MINES SPACE

MSE
MACH1.1

RAPPORT MARSAUT 3 - MSE

VERSION N°02

2023

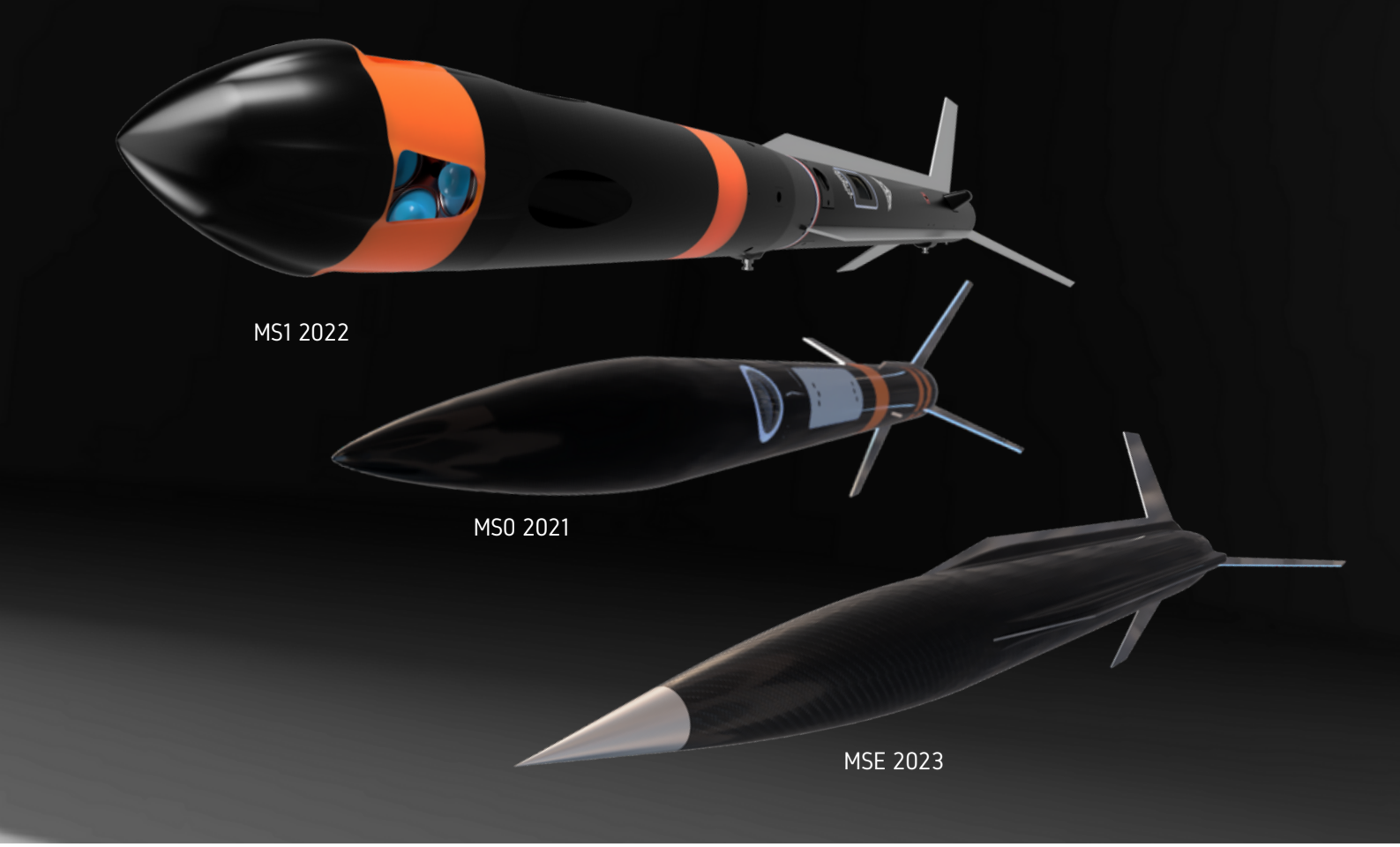
RÉALISATION D'UNE FUSÉE EXPÉRIMENTALE
DANS LE CADRE DU C'SPACE

MSE
MACH 1.1

MINES SPACE 2023

MSE
MACH 1.1

SOMMAIRE :



LA MISSION

MSE MACH 1.1

L'histoire de Mines Space a commencé en 2019, avec le rassemblement de plusieurs étudiants passionnés d'aérospatiale. Poussé par l'ambition de certains, l'objectif premier était de pouvoir participer à la réalisation d'une fusée-sonde. Face à l'ampleur qu'a pris ce groupe, nous avons décidé de fonder Mines Space.

Après avoir réalisé les fusex Marsaut 0 et Marsaut 1, nous avons décidé de lancer un nouveau défi de taille : MSE, une fusée supersonique capable d'emmener une charge utile au-dessus du mach.

CONTEXTE

On entend régulièrement parler de « FUSEX », contraction de « FUSée » et de « EXpérimentale », ces deux termes désignant le même vecteur, un lanceur de masse moyenne environ de 5 à 10 kg pouvant atteindre des altitudes supérieures à 1 000 m.
Chaque FUSEX est unique en possédant sa propre expérience scientifique ou innovation technique. De cette façon, il y a de nombreuses expériences et démonstrateurs techniques qui peuvent être utilisés dans une FUSEX, comme les études atmosphériques à travers l'utilisation d'un ou plusieurs capteurs, l'analyse des propriétés dynamiques du vol, ou encore l'essai de nouveau système modifiant ou contrôlant une caractéristique du lanceur (nouveau type de parachute, système de télémesure, contrôle de roulis ...).

La réalisation d'un projet du type Fusex en France se déroule avec l'aide du CNES (Le Centre National d'Étude spatiale) et planète Science (Association dédiée à la Science et à la jeunesse). Ils soutiennent l'initiative en encadrant l'ensemble des projets spatiaux en France. Chaque année, ils organisent la campagne nationale de lancement des projets étudiants dans le domaine de l'espace, le C'Space. C'est pour cette occasion après un contrôle minutieux que les fusées sont mises à feu.

OBJECTIFS

Mines Space est aujourd'hui une association en constante expansion : possédant maintenant une structure stable ainsi que plusieurs sources de financement, l'association à la capacité d'accueillir plusieurs projets portés par des équipes d'étudiants des Mines de Saint-Etienne, alternants ou non. Le projet MSE (MarSonique, dite Marsaut E) est porté par la première génération de membre de l'association et s'inscrit dans l'objectif de créer des plateformes de bases pour différents cas d'usage:

- Marsaut 0 : Petite Fusex, pro54, abordable et accessible pour de petites expériences (environ 1kg à 2000m) ou pour une équipe débutante;
- Marsaut 1 : Fusex de taille moyenne, pro75, capacité de larguer 3kg à 1900m avec son système de séparation.
- Marsaut E : Fusex supersonique, pro54, capacité d'emport de 350g à 2500m avec une pointe à mach 1.1 (380m/s).

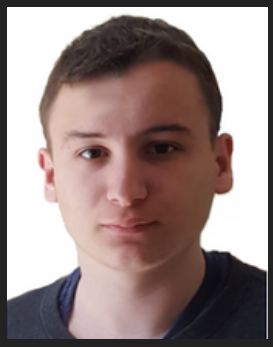
L'objectif de cette année pour l'équipe #1 de Mines Space est donc de développer cette plateforme supersonique et qualifier un système de calcul, simulation et étude de la stabilité et des contraintes d'une Fusex supersonique.
Les nouvelles équipe de Mines Space s'occuperons de créer différentes bases pour mini-fusées.

Organisation du club

Les personnes derrière MSE

PAUL BOYMOND

Ingénieur chez VALOTEC

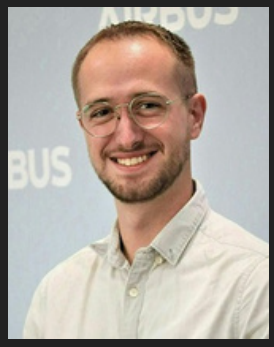


PAUL MIAILHE

Etudiant à l'ISAE-SUPAERO

LOUIS BARBIER

Ingénieur chez Airbus Defence & Space



L'équipe de MSE est composée de 3 passionnés provenant de deux écoles différentes :

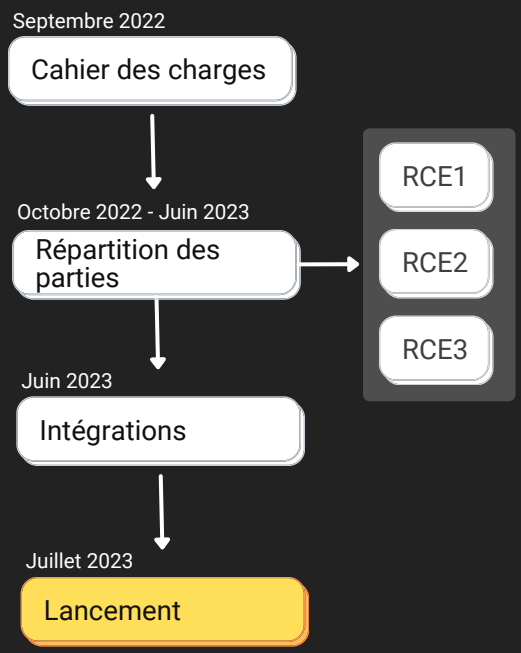
- École des Mines de Saint-Étienne
- École supérieure d'électricité (Supelec)

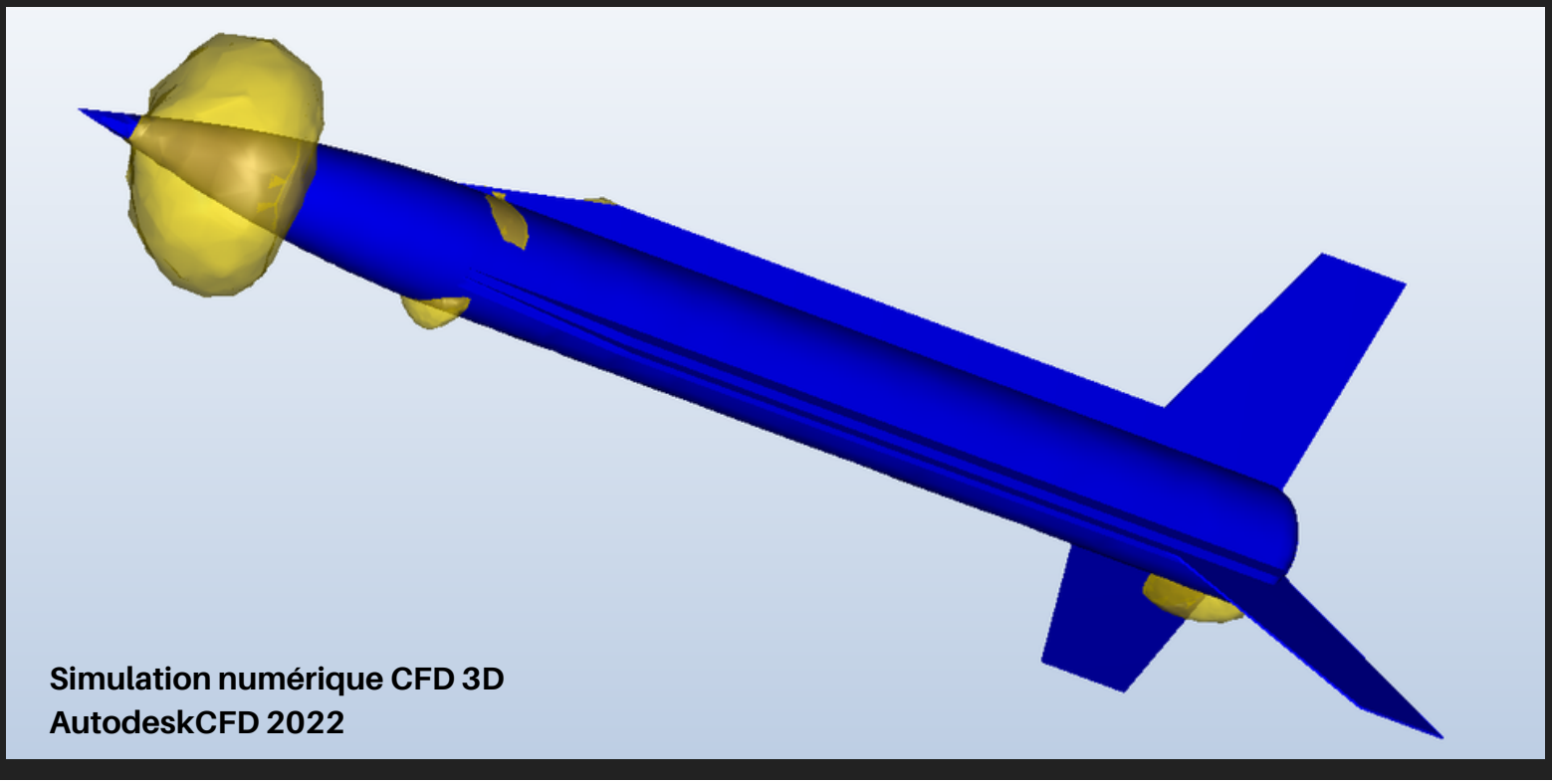
Rôles :

- Paul Boymond : Responsable mécanique, conception, intégration et production,
- Paul Miailhe : Responsable électronique, étude de stabilité, et station sol
- Louis Barbier : Responsable logiciel embarqué.

L'organisation des projets chaque années

L'organisation de Mines Space est particulière du fait que les membres sont en alternance. Ainsi, durant les périodes entreprises (70% du temps), nous travaillons en autonomie (aussi bien dans les développements que pour le matériel : nous n'avons pas de locaux et utilisons donc notre propre matériel : imprimante 3D, Fraiseuse numérique, four CMS, Oscilloscope ...). Nos entreprises n'étant pas toutes à Saint-Étienne, nous sommes repartis sur plusieurs villes, Toulouse, Paris, Saint-Étienne, Valence ... Cela rend le pilotage de projet particulièrement challengeant : nous organisons des points d'avancements réguliers et utilisons des services en ligne simplifiant la collaboration.
2 semaines tous les 1.5mois, nous nous retrouvons tous pour les "périodes école" et procédons à une phase d'intégration : nous réunissons toutes les parties individuelles et vérifions le bon fonctionnement.





Simulation numérique CFD 3D
AutodeskCFD 2022

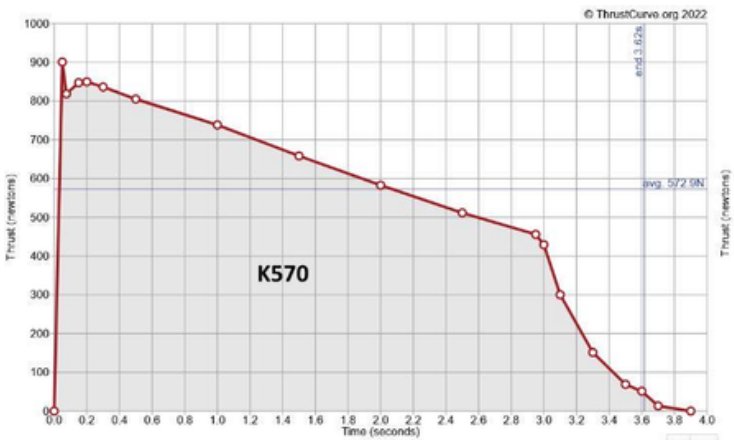
Ligne directrice de MSE

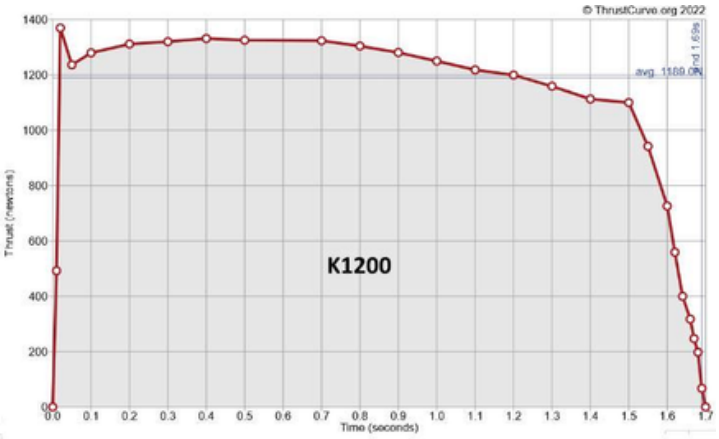
MSE est une petite fusex, proche du gabarit de MS0 : 1150mm de hauteur pour 90mm de diamètre. La différence se trouve dans la masse, 5.5kg sans propulseur pour MS0 contre 3kg pour MSE.

Nous avons choisi de faire une petite fusex propulsée par un pro54 pour les raisons suivantes :

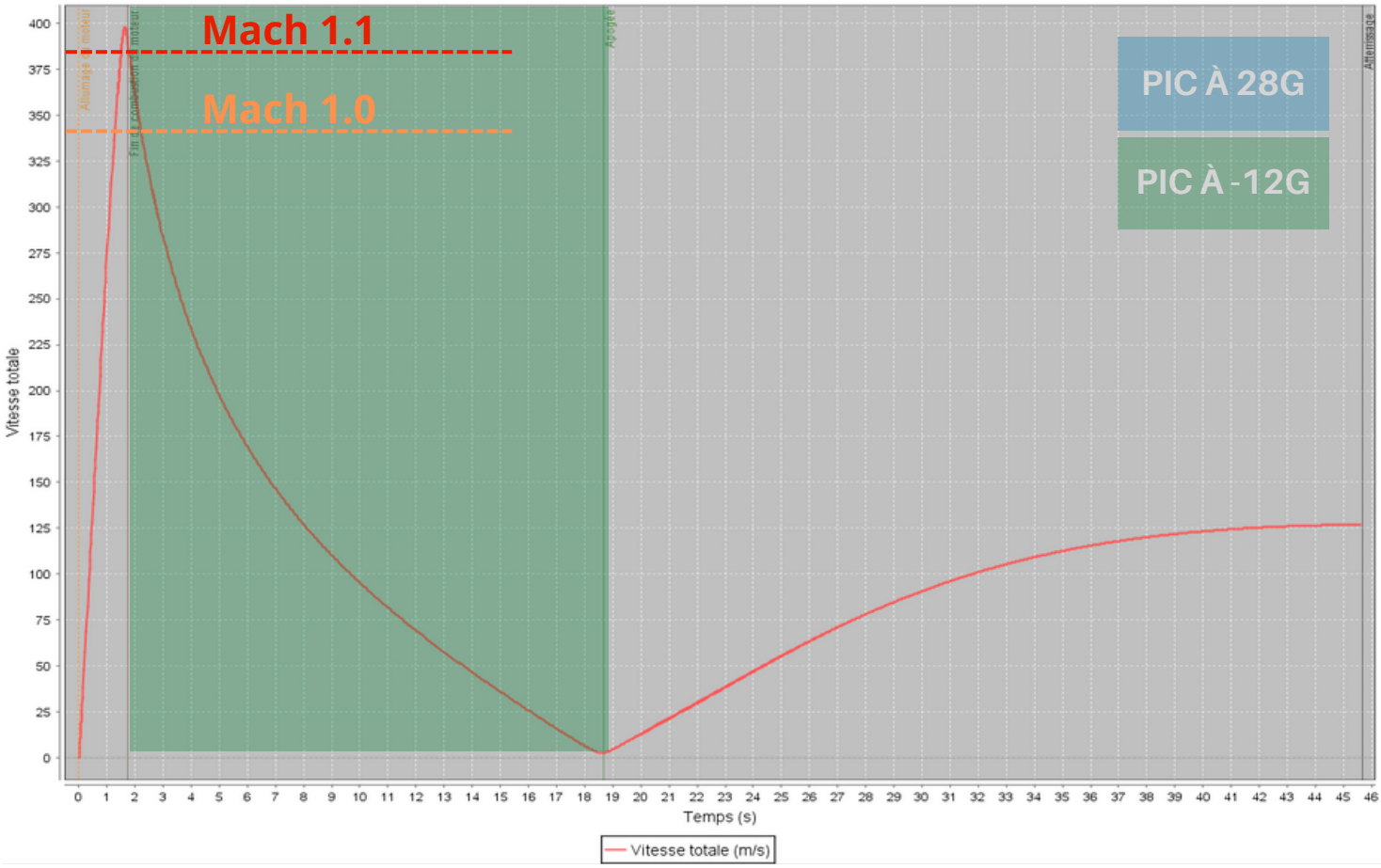
- Le pro54 et la faible inertie de la fusex font qu'elle ne dépasse pas les 2300m d'altitude,
- Une portée limitée, à la fois en balistique (1km) et sous parachute (<1km), nous visons également une portance basse.

Le propulseur pro54 habituellement utilisé ne permet pas d'atteindre notre objectif de mach 1.1 avec une telle fusée, c'est pourquoi nous souhaitons faire une demande pour une autre variante de pro54 : le K1200. Encore réduire la masse de la fusex n'est pas envisageable pour des raisons de faisabilité. 3kg étant déjà particulièrement challengeant. Cela nous exposerait également a des accélérations encore plus importantes (>35g).





Cette variante de propulseur a une durée de combustion presque 2 fois moindre que le K570 et garde une impulsion quasi-constante.
Ce propulseur nous permettrait d'atteindre la vitesse maximum de 398ms (mach 1.16) et dépasser le mur du son pendant environ 0.9s. Contre 339m/s (mach 0.98) en pointe avec un K570.



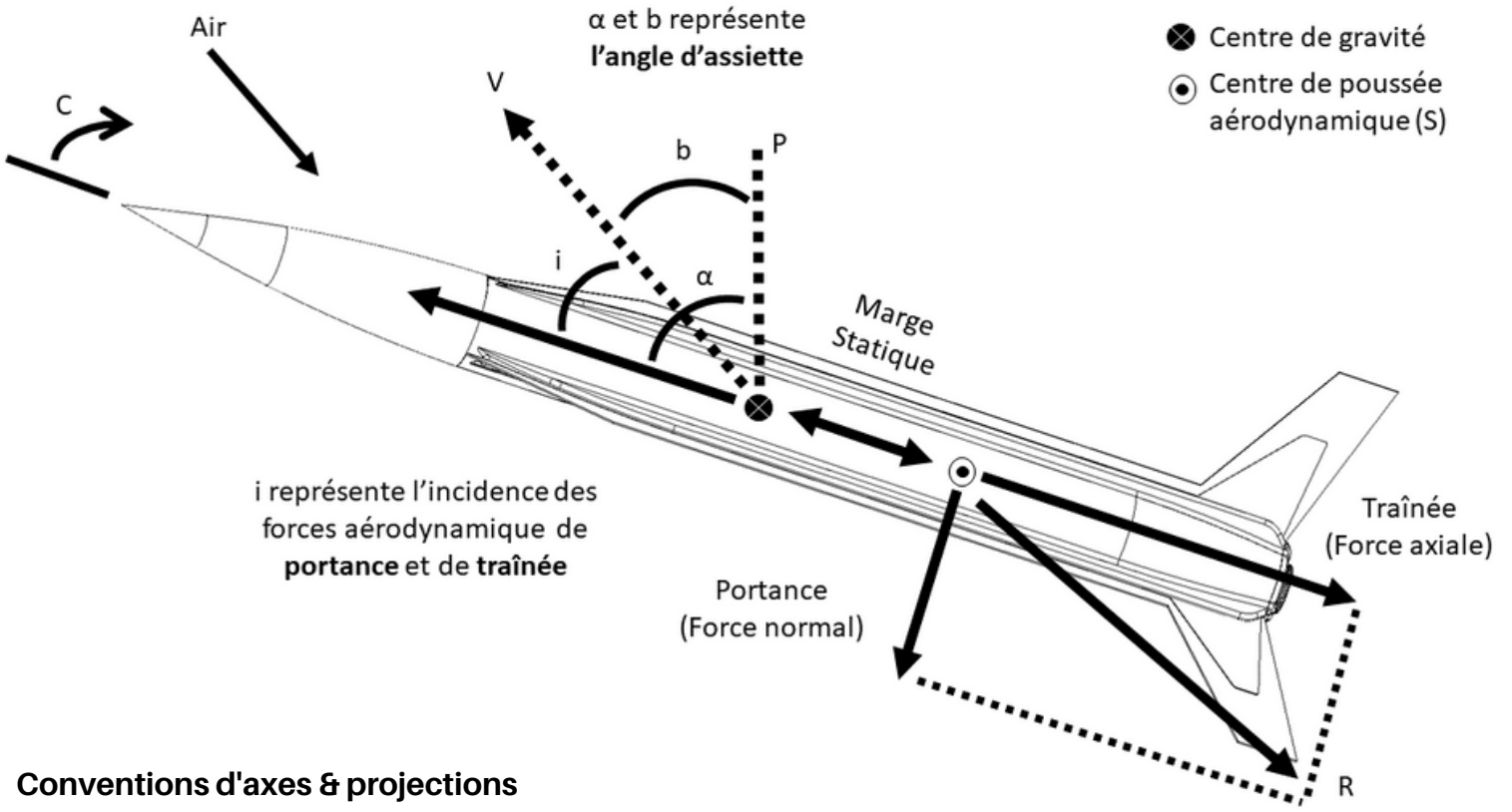
La combustion plus rapide de ce propulseur implique une plus grande accélération. On observe un pic à 28g et 2s à plus de 20g pendant le décollage. Lorsque le propulseur s'éteint, la faible masse de la fusée et la haute vitesse implique d'importants frottements qui vont fortement ralentir la fusex, avec un pic à 12g , cette fois ci négatif. (Les simulations ont été réalisé avec OpenRocket).

L'ambition du projet MSE

Deux raisons principales nous poussent à réaliser ce projet :

- L'ensemble des logiciels de simulations de fusée expérimentale se base sur la méthode d'étude de stabilité statique décrite dans le rapport de James S. Barrowman, ingénieur de la NASA. Cette méthode simple permet de calculer avec facilité les stabilités des fusées dans le domaine subsonique. Néanmoins, cette méthode se limite pour les fusées ayant une vélocité inférieure à MACH 0.6 (démontré par des essais en soufflerie réalisés par la NASA sur deux modèles de fusée réaliser avec la méthode Barrowman).De nombreux clubs aimeraient réaliser des expériences dans le domaine transsonique ou supersonique. Nous voulons ainsi fournir au des outils, une méthode et modèle pour faciliter le calcul de la stabilité à travers la réalisation de ce projet.

- Dans le cadre actuel du C'Space organisé à Tarbes sur la base militaire du 1er régiment de hussards parachutistes, il existe une restriction d'altitude. Elle s'établit entre 2500 à 3000 mètres. Pour faciliter la conception de fusée supersonique, les clubs utilisent majoritairement des moteurs puissants (pro 75) amenant leur projet au-delà des limites réglementaires. Nous voulons à travers MSE démontrer la possibilité de réaliser une fusée supersonique (MACH 1.1) en utilisant un moteur moins puissant (Pro54) tout en respectant les limites d'altitudes imposées. Cette fusée sera partagée en open source comme les précédentes réalisations du club.



Conventions d'axes & projections de MSE (engin stable).

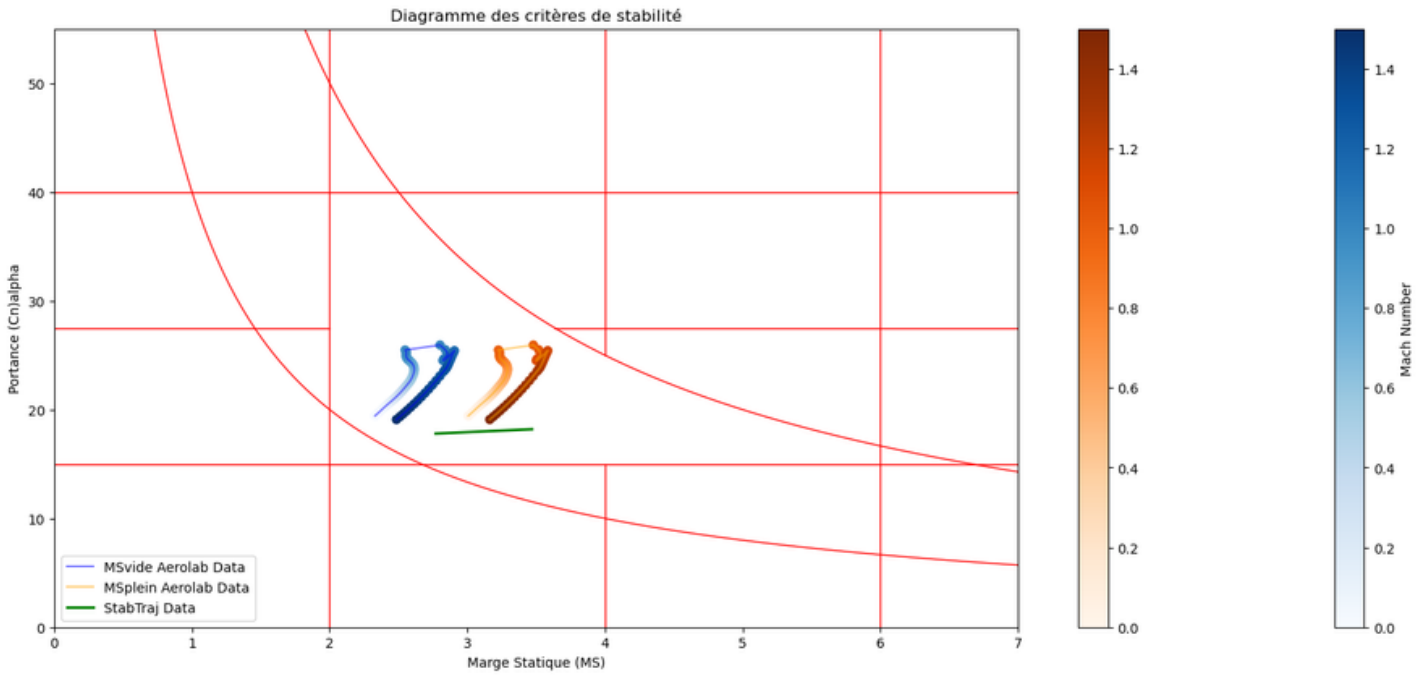
L'étude de la stabilité

En 1966, James Barrowman formalisa la méthode d'estimation de la Portance et sa position d'après la théorie des écoulements des fluides linéaires. Cette méthode a très largement été adoptée par les fuséologues amateurs et est encore utilisée de nos jours. Cette méthode reste néanmoins imparfaite, car en déterminant ces coefficients aérodynamiques en ne prenant pas en compte les variations des écoulements des filets d'air, elle reste imprécise pour les engins à forte vitesse (> MACH 0,6). Ainsi, il est alors essentiel de recalculer ces coefficients. Il existe deux méthodes pour déterminer ces paramètres :

- L'utilisation d'une soufflerie (onéreux et peu accessible)
- L'utilisation d'outils numériques de simulation (moins précise, mais plus accessible)

Dans notre cas, il est seulement envisageable d'utiliser des solutions de simulation numérique. Aujourd'hui, il existe de nombreuses solutions. La plus simple est d'utiliser le logiciel AEROLAB. Il a été conçu à l'aide de plusieurs études CFD et d'étude sous soufflerie réalisée par la NASA. Il est actuellement non certifié dû au manque d'information. Il est disponible en open source sur le site de Richard Nakka. L'autre solution consiste à réaliser des simulations CFD pour obtenir les coefficients aérodynamiques de notre fusée. Pour le développement de MSE, on va utiliser les deux méthodes pour déterminer les coefficients aérodynamiques et ainsi déterminer la stabilité de notre fusée.

Le CNES effectue également une étude CFD pour valider la stabilité. C'est cette analyse, effectuée quelques mois avant le lancement, qui confirme la viabilité du projet.



Cette année, nous avons mis l'accent sur l'exploitation des données d'Aerolab afin d'effectuer un pré-dimensionnement de notre fusée expérimentale. Le script que nous avons développé en Python offre une analyse approfondie de chaque élément des critères de stabilité (tels que la marge statique, la portance...) en relation avec la vitesse de la fusée. L'ensemble des données aérodynamiques comme le centre de pression, le gradiant de portance fut produit grâce à Aerolab.
Un rapport de stabilité en régime supersonique décrit en détaille notre démarche et notre script pour déterminer les paramètres. (Voir le rapport en annexe)

Toutefois, l'estimation de la marge statique s'est avérée légèrement inférieure selon les données d'OPENFOAM simulées par le CNES. Puisque l'étude du CNES fait référence, nous avons modifié notre centre de masse pour respecter le seuil minimal de la marge statique. (Voir le rapport du CNES joint en annexe.)

Le document de description de l'analyse de la stabilité dans le régime supersonique est disponible à la fin de ce rapport de projet.

L'architecture de MSE

Capteur Pitot:

- Capteur de pression différentiel 26PCGFA6D Honeywell: Ce capteur est un élément clé pour la détermination précise de la vitesse de notre fusée supersonique.
- Prise d'air sur l'ogive en loi de haack: Cette caractéristique, optimisée pour l'étude du choc supersonique, capte l'air directement depuis la pointe de l'ogive, offrant des mesures précises et fiables.

Avionique :

- Séquenceur : Un dispositif électronique crucial pour le déroulement des étapes du vol. Il détecte le décollage et l'apogée, activant ainsi le système de récupération.
- Expérience : Système électronique dédié à l'acquisition des données des capteurs et à la transmission de celles-ci via radio, en utilisant une modulation LoRa pour une communication efficace et fiable.

Système de récupération :

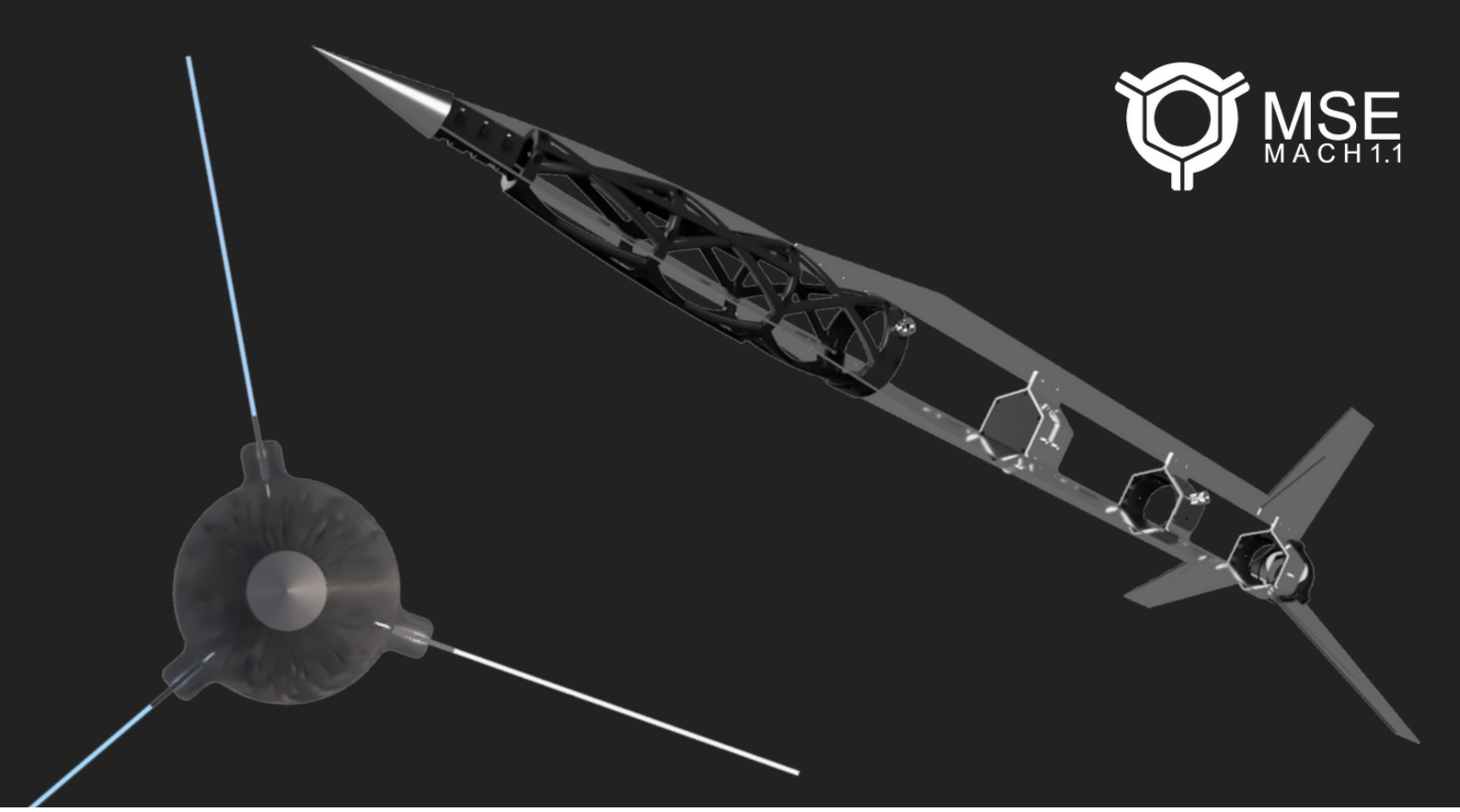
- Trappe parachute : Le mécanisme simple action privilégie la robustesse et la fiabilité. il est muni de deux motoréducteurs, actionnant des tiges filetées directement vissées dans la trappe. Un système de ressort vient plaquer la trappe sur la fusée sans risque de bloquer les motoréducteurs

Accessoires :

- Rack de Batterie modulable : Un ensemble de batteries Keeppower 14500 de 3,7V et 1100mAh fournissent l'énergie nécessaire pour alimenter toute la fusée.
- Double Caméra : Equipée d'une carte de traitement, ces caméras sont activées par l'avionique et enregistrent les images en local, offrant une vue précieuse de la mission.

Propulseur:

- Cesaroni Pro54 k1200, fourni par le cnes. Ce propulseur libère sa poussée 2 fois plus vite que les Pro54 habituellement utilisés au CSpace, nous permettant d'atteindre une plus grand vitesse tout en respectant le plafond maximum (et impliquant 30g au décollage !).

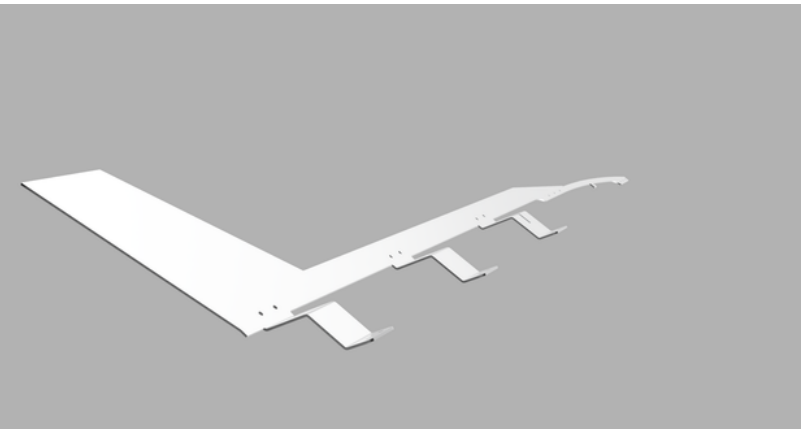


Architecture Mécanique

De la même façon que nos fusex précédentes, MSE repose sur une structure porteuse dite "externe" car principalement à l'extérieur de la peau. Pour atteindre notre vitesse cible de mach 1.1 (377m/s) avec un Cesaroni pro54, nous avons d'importantes contraintes de masse : objectif moins de 3.5kg sans propulseur (Nous atteindrons finalement 3.3Kg sans propulseur).

Voici les points principaux de la mécanique de MSE :

- Structure composée de 3 tôles pliées en aluminium, parcourant toute la hauteur de la fusée,
- Anneaux anti-torsion répartis le long de la fusée,
- Pointe de la coiffe en aluminium directement soutenu par la structure (structure interne au niveau de la coiffe),
- Treillis interne en polycarbonate "anti-écrasement" sous la coiffe,
- Coiffe en polycarbonate, neutre pour les antennes,
- Peau non porteuse en sandwitch composite (Carbone),
- Pièce imprimée en Polycarbonate, renforcé à 20% en fibre de carbone.

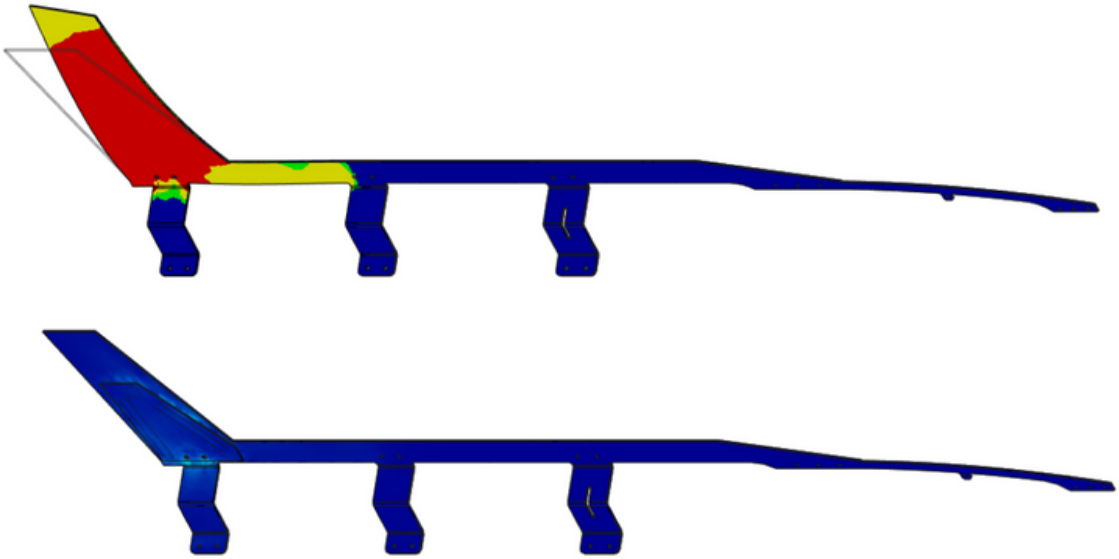


La photo ci-contre montre l'une des trois tôles composant la structure.
L'utilisation de tôles pliées permet de simplifier la structure et les assemblages, en limitant l'utilisation de bagues de centrages. Les trois tôles identiques viennent se visser les unes dans les autres. On retrouve quelques bagues "anti-torsions" permettant de maintenir l'angle de 120° entre les ailerons pendant tout le vol. Une seule tôle ne pèse que 275g.

Lien du CAD de MSE
https://grabcad.com/library/marsaut-e-mse-fusex-2023-1

Particularités de MSE

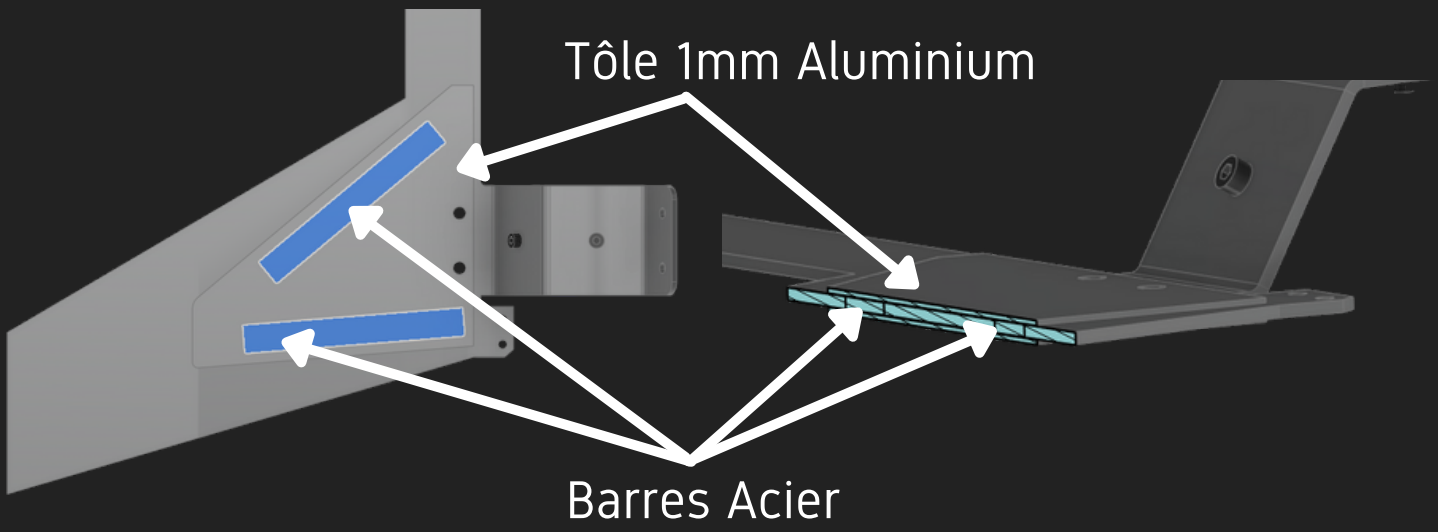
Le cahier des charges Fusex nous indique que les ailerons doivent pouvoir subir une force transversale de 250N sur leur centre de masse.



Les simulations ci-dessus montre en premier temps un ailerons uniquement constitué d'une tôle d'aluminium de 2mm d'épaisseur. On observe un déplacement de plus de 100mm et une grande zone de rupture de la pièce.
La deuxième simulation présente l'aileron renforcé (comme expliqué plus bas) avec deux barres en acier et deux tôles de 1mm d'aluminium. Le facteur de sécurité passe à 15 et le déplacement à moins de 2mm en bout d'aileron, réduisant ainsi considérablement le risque de rupture de l'aileron.

Le sandwich est réalisé de la manière suivante :

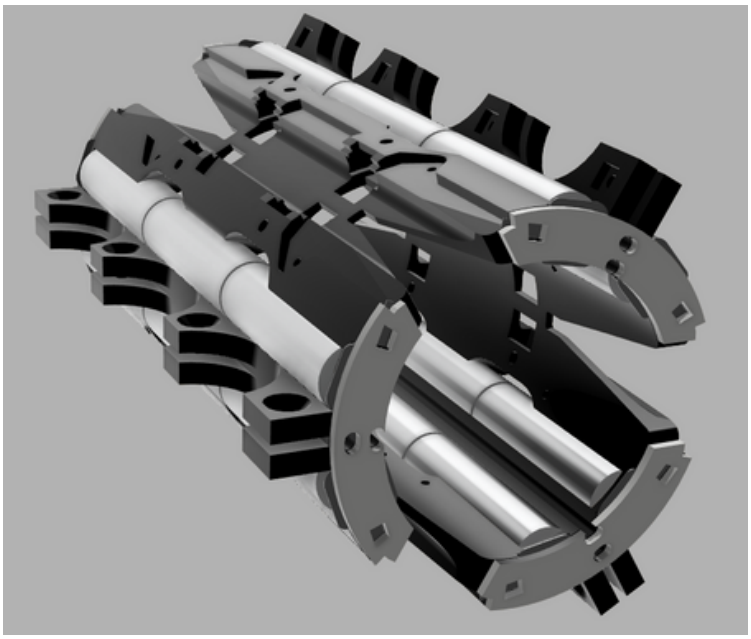
- Tôle 1mm Aluminium > Barre Acier/Aileron > Tôle 1mm Aluminium
- Le tout assemblé avec une colle bi-composant époxy.



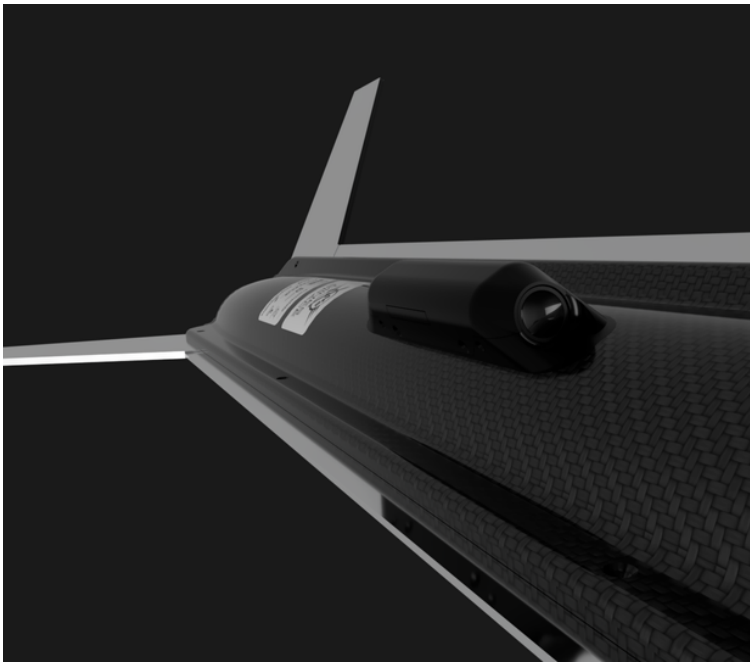
Concevoir et fabriquer MSE représente un challenge bien plus complexe que nos fusex précédentes, du aux contraintes que le vol supersonique apporte : 30g ainsi qu'une forte compression au niveau de la coiffe. N'ayant pas d'objectif particulier sur la vitesse ou l'altitude sur nos projets précédant, nous utilisions habituellement une peau en aluminium de 2 à 3mm d'épaisseur pesant entre 1 et 2kg. Cela n'est pas envisageable pour MSE. Nous avons décidé de réaliser la peau de MSE en sandwich composite carbone, nous amenant à 90g par peau (<300g par fusée).

Le document de description des peaux et du procédé de fabrication est disponible à la fin de ce rapport de projet.

La photo ci-dessus montre le compartiment batteries, les deux caméras et les interrupteurs de réglage de la trappe et d'alimentation. On y voit également l'électronique déportée des deux caméras, reliées par deux napes. Les câbles d'alimentation remontent ensuite vers le rack électronique (vers la gauche sur la fusée) dans les gaines tressées.



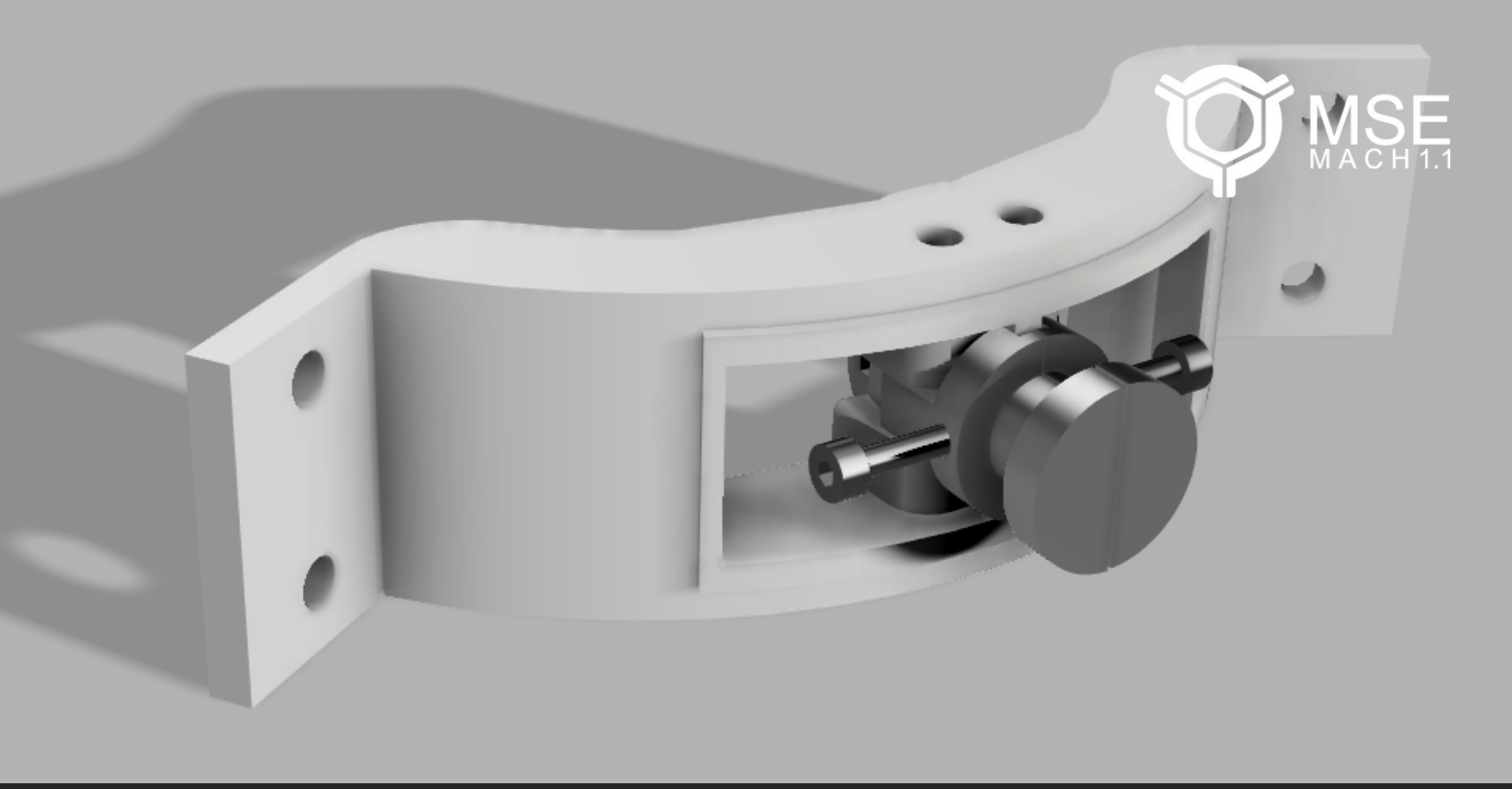
Du à un manque de place dans la coiffe, nous avons choisi de déplacer les batteries plus bas dans la fusée (ce qui posera des problèmes de centre de gravité plus tard...). Ne disposant que de seulement 16mm entre le propulseur et la peau, nous avons choisi d'utiliser des accus lithium 14500 en 3,7v et 1,1Ah (14mm de diamètre) tout autour du propulseur. Ce système modulable s'adapte à notre besoin, il permet de fournir une large gamme de tensions et de capacités en branchant en série ou parallèle les 6 jeux de 3 accus. Pour ce lancement, nous avions besoins de 4 batteries de 9v., soit 4 jeux de 3 accus en série.

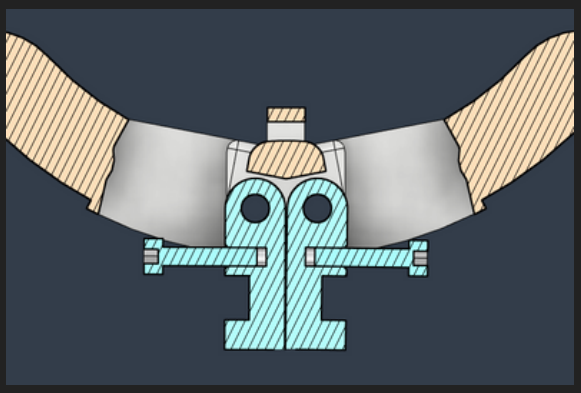


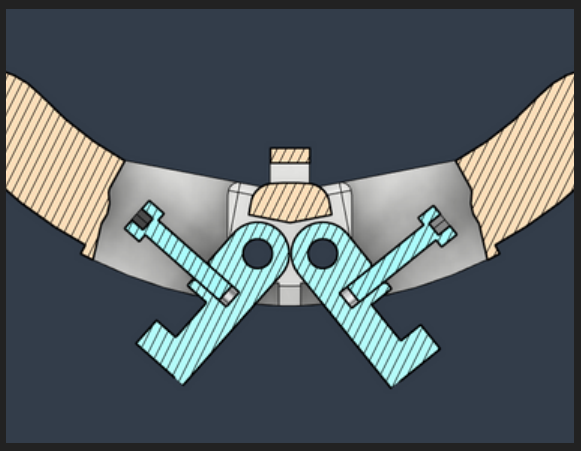
Bien que cela n'aide pas l'aérodynamisme, nous avons fait le choix de faire sortir deux caméras du gabarit de la fusée, une orientée vers le bas pour regarder la déformation des ailerons et une vers le haut, pour regarder la trappe parachute et la coiffe.

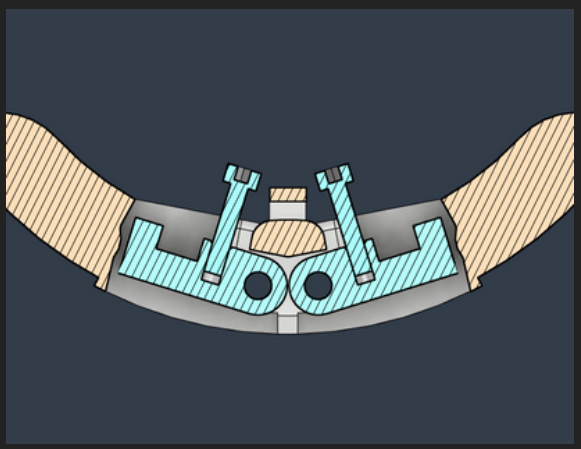
Ces caméras possèdent une IMU pour stabiliser les images une fois au sol. Elles sont également contrôlées par le séquenceur, qui les allume juste avant le décollage et les éteint quelques minutes après l'apogée.

Leur électronique déportée permet de limiter l'encombrement autour des caméras et de la placer plus bas dans la fusée où il reste quelques cm3 !









Dans l'optique d'améliorer l'aérodynamisme (pas de commentaire sur les caméras qui sortent du gabarit), nous avons voulu mettre à l'essais un système permettant de rentrer les patins de guidage de la trappe.

Ce système est composé d'un patin de forme standard coupé en deux. Sa forme lui permet d'être verrouillé par la rampe en position ouverte. Une fois celle-ci quittée, un élastique ouvre le patin en deux et le ramène dans le gabarit de la fusée. Aucune électronique ni actionneur n'est donc nécessaire. Par faute de place, nous avons finalement décidé de ne pas mettre de trappe pour couvrir le mécanisme.

Nous avons choisi de réaliser les patins en impression 3D SLS en inox. Bien que désignés pour être usinés de manière conventionnelle, l'impression 3D était bien moins couteuse. A la vue de la trajectoire de la fusée en sortie de rampe, nous pouvons affirmer que le système a bien fonctionné.

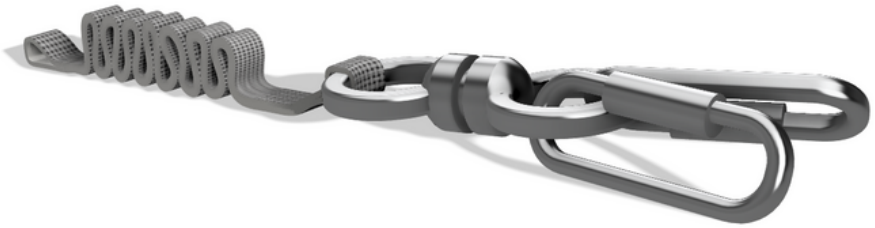
Système de récupération

Au lancement de ce projet, nous avons comme à notre habitude, fait un état de l'art sur les fusex supersoniques ayant volées au C'Space. La première chose que l'on peut observer est qu'aucune des fusex n'ont fait un 'nominal' (retour sous parachute en bon état). Parmi les crashs, on observe 2 raisons principales :

- Arrachement de la trappe lors du passage du mach,
- Casse sur la ligne de récupération (suspente, émerillon, ...).

Nous avons donc bien compris que le système de récupération va se voir imposer de fortes contraintes.

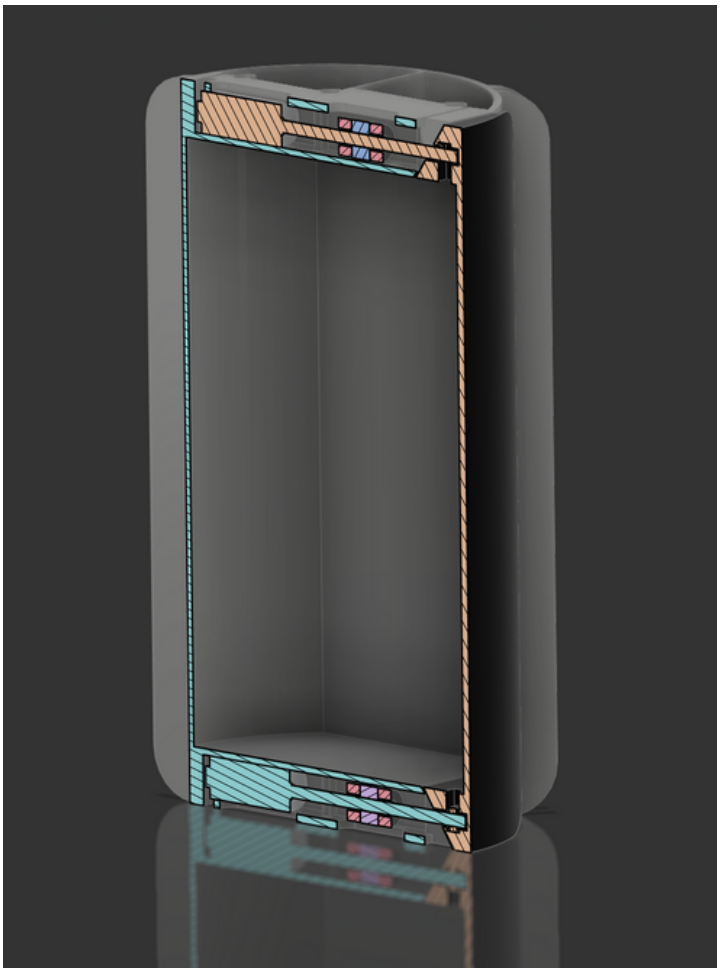
Une fois de plus, la décision d'acheter un parachute commercial a été prise. Seul particularité, nous avons décidé d'installer un parachute de ralentissement, visant à absorber le premier choc puis de déployer le parachute principal.



Le système à deux parachutes est l'une des plusieurs technique que nous avons pu tester avec succès sur Marsaut 1. On peut également trouver le harnais 4 points attaché à la fusée, ainsi qu'une sangle permettant de réduire le choc à l'ouverture.

Nous sommes également en train de tester de nouvelles techniques pour notre club, toujours dans l'objectif de réduire les contraintes sur le parachute à l'ouverture. On retrouve par exemple un "slider ring" qui ralentis l'ouverture du parachute, un système élastique en parallèle de la suspente principale, ...

Système de récupération

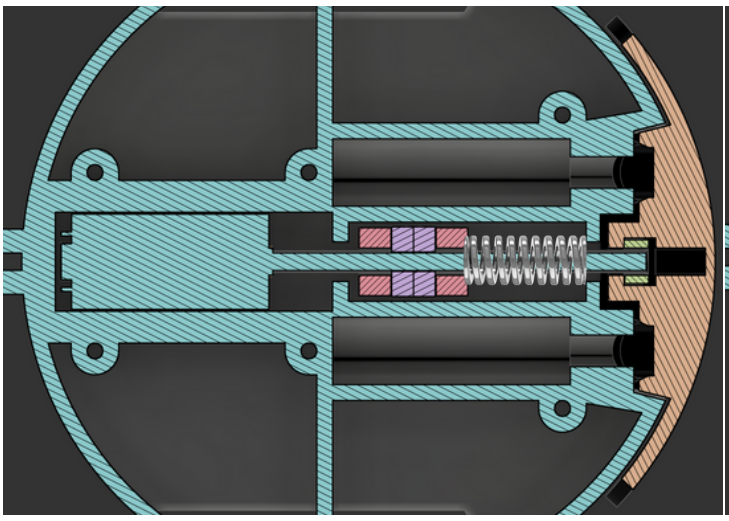


Encore une fois, ce système vient directement de nos missions précédentes. Après avoir fortement simplifié le mécanisme sur MS1, notre système à double motoréducteurs revient toujours plus simple avec seulement 2 pièces en mouvement.

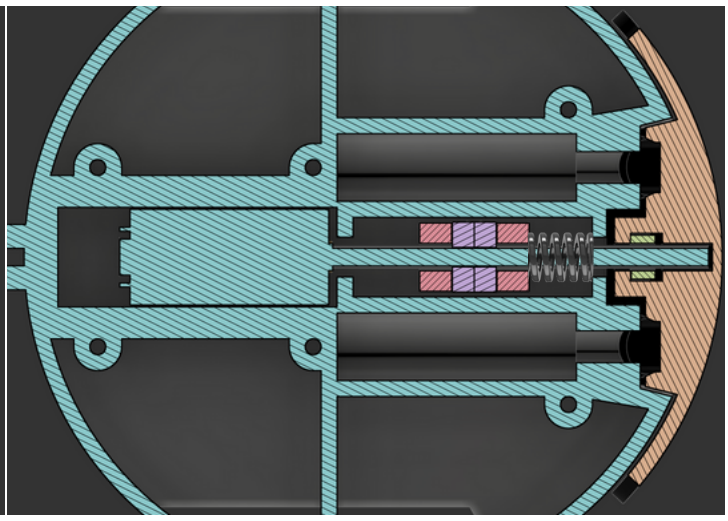
Les motoréducteurs ont une tige filetée M3 comme axe, qui vient directement se visser et plaquer la trappe contre le corps de la case parachute. Un jeu de piston à ressort vient se comprimer et permettra d'expulser la trappe de manière franche. Le parachute quand à lui est éjecté par un jeu d'élastique comme sur MS1.

Ces "mini-motoréducteurs" ont beaucoup de couple (en fonction du ratio que l'on choisi). Leur faiblesse est une force dans le sens de l'axe, de compression ou traction. Pour éviter cela quand on serre la trappe, le motoréducteurs est placé dans un logement qui lui autorise un mouvement de glissière.

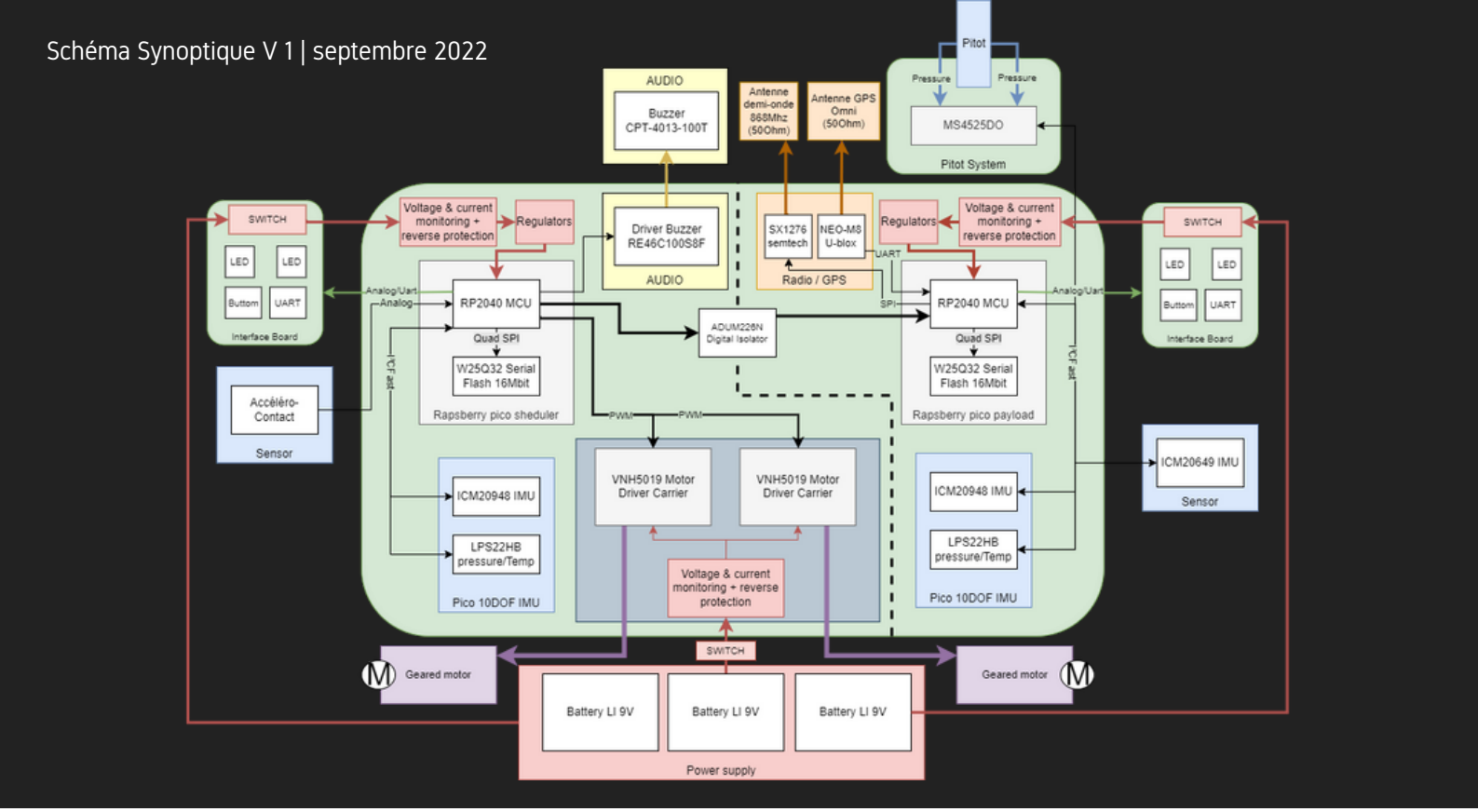
Une bague est fixée à l'axe et bloquée latéralement par des roulements, eux même gardés en position par des ressorts. Ainsi, lors du verrouillage du système, le motoréducteur va visser son axe dans la trappe et se tirer lui même vers l'avant en comprimant le ressort. La trappe sera donc solidement maintenu par la vis mais également par le ressort (on limite ainsi la traction sur l'axe du motoréducteur). Cela évite également que le moteur cale au démarrage par manque de couple car l'écrous de la trappe n'est jamais complétement serré, mais juste mis en tension par le ressort.



Système verrouillé



Système verrouillé et serré



Architecture électronique

Présentation du concept de l'avionique pour la fusex MSE

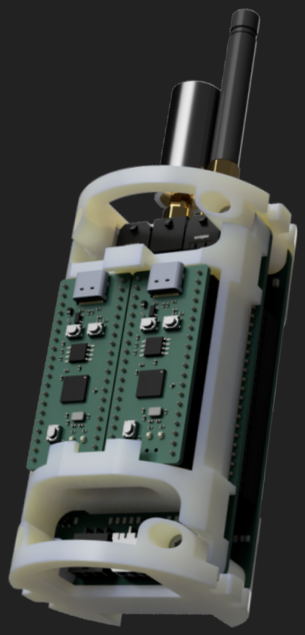
Le système avionique principal de notre fusée a été élaboré à partir de l'expérience acquise avec Marsaut 0 et Marsaut 1. Durant ces projets antérieurs, nous avions axé nos efforts sur la conception de systèmes avancés, s'alignant avec les standards industriels, ce qui a introduit une complexité et des coûts additionnels dans le design, la fabrication et l'intégration.
Notre objectif principal avec la nouvelle avionique est d'opter pour une approche simplifiée.
Notre objectif principal avec la nouvelle avionique est d'opter pour une approche simplifiée. C'est à dire concilier efficacité et simplicité, afin de développer un produit robuste et économiquement efficient, sans pour autant sacrifier la performance.

L'architecture de l'avionique est organisée en trois modules distincts :

- Le Séquenceur : Chargé de contrôler les moteurs et de détecter les étapes cruciales du vol comme le décollage et l'apogée, en s'appuyant sur l'IMU ou le baromètre. Pour garantir un suivi détaillé, il est doté d'une mémoire intégrée destinée à enregistrer des données essentielles en cas d'anomalies.
- Le Transceiver : Responsable de la gestion et de la récupération des données provenant des capteurs de suivi de vol, incluant l'IMU, le baromètre et le GPS. Il reçoit également les informations de l'état du séquenceur et les données expérimentales. Un élément clé de ce module est sa capacité à transmettre l'ensemble des données collectées en temps réel via une puce LoRa en 868 MHz, assurant ainsi une communication rapide et efficace avec la station au sol. Pour garantir un suivi détaillé, il est doté d'une mémoire intégrée destinée à enregistrer des données essentielles en cas d'anomalies.
- L'Expérience : Focalisée sur l'étude aérodynamique de la fusée, réalisée à travers une sonde Pitot totalement analogique qui comprend une chaîne de mesure et de conversion pour l'envoi des données au transceiver.

Architecture électronique

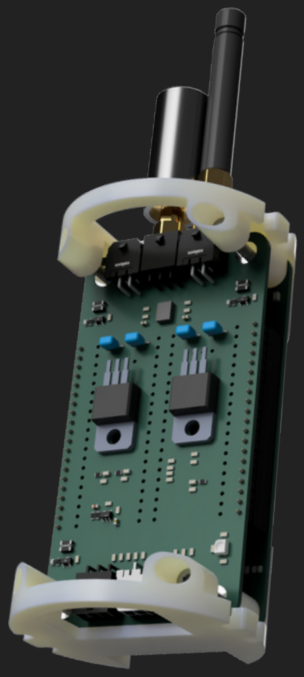
Aperçu de l'Assemblage des PCB sur l'avionique principale



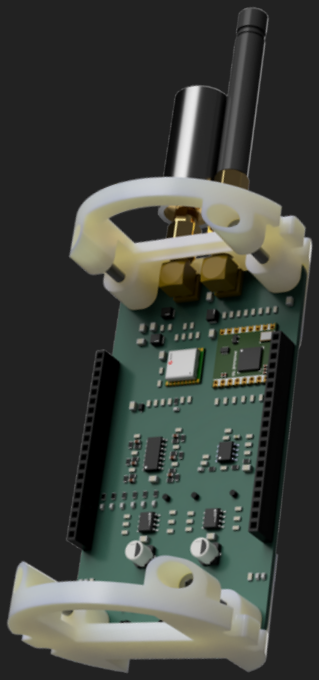
Avionique complet

Pour surmonter la limitation d'espace dans la fusée, nous avons sélectionné des Raspberry Pi Pico modifiées, équipées d'un microcontrôleur RP2040 et de 128 Mbit de mémoire Flash. Ce choix garantit une programmation et un stockage efficaces, tout en étant économique et adaptable à d'autres projets. Notre avionique se compose de deux cartes distinctes pour le séquenceur et le transceiver, chacune avec une mémoire flash pour l'enregistrement des données. Une carte 10-DOF, compatible avec le format Pico, est également intégrée, permettant l'utilisation d'un IMU MPU9250 et d'un baromètre LPS22HB pour un suivi précis du vol.

Nous avons conçu un PCB d'interface unique pour héberger à la fois le séquenceur et le transceiver, séparés par une rupture du plan de masse, optimisant ainsi l'espace. L'expérience est montée sur un PCB séparé et est reliée au transceiver via un connecteur Molex Microfit. Un ADUM1286 est utilisé pour la communication électriquement isolée entre le séquenceur et le transceiver via un bus UART. L'alimentation stable est assurée par un régulateur linéaire LM340AT, minimisant le bruit. Des broches de 2,54 mm sur le PCB facilitent la connexion avec la seconde carte, une conception interne de notre club, illustrant notre aptitude à développer des solutions sur mesure.



Carte Interface



Carte avionique

Le cœur du système réside dans la carte avionique, hébergeant toutes les fonctions avioniques intégrées. Pour le transceiver, cette carte est équipée de la puce GPS MAX-M10S et de la puce LoRa RFM95W (SX1276), facilitant la réception des données GPS et la transmission des données à 868 MHz via LoRa. En ce qui concerne le séquenceur, cette section est consacrée à la gestion des motoreducteurs. Elle utilise deux drivers pont en H DRV8871DDARQ1 et 74HC14D,653 pour actionner les commutateurs, permettant ainsi de déterminer la position de la trappe via des boutons.

QR Code du projet Avionic-MSE
https://github.com/axpaul/Avionic-MSE

Logiciel de vol

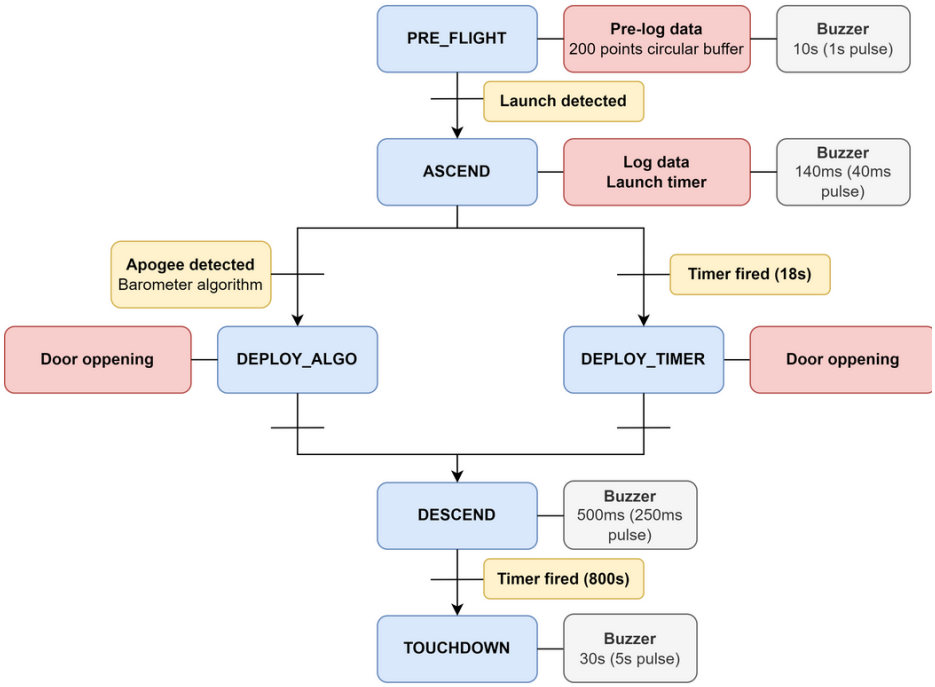
Le logiciel de vol est composé de deux parties :

- Le code séquenceur qui s'occupe de détecter le décollage par un système de contacteur activé par une masse (accéléro-contact) et d'ouvrir la trappe parachute à l'apogée au bout d'un temps prédéfini
- Le code charge utile qui s'occupe d'acquérir les données des différents capteurs et expérience (IMU, sonde pitot, baromètre, GNSS) et d'enregistrer ces données sur la flash embarquée et d'envoyer une partie de ces données par la télémétrie LORA à 2Hz

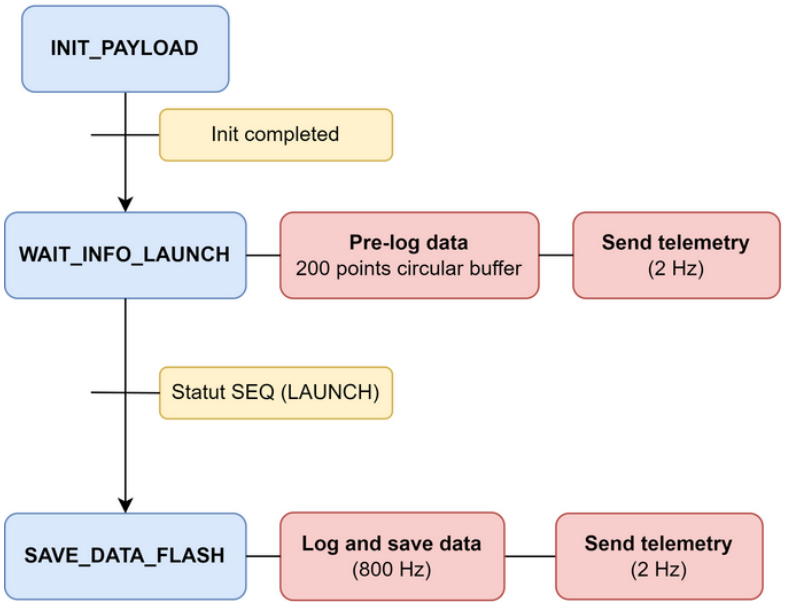
Déroulement du code du séquenceur

Le séquenceur a 7 états permettant de suivre l'état de la fusée :

- L'état pre-flight permet d'initialiser les capteurs ainsi que de manœuvrer la trappe parachute
- L'état pyro ready permet de déclencher l'enregistrement des caméras et de dévérrouiller l'acquisition de l'accéléroo-contact
- Une fois le décollage détecté, le séquenceur passe en état ASCEND qui permet d'enregistrer les données du vol et de détecter l'apogée
- A l'apogée ou à la fin du temps calculé de l'apogée, le séquenceur ouvre la trappe parachute
- Finalement les deux états suivant permettent de suivre la fusée pendant la descente et après l'atterrissage en changeant le tempo du buzzer



Déroulement du code de la charge utile



La charge utile a 3 états :

- L'initialisation permet de configurer tous les capteurs ainsi que le GNSS et la télémétrie
- La charge utile attend ensuite le décollage détecté par le séquence (communication par optocoupleur)
- Quand le décollage est détecté, les données de tous les capteurs sont enregistrés sur la flash intégrée à la carte Pico

Le débit de données est d'environ 256kbits/s pour une fréquence de 800Hz
Le débit de la télémétrie est d'environ 0.5kbits/s pour une fréquence des données de 2Hz

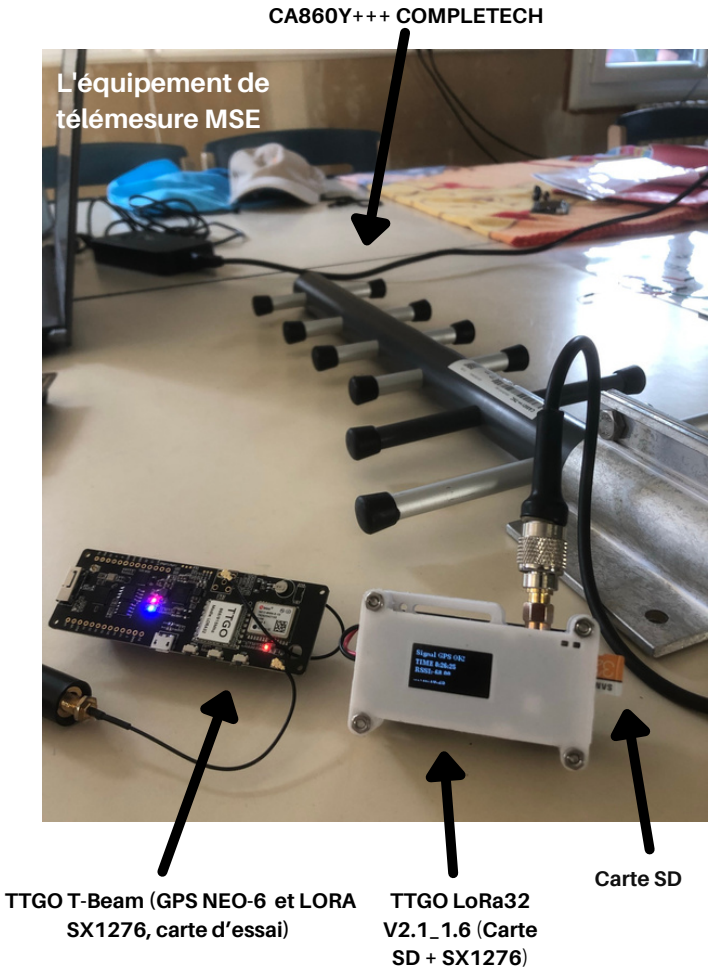
Lien du code du séquenceur
https://github.com/lou270/mse-seq

Lien du code de la charge utile
https://github.com/lou270/mse-pld

Station sol & émetteur

Pour pouvoir récupérer l'ensemble des informations transmises par la fusée lors du vol, nous utiliserons la station sol conçu par le club.
Elle se décompose en deux parties :

- Une partie réception (antenne+microcontrôleur)
- Une partie IHM (Interface graphique sur un PC reliée en liaison série).



Partie réception :
Le module se décompose en deux éléments, l'antenne avec une puce de Semtech SX1276 pour protocole Lora et un microcontrôleur du type ESP en temps réel. Directement relié à un ordinateur, le module décodera les informations reçues et transmettra les données via un protocole serial Virtual COMPORT.
Pour avoir un bilan liaison convenable, nous privilégierons une antenne Yagi de CompleTech, le modèle CA860Y+++, ayant un gain de 11Db, et une bande de fréquences variant de 830 à 890 mHz, parfait pour nos applications actuelles et futures.

Partie IHM :
Notre interface graphique est conçue à partir du langage de programmation C++ en utilisant le framework robuste et polyvalent Qt. Cette combinaison assure une performance optimale et une grande flexibilité dans le développement de notre interface utilisateur.
Un des atouts majeurs de notre logiciel est sa capacité à traiter et afficher les trames de données reçues de la fusée en temps réel. Cela inclut une variété d'informations essentielles, avec une attention particulière portée aux données de localisation.

Pour améliorer l'efficacité de notre suivi en temps réel, nous avons intégré la bibliothèque Leaflet, un choix stratégique qui nous permet d'incorporer et de manipuler des cartes OpenStreetMap au sein de notre interface. Cette bibliothèque, connue pour sa légèreté et son efficacité, est utilisée pour afficher dynamiquement le dernier point GPS reçu de la fusée sur une carte OpenStreetMap, offrant ainsi une visualisation intuitive et précise de la localisation de la fusée.

RocketGroundStation

Lien du projet RocketGroundStation
https://github.com/axpaul/RocketGroundStation

Déroulé du vol

La qualification de MSE a été retardé par un problème de centre de gravité trop bas suite à notre choix de déplacer les batteries sous la bague moteur. Nous avons donc eu l'autorisation de tirer le jeudi après midi. Le ciel était couvert comme on peut le voir sur l'image ci-dessus, avec un plafond nuageux très bas (environ 900m).

Une fois la fusée mise en rampe, les pyrotechniciens ont pu installer le pro54 K570 (le K1200 n'étant pas arrivé à temps) et retirer la dernière flamme de la fusée. Cette flamme permet d'indiquer à la fusée que la prochaine étape sera le décollage. Elle va donc désinhiber l'accélérocontact (jusqu'ici désactivé pour éviter un faux positif), allumer l'enregistrement des caméras et activer l'enregistrement des capteurs à haute fréquence.

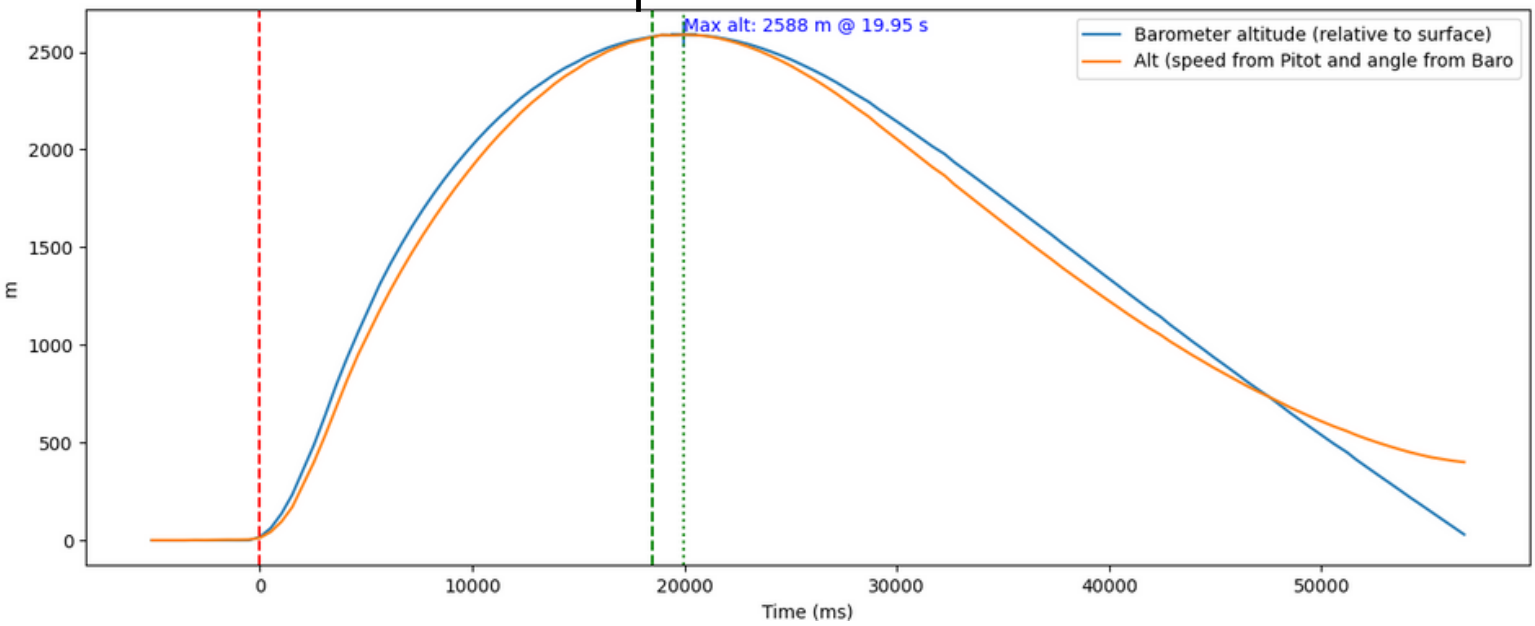
Quelques minutes plus tard, MSE décolle avec une trajectoire bien droite dans l'axe de la rampe et atteint les nuages en 7 secondes. A peu près au même moment, nous pouvons entendre au sol le bruit typique d'une ouverture parachute, alors qu'elle devrait avoir lieu 12 secondes plus tard. Les informations reçu pendant tout le vol nous permettent rapidement de déterminer que le vol sera balistique. Nous perdons le signal radio 56 secondes après le décollage, à une altitude de 90m au dessus du sol et une vitesse de 75m/s.

Bien qu'équipée d'un GPS, nous n'avons pas réussi à retrouver la fusée dans la zone présumée, trop accidentée et difficile d'accès.

Etude des données

N'ayant pas retrouvé la fusée, nous n'avons pas pu récupérer les enregistrements de capteur haute vitesse ainsi que les images des deux caméras. Nous ne pouvons donc utiliser que les données reçu par la radio à une fréquence de 2Hz.

Altitude en fct du temps



Avec le baromètre :
En connaissant les paramètres météo de la journée, nous pouvons facilement convertir la pression mesurée par notre baromètre en altitude. Ce qui donne 2588m au dessus du sol à l'apogée.

Avec le baromètre et la sonde pitot :
Nous n'avions malheureusement pas assez de débit sur la radio pour envoyer les mesures des gyroscopes, nous nous retrouvons donc sans aucune données nous permettons d'avoir l'angle de la fusée. Il est cependant possible de poser le postulat suivant : la fusée ne possédant pas de système actif de contrôle de trajectoire, seuls les ailerons agissent sur l'angle. On peut donc conclure que ce dernier est tangent à la courbe de l'altitude fournie par le baromètre (sauf pour les 5 première secondes où l'angle est fixé sur celui de la rampe). En couplant cet angle et la vitesse de la pitot, on obtient une deuxième courbe (orange) montrant l'évolution de l'altitude. Cette méthode nous donne la même altitude que la première à 1 mètre près.

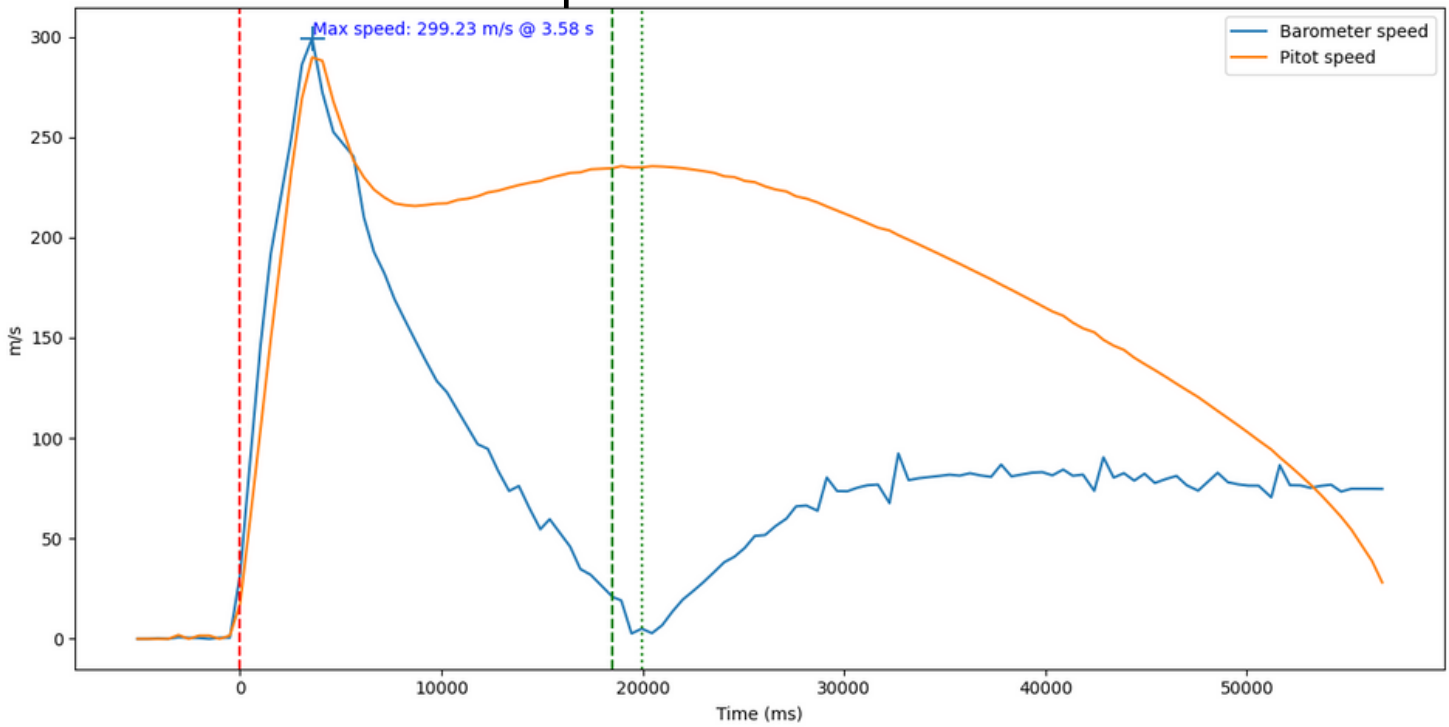
Au final ces courbes montrent que la fusée a eu une apogée autour de 20 secondes ce qui correspond aux simulations effectuées en amont. Le temps de vol est plus court que prévu avec seulement 56 secondes au lieu de 245 secondes pour un vol nominal. En revanche la durée théorique du vol balistique n'est que de 47 secondes, soit 9 secondes de moins que le vol réel.

La vitesse maximum atteinte est d'environ 300 m/s (1080 km/h) soit mach 0.87, mesurée par le baromètre en dérivant la pression et par la sonde pitot. La mesure pitot seule dévie de la mesure de la vitesse par baromètre après le pic de vitesse surement du à l'orientation de la fusée durant la suite du vol.
L'apogée se retrouve aussi à 20 secondes avec une vitesse nulle suivie d'une accélération constante pour au final maintenir un vitesse durant la fin du vol aux alentours de 80 m/s (288 km/h).

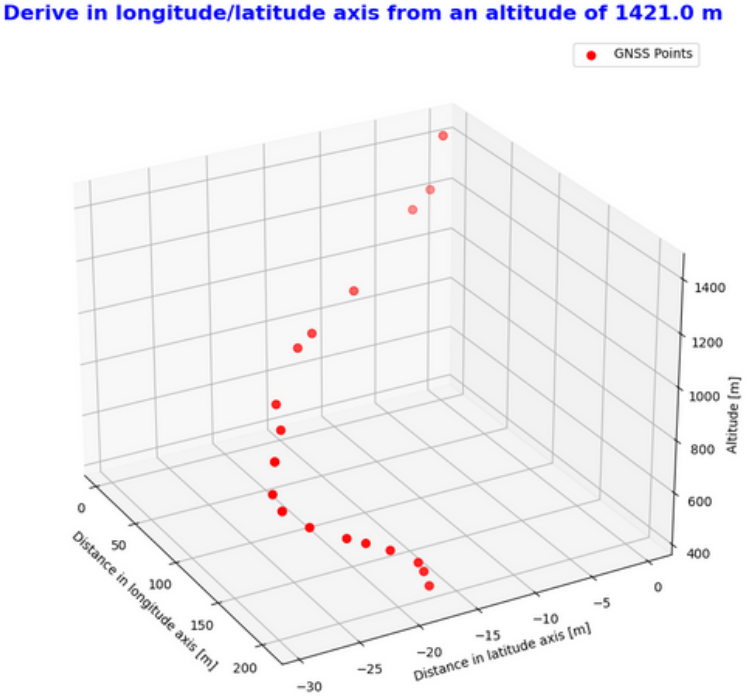
Lien du code d'analyse des données de télémétrie
https://github.com/lou270/mse-tm-utils

Etude des données

Vitesse en fct du temps

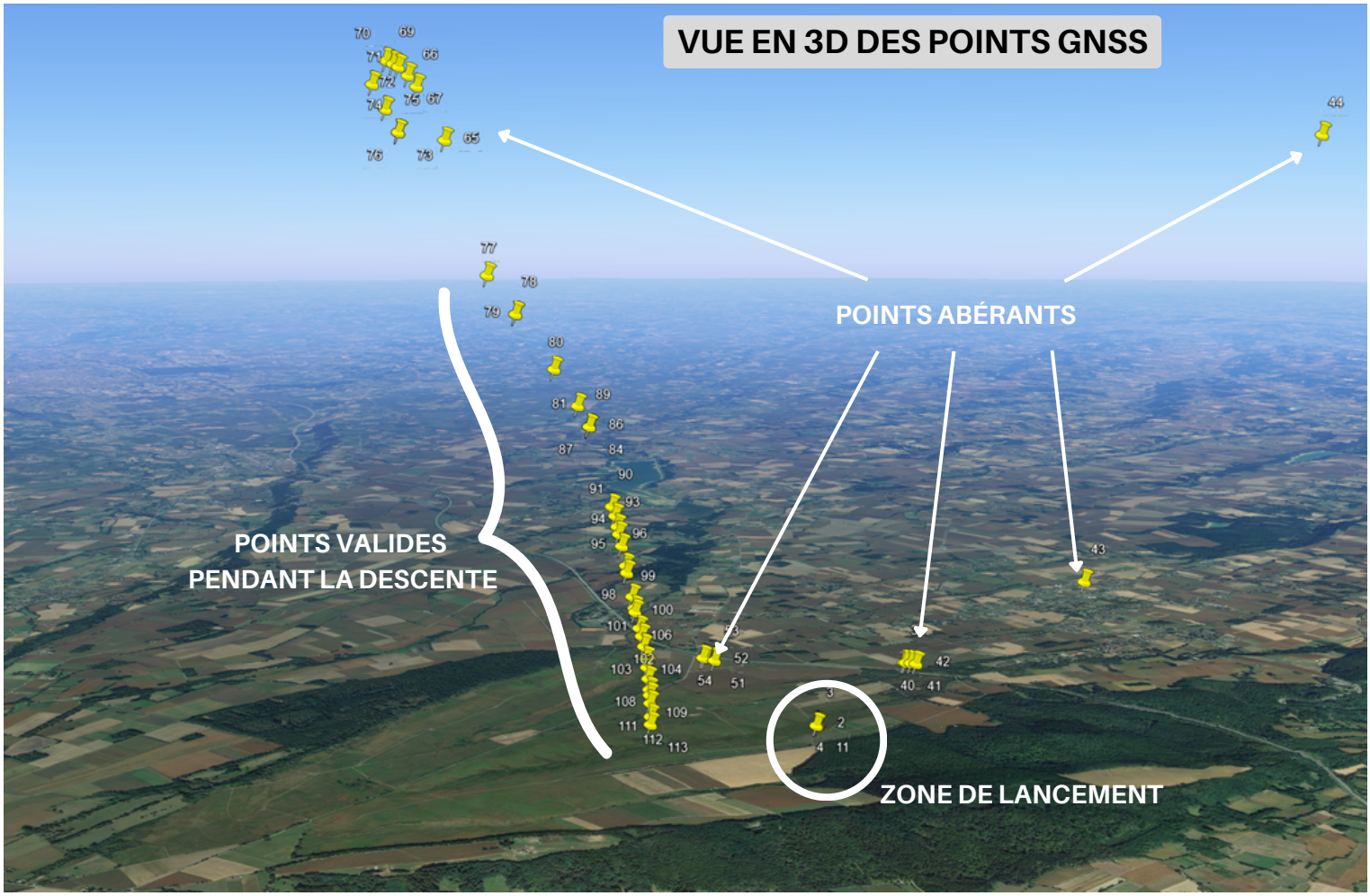


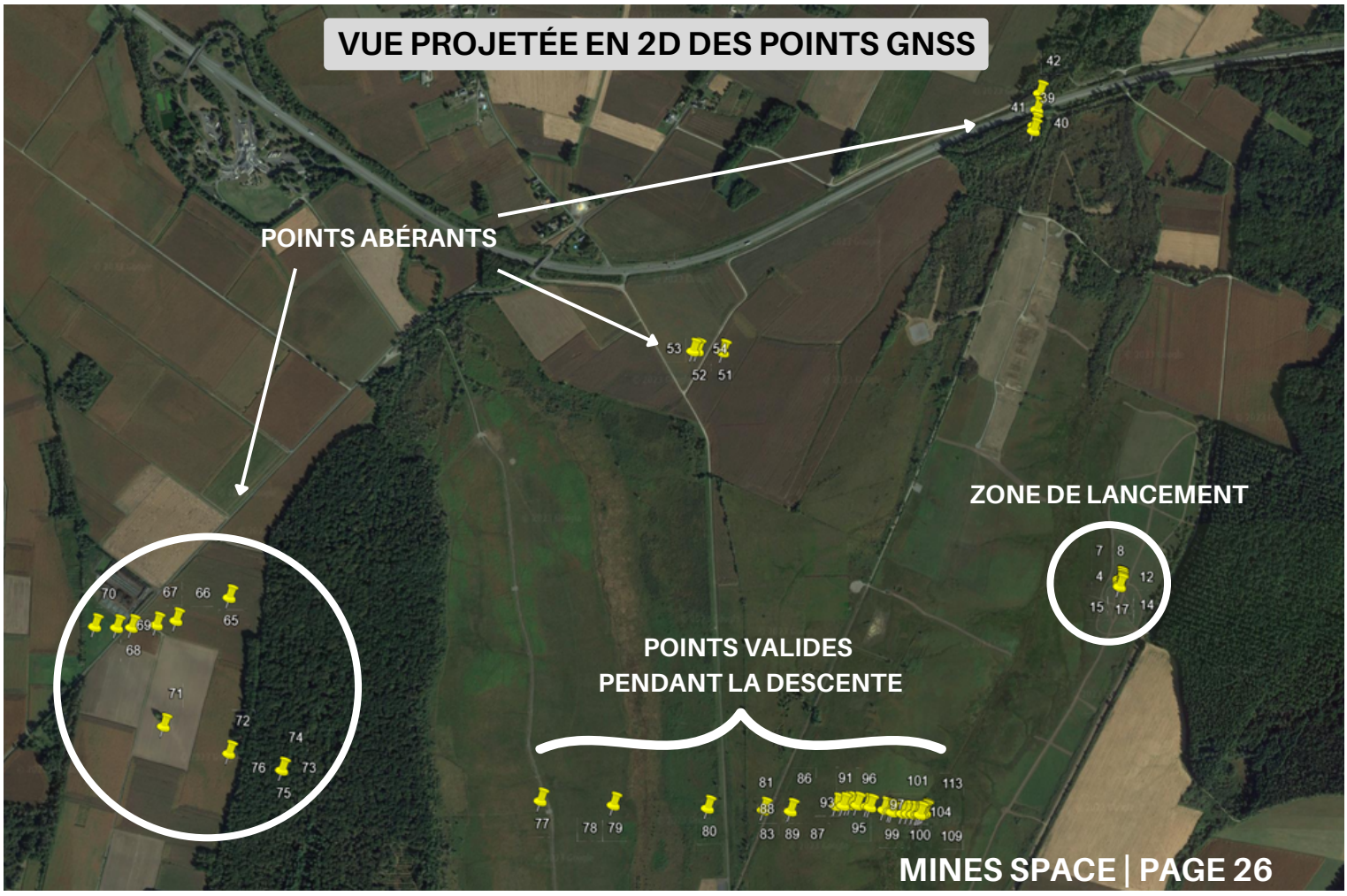
Le GNSS a perdu le fix satellite après le décollage car l'accélération était trop importante pour le composant choisi. Par contre le retour du fix satellite a été assez rapide pour suivre la descente de la fusée.



En traçant les données GNSS à la descente (quand la fusée est en dessous de 1500m relatif au sol), nous pouvons voir que sa trajectoire n'est pas parabolique comme un vol balistique pur aurait pu être reconnu. Il semble que la fusée ait dérivée plusieurs dizaines de seconde avant de toucher le sol.
Ces données GNSS en plus des données de vitesse par baromètre semble indiqué une descente au pire semi-balistique, avec quelque chose qui aurait freiné la fusée. Malheureusement aucune hypothèse ne peut être confirmée ou infirmée sans les données et vidéos bord de la fusée que nous espérons retrouver un jour ...

Etude des données





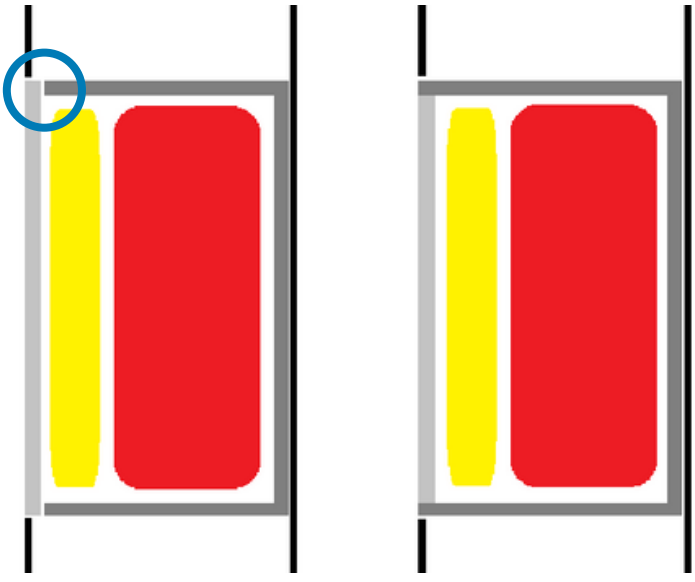
Conclusion

Malgré les tentatives de recherches et toutes les données récupérées, nous n'avons pas encore retrouvé MSE. Cela rend la compréhension du vol particulièrement difficile : les images des caméras, dont une avec une vue direct sur la trappe parachute nous donnerait toute les informations nécessaires. Pour le moment, nous ne pouvons émettre que quelques hypothèse en fonction de ce qui a été entendu au sol.

Les peaux en carbone et leur forme n'assure pas une bonne étanchéité, de plus, un jeu avait été laissé entre la case parachute et la peau qui a pu permettra à un courant d'air de s'infiltrer dans la case parachute et gonfler celui-ci directement à l'intérieur de la fusée. Cela aurait comme conséquence de briser la trappe en deux et de possiblement déformer les tiges filetées et d'empêcher le mécanisme de fonctionner. L'analyse des données à montré que le bruit entendue au sol à t+7s a en réalité eu lieu à t+2.5s, soit le moment où la vitesse est la plus élevée sur tout le vol. On parle alors ici d'une force de plusieurs milliers de Newton sur le parachute. Ce jeu n'est pas existant sur les fusées précédentes mais a été introduit suite à l'incertitude de l'épaisseur finale de la peau en carbone.

Le schéma ci contre représente deux cases parachute en vue de coté. Celle de gauche représente celle utilisée sur MSE. Le rond bleu montre le jeu par lequel le courant d'air a pu passer et venir gonfler le premier parachute (zone jaune). En effet, il est possible que seul le premier parachute ait "explosé" à l'intérieur de la case parachute et que le système ait été trop obstrué ou abimé pour éjecter le parachute principal (zone rouge).

Sur la version corrigée à droite, on peut imaginer qu'il soit bien moins probable que cela arrive.



Chiffres

305m/s
mach 0.89 | 1100km/h

2588m
Par rapport au sol

56s

940m

1150mm

∅ 90mm

3,3kg
Sans propulseur

Nombre de composants Hors visserie, électronique et charge utile	MS0	MS1	MSE
	140	90	50

Ce projet vient conclure notre activité de membre de l'association Mines Space et ces 3 ans de CSpace. MSE est le fruit du retour d'expérience de nos deux lancements précédent et vient poser une base technique solide pour les prochains projet de l'association et de ces membres. Aussi bien pour les éléments mécaniques, électroniques et logiciels.

Remerciements

MINES SPACE

Planète Sciences – une aventure pour les jeunes

CompleTech – ComAnt® -antennas – by CompleTech, Finland

RS RADIOSPARES

ALTIUM DESIGNER

MINES Saint-Étienne

MBDA MISSILE SYSTEMS

cnes CENTRE NATIONAL D'ÉTUDES SPATIALES

easycomposites

CONTACTS

Qui contacter ?

Par mail

Adresse mail du club : **minesspace.asso@gmail.com**
Site Internet du club : **https://minesspace.fr**

Adresse mail des membres projets :
Etudiant : - paulboymond@gmail.com
- paul.miailhe@gmail.com
- louis.barbier27@gmail.com

Par les réseaux sociaux

Mines Space

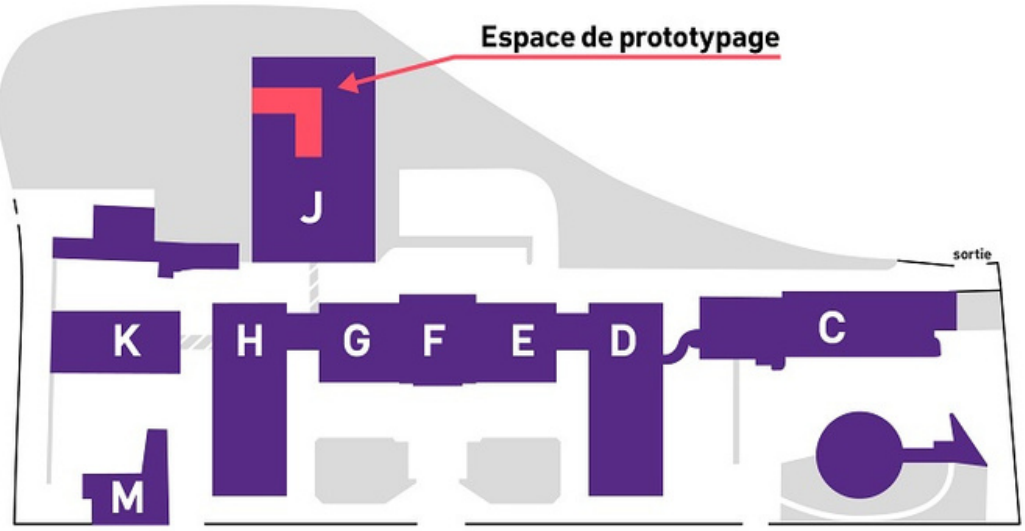
mines_space

@MinesSpace

mines-space

Où nous trouver ?

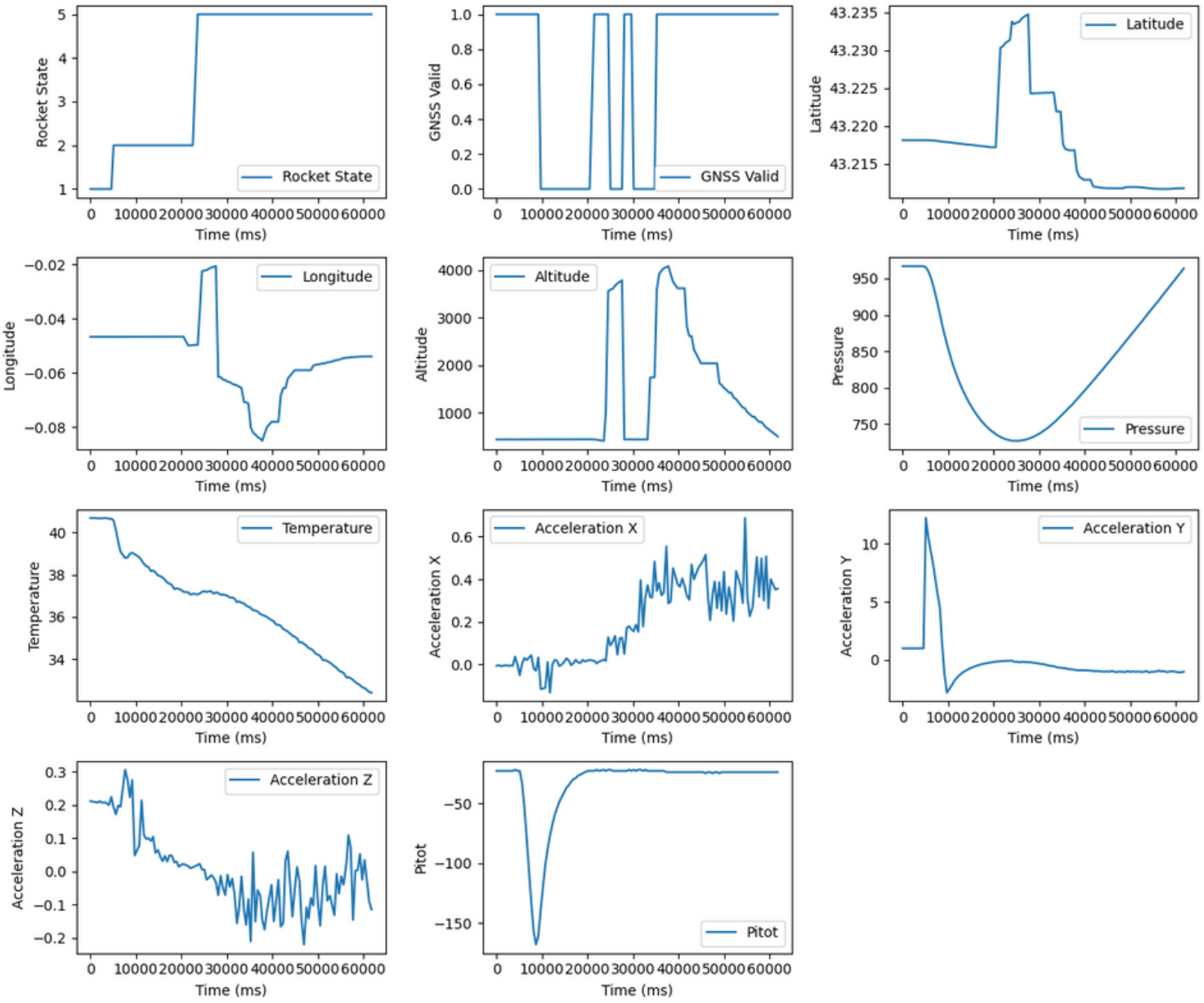
Notre quartier général se trouve actuellement au FabLab de l'Ecole des Mines de Saint-Étienne (Cours Fauriel). C'est ici que l'on se retrouve pour organiser des réunions.



- 158 Cours Fauriel -

Annexe
Données brutes & Données de stabilité CNES

MSE
MACH 1.1



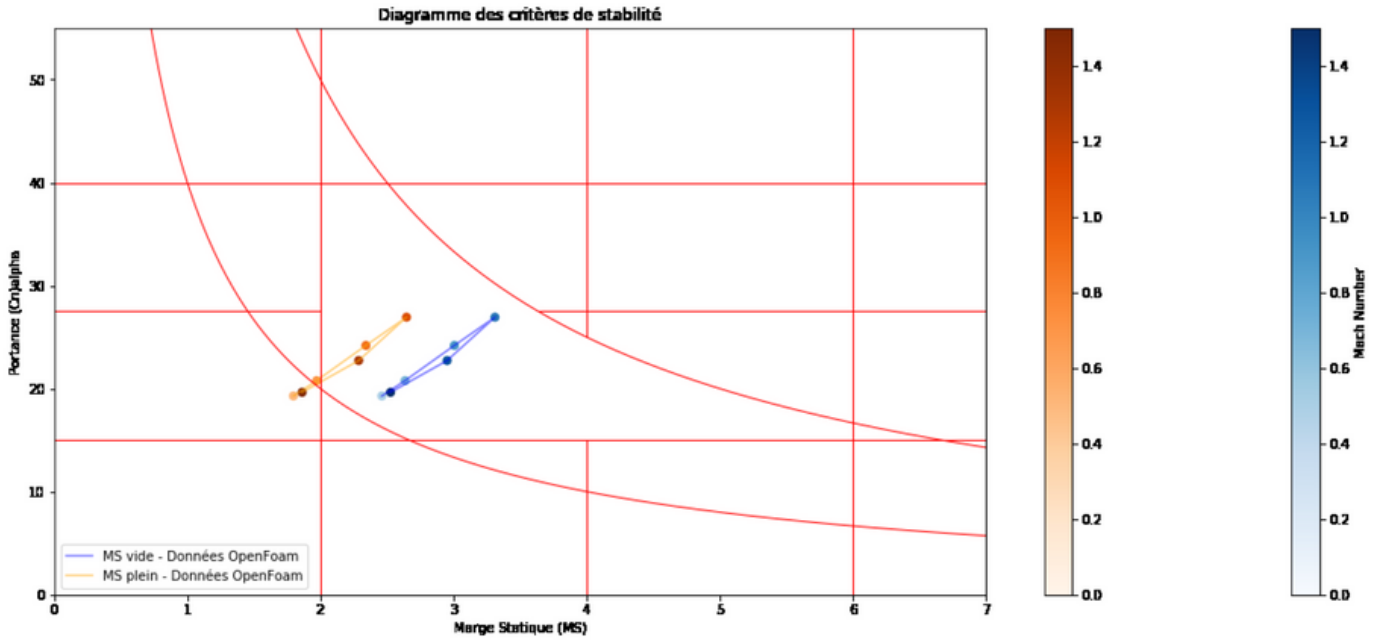


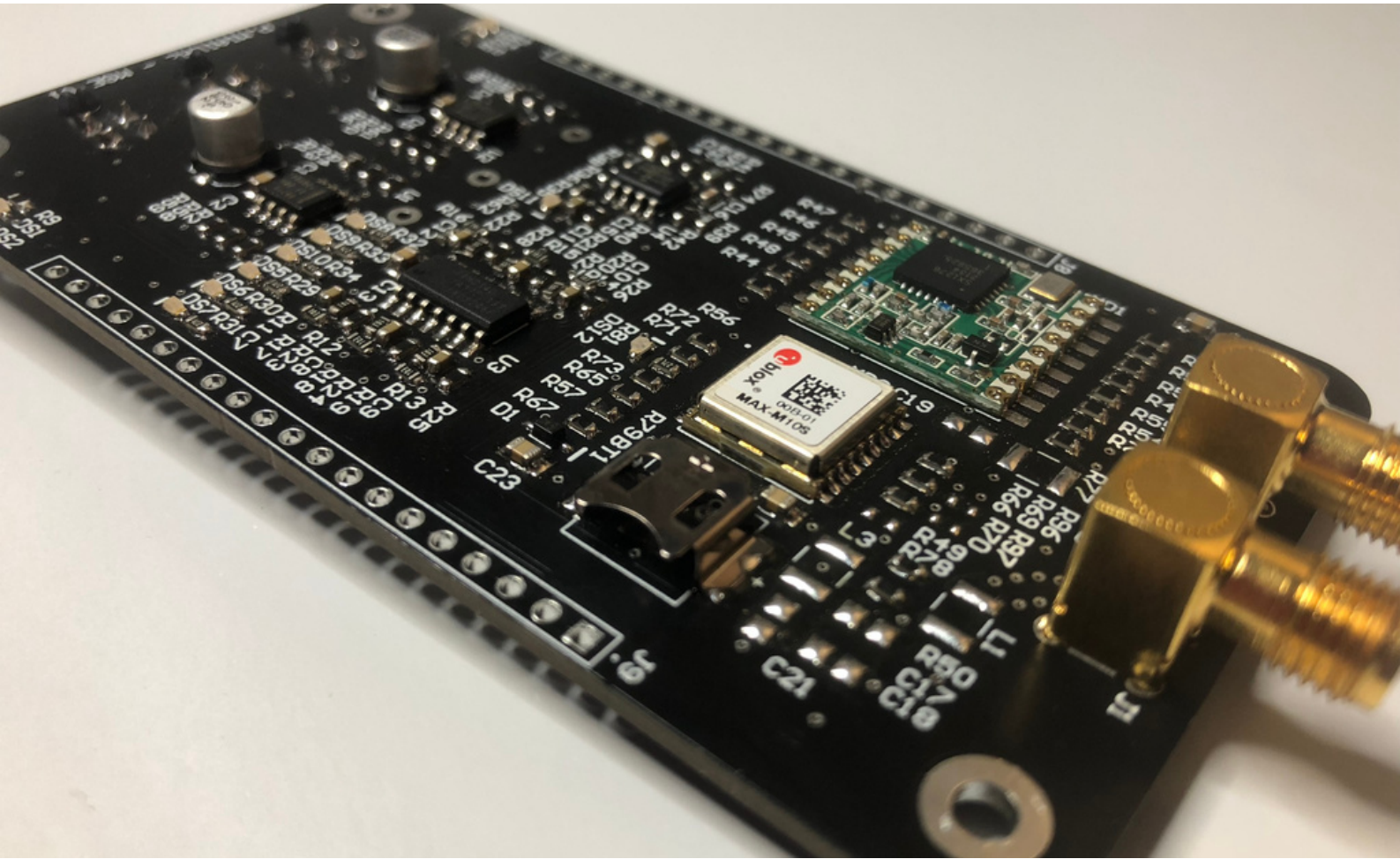
Figure 1 : Diagramme des critères de stabilité – FX37

Annexe - Photo

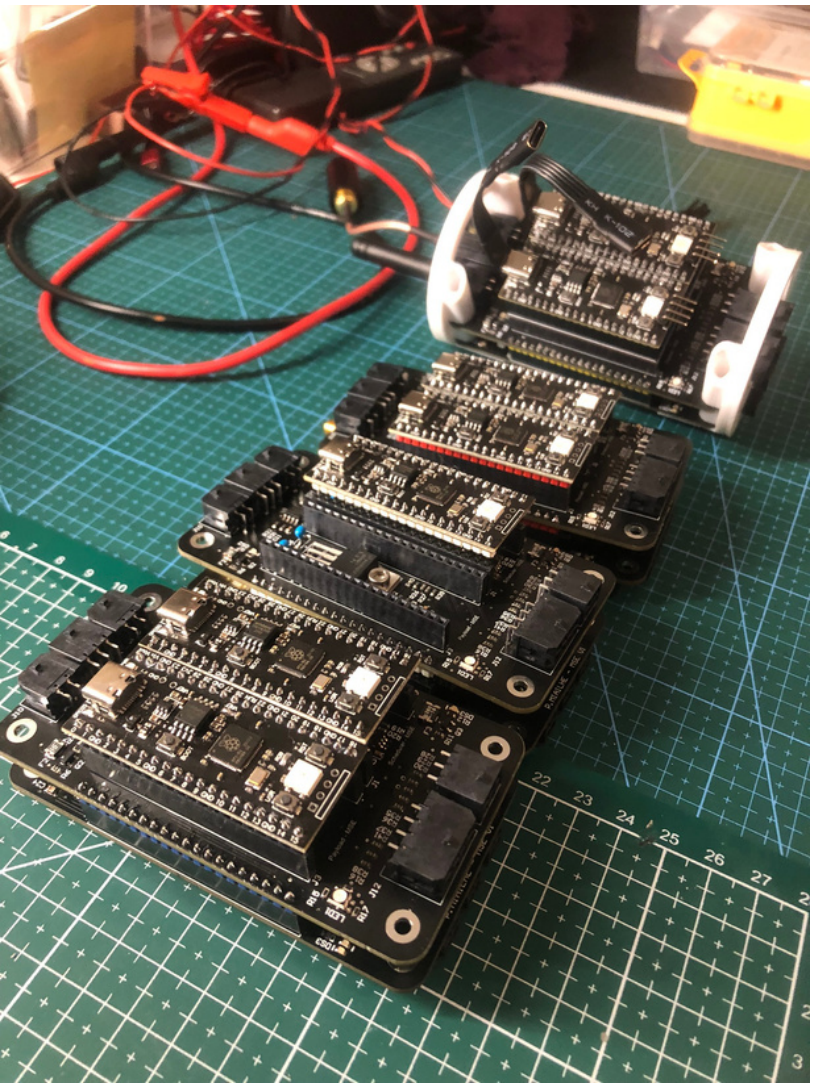
MSE
MACH 1.1

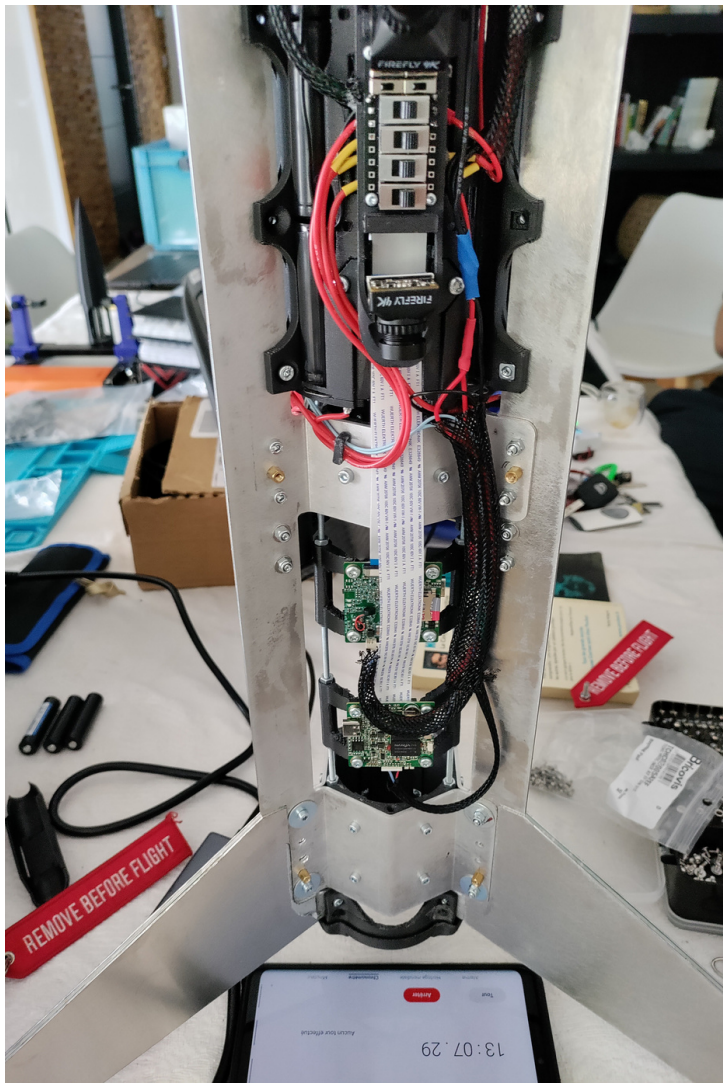

REMOVE BEFORE FLIGHT


REMOVE BEFORE FLIGHT
MINES SPACE
©Charles Pilon


ublox
MAX-M10S

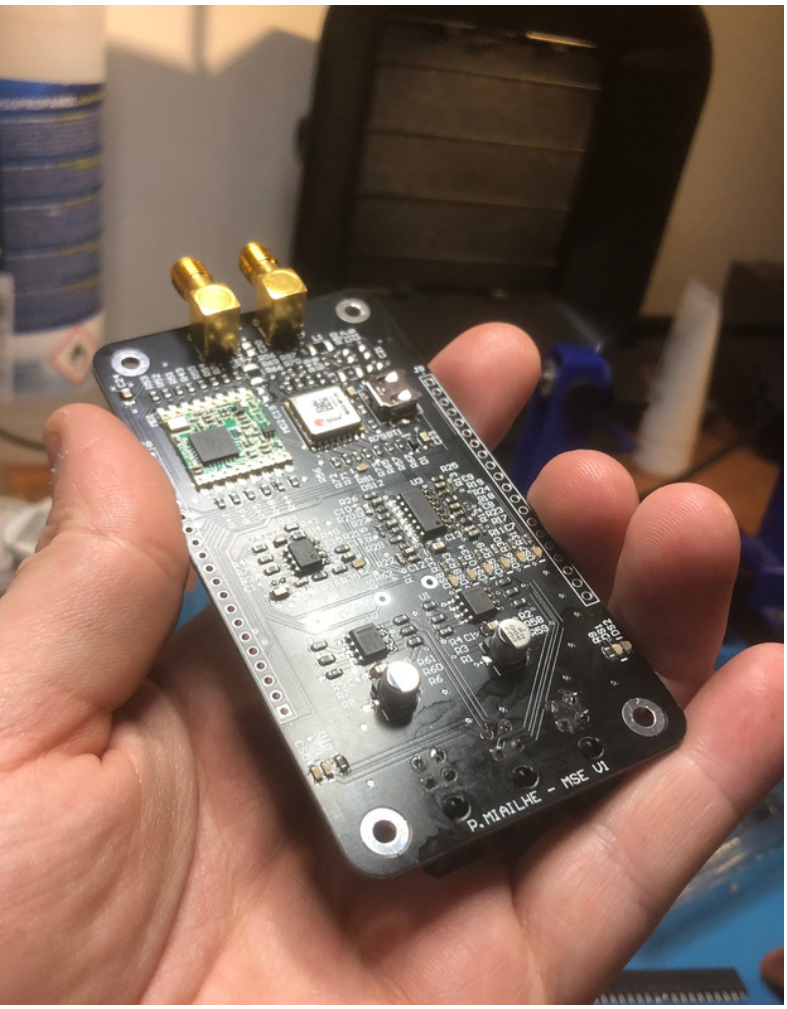

esa




FIREFLY 9K
REMOVE BEFORE FLIGHT
13:07.29


REMOVE BEFORE FLIGHT


Enzo
PICAS
Louis
BARBIER


P.MAILHE - MSE U1

Annexe - Rapport

MINES SPACE

ÉTUDE DE LA TRAJECTOIRE ET DE LA STABILITÉ DE LA FUSEX MSE

VERSION N°01

2023

CRÉATION D'UNE FUSÉE EXPÉRIMENTALE DANS LE CADRE DE C'SPACE

PAUL MIAILHE

MSE
MACH 1.1

MINES SPACE 2023

OBJECTIFS

Le projet MarSoniquE (MSE, également connu sous le nom de Marsaut E), a pour ambition de mettre en place une plateforme fiable pour mener à bien des études dans le domaine supersonique. Jusqu'à présent, lors des événements du C'Space organisés sur le site d'essai du 1er RHP, aucune fusée expérimentale n'a réussi à réaliser un vol nominal présentant de telles caractéristiques.
En partageant notre approche et nos résultats via une démarche Open Source, nous aspirons à définir les bases d'un modèle de projet permettant de surmonter ce défi. Notre méthodologie est structurée autour de trois axes principaux :

- **Génération et exploitation de données aérodynamiques :**

Nous décrirons les procédures pour acquérir des données aérodynamiques pertinentes en lien avec la conception d'une fusée supersonique. Nous aborderons également les méthodes permettant d'utiliser ces données efficacement pour l'analyse et l'optimisation de la stabilité et de la trajectoire de la fusée.

- **Élaboration d'un script d'analyse :**

Nous mettrons à disposition un script détaillé qui servira à analyser la stabilité et la trajectoire de la fusée à partir des données aérodynamiques générées. Ce script représente un outil précieux pour évaluer et améliorer les performances de la fusée.

- **Construction d'une architecture mécanique et électronique robuste :**

Nous discuterons des considérations clés pour concevoir une architecture mécanique et électronique résiliente face aux contraintes du vol supersonique. Cela comprendra des aspects tels que la résistance structurelle, le système de récupération et le système électronique de la fusée.

En somme, le projet MSE a un objectif bien défini : créer la première plateforme supersonique fiable (Mach 1.1) destinée à des études à basse altitude à l'aide d'une fusée expérimentale.

CONTEXTE

Dans ce document, nous entreprendrons une première évaluation de la stabilité dynamique d'une fusée supersonique à l'aide du logiciel Aerolab.
Nous commençons par examiner les facteurs clés qui influencent la stabilité de la fusée pendant son vol.
Ensuite, nous décrirons la conception d'un simulateur de trajectoire oblique, qui permet de prédire la trajectoire de la fusée en se basant sur le modèle aérodynamique que nous aurons précédemment étudié.

L'objectif principal de cette étude est d'établir une première approximation des performances globales du vol de la fusée supersonique, tout en veillant à maintenir sa stabilité. Cependant, il convient de souligner que pour une analyse plus détaillée et exhaustive, des études supplémentaires seront nécessaires. Ces études futures utiliseront des simulations numériques en dynamique des fluides (CFD) à l'aide de logiciels tels que Fluent ou OpenFoam.

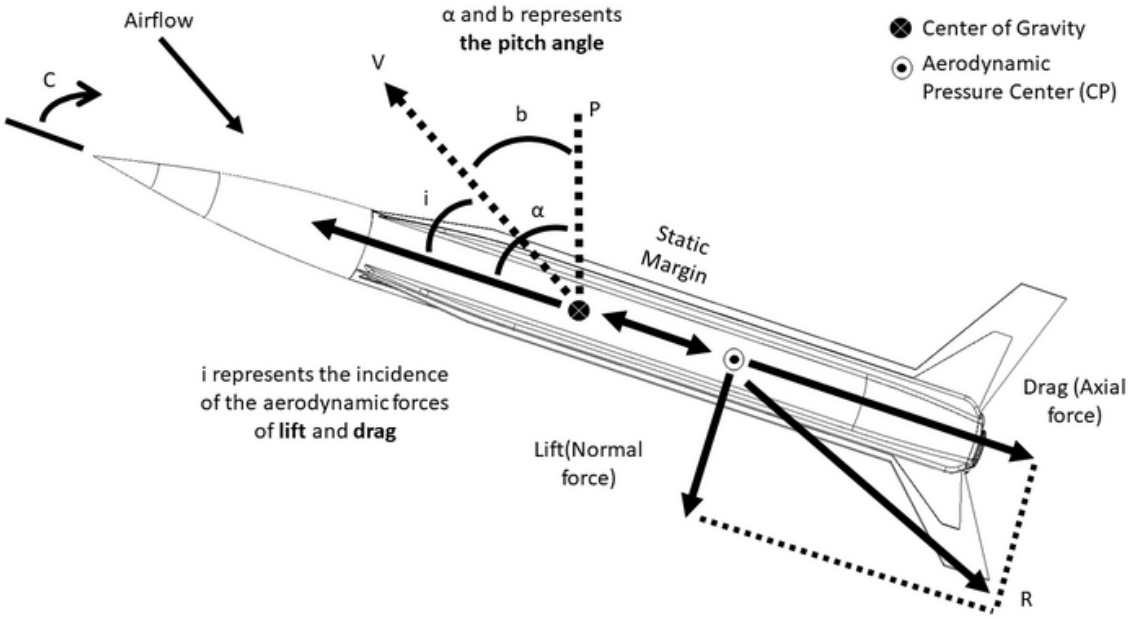
AEROLAB

Aerolab est un logiciel de simulation aérodynamique dédié spécifiquement à l'estimation de la traînée, de la portance et de la stabilité des fusées, ce à un angle d'attaque nul sur une gamme de vitesses allant de 0 à Mach 8. Il repose sur des données recueillies à partir d'expériences en soufflerie et de modèles numériques, faisant appel à des méthodes empiriques et semi-empiriques.
Le logiciel donne à l'utilisateur la possibilité de saisir les dimensions et les propriétés massiques de la fusée.

De plus, Aerolab est capable de calculer le centre de gravité ainsi que les moments d'inertie autour de l'axe de tangage/lacet et de l'axe de roulis, en supposant une symétrie de révolution pour la distribution des masses.
Il s'agit d'un outil largement utilisé pour faciliter l'obtention d'un modèle aérodynamique d'une fusée sans avoir à réaliser de nombreuses simulations de dynamique des fluides computationnelle (CFD) pour chaque vitesse et angle d'attaque de l'engin, ou à réaliser des essais en soufflerie réelle. Cependant, il est important de rappeler que ce logiciel fournit un modèle.

DEFINITION

Avant de se pencher sur les données d'Aerolab, il est crucial de comprendre les concepts clés de la stabilité en aérodynamique pour interpréter correctement les données fournies par Aerolab ou tout autre logiciel de simulation. (Voir le Vol de la Fusée, Stabilité et Trajectographie Version 2.0). Voici quelques termes clés :

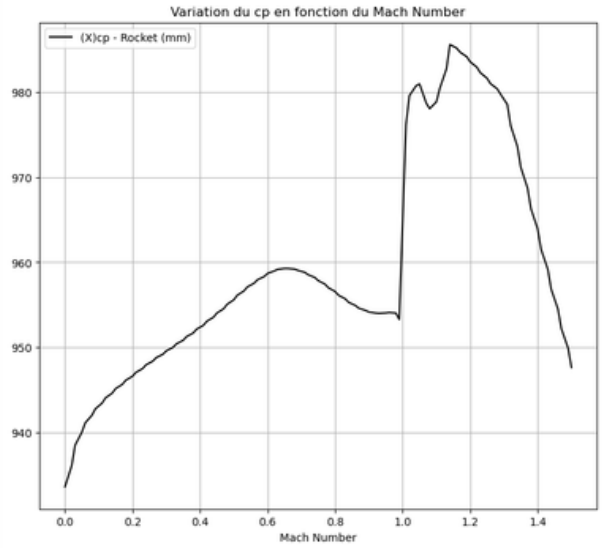


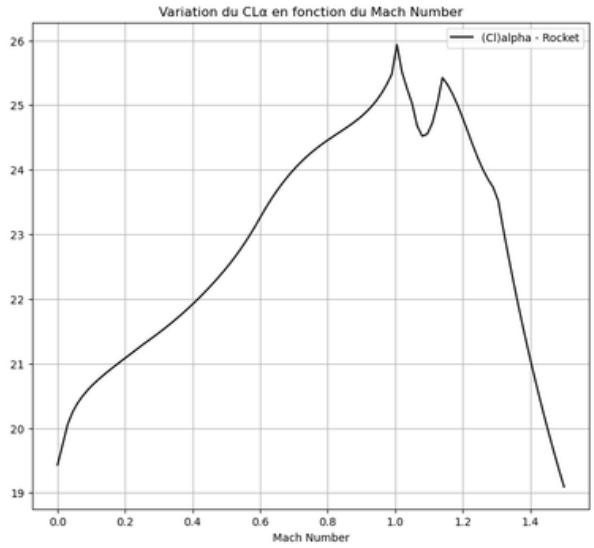
- **Centre de Gravité (CG)** : Il s'agit du point où l'ensemble du poids d'un objet (ou fusée) peut être considéré comme concentré. En d'autres termes, c'est le point d'équilibre de la fusée.
- **Centre de Pression (CP)** : Le CP est le point où la somme vectorielle de toutes les pressions aérodynamiques agissant sur la fusée est appliquée. En général, le CP se déplace le long du fuselage de la fusée en fonction de l'angle d'attaque et de la vitesse.
- **Marge statique** : La marge statique est la distance entre le CG et le CP, généralement mesurée en calibres de fuselage. Pour une fusée stable, cette marge doit être positive, c'est-à-dire que le CP doit être en arrière du CG.
- **Angle d'attaque (α) :** C'est l'angle entre le vecteur de vitesse de la fusée et la ligne de référence du fuselage de la fusée.
- **Coefficient de portance (CL) et Coefficient de traînée (CD)** : Ce sont des coefficients adimensionnels qui représentent la portance et la traînée générées par une fusée, respectivement, en fonction des conditions d'écoulement. Ils sont souvent présentés comme des fonctions de l'angle d'attaque.
- **Coefficient de pente de la courbe de portance (CLα)** : Le Clα ou encore le Gradient de portance caractérise la relation entre l'angle d'attaque et le coefficient de portance. Un CLα élevé indique que la portance augmentera rapidement avec l'angle d'attaque.

STABILITE

LES DEUX VARIABLES CLES DE LA STABILITE

Il y a deux variables clés a acquérir pour pouvoir calculer la stabilité en fonction de la vélocité de notre véhicule, le centre de Pression (CP) et le coefficient de pente de la courbe de portance ou gradient de portance (CLα) en fonction du nombre de Mach.





- Variation du Centre de Pression (CP) en fonction du nombre de Mach : Ce graphique démontre la façon dont la position du CP varie en fonction du nombre de Mach, impactant directement la stabilité de la fusée. L'analyse de cette tendance vous permet de déterminer la position du CP par rapport au Centre de Gravité (CG) à différentes vitesses de vol. Pour assurer une stabilité optimale, le CP doit toujours être positionné derrière le CG à tous les nombres de Mach, créant ainsi un moment de rappel qui préserve l'orientation de la fusée.

- Le gradient de portance (CLα) en fonction du nombre de Mach : Ce graphique met en lumière la sensibilité de la force de portance face aux changements d'angle d'attaque à différents nombres de Mach. Un CLα plus élevé signifie une variation plus importante de la force de portance par rapport aux variations de l'angle d'attaque. La force de portance générée par la fusée influe directement sur la position du CP et, par conséquent, sur la stabilité de la fusée. En comprenant comment le CLα évolue en fonction du nombre de Mach, nous pouvons évaluer les caractéristiques de stabilité et de contrôle de la fusée tout au long de son domaine de vol.

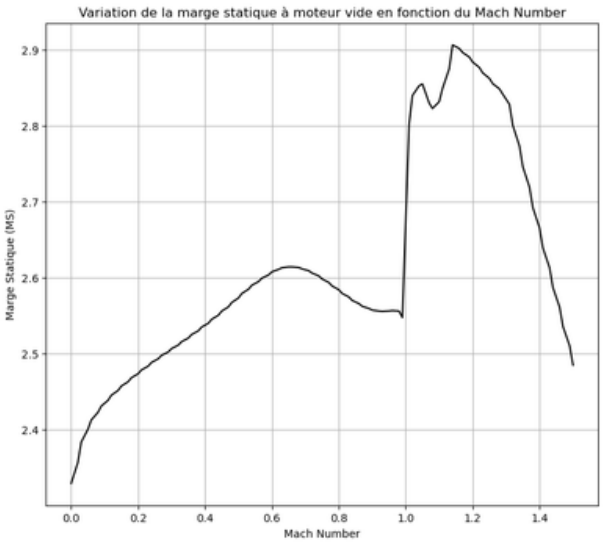
STABILITE

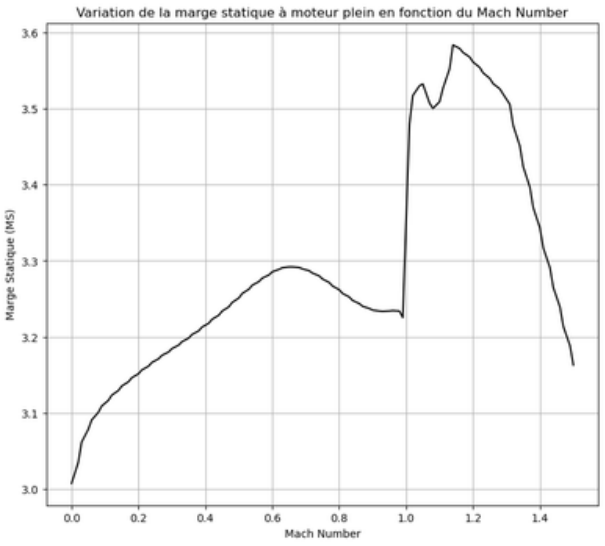
MARGE STATIQUE

La marge statique est un indicateur de la stabilité aérodynamique d'un véhicule, comme une fusée. Elle est définie comme la distance entre le Centre de Pression (CP) et le Centre de Gravité (CG) le long de l'axe longitudinal du véhicule, généralement exprimée en pourcentage ou en multiple du diamètre du corps de la fusée.

- **Marge statique = (Position du CP - Position du CG) / Diamètre du corps**

Une marge statique positive indique que le CP est situé derrière le CG, ce qui engendre une configuration stable. En cas de perturbations de l'angle d'attaque, cette configuration génère un moment de rappel ramenant la fusée à son orientation d'origine. Plus la marge statique est élevée, plus la fusée sera stable. Cependant, une grande marge statique peut engendrer un régime surstable provoquant des vibrations lors du vol de la fusée. En revanche, une marge statique négative se produit lorsque le CP se trouve devant le CG, menant à une configuration instable. Dans cette situation, toute perturbation de l'angle d'attaque sera amplifiée par les forces aérodynamiques, rendant la fusée difficile, voire impossible, à avoir une trajectoire parabolique.
Au cours de la conception d'une fusée, il est crucial de maintenir une marge statique positive tout au long du vol pour assurer des caractéristiques de vol stables. Dans notre cas, notre fusée rencontrera plusieurs régimes d'écoulement, provoquant un déplacement significatif du centre de pression et donc de la marge statique. Le logiciel Stabtraj n'effectuant pas cette analyse, nous prendrons ce facteur en compte dans notre évaluation des critères de stabilité. (Voir "Introduction to Flight" de John D. Anderson)



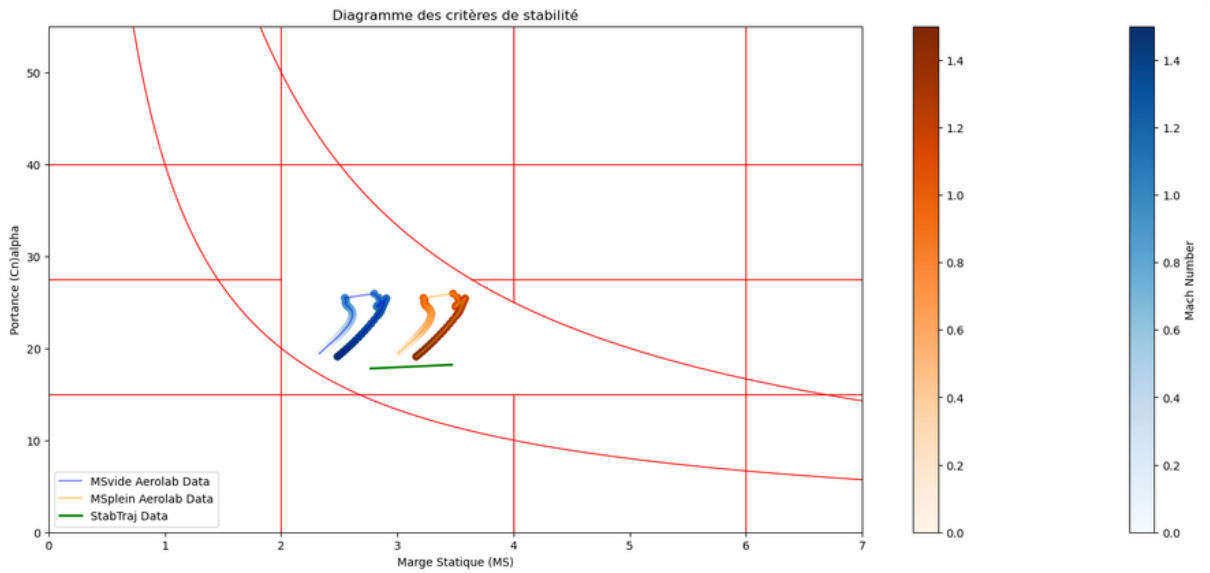


STABILITE

LE DIAGRAMME DES CRITÈRES DE STABILITE

Pour évaluer correctement la stabilité d'une fusée, il est indispensable de comprendre le diagramme des critères de stabilité. Ce diagramme est un outil essentiel pour examiner la stabilité des fusées. Il met en relation deux paramètres clés : la **marge statique (MS)**, et le **gradient de portance (CLα)**. Ce diagramme aide à visualiser la stabilité dynamique de la fusée, où les régions du diagramme peuvent être classées comme stables, instables ou substable. En général, une marge statique positive et un gradient de portance positif sont souhaités pour assurer la stabilité de la fusée. Cependant, une marge statique trop élevée ou un gradient de portance excessif peut conduire à des conditions de surstabilité, qui peuvent être également problématiques.

Dans notre cas, le logiciel Stabtraj ne peut pas analyser le régime supersonique, car il ne prend pas en compte les variations de ces variables en fonction de la vitesse et donc du régime d'écoulement. Pour trouver un équilibre entre ces deux paramètres afin d'optimiser la stabilité de la fusée tout au long de son vol, nous avons dû prendre en compte les variables fluctuantes telles que la vitesse, le centre de pression (CP) et le centre de gravité (CG). Cette analyse approfondie nous permet de prendre des décisions éclairées lors de la conception de la fusée. Dans notre diagramme, nous avons seulement fait varier le CP et le gradient de portance en fonction du nombre de Mach. Pour tenir compte du déplacement du CG dû à la consommation de carburant du moteur, nous avons examiné deux scénarios pour le calcul de la marge statique : un avec le **moteur plein** (masse maximale) et un avec le **moteur vide** (masse minimale).



STABILITE

LE DIAGRAMME DES CRITÈRES DE STABILITE

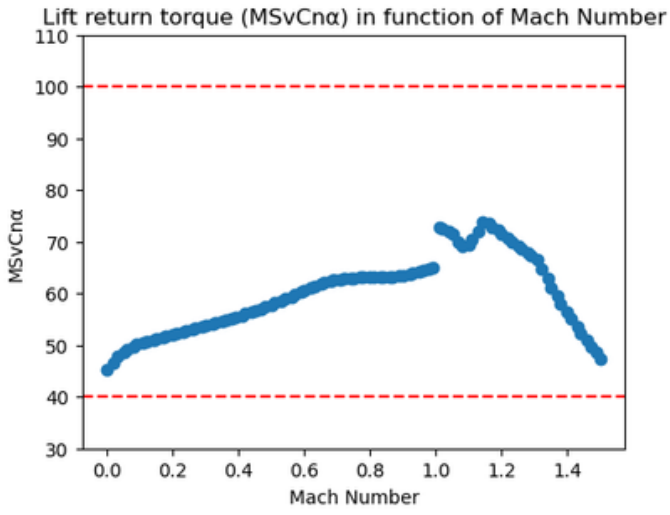
Le produit de la marge statique et du coefficient de pente de la courbe de portance (CLα) représente le couple de rappel de la portance, qui est un paramètre essentiel pour comprendre la stabilité aérodynamique et le contrôle d'une fusée ou de tout autre corps aérodynamique.

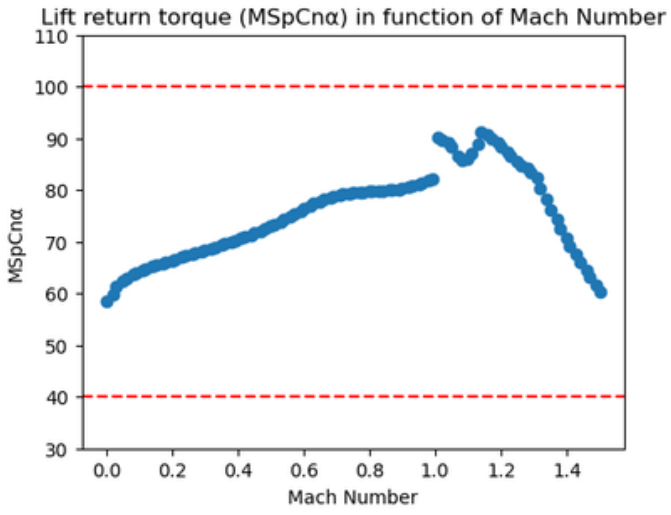
- **Couple de retours de portance = Marge statique × CLα**

Le couple de rappel de portance quantifie l'efficacité du moment de rappel aérodynamique généré par la force de portance en réponse aux changements de l'angle d'attaque. Ce paramètre permet de déterminer la rapidité et l'efficacité avec laquelle la fusée reprend son orientation initiale après avoir subi une perturbation.

En multipliant ces deux facteurs, on obtient le couple de rappel de la portance, qui donne une indication de la réactivité et de la stabilité globale de la fusée. Une valeur plus élevée du couple de rappel de portance indique que la fusée a un moment de rappel plus fort, ce qui lui permet de revenir plus efficacement à son orientation initiale après une perturbation.

(Voir l'article de Arnaud COLMON & Henri KANDEM publié dans le 32 Info n°60 de décembre 1997)





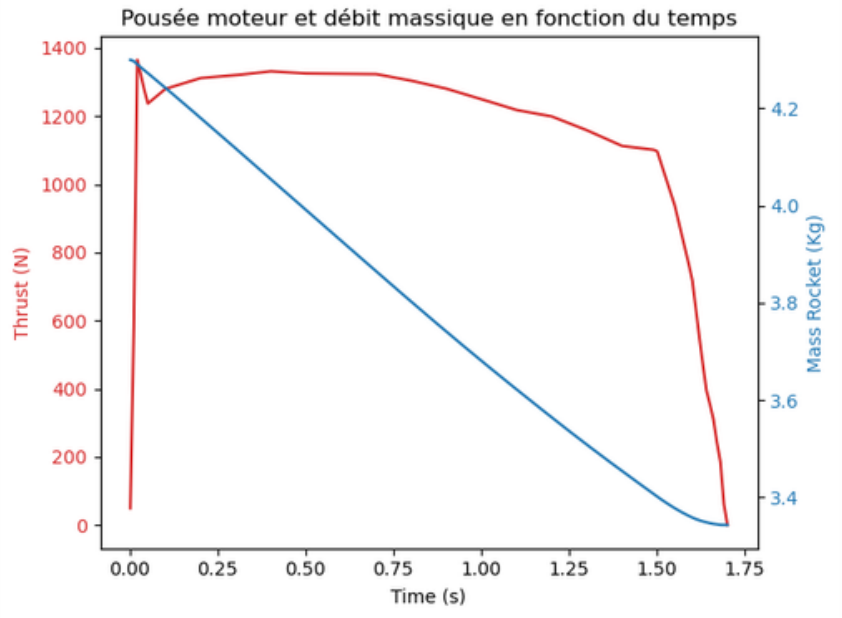
MOTEUR

INTERPOLATION DE LA POUSEE MOTEUR K1200

Grâce au site ThrustCurve.org, nous avons accès à l'ensemble des courbes de poussée des moteurs proposés par Cesaroni, dont le Pro 54 White Thunder (2014K1200-16A) que nous souhaitons utiliser pour notre projet. Toutefois, cette courbe se compose de seulement 26 points, ce qui est insuffisant pour l'utiliser dans le cadre d'une simulation de vol oblique ou pour évaluer les critères de stabilité. Pour résoudre ce problème, nous allons effectuer une interpolation à partir de ces données. L'interpolation linéaire est une méthode simple qui permet d'estimer la valeur d'une fonction entre deux points connus. Dans notre cas, nous utilisons l'interpolation linéaire pour estimer la poussée d'un moteur (exprimée en Newtons) à un moment précis, en nous basant sur les données disponibles.
La formule de base de l'interpolation linéaire entre deux points (x0, y0) et (x1, y1) est :

- **Formule de Taylor-Young du premier ordre : y = y0 + (y1 - y0) * (x - x0) / (x1 - x0)**

Dans le cadre de cette fonction, x0 et x1 représentent des instants temporels (t0 et t1), tandis que y0 et y1 correspondent à des valeurs de poussée (thrust0 et thrust1). "x" est l'instant pour lequel nous souhaitons estimer la poussée, et "y" est la poussée estimée à cet instant. Ainsi, grâce à l'interpolation linéaire, nous pouvons obtenir une courbe de poussée plus détaillée qui sera plus adaptée pour nos simulations de vol et évaluations de stabilité.



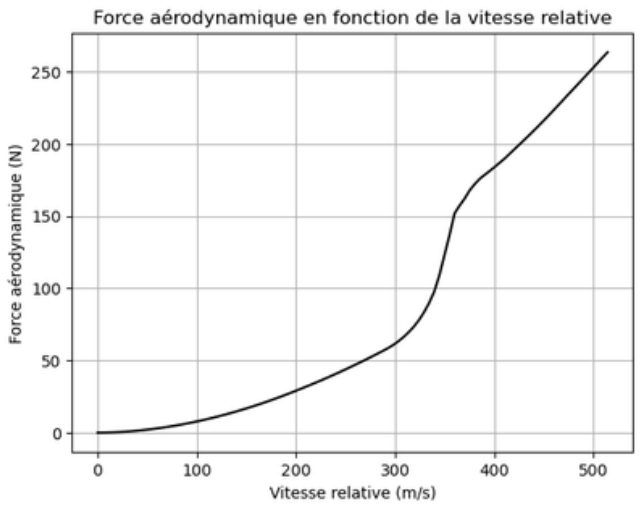
TRAJECTOIRE

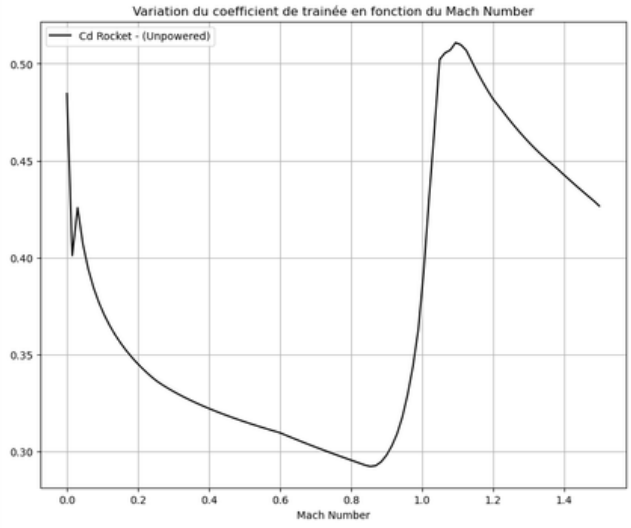
LA DIVERGENCE DE LA TRAINE

La divergence de traînée, aussi connue sous le nom de traînée d'onde ou augmentation de la traînée, est un phénomène critique qui affecte les performances des fusées lorsqu'elles atteignent et dépassent la vitesse du son (Mach 1). Lorsqu'une fusée se déplace à des vitesses transsoniques (environ entre 0,8 et 1,2 Mach), l'écoulement d'air autour de sa structure peut atteindre ou dépasser la vitesse du son localement, même si la vitesse globale de la fusée est encore subsonique.

À mesure que la fusée s'approche de la vitesse du son, la distribution de pression autour de sa structure change. Initialement, la pression chute en raison de la compression des molécules d'air dans les zones où l'écoulement est supersonique. Ce phénomène, appelé crise de traînée, peut entraîner une diminution de la traînée globale que la fusée subit. Cependant, lorsque la fusée continue d'accélérer et atteint des vitesses supérieures à Mach 1, des ondes de choc commencent à se former sur sa surface.

Dans le contexte d'un simulateur de vol de fusée en pas à pas oblique, avoir une estimation précise de la traînée à différents nombres de Mach est crucial. Cela permet d'obtenir des résultats de simulation plus précis et fiables, et d'anticiper correctement le comportement de la fusée pendant le vol, notamment à des vitesses proches ou supérieures à la vitesse du son où la divergence de traînée se produit.





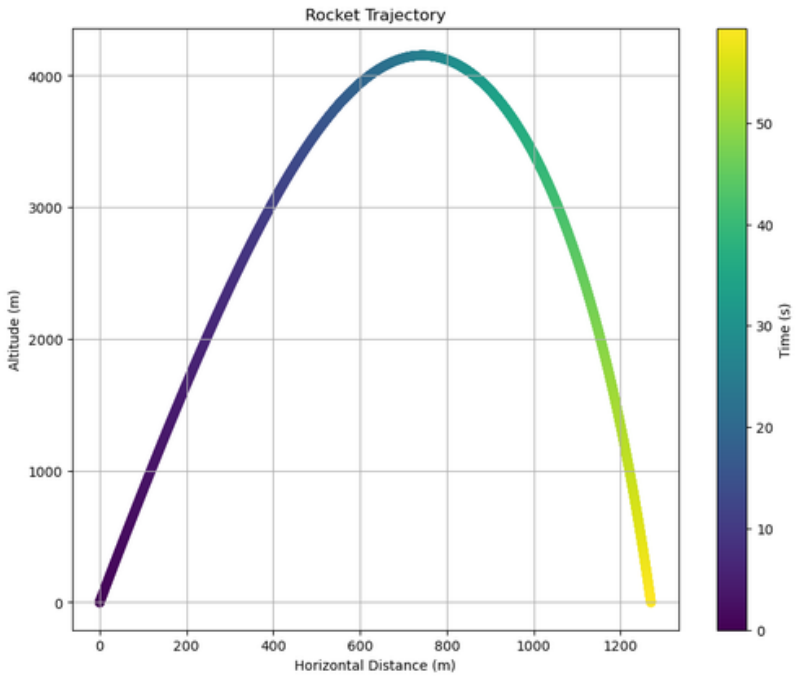
TRAJECTOIRE

SIMULATEUR OBLIQUE 2D - X & Y

La simulation du vol d'une fusée est une tâche complexe qui nécessite de prendre en compte une multitude de forces et de facteurs. Le document du vol de la fusée, rédigé par le CNES et l'association Planète Sciences, décrit plusieurs types de simulations possibles. Cependant, pour notre étude, il est crucial de mener une simulation complémentaire à celle réalisée par logiciel StabTraj. En effet, StabTraj ne prend pas en compte l'évolution de la traînée en fonction de la vitesse, ce qui peut entraîner des imprécisions dans les prévisions de trajectoires, en particulier pour les fusées supersoniques.

Afin de pallier cette lacune et d'optimiser notre étude, nous avons choisi de développer un simulateur de vol oblique en 2D. Ce choix nous permet d'obtenir des données clés sur la fusée, telles que le temps et l'altitude de l'apogée, la vitesse de sortie de la rampe, et la portée balistique en fonction de l'angle de lancement. Ce simulateur ne prend en compte que deux axes, X (horizontal) et Z (vertical), pour simplifier les calculs tout en obtenant des résultats suffisamment précis pour notre étude.

L'intégration d'un modèle de traînée plus précis dans cette simulation est essentielle pour améliorer l'exactitude des résultats. En effet et comme précisé précédemment, la traînée a un impact significatif sur le comportement de la fusée, notamment lorsqu'elle atteint des vitesses supersoniques. Par conséquent, l'inclusion de la variation de la traînée en fonction de la vitesse dans notre simulateur est un élément crucial pour obtenir une simulation plus réaliste et précise du vol de notre fusée.



MINES SPACE

DESIGN AND MANUFACTURING OF MSE SKINS

VERSION N°02

SPECIAL THANKS TO

easycomposites™ share the knowledge

CREATION OF AN EXPERIMENTAL ROCKET IN THE FRAMEWORK OF C'SPACE

PAUL BOYMOND

MINES SPACE 2023

Document objectives

This document aims to present and validate the design and production techniques of the 3 skins of the MSE fusex. As a reminder, this project is to carry out measurements and study the behaviour of a supersonic vehicle (mach 1.1). To do this, we must optimise the mass of the fusex and therefore switch to carbon fiber.

Some information about the mission:

- Mass of the fusex (with thruster): 4.5kg;
- Maximum speed: 400m/s (mach 1.16);
- Maximum acceleration (propelled phase): 28g;
- Minimum acceleration (ballistic phase) : -12g;
- Rocket length : 115mm;
- Rocket diameter : 90mm

The carbon skins of the Marasaut E (MSE), represent a new challenge for the association, which usually uses aluminium skins, cut by plasma.

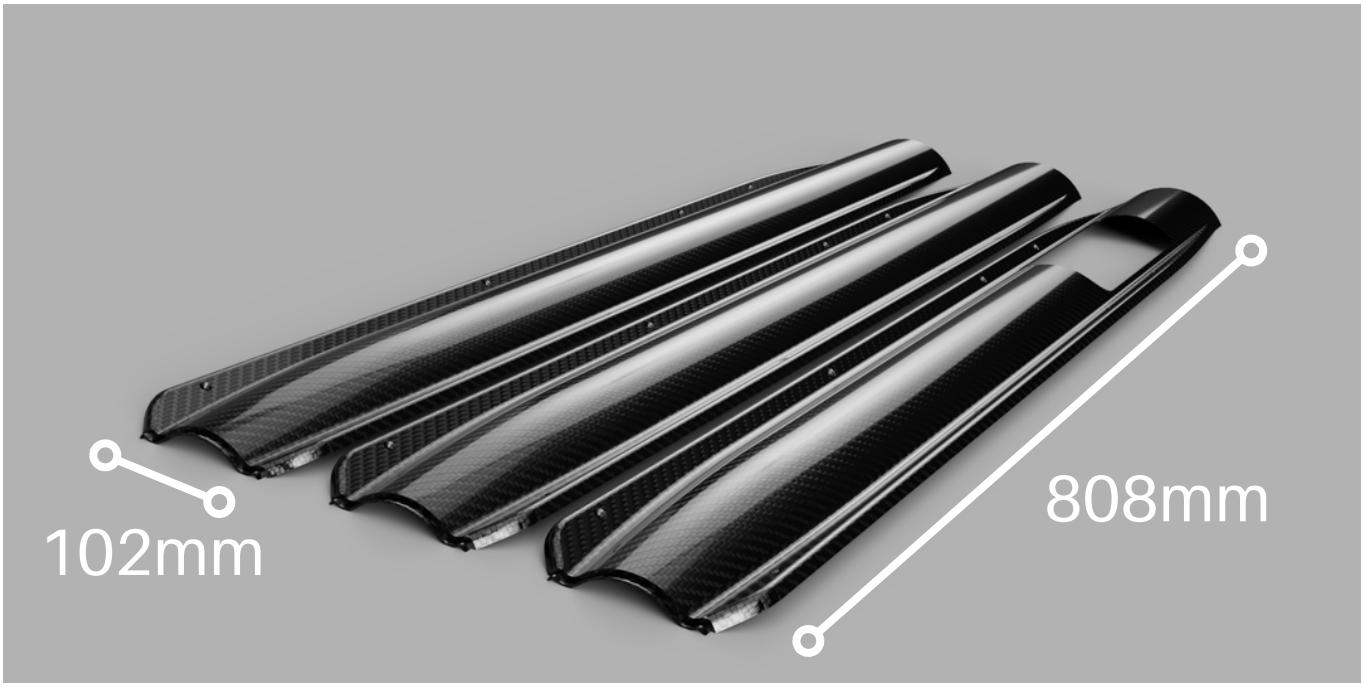
For more Information on the project :

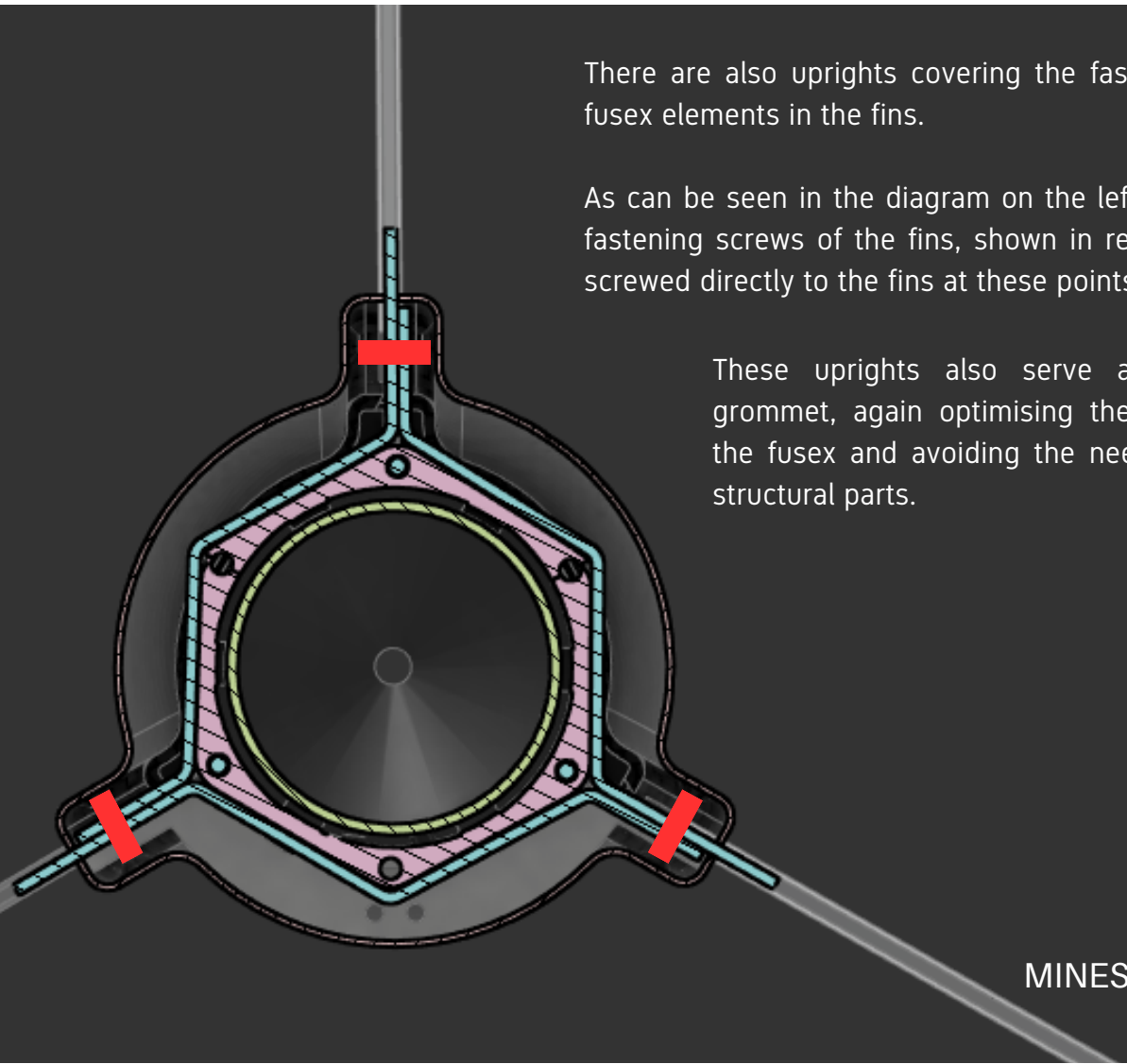
https://minesspace.fr/en/

MSE
MACH 1.1

Skins Shape

The architecture of the MARSAUT fusex with its fins and its external structure implies cutting the skin in 3 identical parts, themselves reworked by adding drillings and cuttings for the various elements of the fusex (parachute, skids, hatches, ...).





There are also uprights covering the fasteners of the various fusex elements in the fins.

As can be seen in the diagram on the left, the skin covers the fastening screws of the fins, shown in red. The skins are also screwed directly to the fins at these points.

These uprights also serve as a cable or pipe grommet, again optimising the volume available in the fusex and avoiding the need to drill or cut into structural parts.

Manufacturing

Process

The best technique for making these particular shapes is vacuum resin infusion. This consists of impregnating carbon fiber cloth with resin, which is then pressed against a mould by a plastic bag under vacuum. To do this, you must first make a fiberglass mould of the desired final shape. Here are the main steps:

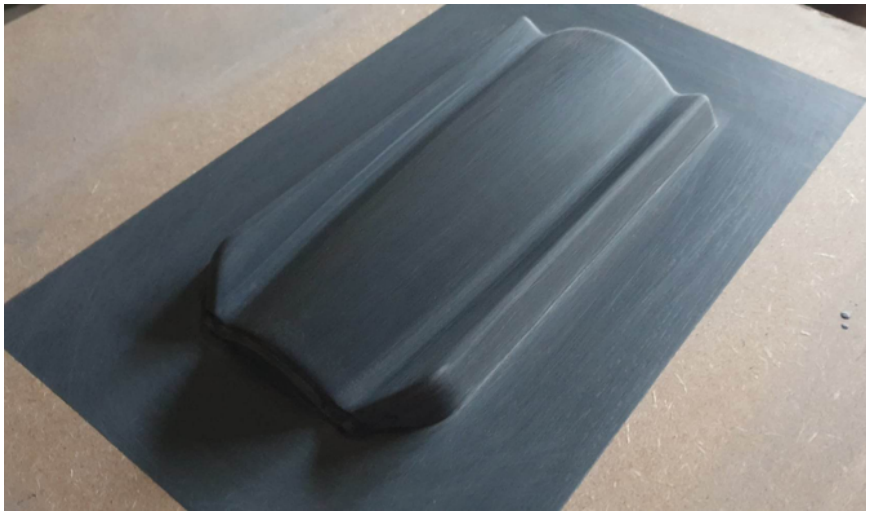
- Printing the model of the desired shape (PLA),
- Processing the model (sanding, priming, polishing and primer),
- Making the mould (Uni-Mould from EasyComposites),
- Processing of the mould (sanding and polishing),
- Installation of the carbon fiber,
- Installation of the infusion equipment,
- Infusion of resin into the carbon fiber,
- Processing and cleaning of the moulded carbon fiber
- Drilling and cutting the shapes according to the skin.

Making these parts requires a large number of steps, costly in time and material, which is why we cannot make a specific mould for each skin, which would have allowed us to mark the location of the cut-outs directly with the mould.

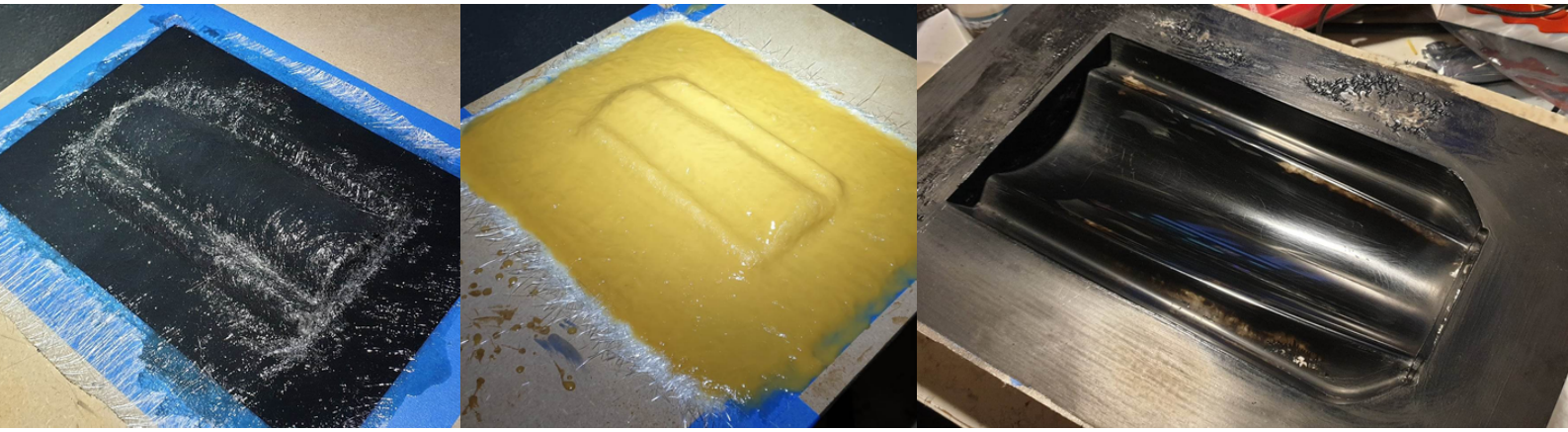
This document lists the steps and materials used, purchased from EsayComposites.

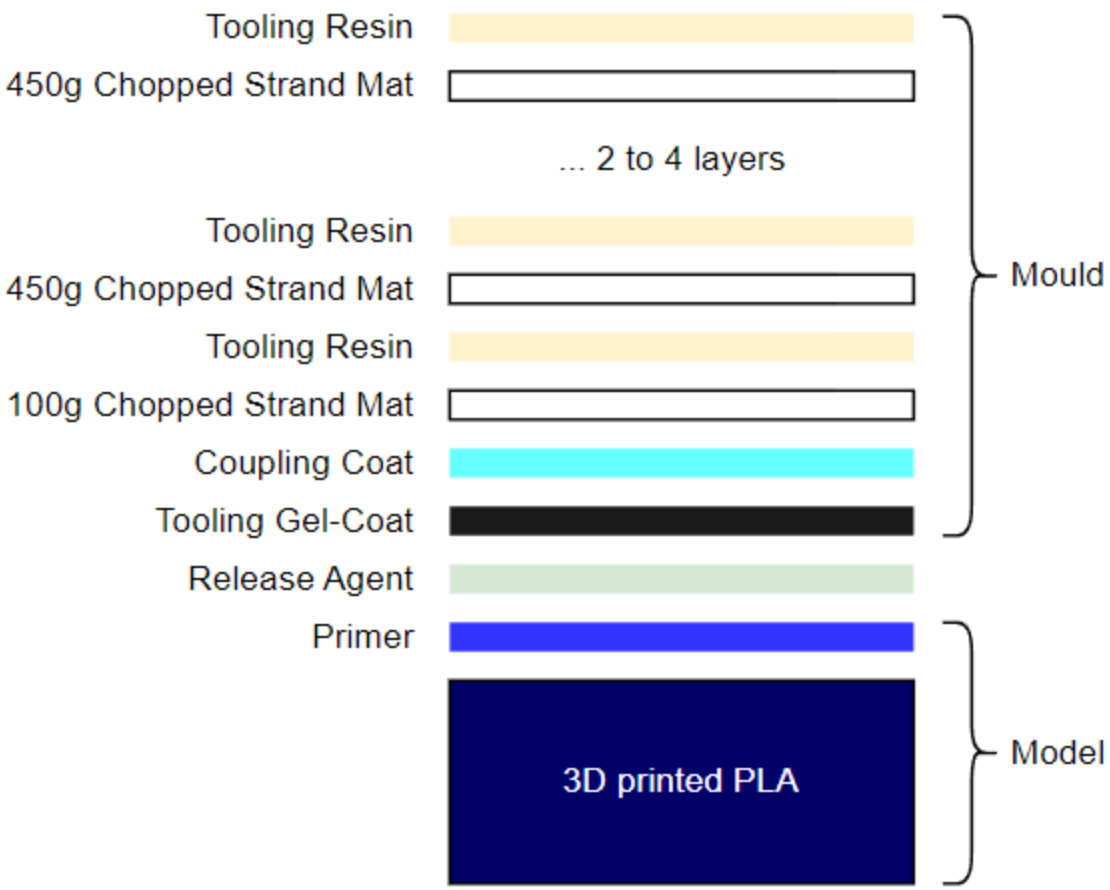
Model

The model is printed in PLA and then sanded with a finer and finer grain. It is then glued to a medium or polypropylene board. A primer (PCP Polyester Pattern-Coat Primer) is applied and allowed to extend a few centimetres (as shown in the photo). Then we sand and polish the model. Every imperfection of the model will be transferred to the mould.



Mould



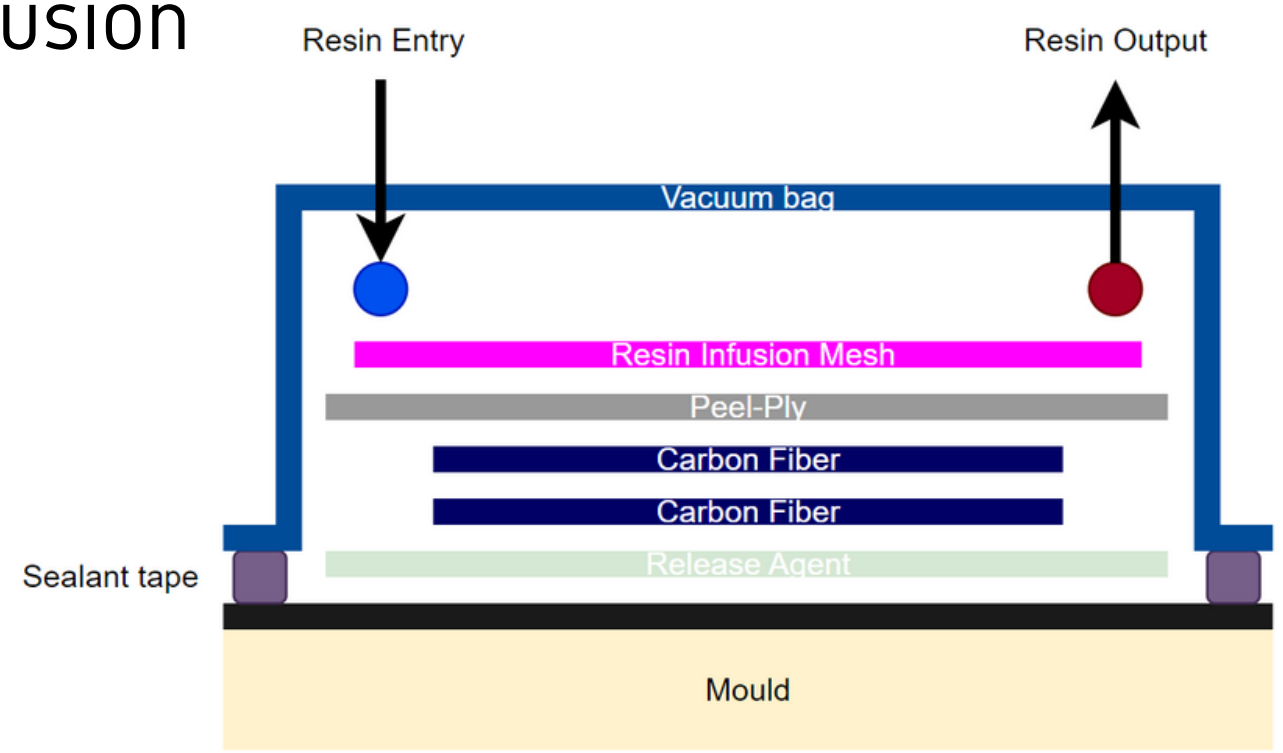


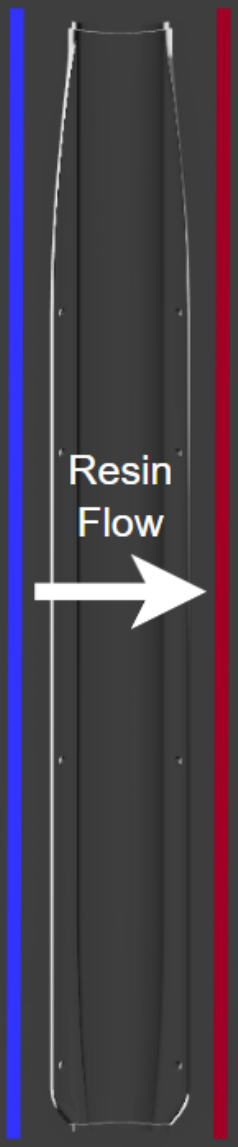
The production of the mould follows EasyComposites' guidelines:

- Several layers of release agent (6 layers at 20 min intervals),
- One or two layers of Tooling Gel-Coat, which will become the surface of the mould,
- One layer of Coupling Coat, which will be the bonding agent between the Gel-Coat and the fiberglass,
- Several layers of resin-soaked fiberglass (between 3 and 6).

It is then quite simple to separate the mould from the model, using spatulas or directly by blowing with a compressor. The mould can then be cleaned, sanded and polished.

Infusion





The infusion process follows EasyComposites' guidelines:

- Prepare the mould by cleaning it and applying 6 layers of release agent,
- Installation of the carbon fiber fabric (4 layers, see "Choice of materials" section).
- Installation of the Peel-Ply, which prevents the infusion mesh and the Vaccum Bag from sticking to the carbon fiber,
- Installation of the Mesh Infusion, which allows the resin to flow along the carbon fiber when it is compressed by vacuum.
- Installation of the resin inlet and outlet spiral hoses on each side of the carbon fiber (as shown in the left image).
- Installing the seals and vacuum bag, to provide a vacuum tight environment.
- Connecting the inlet pipe to a pot of special infusion liquid resin,
- Connect the outlet pipe to a catch pot to prevent resin from entering the vacuum pump.

After making sure that the system is airtight, you can start the infusion with a slow resin (90min to cure). The vacuum pump will suck in the resin which will move around and infuse into the carbon fiber. The resin entry is closed regularly to avoid air bubbles against the mould, as the resin moves less well in the deeper layer of carbon. Once the infusion is complete, the pipes are closed and left to cure for 24 hours (20°C).

As we wish to optimise the mass, the finishing will only be done with a sanding and polishing (without clear-coat).

Materials

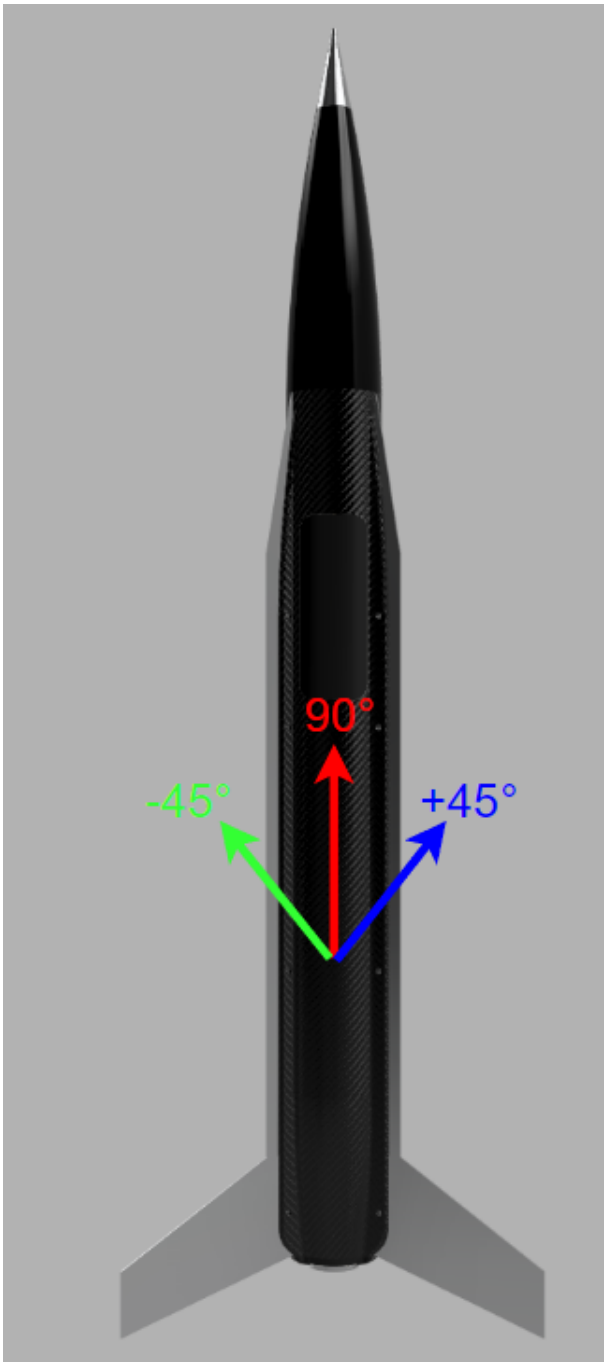
Although the skins are not structural elements of the fusex, they will experience a high vertical acceleration (28g) and may also encounter a torsional force. For this purpose, we decided to use 3 layers of carbon fiber in the following way:

- 90g Plain Weave 1k,
- 300g +/-45 Biaxial 3k,
- 90g Plain Weave 1k.

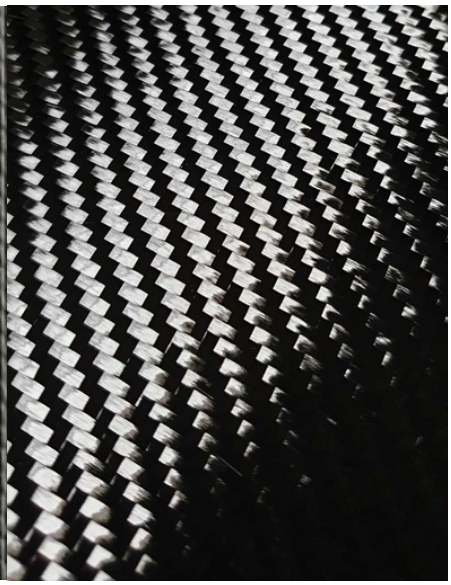
This stacking allows the skins to be reinforced in different directions, as well as to counteract torsion.

Layers 1 and 3 are oriented at 90°, in the direction of acceleration.

Layers 2 is oriented at -45° and +45°, against torsion and acceleration.



Layer 2 and 3



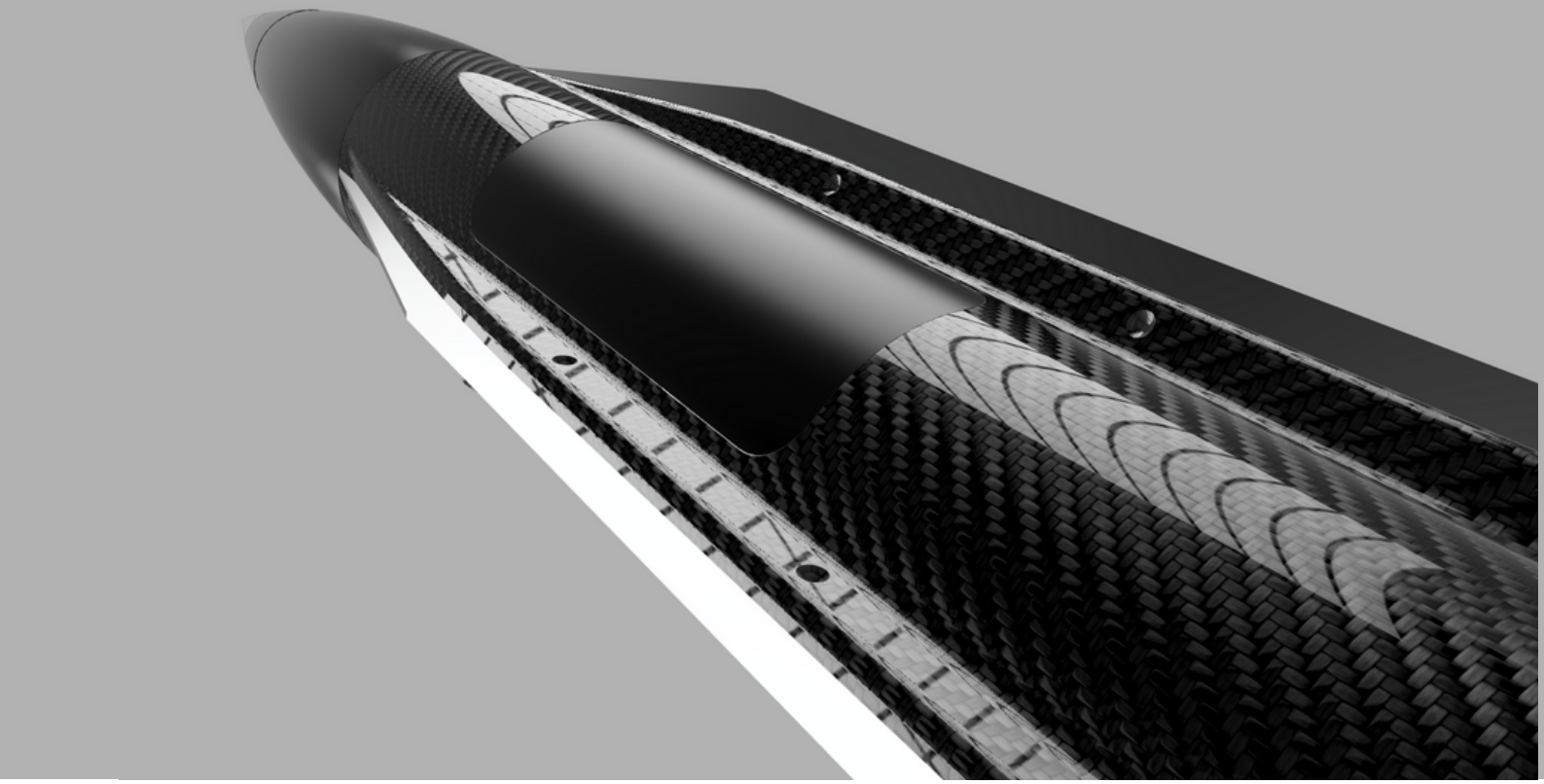
Layer 1 and 4

Each skin has a surface of 0.09m², with a ratio of 60% carbon fiber and 40% resin, we obtain the following results

- 0.36m² and 70g of carbon fiber per skin
- 98.28g total per skin (respected by +/-10% on the produced skins),
- 294.84g for the 3 skins.

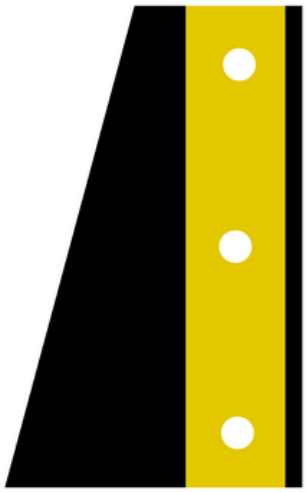
It is quite difficult to simulate or calculate the strength of such an assembly, so we will perform our tests empirically and add a high safety margin.

Reinforcements



The skins are fixed with screws. Carbon fiber is sensitive to tearing around the holes, which could happen due to the high acceleration. To reinforce the holes, other materials can be added to the sandwitch, such as Kevlar fabrics or aluminium inserts.

Reinforcement buried in the carbon fiber. An insert is added to each hole to increase the surface area of the hole on the carbon. This technique seems complicated to implement because the inserts must be added at the time of infusion. It is also possible to glue them in when the part is finished as a washer, but this is less resistant and goes beyond the template.



Reinforcement with a carbon-kevlar fabric. A strip of Kevlar is added to the full height of the skin at the location of the holes. Kevlar is very resistant to impact and allows for a better distribution of the forces on the rest of the carbon fiber skin. This solution seems relatively simple to implement.

However, we can ask ourselves the question of the necessity to reinforce the holes, indeed the acceleration will be very important but the skins being very light, the carbon fiber could be enought.





Stability study of MSE Rocket

- Author : Paul Miailhe
- Version : n°10
- Date : 16/05/2023
- CC BY-NC-SA

In [1]:
```
#Library:
import math
    # Convert PDF
import nbconvert
    # linear algebra
import numpy as np
    # data processing
import pandas as pd
    # data visualization
import seaborn as sns
%matplotlib inline
from matplotlib import pyplot as plt
from matplotlib import style
```

Aerolab data :

In [2]:
```
# Load aerolab data :
cp = pd.read_csv('aerolab/CP.csv', sep = ';',  decimal=',')
th = pd.read_csv('engine/Cesaroni_2014K1200-16A.csv', sep = ';',  decimal=',')
ci = pd.read_csv('aerolab/(CI)apha2.csv', sep = ';',  decimal=",")
dg = pd.read_csv('aerolab/drag.csv', sep = ';',  decimal=",")
#cp.info()
#th.info()
#ci.info()
dg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101 entries, 0 to 100
Data columns (total 10 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   velocity                 101 non-null    float64
 1   Mach Number              101 non-null    float64
 2   Cd Rocket - (Unpowered)  101 non-null    float64
 3   Body Skin Friction       101 non-null    float64
 4   Nose Pressure Drag       101 non-null    float64
 5   Base Drag                101 non-null    float64
 6   Body Pressure Drag       101 non-null    float64
 7   Fin Pressure Drag        101 non-null    float64
 8   Interference Drag        101 non-null    float64
 9   Fin Skin Friction        101 non-null    float64
dtypes: float64(10)
memory usage: 8.0 KB
```

In [3]:
```
# Data Exploration/Analysis:

## CP :
cp.describe()
cp.head(4)
print(cp)
```

```
     Velocity (m/s)  Mach Number  (X)cp - Rocket (mm)  (X)cp - Nose/Body (mm)  \
0             0.000         0.00               933.66                   69.69
1             5.145         0.02               936.14                   69.43
2            10.290         0.03               938.54                   69.17
3            15.435         0.05               940.06                   68.93
4            20.580         0.06               941.16                   68.71
..              ...          ...                  ...                     ...
96          493.920         1.44               956.84                  412.42
97          499.065         1.46               954.52                  409.91
98          504.210         1.47               952.23                  407.47
99          509.355         1.49               949.94                  405.09
100         514.500         1.50               947.66                  402.69

     (X)cp - Wing  (X)cp - Tail  (X)cp - Body-Wing  (X)cp - body-Tail
0          435.63       1076.99              412.66            1071.92
1          435.63       1076.99              412.66            1071.92
2          435.62       1076.99              412.65            1071.92
3          435.61       1076.99              412.65            1071.92
4          435.60       1076.99              412.64            1071.92
..            ...          ...                 ...                ...
96         444.82       1098.24              811.64            1113.88
97         445.32       1098.27              811.64            1113.88
98         445.80       1098.30              811.64            1113.88
99         446.28       1098.33              811.64            1113.88
100        446.76       1098.35              811.64            1113.88

[101 rows x 8 columns]
```

In [4]:

```
## CI :
ci.describe()
ci.head(5)
print(ci)
```

```
     Velocity  Mach Number  (Cl)alpha - Rocket  (Cl)alpha - Nose/Body  \
0       0.000        0.000           19.430981               1.937276
1       5.145        0.015           19.745208               1.932269
2      10.290        0.030           20.059803               1.927537
3      15.435        0.045           20.254768               1.923070
4      20.580        0.060           20.393504               1.918859
..        ...          ...                 ...                    ...
96    493.920        1.440           20.215841               3.236806
97    499.065        1.455           19.919270               3.230479
98    504.210        1.470           19.633769               3.224319
99    509.355        1.485           19.358678               3.218364
100   514.500        1.500           19.093378               3.212647

     (Cl)alpha - Wing  (Cl)alpha - Tail  (Cl)alpha - Body-Wing  \
0            0.756620         13.809403               0.505410
1            0.756815         14.080854               0.505540
2            0.756995         14.352400               0.505661
3            0.757079         14.522013               0.505717
4            0.757131         14.643594               0.505752
..                ...               ...                    ...
96           0.785726         14.808225               0.524852
97           0.786358         14.549808               0.525275
98           0.786973         14.301584               0.525685
99           0.787571         14.062938               0.526085
100          0.788152         13.833300               0.526473

     (Cl)alpha - body-Tail  (Cl)alpha - Wing-Tail
0                 3.017584              -0.595313
1                 3.076901              -0.607171
2                 3.136238              -0.619027
3                 3.173301              -0.626412
4                 3.199869              -0.631701
..                     ...                    ...
96                1.519546              -0.659315
97                1.475684              -0.648333
98                1.432981              -0.637772
99                1.391329              -0.627609
100               1.350623              -0.617818

[101 rows x 9 columns]
```

In [5]:

```
## Drag :
dg.describe()
dg.head(5)
#np.mean(dg['Cd Rocket - (Unpowered)'])
print(dg)
```

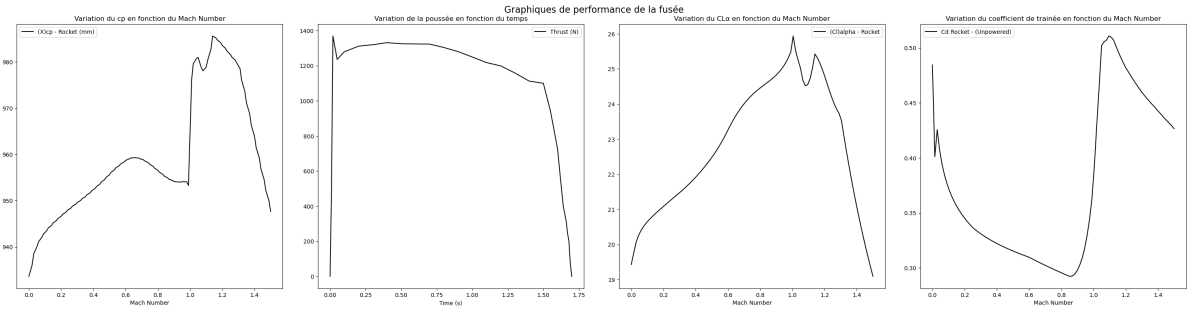
```
     velocity  Mach Number  Cd Rocket - (Unpowered)  Body Skin Friction  \
0       0.000        0.000                 0.484539            0.134449
1       5.145        0.015                 0.401070            0.095120
2      10.290        0.030                 0.425743            0.143688
3      15.435        0.045                 0.406942            0.140679
4      20.580        0.060                 0.394139            0.138081
..        ...          ...                      ...                 ...
96    493.920        1.440                 0.436175            0.087268
97    499.065        1.455                 0.433801            0.086929
98    504.210        1.470                 0.431464            0.086591
99    509.355        1.485                 0.429162            0.086253
100   514.500        1.500                 0.426606            0.085917

     Nose Pressure Drag  Base Drag  Body Pressure Drag  Fin Pressure Drag  \
0              0.000000   0.050617            0.008162       3.640000e-12
1              0.000000   0.060179            0.005774       2.620000e-09
2              0.000000   0.048963            0.008722       2.090000e-08
3              0.000000   0.049484            0.008540       7.060000e-08
4              0.000000   0.049947            0.008382       1.670000e-07
..                  ...        ...                 ...                ...
96             0.070105   0.120039            0.019625       2.545962e-02
97             0.069938   0.119389            0.019368       2.495983e-02
98             0.069768   0.118739            0.019122       2.448363e-02
99             0.069597   0.118089            0.018887       2.402919e-02
100            0.069426   0.117150            0.018663       2.359490e-02

     Interference Drag  Fin Skin Friction
0             0.042351           0.248960
1             0.029963           0.210034
2             0.045262           0.179108
3             0.044314           0.163926
4             0.043495           0.154233
..                 ...                ...
96            0.036250           0.077427
97            0.036132           0.077086
98            0.036015           0.076746
99            0.035898           0.076408
100           0.035782           0.076072

[101 rows x 10 columns]
```

In [6]:

```
## Thrust :
th.describe()
th.head(5)
print(th)
```

	Time (s)	Thrust (N)
0	0.00	0.00
1	0.01	492.25
2	0.02	1369.46
3	0.05	1236.01
4	0.10	1279.47
5	0.20	1311.39
6	0.30	1319.85
7	0.40	1331.39
8	0.50	1325.23
9	0.70	1323.31
10	0.80	1304.08
11	0.90	1280.62
12	1.00	1249.86
13	1.10	1217.94
14	1.20	1199.29
15	1.30	1158.77
16	1.40	1112.56
17	1.50	1099.87
18	1.55	941.81
19	1.60	726.07
20	1.62	559.17
21	1.64	399.95
22	1.66	317.66
23	1.67	247.28
24	1.68	198.05
25	1.69	67.30
26	1.70	0.00

In [7]:
```
# Création d'une figure et d'une grille de sous-tracés (3 lignes, 1 colonne)
fig, axs = plt.subplots(1, 4, figsize=(30, 8))

# Affichage du premier graphique sur le premier sous-tracé
cp.plot(x='Mach Number', y='(X)cp - Rocket (mm)', ax=axs[0], color='black')
axs[0].set_title('Variation du cp en fonction du Mach Number')

# Affichage du deuxième graphique sur le deuxième sous-tracé
th.plot(x='Time (s)', y='Thrust (N)', ax=axs[1], color='black')
axs[1].set_title('Variation de la poussée en fonction du temps')

# Affichage du troisième graphique sur le troisième sous-tracé
ci.plot(x='Mach Number', y='(Cl)alpha - Rocket', ax=axs[2], color='black')
axs[2].set_title('Variation du CLα en fonction du Mach Number')

# Affichage du quatrième graphique sur le troisième sous-tracé
dg.plot(x='Mach Number', y='Cd Rocket - (Unpowered)', ax=axs[3], color='black')
axs[3].set_title('Variation du coefficient de trainée en fonction du Mach Number')

# Ajustement des espaces entre les sous-tracés et ajout d'un titre pour
#l'ensemble des graphiques
fig.tight_layout(pad=3.0)
fig.suptitle('Graphiques de performance de la fusée', fontsize=16)

# Ajout d'une grille
#ax.grid(True)

# Affichage de la figure
plt.show()
```



Static Margin & Lift return torque :

In [8]:

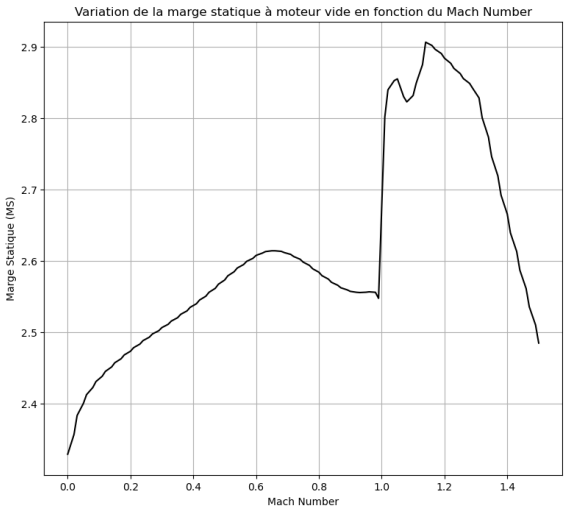
```
# Définition de trois variables :
Dref = 90 # Diamètre de référence
CdGv = 724 # Centre de gravité à vide
CdGp = 663 # Centre de gravité en charge

# Calcul de quatre variables de stabilité :
# Position du centre de pression de la fusée
CPr = cp['(X)cp - Rocket (mm)']
# Vitesse de Mach
MACH = cp['Mach Number']
# Pente de la courbe de portance (coefficient de pente)
Cla = ci['(Cl)alpha - Rocket']

# Calcul de la marge statique à vide de la fusée
MSv = (CPr - CdGv) / 90
# Calcul de la marge statique en charge de la fusée
MSp = (CPr - CdGp) / 90

# Calcul du produit MS.Cna représente le couple de rappel à vide de la fusée
MsvCla = MSv*Cla
# Calcul du produit MS.Cna représente le couple de rappel en charge de la fusée
MspCla = MSp*Cla

# Création d'un DataFrame "ft" :
ft = pd.DataFrame({
    # Ajout des données de vitesse de Mach à la colonne "Mach Number"
    "Mach Number": MACH,
    # Ajout des données de pente de la courbe de portance à la colonne "(Cl)alpha"
    "(Cl)alpha": Cla,
    # Ajout des données de marge statique à vide à la colonne "MSvide"
    "MSvide": MSv,
    # Ajout des données de marge statique en charge à la colonne "MSplein"
    "MSplein": MSp,
    # Ajout des données du produit MS.Cnα en charge à la colonne "MSvide"
    "MSvCnα": MsvCla,
    # Ajout des données du produit MS.Cnα en charge à la colonne "MSplein"
    "MSpCnα": MspCla
})

# Afficher toutes les lignes du DataFrame
pd.set_option('display.max_rows', None)
ft.head(5)
```

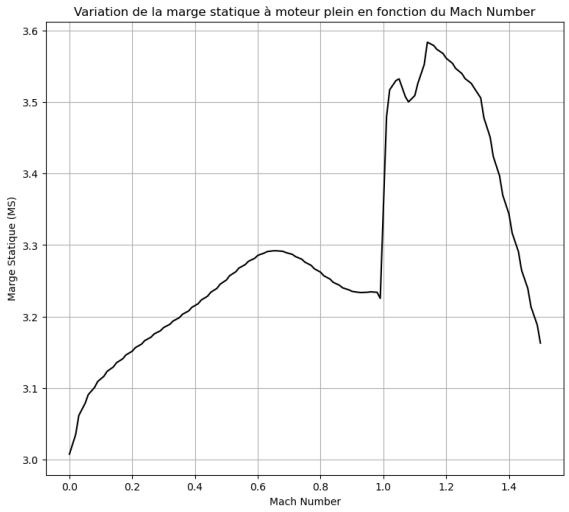
Out[8]:

	Mach Number	(Cl)alpha	MSvide	MSplein	MSvCnα	MSpCnα
0	0.00	19.430981	2.329556	3.007333	45.265551	58.435438
1	0.02	19.745208	2.357111	3.034889	46.541648	59.924511
2	0.03	20.059803	2.383778	3.061556	47.818113	61.414202
3	0.05	20.254768	2.400667	3.078444	48.624945	62.353177
4	0.06	20.393504	2.412889	3.090667	49.207259	63.029522

In [9]:

```
# Création d'une figure
fig, ax = plt.subplots(1, 2, figsize=(20, 8))

# Affichage du graphique sur le tracé en noir
ax[0].plot(cp['Mach Number'], ft['MSvide'], color='black', label='MSvide Aerolab Da
ax[1].plot(cp['Mach Number'], ft['MSplein'], color='black', label='MSplein Aerolab

# Titre du graphique
ax[0].set_title('Variation de la marge statique à moteur vide en fonction du Mach N
ax[1].set_title('Variation de la marge statique à moteur plein en fonction du Mach

# Ajout d'une grille
ax[0].grid(True)
ax[1].grid(True)
ax[1].set_xlabel('Mach Number')
ax[1].set_ylabel('Marge Statique (MS)')
ax[0].set_xlabel('Mach Number')
ax[0].set_ylabel('Marge Statique (MS)')

# Afficher la figure
plt.show()
```





Stability criteria diagram :

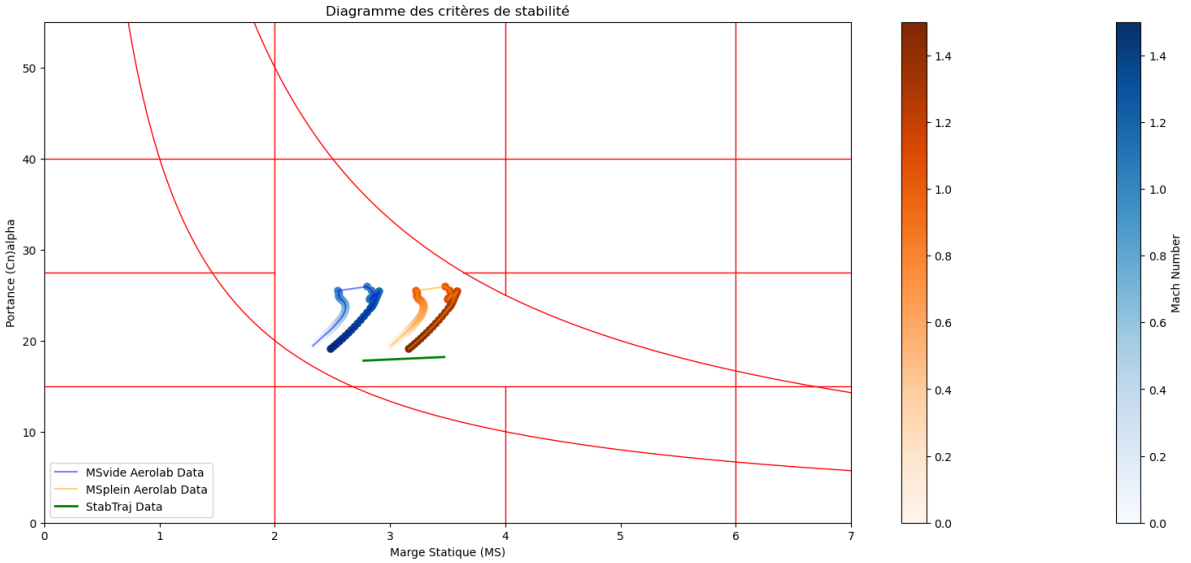
Requirement :

- Static Margin MS between 2 and 6 gauges.
- Gradient of Lift Clα between 15 and 40.
- Product MS.Cnα between 40 and 100.
- Fineness L/D between 10 and 35.
- Ramp exit speed greater than 20 m/s.

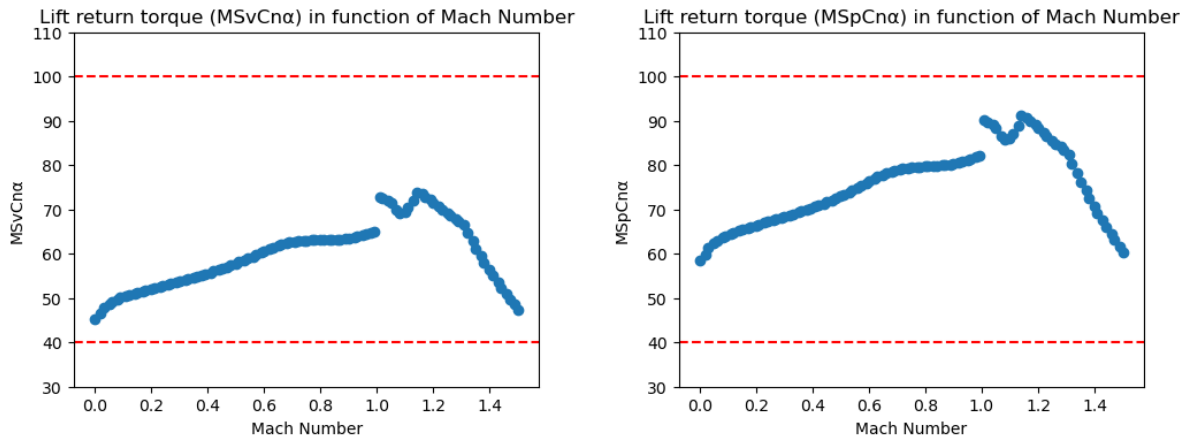
In [10]:

```
# Création de la figure et d'un sous-graphique
fig, ax1 = plt.subplots(1, 1, figsize=(20, 8))

# Définition des limites de l'axe X du premier sous-graphique
ax1.set_xlim([0, 7])

# Définition des limites de l'axe Y du premier sous-graphique
ax1.set_ylim([0, 55])

# Création d'un nuage de points avec des couleurs en fonction du nombre de Mach
# pour les données MSvide et (Cl)alpha
sc = ax1.scatter(ft['MSvide'], ft['(Cl)alpha'], c=ft['Mach Number'], cmap='Blues')

# Ajout d'une ligne reliant les points du nuage de points MSvide
ax1.plot(ft['MSvide'], ft['(Cl)alpha'], color='blue', alpha=0.5, label='MSvide Aer

# Création d'un deuxième nuage de points avec des couleurs en fonction du nombre de
# pour les données MSplein et (Cl)alpha
sc2 = ax1.scatter(ft['MSplein'], ft['(Cl)alpha'], c=ft['Mach Number'], cmap='Orang

# Ajout d'une ligne reliant les points du nuage de points MSplein
ax1.plot(ft['MSplein'], ft['(Cl)alpha'], color='orange', alpha=0.5, label='MSplein

# Création d'un plot pour les données StabTraj
ax1.plot([2.77, 3.47], [17.8, 18.2], color='green', linewidth=2, label='StabTraj Da

# Définition des limites de la barre de couleurs pour chaque nuage de points
sc.set_clim(0, 1.5)
sc2.set_clim(0, 1.5)

# Ajout d'une barre de couleurs pour chaque nuage de points
cbar1 = fig.colorbar(sc)
cbar1.set_label('Mach Number')
cbar2 = fig.colorbar(sc2)

# Ajout de lignes rouges personnalisées sur le premier sous-graphique
ax1.plot([0, 7], [15, 15], color='red', linewidth=1)
ax1.plot([0, 7], [40, 40], color='red', linewidth=1)
ax1.plot([2, 2], [0, 55], color='red', linewidth=1)
ax1.plot([6, 6], [0, 55], color='red', linewidth=1)
ax1.plot([4, 4], [0, 15], color='red', linewidth=1)
ax1.plot([4, 4], [25, 55], color='red', linewidth=1)
ax1.plot([0, 2], [27.5, 27.5], color='red', linewidth=1)
ax1.plot([3.636363, 7], [27.5, 27.5], color='red', linewidth=1)

# Définition des axes X et Y pour les courbes personnalisées
x1 = np.linspace(0.01, 7, 300)
y1 = 100 * x1 ** -1
x2 = np.linspace(0.01, 7, 300)
y2 = 40 * x2 ** -1

# Ajout des courbes personnalisées sur le premier sous-graphique
ax1.plot(x1, y1, color='red', linewidth=1)
ax1.plot(x2, y2, color='red', linewidth=1)

# Ajout des étiquettes pour les axes et le titre du graphique
plt.xlabel('Marge Statique (MS)')
plt.ylabel('Portance (Cn)alpha')
plt.title('Diagramme des critères de stabilité')

# Ajout de la légende pour les nuages de points
ax1.legend()
```

```
# Affichage abaque
plt.show()
```



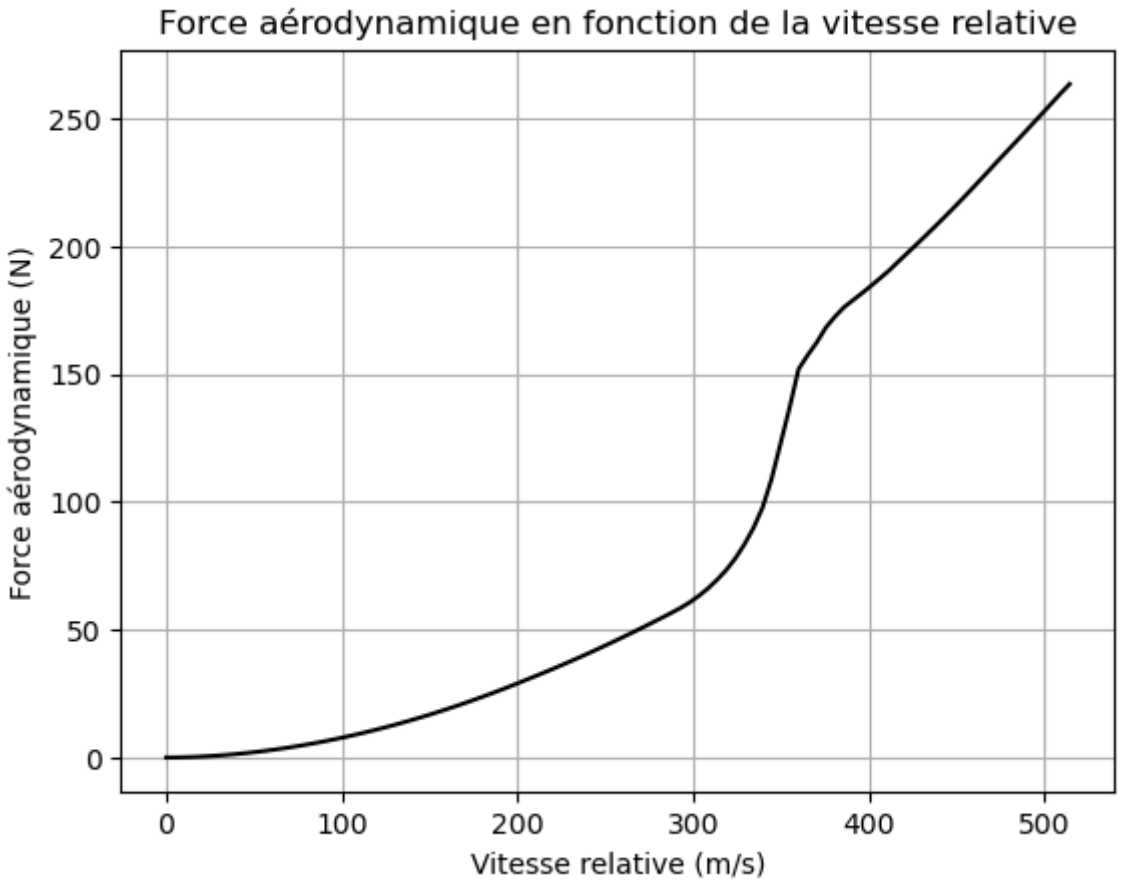
In [11]:
```
# Créer une figure avec 2 subplots, 1 ligne et 2 colonnes
fig, axs = plt.subplots(1, 2, figsize=(12, 4))

# Tracer le graphe pour MSvCnα sur le premier subplot
axs[0].scatter(ft['Mach Number'], ft['MSvCnα'])
axs[0].set_xlabel('Mach Number')
axs[0].set_ylabel('MSvCnα')
axs[0].set_title('Lift return torque (MSvCnα) in function of Mach Number')
axs[0].set_ylim([30, 110])
axs[0].axhline(y=40, color='r', linestyle='--')
axs[0].axhline(y=100, color='r', linestyle='--')

# Tracer le graphe pour MSpCnα sur le deuxième subplot
axs[1].scatter(ft['Mach Number'], ft['MSpCnα'])
axs[1].set_xlabel('Mach Number')
axs[1].set_ylabel('MSpCnα')
axs[1].set_title('Lift return torque (MSpCnα) in function of Mach Number')
axs[1].set_ylim([30, 110])
axs[1].axhline(y=40, color='r', linestyle='--')
axs[1].axhline(y=100, color='r', linestyle='--')

# Ajuster les espaces entre les subplots
plt.subplots_adjust(wspace=0.3)

# Afficher les subplots
plt.show()
```



Drag Analisys, the drag divergence and the drag rise:

In [12]:
```python
def calculate_drag(v):

    # Surface de référence, représentant la dimension du corps de la fusée (m²)
    Sref = 0.00361173
    # Densité de l'air (kg/m3)
    rho = 1.292
    # Coefficients Aérodynamiques (sans unité) fournis par aerolab
    #Coeff = dg['Cd Rocket - (Unpowered)']

    # Chercher le coefficient d'aérodynamique correspondant à la vitesse donnée
    Coeff = None
    for i in range(len(dg)):
        if dg['velocity'][i] >= v:
            if i == 0:
                Coeff = dg['Cd Rocket - (Unpowered)'][i]
            else:
                # Interpolation linéaire
                x1, x2 = dg['velocity'][i-1:i+1]
                y1, y2 = dg['Cd Rocket - (Unpowered)'][i-1:i+1]
                Coeff = y1 + (v-x1)*(y2-y1)/(x2-x1)
            break

    # Calcul de la force de traînée en Newtons
    DfDrag = Sref * 0.5 * rho * Coeff * math.pow(v, 2)

    return DfDrag
```

In [13]:
```python
### Calculation of the drag in N :

# Vitesse relative, ou Vent relatif ((m/s)²)
v = dg['velocity']
# Calcul de la force aérodynamique totale en Newtons
Drag = [calculate_drag(t) for t in v]

dda = pd.DataFrame({
    "velocity (m/s)": v,
    "Drag (N)": Drag
})

dda.to_csv('Drag')

# Tracer un graphique de la force aérodynamique en fonction de la vitesse relative
plt.plot(v, Drag, color='black')

# Ajouter des noms d'axes et un titre
plt.xlabel('Vitesse relative (m/s)')
plt.ylabel('Force aérodynamique (N)')
plt.title('Force aérodynamique en fonction de la vitesse relative')
plt.grid(True)

# Afficher le graphique
plt.show()
```



Interpolation de la pousée :

L'interpolation linéaire est une méthode simple pour estimer la valeur d'une fonction entre deux points connus. Dans le cas de cette fonction, nous utilisons l'interpolation linéaire pour estimer la poussée d'un moteur (en Newtons) à un temps donné, basée sur les données disponibles. La formule de base pour l'interpolation linéaire entre deux points (x0, y0) et (x1, y1) est la suivante :

- formule de Taylor-Young du premier ordre : y = y0 + (y1 - y0) * (x - x0) / (x1 - x0)

Dans le contexte de cette fonction, x0 et x1 correspondent à des temps (t0 et t1), et y0 et y1 correspondent à des valeurs de poussée (thrust0 et thrust1). x est le temps pour lequel nous voulons estimer la poussée, et y est la poussée estimée à ce temps.

In [14]:
```python
# Fonction d'interpolation linéaire de la poussée
def calculate_thrust(t):
    data = th[["Time (s)", "Thrust (N)"]].to_records(index=False)
    # Parcourir toutes les paires de points de données consécutives
    for i in range(len(data) - 1):
        # Récupérer les temps et les poussées pour les points de données actuels et
        t0, thrust0 = data[i]
        t1, thrust1 = data[i + 1]

        # Vérifier si le temps demandé est compris entre les temps des points de da
        if t0 <= t <= t1:
            # Calculer la poussée interpolée à l'aide de l'interpolation linéaire e
            thrust = thrust0 + (thrust1 - thrust0) * (t - t0) / (t1 - t0)
            # Retourner la poussée interpolée pour le temps demandé
            return thrust
```

```
    # Retourner 0 si le temps demandé est en dehors de la plage des données fournie
    return 0
```

In [15]:
```
# Définition des paramètres de simulation :

MaxTimeThrust = 1.7 # Temps de pousée moteur
pas = 0.01 # Pas de la simuation
mass = 4.3 # Masse initial de la fusée(Kg)
ISP = 213.9 # Inpulsion Spécifique PRO-54 White Thunder (s)
g = 9.80665 # Accélération de la pesanteur (m/s2)

# Définition temporel de la simuation
```

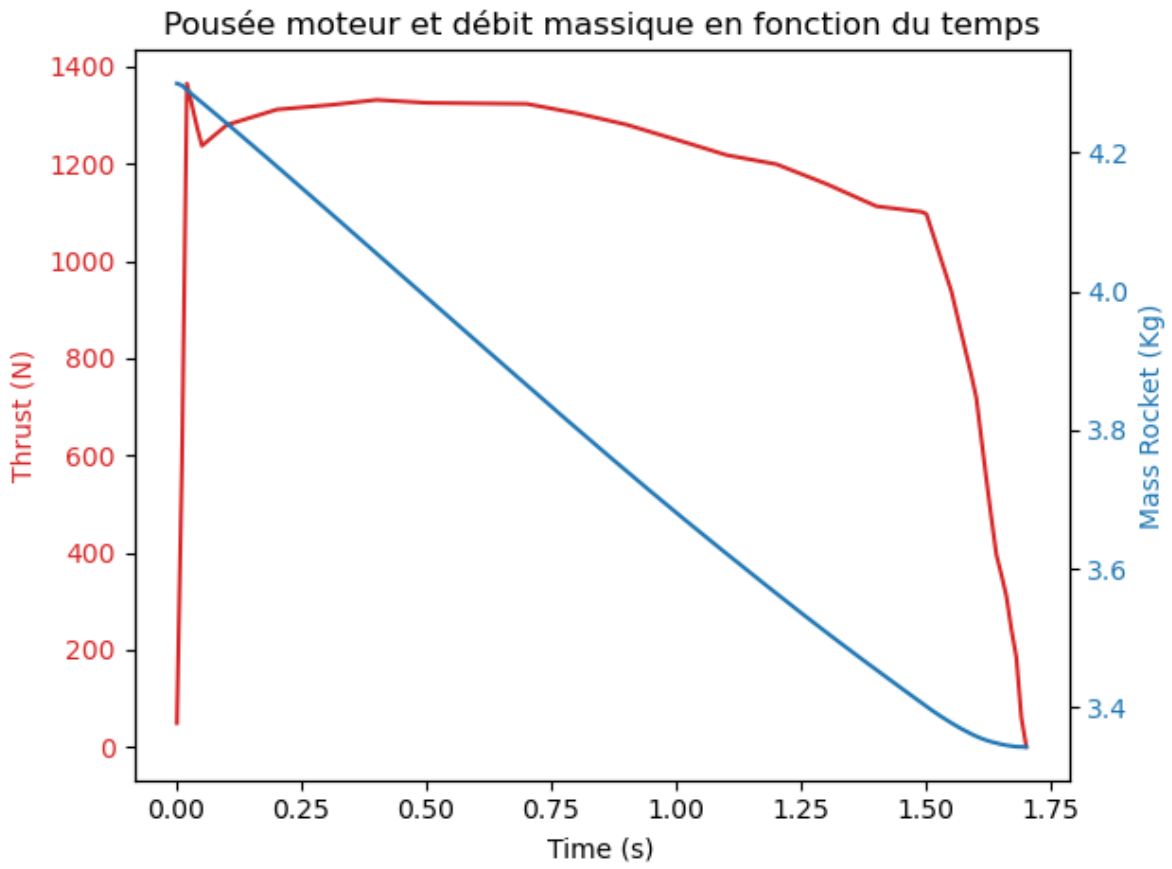
In [16]:
```
# Calcul des interpolations pour chaque pas de temps (s)
time = np.arange(0.001, MaxTimeThrust + pas, pas)

#  Calcul du Thrust (N)
thrust = [calculate_thrust(t) for t in time]
thrust_array = np.array(thrust)

#  Calcul de l'évolution du Flow (Kg/s)
flow = thrust_array / (g * ISP)

# Cacul de la masse
rocketMass = np.zeros(len(time))
rocketMass[0] = mass
for i in range(1, len(flow)):
    rocketMass[i] = rocketMass[i-1] - (flow[i]*pas)

# Création d'un DataFrame "df" :
df = pd.DataFrame({
    "Time (s)": time,
    "Thrust (N)": thrust,
    "Flow (Kg/s)" : flow,
    "Mass Rocket (Kg)" : rocketMass
})
#df.to_csv('Thrust.csv')
df.describe()
```

Out[16]:

	Time (s)	Thrust (N)	Flow (Kg/s)	Mass Rocket (Kg)
count	171.000000	171.000000	171.000000	171.000000
mean	0.851000	1174.055643	0.559702	3.784333
std	0.495076	266.115350	0.126864	0.294164
min	0.001000	0.000000	0.000000	3.343143
25%	0.426000	1160.362350	0.553175	3.522084
50%	0.851000	1274.160400	0.607425	3.771487
75%	1.276000	1320.246867	0.629396	4.038680
max	1.701000	1365.011667	0.650736	4.300000

In [17]:
```
# Affichage des deux graphiques sur deux axes distincts
fig, ax1 = plt.subplots()

color = 'tab:red'
ax1.set_xlabel('Time (s)')
ax1.set_ylabel('Thrust (N)', color=color)
ax1.plot(df["Time (s)"], df["Thrust (N)"], color=color)
```

```
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()

color = 'tab:blue'
ax2.set_ylabel('Mass Rocket (Kg)', color=color)
ax2.plot(df["Time (s)"], df["Mass Rocket (Kg)"], color=color)
ax2.tick_params(axis='y', labelcolor=color)

plt.title('Pousée moteur et débit massique en fonction du temps')

fig.tight_layout()
plt.show()
```



Vol oblique 1D - Z

In [18]:

```
# Fonction d'interpolation linéaire de la poussée
def calculate_thrust(t):
    data = th[["Time (s)", "Thrust (N)"]].to_records(index=False)
    # Parcourir toutes les paires de points de données consécutives
    for i in range(len(data) - 1):
        # Récupérer les temps et les poussées pour les points de données actuels et
        t0, thrust0 = data[i]
        t1, thrust1 = data[i + 1]

        # Vérifier si le temps demandé est compris entre les temps des points de do
        if t0 <= t <= t1:
            # Calculer la poussée interpolée à l'aide de l'interpolation linéaire e
            thrust = thrust0 + (thrust1 - thrust0) * (t - t0) / (t1 - t0)
            # Retourner la poussée interpolée pour le temps demandé
            return thrust

    # Retourner 0 si le temps demandé est en dehors de la plage des données fournie
    return 0
```

In [19]:
```python
def calculate_drag(v, h):

    # Constante:
    Sref = 0.00361173 # Surface de référence, représentant la dimension du corps de
    rhoi = 1.292  # Densité de l'air initial (kg/m3)

    # Calcul du Coefficient de la densité de l'air (valable jusqu'à 11km) :
    #rho = rhoi * ((2000-h)/(2000+h))
    rho = rhoi
    # Chercher le coefficient d'aérodynamique correspondant à la vitesse donnée
    Coeff = None
    for i in range(len(dg)):
        if dg['velocity'][i] >= v:
            if i == 0:
                Coeff = dg['Cd Rocket - (Unpowered)'][i]
            else:
                # Interpolation linéaire
                x1, x2 = dg['velocity'][i-1:i+1]
                y1, y2 = dg['Cd Rocket - (Unpowered)'][i-1:i+1]
                Coeff = y1 + (v-x1)*(y2-y1)/(x2-x1)
            break

    # Si Coeff est toujours None, attribuer la dernière valeur connue du coefficien
    if Coeff is None:
        Coeff = dg['Cd Rocket - (Unpowered)'].iloc[-1]

    # Calcul de la force de traînée en Newtons
    DfDrag = Sref * 0.5 * rho * Coeff * math.pow(v, 2)

    return DfDrag
```

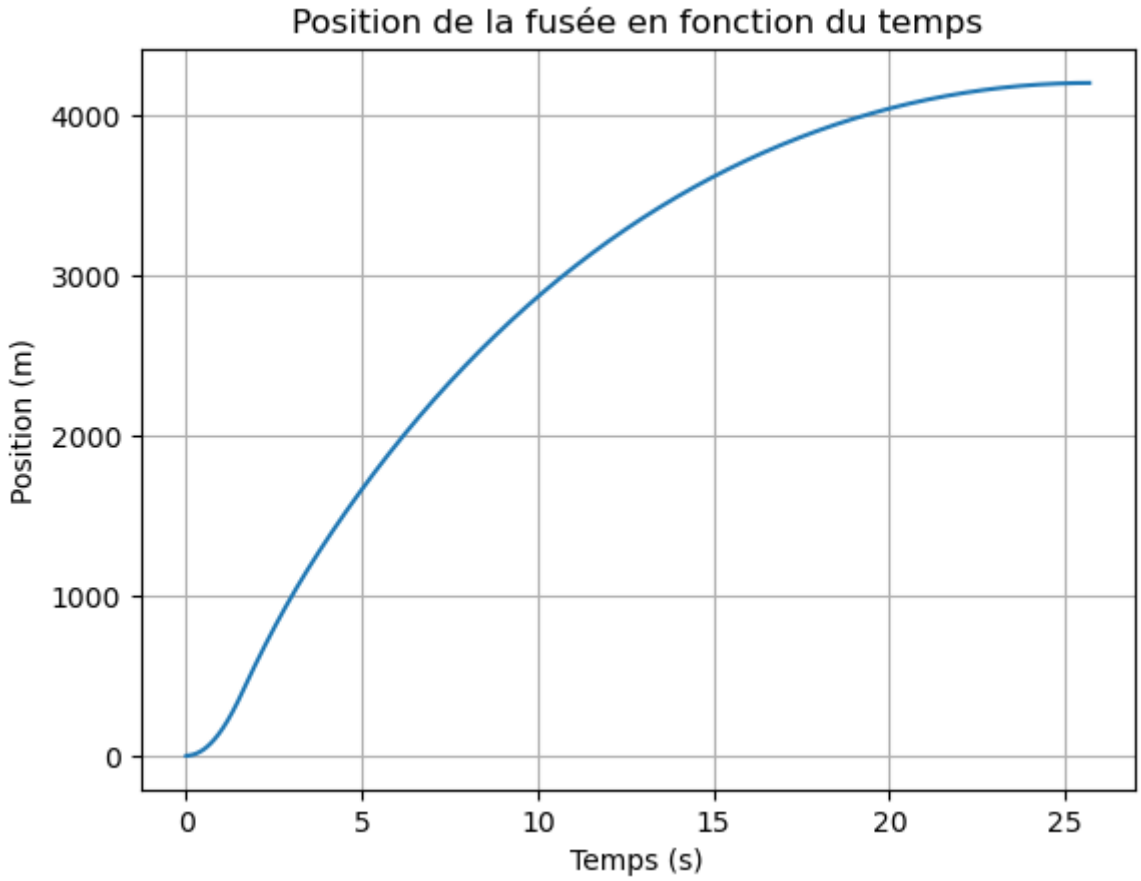
In [20]:
```python
# Calcule la variation de la constante de pesanteur en fonction de la hauteur z
def variation_gravite(z):

    # Constante :
    R = 6371000  # Rayon moyen de la Terre en mètres
    go = 9.80665   # Valeur moyenne de la constante de pesanteur à la surface de la

    # Calcul :
    g = go * ((R / (R + z)) ** 2)
    variation = go - g
    return g
```

In [21]:
```python
compute = True

if compute == True:
    # Paramètres de simulation:
    pas = 0.01 # Pas de la simulation en secondes
    temps_max = 35 # Temps maximal pour la simulation en secondes

    # Constantes:
    g0 = 9.80665 # Accélération de la pesanteur (m/s2) à la surface terrestre
    ISP = 213.9 # Impulsion Spécifique PRO-54 White Thunder (s)

    # Paramètres de vol:
    mi = 4.300 # Masse initial de la fusée (Kg)

    ai = 0 # Accélération initiale (m/s²)
    vi = 0 # Vitesse initiale (m/s)
    mci = 0 # Vitesse initiale (mach)
    zi = 0 # Position initiale (m)
```

```
pi = 0 # Pouseé moteur initiale (N)
ri = 0 # Trainée fusée initiale (N)
di = 0 # Débit moteur initiale (Kg/s)


# Variables pour sauvegarder les paramètres
positions = [zi]
vitesses = [vi]
mach = [mci]
masses = [mi]
poussees = [pi]
trainees = [ri]
accelerations = [ai]
time = [0]

temps = 0

while temps < temps_max:

    #print("Temps fusée (s):", temps)
    time.append(temps)

    # Phase n°1 (pi, di, mi):
    if temps == 0:
        pi = calculate_thrust(pas)
        di = pi / (g0 * ISP)
        mi = mi - (di * pas)
    else:
        pi = calculate_thrust(temps)
        di = pi / (g0 * ISP)
        mi = mi - (di * pas)

    #print("Pousée moteur (N):", pi)
    #print("Débit moteur initiale (Kg/s):", di)
    #print("masse fusée (kg):", mi)
    poussees.append(pi)
    masses.append(mi)

    # Phase n°2 (gi, ai):
    gi = variation_gravite(zi)
    ai = ((pi - ri) / mi) - gi

    #print("Accélération de la pesanteur (m/s2):", gi)
    #print("Accélération fusée (m/s²):", ai)
    #print("Accélération fusée (g):", ai/g)
    accelerations.append(ai)

    # Phase n°3
    vi = vi + (ai * pas)
    mci = vi / 340
    #print("Vitesse fusée (m/s):", vi)
    #print("Vitesse fusée (MACH)", mci)
    vitesses.append(vi)
    mach.append(mci)

    # Phase n°4
    zi = zi + (vi * pas) + ((1/2) * ai * (pas ** 2))
    #print("Position fusée (m):", zi)
    positions.append(zi)

    # Phase n°5
    ri = calculate_drag(vi, zi)
    #print("Trainée fusée (N):", ri,  "\n")
    trainees.append(ri)
```

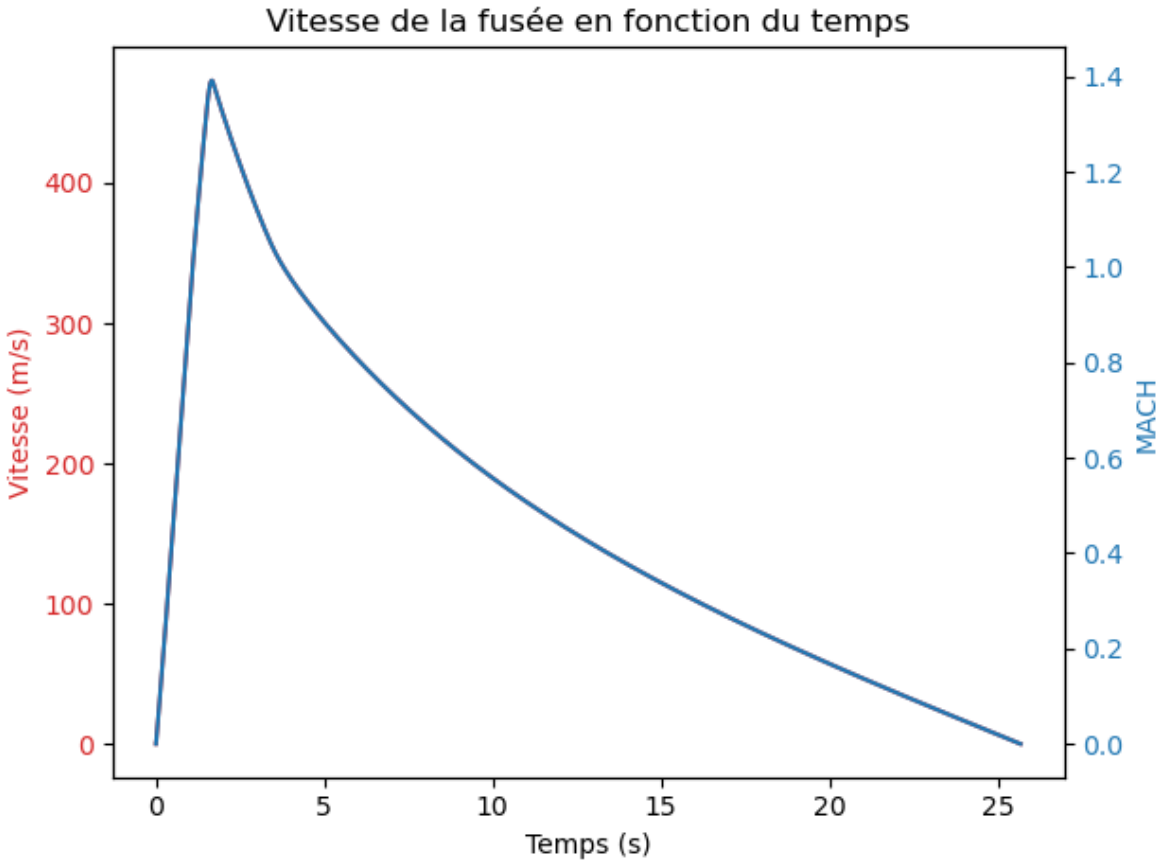
```
        # Vérifie si l'apogée est atteinte (vitesse verticale nulle ou négative)
        if vi <= 0:
            break

        temps += pas

    # Conversion des listes en tableaux numpy pour une manipulation plus facile
    positions = np.array(positions)
    vitesses = np.array(vitesses)
    mach = np.array(mach)
    masses = np.array(masses)
    poussees = np.array(poussees)
    trainees = np.array(trainees)
    accelerations = np.array(accelerations)
    time = np.array(time)
```

In [22]:
```
if compute == True:
    # Affichage des positions
    plt.figure()
    plt.plot(time, positions)
    plt.xlabel("Temps (s)")
    plt.ylabel("Position (m)")
    plt.title("Position de la fusée en fonction du temps")
    plt.grid(True)

    # Affichage
    plt.show()
```

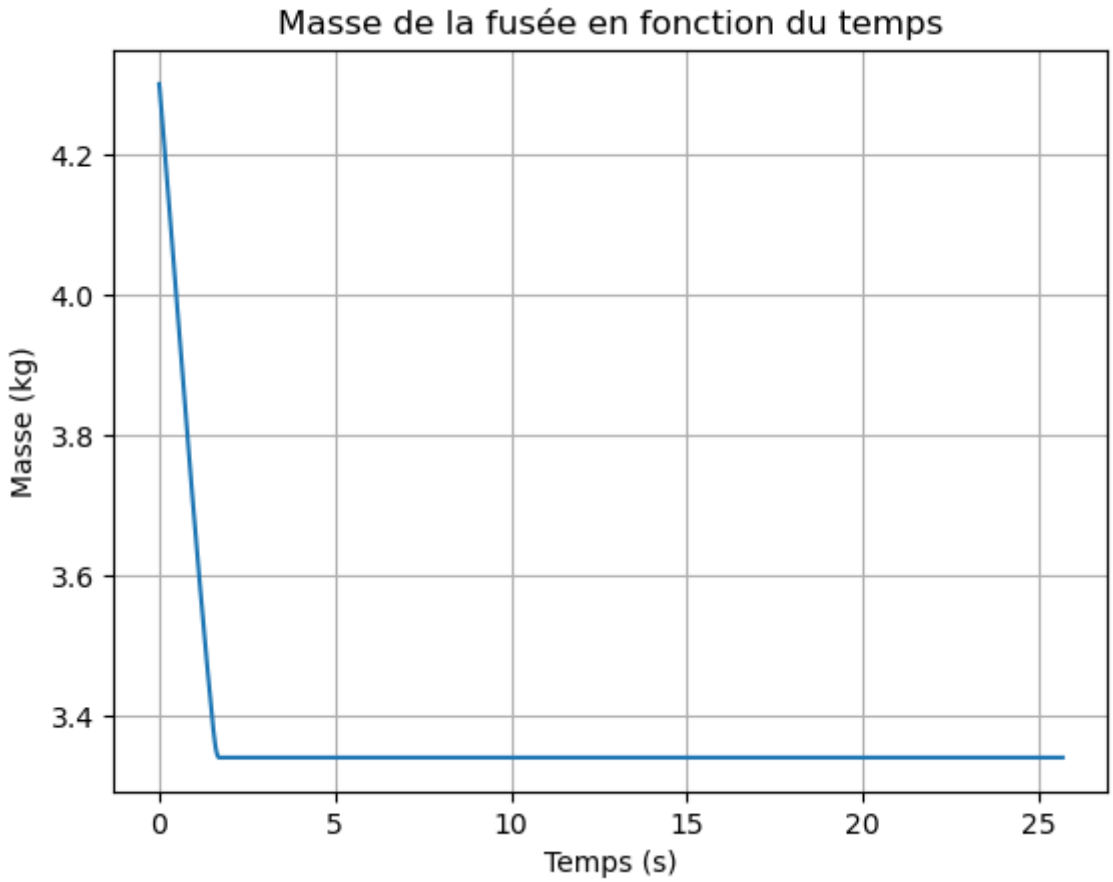


In [23]:
```
if compute == True:
    # Affichage des deux graphiques sur deux axes distincts
    fig, ax1 = plt.subplots()

    color = 'tab:red'
    ax1.set_xlabel('Temps (s)')
```

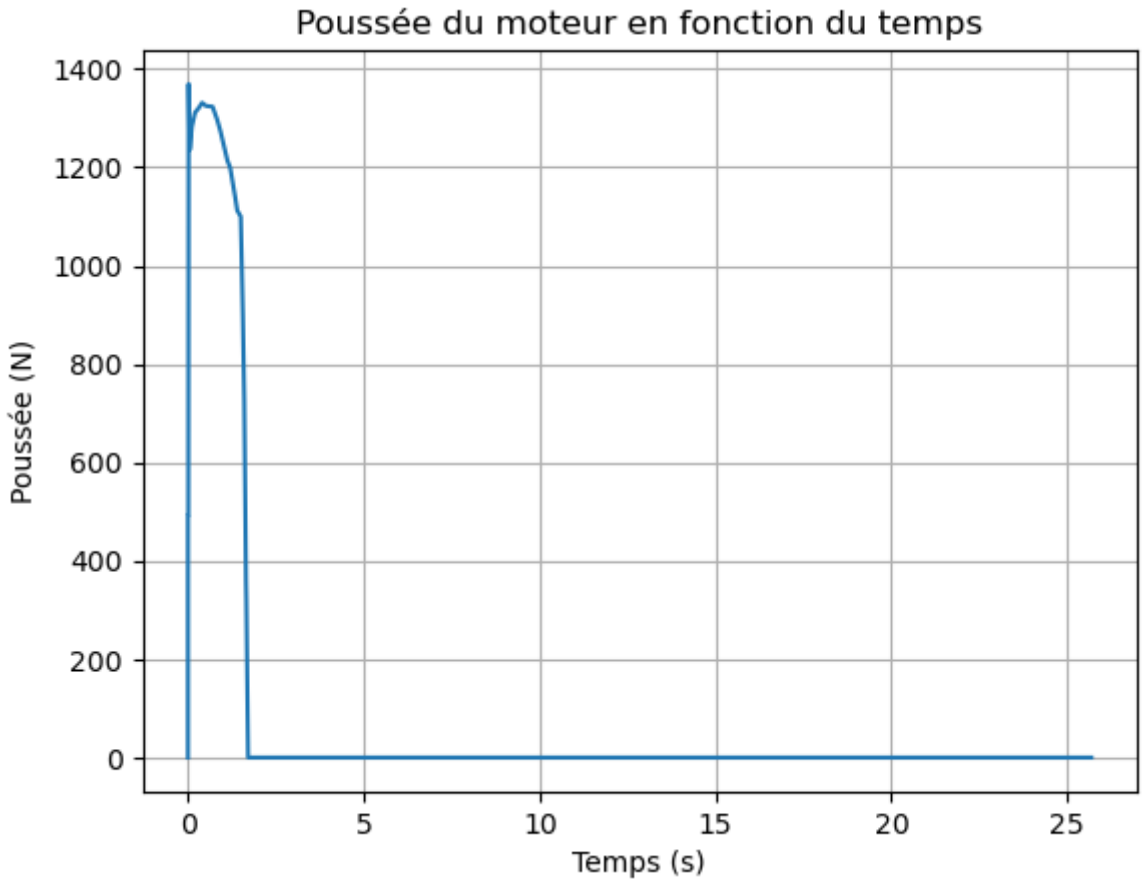
```
    ax1.set_ylabel('Vitesse (m/s)', color=color)
    ax1.plot(time, vitesses, color=color)
    ax1.tick_params(axis='y', labelcolor=color)

    ax2 = ax1.twinx()

    color = 'tab:blue'
    ax2.set_ylabel('MACH', color=color)
    ax2.plot(time, mach, color=color)
    ax2.tick_params(axis='y', labelcolor=color)

    plt.title("Vitesse de la fusée en fonction du temps")

    fig.tight_layout()
    plt.show()
```



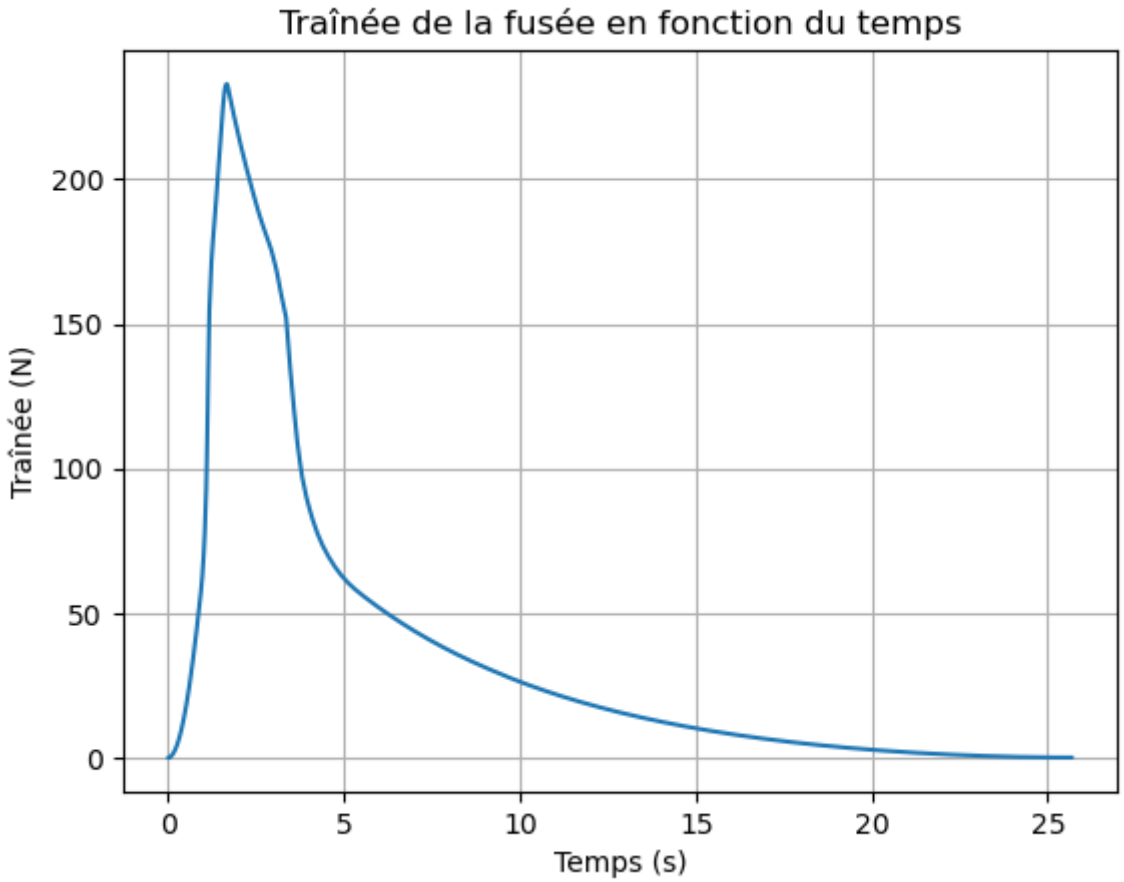
In [24]:
```
if compute == True:
    # Affichage des masses
    plt.figure()
    plt.plot(time, masses)
    plt.xlabel("Temps (s)")
    plt.ylabel("Masse (kg)")
    plt.title("Masse de la fusée en fonction du temps")
    plt.grid(True)

    # Affichage
    plt.show()
```



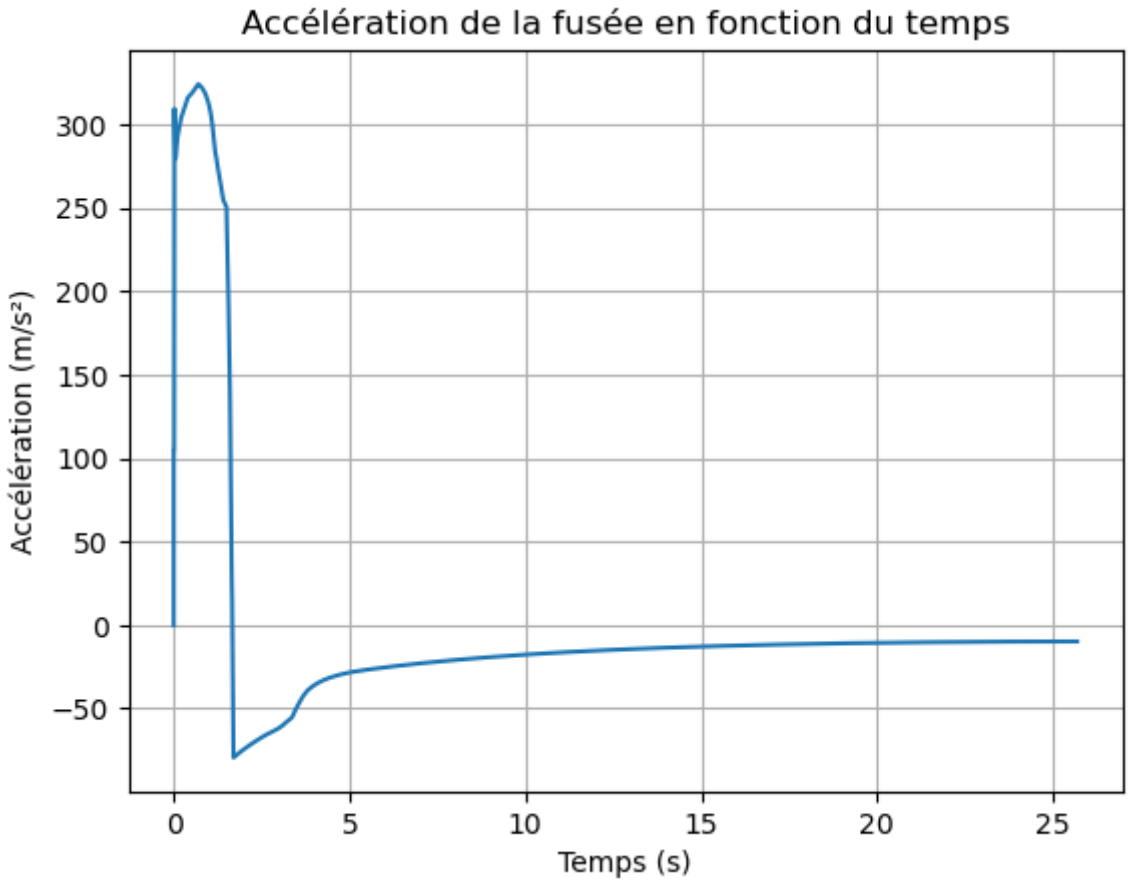
In [25]:
```python
if compute == True:
    # Affichage des poussées
    plt.figure()
    plt.plot(time, poussees)
    plt.xlabel("Temps (s)")
    plt.ylabel("Poussée (N)")
    plt.title("Poussée du moteur en fonction du temps")
    plt.grid(True)

    # Affichage
    plt.show()
```



In [26]:
```python
if compute == True:
    # Affichage des traînées
    plt.figure()
    plt.plot(time, trainees)
    plt.xlabel("Temps (s)")
    plt.ylabel("Traînée (N)")
    plt.title("Traînée de la fusée en fonction du temps")
    plt.grid(True)

    # Affichage
    plt.show()
```



In [27]:
```python
if compute == True:
    # Affichage des accélérations
    plt.figure()
    plt.plot(time, accelerations)
    plt.xlabel("Temps (s)")
    plt.ylabel("Accélération (m/s²)")
    plt.title("Accélération de la fusée en fonction du temps")
    plt.grid(True)

    # Affichage
    plt.show()
```



Vol oblique 2D - Z

In [28]:

```
# Paramètres de simulation:
pas = 0.01 # Pas de la simulation en secondes
temps_max = 90 # Temps maximal pour la simulation en secondes

# Constantes:
g0 = 9.80665 # Accélération de la pesanteur (m/s2) à la surface terrestre
ISP = 213.9 # Impulsion Spécifique PRO-54 White Thunder (s)

# Paramètres de vol:
temps = 0
angi = math.radians(85) # Angle de décollage en degré
mi = 4.3 # Masse initial de la fusée (Kg)
vi = 0 # Vitesse initiale (m/s)
mci = 0 # Vitesse initiale (mach)
di = 0 # Débit moteur initiale (Kg/s)
ri = 0 # Trainée fusée initiale (N)
pi = 0 # Pouseé moteur initiale sur le repère de la fusée (N)

ax = 0 # Accélération initiale sur l'axes X (m/s²)
vx = 0 # Vitesse initiale sur l'axes X (m/s)
mcx = 0 # Vitesse initiale sur l'axes X (mach)
zx = 0 # Position initiale ur l'axes X (m)

az = 0 # Accélération initiale sur l'axes Z (m/s²)
vz = 0 # Vitesse initiale sur l'axes Z (m/s)
mcz = 0 # Vitesse initiale sur l'axes Z (mach)
zz = 0 # Position initiale ur l'axes Z (m)

# Variables pour sauvegarder les paramètres

accelerations_X = [ax]
positions_X = [zx]
```

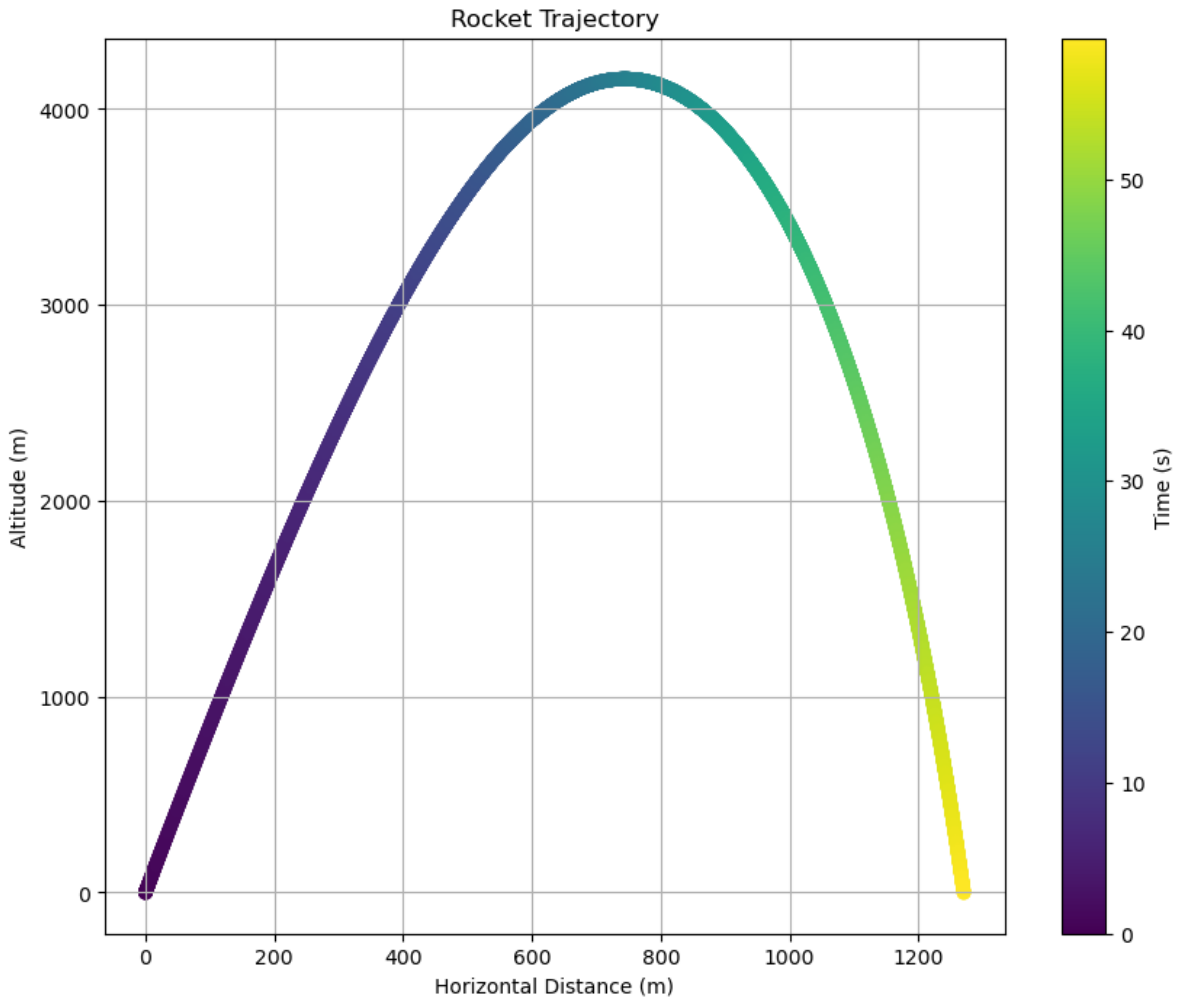
```
vitesses_X = [vx]
mach_X = [mcx]

accelerations_Z = [az]
positions_Z = [zz]
vitesses_Z = [vz]
mach_Z = [mcz]

trainees = [ri]
vitesses = [vi]
mach = [mci]
poussees = [pi]
masses = [mi]
angle = [angi]
time = [temps]

while temps < temps_max:

    #print("Temps fusée (s):", temps)
    time.append(temps)

    # Phase n°1 (pi, di, mi):
    if temps == 0:
        pi = calculate_thrust(pas)
        di = pi / (g0 * ISP)
        mi = mi - (di * pas)
    else:
        pi = calculate_thrust(temps)
        di = pi / (g0 * ISP)
        mi = mi - (di * pas)

    #print("Pousée moteur (N):", pi)
    #print("Débit moteur initiale (Kg/s):", di)
    #print("masse fusée (kg):", mi)
    poussees.append(pi)
    masses.append(mi)

    # Phase n°2 (gi, ai):
    gi = variation_gravite(zz)
    ax = ((pi - ri) / mi) * math.cos(angi)
    az = (((pi - ri) / mi) * math.sin(angi)) - gi

    #print("Accélération de la pesanteur  (m/s2):", gi)
    #print("Accélération fusée sur l'axes X (m/s²):", ax)
    #print("Accélération fusée sur l'axes Z (m/s²):", az)
    #print("Accélération fusée sur l'axes X (g):", ax/g)
    #print("Accélération fusée sur l'axes Z (g):", az/g)
    accelerations_Z.append(az/g)
    accelerations_X.append(ax/g)

    # Phase n°3
    vx = vx + (ax * pas)
    vz = vz + (az * pas)
    mcx = vx / 340
    mcz = vz / 340
    #print("Vitesse fusée sur l'axes X (m/s):", vx)
    #print("Vitesse fusée sur l'axes Z (m/s):", vz)
    #print("Vitesse fusée sur l'axes X (MACH)", mcx)
    #print("Vitesse fusée sur l'axes Z (MACH)", mcz)
    vitesses_Z.append(vz)
    vitesses_X.append(vx)
    mach_Z.append(mcz)
    mach_X.append(mcx)
```

```
    # Phase n°4
    zx = zx + (vx * pas) + ((1/2) * ax * (pas ** 2))
    zz = zz + (vz * pas) + ((1/2) * az * (pas ** 2))
    #print("Position fusée sur l'axes X (m):", zx)
    #print("Position fusée sur l'axes Z (m):", zz)
    positions_Z.append(zz)
    positions_X.append(zx)


    # Phase n°5
    vi = math.sqrt(pow(vx,2)+pow(vz,2))
    mci = vi / 340
    #print("Vitesse fusée sur l'axes de la fusée (m/s):", vi)
    #print("Vitesse fusée sur l'axes de la fusée (MACH)", mci)
    vitesses.append(vi)

    # Phase n°6
    ri = calculate_drag(vi, zz)
    angi = math.atan(vz/vx)

    #print("Trainée fusée (N):", ri)
    #print("Angle fusée (rad):", angi)
    #print("Angle fusée (°):", math.degrees(angi))
    #print("\n")
    trainees.append(ri)
    angle.append(math.degrees(angi))


    # Vérifie si l'apogée est atteinte (vitesse verticale nulle ou négative)
    if zz <= 0:
        break

    temps += pas

# Conversion des listes en tableaux numpy pour une manipulation plus facile
accelerations_X = np.array(accelerations_X)
positions_X = np.array(positions_X)
vitesses_X = np.array(vitesses_X)
mach_X = np.array(mach_X)

accelerations_Z = np.array(accelerations_Z)
positions_Z = np.array(positions_Z)
vitesses_Z = np.array(vitesses_Z)
mach_Z = np.array(mach_Z)

trainees = np.array(trainees)
vitesses = np.array(vitesses)
mach = np.array(mach)
poussees = np.array(poussees)
masses = np.array(masses)
angle = np.array(angle)
time = np.array(time)
```

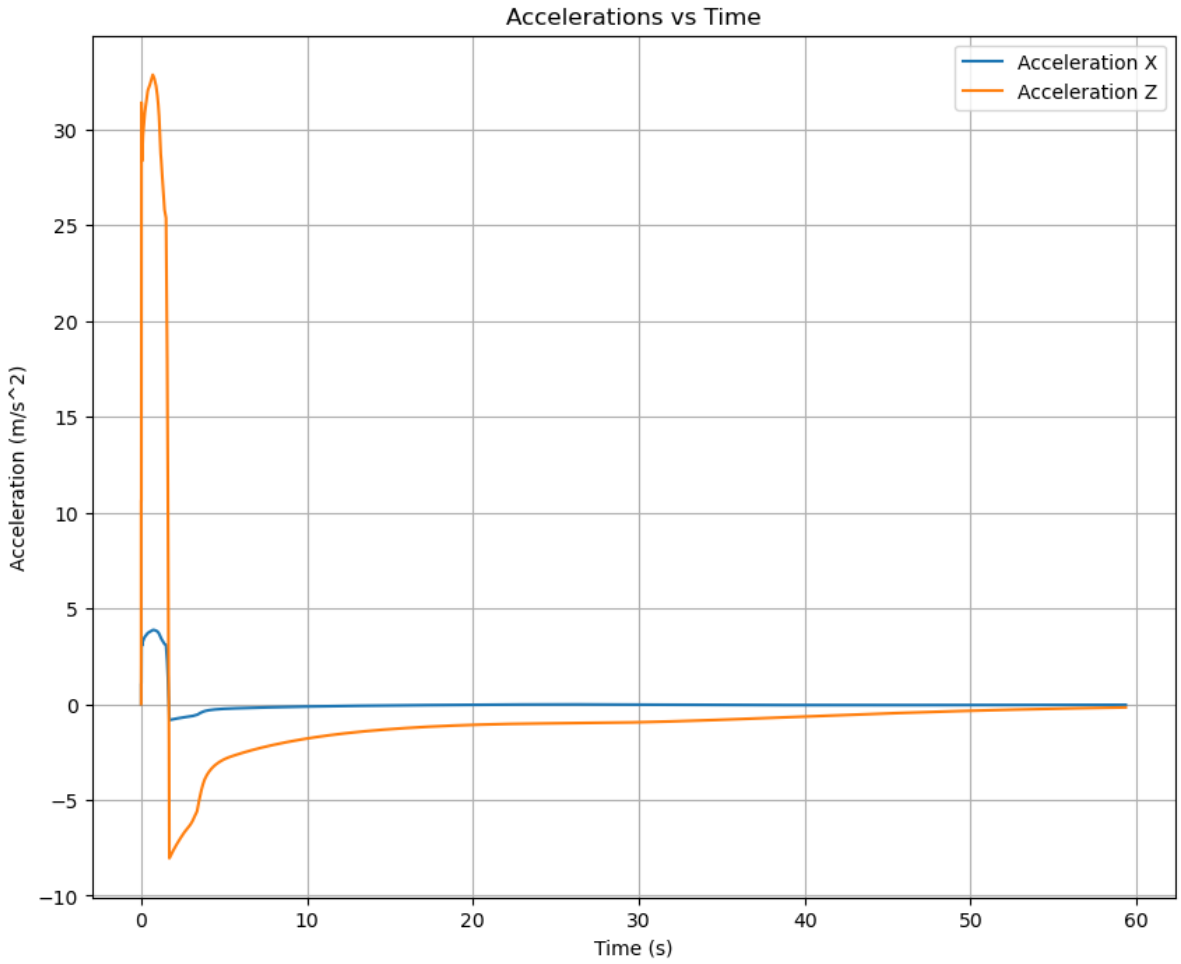
In [29]:

```
# Création d'un scatter plot avec le temps comme couleur
plt.figure(figsize=(10, 8))
scatter = plt.scatter(positions_X, positions_Z, c=time, cmap='viridis')

# Ajout d'une colorbar pour indiquer le temps
plt.colorbar(scatter, label='Time (s)')

plt.title('Rocket Trajectory')
plt.xlabel('Horizontal Distance (m)')
plt.ylabel('Altitude (m)')
```

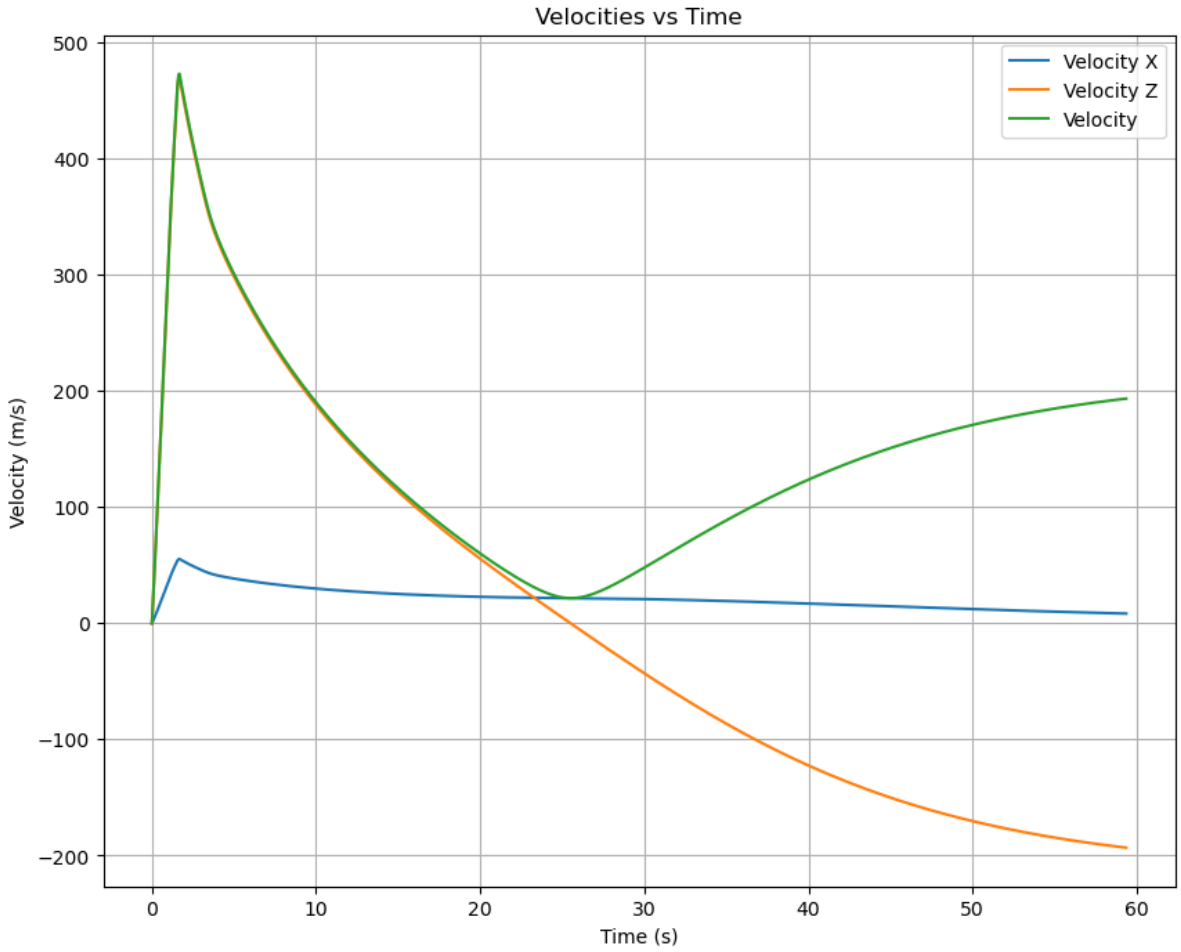
```
plt.grid(True)

plt.show()
```



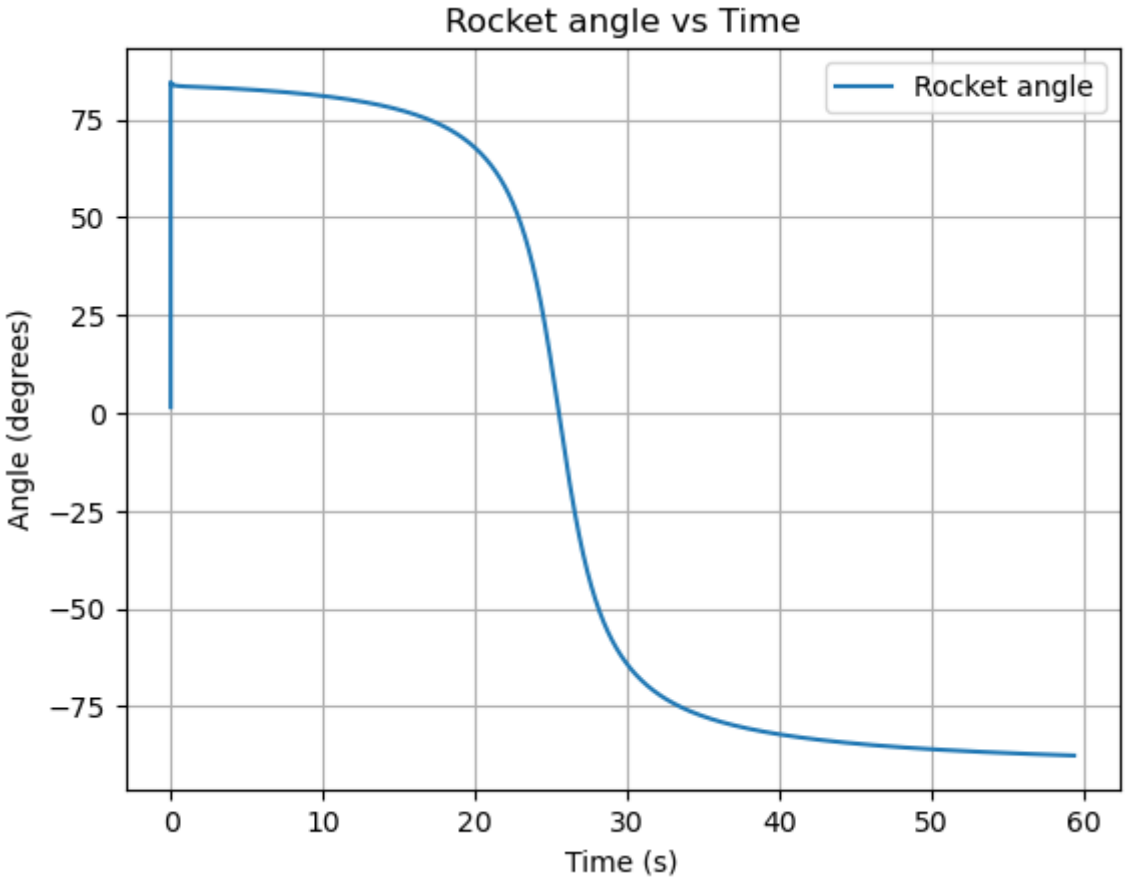
In [30]:
```
plt.figure(figsize=(10, 8))

# Accélération sur l'axe X
plt.plot(time, accelerations_X, label='Acceleration X')

# Accélération sur l'axe Z
plt.plot(time, accelerations_Z, label='Acceleration Z')

plt.title('Accelerations vs Time')
plt.xlabel('Time (s)')
plt.ylabel('Acceleration (m/s^2)')
plt.legend()
plt.grid(True)
plt.show()
```



In [31]:
```
plt.figure(figsize=(10, 8))

# Vitesse sur l'axe X
plt.plot(time, vitesses_X, label='Velocity X')

# Vitesse sur l'axe Z
plt.plot(time, vitesses_Z, label='Velocity Z')

# Vitesse sur l'axe de la fusée
plt.plot(time, vitesses, label='Velocity')

plt.title('Velocities vs Time')
plt.xlabel('Time (s)')
plt.ylabel('Velocity (m/s)')
plt.legend()
plt.grid(True)
plt.show()
```



In [32]:
```
# Angle de la fusée
plt.plot(time, angle, label='Rocket angle')

plt.title('Rocket angle vs Time')
plt.xlabel('Time (s)')
plt.ylabel('Angle (degrees)')
plt.legend()
plt.grid(True)
plt.show()
```


Rocket angle vs Time
Rocket angle
75
50
25
0
−25
−50
−75
Angle (degrees)
0
10
20
30
40
50
60
Time (s)