Rapport de vol

C'Space 2025

Nom du projet : Eos
Matricule : FX50

Club : Astrolab

Membres du projet :

Balassoupramaniane Shashankan, Mika Desgoutte, Levasseur Sophian, Chanalet Victor, Bouzou Charles, Montanier Quentin, Menoux Yvan, Savary Oriane, Samou Ophélie, Lemonsu Hugo, Tillerot Tristan

Plan du rapport

Remerciements 4

Introduction 5

Déroulement global du projet 6

Gestion de projet 7

StabTraj 8

Structure et Matériaux 10

- Généralités 10
- Coiffe 10
- Fuselage 11
- Ailerons 12
- Compartiment parachute 12
- Bague de reprise de la poussée 13
- Anneau de centrage 13
- Trappe parachute 14
- Equerres 15
- Retour d'expérience 15

Microélectronique 16

- Généralités 16
- Télémétrie 18
- Retour d'expérience 19

Capteurs 21

- Généralités 21
- Jauges de déformation 21
- Autres mesures 23
- Retour d'expérience 25

Parachute 26

- Généralités 26

Retour d’expérience 28
Séquenceur 30
Description mécanique 30
Description électronique 30
C’Space 31
Conclusion 32
Annexes 33
Code du séquenceur et de l'ouverture de la trappe 33
Acquisition des données des jauges de déformation 36
Code de l’Arduino 1 38
Code de l’Arduino 2 40
Code de l’Arduino Mega 42

Remerciements

Avant tout, nous souhaitons adresser nos remerciements les plus chaleureux à celles et ceux dont le soutien a été déterminant dans la réussite de ce projet.

- Nous remercions tout particulièrement Marylène Lallemand pour son aide précieuse et son immense soutien tout au long de la réalisation de la fusée, notamment dans la gestion de l'usinage de la pièce de reprise poussée, ainsi que pour son accompagnement dans la mise en place et l'utilisation des jauges de déformation.

- Un grand merci à Alain Kilidjian pour l'intérêt qu'il a porté à notre projet dès ses débuts, ainsi que pour ses conseils éclairés.

- Nos remerciements vont également à Michel Collin pour la fabrication de la pièce de reprise poussée.

- Nous adressons un merci tout particulier à l'association FabLab et à e-Gab, qui nous ont généreusement prêté du matériel tout au long du projet, permettant ainsi sa concrétisation.

- Enfin, nous remercions l'école Centrale Méditerranée pour sa contribution indirecte au projet.

Introduction

Notre association, Astrolab, a pour but de créer un lien entre notre formation et les domaines liés à l'air et à l'espace : l'aéronautique, l'aérospatial et l'astronomie. Pour faire vivre cette passion, nous organisons des conférences, des soirées d'observation du ciel, et menons plusieurs projets concrets comme la conception d'un drone, d'un planeur et d'une fusée expérimentale.

Le projet Eos s'inscrit pleinement dans cette dynamique. C'est un défi que nous avons relevé depuis zéro, avec pour ambition de concevoir entièrement une fusée fonctionnelle, capable d'embarquer des expériences scientifiques développées elles aussi par l'équipe. Ce projet nous a permis de mettre en pratique nos connaissances, mais aussi d'en acquérir de nouvelles. Chacun a pu progresser, apprendre dans des domaines variés, et contribuer activement à une aventure technique et humaine hors du commun.

Déroulement global du projet

Le projet a débuté par une phase de recrutement, menée entre octobre et novembre 2024. À l'issue de celle-ci, nous avons constitué une équipe de douze personnes (dont l'une nous a malheureusement quittés assez rapidement).

Ne disposant d'aucune expérience préalable en conception de fusée expérimentale, nous avons choisi de nous structurer autour de cinq pôles de travail :

- Gestion de Projet (GdP)
- Structure et Matériaux (SeM)
- Microélectronique (Me)
- Charge Utile (CU)
- Atterrissage et Récupération (AeR)

Les débuts ont été plutôt difficiles, notamment parce qu'il était compliqué de savoir par où commencer. Le projet a mis un certain temps à réellement démarrer. Il a fallu plusieurs réunions d'équipe (organisées une fois par semaine) pour commencer à dégager une vision claire du travail à accomplir, de la charge que cela représentait, et des grandes étapes à franchir.

Le projet a pris une tournure plus concrète courant décembre, avec une structuration progressive, l'élaboration des premiers budgets prévisionnels et une meilleure répartition des tâches. Une des difficultés majeures a été la recherche de financements, indispensable pour aller au bout du projet.

La phase de conception s'est poursuivie jusqu'à la RCE 2, qui s'est tenue à Paris le 8 février 2025. Cette rencontre nous a donné un coup de motivation important pour finaliser le projet. Nous avons lancé la construction dès que possible. L'intégration et la fabrication ont finalement pris beaucoup plus de temps que prévu, ce qui nous a demandé de redoubler d'efforts à l'approche de la RCE 3.
Suite à la réussite de celle-ci, nous avons maintenu le rythme de travail établi, compte tenu de l'ampleur des tâches restantes à réaliser d'ici au C'Space, auquel nous sommes allés du 5 au 12 juillet.

Gestion de projet

La gestion du projet s'est appuyée sur des outils simples mais efficaces. Nous avons d'abord utilisé Notion pour organiser les tâches, planifier les étapes avec un diagramme de Gantt, et stocker les documents utiles. Par la suite, nous avons migré l'ensemble de nos ressources vers Google Drive, plus pratique pour le partage et le travail collaboratif.

Tout au long du projet, nous avons tenu une réunion hebdomadaire, qui prenait la forme d'un tour de table structuré. Chacun y présentait l'avancement de son pôle, les difficultés rencontrées, les décisions à prendre et les besoins éventuels en coordination avec les autres groupes. Cela nous a permis de garder une bonne visibilité sur l'évolution du projet, de prendre des décisions collectives rapidement, et de maintenir une dynamique de groupe constante.

Coordonner une équipe de 11 personnes tout en poursuivant nos études a représenté un vrai défi organisationnel. Cela a demandé de la rigueur, un bon sens de la coordination et une implication régulière de chacun. Malgré les emplois du temps chargés, nous avons réussi à maintenir une bonne dynamique, ce qui a grandement contribué à la progression du projet.

StabTraj

STABILITO

Stabilité de fusée à ailerons

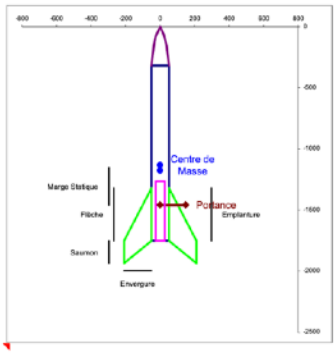
Remplir les cases jaunes

Fusée	
Nom	eos
Club	Astrolab
Matricule	FX_50
Type	Fusée expérimentale.
Masse	6740 g / sans propu
Centre de Masse	1093 mm / sans propu
Longueur totale	1750 mm

Propulseur	
Type	Pro54-5G WT
Position du bas	1750 mm

Coiffe	
Forme	Ogivale (pointue)
Hauteur	320 mm
Diamètre	100 mm

Ailerons	Mono-empennage
Emplanture 'm'	430 mm
Saumon 'n'	186 mm
Flèche 'p'	430 mm
Envergure 'E'	160 mm
Epaisseur 'ep'	2 mm
Nombre	4
Position du bas	1750 mm

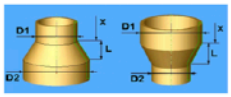


Language/Langue : Français

Fusée mono-diamètre,

	Propu plein	Propu vide	Sans propu
Masse propu	1,632 kg	0,65 kg	-
CdM propu	250 mm	240 mm	-
Masse fusée	8,372 kg	7,39 kg	6,74 kg
CdM fusée	1175 mm	1129 mm	1093 mm

	XCp	Cna
Coiffe	149 mm	2,0
Ailerons	1588 mm	20,2

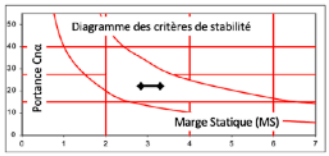


Propulseur	
Haut	1262 mm
Longueur	488 mm
Bas	1750 mm

Ailerons bas	
Haut	1320 mm
Emplanture	430 mm
Bas	1750 mm

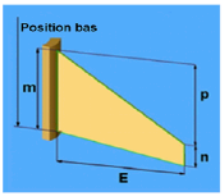
20/10/2025	Min	Résultats		Max
Finesse	10		17,5	35
Portance	15	22,2	22,2	40
MargeStat.	2 D	2,84 D	3,29 D	6 D
Couple	40	63,0	73,2	100
XCp		1458 mm	1458 mm	
MS /L		16% L	19% L	

STABLE



Commentaire libre :

Checksum : propu NOK v3.4.5



Maintenant que votre fusée est stable, vérifiez sa trajectoire via la feuille Trajecto

TRAJECTO

Trajectographie de fusée

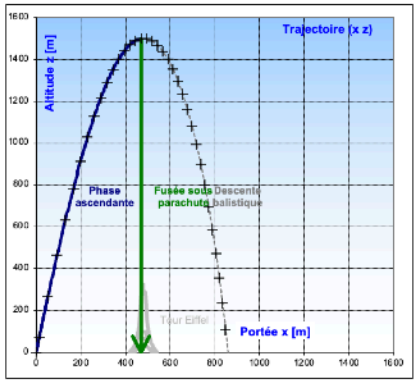
Remplir les cases jaunes

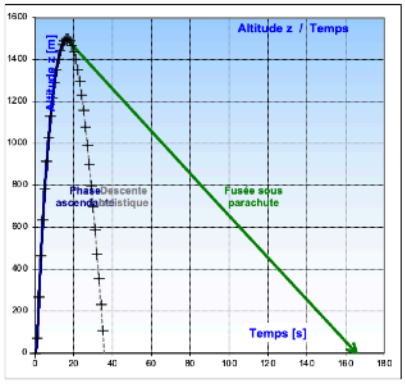
Fusée	
Nom	eos
Club	Astrolab
Matricule	FX50
Masse totale	8,372 kg
Propulseur	Pro54-5G WT

Traînée Aérdynamique	
Surface Réf.	0,009134 m²
Cx	0,6

Rampe de Lancement	
Longueur	4 m
Élévation	80 °
Altitude	0 m

DescenteSousParachute	Fusée	0 satellite
Masse	7,39 kg	
Dépotage	N/A	
Ouverture para	16 s	
Type de para	Croix	
Surface para	1,18 m²	
Cx parachute	1	
Vitesse du vent	5 m/s	
Vitesse descente	10,0 m/s	
Durée descente	150 s	
Durée du vol	166 s	
Déport latéral	±749 m	





20/10/2025	Temps	Altitude z	Portée x	Vitesse	Accélération	Efforts
Sortie de Rampe				33,3 m/s		
Vit max & Acc max				223 m/s	153 m/s²	
Culmination, Apogée	16,4 s	1500 m	480 m	24 m/s		
Ouverture parachute fusée	16,0 s	1499 m	470 m	25 m/s		447,7 N
Impact balistique	35,9 s	~0 m	859 m	128,7 m/s		61196 J

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

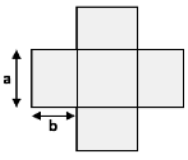
propu NOK

Commentaire libre :

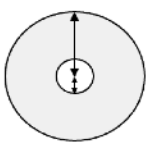
v3.4.5

Calcul de la surface d'un parachute

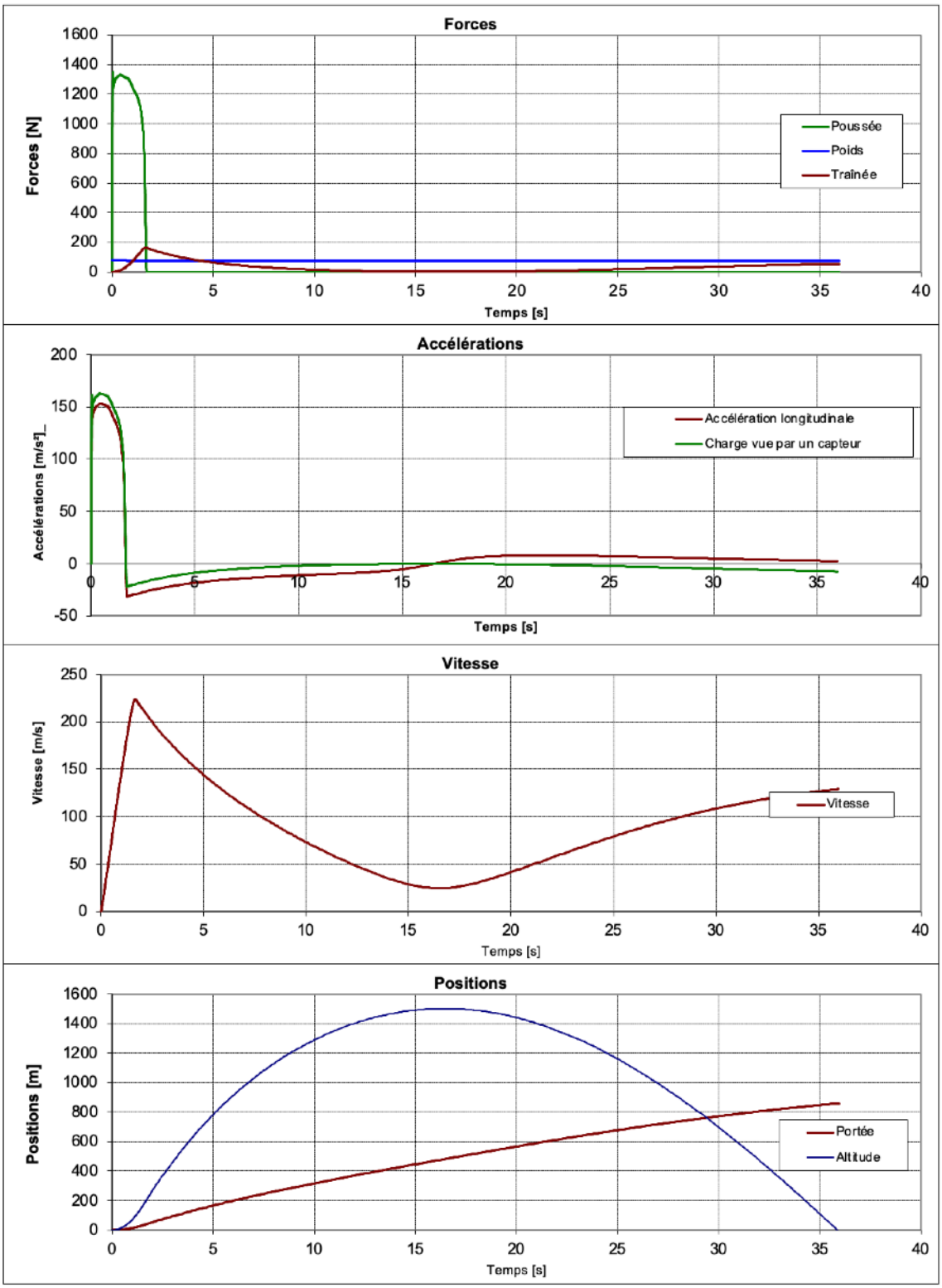
Bord 'a'
1100 mm
Coté 'b'
0 mm
Surface para
1,21 m²



Rayon exterieur
700 mm
Rayon intérieur
100 mm
Surface para
1,51 m²



Résultats détaillés	Temps	Altitude z	Portée x	Vitesse	Accélération	Angle
	s	m	m	m/s	m/s²	°
Décollage	0	0	0	0	-	80
Sortie de Rampe	0,24	3,72	0,66	33,3	149,8	80,0
Vit max & Acc max	-	-	-	223	153,0	-
Fin de Propulsion	1,7	206	40	223	31,8	78,6
Culmination, Apogée	16,4	1500	480	24	9,8	1,0
Impact balistique	35,9	~0	859	129	2,5	-84,4
Ouverture parachute fusée	16,0	1499	470	25	9,9	10,1
Impact fusée sous para.	166	~0	-278 | 1219	10	9,8	-

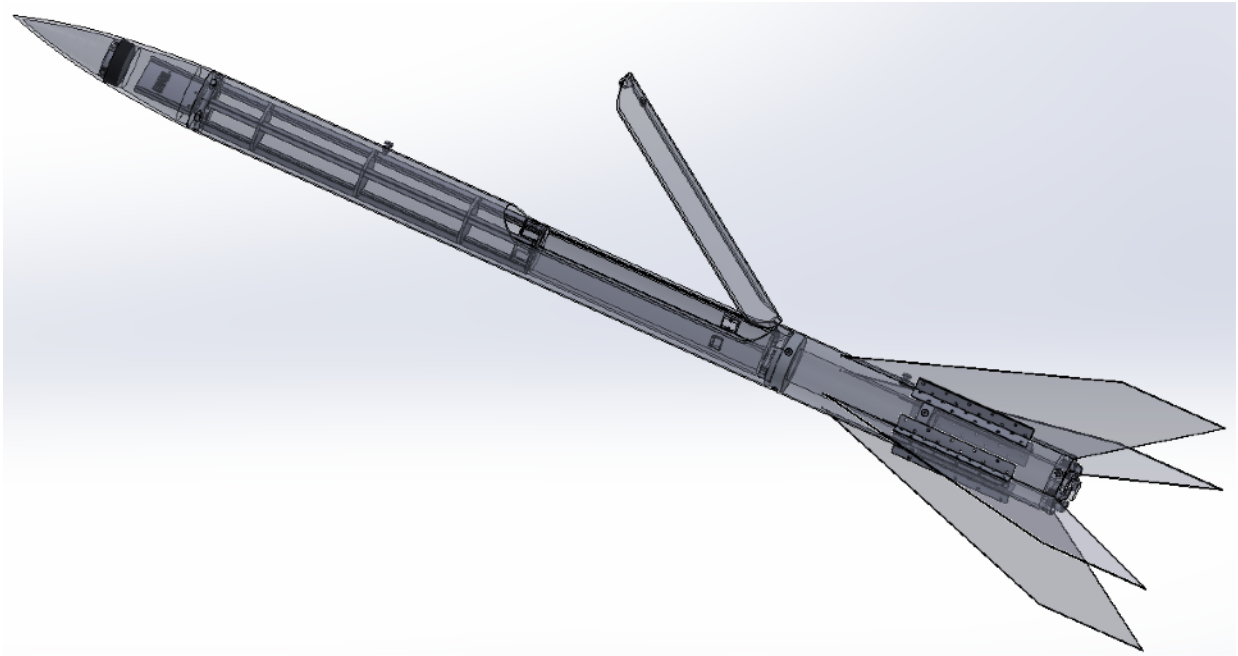


Tout d'abord, afin de réaliser au mieux ce projet nous avons longuement étudier la documentation qui était à notre disposition, en l'occurrence le cahier des charges et notamment les parties structurelles de la fusée ainsi que celle du moteur. Une fois cela fait, nous avons établir une première version du StabTraj menant aux premières modélisations. Puis, lorsque les contraintes imposées par le cahier des charges, de moyens techniques et financier de chaque pôle ont été pris en considération, une version définitive du StabTraj a été validé par l'ensemble du groupe.

Structure et Matériaux

Généralités

Cette version finale du StabTraj a permis la modélisation sur SolidWorks de chacun des éléments au sens mécanique du terme. Voici une vue en transparence de l'assemblage des pièces composant notre Fusée expérimentale, nous détaillerons par la suite les principaux éléments qui la composent.

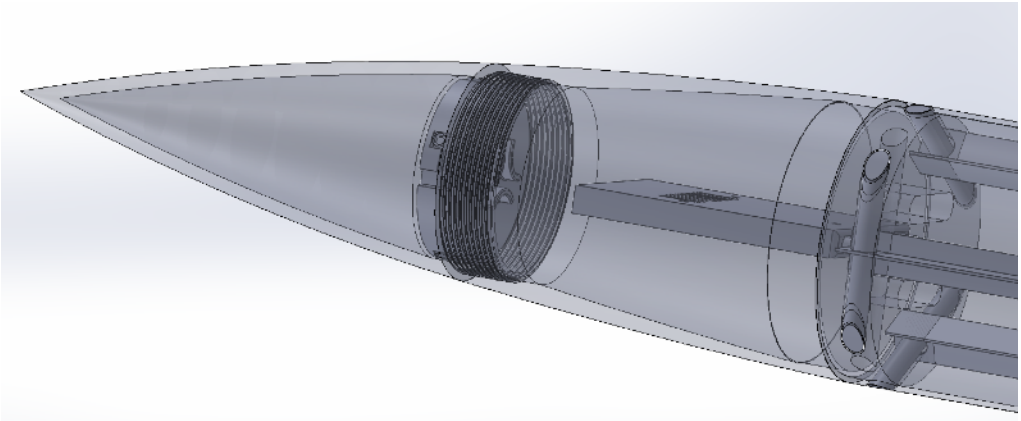


Assemblage de la modélisation 3D de notre FusEx Eos

On y retrouve de haut en bas et de gauche à droite, la coiffe, la peau (autrement dit le fuselage), les ailerons, le moteur (PRO 54-2G WHITE THUNDER), les rails qui s'insèrent dans la peau et qui maintiennent toute la partie électronique/système embarquée ainsi que le compartiment parachute, puis viennent la bague de reprise poussée et un des deux anneaux de centrage du moteur. Pour finir, tout en bas nous avons la trappe refermant l'ouverture sur le compartiment parachute et permettant l'éjection de ce dernier.

Coiffe

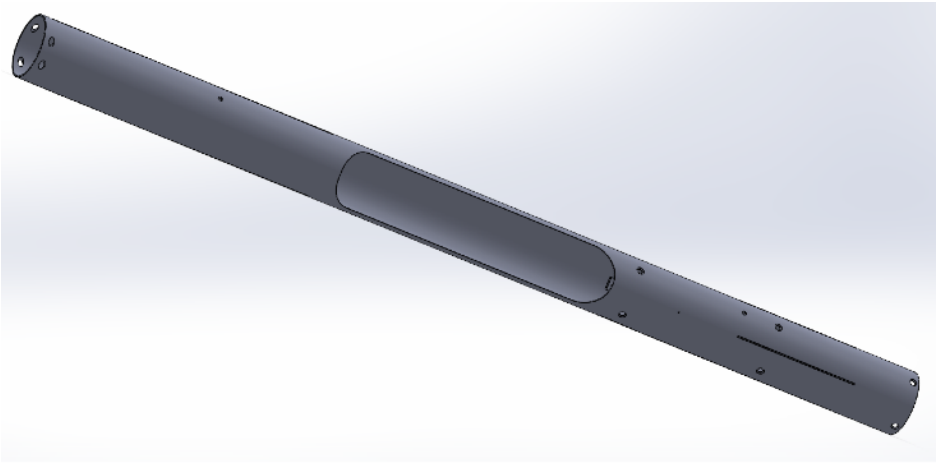
Notre coiffe est composée de trois parties. En ce qui concerne l'ogive elle-même, elle est conçue en une partie inférieure et une supérieure. Le tout est lié par une liaison hélicoïdale, c'est-à-dire, par un pas de vis d'un côté et une extrusion de matière avec les mêmes caractéristiques de l'autre, permettant un emboîtement parfait.



Vue en transparence de l'assemblage de la coiffe

Dans la partie supérieure de la coiffe, on retrouve également un plateau avec des rainures en « U », garantissant une liaison mécanique avec un seul degré de liberté. Le plateau permet de lester la partie la plus haute de la fusée dans le cas où un rehaussement du centre de masse serait nécessaire. La masse du potentiel leste ainsi que celle du plateau lui-même, couplée à l'accélération de la fusée, empêchent tout mouvement de translation sur ce degré de liberté. Quant à elle, la partie basse de la coiffe accueille un support destiné à une carte Arduino, le fuselage est lié au bas de la coiffe par trois petites sections de tiges métalliques assurant une liaison sans aucun degré de liberté.

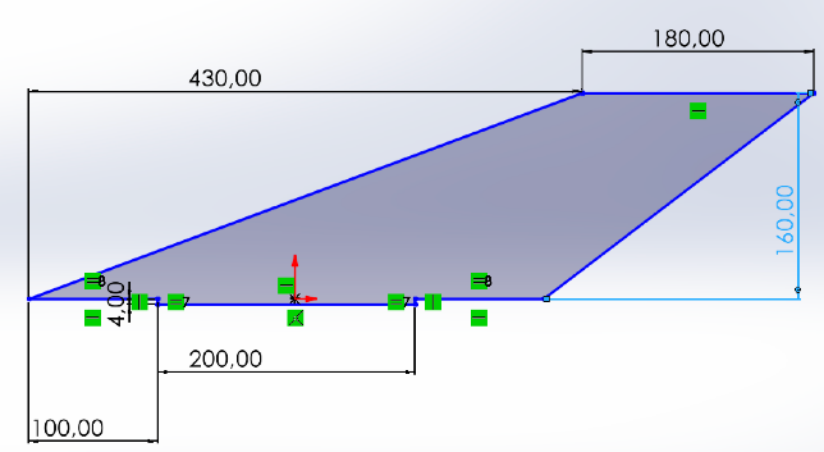
Fuselage



Modélisation 3D du fuselage

Celui-ci comporte plusieurs découpes ainsi que de nombreux perçages, cela dans le but d'assurer des liaisons mécaniques et la transmission de données. Par exemple, l'extraction du parachute au niveau de la découpe qui lui est dédiée ou encore les fentes accueillant les ailerons.

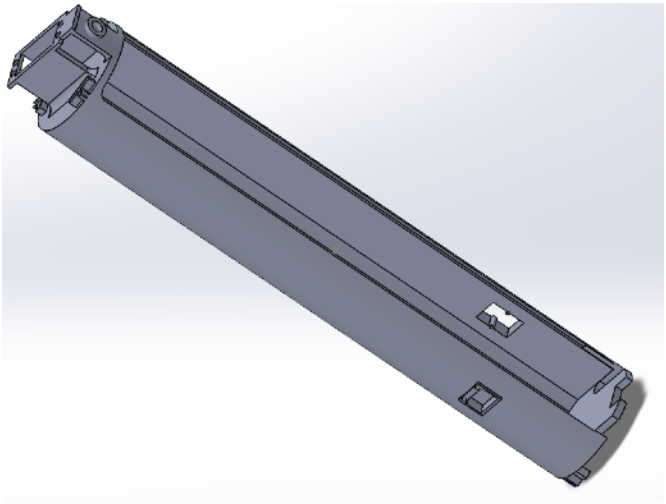
Ailerons



Esquisse SolidWorks des ailerons

Leurs dimensions ont été établies à partir du StabTraj, ce qui nous a permis de les modéliser sans difficultés sans avoir étudié de manière théorique leur profil aérodynamique ainsi que les contraintes qui vont s'exercer dessus étant donné que cela fait partie des expériences portées par le projet. Les ailerons sont fixés de manière permanente au fuselage par des équerres présentes de part et d'autre de chacun d'entre eux.

Compartiment parachute

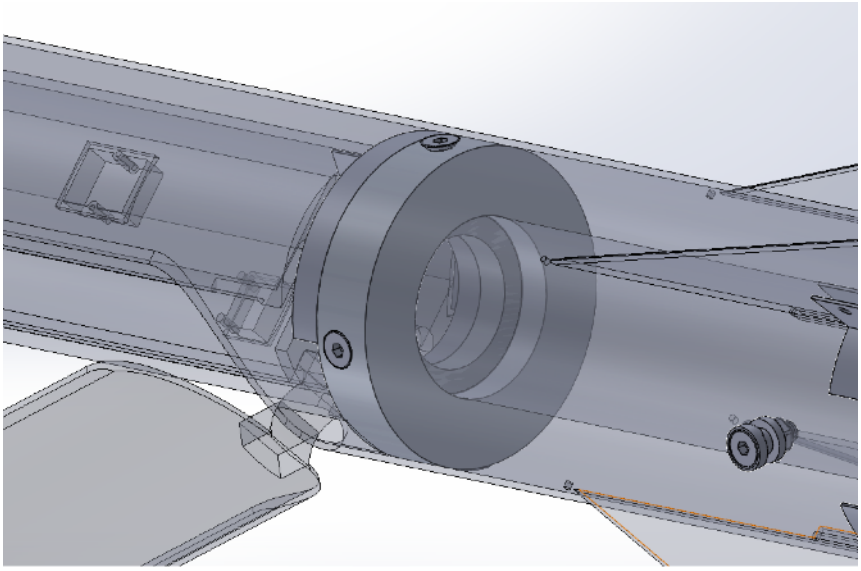


Modélisation SolidWorks du compartiment parachute

Comme son nom l'indique, le compartiment parachute accueille le parachute permettant l'atterrissage de notre fusée expérimentale. Ce compartiment remplit également d'autres fonctions, premièrement le rôle de support au servomoteur et au ressort (en haut à gauche de la capture d'écran) exécutant l'ouverture de la trappe. Dans un second temps, de nouveau de support mais cette fois-ci à deux capteurs (au milieu de part et d'autre de l'ouverture), qui sont respectivement des capteurs de température/ humidité et d'altimétrie.

Bague de reprise de la poussée

Cette pièce est cruciale autant dans sa conception sur le plan géométrique que dans le dimensionnement mécanique aux vues des contraintes qui lui seront imposées. Les dimensions de celle-ci ont été élaborées en fonction des choix techniques que nous avons faits concernant le diamètre du tube du fuselage ainsi que sur la base de la section moteur du cahier des charges. Nous avons choisi une reprise de la poussée par le haut, et d'appliquer une tolérance de 1 mm sur le diamètre intérieur afin d'assurer la bonne insertion et répartition des contraintes. Pour finir, la bague de reprise de la poussée est percée de 4 pas-de-vis distants de 90°, permettant une fixation au fuselage par 4 vis M6.

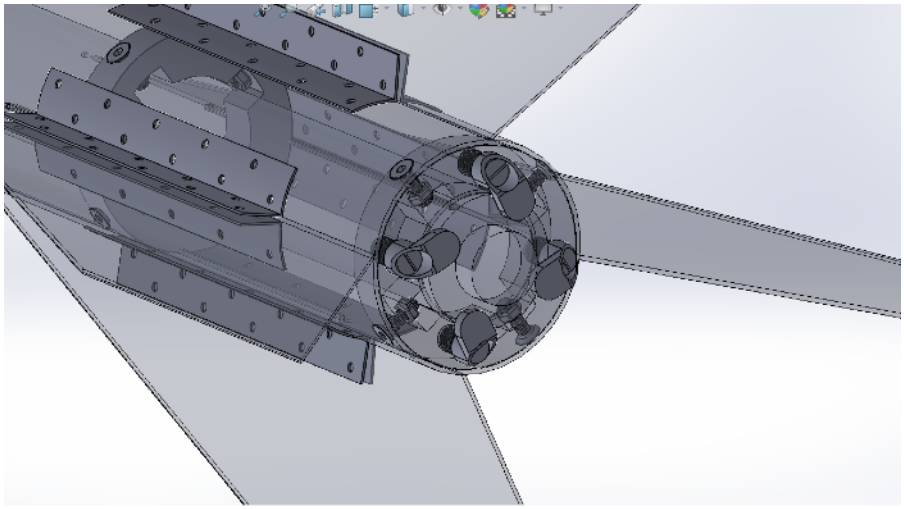


Assemblage SolidWorks de la bague de reprise de la poussée

Anneau de centrage

On en dénombre deux dans l'assemblage final de la fusée, cette pièce a peu près les mêmes dimensions que la bague de reprise de la poussée à la différence que celle-ci ne sera soumise à aucune contrainte mécanique et sert uniquement au maintien du moteur dans le bon axe. L'une se trouve au milieu du compartiment moteur, quant à la

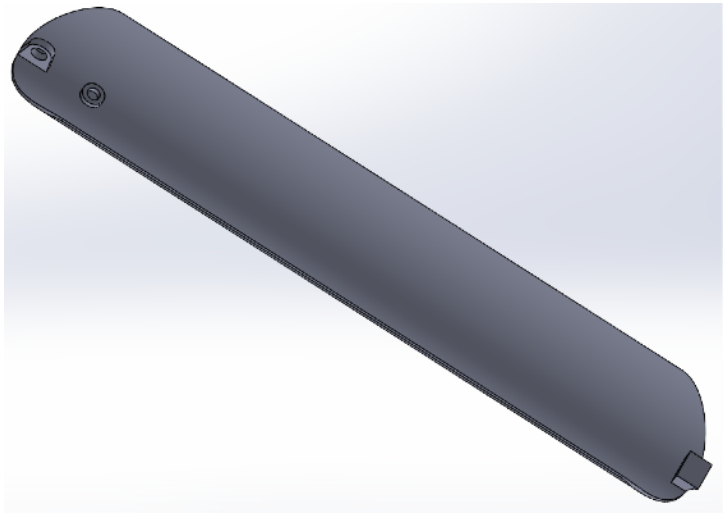
seconde elle est placée en bas et présente des taquets assurant un maintien par le bas du moteur dans la section qui lui est dédiée.



Assemblage SolidWorks de l'anneau de centrage du bas

Trappe parachute

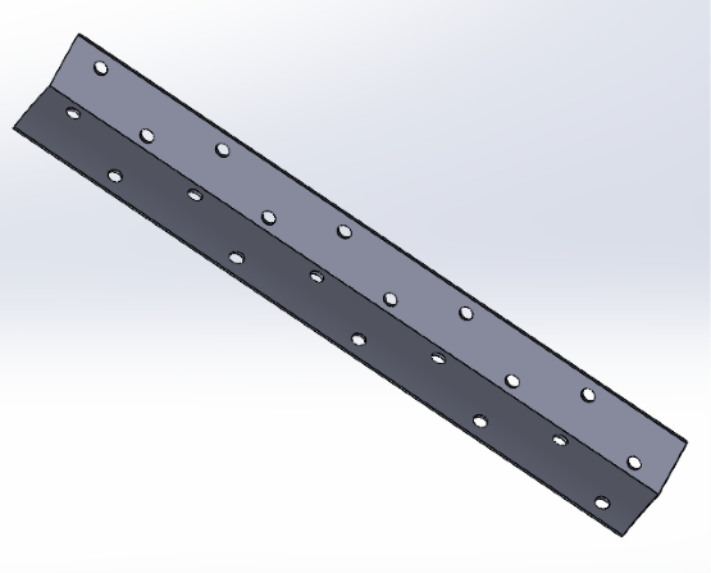
La trappe parachute a pour mission d'assurer l'aérodynamisme du fuselage au niveau de la découpe du compartiment parachute avec le crochet du servomoteur grâce à l'anneau présent à son sommet. Elle permet également une meilleure extraction du parachute en étant reliée à celui-ci et en étant mise sous tension par un ressort qui s'insère dans le renfoncement circulaire présent sous l'anneau. Enfin, la petite protubérance en bas de la trappe permet à cette dernière de reposer de manière stable sur le fuselage grâce à une cale à la géométrie complémentaire et de basculer avec fluidité à l'ouverture.



Modélisation SolidWorks de la trappe parachute

Equerres

L'équerre est un élément structurel qui peut paraître anecdotique, mais qui permet en réalité de garantir une liaison robuste entre notre fuselage (peau porteuse) et nos ailerons. Pour assurer la fixation, nous avons choisi de riveter les équerres aux niveaux des perçages sur la face tangente avec le fuselage et de les visser aux ailerons, ce qui facilite par la suite le transport de la fusée.



CAO de l'équerre utilisée lors de la fabrication

Retour d'expérience

Le pôle Structure & Matériaux a dû faire face à de nombreuses contraintes, notamment celles imposées par les autres pôles, venant s'ajouter aux exigences technico-économiques du cahier des charges et aux réalités de la fabrication de la fusée. Cela a conduit à de multiples itérations et ajustements tout au long du projet, jusqu'à la dernière semaine avant le C'Space.
Une fois le StabTraj et les principaux éléments structurels validés (ailerons, bague de reprise, fuselage), la phase de fabrication a pu débuter. C'est alors que le manque d'outillage et de temps s'est fait sentir. Malgré ces difficultés, nous avons pu présenter une fusée aboutie, tant sur le plan structurel que mécanique.
Il est important de souligner que la réussite de ce projet repose avant tout sur la synergie et l'engagement du pôle. Nos échanges hebdomadaires, particulièrement constructifs, ont permis de confronter nos approches, de prendre en compte les avis de chacun (y compris hors du pôle) et de retenir les solutions techniques les plus pertinentes pour un projet de cette envergure.

Microélectronique

Généralités

Pour la microélectronique, comme nous étions débutants en électronique, nous avons choisi d'utiliser des Arduino, qui répondaient facilement à tous nos besoins, ainsi que des breadboard soudables pour le câblage. Aucun PCB n'a été utilisé. Les breadboard étaient des shields, pouvant se brancher directement sur les pins de l'Arduino. Nous avons conçu l'assemblage en deux parties distinctes :

- **La partie séquenceur** contient le séquenceur et le servomoteur responsables de l'ouverture de la trappe parachute (plus d'informations dans la partie dédiée au séquenceur).

- **La partie expérience** regroupe l'expérience et le panneau de contrôle

Ces deux ensembles sont séparés électroniquement : ils ne partagent pas la même masse et aucune mesure de l'un n'est réalisée sur l'autre. La partie séquenceur est alimentée par deux piles 9V, l'une dédiée au séquenceur et l'autre au servomoteur. Le circuit du séquenceur comporte deux prises jack femelles servant de capteurs : si un câble est arraché, l'Arduino déclenche un compte à rebours basé sur les données du StabTraj. À la fin du compte à rebours, le servomoteur libère son crochet et ouvre la trappe. Un ressort permet alors l'éjection complète du parachute. Un schéma du circuit est présenté ci-dessous :

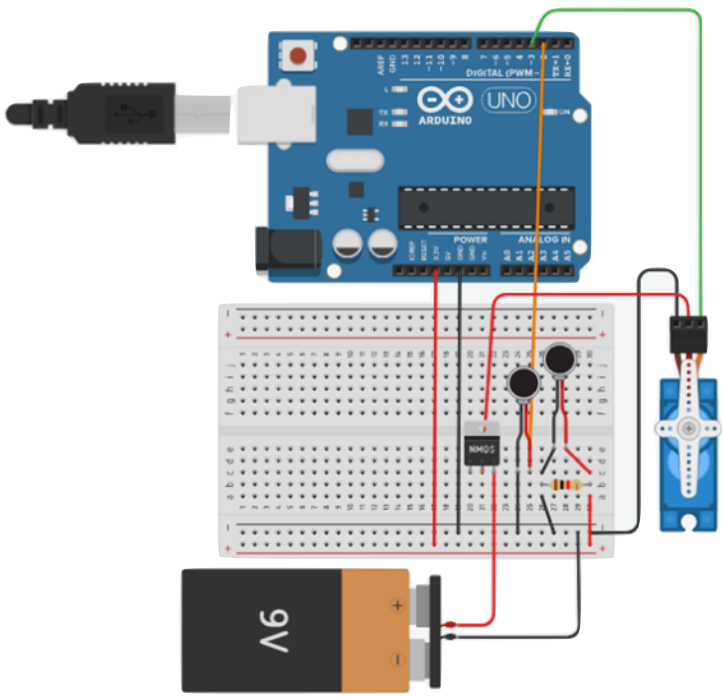
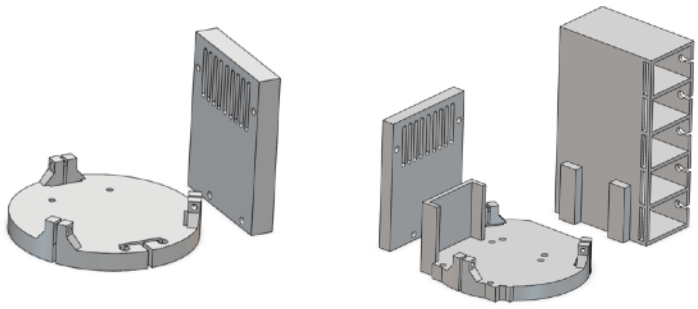


Schéma du séquenceur

Avant de réaliser les branchements définitifs, le circuit a été testé plusieurs fois sur breadboard avec des fils, branchés alors à un Arduino Uno Rev 3, ayant alors un branchement fonctionnel vers fin février. Les programmes du séquenceur sont présents en annexe. Pour la seconde partie, nous avons rencontré plusieurs difficultés. Nous étions fixés à utiliser une même alimentation (soit 3 piles de 9V en parallèle). Les Arduino étaient alimentés par leur port Jack, avec une succession de fils mettant les Arduino en série. En première position, nous avons un Arduino MEGA permettant la centralisation et le stockage des données sur une carte SD pour l'expérience et la récolte des données du gyroscope (pour les détails de l'expérience, cf. la partie charge utile). Le panneau de contrôle sert à afficher, à l'aide de LEDs, l'état des différentes arduinos à travers le circuit (à des fins de simplification, ces LEDs ne seront pas représentés sur les différents schémas électriques). De plus, le panneau de contrôle comporte deux interrupteurs permettant la liaison entre les batteries et les deux parties (un interrupteur par partie).



Les branchements sont tous effectués par des fils avec des prises mâles femelles permettant la séparation des étages. Tous les éléments des étages sont imprimés en PETG, composés d'une partie BASE et d'une partie SUPPORT : la partie basse permet de se reposer sur des guides, reposés sur l'étage précédent, ainsi que de glisser l'étage à travers des profilés en T en aluminium. La partie support permet de porter l'Arduino verticalement, n'ayant pas suffisamment de place horizontalement. Toutes les fixations sont effectuées par des inserts thermofixés et des vis M2, permettant d'intégrer un pas de vis solide et fiable.

Télémétrie

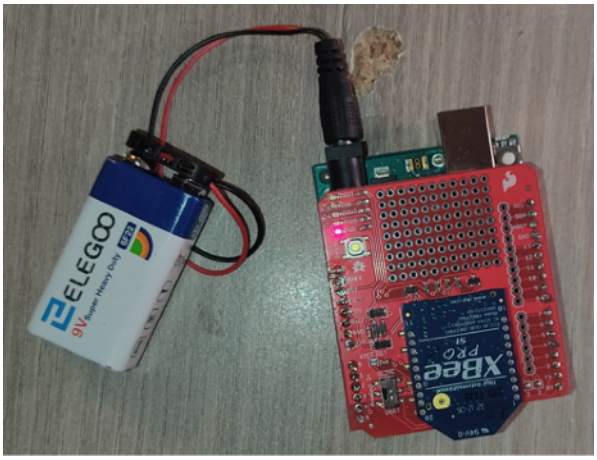
La télémétrie n'a malheureusement pas pu aboutir pour le projet Fusex de cette année. Néanmoins, nous vous présentons l'état d'avancement de cette partie et les difficultés rencontrées.

Première idée : Xbee S1 "classique"

Nous sommes parvenus à faire communiquer un module émetteur avec un module récepteur, et à envoyer des messages. L'inconvénient est que la portée de communication restait faible (~20 m).

Deuxième idée : Xbee S1 Pro

Afin d'augmenter la distance de télécommunication, nous avons opté pour le même modèle que celui cité précédemment, mais sous la version Pro pour pouvoir utiliser des fréquences plus basses et plus puissantes. Cette fois-ci, les données ont pu être captées sur une petite centaine de mètres. Cependant, le signal était instable dès que des obstacles (ex : arbres, grillages…) étaient rencontrés.
Par ailleurs, nous avions des doutes en matière de légalité sur les plages de fréquences utilisées, du fait que les normes des composants non européens diffèrent de celles françaises.

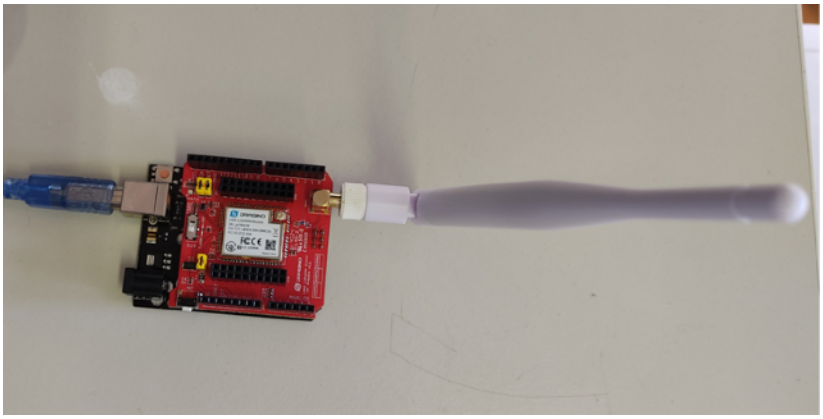


Xbee S1 Pro en fonctionnement autonome

Troisième idée : Shield LoRa LA66 - Dragino

Notre dernière solution envisagée fut d'utiliser le protocole de communication LoRa. Nous avons acheté les Shield LoRa LA66 de chez Dragino. Initialement configurés en LoRaWAN, et malgré de nombreuses tentatives avec le support technique du vendeur,

nous n'avons pas réussi à passer au mode Peer-to-peer permettant un dialogue émetteur-récepteur sans fil. Cet inconvénient technique a donc rendu les modules achetés inutilisables dans notre cas.
Le lancement étant très proche, nous n'avons pas eu le temps de rebondir sur une autre solution de télémétrie. D'autre part, du fait que la solution la plus fiable était les Xbee S1 "classiques" ayant une faible portée, nous avons décidé de ne pas intégrer l'expérience télémétrie dans notre fusée. Ceci nous a permis de nous concentrer sur l'électronique de base afin d'assurer le fonctionnement d'Eos pour ce premier lancement.



Shield LoRa LA66 de chez Dragino

Retour d'expérience

Pour la microélectronique, nous avons rencontré plusieurs complications. Les branchements et soudures reliant les différents Arduino étaient fragiles et peu fiables. Pour renforcer ces connexions, nous avions utilisé des crimps et des câbles JST, mais malgré ces précautions, la robustesse restait insuffisante. Face à ces limites, nous avons commencé à concevoir nos circuits sur PCB. À l'avenir, tous les projets seront directement réalisés sur PCB afin d'assurer une meilleure solidité et une intégration plus propre.

Nous avons également décidé de remplacer les Arduino par des ESP-32. Ces microcontrôleurs présentent de nombreux avantages : ils sont moins chers, plus performants, de taille réduite et consomment moins d'énergie. Ce changement constituera une amélioration significative pour la fiabilité et l'efficacité de nos systèmes.

Cependant, certaines solutions mises en place lors de ce projet se sont révélées efficaces et seront conservées. C'est le cas, par exemple, de l'utilisation de piles 9V pour l'alimentation : les tests seront réalisés avec des piles rechargeables, tandis qu'au pas de tir, des piles jetables seront utilisées pour garantir la fiabilité. De même, l'agencement électronique des microcontrôleurs a donné de bons résultats. En reliant les Arduino en parallèle, nous avons pu maintenir une tension unique sur l'ensemble du circuit. Le

transport d'énergie via des câbles à 11 brins s'est montré pratique et robuste, et cette organisation sera réutilisée pour nos futurs projets.

Malgré l'échec de la télémétrie, nous avons déjà pu acquérir une culture globale sur la télécommunication, et découvert différents types de protocoles de communication. Désormais, nous avons déjà des pistes pour nos prochains systèmes télémétriques, comme par exemple l'utilisation des *ESP LoRa*.

Capteurs

Généralités

Notre association étant novice dans ce genre de projet, nous avons choisi de partir sur du plug-&-play pour la plupart des capteurs, en utilisant des produits Arduino avec lesquels nous étions déjà familiers. Ainsi, voici une représentation de notre circuit :

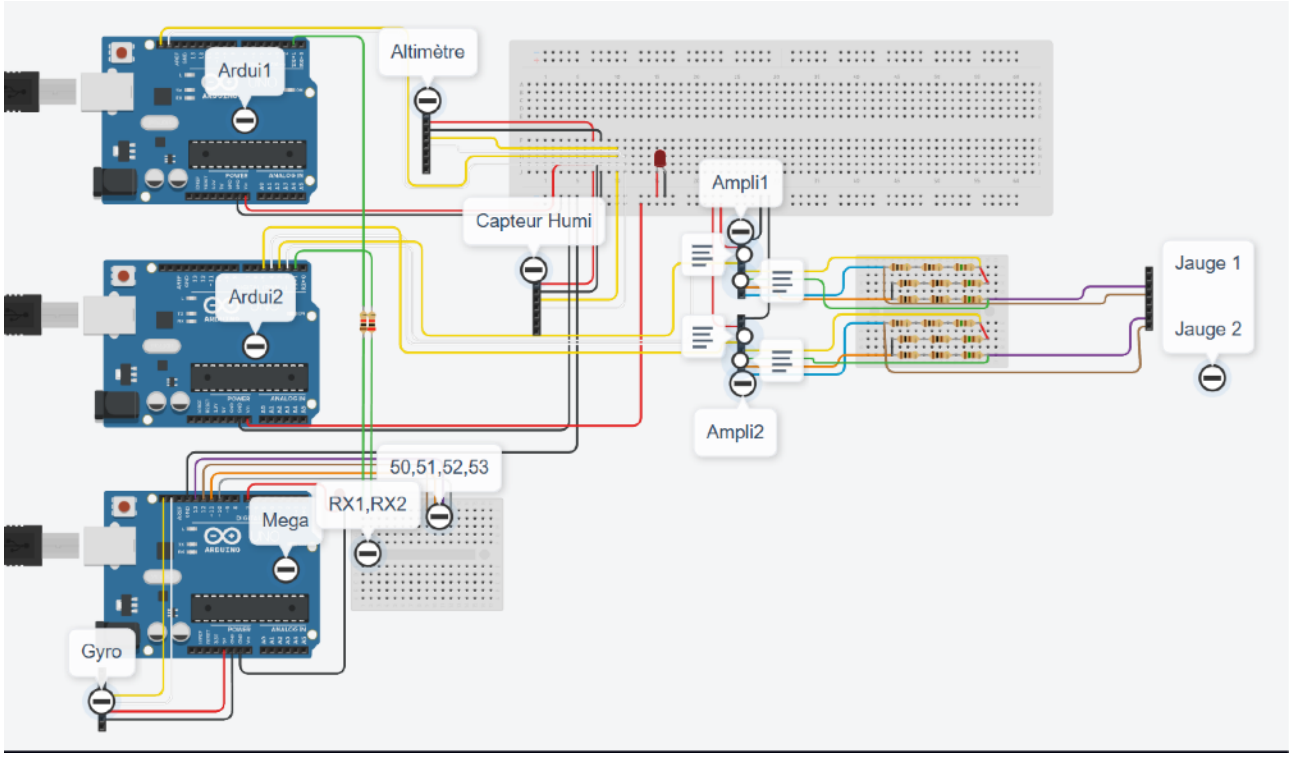


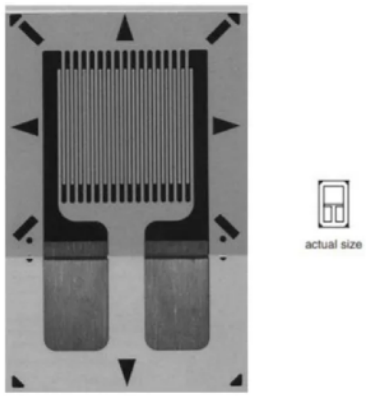
Schéma du montage électrique du pôle charge utile
Les barrettes 8 pins représentent les différents capteurs

Jauges de déformation

Pour la même raison, nous avons choisi de réaliser une expérience pour tester la résistance générale de notre fusée en vol. L'objectif d'une telle expérience était à la fois d'obtenir des données sur notre conception, mais aussi et surtout de pouvoir fournir un retour d'expérience documenté à la prochaine équipe du projet, expérience qui nous a grandement manquée pendant tout notre mandat.

Nous avons donc choisi, pour notre expérience principale, d'étudier les déformations mécaniques des ailerons de la fusée. Pour cela, nous avons décidé d'installer des jauges de déformation sur nos ailerons. Ces jauges sont des résistances variables, installées sur

une pièce soumise à des contraintes mécaniques, et permettant de mesurer les déformations subies par cette pièce selon un axe déterminé.



Jauges de déformation Micro-Measurements utilisées pour notre expérience
Source : fr.rs-online.com

Nous avons mené cette étude sur deux ailerons, en plaçant sur chacun une jauge de déformation montée en quart-de-pont de Wheatstone. Initialement, le montage devait comprendre deux jauges par aileron (quatre jauges en tout), montées en demi-pont afin de mesurer à la fois la traction et la compression sur les faces opposées de chaque aileron. Suite à des problèmes techniques, nous avons été contraints de remplacer deux jauges par des résistances fixes, pour aboutir à l'expérience finale avec deux quarts-de-pont, mesurant donc une déformation sur chacun des deux ailerons.

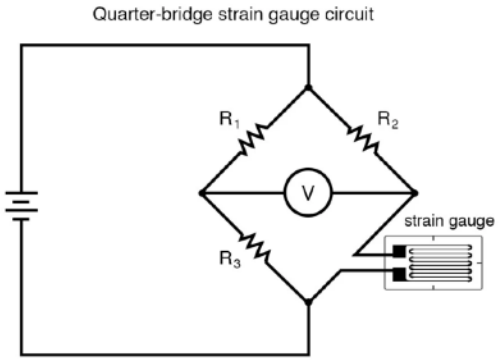


Schéma du montage électrique d'un quart-de-pont de Wheatstone
Source : All About Circuits

Les déformations (et donc les variations de résistance) étant a priori très faibles, nous avons relié les deux jauges à des amplificateurs HX711 (communication analogique). Les jauges et les amplificateurs étaient branchés sur la même carte Arduino, nommée ardui2.

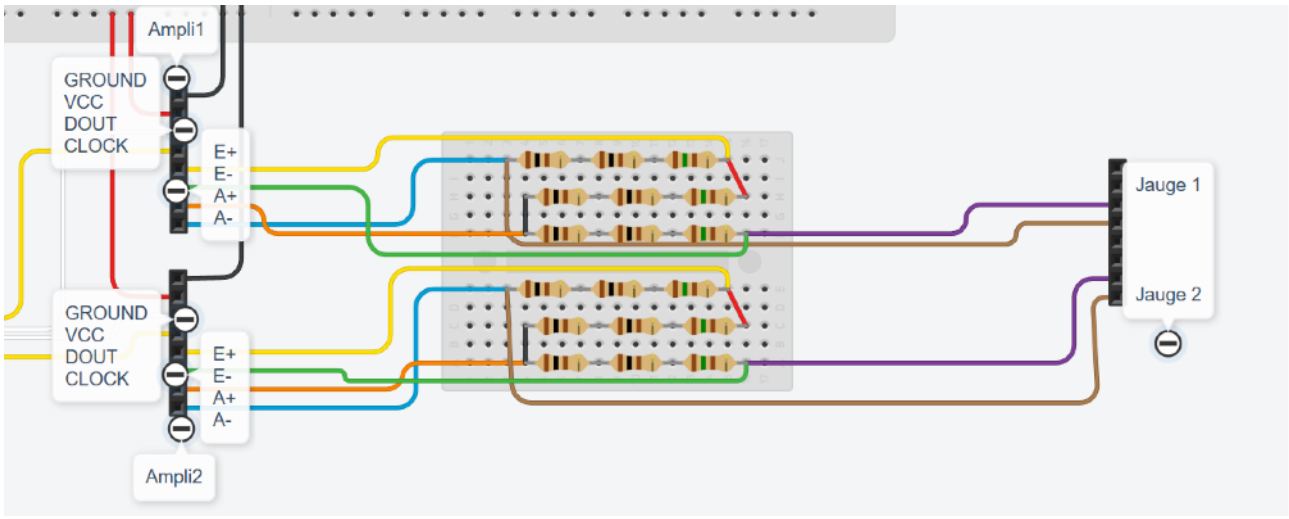


Schéma du montage électrique des amplificateurs
Les barrettes 8 pins représentent les 2 amplificateurs à gauche,
et les 2 jauges à droites

Concernant l'implantation du dispositif dans la fusée, les jauges ont été reliées à la charge utile via de longs fils courant le long des ailerons à l'extérieur, puis le long de la structure interne. De plus, comme l'ensemble des systèmes embarqués devait être amovible, nous avons mis en place un dispositif avec des connecteurs afin de pouvoir séparer facilement les câbles provenant des jauges - donc de l'extérieur de la fusée - avec ceux à l'intérieur de la fusée. Cela permettait également de raccorder les câbles extérieurs et intérieurs rapidement avant le lancement.

Une des grosses difficultés rencontrées au niveau de la récupération des données était la fréquence d'acquisition du thermomètre intérieur utilisé (nous y reviendrons plus tard). En effet, celui-ci nécessite beaucoup de temps entre chaque mesure pour obtenir une précision satisfaisante (2 Hz environ, pour une précision de +/- 0.06 °C). Pour que les jauges puissent mesurer les petites déformations auxquelles nous nous attendions, il nous aurait fallu une fréquence de mesure d'au moins 100 Hz, impossible en raison des autres capteurs embarqués et de la limitation des Arduino. Ainsi, nous avons décidé de ne mesurer que les plus grosses déformations, et nous avons baissé la précision des capteurs (grâce à la bibliothèque fournie) à +/-0.25°C, pour avoir une fréquence de mesure de 10 Hz.

Autres mesures

Comme évoqué précédemment, nous avons d'abord un capteur de température DS18B20 (protocole 1-wire) afin de surveiller la température au point le plus sensible de la fusée, à savoir les batteries et le panneau de contrôle.

Nous avons aussi voulu suivre d'autres paramètres de vol tels que la pression, la température extérieure et l'hydrométrie, et ce en fonction de l'altitude. Nous avons donc utilisé l'altimètre MPL3115A2 (protocole I2C), qui déduit l'altitude de la pression qu'il mesure, et l'hygromètre Grove 101990644 (protocole I2C), qui a aussi un capteur de température. Ceux-ci sont branchés sur les ports SDL et SCA d'une même carte Arduino, nommée ardui1. Suite à une limitation de la fréquence de mesure de l'altimètre, nous avons dû recréer une bibliothèque pour qu'il fonctionne plus rapidement, portant ainsi la fréquence de mesure à 10 Hz également. Pour que l'altimètre fonctionne correctement, nous aurons besoin de la pression au niveau de la mer au moment du décollage, et ce au mbar près pour avoir une bonne précision sur la donnée qu'il renvoie.
Les deux cartes Arduino convertissent leurs données en chaîne de caractères qui contient aussi la valeur renvoyée par la fonction millis() (le temps écoulé depuis le début du programme en ms) afin de pouvoir tracer ces données en fonction du temps. Cette chaîne de caractères est ensuite transmise à une carte Arduino Mega via les ports tx et rx, qui a une shield avec une carte SD. Les données sont donc enregistrées sur la carte SD dans des fichiers nommés ardui1 et ardui2 dès qu'elles ont été reçues par la Mega.

Voici sa représentation :

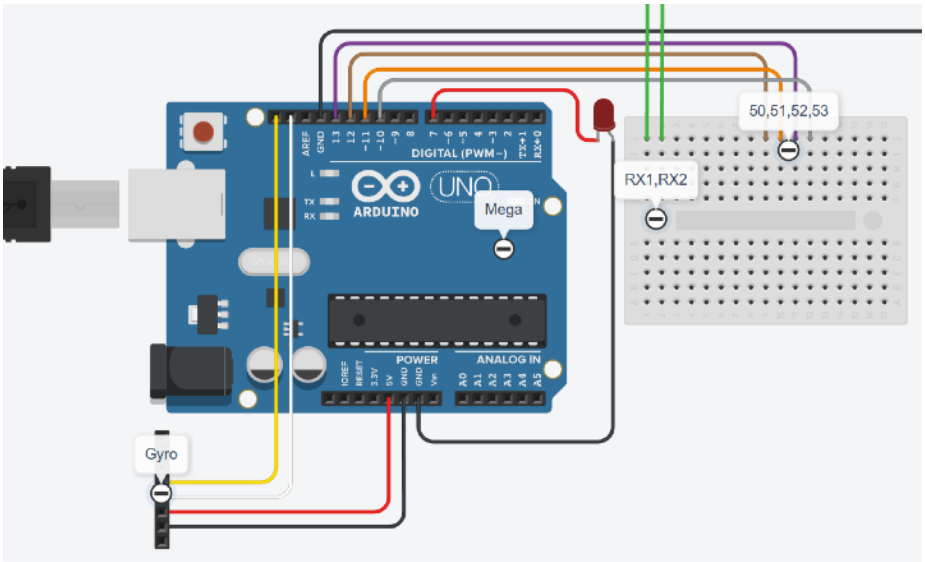


Schéma du montage électrique de la mega, la platine représente les pins supplémentaires. Le shield de la carde sd étant prévu pour une uno 3 rev, des branchements supplémentaires sont nécessaires

De plus, il est branché sur la Mega un dernier capteur, le gyroscope / altimètre / magnétomètre IMU 9DOF 101020585 (protocole I2C). Il va servir à retracer la trajectoire de la fusée et apporter une seconde estimation de l'altitude. Si on avait réussi à mettre en place la transmission de données, il nous aurait théoriquement permis de retrouver la fusée aisément grâce à la boussole. Cette année, les données sont donc stockées sur la carte SD, ce qui nous servira donc à tester cette méthode pour voir si elle est fonctionnelle (nous noterons la position au lancement et celle à l'atterrissage pour

pouvoir la comparer à notre estimation). Un calibrage est nécessaire au moment de l'allumage du capteur, il moyenne les forces qu'il subit sur une durée de 3 secondes. Néanmoins, le gyroscope a un petit défaut, il renvoie par défaut -1°/s même lorsqu'il n'y a aucun mouvement. Après vérification par chronophotographie, il s'agit simplement d'un offset.

La Mega comporte une spécificité : le shield est prévu pour une Arduino 3 rev, donc des branchements supplémentaires sont nécessaires entre les pins de la Mega elle-même.

Retour d'expérience

Nous avons été confrontés à plusieurs difficultés pour la mise en place de l'expérience. Tout d'abord, nous avons eu du mal à trouver de la documentation précise concernant les déformations mécaniques subies par les ailerons de fusées expérimentales. Cela nous a ralenti dans nos choix de réalisation de l'expérience (type de montage à utiliser, positionnement des jauges, choix des jauges, etc.).

Par ailleurs, nous avons rencontré quelques difficultés pour installer les jauges sur les ailerons. Au-delà de la difficulté de trouver les produits nécessaires pour le collage à des prix raisonnables, le processus en lui-même était particulièrement minutieux, et nous ne disposions donc que d'une marge d'erreur réduite. En effet, il aurait été financièrement difficile de racheter des jauges de déformation en cas de problème lors du collage. Malgré tout, nous avons pu coller les jauges avec succès grâce à Mme Lallemand, qui nous a apporté une aide précieuse en nous enseignant la procédure de collage.

Néanmoins, suite aux manipulations ultérieures du corps de la fusée, deux jauges - une sur chaque aileron - ont été endommagées et étaient inutilisables pour l'expérience. Nous avons toutefois su nous adapter en modifiant le montage expérimental pour utiliser des quarts-de-pont et mener l'expérience à son terme.

En revanche, nous avons commencé la procédure d'étalonnage très tardivement, ce qui nous a mis en difficulté pour finaliser l'expérience. Ainsi, nous recommandons vivement au prochain mandat de réfléchir à la procédure d'étalonnage le plus tôt possible, afin d'éviter des difficultés auxquelles ils pourront difficilement faire face en raison des délais dont ils disposent.

Finalement, comme nous n'avons pas pu mettre en place de transmission de données durant le vol, ni récupérer la fusée, nous n'avons pu récupérer aucune donnée chiffrée concernant la déformation des ailerons, et nous ne pouvons pas savoir si les systèmes embarqués ont fonctionné correctement en vol. Un des objectifs principaux pour le prochain projet de notre association sera donc d'assurer la retransmission des données pendant la phase de vol afin d'assurer leur récupération et leur analyse ultérieure.

Parachute

Généralités

Pour la confection du parachute nous avons opté pour la récupération d'une voile de Spinnaker d'un voilier, permettant plusieurs découpes et tests grâce à son importante surface et permettant de réduire l'impact carbone du projet tout en ayant une solution solide et efficace.

La forme du parachute est hémisphérique et comporte 6 gores cousues entre elles à la machine à coudre par deux coutures solides et une surpiqure.

Coutures des gores

Le rayon du parachute est de 70cm et nous avons opté pour un « spill hole » comptant pour 20% du diamètre total afin d'augmenter la stabilité.

Ci-dessus :
A gauche une des 6 gores (dont le patron a été découpé à la découpeuse laser pour une plus grande précision)
À droite la même gore découpée (le bas est droit mais ici est courbé à cause de la perspective)

La longueur des suspentes est de 150 mm (sans les nœuds de fixation), 2,14 fois le rayon du parachute. Les suspentes sont fixées au parachute par des œillets 4 mm insérés dans 4 renforts en toile de spi collés pour assurer une pré-tension puis cousus de manière circulaire (afin d'éviter les points de tension) sur le ralentisseur pour répartir les charges sur une plus grande surface du tissu. Nous avons ensuite effectué des tests statiques et dynamiques de traction pour valider la résistance au choc de l'ouverture. En supposant 3000 N de force totale exercée pour une vitesse à l'apogée de 30 m/s.

En utilisant la formule du cahier des charges et la vitesse à l'ouverture :

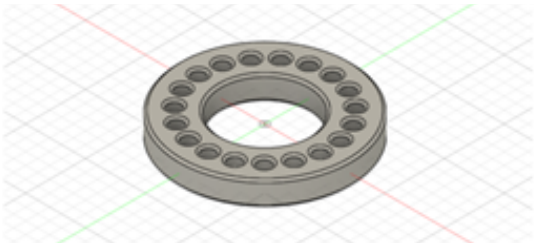
Force maximale à l'ouverture : $F = 0.5 \times 1.3 \times Surface \times V_{apogee}^2$ (en Newton)

Afin d'alléger la force au moment de l'ouverture sur les fixations des suspentes au parachute, nous avons opté pour 19 suspentes en nylon 2 mm.

Figure 1 : Anneaux

Figure 2 : Émerillon

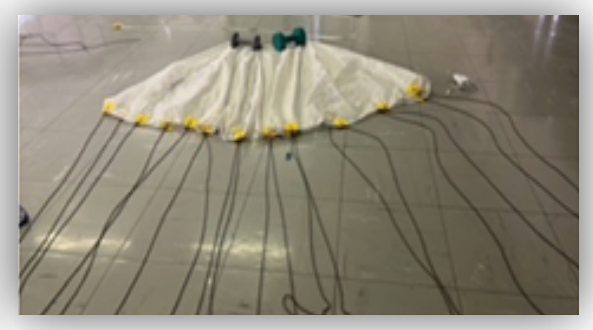
Le système anti-torche est composé d'un émerillon conçu pour supporter 300 kg (ci-dessus à droite) relié par une sangle 800 kg à un anneau qui rassemble les suspentes d'un côté (ci-dessus à gauche) et de l'autre par une sangle 800 kg de 2 m de long qui s'accroche à l'anneau d'attache du parachute (ci-dessus à gauche). Toutes les jonctions sont faites par des coutures solides en "X". Les suspentes sont maintenues séparées par un anneau anti-torche imprimé en PETG qui est retenu en bas des suspentes pour éviter qu'il n'empêche l'ouverture du ralentisseur.



Anneau anti-torche

Ci-dessus, l'anneau anti-torche (70 mm de diamètre extérieur et 36 mm de diamètre intérieur, des trous à 5,5 mm de diamètre) des congés et chanfreins sont ajoutés pour permettre un glissement plus efficace de la pièce sur les suspentes et éviter le risque de coupure.

Gonflement du parachute



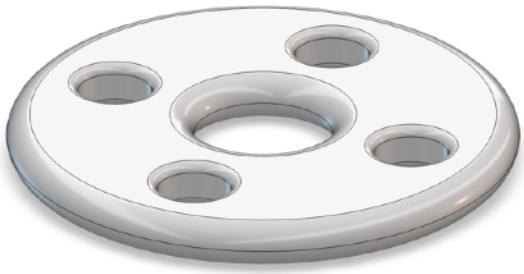
Fixation et répartition des suspentes au parachute

Retour d'expérience

Malheureusement, au cours des nombreux tests durant le C'Space, le parachute n'a pas survécu aux contraintes imposées multipliées par le facteur de sécurité. La toile, et non les coutures ou les suspentes, s'est déchirée. Mais grâce au contrôleur de planète science, Gilles Tahan, qui nous a très gracieusement prêté un parachute de rechange. Un cruciforme en toile de spi. On a appris qu'il est mieux taillé pour les hautes vitesses ; en effet, à haute vélocité, il se contracte et donc diminue sa surface, ainsi la force qu'il subit est diminuée et il résiste mieux. Le pôle parachute a beaucoup appris pendant le C'Space sur la qualité des toiles et les méthodes de pliage, par exemple en maison pour les parachutes cruciformes.

Parachute cruciforme en cours de pliage

Au niveau des attaches des suspentes à la toile, nous avons appris différentes techniques de couture pour répartir la force subie plutôt que d'utiliser des œillets pour faire passer les suspentes. Enfin, nous avons changé l'anneau anti torche afin d'accommoder les suspentes du nouveau parachute :



Anneau anti-torche version finale

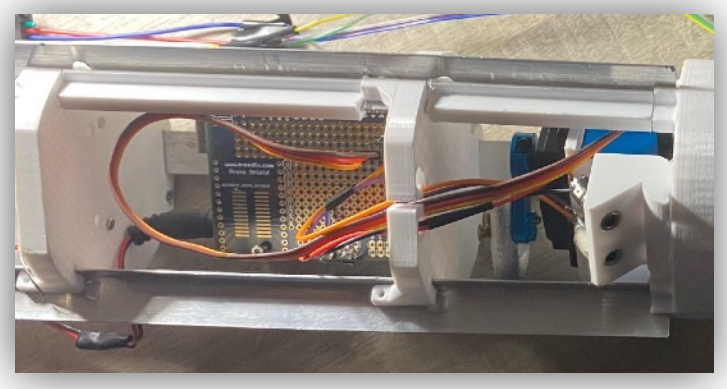
Séquenceur

Description mécanique

L'ouverture de la trappe du parachute est provoquée par un servomoteur FT35 et déclenchée au bout de 16 secondes, ce qui correspond à l'apogée de la fusée.
Le parachute est attaché à la trappe. L'ouverture de la trappe est garantie par la présence d'un ressort placé entre celle-ci et l'armature intérieure de la fusée. Le déploiement du parachute est quant à lui assuré par la vélocité de l'air.

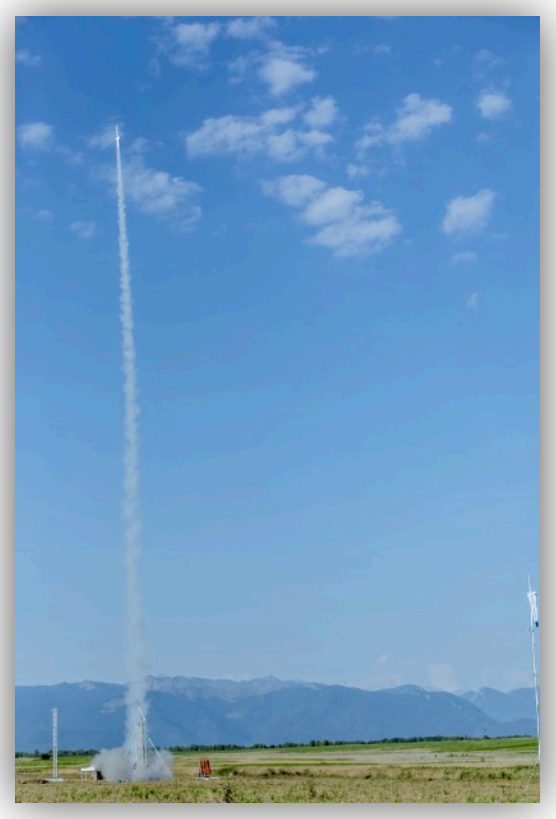
Description électronique

Le séquenceur a pour composants principaux une carte Arduino Rev 3 et deux ports jack 3,5 mm. Au décollage, un câble jack 3,5 mm passant à travers la rampe de lancement reliera les deux ports jack. Le séquenceur détectera la rupture de contact entre les deux ports ; ainsi, même si une des deux extrémités du câble ne se débranche pas proprement (s'arrache), le décollage sera détecté. À l'ouverture du parachute, l'Arduino déclenche le servomoteur FT35 mentionné dans la description mécanique.



C’Space

L’équipe a vécu un premier C’Space particulièrement intense, et a appris plus que jamais durant cette période de travail acharné (mention particulière à Christophe Desgoutte et Damien Montanier, respectivement père de Mika et Quentin, qui ont apportés une aide plus que précieuse lors de ce C’Space 2025). Après de très nombreuses modifications, que ce soit en structure ou en microélectronique, d’innombrables soudures et re-soudures, et des moments d’inquiétude comme de soulagement, Eos fut qualifiée le 6e jour du C’Space et lancée le 7e jour (11/07/2025) à 09h09min31s du matin. Elle a effectué un vol nominal couronné de succès. Seul bémol de l’opération : la fusée a atterri en zone rouge. Ce C’Space a été une expérience extraordinairement enrichissante, à travers les nombreuses rencontres permises par le C’Space et laisse un souvenir inoubliable à toute l’équipe.



Conclusion

Ce projet fut couronné de succès sur tous les plans et a pleinement rempli ses objectifs. Il établit des bases solides pour les futures équipes, afin d'assurer la continuité et la progression d'un tel projet au sein de l'association Astrolab.

Plus qu'un succès technique, cette première réalisation constitue une étape fondatrice. Elle illustre la capacité de notre équipe, partie d'une page blanche, à mener à bien la conception, la fabrication et le lancement d'une fusée expérimentale en seulement 7 mois, dans le respect des contraintes du C'Space.

La télémétrie constituera un axe majeur d'amélioration pour les prochains projets. L'objectif est de pouvoir récupérer les données de vol en temps réel. Cette évolution permettra d'analyser les données même si la fusée devient irrécupérable après l'atterrissage

Tout au long de l'aventure, chaque membre de l'équipe a pu développer des compétences concrètes dans des domaines variés : CAO, électronique embarquée, fabrication, gestion de projet et communication. Chacun a trouvé sa spécialité, tout en apprenant à collaborer dans un environnement technique exigeant.
Cette expérience a ainsi non seulement donné naissance à Eos, mais aussi formé une génération de membres prêts à transmettre leur savoir-faire et à porter encore plus haut les ambitions d'Astrolab.

Annexes

Code du séquenceur et de l'ouverture de la trappe

```
#include <Servo.h>

const int jackPin = 2;   // Broche où le câble jack est connecté
const int servoPin = 3;  // Broche de contrôle du servomoteur
const int lightPin = 12; // Broche de contrôle de la led

Servo monServo;

bool lastJackState;  // Dernier état stable du jack
bool jackState;

bool trappeState;

unsigned long debounceDelay = 1000; // Temps de stabilisation en ms
unsigned long lastJackChange = 0;

unsigned long countDown = 15400; //16400 - 1000 de sensi;
bool lightState = LOW;

unsigned long lastPrint = millis();

const int OUVERT = 144;
const int FERME = 123;
const int OPEN = true;
const int CLOSE = false;


bool decollage = false;

void setup() {
   Serial.begin(9600);             // Initialisation de la communication série
(optionnel, pour débogage)
   Serial.println("\\n-------------------- RESTARTING -------------------- \\n\
\n\\n");
  Serial.println("Initialisation");

  // Jack
  pinMode(jackPin, INPUT);  // Déclare la broche jackPin comme entrée
  monServo.attach(servoPin); // Attache le servomoteur à la broche servoPin
  monServo.write(OUVERT);         // Position initiale du servomoteur
  trappeState = OPEN;

  // Lumière
  pinMode(lightPin, OUTPUT);
  digitalWrite(lightPin, LOW);

  jackState = digitalRead(jackPin);
  lastJackState = digitalRead(jackPin);  // Dernier état stable du jack
}

void loop() {
  jackState = digitalRead(jackPin);

  // ---------------------- Detection des faux contacts ----------------------
  if (jackState != lastJackState) {
    lastJackChange = millis(); // Reset du timer si l'état change
```

```
    Serial.println("Changement détecté, attente stabilisation...");

    while ((millis() - lastJackChange) < debounceDelay)
    {
      if (digitalRead(jackPin) != jackState)
      {
        Serial.println("Faux contact");
        return;
      }
    }
    Serial.println("Stabilisation effectuée");
  }



  // ----------------------- Changement des branchements ----------------------
  if (jackState != lastJackState) {


          //  ----------------  Câbles  débranchés  ----------  DECOLLAGE
-------------------------
    if (jackState == LOW && lastJackState == HIGH && trappeState == CLOSE) {
      Serial.println("Décollage EOS - Attente ouverture trappe");



      unsigned long liftOff = millis();

      unsigned long lastChangeLight = millis();
      while(millis() - liftOff < countDown) {
        if (millis() - lastChangeLight > 50) {
            lastChangeLight = millis();
            lightState = changeLight(lightState);
        }
      }
      digitalWrite(lightPin, LOW);

      Serial.println("Ouverture trappe");
      monServo.write(OUVERT);
      trappeState = OPEN;

    }


          //  ----------------  Câbles  branché  ----------  SYSTEME  ARME
-------------------------
    if (jackState == HIGH && lastJackState == LOW) { // On les branche
      Serial.println("Armé");
      digitalWrite(lightPin, HIGH);

      monServo.write(FERME);
      trappeState = CLOSE;

    }
  }
  lastJackState = jackState;



  // ----------------------- Affichage ----------------------
  if (millis() - lastPrint > 1000) {
    // Debug pour vérifier si la condition de debounce fonctionne
    Serial.print("Etat Jack: ");
    Serial.println(digitalRead(jackPin));

    Serial.print("Temps écoulé depuis dernier changement: ");
```

```
    Serial.println(millis() - lastJackChange);
    lastPrint = millis();
  }


  delay(10); // Petite pause pour éviter la surcharge CPU


}
bool changeLight(bool state) {
  if (state == LOW) {
    digitalWrite(lightPin, HIGH);
    return HIGH;
  } else {
    digitalWrite(lightPin, LOW);
    return LOW;
  }
}
```

Acquisition des données des jauges de déformation

```
#include "HX711.h"
#include <OneWire.h>
#include <DallasTemperature.h>
#define ONE_WIRE_BUS 6

OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

#define DOUT  2
#define CLK  3

#define DOUT2  4
#define CLK2  5

HX711 scale1;
HX711 scale2;
char transmi[40];

float calibration_factor = -700000; // voir s'il faut le calibrer
float calibration_factor2 = -700000;

DeviceAddress tempSensor;

void setup()
{
  Serial.begin(9600);

  // sensors.begin();
  // if (sensors.getDeviceCount() == 0) {
  //   Serial.println("Aucun capteur de température détecté !");
  //   while (1);
  // }
  // sensors.getAddress(tempSensor, 0);
  // sensors.setResolution(tempSensor, 10);  // Forcer 9 bits = ~93 ms

  scale1.begin(DOUT, CLK);
  scale1.set_scale();
  scale1.tare();

  scale2.begin(DOUT2, CLK2);
  scale2.set_scale();
  scale2.tare();
}


void loop()
{
  static unsigned long lastMeasurementTime = 0;

  if (millis() >= 600000) { //à modifier, faut y réfléchir, là c'est 5min
    Serial.println("Arrêt du programme");
    while (1);
  }

  if (millis() - lastMeasurementTime > 100) {
    lastMeasurementTime = millis();

    //Amplis

    scale1.set_scale(calibration_factor);
    scale2.set_scale(calibration_factor2);
```

```
    float val1 = scale1.get_units();
    float val2 = scale2.get_units();

    //thermometre interieur

    // sensors.requestTemperatures();
    // float tempIn = sensors.getTempCByIndex(0);

    char transmi[64]; // Buffer pour transmission
    char buffer[16];  // Temp pour conversion nombre->texte

    transmi[0] = '\0'; // Nettoyer la chaîne

    //temps

    ultoa(millis(), buffer, 10);  // convertir millis() en string
    strcat(transmi, buffer);
    strcat(transmi, ";");


    // val1
    dtostrf(val1, 8, 5, buffer); // (valeur, largeur, décimales, tampon)
    strcat(transmi, buffer);
    strcat(transmi, ";");

    // val2
    dtostrf(val2, 8, 5, buffer);
    strcat(transmi, buffer);
    strcat(transmi, ";");

    // // température
    // dtostrf(tempIn, 6, 2, buffer);
    // strcat(transmi, buffer);


    Serial.println(transmi);

    }


}
```

Code de l'Arduino 1

```
#include <Wire.h>
#include "AHT20.h"

#define MPL3115A2_ADDR 0x60
#define SEA_LEVEL_PRESSURE 1021.6  // hPa

AHT20 AHT;
char transmi[80];  // Buffer ligne complète

void setup() {
  Serial.begin(9600);
  Wire.begin();
  AHT.begin();
  initMPL3115A2();
}

void loop() {
  static unsigned long lastRead = 0;
  unsigned long now = millis();

  // Mesure toutes les 100 ms (10 Hz)
  if (now - lastRead >= 100) {
    lastRead = now;

    // Lecture capteur pression
    float pressurePa = readPressure();       // en Pascals
    float pressure = pressurePa / 100.0;     // en hPa
     float altitude = 44330.0 * (1.0 - pow(pressure / SEA_LEVEL_PRESSURE, 1.0 /
5.255));

    // Lecture capteur humidité/température
    float humi = 0.0, tempHum = 0.0;
    AHT.getSensor(&humi, &tempHum);

    // Conversion float -> string
    char bufTime[12], bufHumi[10], bufPres[10], bufAlt[10], bufTemp[10];
    ultoa(now, bufTime, 10);
    dtostrf(humi,     6, 3, bufHumi);
    dtostrf(pressure, 6, 1, bufPres);
    dtostrf(altitude, 6, 1, bufAlt);
    dtostrf(tempHum,  5, 1, bufTemp);

    // Construction ligne
    strcpy(transmi, bufTime);
    strcat(transmi, ";");
    strcat(transmi, bufHumi);
    strcat(transmi, ";");
    strcat(transmi, bufPres);
    strcat(transmi, ";");
    strcat(transmi, bufAlt);
    strcat(transmi, ";");
    strcat(transmi, bufTemp);

    // Envoi
    Serial.println(transmi);
  }
}

// === Initialisation MPL3115A2 en mode baro continu 10 Hz ===
void initMPL3115A2() {
  writeRegister(0x26, 0x00);         // standby
  writeRegister(0x13, 0x07);         // enable data flags
```

```
  writeRegister(0x26, 0b00111000);   // baro, OSR=128
  writeRegister(0x26, 0b00111001);   // active
}

// Lecture pression brute (en Pascals)
float readPressure() {
  Wire.beginTransmission(MPL3115A2_ADDR);
  Wire.write(0x01);  // registre pression
  Wire.endTransmission(false);
  Wire.requestFrom(MPL3115A2_ADDR, 3);

  byte msb = Wire.read();
  byte csb = Wire.read();
  byte lsb = Wire.read();

  long raw = ((long)msb << 16) | ((long)csb << 8) | lsb;
  raw >>= 4;

  return raw / 4.0;  // donne pression en Pa
}

// Écriture registre I2C
void writeRegister(byte reg, byte value) {
  Wire.beginTransmission(MPL3115A2_ADDR);
  Wire.write(reg);
  Wire.write(value);
  Wire.endTransmission();
}
```

Code de l'Arduino 2

```
#include "HX711.h"
#include <OneWire.h>
#include <DallasTemperature.h>
#define ONE_WIRE_BUS 6

OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

#define DOUT  2
#define CLK  3

#define DOUT2  4
#define CLK2  5

HX711 scale1;
HX711 scale2;
char transmi[40];

float calibration_factor = -700000; // voir s'il faut le calibrer
float calibration_factor2 = -700000;

DeviceAddress tempSensor;

void setup()
{
  Serial.begin(9600);

  sensors.begin();
  if (sensors.getDeviceCount() == 0) {
    Serial.println("Aucun capteur de température détecté !");
    while (1);
  }
  sensors.getAddress(tempSensor, 0);
  sensors.setResolution(tempSensor, 10);  // Forcer 9 bits = ~93 ms

  scale1.begin(DOUT, CLK);
  scale1.set_scale();
  scale1.tare();

  scale2.begin(DOUT2, CLK2);
  scale2.set_scale();
  scale2.tare();
}


void loop()
{
  static unsigned long lastMeasurementTime = 0;

  if (millis() >= 600000) { //à modifier, faut y réfléchir, là c'est 5min
    Serial.println("Arrêt du programme");
    while (1);
  }

  if (millis() - lastMeasurementTime > 100) {
    lastMeasurementTime = millis();

    //Amplis

    scale1.set_scale(calibration_factor);
    scale2.set_scale(calibration_factor2);
```

```
    float val1 = scale1.get_units();
    float val2 = scale2.get_units();

    //thermometre interieur

    sensors.requestTemperatures();
    float tempIn = sensors.getTempCByIndex(0);

    char transmi[64]; // Buffer pour transmission
    char buffer[16];  // Temp pour conversion nombre->texte

    transmi[0] = '\0'; // Nettoyer la chaîne

    //temps

    ultoa(millis(), buffer, 10);  // convertir millis() en string
    strcat(transmi, buffer);
    strcat(transmi, ";");


    // val1
    dtostrf(val1, 8, 5, buffer); // (valeur, largeur, décimales, tampon)
    strcat(transmi, buffer);
    strcat(transmi, ";");

    // val2
    dtostrf(val2, 8, 5, buffer);
    strcat(transmi, buffer);
    strcat(transmi, ";");

    // température
    dtostrf(tempIn, 6, 2, buffer);
    strcat(transmi, buffer);


    Serial.println(transmi);

    }


}
```

Code de l'Arduino Mega

```
#include <SD.h>
#include "AK09918.h"
#include "ICM20600.h"
#include <Wire.h>

// === CONFIG DEBUG ===
#define DEBUG_MODE 0 // 1 pour afficher les logs série

#if DEBUG_MODE
  #define DEBUG_PRINT(x) Serial.print(x)
  #define DEBUG_PRINTLN(x) Serial.println(x)
#else
  #define DEBUG_PRINT(x)
  #define DEBUG_PRINTLN(x)
#endif

// === CONSTANTES ===
#define SD_CS 53
double declination = 2.63;

// === CAPTEURS ET VARIABLES ===
AK09918 ak09918;
ICM20600 icm20600(true);
AK09918_err_type_t err;

int16_t acc_x, acc_y, acc_z;
int16_t rot_x, rot_y, rot_z;
int32_t x, y, z;
int32_t offset_x = 0, offset_y = 0, offset_z = 0;
double roll, pitch;
File Ardui1;
File Ardui2;
File Mega;

// === CALIBRATION ===
void calibrate(uint32_t timeout, int32_t *offsetx, int32_t *offsety, int32_t
*offsetz) {
  int32_t x_min, x_max, y_min, y_max, z_min, z_max;
  ak09918.getData(&x, &y, &z);
  x_min = x_max = x;
  y_min = y_max = y;
  z_min = z_max = z;

  uint32_t timeStart = millis();
  while ((millis() - timeStart) < timeout) {
    ak09918.getData(&x, &y, &z);
    if (x < x_min) x_min = x;
    if (x > x_max) x_max = x;
    if (y < y_min) y_min = y;
    if (y > y_max) y_max = y;
    if (z < z_min) z_min = z;
    if (z > z_max) z_max = z;
    delay(100);
  }

  *offsetx = x_min + (x_max - x_min) / 2;
  *offsety = y_min + (y_max - y_min) / 2;
  *offsetz = z_min + (z_max - z_min) / 2;
}

extern unsigned int __heap_start;
extern void *__brkval;
```

```
void freeMemory() {
  int32_t free_memory;
  if ((int32_t)__brkval == 0) {
    free_memory = ((int32_t)&free_memory) - ((int32_t)&__heap_start);
  } else {
    free_memory = ((int32_t)&free_memory) - ((int32_t)__brkval);
  }
  Serial.print("Mémoire libre : ");
  Serial.print(free_memory);
  Serial.println(" octets");
}

// === SETUP ===
void setup() {
  Serial.begin(9600);
  Serial1.begin(9600);
  Serial2.begin(9600);
  Wire.begin();

  pinMode(SD_CS, OUTPUT); // Important sur Mega

  Serial.println("- setup start -");
  freeMemory();

  err = ak09918.initialize();
  if (err != AK09918_ERR_OK) {
    DEBUG_PRINTLN("Echec magneto");
    while (1);
  }

  icm20600.initialize();

  if (ak09918.switchMode(AK09918_POWER_DOWN) != AK09918_ERR_OK ||
      ak09918.switchMode(AK09918_CONTINUOUS_100HZ) != AK09918_ERR_OK) {
    DEBUG_PRINTLN("Echec switchMode");
  }

  while (ak09918.isDataReady() != AK09918_ERR_OK) delay(100);

  if (!SD.begin(SD_CS)) {
    DEBUG_PRINTLN("Echec SD");
    while (1);
  }
  delay(500);

  Ardui1 = SD.open("Ardui1", FILE_WRITE);
  if (Ardui1) {
    DEBUG_PRINT("Fichier créé : ");
    DEBUG_PRINTLN("Ardui1");
    Ardui1.println("Temps(ms); humi; pressure; altitude; tempHum");
    Ardui1.flush();
  } else {
    DEBUG_PRINTLN("Echec CSV");
    while (1);
  }

  DEBUG_PRINTLN("Ardui1 ok");

  Ardui2 = SD.open("Ardui2", FILE_WRITE);
  if (Ardui2) {
    DEBUG_PRINT("Fichier créé : ");
    DEBUG_PRINTLN("Ardui2");
    Ardui2.println("Temps(ms); humi; pressure; altitude; tempHum");
    Ardui2.flush();
  } else {
```

```
    DEBUG_PRINTLN("Echec CSV");
    while (1);
  }

  DEBUG_PRINTLN("Ardui2 ok");

  Mega = SD.open("Mega", FILE_WRITE);
  if (Mega) {
    DEBUG_PRINT("Fichier créé : ");
    DEBUG_PRINTLN("Mega");
     Mega.println("Temps(ms);Acc X;Acc Y;Acc Z;Rot X;Rot Y;Rot Z;Mag X;Mag Y;Mag
Z;Roll;Pitch;Heading");
    Mega.flush();
  } else {
    DEBUG_PRINTLN("Echec CSV");
    while (1);
  }

  DEBUG_PRINTLN("Mega ok");

  delay(1000);
  freeMemory();
  calibrate(10000, &offset_x, &offset_y, &offset_z);
  DEBUG_PRINTLN("Calibration ok");
  Serial.println("- setup ok -");
}

// === LOOP ===
void loop() {
  static unsigned long lastMeasurementTime = 0;

  if (millis() >= 600000) {
    DEBUG_PRINTLN("Arrêt du programme");
    Ardui1.close();
    Ardui2.close();
    Mega.close();
    while (1);
  }

  if (Serial1.available()){

    if (Ardui1) {
      Ardui1.println(Serial1.readStringUntil('\n'));
      Ardui1.flush();
    } else {
      DEBUG_PRINTLN("Err ecriture SD - Ardui1");
    }
  }

  if (Serial2.available()){

    if (Ardui2) {
      Ardui2.println(Serial2.readStringUntil('\n'));
      Ardui2.flush();
    } else {
      DEBUG_PRINTLN("Err ecriture SD - Ardui2");
    }
  }

  if (millis() - lastMeasurementTime > 100) {
    lastMeasurementTime = millis();

    acc_x = icm20600.getAccelerationX();
    acc_y = icm20600.getAccelerationY();
    acc_z = icm20600.getAccelerationZ();
```

```
    rot_x = icm20600.getGyroscopeX();
    rot_y = icm20600.getGyroscopeY();
    rot_z = icm20600.getGyroscopeZ();

    ak09918.getData(&x, &y, &z);
    x -= offset_x;
    y -= offset_y;
    z -= offset_z;

    roll = atan2((float)acc_y, (float)acc_z);
      pitch = atan2(-(float)acc_x, sqrt((float)acc_y * acc_y + (float)acc_z *
acc_z));

       double Xheading = x * cos(pitch) + y * sin(roll) * sin(pitch) + z *
cos(roll) * sin(pitch);
    double Yheading = y * cos(roll) - z * sin(pitch);
    double heading = 180 + 57.3 * atan2(Yheading, Xheading) + declination;

    // === Construction de transmi en char[] ===
    char transmi[128];
    char tmp[16];

    sprintf(transmi, "%lu;", millis());

    sprintf(tmp, "%d;", acc_x); strcat(transmi, tmp);
    sprintf(tmp, "%d;", acc_y); strcat(transmi, tmp);
    sprintf(tmp, "%d;", acc_z); strcat(transmi, tmp);

    sprintf(tmp, "%d;", rot_x); strcat(transmi, tmp);
    sprintf(tmp, "%d;", rot_y); strcat(transmi, tmp);
    sprintf(tmp, "%d;", rot_z); strcat(transmi, tmp);

    sprintf(tmp, "%ld;", x); strcat(transmi, tmp);
    sprintf(tmp, "%ld;", y); strcat(transmi, tmp);
    sprintf(tmp, "%ld;", z); strcat(transmi, tmp);

    sprintf(tmp, "%.2f;", roll ); strcat(transmi, tmp);
    sprintf(tmp, "%.2f;", pitch); strcat(transmi, tmp);
    sprintf(tmp, "%.2f", heading); strcat(transmi, tmp);

    if (Mega) {
      Mega.println(transmi);
      Mega.flush();
    } else {
      DEBUG_PRINTLN("Err ecriture SD - mega");
    }

    Serial1.write(transmi);
  }
}
```