cela la première solution envisagée était un servo-moteur comme dans beaucoup d'autres fusées. Cependant devant l'imprécision de certains systèmes ou la nécessité de devoir transformer le mouvement de rotation du servo en mouvement de translation, nous avons choisi d'opter pour un solénoïde. Attention, il est important de choisir un solénoïde normalement étendu (PULL) afin que la trappe ne s'ouvre pas en cas de coupure de courant.

Notre choix s'est porté sur le solénoïde LZ 2560 (GoTronic) car il offre une grande force de traction et nous n'arrivions pas à estimer la force nécessaire. Cependant ce choix n'est surement pas le meilleur car le solénoïde est lourd, encombrant et demande une forte intensité.

ITS-LZ 2560

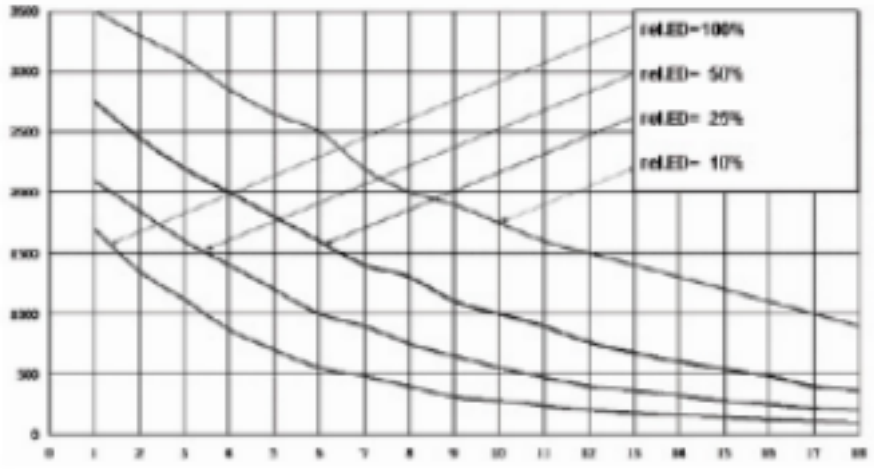
Beschreibung:

Zylindermagnete haben durch ihre geschlossene Bauform einen höheren Wirkungsgrad als Bügelmagnete.
Sie können in rauer Umgebung eingesetzt werden, da das Gehäuse sehr stabil und unempfindlich ist.
Der Einbau erfolgt über das Schraubgewinde.
Angaben für 100% Einschaltdauer, Grenztemperatur: 130°C

Bauform

Spulendaten

Relative ED (Prozent)	100	50	25	10
Max. ED (Sekunden)	∞	360	32	8



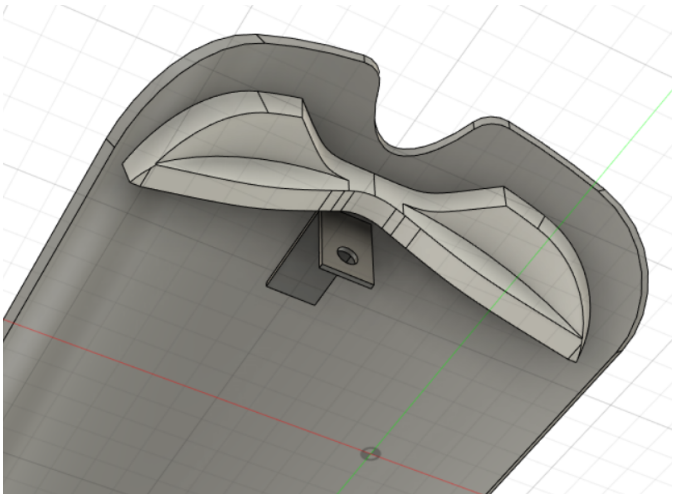
Initialement, on a fait le choix d'alimenter la carte de puissance du solénoïde en 12V (directement alimenté par l'accu 3S du séquenceur). Il s'est avéré par la suite que la force de rétraction était insuffisante pour tirer la goupille et libérer la trappe. Ceci était dû notamment aux frottements générés par l'état de surface de l'équerre en aluminium de la trappe. Pour tenter de réduire ces frottements, on a ajouté de la graisse liquide sur les contacts.

Pour augmenter la puissance du solénoïde, on a d'abord ajouté un convertisseur boost avant la carte de puissance pour augmenter la tension et par conséquent la puissance transmise au solénoïde, de 10 à 30W. Cette solution a permis de passer les tests de la RCE3 mais le convertisseur a grillé durant le C'Space. On a finalement décidé d'ajouter un accu LiPo en série pour augmenter la tension à 24V. Ce qui a été une réussite bien que cette accu n'avais pas de circuit de protection...

Une fois le solénoïde retiré, la trappe est poussée par le parachute plié, lui-même poussé par un morceau de lycra tendu entre les deux U. La trappe est attachée au parachute afin d'aider à l'extraction de celui-ci. Nous avons retenu du précédent CSpace l'importance de n'avoir aucun élément qui pourrait potentiellement accrocher la toile dans le compartiment parachute ainsi que le fait que le parachute doit sortir de sa trappe de façon naturelle même au sol.

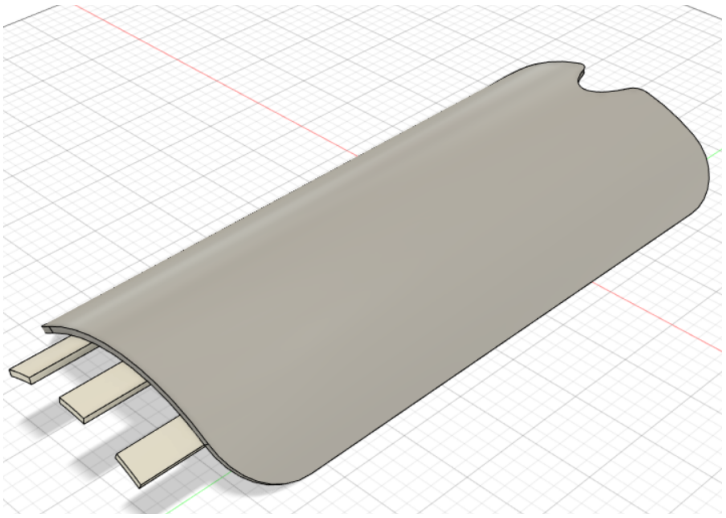
Tissu en lycra rouge permettant l'éjection du parachute et pièce de maintien du solénoïde en blanc dans le compartiment au-dessus.

Du point de vue de la fabrication nous avons donc réalisé une pièce en impression 3D pour supporter le solénoïde, et deux bagues alu de chaque côté de la trappe. La trappe est un bout de tube découpé aux bonnes dimensions qui a été renforcé par une impression 3D en PLA. Le renfort permet à la trappe de garder sa forme même si le vent s'y engouffre.



Une équerre vient aussi verrouiller la trappe avec la bague du solénoïde.

Afin que la trappe bascule, trois languettes en métal sont collées en bas de la trappe. Elles servent aussi à maintenir la trappe dans son emplacement pendant le début du vol.



4.4.2 Toile de Parachute

Pour le parachute nous avons décidé de partir sur un parachute circulaire (et non pas en croix) fabriqué avec un tissu Spinnaker CX2 (45gr) sur le site Cerf-volant service. A l'aide du Stabtraj, nous avons déterminé la surface qu'il faudrait alors pour respecter le cahier des charges.

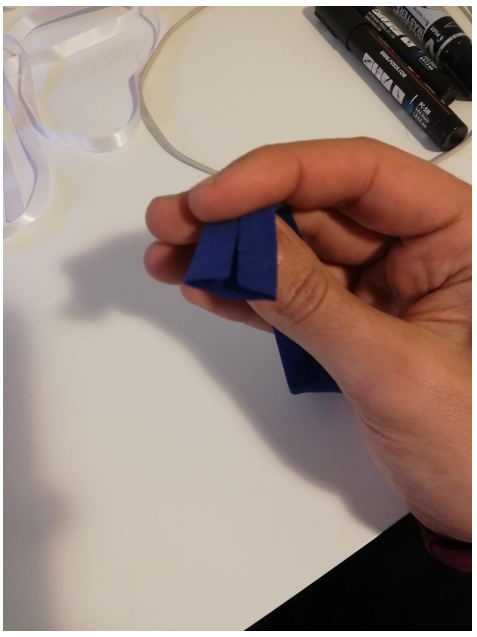
En effet, la réglementation REC2 du cahier des charges nous impose une vitesse verticale d'arrivée au sol comprise entre 5m/s et 15m/s, calculée selon :

$$V_d = \sqrt{\frac{2Mg}{\rho C_x S}}$$

M : Masse en kg de la fusée avec un propulseur vide
S : Surface en m^2 du parachute déployé
g : 9.81 m/s^2
ρ_0 : 1.3 kg/m^3
C_X : 1

Pour une masse à vide (Catia) de 8,638kg, nous estimons un poids total aux alentours des 10kg ce qui nous porte à choisir une surface de πm^2, soit un rayon de 1 m. Une cheminée de 10 cm de diamètre est découpée pour laisser une partie de l'air s'échapper et éviter le phénomène d'oscillation du parachute qui mènerait la fusée à sa perte.

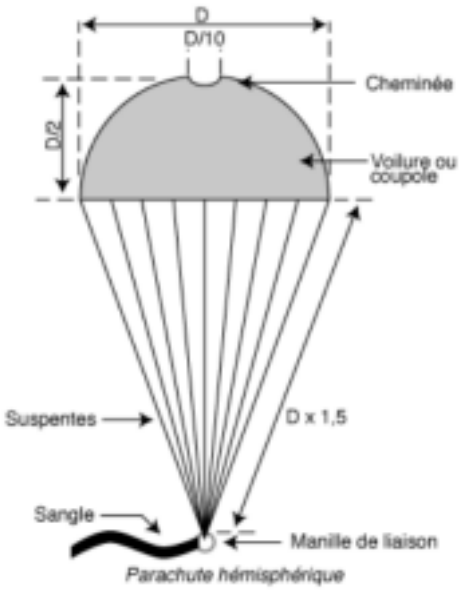
Les bords de la toile sont renforcés par un tissu acheté qui se replie sur lui-même, 4 épaisseurs de tissu sont présentes ce qui permet de ne pas émailler la toile.



4.4.3 Système d'accroche

4.4.3.1 Les suspentes

Elles doivent être de même longueur (D x1,5), symétrique, 6 ou plus, positionnées autour du parachute cf schéma ci-contre (source : planète science) :

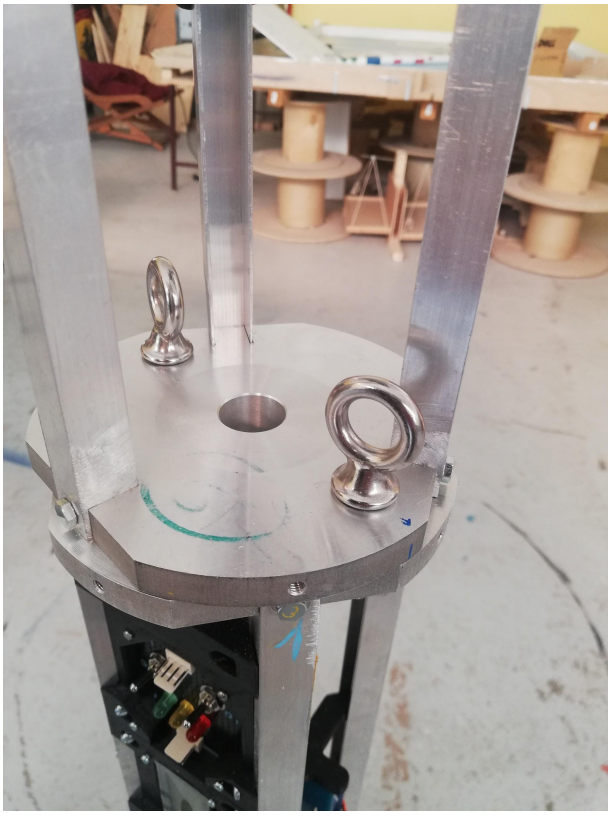


La corde à suspente est celle de Leroy Merlin, elle est suffisante pour notre utilisation. Il faudra veiller à bien maîtriser l'ouverture afin de ne pas gagner trop de vitesse et dépasser la force d'arrachement limite du parachute.

4.4.3.2 Fixations des suspentes

Les suspentes reprennent tout l'effort de traction du parachute. Nous choisissons de répartir la force entre 16 points d'accroche pour être suffisamment sécurisé vis-à-vis de la corde utilisée.


Les attaches sont des morceaux de tissu cousus sur le contour bleu. Une couture carré permet de garantir la solidité. Une boucle est alors créée, on y passe la suspente en y faisant un nœud de chaise. Côté émerillon, toutes les suspentes passent par un “désemméleur”. La grosse corde côté fusée est attachée aux deux anneaux de l'inter-étage, les deux anneaux doivent participer à la reprise de la force d’arrachement. Un nœud de chaise est réalisé sur l’anneau du fond. La corde est simplement passée dans le premier anneau.



4.4.3.3 Pliage du parachute

- Démêler les suspentes, c’est-à-dire vérifier les éventuelles torsades au sein d’un même brin avant le jour J.

- Plier le parachute « en sapin », en deux, plusieurs fois de suite puis le plier en accordéon.

- Laisser la pointe du parachute ou l'extrémité de la banderole à l'extérieur du pliage. - N'enroulant jamais les suspentes autour du parachute.

- Introduire le parachute dans le corps de la fusée.

- Mettre en dernier lieu les suspentes et les sangles.

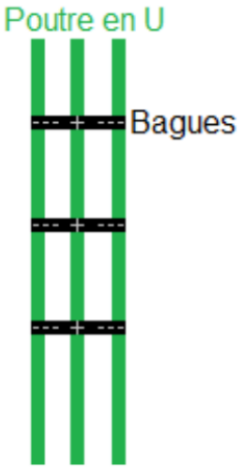
- Le parachute ne doit être introduit que quelque temps avant le lancement pour éviter qu'il prenne des plis empêchant son ouverture.

4.5 Pôle mécanique et structure

4.5.1 Structure générale de la Fusée

Avant de concevoir la fusée, nous avons réfléchi aux méthodes de fabrication et d'assemblage. En effet, il est compliqué de fixer les éléments de notre fusée directement dans le tube si celui-ci est trop long. Cela pose des problèmes d'accessibilité pour le bras.

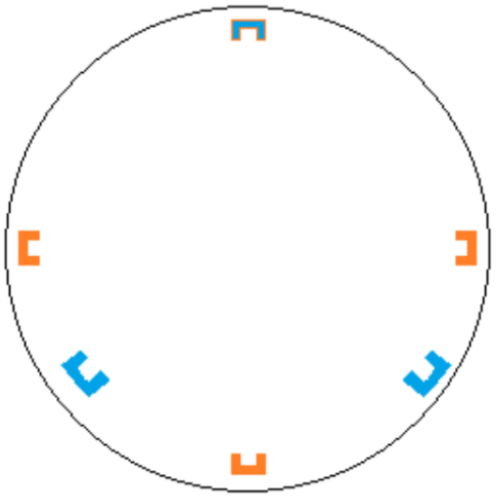
Nous avons donc conçu la structure interne de la fusée en la séparant du tube. Pour cela nous utilisons des poutres de section en forme de U, auxquelles nous fixons les bagues. Ainsi, tous les éléments de la fusée sont accessibles facilement. La structure de la fusée et les poutres en U sont représentées sur le schéma suivant. Les bagues sont vissées aux poutres en U qui sont elles-mêmes vissées au tube en certains points régulièrement espacés.



Afin de renforcer la résistance en flexion de la structure, nous avons disposé quatre poutres en U. Cependant, cela posait un problème dans la partie haute de la fusée puisque c'est dans cette partie que se trouve le compartiment du parachute qui prend une place importante et qui nous empêche de placer quatre poutres en U. C'est pourquoi, dans la partie supérieure de la fusée, nous n'utilisons que trois poutres en U.

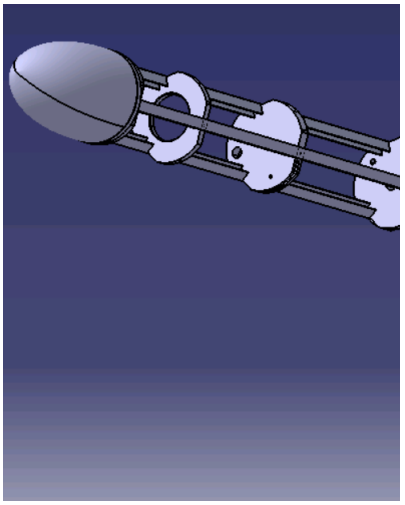
Nous avons donc dessiné deux bagues fixées entre elles, qui permettent de faire la transition entre la partie inférieure qui contient quatre poutres et la partie supérieure qui en contient trois. Ainsi, la première est fixée aux quatre poutres de la partie basse et le seconde aux trois poutres de la partie haute.

Afin de simplifier la conception de la structure, nous choisissons de placer une des trois poutres de la partie haute dans le prolongement d'une des quatre poutres de la partie basse. Cette situation est représentée dans le schéma ci-dessous. Ainsi, la poutre située dans la partie supérieure du schéma est commune aux deux parties de la fusée.



Cette structure facilite également le transport de la fusée puisqu'il est possible de la séparer en deux et d'assembler les deux parties avant le décollage.

Nous avons modélisé la fusée sur Catia. Cela nous a permis d'avoir une vue d'ensemble des différents éléments de la fusée et de créer facilement des plans pour la fabrication. Ne pas oublier de bien paramétrer le produit et d'utiliser un squelette pour faciliter la conception.



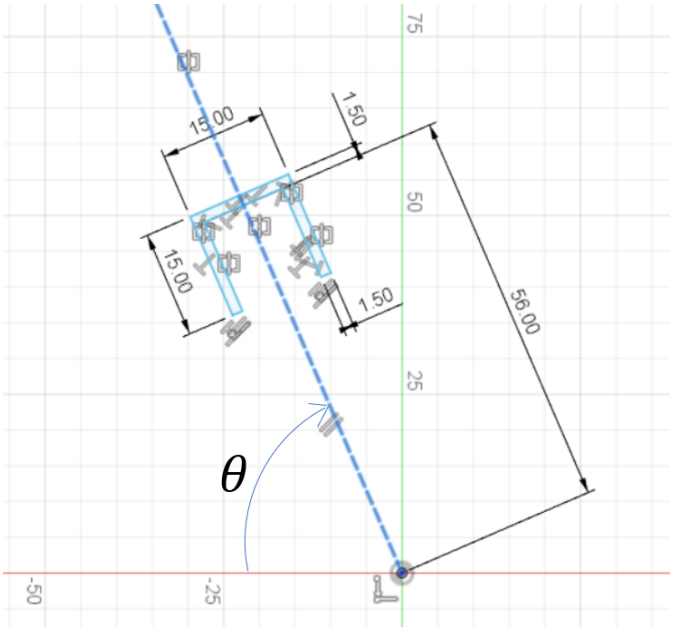
De plus, nous avons choisi un diamètre extérieur de 129 mm de diamètre ce qui correspondait à nos exigences et aux tubes trouvables dans le commerce. Le tube en pvc provient d'amazon, il s'agit d'un tube de ventilation, il a l'avantage d'être plus légé que les tubes pvc pour évacutation, mais aussi d'être blanc. Son usinage est assez similaire aux tubes pvc de plomberie.

4.5.2 Calcul de la flèche théorique de la fusée

Dans l'idée de pouvoir prédire la flèche de la fusée avant de la construire, quelques hypothèses sont réalisées pour éviter de devoir s'en remettre aux logiciels de MEF. La fusée en elle-même peut répondre à la définition d'une poutre si l'on considère que les bagues agissent comme des forces ponctuelles de par leur poids. Les bagues ont aussi l'avantage de garantir la perpendicularité des sections de la fusée par rapport à la ligne moyenne ce qui permet d'utiliser les hypothèses classiques de la RDM.

Dans notre cas, la fusée est encastrée au niveau du moteur comme les contrôles l'indique. La flèche maximale dépend alors de toutes les forces ponctuelles et du poids propre des profilés en U. La peau n'est pas modélisée ici, elle participe à rajouter de la rigidité et peu de poid, nous serons en présence d'un cas pire.

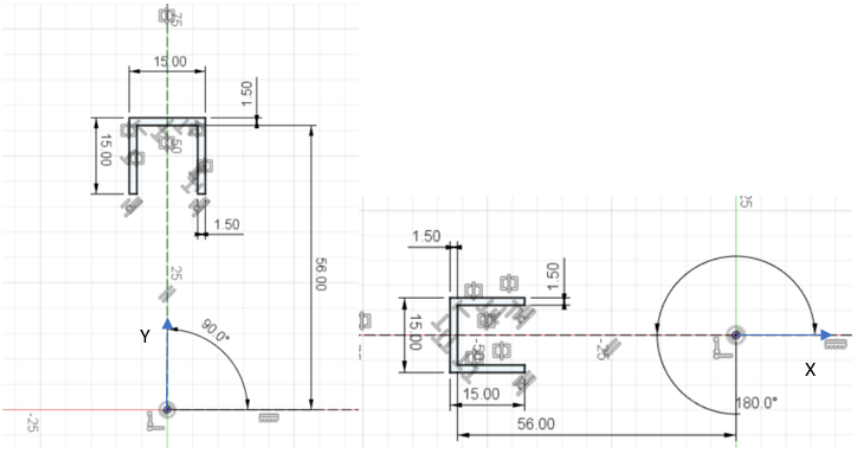
De manière assez intuitive, on pourrait penser que le moment quadratique de la structure de la fusée dépend en partie de l'angle de rotation des poutres en U autour de l'axe de la fusée. Cet angle sera repéré à partir de l'axe horizontal x. Comme sur la figure suivante.



On cherche à calculer le moment quadratique d'une telle poutre.

$$I = cos(\theta)^2 I_x + sin(\theta)^2 I_y + sin(2\theta) I_{xy}$$

Les moments quadratiques Ix et Iy sont plus facilement calculables, cf la figure suivante



L'épaisseur du profilé est noté e, sa largeur et sa profondeur sont égales et notées L. L'écart entre l'axe de la fusée et l'intérieur du U est noté d (égal à 56 mm sur Volokna).

En décomposant le profil selon plusieurs rectangle nous pouvons alors écrire :

$$I_x = L^3 e/6 + 2eL(d - L/2 + e)^2 + e^3(L - 2e)/12 + (L - 2e)e(d + e/2)^2$$

$$I_y = 2(e^3 L/12 + Le(L/2 - e)^2) + (L - 2e)^3 e/12$$

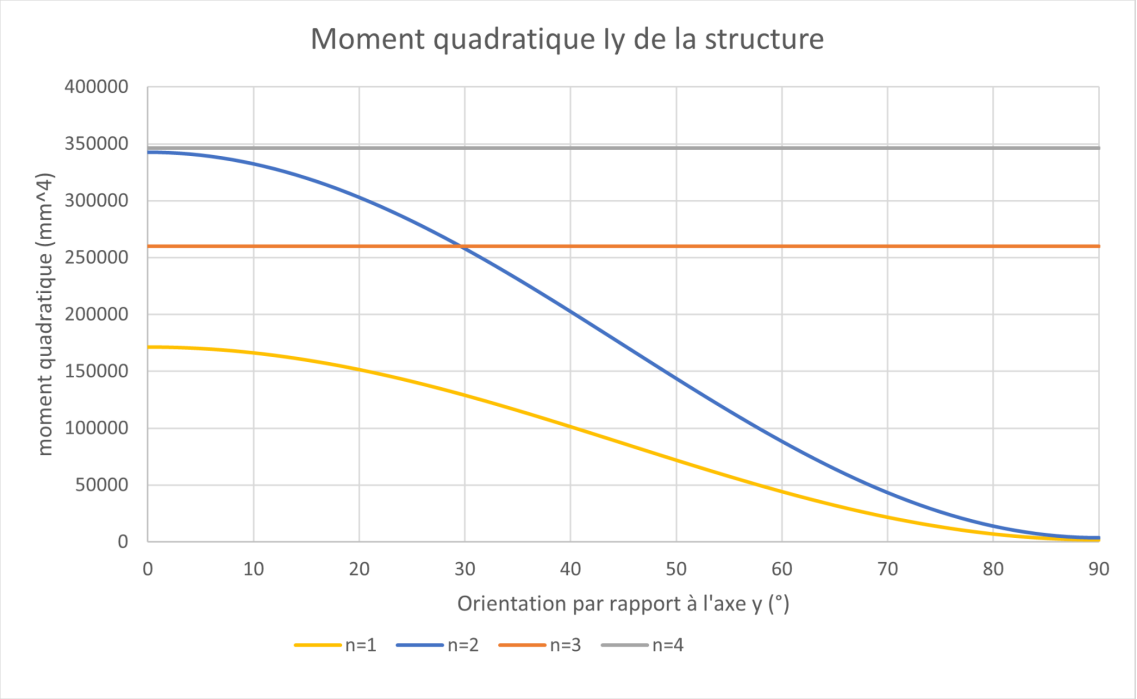
$$\mathrm{I} = \mathrm{I}_x \cdot \sum_{i=0}^{n-1} cos^2(\theta + \tfrac{2i\pi}{n}) + I_y \cdot \sum_{i=0}^{n-1} sin^2(\theta + \tfrac{2i\pi}{n})$$

En se servant de l'espace complexe on peut montrer que pour n> 2 (notre cas) on a $I = (I_x + I_y)n/2$

Cette équation nous montre qu'à partir de 3U, le moment quadratique ne dépend plus de thêta ce qui n'était pas prévu à la base. Cette observation permet de faciliter les essais lors du test de flexion.

n	Moment quadratique (mm^4)
3	259742
4	346323

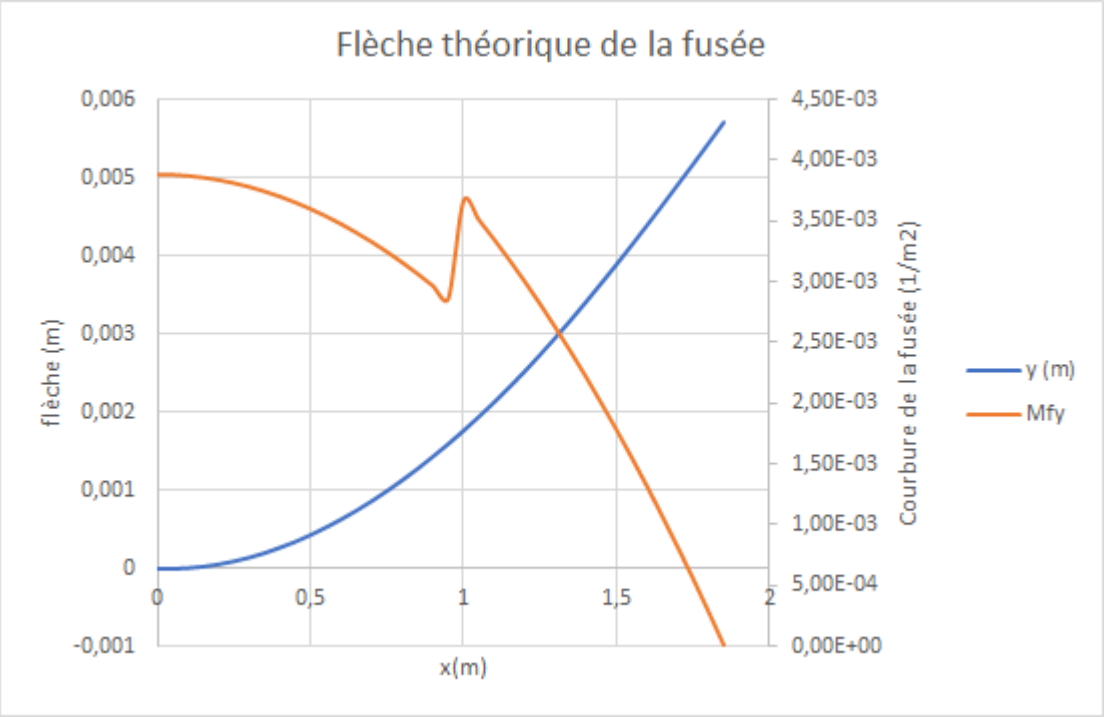
En bonus on peut tracer les moments quadratiques pour n<=4 :



On pourra prendre dans un premier temps l'hypothèse d'une bonne répartition de la masse dans la fusée (le centre de masse est situé proche du milieu de la fusée). Dans ce cas les efforts sont répartis tout du long de la fusée et on obtient :

$$EI_y \frac{d^2y}{dx^2} = M_{fy}$$

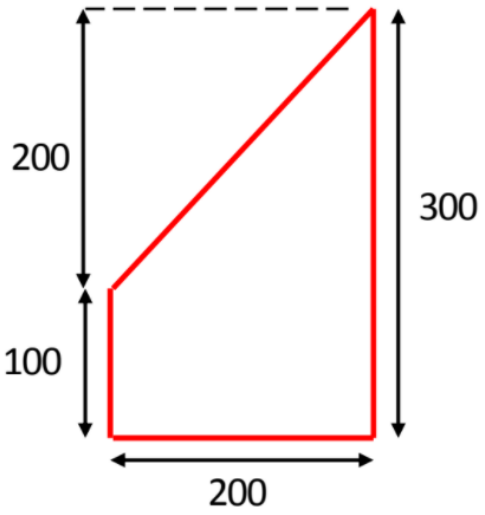
En découpant la structure en deux parties avec la partie 3U et la partie 4U on obtient la courbe de flèche par intégration. En effet en x= 0, y = 0 et la pente de la flèche est nulle.



Ce graphique permet aussi de se rendre compte que la transition 4U 3U est un point de rupture de la dérivé de la flèche. Enfin, la flèche théorique maximale est de 0.31% de la longueur de la fusée, ce qui est un critère validé du cahier des charges. En réalité la flèche mesurée est encore plus faible de par le tube qui rajoute de la rigidité et le moteur qui maintient quelques bagues de guidage.

4.5.3 Fixation des ailerons

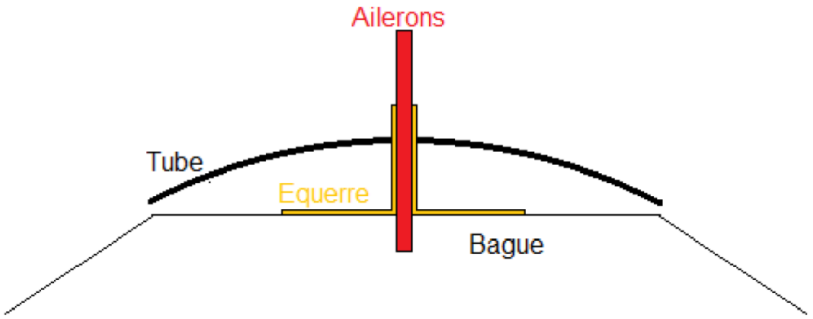
Les ailerons assurent en partie la stabilité de la fusée. Ils doivent être bien dimensionnés et bien fixés pour que la fusée soit stable. Nous avons réalisé leur dimensionnement à l'aide du logiciel Stabtraj fourni par Planète Sciences.



Nous avons quatre ailerons qui font 2 mm d'épaisseur.

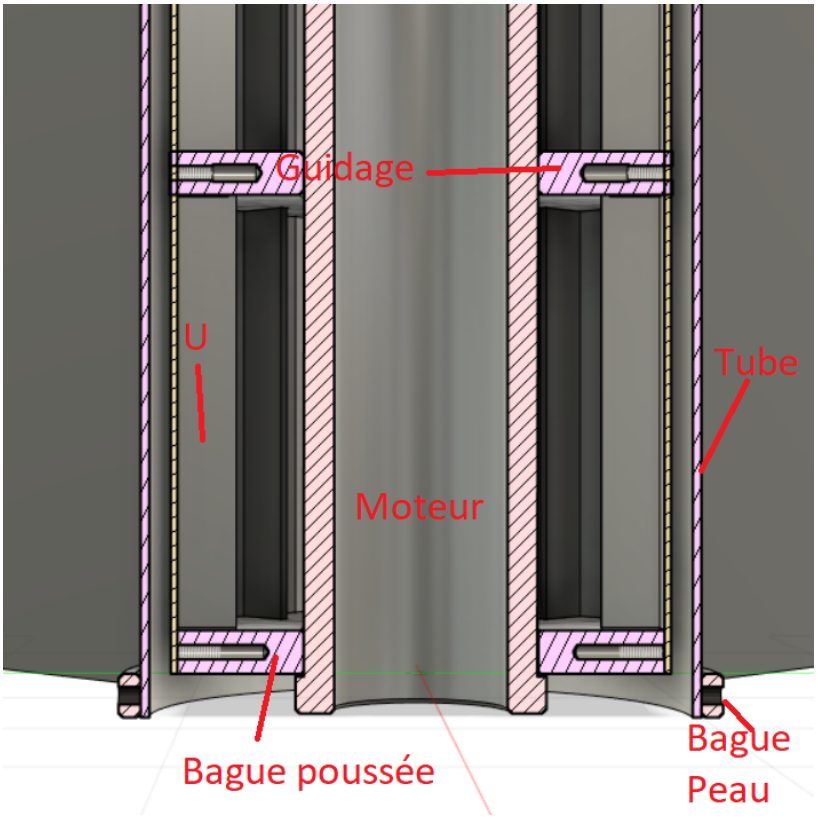
Nous utilisons des équerres pour fixer les ailerons. Ces équerres ne sont pas fixées au tube mais aux bagues à l'intérieur de la structure interne de la fusée. On crée alors une fente dans le tube pour laisser passer les équerres et les ailerons.

Les équerres sont également fixées aux ailerons à l'aide d'une vis. Les bagues de fixation contiennent des fentes pour que les ailerons soient fixés bien perpendiculairement à côté de la bague.

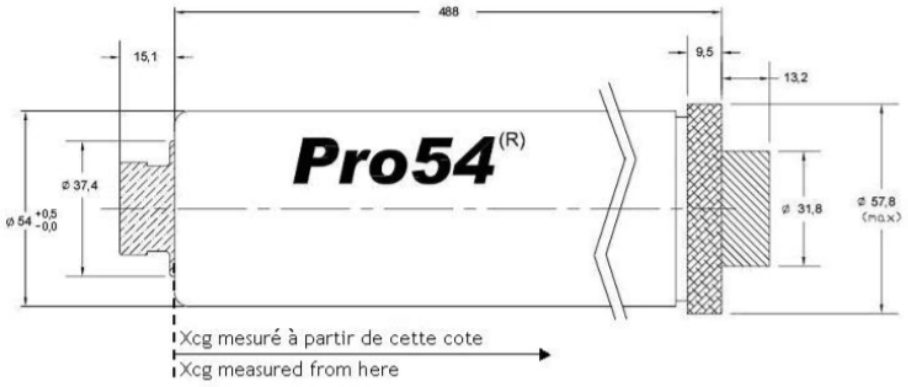


4.5.4 Fixation du propulseur

Nous utilisons la méthode de reprise de poussée par le bas de la fusée. Le propulseur est donc retenu dans la fusée, lors de la phase de poussée, à l'aide d'une bague fixée à l'extrémité basse du tube qui doit permettre de transmettre la poussée à la fusée. Le trou central de cette bague est dimensionné afin de laisser pénétrer le tube du propulseur mais pas la partie basse du propulseur qui a un diamètre légèrement plus important. Nous utilisons trois bagues réparties le long du propulseur pour que celui-ci soit bien centré.



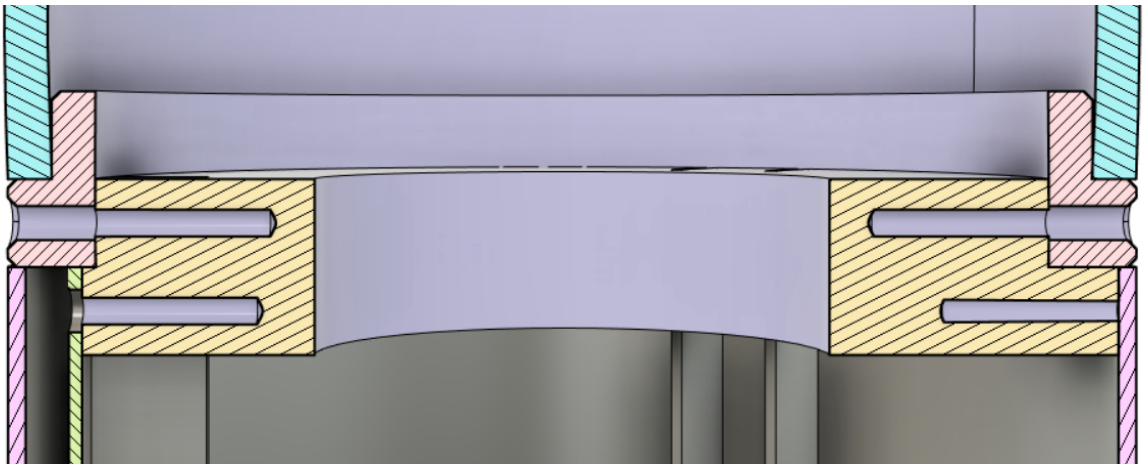
Nous utilisons le propulseur Pro54 qui est adapté des fusées expérimentales subsoniques. Les dimensions sont données dans le plan ci-dessous. Nous dimensionnons les bagues de fixations à partir de ces dimensions. Nous laissons entre 0.5 et 1 mm de jeu entre les bagues et le propulseur

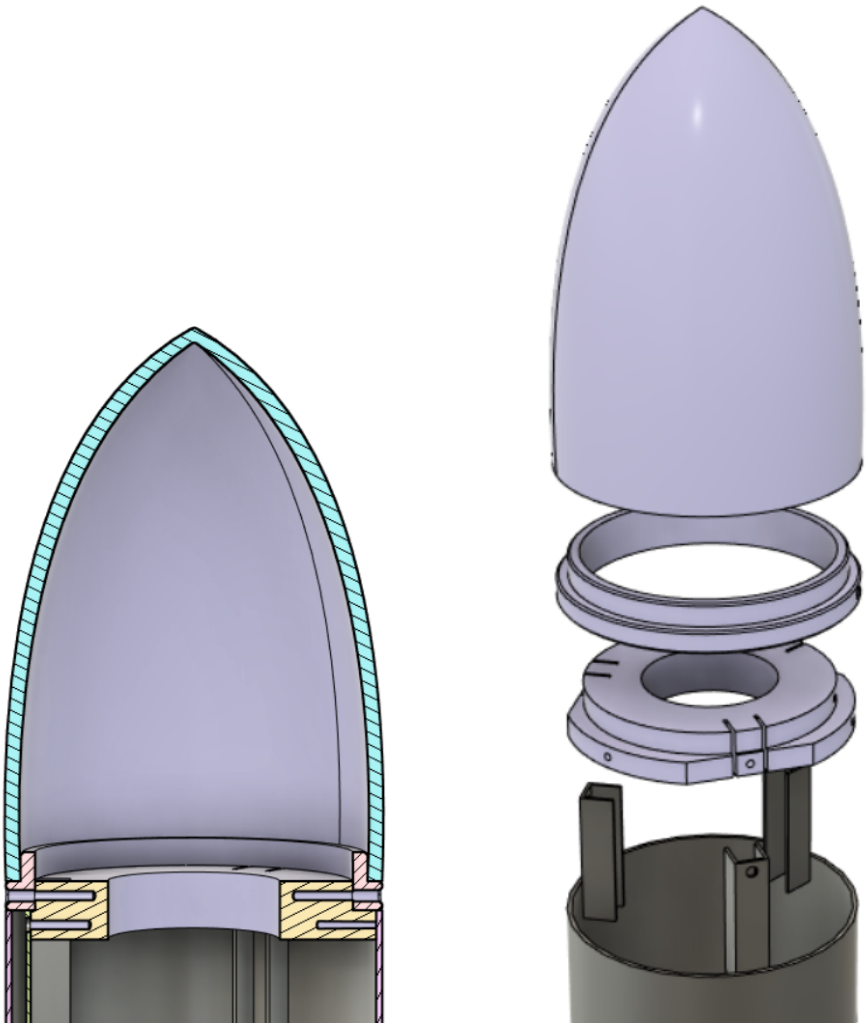


Dimensions en millimètres

4.5.5 Fixation de la coiffe

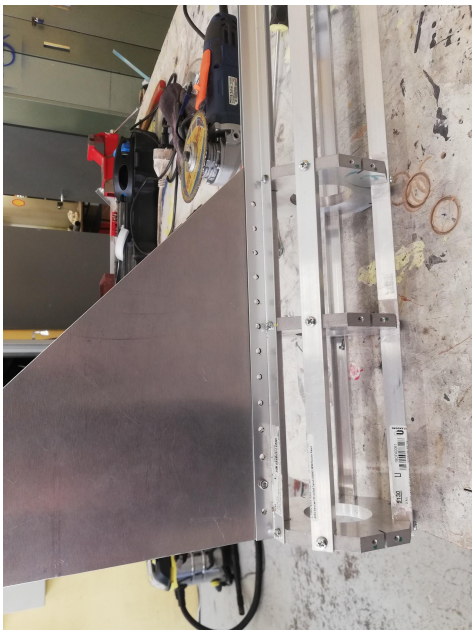
La coiffe est positionnée au sommet de la fusée. Elle est fixée grâce à deux bagues. L'une est fixée aux trois poutres en U et l'autre est fixée à la première. La coiffe est fixée à cette deuxième bague.

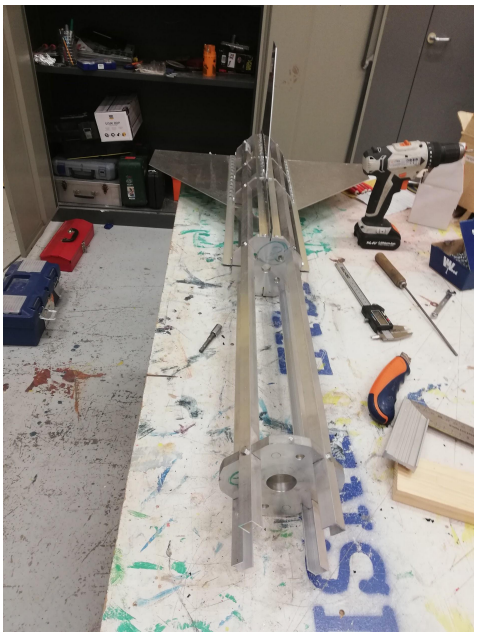




4.5.6 Fabrication de la fusée et difficultés

La fabrication a débuté en Janvier 2021, l'atelier étant fermé avant. Quelques points clefs de la fabrication se doivent d'être présentés pour mieux appréhender les prochaines conceptions. Nous avons commencé par la partie basse. La complexité du montage est de pouvoir percer les poutres de manière assez précise pour garder les bagues parfaitement parallèles. Un décalage de la bague de poussée serait mauvais pour la stabilité de la fusée !



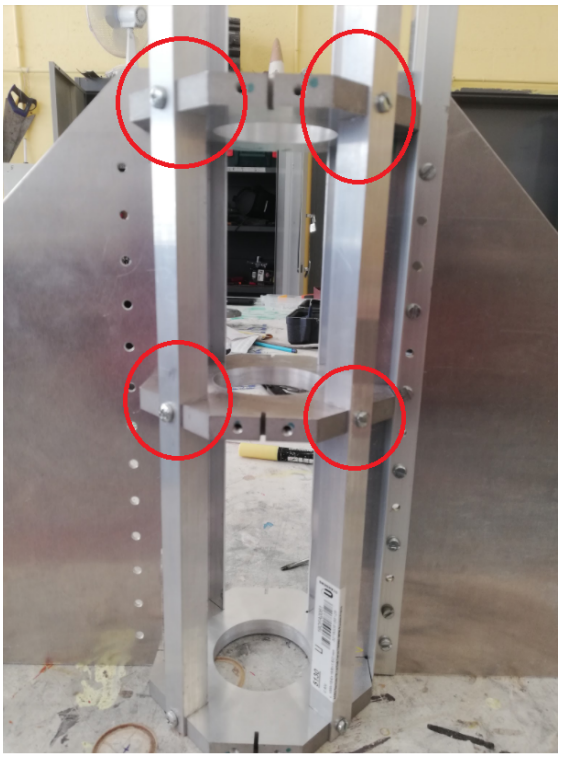


Quelques erreurs de conception se révèlent alors :
— La tolérance des fentes pour les U ne sont pas respectées.
— Les fentes pour les ailerons ne sont pas assez profondes

— La tolérance sur le diamètre du moteur n'est pas renseignée sur nos plans.
— Le centrage des perçages sur les faces des pièces usinées est mauvais.

Il faut savoir que les poutres en U achetées chez Leroy Merlin sont des profils extrudés.
Les tolérances de fabrications ne sont pas compatibles avec l'aérospatiale. Il faut tenir compte que les cotes sont grossières. Si un U doit être inséré dans des fentes, il faut donc prévoir plus de marge. En particulier, nous avons subi l’erreur du technicien réalisant l’électro-érosion, qui a pris une côte trop courte de 0,2 mm pour toutes nos fentes.

La solution pour palier au problème a été de poncer l'épaisseur du U localement pour le faire rentrer dans un espace plus restreint.



4.5.6.1 La coiffe en composite

Habituellement les coiffes de fusée du club sont faites en PLA en fabrication additive. Mais ayant un an de plus pour la production nous nous sommes lancés le défi de faire la coiffe en composite avec la solution de repli d'une coiffe en PLA si nous n'arrivions pas à atteindre notre objectif.

L'arrivée d'un tour à bois à l'atelier a facilité la réalisation du moule en permettant de tourner de la mousse pour lui donner la forme voulue. Deux paramètres étaient alors imposés : le diamètre en bas de la coiffe afin qu'elle puisse être collée sur la bague en aluminium et la longueur de la coiffe d'une vingtaine de centimètres. La forme était ensuite assez libre en essayant d'avoir une forme bien symétrique.

Pour vérifier que le diamètre à la base n'était pas trop petit, nous avons découpé un demi-disque du diamètre souhaité dans une planche lorsque le profil correspondait avec notre forme le diamètre était bon !

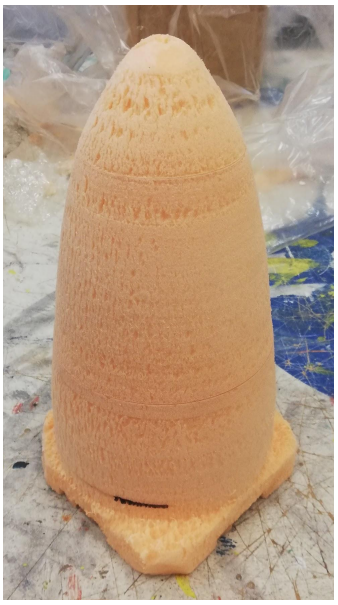


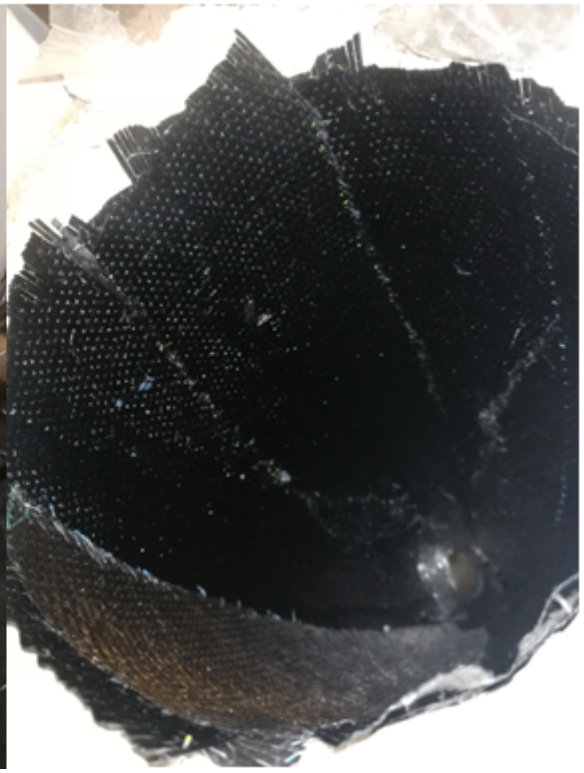
Figure : Coiffe en mousse

Une fois cette forme en mousse réalisée, soit nous drappions immédiatement dessus avec la difficulté de draper sur une forme externe et la mousse serait restée prise dans la coiffe (suffisamment léger pour ne pas poser de problème) soit il fallait créer un contre moule qui serait un négatif de cette coiffe en mousse. Nous avons choisi la deuxième option et notre contre-moule a été réalisé en plâtre dans un bac en bois construit à cet effet. Avant de mouler dans le plâtre nous avons recouvert la coiffe d'enduit afin de retravailler la forme et essayer que le plâtre ne s'accroche pas à la mousse.

Pour démouler la pièce en mousse prise dans le plâtre nous avons utilisé de l'air comprimé pour la pousser par dessous hors du plâtre sec.

Enfin nous avons nappé l'intérieur de ce contre-moule avec des fibres de carbones préalablement imbibées d'époxy avec un pinceau. Les fibres étaient coupées en triangle ainsi

qu’un rond pour le bout. Attention à cette étape a avoir la forme la plus propre possible c’est un matériel difficile à poncer par la suite.



Le démoulage a été très compliqué et nous avons fini par détruire la base en plâtre au marteau, ce qui a été un travail long et fastidieux. Après une nuit dans le vinaigre pour tenter de dissoudre les éléments de plâtres restés collés à la surface (spoiler : ça marche pas), nous avons poncé en détail la pièce. Une fois poncée, une nouvelle couche d’époxy a été déposée afin de reboucher les trous et essayer d’avoir une forme plus homogène.

La coiffe est alors terminée, elle a été peinte, découpée à la bonne longueur à la Dremel puis collée à la bague prévue à cet effet.


En conclusion, le résultat de cette expérimentation est plutôt positif car la coiffe a été autorisée à voler mais quelques réticences par les contrôleurs ont été émises notamment sur la verticalité de la coiffe. Le contre moule n'était finalement peut-être pas la meilleure solution et le drapage sur la mousse aurait pu être intéressant, enfin il pourrait être intéressant de faire de l'injection de résine sous vide pour un résultat plus propre, il faudrait voir avec Jean-Marie Petit ou Bastien Robert.

5 Déroulement du C'space

Le C'Space est le dernier événement important de la vie du projet, il s'articule autour de deux parties distinctes. D'abord il s'agit de passer les qualifications pré-vol, qui valident la conformité de la fusée au cahier des charges émis par planète science. Chaque point est cité, testé et validé par un contrôleur. Lorsqu'un point ne respecte pas le cahier des charges, on nous demande de régler ce point pour y aller progressivement. Il n'est pas conseillé d'attendre d'être prêt pour passer tous les contrôles en même temps. En effet la salle des contrôles risque d'être très prisée au fur et à mesure de la semaine, il vaut mieux répartir les validations. Les avancements sont bien décrits sur le site SCAE et rendent compte des tâches prioritaires. Les RCE permettent de se préparer aux qualifications.

Conception

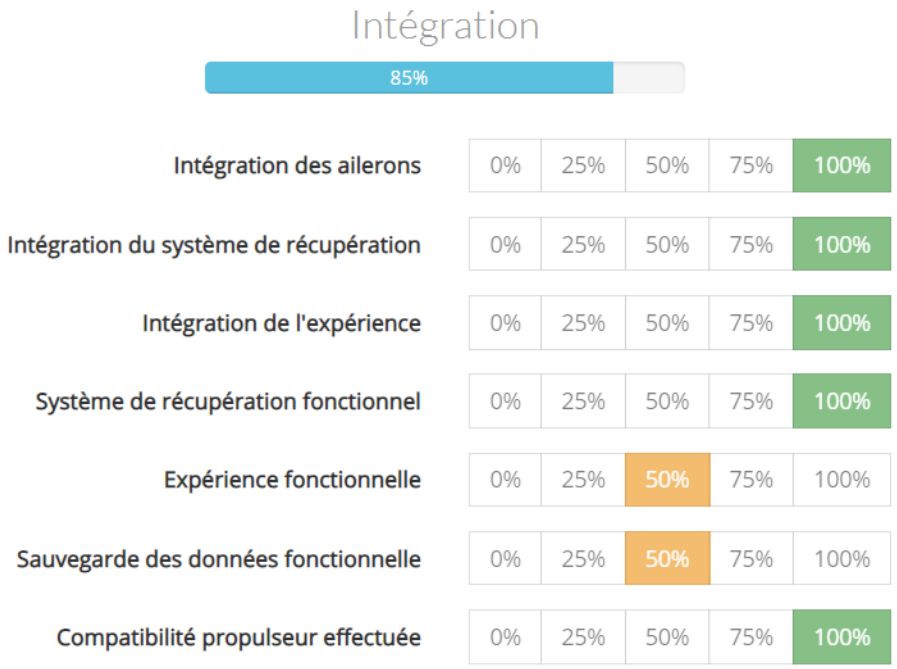
100%

Conception du corps	0%	25%	50%	75%	**100%**
Conception de l'ogive	0%	25%	50%	75%	**100%**
Conception de la structure interne	0%	25%	50%	75%	**100%**
Plan d'intégration global	0%	25%	50%	75%	**100%**
Dimensionnement du parachute	0%	25%	50%	75%	**100%**
Choix du système de libération du parachute	0%	25%	50%	75%	**100%**
Choix des expériences	0%	25%	50%	75%	**100%**
Plans des cartes électroniques du séquenceur	0%	25%	50%	75%	**100%**
Plans des cartes électroniques d'expérience	0%	25%	50%	75%	**100%**
Mise au point de la méthode d'étalonnage	0%	25%	50%	75%	**100%**

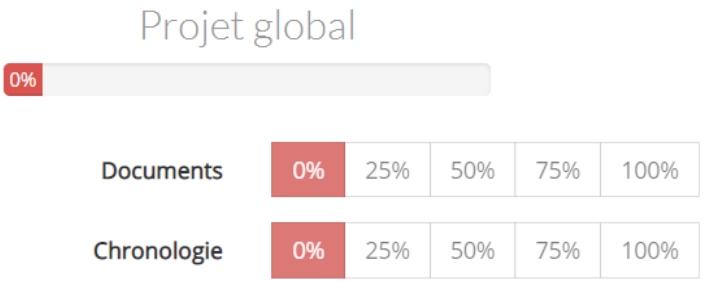
Réalisation

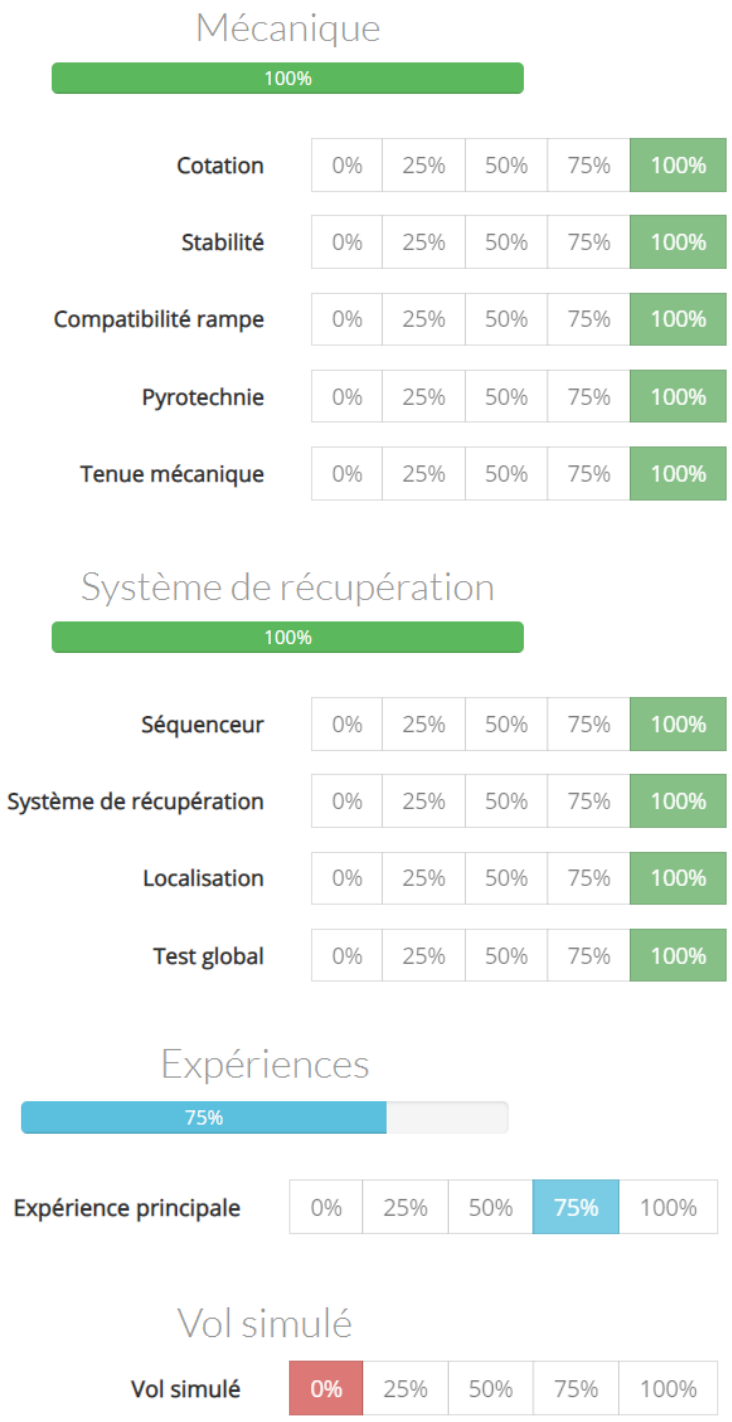
87%

Type de structure	-	peau porteuse	treillis porteur		
Réalisation du corps	0%	25%	50%	75%	**100%**
Réalisation de l'ogive	0%	25%	50%	**75%**	100%
Réalisation de la structure interne	0%	25%	50%	75%	**100%**
Réalisation du logement moteur	0%	25%	50%	75%	**100%**
Réalisation des ailerons	0%	25%	50%	75%	**100%**
Réalisation du parachute	0%	25%	50%	75%	**100%**
Réalisation du séquenceur	0%	25%	50%	75%	**100%**
Étalonnage des expériences	0%	**25%**	50%	75%	100%
Test des cartes d'experience	0%	25%	50%	**75%**	100%
Nombre de cartes électroniques d'expérience	1				



Les qualifications concernent l'application du cahier des charges. On pourra noter que des documents sont nécessaires pour la qualification. Il faudra notamment fournir les circuits de câbles de la fusée, la calibration de l'expérience, les plans de la fusée ...





Durant l'année, une partie des tests peuvent être fait à l'atelier de l'ESP. Les problèmes suivants sont survenus et n'étaient pas anticipés. Il est conseillé d'en prendre connaissance pour gagner du temps.

La mécanique

- Compatibilité propulseur

  Ne pas hésiter à demander la compatibilité avec plusieurs propulseurs, certains passaient mais pas d'autres. En effet, le CNES réutilise des enveloppes moteur, donnant lieu à des diamètres extérieurs différents.

- Alignement de la prise jack

  La prise jack doit toujours être dans l'alignement parfait des ailerons pour éviter de se prendre dans la cage (sauf en rampe). Il est demandé de l'incliner de 30° par rapport au corps de la fusée. Cependant, une inclinaison supérieure peut s'avérer optimale.

- Rigidité de la trappe parachute

  Si la trappe se déforme facilement, elle ne passera pas les contrôles, il faut la rigidifier avec des impression 3D, ou un revêtement en fibre de carbone / fibre de verre. Notre problème était tel que le vent pouvait s'engouffrer dans le logement parachute et présenter un risque d'un point de vue mécanique. Le fait d'attacher la trappe sur son axe fait qu'elle peut d'une part pivoter selon cet axe, et d'autre part présenter les soucis précédents si la trappe se déforme et présente un rayon de courbure plus grand que la peau de la fusée. Une solution pourrait être d'attacher cette trappe à chacun de ses bords.

- Sécurité de la trappe parachute

  La trappe ne peut pas faire une chute libre et doit être attachée. On peut l'attacher à la cheminée du parachute permettant ainsi d'augmenter les chances de voir le parachute sortir grâce à la prise au vent de la trappe.

- Compatibilité rampe

  Éviter d'avoir plusieurs diamètres sur la fusée est un grand gain de simplicité pour la compatibilité cage et le passage des contrôles (passage de la bague de maintien de peau externe en bague interne). En effet, les patins de la cage de lancement sont réglés au diamètre extérieur le plus grand si la fusée en présente des différents.

Le séquenceur

- Ouverture prématurée du parachute

  Il vaut mieux prévoir un déclenchement trop tôt que trop tard, sur notre vol nous avions prévu 0.7 s avant l'apogée calculée.

- Sous alimentation de la carte arduino

  Si la carte arduino est alimentée via un régulateur de tension 7805, il faut l'alimenter par le pin 5v et non le pin Vin.

- Non déclenchement de l'expérience

  Le déclenchement de l'expérience s'effectue via les optocoupleurs, pour que la diode s'active il faut l'alimenter via une résistance bien choisie pour y faire passer le bon courant, 500 Ohms était suffisant pour nous, 10 kOhms ne fonctionnait pas !

- Panne du système de récupération

  La déconnexion de la prise jack n'était pas détectée par le pin 13 (pin jack) de l'arduino Nano. En fait, le seuil de détection logique n'était pas dépassé (état LOW en dessous de 2.5V et HIGH au-dessus). Cela provient du fait que le pin 13 possède une

led onboard, cette led provoque une chute de tension, à la manière d'un pont diviseur de tension, donc l'état était toujours à LOW. Ainsi la tension mesurée était trop basse pour détecter le décollage. Nous avons donc détruit la led pour régler le problème.

- Prévention de la surchauffe du régulateur de tension

  Ne pas oublier de mettre des dissipateurs thermiques pour prévoir les longs temps d'inactivité.

Le parachute

- Faiblesse des attaches de suspentes (cf. partie récupération)

  Faire attention aux types de nœuds utilisés, surtout s'ils fragilisent la corde. Nous avons constaté qu'une suspente s'est arrachée au niveau du nœud à l'émerillon. Peut-être que cette suspente était plus courte que les autres et a été plus sollicitée lors de l'ouverture du parachute. Nous avons eu la chance que cet effet ne se produise pas en domino sur les suspentes. Il faut donc vérifier leur tenue sous l'effort imposé (avec coefficient de sécurité). Pour ce test, un pèse valise peut faire l'affaire.

- Emmêlage des suspentes parachute
  Il faut veiller à ne pas emmêler les suspentes en faisant passer le parachute entre deux suspentes. En tirant sur le désemmeleur, on peut vérifier qu'il n'y a pas de nœuds.

- Pliage du parachute

  Bien respecter la procédure, la taille est très importante, il faut qu'il prenne toute la longueur de la trappe pour ne pas être trop haut et trop pousser sur la trappe. Il faut le plier en accordéon afin qu'il se déroule plus facilement. D'autre part, il faut essayer au maximum de réaliser un pliage où les suspentes sont regroupées (en mode sapin où le tronc est schématisé par les suspentes et le feuillage par le cône du parachute).

- Test de la résistance des suspentes

  Une corde de suspente devrait être testée en traction pour vérifier si les exigences sont respectées. Comme vu au-dessus, l'endroit critique peut être au niveau du nœud.

L'expérience

- Refroidissement du convertisseur de tension

  Ne pas oublier le dissipateur, peut-être prévoir plusieurs alimentations pour les différentes parties de la carte afin de limiter la dissipation sur un seul composant. Ou bien utiliser un convertisseur à découpage plutôt qu'un régulateur linéaire.

- Défaut de résistance d'un pont de wheatstone

  Il vaut mieux utiliser un module du commerce pour éviter les problèmes de mauvaise résistance ou de court-circuit. Cela permet aussi de pouvoir changer le pont si celui-ci est défectueux.

- Retrait de la référence de tension 5.0V

  La référence de tension a été retirée parce que l'amplificateur suiveur associé ne délivrait pas assez de courant pour alimenter les ponts et les amplificateurs différentiels.

- Modification de la référence de tension pré-amplification

  Après quelques recherches, les amplificateurs différentiels ont une impédance finie sur le pin de référence, ce qui provoque un chute de tension à sa borne. Le simple diviseur de tension alors utilisé ne permettait pas de diviser par deux la tension de référence.

- Pose de cosse pour l'équilibrage des ponts de wheatstone

  Afin de connaître la tension différentielle en entrée d'amplificateur nous avons placé des cosse de mesure. Il peut être utile d'en placer plus régulièrement.

Chronologie

La chronologie est un élément essentiel du lancement de la fusée. C'est une série d'instructions datées jusqu'au décollage et au-delà. Chaque action est effectuée à un endroit donné par des acteurs équipés d'outils. Une personne, souvent le chef de projet, lis chaque ligne une à une. Avant de passer à la suivante, il s'assure que l'action est terminée avec un accord oral. C'est lui qui dicte la vitesse de la chronologie. Sur Volokna, très peu d'actions sont effectuées sur le pas de tir, c'est un point très positif. Les actions plus critiques sont effectuées à l'atelier club, avec moins de stress.

Liste de materiel	
Outil	Composant
Boitier Commande JUPITER	carte SD
Clé à laine M3	P54
petit tourne vis cruciforme 1	long cable jack
petit tourne vis cruciforme 2	cable jack
Clé M10	batterie seq
chargeur batterie	batterie exp
tourne vis peau	vis secours
tourne vis moteur + scotch	
Petit tourne vis plat	

Liste d'intervenants		
NOM	Prénom	Fonction
COLLET	Alexis	CDP
BARTELEMY	Anthony	Chef ELEC
BAEUMLIN	Zacharie	Chef RECUP
?	?	PYRO

Identifiant	QUAND	OU	QUI	OBJET	AVEC QUOI	COMMENTAIRE
PRE01	H-24	ATELIER CLUB	Zacharie	Chargement batteries	chargeur	sécuriser la charge des batteries
PRE02	H-24	ATELIER CLUB	Anthony	Vérification de l'inter-étage	Clé M10	solidité de la liaison, et fixation crochet parachute
PRE03	H-24	ATELIER CLUB	Alexis	Insérer carte SD exp	carte SD	Verouiller correctement la carte SD en entendant le click
PRE04	H-24	ATELIER CLUB	Anthony + Zacharie + Alexis	Mise en place peau	tourne vis peau	bonne orientation, trous de la peau alignés avec les trous des bagues
PRE05	H-24	ATELIER CLUB	Anthony	Mise en place bague de peau	Clé à laine M3	x
PRE06	H-24	ATELIER CLUB	Alexis	Verification de la bague de coiffe	tourne vis peau	coiffe bien en place
PRE07	H-3	ATELIER CLUB	Anthony + Zacharie + Alexis	Pliage parachute	x	suivre la méthode de pliage du parachute
PRE08	H-2	ATELIER CLUB	Zacharie	Batterie expérience	batterie 1	respecter les nomenclatures
PRE09	H-2	ATELIER CLUB	Anthony	Batterie séquenceur	batterie 2	respecter les nomenclatures
PRE10	H-2	ATELIER CLUB	Anthony + Zacharie + Alexis	Fermeture et vérouillage trappe	Petit tourne vis plat	vérouillage solénoïde délicat
PRE11	H-2	ATELIER CLUB	Alexis	Prendre vis de rechange peau	x	x
PRE12	M-45	ATELIER CLUB	Zacharie	fermeture de la trappe LCD	petit tourne vis cruciforme 1	2 vis trappe
PRE13	M-45	ATELIER CLUB	Anthony	fermeture trappe batterie exp	petit tourne vis cruciforme 2	2 vis trappe
PRE14	M-35	PAS DE TIR	Anthony + Zacharie + Alexis	Insert Volokna rampe	VOLOKNA	cage rampe
PRE15	M-33	PAS DE TIR	Anthony	Laisser tournevis	tournevis moteur et scotch	x
PRE16	M-32	PAS DE TIR	Anthony	Connection jack rampe	long cable jack	faire un nœud en dessous du point d'attache jack fusée pour detecter le décollage tot
PRE17	M-32	PAS DE TIR	Zacharie	Connection jack fusée	x	vérifier la bonne connection
PRE18	M-28	PAS DE TIR	Zacharie	allumage sequenceur	x	x
PRE19	M-28	PAS DE TIR	Zacharie	allumage expérience	x	x
PRE20	M-27	PAS DE TIR	Zacharie + Alexis	Vérification de la signaletique	x	vérifier les indications LCD et LED verte allumé
PRE21	M-15	PAS DE TIR	BENEVOL	Elevation de la rampe	x	85°
PRE22	M-7	PAS DE TIR	PYRO	montage P54	P54	x
PRE23	M-5	PAS DE TIR	PYRO	verouillage moteur	Tournevis moteur et scotch	Tournevis scothché sur aileron
VOL01	T +0 s	CIEL		mise à feu	Boitier Commande	
VOL02	T +3 s	CIEL	VOLOKNA	fin aquisition experience	x	
VOL03	T +14.7s	CIEL	VOLOKNA	récupération déployée	x	
VOL04	T +15.3s	CIEL	VOLOKNA	récupération déployée	x	
VOL05	T +6min	CIEL	VOLOKNA	Atterissage sous parachute	x	
REC01	UNDEF	CHAMP	Anthony	extinction experience	x	x
REC02	UNDEF	CHAMP	Anthony	extinction sequenceur	x	x
REC03	UNDEF	CHAMP	Zacharie + Alexis	retrait de la batterie sequenceur	tourne vis trappe	trappe à retirer et placer la batterie dans le sac ignifugé
REC04	UNDEF	CHAMP	Zacharie + Anthony	retrait de la batterie expérience	tourne vis trappe	trappe à retirer et placer la batterie dans le sac ignifugé
REC05	UNDEF	CHAMP	Anthony	pliage rapide parachute	x	pliage rapide en 4 et enroulage des suspentes pour ne rien perdre
REC06	UNDEF	CHAMP	Equipe	retour à la base	x	Equipe soulagée
REC07	UNDEF	CHAMP	Alexis	mise en sécurité carte SD	x	placer la carte dans un endroit sécuriser pour récuperer les données

Le vol

La fusée a décollé sur la rampe Toutatis. Le cable jack est fixé à l'aide de plusieurs nœuds dans l'alignement de la prise. Une marge est prise pour ne pas forcer sur le câble et risquer de déconnecter le câble. La chronologie a été respectée sans aucun problème.

Il était très intéressant d'entendre le pyrotechnicien confirmer le statut "Prêt !" sur l'écran LCD de la fusée juste avant le lancement.

La fusée a fait un décollage à 85° de la verticale sans problèmes particuliers. Aucune déviation de trajectoire n'a été observée à cause d'un mauvais positionnement des ailerons ou de l'état de surface de la coiffe.

La récupération de la fusée

Dû à la faible vitesse de descente (voir post traitement et stabtraj) la fusée à eu le temps de se déplacer à la seule force du vent. La déviation l'a emportée à l'est de la zone de lancement dans la zone rouge.

Cette zone est interdite au public, c'est la raison pour laquelle nous n'avons pas pu la récupérer. L'armée est allée la chercher en Defender.

Nous observons que la fusée est en très bon état. Seul un aileron est plié à cause du choc à l'atterrissage, coiffe vers le haut. La coiffe est légèrement recouverte de terre lorsque la fusée s'est couchée.

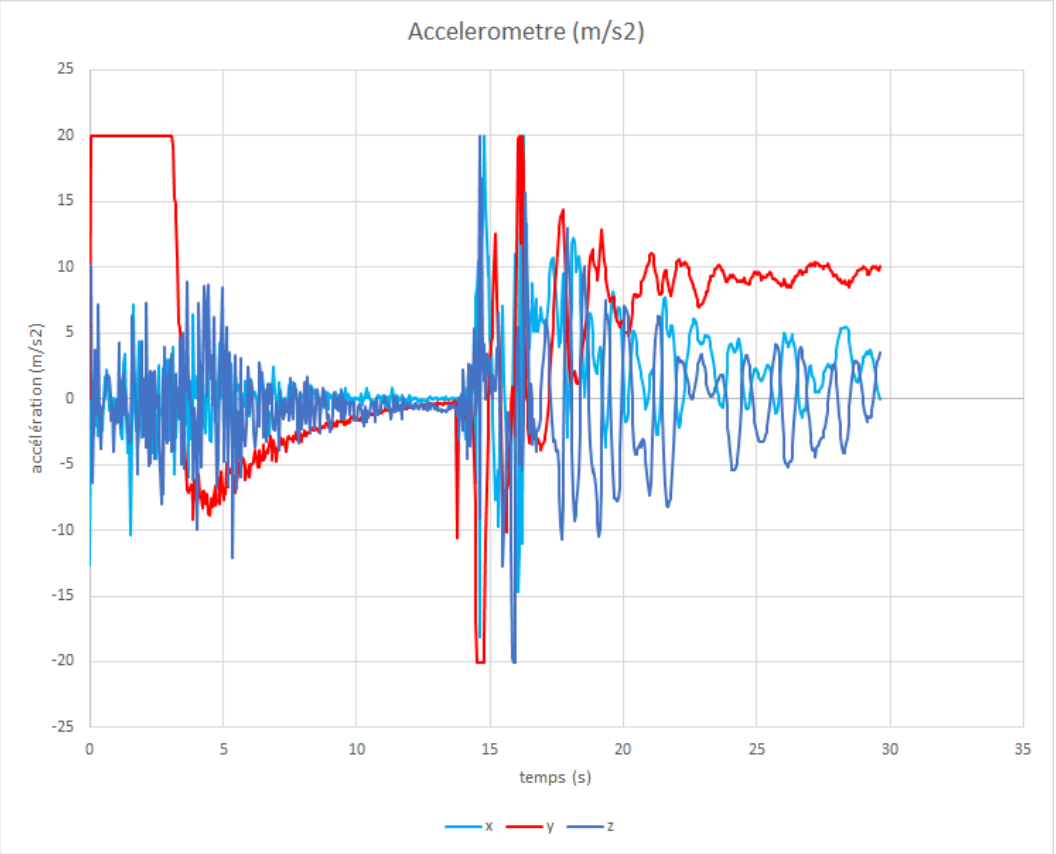

La fusée était toujours en fonctionnement pendant la récupération, le signal sonore de localisation aura fonctionné pendant 1h environ avant que la fusée soit récupérée.

6 Analyse des données recueillies

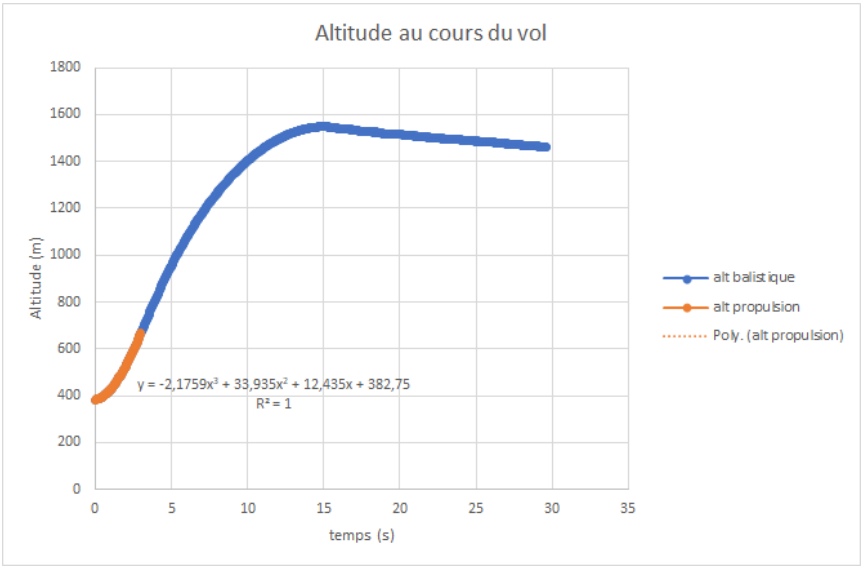
Une fois les données converties en fichier texte, nous avons pu utiliser le logiciel de post traitement pour préparer le classeur Excel.

Il y a malencontreusement eu un problème avec la mesure de déformation, ce qui rend ces données pratiquement inutilisables (les mesures sont quasi constantes tout au long du vol) et nous empêche donc d'étudier ce que nous avions prévu à la base : la comparaison entre le modèle théorique de déformation et les déformations mesurées et le calcul du *Cx*. Nous allons quand même présenter les résultats et la manière de les obtenir.

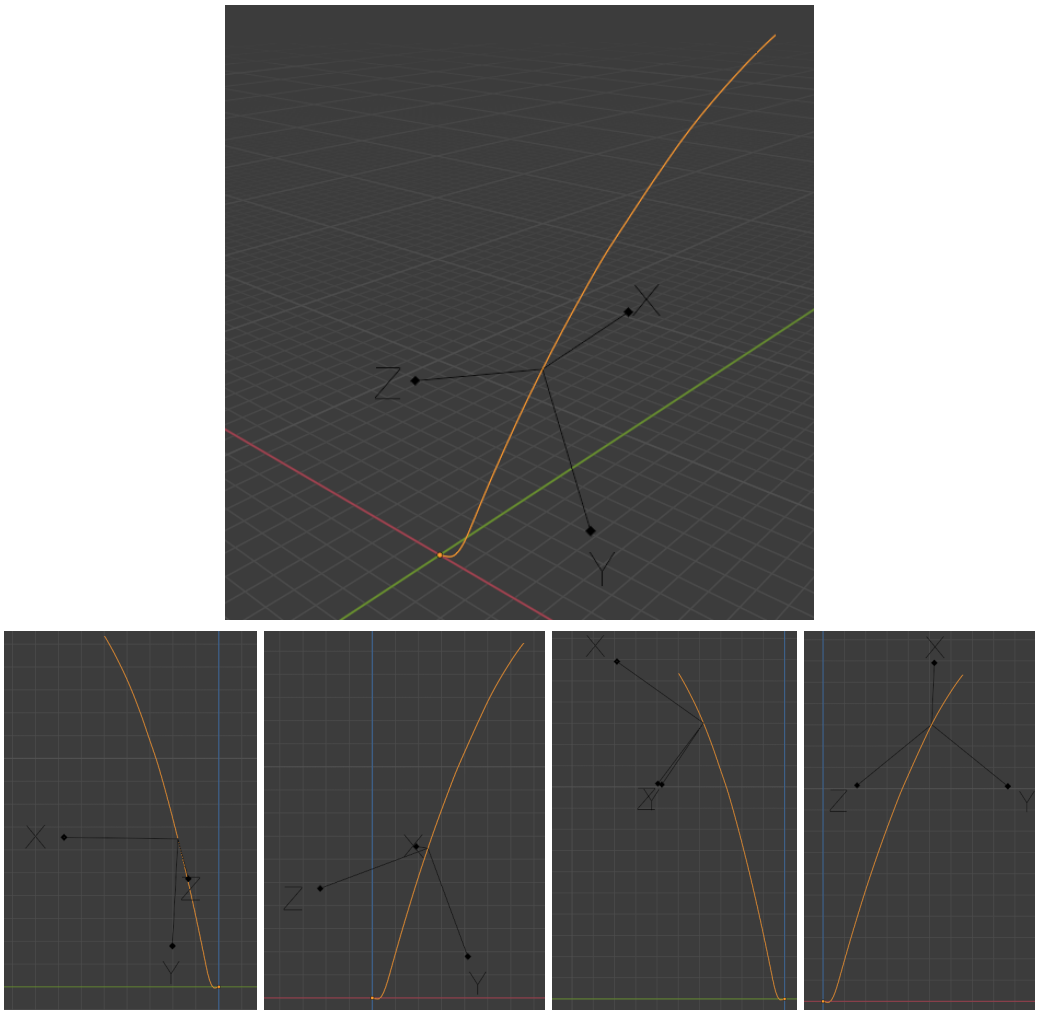
Comme il est possible de constater, l'échantillonnage de la jauge ne s'est pas bien passé, seuls 5 niveaux logiques sont utilisés pour décrire le vol. Un problème d'amplification est sûrement à l'origine de cette erreur puisque le moteur n'a pas sous performé (nous avons sûrement eu de bonnes déformations). La "poussée moteur" est quand même calculée en utilisant les formules énoncées plus haut. L'accélération axiale utilisée dans le calcul ne peut pas provenir de l'accéléromètre car ce dernier a été saturé pendant la phase propulsé à cause d'un mauvais calibrage de gamme. (voir ci dessous).



C'est donc en dérivant l'altitude que nous avons pu recalculer une accélération axiale pendant la phase propulsée en sachant que la fusée était lancée à 85° de l'horizontale et en supposant que la trajectoire était rectiligne pendant les 3 premières secondes.



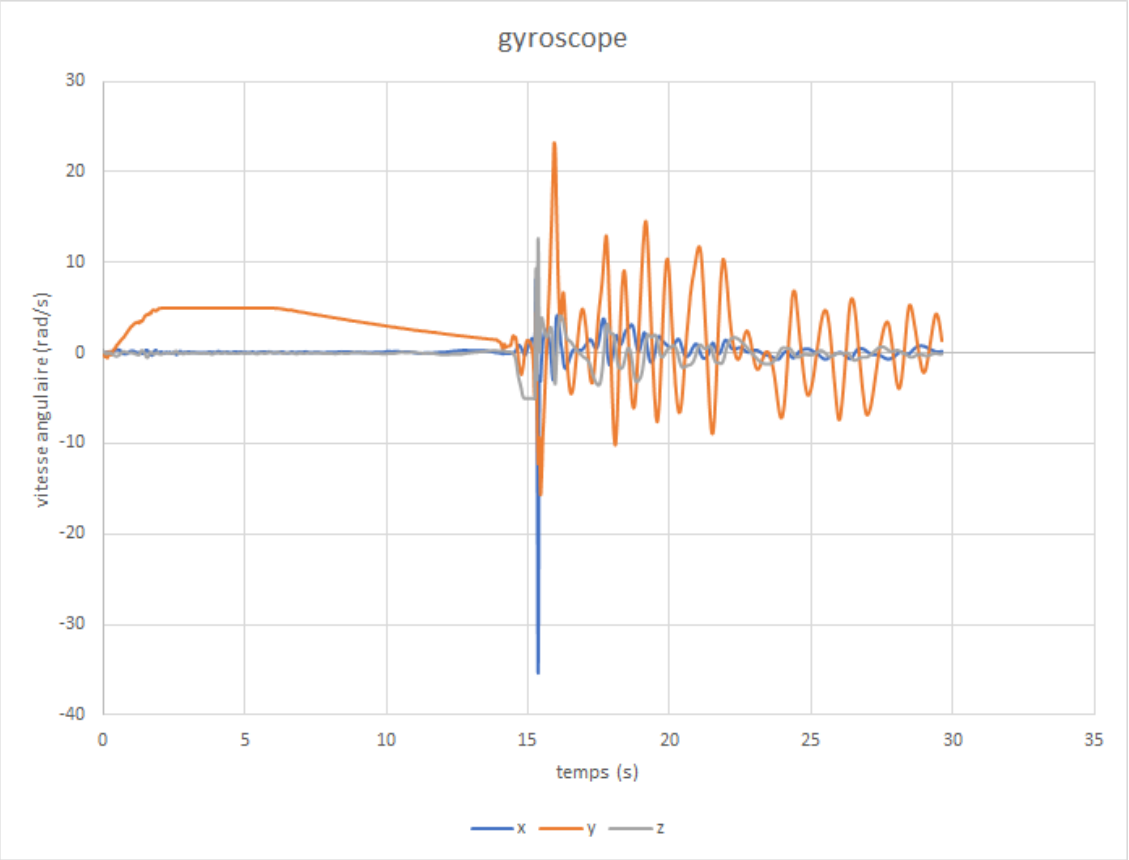
Cette reconstruction de données avait aussi pour but de nous permettre d'expérimenter un peu plus avec l'algorithme AHRS, mais même avec ces nouvelles données les programmes et scripts écrits n'ont pas permis d'obtenir des trajectoires réalistes.

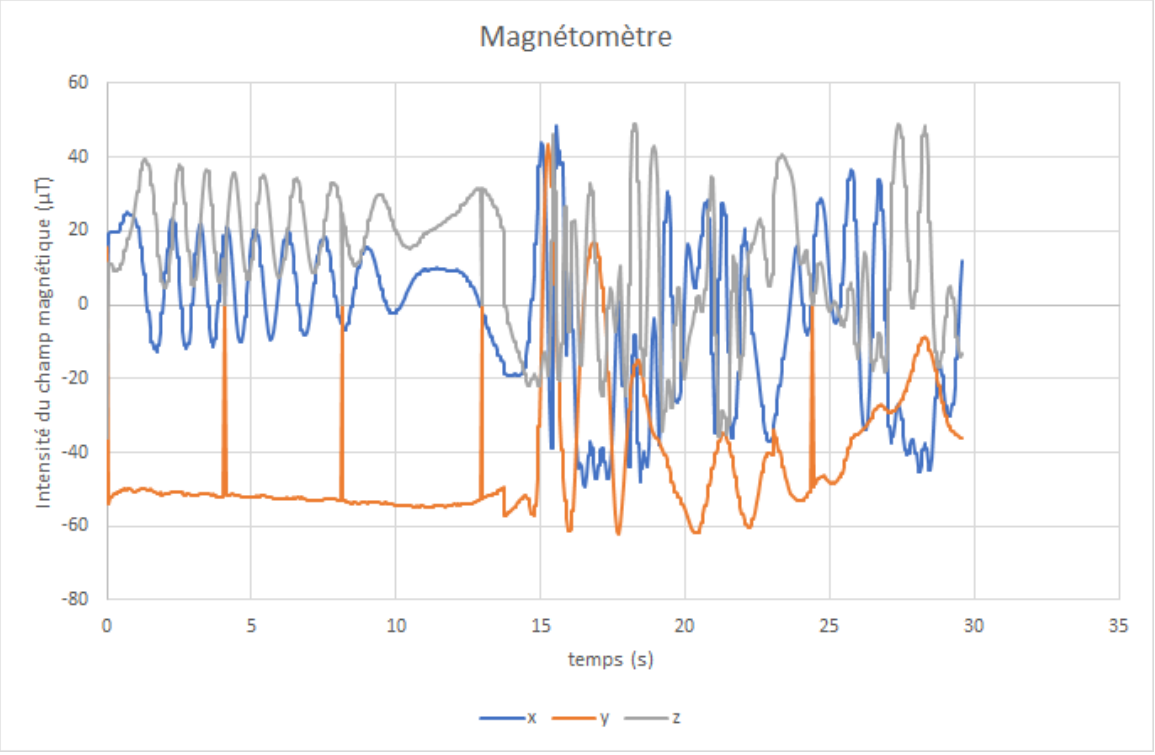


En orange, la trajectoire obtenue avec l'algorithme AHRS et en noir les trois axes de la fusée

La trajectoire réelle de la fusée est une parabole et un des axes de la fusée devrait rester tangent à la trajectoire, ce qui n'est pas le cas dans la trajectoire calculée. Nous pouvons remarquer un léger coude au début de la trajectoire, ce qui indique que l'algorithme détecte bien un changement de direction.

Nous pouvons tout de même présenter les graphiques des données enregistrées, où il est facile de reconnaître les différentes phases de vol :





Dans ces graphes nous pouvons reconnaître les trois phases du vol :

- La phase propulsée : Le premier quart du vol, avec la saturation d'un capteur d'accélération, le bruit sur les deux autres et la vitesse de rotation axiale croissante
- La montée sans propulsion : Le second quart du vol où les accéléromètres se stabilisent vers un profil balistique en apesanteur.
- L'ouverture du parachute : La deuxième moitié du vol, marqué par les grosses secousses causées par l'ouverture du parachute.

Des données du magnétomètre, nous pouvons déduire que la fusée tournait autour de son axe jusqu'à l'ouverture du parachute. Il est intéressant de recouper les informations du magnétomètre et du gyroscope pour voir qu'elles sont cohérentes entre elles.

Un autre problème soulevé a été la mauvaise fréquence d'échantillonnage de la fusée contrairement à ce qui a été demandé. Suite à une erreur de programmation, cette fréquence a été divisée par 4, mais avec le même nombre de points. Cela impliquait de capturer l'ascension en plus de la phase propulsée (un mal pour un bien donc).

7 Remerciements

Pour l'accompagnement de notre projet nous tenions à remercier chaleureusement Mme Chocinski Laurence pour sa motivation, son implication et tout le temps qu'elle passe à nous expliquer comment fonctionne le remboursement du C'Space. En d'autres termes, l'ESP aurait beaucoup de mal à fonctionner correctement sans sa présence et sa représentation vis-à-vis de l'administration.

Merci aussi à Mr Petit et Mr Degeay pour nous avoir permis d'avoir les bagues en aluminium pour Volokna mais aussi d'avoir proposé des idées et des outils pour corriger les erreurs d'usinage. Leur forte expérience permet toujours de simplifier les coûts et la matière.

Merci à mon amie Philippine Goineau pour avoir confectionné le parachute dans des soirées tendues de révisions des partiels d'A3, ce fut un plaisir !

Merci aussi à Célia Soudarin pour avoir cousu la première chaussette libératrice de parachute de l'ESP !

Merci aussi à mon sincère ami Antoine Carton qui m'a supporté tout du long du projet à essayer de faire avancer chaque partie et qui m'a appris à ne pas se lancer dans un moulage de coiffe impliquant du plâtre.

8 Conclusion

Volokna est une fusex mono étage qui devait permettre de bâtir une base solide pour l'avenir de l'ENSMA Space Project. Ce projet a réuni en tout une vingtaine d'élèves issus des promotions 2021, 2022 et 2023. L'objectif était d'abord de concevoir une structure rigide offrant un grand espace pour des expériences à volume variable. Cette structure pourra être utilisée pour d'autres années puisque sa validation au C'Space 2021 a été un succès. L'expérience étant indépendante de la structure nous avons pu découpler les problèmes mais aussi le développement. C'est un choix qui nous a fait gagner un temps précieux.

Le système de récupération, lui aussi développé à partir de zéro, est une grande réussite. On pourra utiliser ce mécanisme très solide et robuste en y ajoutant des raidisseurs pour la trappe et éviter des remarques (constructives) des contrôleurs.

Le séquenceur est aussi un élément qui se voulait très indépendant de l'expérience et du système de récupération choisi. Sa modularité permettait de connecter un solénoïde aussi bien qu'un servo moteur, des prises de communication auraient aussi permis d'y connecter une télémétrie (chose que nous n'avons pas faite). Un aspect très apprécié pendant les phases de débogage du séquenceur est le panneau LCD qui nous a permis de comprendre beaucoup de panne mais aussi de vérifier à de nombreuses reprises que la fusée était encore armée sur la rampe et ce sans action externe ! Le seul défaut du séquenceur reste sa taille. Il est bien trop imposant devant sa fonction. Le futur design pourra reprendre sa conception en resserrant les composants pour le faire tenir dans un endroit plus restreint.

L'experience est le seul point noir de la fusée. C'est un aspect qui a été étudié trop tardivement puisque les jauges n'ont été collé que très tardivement. En conséquence, la

carte électronique n’a été testée réellement qu’un mois avant le C’Space. Toutes les modifications apportées auraient nécessité plus d’attention voir un redesign de la carte. Avec du recul il ne fallait pas chercher à créer une carte d’acquisition de toute pièce mais plutôt de récupérer des briques élémentaires. En effet, il existe des ponts de wheatstone déjà calibrés avec l’amplificateur associé. Un convertisseur Analogique Numérique à haute résolution aurait donné plus de précisions à nos mesures tout en garantissant une autre vitesse d'acquisition !

D’autre part, concernant la programmation, il n’est pas normal de voir que la fréquence réelle d'acquisition était divisée par 4. Ce problème aurait dû être repéré et corrigé avant le vol bien que difficile à reproduire. Cette fréquence d'acquisition est en partie limitée par la centrale inertielle et les temps d’accès avec le protocole i2c. Il pourrait convenir d’utiliser plusieurs petites cartes experience s’occupant chacune d’une tâche en particulier (récolte des données de l’IMU, acquisition des jauges), synchronisées par le même signal de démarrage envoyé par le séquenceur.

Concernant le parachute, sa conception était parfaite à l'exception d’une suspente qui s’est arrachée du côté émerillon. On pourra reprendre le pliage qui a fait ses preuves.

Sur le management de projet, la continuité des actions était compliquée à suivre. La crise du covid nous a forcé à abandonner les fabrications et souder l’équipe via des réunions et des conceptions sur teams avec un tableau blanc. Ces productions n’ont pas été productives mais permettaient de garder le projet en éveil pour survivre jusqu'à l'ouverture de l’atelier. Bien que l’objectif initial d’un an pour le projet aurait pu être respecté, c’est pour l’anniversaire de ses deux ans que Volokna a décollé.

Team C'Space, de gauche à droite Alexis Collet, Zacharie Baeumlin, Anthony Barthélemy
photo prise à 8h02 du matin le jeudi 22 juillet 2021 juste avant le vol

Résumé : Ce rapport a pour but d’expliquer comment concevoir une fusée expérimentale de manière très facile dans l’optique de perpétrer un savoir-faire. De plus, il doit apporter le plus de réponses possibles pour que les personnes l'utilisant ne doivent pas chercher à un autre endroit.
Tous les sujets sont traités, de la programmation à la conception de la structure en passant par le parachute, ainsi cet ouvrage le plus complet possible aidera tout futur ensmatique à réaliser une fusex en 1 an sans problème.

Mots clés : Fusex / Ensma / ESP / Conception / Guide / Explication.

ISAE-ENSMA
Ensma Space Project
1 Avenue Clément ADER
86360 Chasseneuil-du-Poitou

ANNEXE

Code séquenceur

```
#include <Wire.h>
#include <Arduino.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h> // include i/o class header

hd44780_I2Cexp lcd; // declare lcd object: auto locate & config display for hd44780
chip

int pin_exp = A0;
int pin_telem = A1;
int pin_jack = 13;
int pin_buzzer = 10;
int pin_led_V = 8; //sequenceur sous tension
int pin_led_O = 7; //décollage de la fusée
int pin_led_R = 6; //deployment parachute
boolean led_R_status = false;
int pin_recover_ok = A3;
int pin_recover = 9;
int pin_battery_state = A2;

unsigned long T1 = 14000;
unsigned long T2 = 14700;

int ExpReadyCode = 1;
int ExpFailCode = 2;
int TelemReadyCode = 3;
int TelemFailCode = 4;

int battery_limit_voltage = 11;

int level_jack;
int level_recover;

int level_battery;
boolean warning_buzzer = false;
boolean warning_buzzer_recup = false;
unsigned long date_millis_alert = millis();
boolean buzzer_status = false;
boolean led_r_status = false;
unsigned long delay_bip = 500;

boolean flight_in_progress = false;

unsigned long time_up = 0; //date du décollage
boolean recover_in_progress = false;
boolean recover_done = false;
```

```
void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(20, 4);
  // Print a message to the LCD.
  lcd.print("Initializing ...");

  pinMode(pin_jack, INPUT);
  pinMode(pin_exp, OUTPUT);
  pinMode(pin_telem, OUTPUT);
  pinMode(pin_buzzer, OUTPUT);
  pinMode(pin_led_V, OUTPUT);
  pinMode(pin_led_O, OUTPUT);
  pinMode(pin_led_R, OUTPUT);
  pinMode(pin_recover_ok, INPUT);
  pinMode(pin_recover, OUTPUT);
  pinMode(pin_battery_state, INPUT);

  Serial.begin(9600);

  digitalWrite(pin_led_V, HIGH);


  eraze_display();
  displayT1T2();
  delay(2000);
  //test_sequence();
  eraze_display();
  lcd.setCursor(0,0);
  lcd.print("Pret !");

}

void displayT1T2(){
  lcd.setCursor(0, 0);
  lcd.print("T1 :");
  lcd.setCursor(4,0);
  lcd.print(T1);
  lcd.setCursor(9,0);
  lcd.print("ms");
  lcd.setCursor(0, 1);
  lcd.print("T2 :");
  lcd.setCursor(4,1);
  lcd.print(T2);
  lcd.setCursor(9,1);
  lcd.print("ms");
}
```

```
void test_sequence(){
  int r = 0;
  //test exp
  eraze_display();
  lcd.setCursor(0,0);
  lcd.print("Test Exp ...");
  analogWrite(pin_exp, 1023);
  for(int i = 0; i <10; i++){

    if (Serial.available() ){
      r = Serial.read();
      if (r = ExpReadyCode) {
        lcd.setCursor(0,1);
        lcd.print("OK !");
        break;}
    }
    delay(50);
  }
  analogWrite(pin_exp, 0);
  if (r != ExpReadyCode){
  lcd.setCursor(0,1);
  lcd.print('Exp fail !');
  }


  eraze_display();
  lcd.setCursor(0,0);
  lcd.print("Test LED ...");
  digitalWrite(pin_led_O, HIGH);
  digitalWrite(pin_led_R, HIGH);
  delay(1000);
  digitalWrite(pin_led_O, LOW);
  digitalWrite(pin_led_R, LOW);
  lcd.setCursor(0,1);
  lcd.print("OK !");

  //test telem
  eraze_display();
  lcd.setCursor(0,0);
  lcd.print("Test Telem ...");
  analogWrite(pin_telem, 1023);
  for(int i = 0; i <10; i++){

    if (Serial.available() ){
      r = Serial.read();
      if (r = TelemReadyCode) {
        lcd.setCursor(0,1);
        lcd.print("OK !");
```

```
        break;}
    }
    delay(50);
  }
  analogWrite(pin_telem, 0);
  if (r != TelemReadyCode){
  lcd.setCursor(0,1);
  lcd.print("Telem fail !");
  }


  //test buzzer
  eraze_display();
  lcd.setCursor(0,0);
  lcd.print("Test BUZZ ...");
  digitalWrite(pin_buzzer, HIGH);
  delay(1000);
  digitalWrite(pin_buzzer, LOW);
  lcd.setCursor(0,1);
  lcd.print("OK !");

  //test parachute
  eraze_display();
  lcd.setCursor(0,0);
  lcd.print("Test sep...");
  int secure = analogRead(pin_recover_ok);
  lcd.setCursor(0,1);
  if (secure <100) {
    lcd.print("OK !");
  }
 else {
   lcd.print("Fail !");
 }
 delay(1000);


}

void eraze_display(){
  lcd.setCursor(0,0);
  lcd.print("                ");
  lcd.setCursor(0,1);
  lcd.print("                ");
}

boolean battery_check(){
  int level_battery = analogRead(pin_battery_state);
  float Abattery_actual = map(level_battery, 0, 1023, 0, 5);
```

```
  float battery_actual = Abattery_actual / 0.2420;
  if (battery_actual <= battery_limit_voltage){
    eraze_display();
    lcd.setCursor(0,0);
    lcd.print("Batterie faible");
    }
  return battery_actual <= battery_limit_voltage;

}

boolean external_recover(){
  return false;
}



void loop() {



  if (millis()%10000 <100 and (not battery_check()) ) {  //on ne test pas tout le
temps la batterie mais toutes les 10s
    warning_buzzer = false;
  }

  if (millis()%10000 <=100 and battery_check() ) { // on verifie de temps en temps si
la batterie est chargée
    warning_buzzer = true;
  }



  Serial.println(digitalRead(pin_jack));
  if (digitalRead(pin_jack)==HIGH and not flight_in_progress) { // décollage détécté
    flight_in_progress = true;
    digitalWrite(pin_exp, HIGH);
    digitalWrite(pin_telem, HIGH);
    digitalWrite(pin_led_O, HIGH);
    time_up = millis();
    eraze_display();
    lcd.setCursor(0,0);
    lcd.print("Decollage !");

  }

  lcd.setCursor(0,1);
  lcd.print(millis()-time_up);
```

```
  if (((millis()-time_up > T1) and (millis()-time_up < T2)) and flight_in_progress) {
//sauvegarde dans l'intervalle
    recover_in_progress = external_recover();
  }

  if (millis()-time_up >= T2 and flight_in_progress) {//on force le deployement
    recover_in_progress = true;
  }

  if (recover_in_progress and not recover_done ){ // deployement
    //recover_servo.write(servo_final_rotation);si la recuperation se fait avec un
servomoteur
    digitalWrite(pin_recover, HIGH);
    delay(10000);
    digitalWrite(pin_recover, LOW);
    digitalWrite(pin_led_R, HIGH);
    recover_done = true;

  }

  if (recover_done and digitalRead(pin_recover_ok) == HIGH){// cas normal
      eraze_display();
      lcd.setCursor(0,0);
      lcd.print("Separation OK !");
      warning_buzzer_recup = false;
      digitalWrite(pin_led_R, HIGH);


    }
  if (recover_done){// cas normal
      eraze_display();
      lcd.setCursor(0,0);
      lcd.print("SEPARATION !");
      warning_buzzer_recup = true;
  }
  else{
      warning_buzzer_recup = false;
      if (not flight_in_progress){
        lcd.setCursor(0,0);
        lcd.print("Pret !        ");
        }
  }


  if (warning_buzzer or warning_buzzer_recup) { //alerte batterie ou recuperation
    if (millis() - date_millis_alert > delay_bip)  {
      buzzer_status = not buzzer_status;
      digitalWrite(pin_buzzer, buzzer_status);
```

```
      date_millis_alert = millis();

      if (warning_buzzer_recup) {
        led_r_status = not led_r_status;
        digitalWrite(pin_led_R, led_r_status);
        }
    }
  }
  else
  {
    digitalWrite(pin_buzzer, LOW);
    }
}
```

Code expérience

```
#include <Arduino.h>
#include <i2c_t3.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>
#include <Adafruit_BMP085_U.h>
#include <Adafruit_L3GD20_U.h>
#include <Adafruit_10DOF.h>
#include <SD.h>
#include <SPI.h>
#include <String.h>

Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(30301);
Adafruit_LSM303_Mag_Unified   mag   = Adafruit_LSM303_Mag_Unified(30302);
Adafruit_BMP085_Unified       bmp   = Adafruit_BMP085_Unified(18001);
Adafruit_L3GD20_Unified       gyro  = Adafruit_L3GD20_Unified(20);

const int ADDRESS_EEPROM[8] = {0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57};

#define ADDRESS_ACC  0x50
const int pin_capteurs[5] = {A1, A2, A3, A4, A5};

const int freq_echant = 100; //Hz
const int record_time = 7400; //ms
const int n_donnees = int(freq_echant*record_time/1000.0)+1;

const float tension_nominale_battery = 12.0; // V
const float seuil_limit_bat = 10.0; // V
const float rapport_pont_diviseur_batterie = 0.4387;

int off_set_jauges[5] = {0, 0, 0, 0, 0};
const int pin_battery = A21;

//DÃ(c)finition des tableaux de donnÃ(c)es
int data_strain[5][n_donnees] = {0}; //5 cases pour les capteurs
float data_acc[3][n_donnees] = {0}; // donnÃ(c)es accÃ(c)lÃ(c)ration x, y, z repÃ¨re
fusÃ(c)e
float data_gyro[3][n_donnees] = {0}; // donnÃ(c)es accÃ(c)lÃ(c)ration x, y, z,
repÃ¨re fusÃ(c)e
float data_time[n_donnees] = {0.0}; // prise prÃ(c)cise du temps.
float data_mag[3][n_donnees] = {0.0}; // magnetométre
float data_alt[n_donnees] = {0.0}; //altitude

#define pin_buzzer 23
int date_buzzer = 0;
```

```
#define pin_wheatstone 24
#define pin_led_erreur 13
#define pin_button A22//67// verifier ca
#define pin_WP 39
#define pin_SDA_EEPROM 38
#define pin_SCL_EEPRIM 37
#define pin_TX_seq 32
#define pin_RX_seq 31
#define pin_led_record 28
#define pin_led_save 27
#define pin_SCL_ACC 3
#define pin_SDA_ACC 4
#define pin_dem_exp A7

byte highAddress;
byte lowAddress;
byte data;

bool etat_buzzer = false;
bool alarm_buzzer = false;
int delay_bip_alarm = 100;

unsigned long date_rising_edge = 0;
unsigned long date_falling_edge = 0;

bool on_record = false;
bool ready_ = false;
bool etat_led_ready = false;
bool button_push = false;
unsigned long date_led_ready = 0;
int delay_ready_flash = 500;


void setup() {

  Serial.begin(9600);
  delay(1000);
  Serial.println("BEGIN SETUP");
  Wire2.begin();
  Wire1.begin();

  if (!gyro.begin())
  {
    Serial.println("Failed to autodetect gyro type!");

  }
  else
  {
```

```
    Serial.println("Gyro Connected !");
  }

  gyro.enableAutoRange(true);

  if (! accel.begin()) {
    Serial.println("LSM303 Couldnt start");

  }
  Serial.println("LSM303 found !");

//accel.setRange(LSM303_RANGE_8_G);

//  Serial.print("Range = "); Serial.print(2 << accel.getRange());
// Serial.println("G");

  if(!mag.begin())
  {
    /* There was a problem detecting the LSM303 ... check your connections */
    Serial.println("Ooops, no LSM303 Mag detected ... Check your wiring!");
  }
  else{
    Serial.println("Mag connected !");
    }

  if(!bmp.begin())
  {
    /* There was a problem detecting the BMP085 ... check your connections */
    Serial.print("Ooops, no BMP085 detected ... Check your wiring or I2C ADDR!");
  }
  else{
    Serial.println("BMP085 Connected !");
    }

  pinMode(pin_battery, INPUT);
  pinMode(pin_buzzer, OUTPUT);
  pinMode(pin_button, INPUT);
  pinMode(pin_led_erreur, OUTPUT);
  pinMode(pin_button, INPUT);
  pinMode(pin_WP, OUTPUT);
  pinMode(pin_led_save, OUTPUT);
  pinMode(pin_led_record, OUTPUT);
  pinMode(pin_wheatstone, OUTPUT);
  pinMode(pin_dem_exp, INPUT);

  int i = 0;
  for (i = 0; i < 5; i++)
  {
```

```
    pinMode(pin_capteurs[i], INPUT);
  }
  ready_ = true;

  digitalWrite(pin_led_erreur, HIGH);
  digitalWrite(pin_led_save, HIGH);
  digitalWrite(pin_led_record, HIGH);
  delay(500);
  digitalWrite(pin_led_erreur, LOW);
  digitalWrite(pin_led_save, LOW);
  digitalWrite(pin_led_record, LOW);
  scan_i2c();
  digitalWrite(pin_WP, HIGH);

  Serial.println("\n");
  Serial.println("FIN SETUP");

}//fin setup

void eepromWrite(int index, byte highAddress, byte lowAddress, byte data) {

  Wire1.beginTransmission(ADDRESS_EEPROM[index]);
  delay(10);
  Wire1.write(highAddress);
  delay(10);
  Wire1.write(lowAddress);
  delay(10);
  Wire1.write(data);
  delay(10);
  Wire1.endTransmission();
  delay(10);
}

byte eepromRead(int index, byte highAddress, byte lowAddress) {

  Wire1.beginTransmission(ADDRESS_EEPROM[index]);
  delay(10);
  Wire1.write(highAddress);
  delay(10);
  Wire1.write(lowAddress);
  delay(10);
  Wire1.endTransmission();
  delay(10);
  Wire1.requestFrom(ADDRESS_EEPROM[index], 1); // one byte requested
  delay(10);
  while (!Wire1.available())
  {
    delay(10);
```

```
  }
  delay(10);
  return Wire1.read();

}
void eepromReadAll(){

  char data = 0;
  int i =0;
  int rank = 0;
  int index_EE_acc = 5;
  int index_EE_gyro = 6;
  int index_EE_time = 7;
  for(i=0; i<5; i++){
    Serial.print("Jauge");
    Serial.println(i);
    for(rank = 0; rank<n_donnees*2; rank++){
      data = eepromRead(i, highByte(rank), lowByte(rank));
      Serial.println(data, HEX);
      }
    }

  Serial.println("Acc");
  for(rank=0; rank<12*n_donnees; rank++){
      data = eepromRead(index_EE_acc, highByte(rank), lowByte(rank));
      Serial.println(data, HEX);
    }
  Serial.println("Gyro");
  for(rank=0; rank<12*n_donnees; rank++){
      data = eepromRead(index_EE_gyro, highByte(rank), lowByte(rank));
      Serial.println(data, HEX);
    }
  Serial.println("Tps");
  for(rank=0; rank<2*n_donnees; rank++){
      data = eepromRead(index_EE_time, highByte(rank), lowByte(rank));
      Serial.println(data, HEX);
    }
}

bool check_battery() {

  int niveau = analogRead(pin_battery);
  float tension_lue = float(niveau)*3.3/1023;
  //Serial.println("tension batterie (V):");
  //Serial.println(tension_lue / rapport_pont_diviseur_batterie);
  return false;
}
```

```
void float2Bytes(float val,byte* bytes_array){
  // Create union of shared memory space
  union {
    float float_variable;
    byte temp_array[4];
  } u;
  // Overite bytes of union with float variable
  u.float_variable = val;
  // Assign bytes to input array
  memcpy(bytes_array, u.temp_array, 4);


}


void saveFloat(int index_EE, int begin_adress, float dataF){
  int i = 0;
  byte bytes_f[4];
  float2Bytes(dataF, &bytes_f[0]);
  for(i=0; i<4; i++){
    eepromWrite(index_EE, highByte(begin_adress+i),lowByte(begin_adress+i),
bytes_f[i]);
  //Serial.print(dataF);
  //Serial.print(" ");
  //Serial.println(bytes_f[0], HEX);
  //Serial.println(bytes_f[1], HEX);
  //Serial.println(bytes_f[2], HEX);
  //Serial.println(bytes_f[3], HEX);
  //Serial.println(" Fin");
    }
  }

void save_eeprom() {

  int i = 0;
  int rank = 0;
  int max_addr = pow(2, 7);
  int index_EE_acc = 5;
  int index_EE_gyro = 6;
  int index_EE_time = 7;
  byte addr[2] = {0x0, 0x0};
  digitalWrite(pin_WP, LOW);
  for (i = 0; i < 5; i++)//adresse EEPROM jauges 0 1 2 3 4
  {
    for (rank = 0; rank < n_donnees; rank++)
    {
      Serial.println("Sauv JAUG tps suiv");
      addr[0] = lowByte(2*rank);
```

```
      addr[1] = highByte(2*rank);
      eepromWrite(i, addr[1], addr[0], lowByte(data_strain[i][rank]));
      addr[0] = lowByte(2*rank+1);
      addr[1] = highByte(2*rank+1);
      eepromWrite(i, addr[1], addr[0], highByte(data_strain[i][rank]));
    }
  }
  for (rank = 0; rank < n_donnees; rank ++) { //les donnÃ(c)es d'un meme temps sont
stockÃ(c) Ã  la suite acc_x, acc_y, acc_z
    Serial.println("Sauv IMU tps suiv");
    saveFloat(index_EE_acc, 12*rank, data_acc[0][rank]);
    saveFloat(index_EE_acc, 12*rank+4, data_acc[1][rank]);
    saveFloat(index_EE_acc, 12*rank+8, data_acc[2][rank]);

    saveFloat(index_EE_gyro, 12*rank, data_gyro[0][rank]);
    saveFloat(index_EE_gyro, 12*rank+4, data_gyro[1][rank]);
    saveFloat(index_EE_gyro, 12*rank+8, data_gyro[2][rank]);
  }

  for (rank = 0; rank < n_donnees; rank++) {
    Serial.println("Sauv tps suiv");
      addr[0] = lowByte(2*rank);
      addr[1] = highByte(2*rank);
      eepromWrite(index_EE_time, addr[1], addr[0],
lowByte(int(data_time[rank]*1000)));
      addr[0] = lowByte(2*rank+1);
      addr[1] = highByte(2*rank+1);
      eepromWrite(index_EE_time, addr[1], addr[0],
highByte(int(data_time[rank]*1000)));
  }
  digitalWrite(pin_WP, HIGH);
}

void scan_i2c(){

    int nDevices = 0;

  Serial.println("Scanning...");

  for (byte address = 1; address < 127; ++address) {
    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    Wire1.beginTransmission(address);
    byte error = Wire1.endTransmission();

    if (error == 0) {
      Serial.print("I2C device found at address 0x");
```

```
      if (address < 16) {
        Serial.print("0");
      }
      Serial.print(address, HEX);
      Serial.println("  !");

      ++nDevices;
    } else if (error == 4) {
      Serial.print("Unknown error at address 0x");
      if (address < 16) {
        Serial.print("0");
      }
      Serial.println(address, HEX);
    }
  }
  if (nDevices == 0) {
    Serial.println("No I2C devices found\n");
  } else {
    Serial.println("done\n");
  }
  }

void set_offset() {
  int i = 0;
  digitalWrite(pin_wheatstone, HIGH);
  for (i = 0; i < 4; i++) {
    off_set_jauges[i] = analogRead(pin_capteurs[i]);
  }
  digitalWrite(pin_wheatstone, LOW);
}

int lecture_jauge(int i) {

  return analogRead(pin_capteurs[i]) - off_set_jauges[i];
}


void record() {//enregistrement du vol jusqu'a la fin
  float t = 0.0;
  float T = 1 / float(freq_echant); //en seconde
  int j,i = 0;
  j = 0;
  sensors_event_t event;
  float temperature;
  float seaLevelPressure;
  unsigned long t_ref = millis();
```

```
  while (j < n_donnees) {
    t_ref = millis();
    digitalWrite(pin_wheatstone, HIGH);
    for (i = 0; i < 5; i++)
    {
      data_strain[i][j] = lecture_jauge(i);
      Serial.print("Jauge ");
      Serial.print(i);
      Serial.print(" : ");
      Serial.println(data_strain[i][j]);
    }
    digitalWrite(pin_wheatstone, LOW);

    t += T;
    j += 1;
    accel.getEvent(&event);
    data_acc[0][j] = event.acceleration.x;
    data_acc[1][j] = event.acceleration.y;
    data_acc[2][j] = event.acceleration.z;

    gyro.getEvent(&event);
    data_gyro[0][j] = event.gyro.x;
    data_gyro[1][j] = event.gyro.y;
    data_gyro[2][j] = event.gyro.z;

    mag.getEvent(&event);
    data_mag[0][j] = event.magnetic.x;
    data_mag[1][j] = event.magnetic.y;
    data_mag[2][j] = event.magnetic.z;

    bmp.getEvent(&event);
    bmp.getTemperature(&temperature);
    seaLevelPressure = SENSORS_PRESSURE_SEALEVELHPA;
    data_alt[j] = bmp.pressureToAltitude(seaLevelPressure, event.pressure,
temperature);

    Serial.print("Acc x :");
    Serial.println(data_acc[0][j]);
    Serial.print("Gyro x :");
    Serial.println(data_gyro[0][j]);
    Serial.print("Mag x:");
    Serial.println(data_mag[0][j]);
    Serial.print("Alt :");
    Serial.println(data_alt[j]);
    data_time[j] = t;
    if (millis() - t_ref < T * 1000) { //millis()-t_ref est le temps de l'operation
      delay( T * 1000 - millis() + t_ref); // on attend le temps restant
    }
```

```
  }

}

void SDsave(){
   Serial.print("Initializing SD card...");
  if (!SD.begin(BUILTIN_SDCARD)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

  // open the file.
  File myFile;
  const char nom[] = "save";
  String file_name;
  int file_number = 1;


  do{
    file_name= "";
    file_name.concat(nom);
    file_name.concat(file_number);
    file_name.concat(".txt");
    file_number++;
    Serial.print("try to open file : ");
    Serial.println(file_name);
    }while(SD.exists(file_name.c_str()));
  Serial.println("File name :");
  Serial.println(file_name);
  myFile = SD.open(file_name.c_str(), FILE_WRITE);


  // if the file opened okay, write to it:
  if (myFile) {
    Serial.println("Enregistrement sur la carte ...");

    for(int i=0; i<5; i++){
      myFile.print("Jauge Vlue en V:");
      myFile.println(i);
      for(int j = 0; j < n_donnees; j++){
        myFile.println(float(data_strain[i][j])*3.3/1023);
        }
      }

    for(int i=0; i<3; i++){
      myFile.print("Accelerometre coordonnee m/s^2:");
```

```
      myFile.println(i);
      for(int j = 0; j < n_donnees; j++){
        myFile.println(data_acc[i][j]);
        }
      }
    for(int i=0; i<3; i++){
      myFile.print("Gyroscope coordonnee rad/s :");
      myFile.println(i);
      for(int j = 0; j < n_donnees; j++){
        myFile.println(data_gyro[i][j]);
        }
      }

    for(int i=0; i<3; i++){
      myFile.print("Mag Field coordonnee (uT) :");
      myFile.println(i);
      for(int j = 0; j < n_donnees; j++){
        myFile.println(data_mag[i][j]);
        }
      }

     myFile.println("Alt (m) :");
    for(int j = 0; j < n_donnees; j++){
        myFile.println(data_alt[j]);
        }

    myFile.println("temps (s) :");
    for(int j = 0; j < n_donnees; j++){
        myFile.println(data_time[j]);
        }

// close the file:
    myFile.close();
    Serial.println("done.");
  } else {
    // if the file didn't open, print an error:
    Serial.println(strcat("error opening",nom));
  }

  // re-open the file for reading:
  myFile = SD.open(nom);
  if (myFile) {
    Serial.println("Contenu du fichier :");

    // read from the file until there's nothing else in it:
    while (myFile.available()) {
      Serial.write(myFile.read());
    }
```

```
    // close the file:
    myFile.close();
  } else {
    // if the file didn't open, print an error:
    Serial.println(strcat("error opening",nom));
  }
}

void loop() {
  // put your main code here, to run repeatedly:

  if (millis() % 10000 <= 25) { // toute les 10s

    alarm_buzzer = check_battery(); //on vÃ(c)rifie le niveau de batterie
  }

  if (alarm_buzzer){
    if(millis() - date_buzzer > signed(delay_bip_alarm) ) { // bip en alternance si
l'alarme est dÃ(c)clanchÃ(c)e
    etat_buzzer = not etat_buzzer;
    date_buzzer = millis();
    digitalWrite(pin_buzzer, etat_buzzer);
  }}
  else{
    digitalWrite(pin_buzzer, LOW);
  }


  if (digitalRead(pin_dem_exp) == HIGH) { //demarage exp
    on_record = true;
    digitalWrite(pin_led_record, HIGH);
    ready_ = false;
    Serial.println("Demarrage Exp...");
    record();
    digitalWrite(pin_led_record, LOW);
    on_record = false;
    SDsave();
  }

  if (analogRead(pin_button)> 500 and not button_push and not on_record) { // remise
Ã  zero offset jauges appui court // save EEPROM appui long, on interdit le offset
pendant le vol
    date_rising_edge = millis();
    button_push = true;
  }
  if (analogRead(pin_button)<100 and button_push and not on_record){
    date_falling_edge = millis();
```

```
    if (date_falling_edge - date_rising_edge > 1000) { // appuis long -> save
(1000ms)
      digitalWrite(pin_led_save, HIGH);
      Serial.println("Sauvegarde sur EEPROM");
      save_eeprom();
      digitalWrite(pin_led_save, LOW);
      eepromReadAll();
    }
    else { //appui court -> offset zero
      if (date_falling_edge - date_rising_edge > 50) {
      digitalWrite(pin_led_save, HIGH);
      Serial.println("Offset jauges ...");
      set_offset();
      delay(100);
      digitalWrite(pin_led_save, LOW);
      delay(100);
      digitalWrite(pin_led_save, HIGH);
      delay(100);
      digitalWrite(pin_led_save, LOW);
    }}
    button_push = false;
  }

  if (ready_){
    if( millis() - date_led_ready > signed(delay_ready_flash)) { //clignotement ready
      etat_led_ready = not etat_led_ready;
      date_led_ready = millis();
      digitalWrite(pin_led_erreur, etat_led_ready);
    }
  }


}
```

Code banc d'essai

Basiquement le code du banc d'essai est le même que celui de la carte expérience. Les données ne sont pas enregistrées mais transmises à un fréquence plus faible via le port série.

Liste des élèves participants

A3 :
Alexis Collet *CDP*
Antoine Carton
Yvan Dilem
Victor De Jesus
Baptiste Egreteau
Nicolas Moscatchelli

A2 :
Clément Lingois *CDP*
Tom Roy
Valentin Bonnet
Rémi Lemaire
Louis Collignon

A1 :
Zacharie Baeumlin
Anthony Barthélemy
Félicien Christiaens
Pierre Bobon

STABILITO

Stabilité de fusée à ailerons

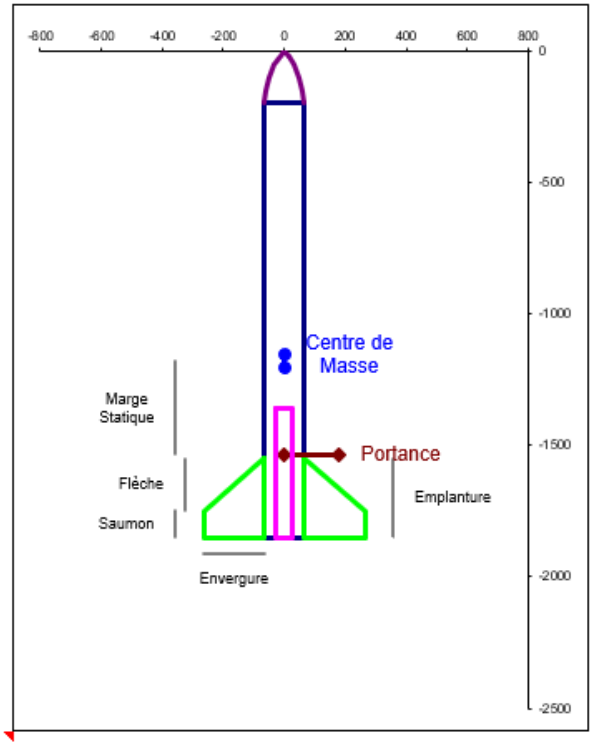
Remplir les cases jaunes

Fusée	
Nom	Volokna
Club	ESP
Type	Fusée expérimentale.
Masse	8000 g sans propu
Centre de Masse	1116 mm sans propu
Longueur totale	1850 mm

Propulseur	
Type	Barasinga (Pro54-5G)
Position du bas	1850 mm

Coiffe	
Forme	Ogivale (pointue)
Hauteur	200 mm
Diamètre	130 mm

Ailerons	Mono-empennage,
Emplanture 'm'	300 mm
Saumon 'n'	100 mm
Flèche 'p'	200 mm
Envergure 'E'	200 mm
Epaisseur	3 mm
Nombre	4
Position du bas	1850 mm



29/07/2021	Min	Résultats		Max
Finesse	10	14,2		35
Portance	15	20,9		40
Marge Stat.	2 D	2,56 D	2,94 D	6 D
Couple	40	53,3	61,3	100
XCp		1535 mm		
MS /L		18% L	21% L	

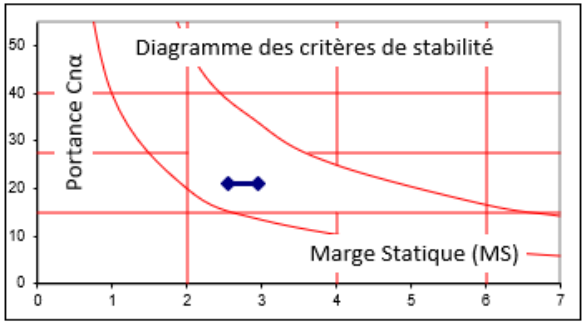
STABLE

Language/Langue	Français

Fusée mono-diamètre,

	Propu plein	Propu vide	Sans propu
Masse propu	1,685 kg	0,652 kg	-
CdM propu	250 mm	240 mm	-
Masse fusée	9,685 kg	8,652 kg	8 kg
CdM fusée	1202 mm	1153 mm	1116 mm

	XCp	Cnα
Coiffe	93 mm	2,0
Ailerons	1688 mm	18,9



Stabtraj à jour

TRAJECTO

Trajectographie de fusée

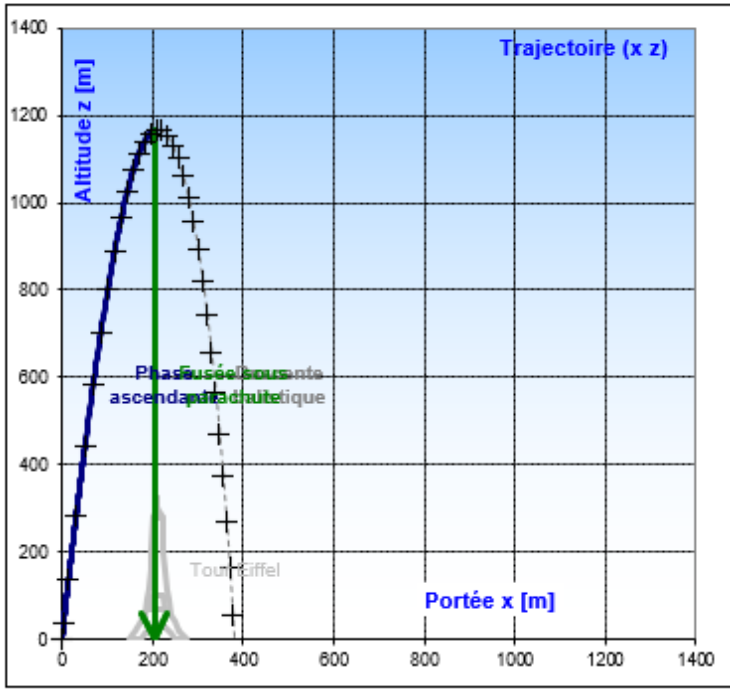
Remplir les cases jaunes

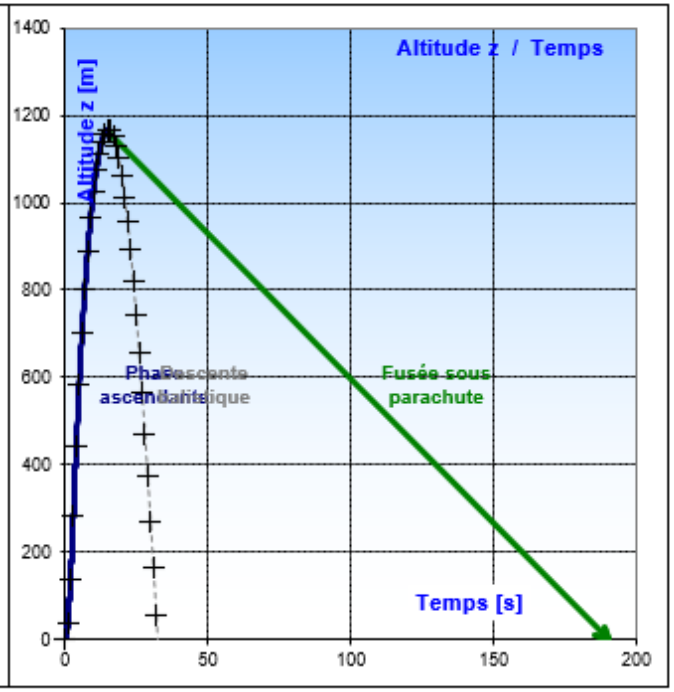
	Fusée
Nom	Volokna
Club	ESP
Masse totale	9,685 kg
Propulseur	Barasinga (Pro54-5G)

	Traînée Aérdynamique
Surface Réf.	0,015673 m²
Cx	0,55

	Rampe de Lancement
Longueur	4 m
Élévation	85 °
Altitude	0 m

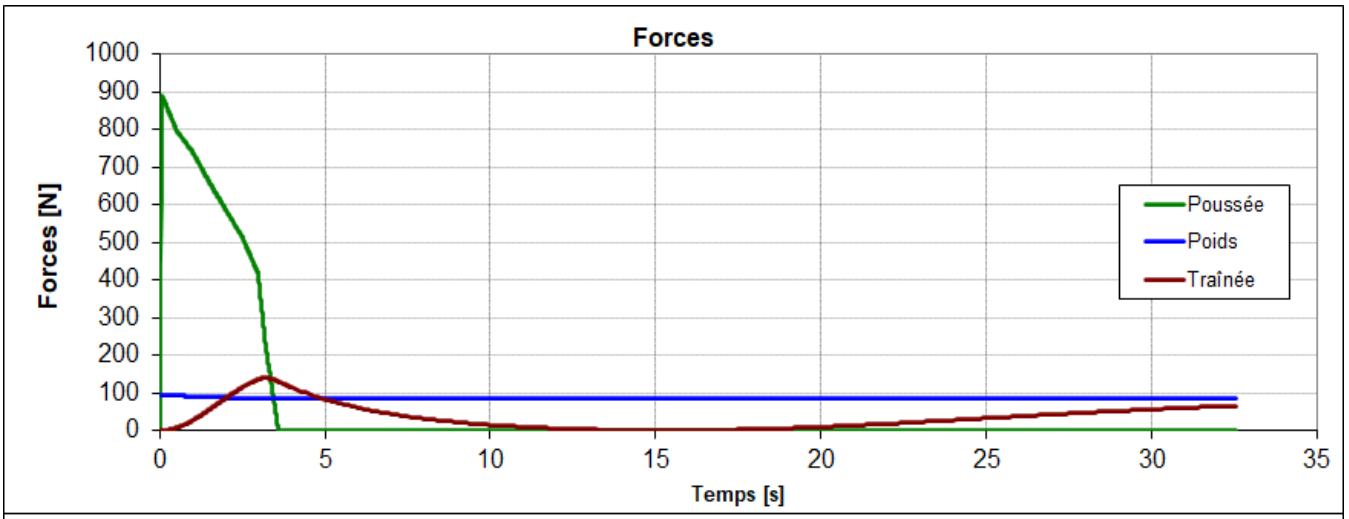
	DescenteSousParachute	
	Fusée	0 satellite
Masse	8,652 kg	
Dépotage	N/A	
Ouverture para	14,7 s	
Surface para	3,14 m²	
Cx parachute	1	
Vitesse du vent	5 m/s	
Vitesse descente	6,6 m/s	
Durée descente	175 s	
Durée du vol	190 s	
Déport latéral	± 877 m	

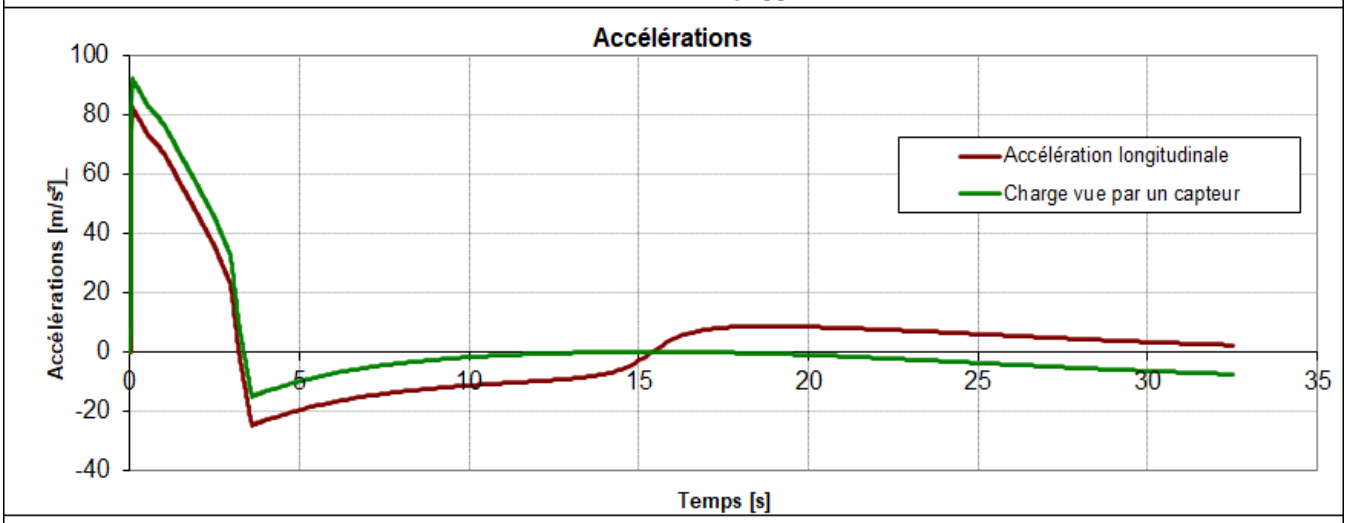


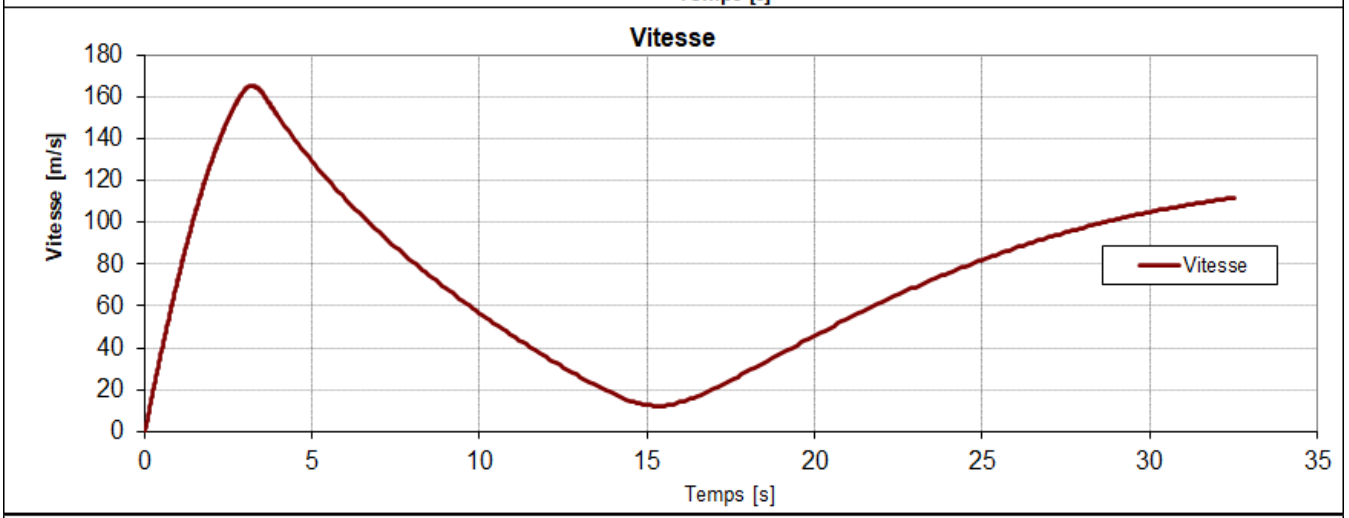


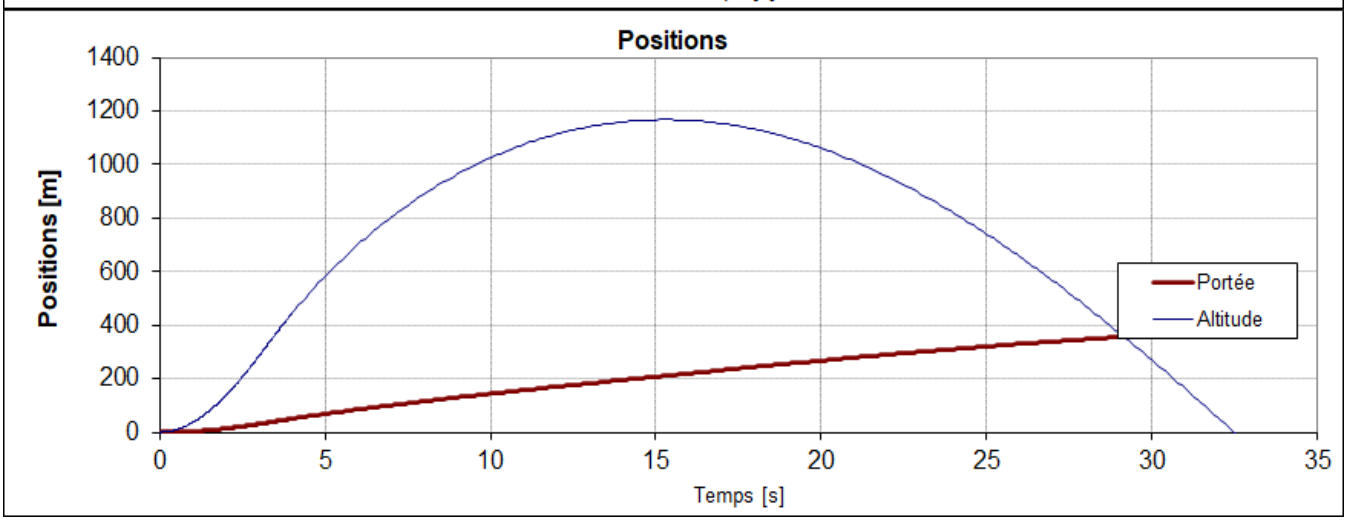
29/07/2021	Temps	Altitude z	Portée x	Vitesse	Accélération	Efforts
Sortie de Rampe				25,0 m/s		
Vit max & Acc max				165 m/s	82 m/s²	
Culmination, Apogée	15,3 s	1167 m	211 m	12 m/s		
Ouverture parachute fusée	14,7 s	1165 m	204 m	14 m/s		360,2 N
Impact balistique	32,5 s		380 m	111,7 m/s		53998 J

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


Forces
Forces [N]
Temps [s]
Poussée
Poids
Traînée


Accélérations
Accélérations [m/s²]
Temps [s]
Accélération longitudinale
Charge vue par un capteur


Vitesse
Vitesse [m/s]
Temps [s]
Vitesse


Positions
Positions [m]
Temps [s]
Portée
Altitude

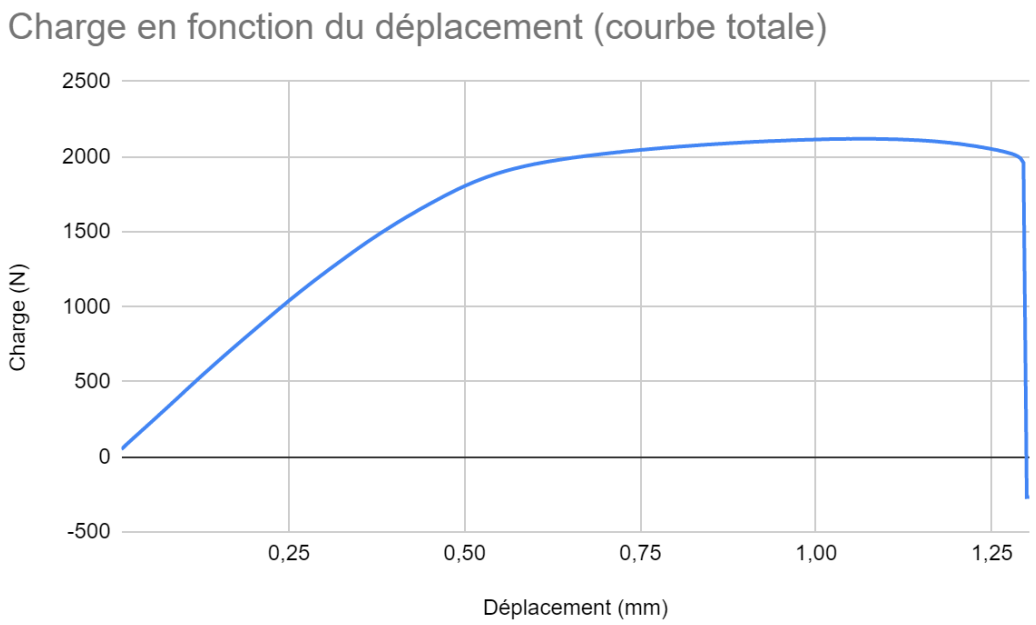
Méthode de mesure du module d'young

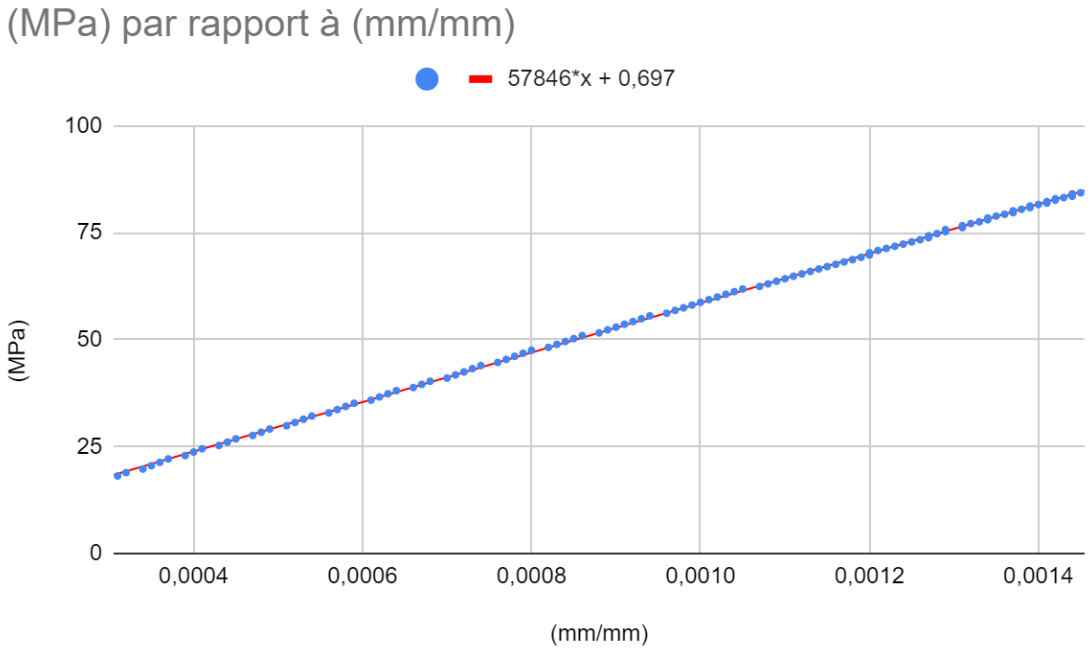
La fusex PLUME du NAASC dont le développement a débuté en octobre 2020 est une fusée multi-établissement, qui se doit d'être irréprochable du point de vue de la méthode et du management. Cette fusex se propose entre autres de reprendre la structure de la partie basse de Volokna. Pour prouver la résistance de la structure, l'équipe de PLUME devait mesurer le module d'young et la résistance à la traction de ses profilés en U. Les profilés en U sont ceux que Volokna utilise et qui sont achetés chez Leroy Merlin.

Méthode de traitement :
Module d'Young
Pour les éprouvettes avec extenso :
1) Tracer la charge en fonction du déplacement
2) Réduire à la zone d'élasticité linéaire en modifiant la plage de donnée par essais successifs
3) Tracer la contrainte en fonction de la déformation sur cette plage de linéarité
4) Régression linéaire et noter le module d'Young (la pente)

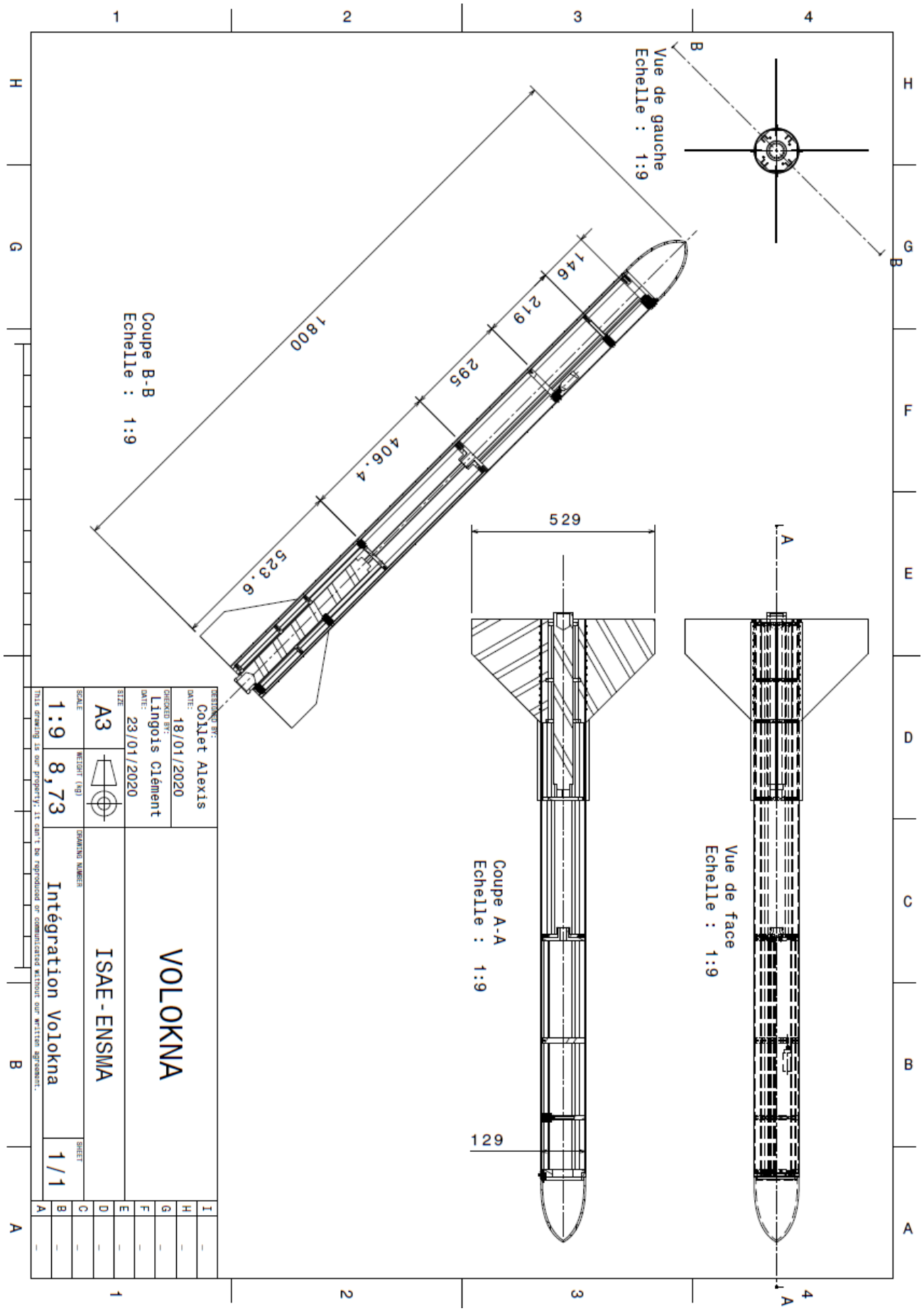
Éprouvettes	Longues	
Extenso	Sans	Avec
1		E (MPa) = 57846
2		E (MPa) =54895
3		E (MPa) =57685
Moyenne		E (MPa) =56808,66667
Écart-type		1659,237877 MPa

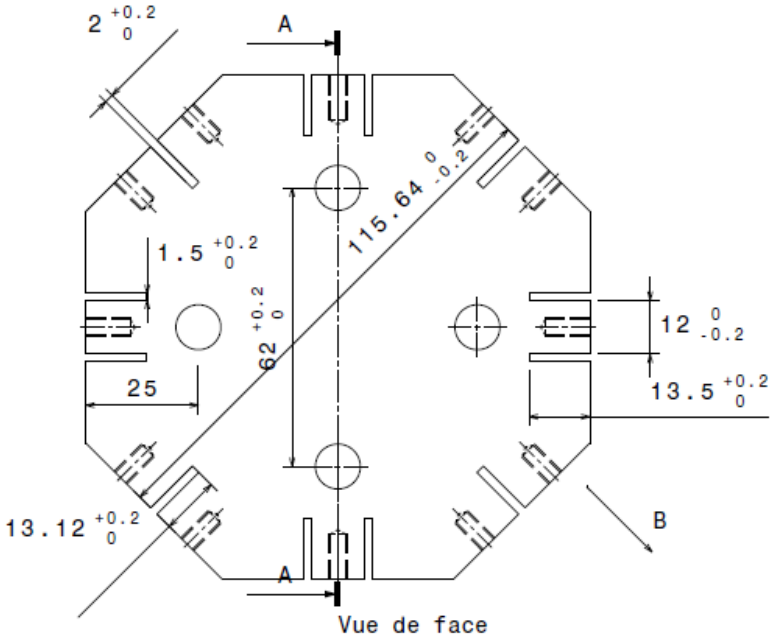
Les données sur des éprouvettes plus courtes ne sont pas satisfaisantes, le module d'young est alors calculés à partir de 3 essais avec des extensomètres à couteau. Un essai typique jusqu'a la striction est bien décrit sur le graphique contrainte / déformation.


Charge en fonction du déplacement (courbe totale)
2500
2000
1500
1000
500
0
-500
Charge (N)
0,25
0,50
0,75
1,00
1,25
Déplacement (mm)

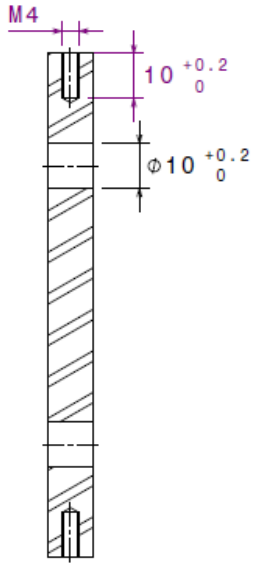

(MPa) par rapport à (mm/mm)
57846*x + 0,697
100
75
50
25
0
(MPa)
0,0004
0,0006
0,0008
0,0010
0,0012
0,0014
(mm/mm)

Plans de la fusée

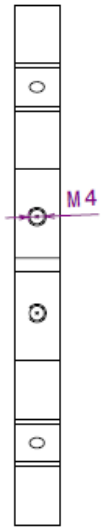




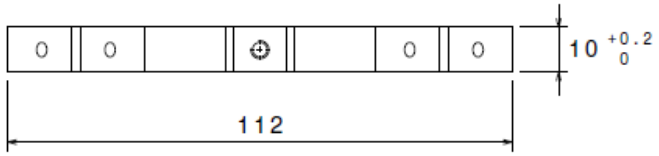
Vue de face
Echelle : 1:1



Coupe A-A
Echelle : 1:1

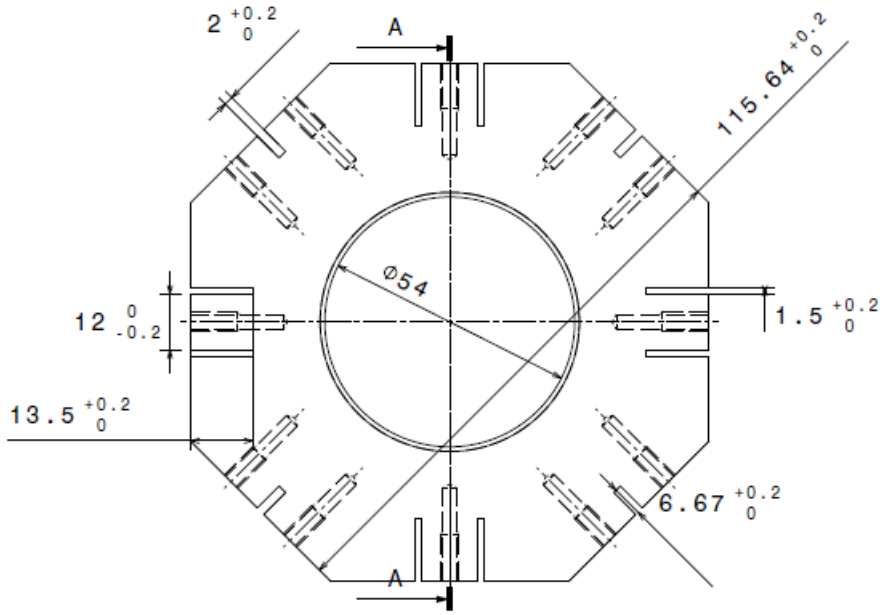


Vue auxiliaire B
Echelle : 1:1

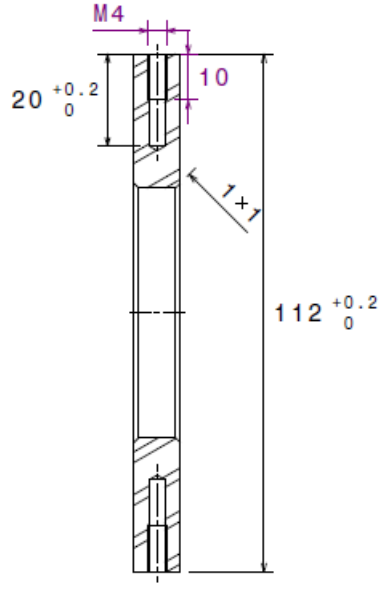


Vue de dessus
Echelle : 1:1

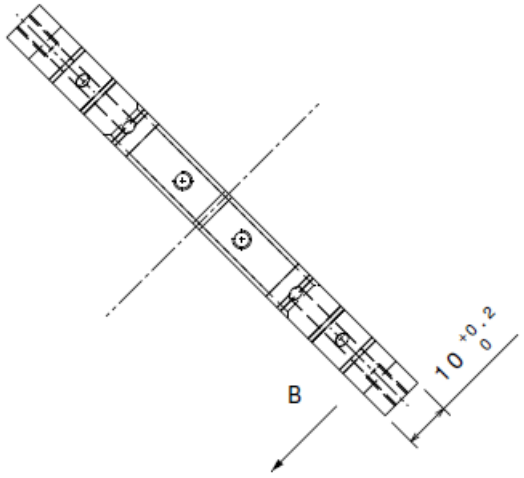
DESIGNED BY: Clement Lingois DATE: 20/01/2020		VOLOKNA	
CHECKED BY: Antoine CARTON DATE: 21/01/2020			
SIZE A3		ISAE-ENSMA	
SCALE 1:1	WEIGHT (kg) XXX	DRAWING NUMBER bague_poussee_moteur	SHEET 1/1



Vue de face
Echelle : 1:1

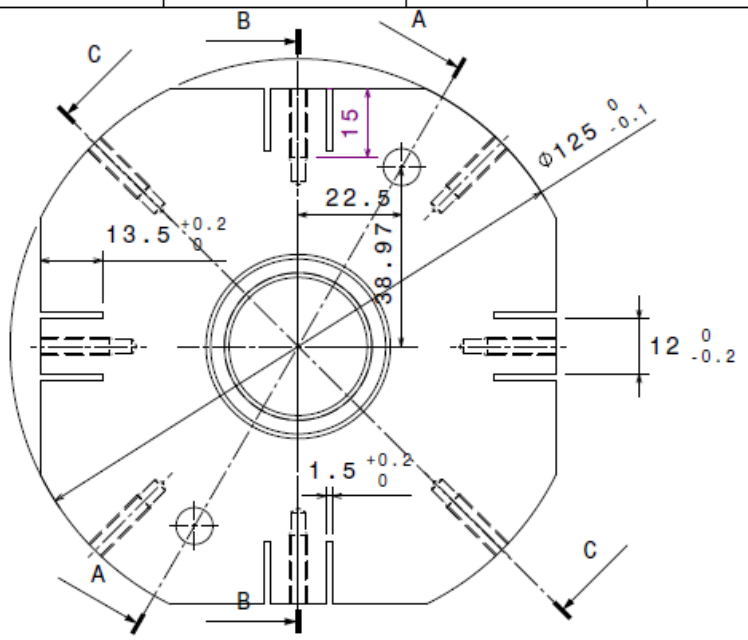


Coupe A-A
Echelle : 1:1

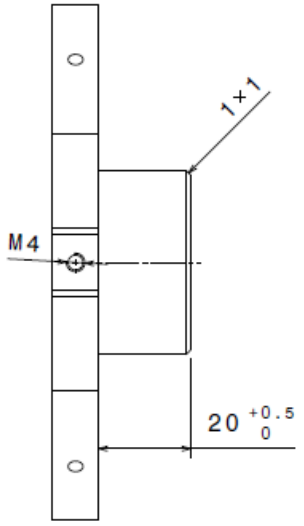


Vue isométrique
Echelle : 1:2

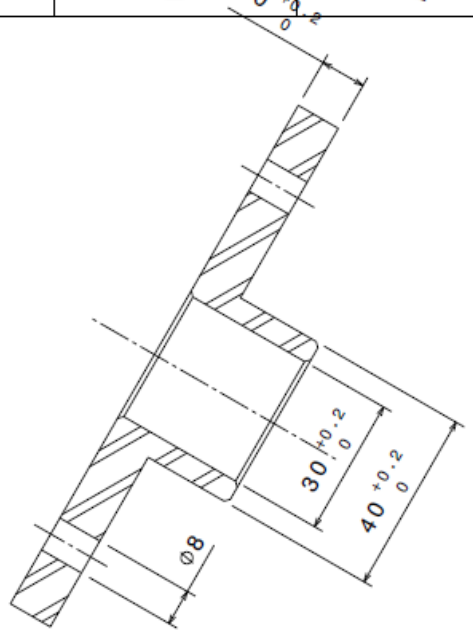
DESIGNED BY: Clement Lingois DATE: 20/01/2020	VOLOKNA		
CHECKED BY: Antoine CARTON DATE: 21/01/2020			
SIZE A3	ISAE-ENSMA		
SCALE 1:1	WEIGHT (kg) XXX	DRAWING NUMBER bague_guidage_moteur	SHEET 1/1



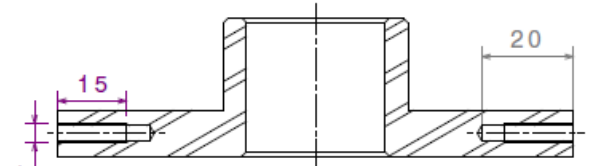
Vue de face
Echelle : 1:1



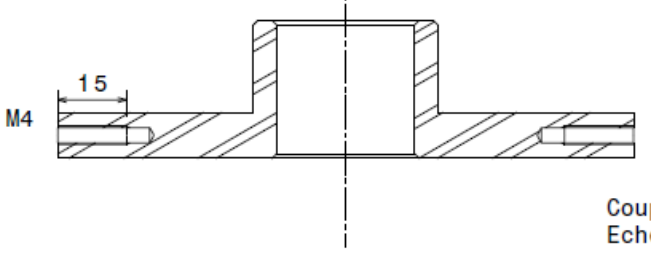
Vue de gauche
Echelle : 1:1



Coupe A-A
Echelle : 1:1

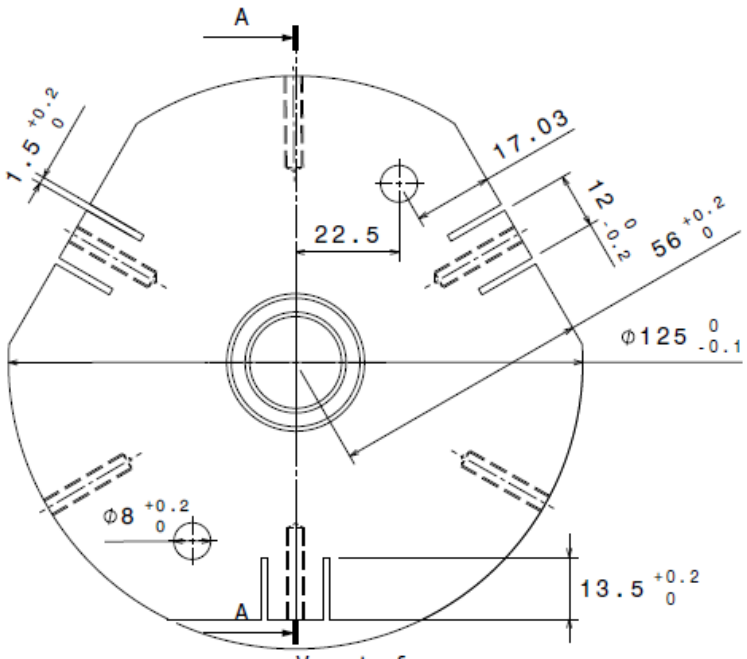


Coupe B-B
Echelle : 1:1



Coupe C-C
Echelle : 1:1

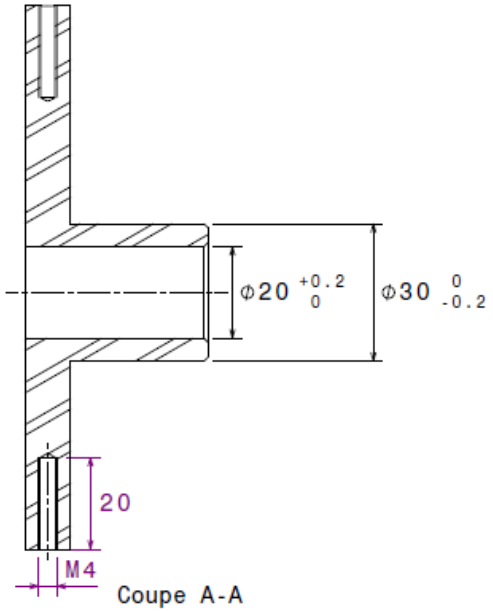
DESIGNED BY: Clement Lingois DATE: 20/01/2020		VOLOKNA	
CHECKED BY: Antoine CARTON DATE: 21/10/2020			
SIZE A3		ISAE-ENSMA	
SCALE 1:1	WEIGHT (kg) 0,31	DRAWING NUMBER Bague 3-4 U inf	SHEET 1/1



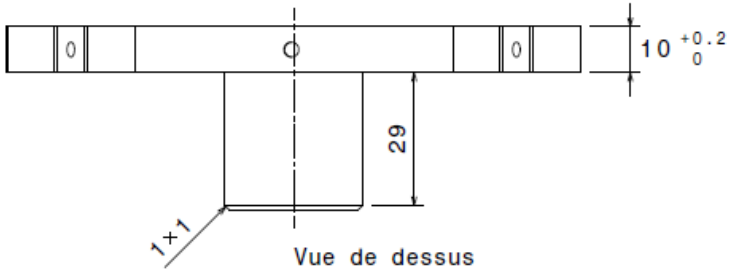
Vue de face
Echelle : 1:1

M4

Profondeur: 20

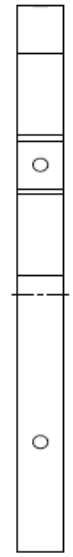


Coupe A-A
Echelle : 1:1

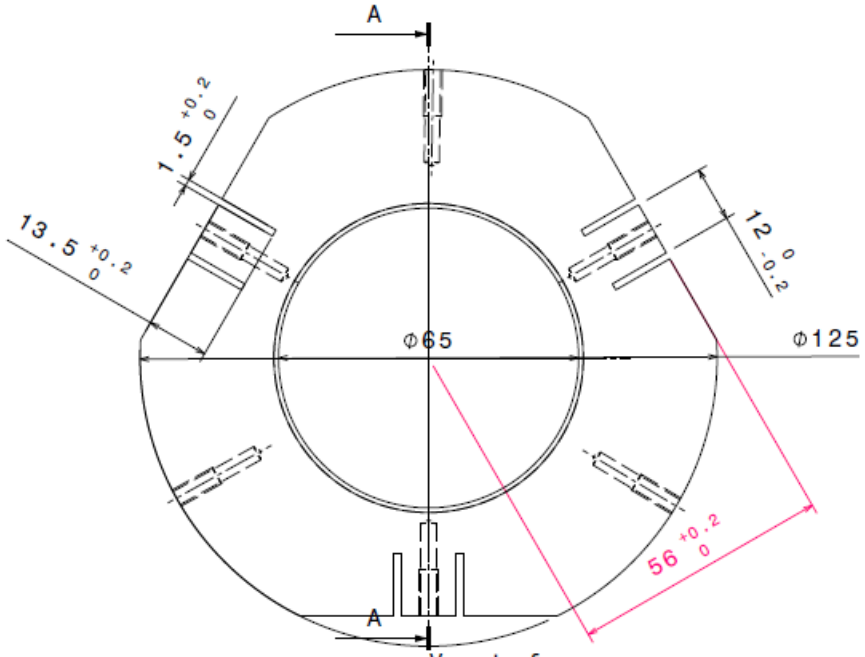


Vue de dessus
Echelle : 1:1

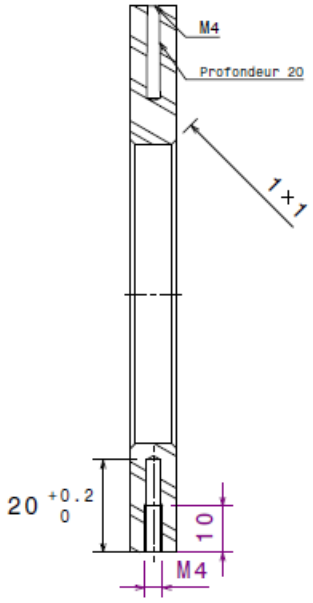
DESIGNED BY: Clement Lingois DATE: 20/01/2020	VOLOKNA		
CHECKED BY: Antoine CARTON DATE: 21/01/2020			
SIZE A3	ISAE-ENSMA		
SCALE 1:1	WEIGHT (kg) 0,33	DRAWING NUMBER Bague 3-4 U sup	SHEET 1/1



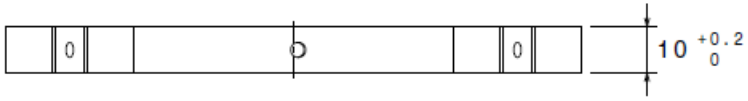
Vue de droite
Echelle : 1:1



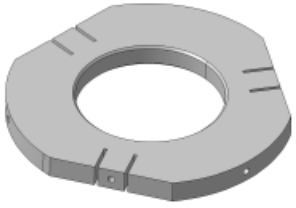
Vue de face
Echelle : 1:1



Coupe A-A
Echelle : 1:1

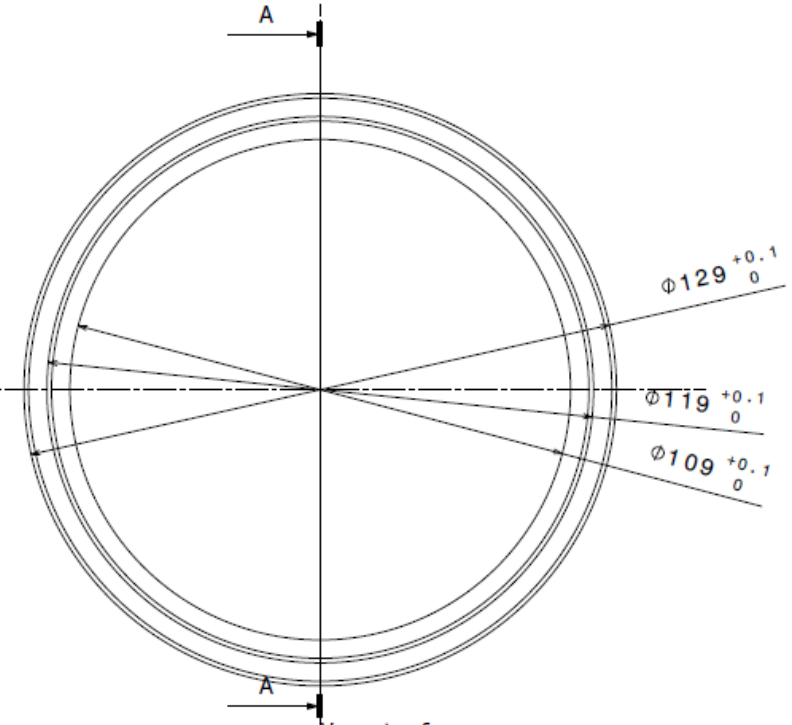


Vue de dessus
Echelle : 1:1

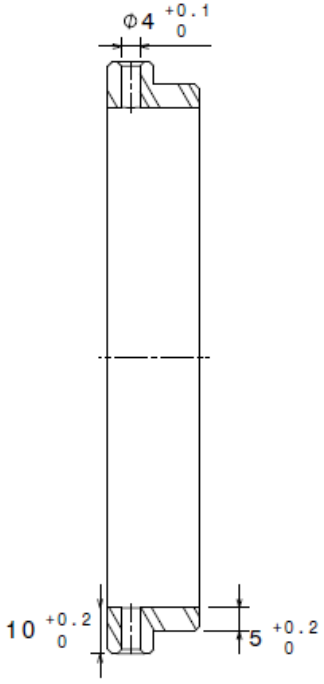


Vue isométrique
Echelle : 1:2

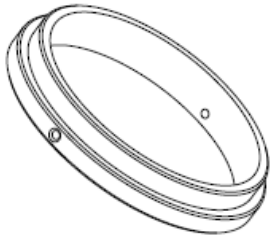
DESIGNED BY: Clement Lingois DATE: 20/01/2020	VOLOKNA	
CHECKED BY: Antoine CARTON DATE: 21/01/2020		
SIZE A3	ISAE-ENSMA	
SCALE 1:1 / WEIGHT (kg) 0,22	DRAWING NUMBER Bague_rigidite	SHEET 1/1



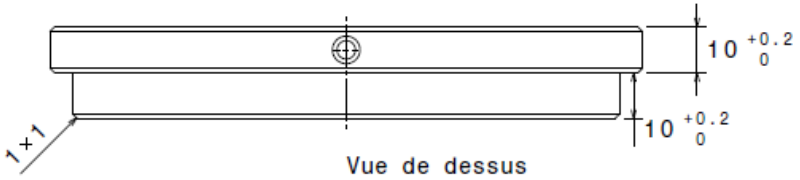
Vue de face
Echelle : 1:1



Coupe A-A
Echelle : 1:1

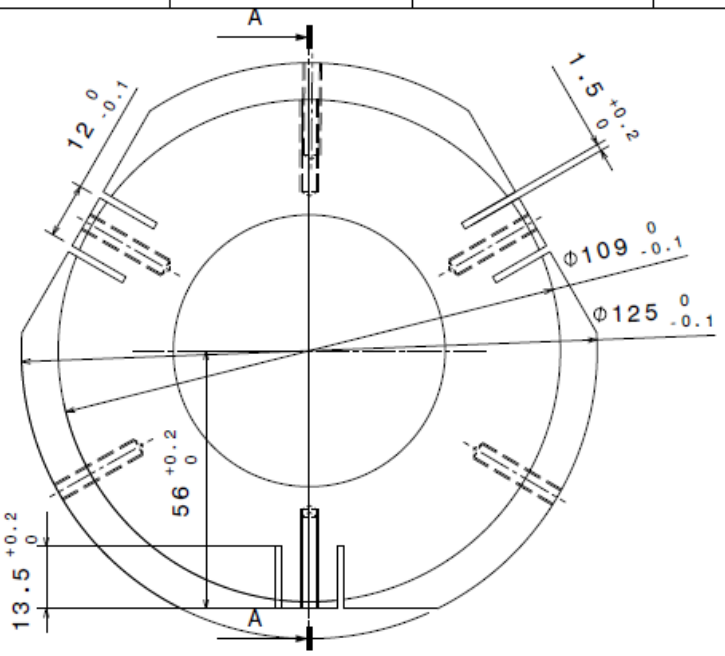


Vue isométrique
Echelle : 1:2

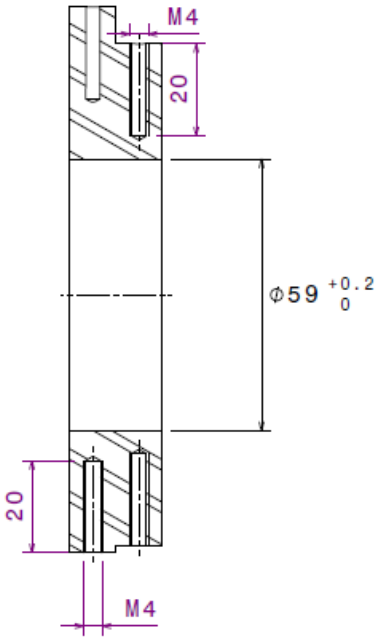


Vue de dessus
Echelle : 1:1

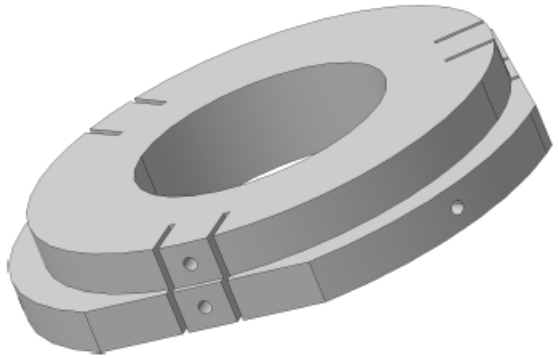
DESIGNED BY: Clément Lingois DATE: 20/01/2020	VOLOKNA		
CHECKED BY: Alexis Collet DATE: 21/01/2020			
SIZE A3	ISAE-ENSMA		
SCALE 1:1	WEIGHT (kg) 0,15	DRAWING NUMBER Bague_attache_coiffe	SHEET 1/1



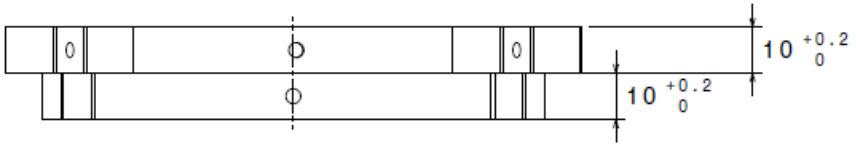
Vue de face
Echelle : 1:1



Coupe A-A
Echelle : 1:1



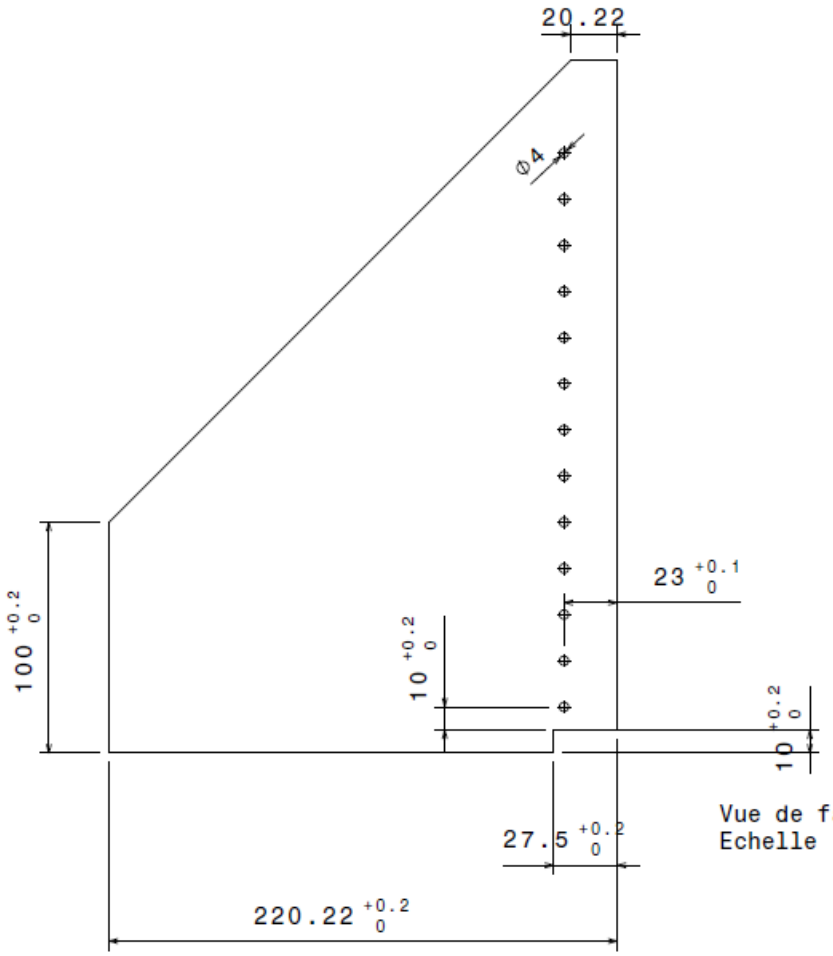
Vue isométrique
Echelle : 1:1



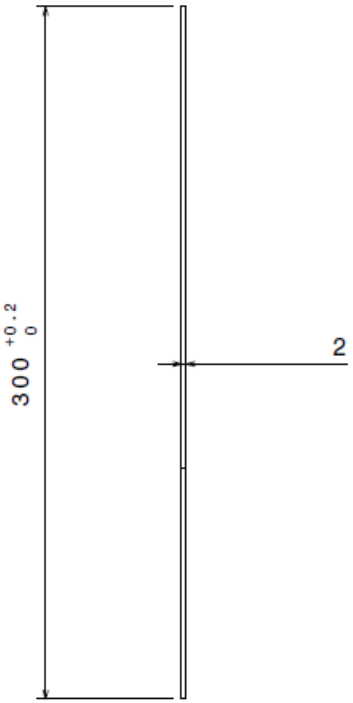
Vue de dessus
Echelle : 1:1

DESIGNED BY: Clement Lingois DATE: 20/01/2020		VOLOKNA	
CHECKED BY: Alexis Collet DATE: 21/01/2020			
SIZE A3		ISAE-ENSMA	
SCALE 1:1	WEIGHT (kg) 0,41	DRAWING NUMBER Bague_lien_corps/coiffe	SHEET 1/1

I	_
H	_
G	_
F	_
E	_
D	_
C	_
B	_
A	_



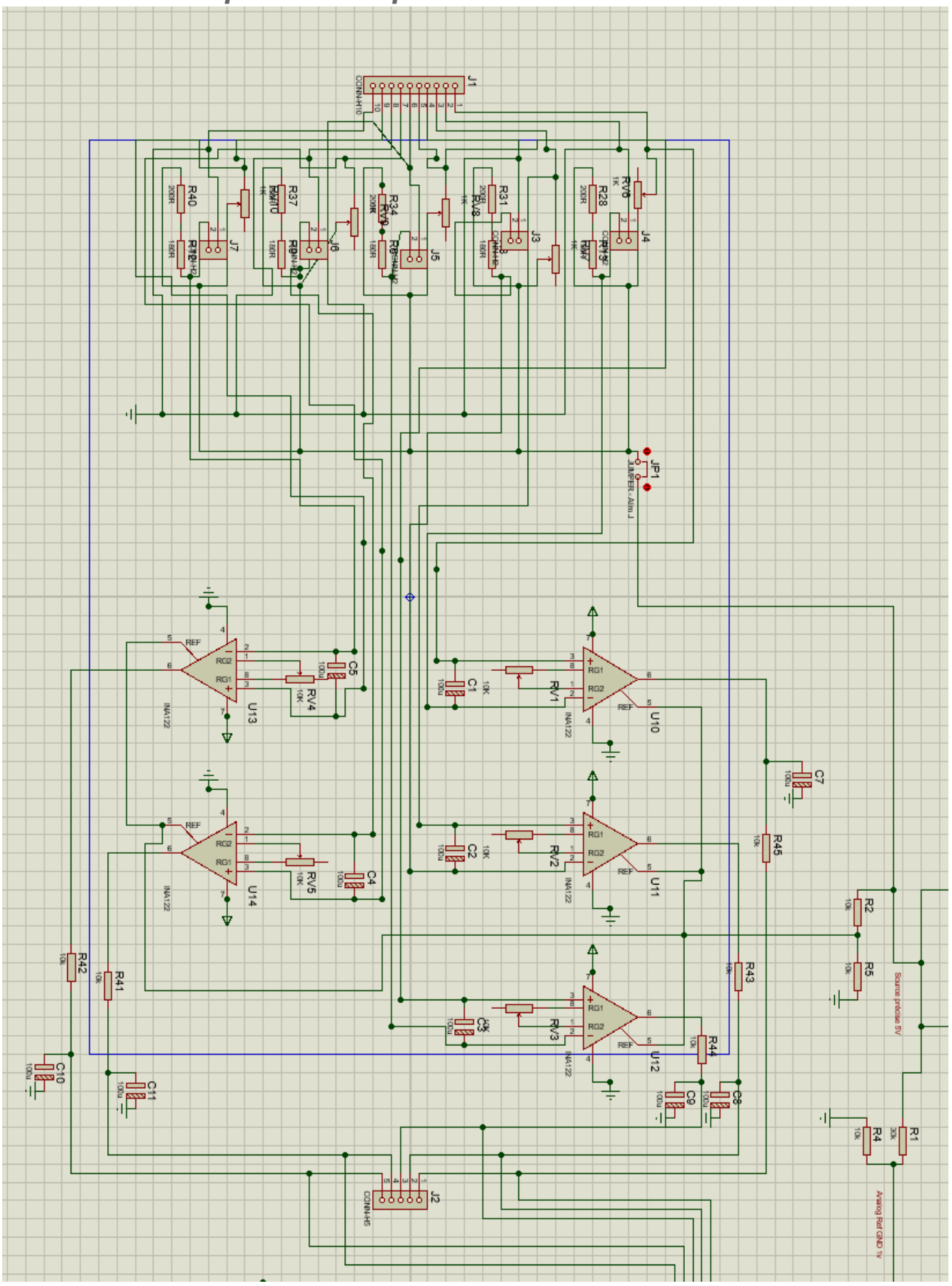
Vue de face
Echelle : 1:2

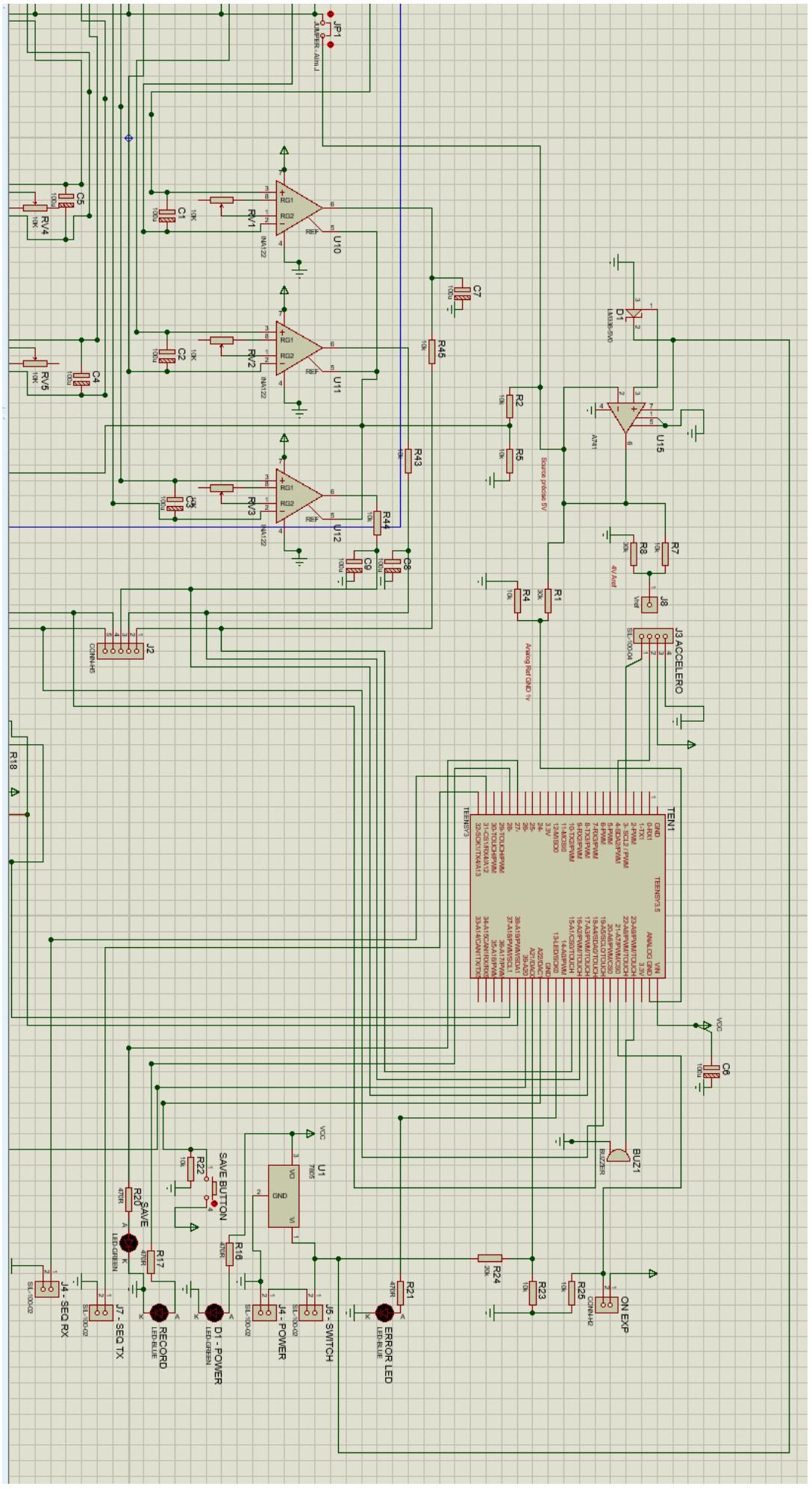


Vue de gauche
Echelle : 1:2

DESIGNED BY: Collet Alexis DATE: 21/01/2020		VOLOKNA	
CHECKED BY: Carton Antoine DATE: 22/01/2020			
SIZE A3		ISAE-ENSMA	
SCALE 1:1	WEIGHT (kg) 0,25	DRAWING NUMBER aileron x4	SHEET 1/1

Schéma électrique carte experience




JP1
JUMPER-Alim J
U10
U11
U12
U15
U1
TEN1
TEENSY3.5
J3.ACCELERO
J2
J8
BUZ1
BUZZER
SAVE BUTTON
R20 SAVE
R17
R18
R21 ERROR LED
D1 - POWER
RECORD
J7 - SEQ TX
J4 - SEQ RX
J4 - POWER
J5 - SWITCH
ON EXP
R22
R23
R24
R25
C6
Source principale 5V
4V Aref
Analog Ref GND 5V

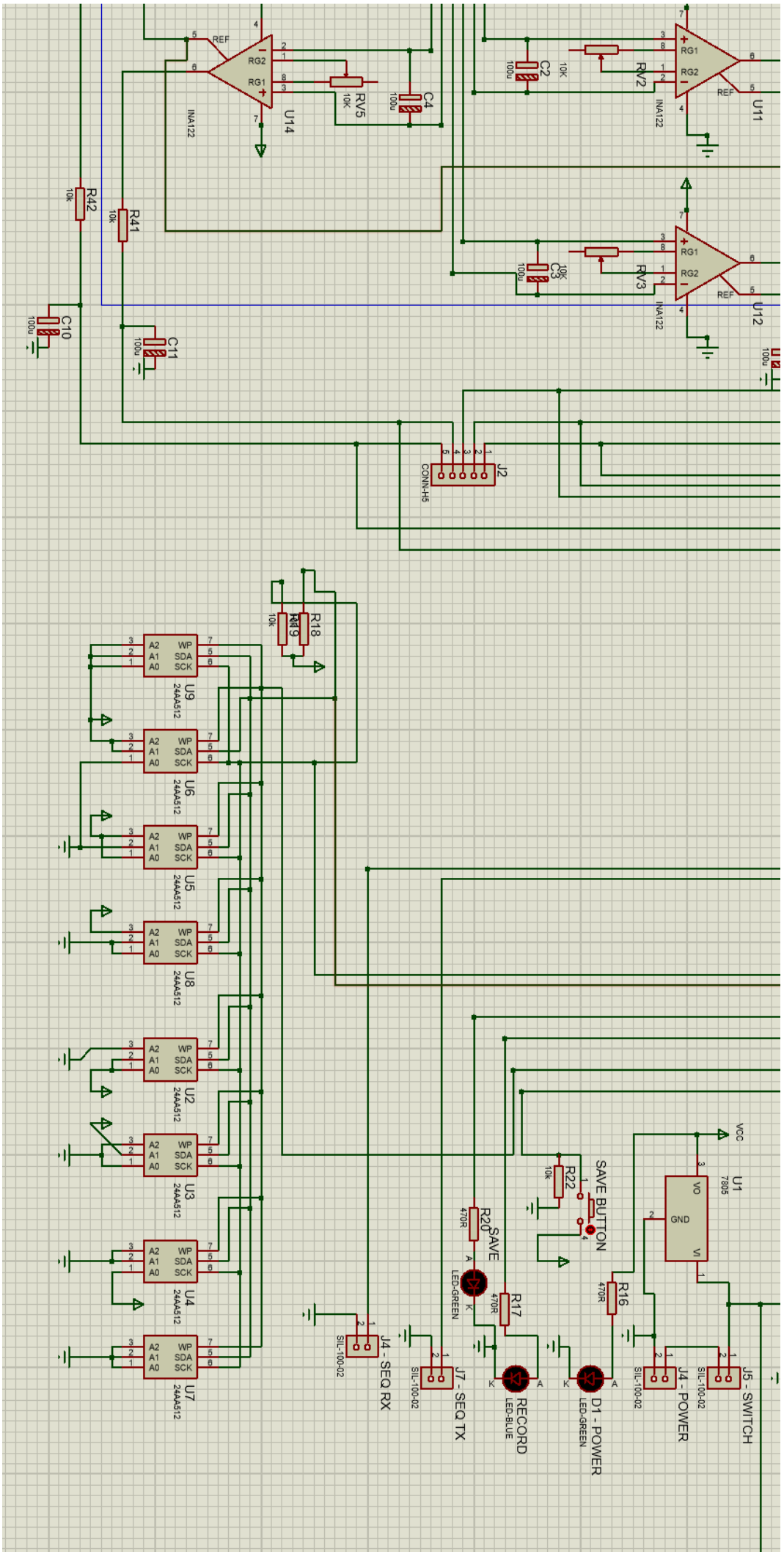

U11
U12
U14
INA122
RG1
RG2
REF
RV2
RV3
RV5
C2
C3
C4
C10
C11
R41
R42
J2
CONN-H5
U9
U6
U5
U8
U2
U3
U4
U7
24AA512
A2
A1
A0
WP
SDA
SCK
R18
R19
U1
7805
VCC
VO
GND
VI
SAVE BUTTON
R22
R20
SAVE
LED-GREEN
R17
RECORD
LED-BLUE
R16
D1 - POWER
LED-GREEN
J4 - SEQ RX
J7 - SEQ TX
J4 - POWER
J5 - SWITCH
SIL-100-02

Schéma électrique séquenceur