LEOFLY

ZENITH – 1

C’Space 2025 [MF31]

DOCUMENTATION

RAPPORT FINAL

Ce document contient les rapports et études menées lors de la réalisation du projet Zenith-1.

Langue : FRANCAIS

18/07/2025

Table des Matières

1 – Présentation du projet

Ce projet de mini-fusée Zenith-1 est le premier projet de la branche nantaise de LéoFly, qui a été établie en septembre 2024. Passionnés par la découverte du monde aérospatial, ce projet technique est le premier de cette échelle que nous réalisons. Nous sommes une équipe de sept étudiants, dont cinq en deuxième année et deux en troisième année.

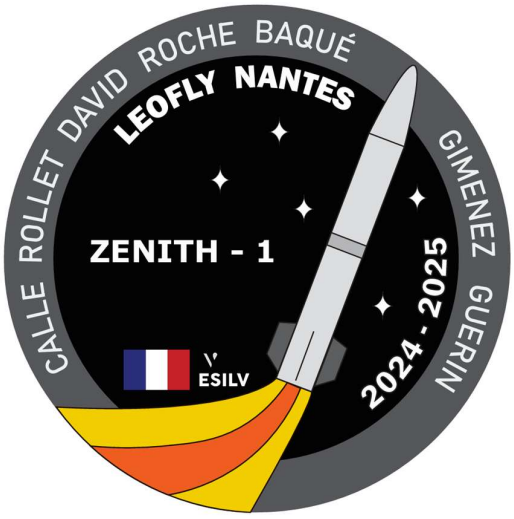
Nous remercions Mathis Pille qui nous a permis de commencer LéoFly à l'ESILV Nantes, ce fut une première année pleine de succès, et motivante pour les projets à venir. Nous remercions également notre directeur technique Tom Alglave, et Matthieu Pecoraro, le président actuel de l'association, qui a fait le suivi de ce projet dans ses dernières étapes.

*

* *

*

Evan CALLE
Alexandre ROLLET
Aurélien ROCHE
Léo GUERIN
Baptiste GIMENEZ
Anne-Linh DAVID
Emile BAQUE



2 – Caractéristiques

2.1 Structure

2.1.1 Fiche de dimensionnement et caractéristiques

Masse totale (sans propulsion) :		**1158 g**
dont 2 batteries de 30g		
Centre de masse :		**$\approx$ 519 mm ($du\ haut$)**
Longueur totale :		**948 mm**
Largeur totale (avec ailerons) :		**404 mm**
Tube	**Longueur :**	748 mm
	Diamètre :	84 mm
Ogive	**Longueur :**	200 mm
	Diamètre :	84 mm
	Surface :	36 107 mm^2
Ailerons	**Emplanture :**	80 mm
	Saumon :	40 mm
	Flèche :	20 mm
	Envergure :	140 mm
	Epaisseur :	3 mm
	Nombre :	4
	Position du bas :	948 mm

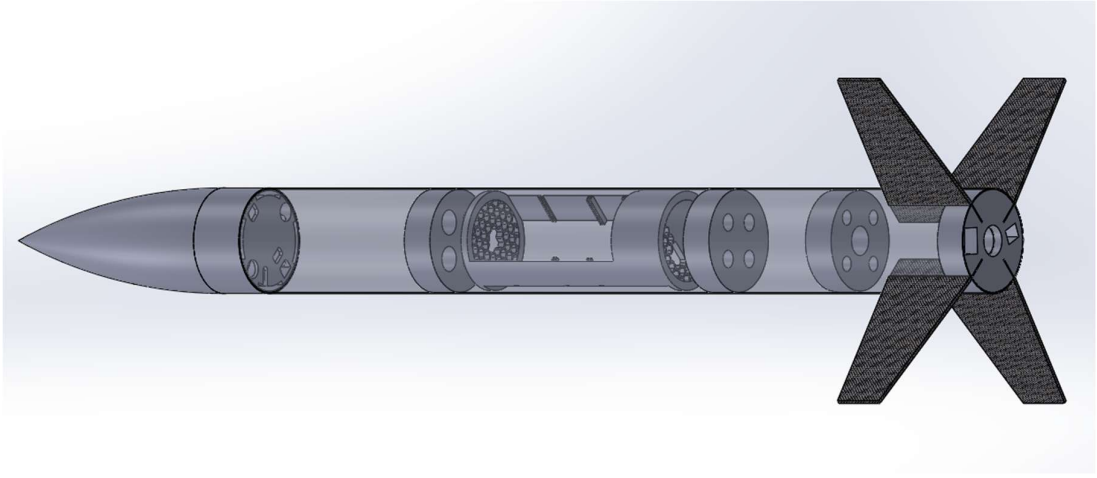


Fig. 1 Modélisation SolidWorks (approx.)

2.1.2 Schéma de la structure générale

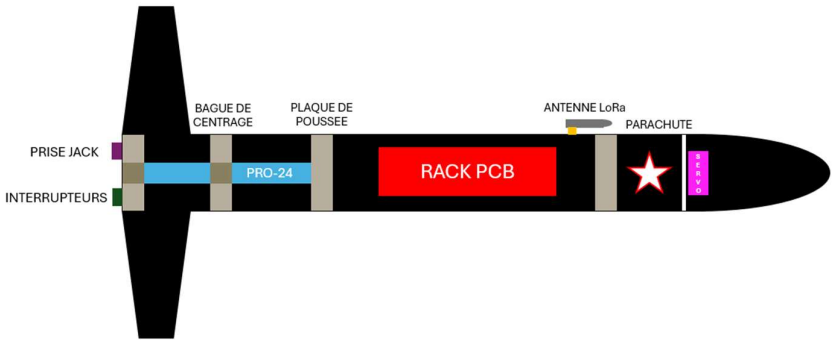


Fig. 2 Structure générale de la miniF

2.1.3 Ogive

Nous avons choisi l'ogive tangentielle comme coiffe pour Zénith-1 car c'est une forme classique d'ogive qui est surtout facile à modéliser par le calcul et donc en logiciel de CAO.

Equation de la courbe de l'ogive :

$$f(x) = \sqrt{\left(\frac{10441}{21}\right)^2 - x^2} - \frac{10441}{21} + 42$$

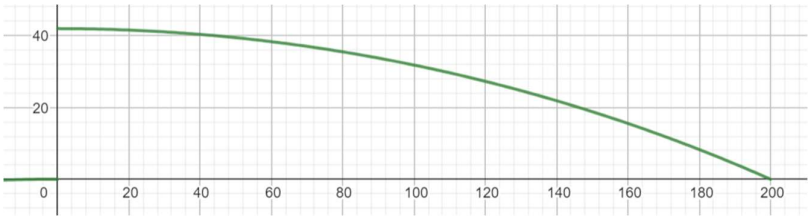


Fig. 3 Equation de l'ogive représentée dans GeoGebra

Puis, nous avons créé l'ogive dans CATIA en faisant une rotation de cette fonction autour d'un axe défini, ce qui donne une forme pleine, la forme de l'ogive en 3 dimensions :

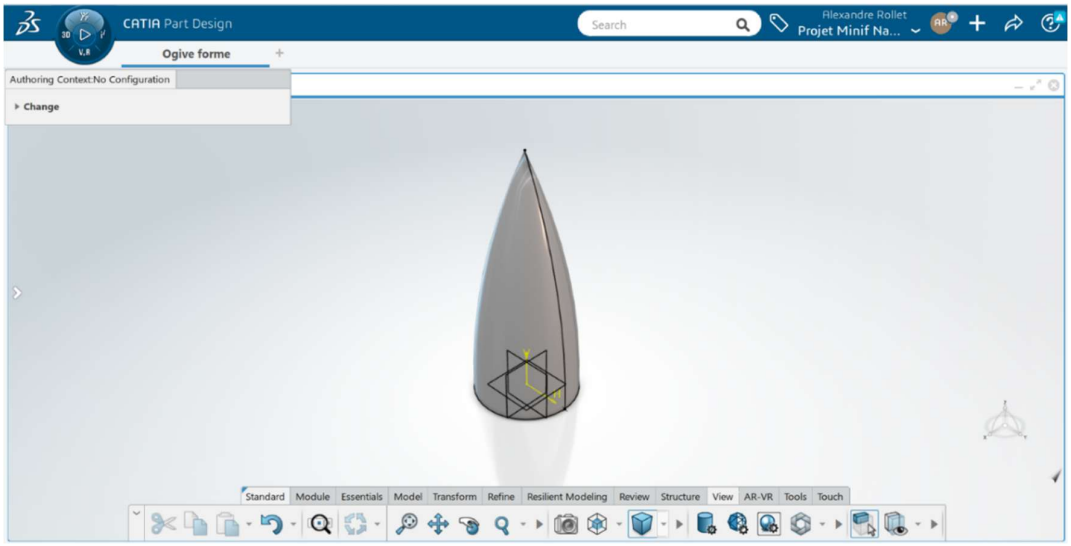


Fig. 4 Ogive Tangentielle modélisée dans CATIA

Nous pouvons aussi déterminer la surface de cette ogive, qui est finalement la surface de contact direct entre la fusée et l'air incident. L'aire de révolution d'une courbe autour d'un axe s'écrit, en prenant ici l'axe vertical z :

$$S_{ogive} = 2\pi \int_{z_1=0}^{z_2=200} f(z)\sqrt{1+\left(f'(z)\right)^2}\,dz$$

En approximant l'intégrale numériquement dans MatLab, on obtient une surface de $36107\ mm^2 \equiv 361{,}07\ cm^2$.

2.1.4 Bas de la miniF

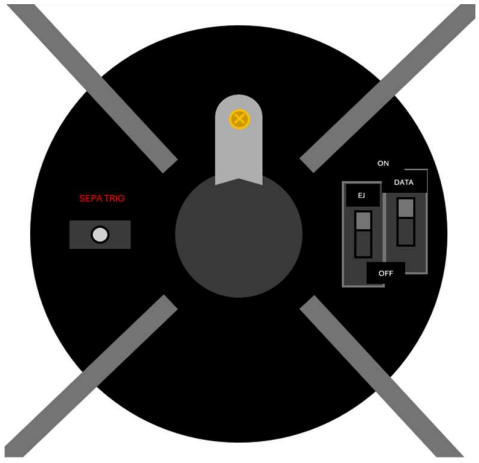


Fig. 5 Bas de la miniF

Les systèmes d'allumage de l'électronique et le branchement du jack se font sur le bas de la miniF, ce qui est peu conventionnel mais cela fonctionne tout aussi bien. Il y a tout de même assez de place sur cette surface pour placer les appuis de la rampe de lancement. Le seul risque de ce placement des câbles est que la chaleur du propulseur défasse une soudure qui n'est donc pas très loin. Cependant les câbles sont bien gainés et protégés, et pour le câble reliant la prise jack au système d'éjection, le code C++ a été fait tel que le chronomètre se déclenche en passe bas, autrement dit si ce câble se défait ou brûle, cela revient au fait de tirer le jack au lancement. On évite donc tout risque que la séparation ne s'active pas.

2.2 Attaches

Notre système d'attaches utilise une cordelette de 3mm de diamètre, suffisamment forte pour résister à l'ouverture du parachute qui freine d'un coup sec la fusée. Nous utilisons deux cordelettes : la première relie le parachute à l'ogive, et est très courte, afin que lors de l'éjection le parachute soit rapidement tiré à l'extérieur du tube. La deuxième est bien plus longue, et relie le parachute au tube. Elle doit être plus longue pour que le parachute et l'ogive puisse se séparer suffisamment loin pour assurer que le parachute ait le temps de s'ouvrir proprement. Sinon avec une cordelette courte, l'ogive serait sèchement retenue par celle-ci, irait cogner les flancs du tube, et le parachute n'aurait plus l'occasion d'être tiré vers l'extérieur. De plus, pour ne pas avoir des câbles électroniques trop longs, le servomoteur est connecté au PCB avec une attache avec des pins, permettant aux pins de se retirer facilement lorsque l'ogive est poussée vers l'avant, évitant que les câbles s'arrachent ou se défassent de leurs soudures.

Le parachute est venu avec une goupille en métal attachée à ses fils, et donc nous avons fait passer nos cordelettes dans cette goupille. C'est ici que les deux cordelettes se retrouvent, et donc le parachute tient à la fois le tube et l'ogive. Si on avait relié le parachute à l'ogive, et l'ogive au tube, le parachute retiendrait les deux à la fois et les attaches des cordelettes dans l'ogive auraient pu céder.

Pour que le tube soit solidement attaché au parachute, une cordelette passe à travers une bague en bois située proche de l'entrée tu tube, et cette bague est vissée. Cela assure une bonne rigidité.

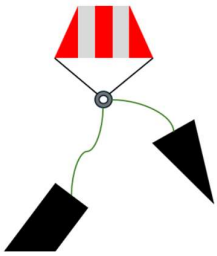


Fig. 6 Attaches reliant le parachute à l'ogive et au tube après séparation

3 – Electronique

3.1 Sur l'ensemble de la minif

L'électronique de la miniF se décline en deux parties majeures, qui ont dû être faites telles qu'elles fonctionnent indépendamment l'une de l'autre :

- La première partie s'occupe du système d'éjection, et est composée d'un PCB. L'électronique contient le code se trouvant dans l'annexe qui permet d'actionner le servomoteur situé dans l'ogive. Un chronomètre contrôle cette action, et sa durée est estimée approximativement par rapport aux temps de vol jusqu'à l'apogée (afin d'avoir une apogée et une séparation concordantes). Ce chronomètre commence une fois que la miniF a quitté la rampe.
- La deuxième partie s'occupe des données du vol, et est composée de deux PCBs. Les données de vol sont stockées sur une carte mini-SD et sont aussi envoyées vers un ordinateur au sol via une antenne LoRa. Ces données permettront de faire un bilan des performances à la suite du vol.

  Le rack électronique qui tient en place les PCBs (cf. 3.3) est le cœur de l'électronique, et c'est à partir d'ici que sont ordonnées les actions cruciales, notamment l'actionnement du servomoteur permettant l'éjection de l'ogive et du parachute ensuite. Ce rack est lié tout d'abord à deux interrupteurs indépendants, chacun activant ses PCBs respectifs selon les deux parties décrites ci-dessus. Il est également lié à une prise jack femelle qui est attachée à un jack mâle situé sur la rampe, ce qui fait qu'au lancement, le débranchement déclenche le décompte d'un chronomètre.

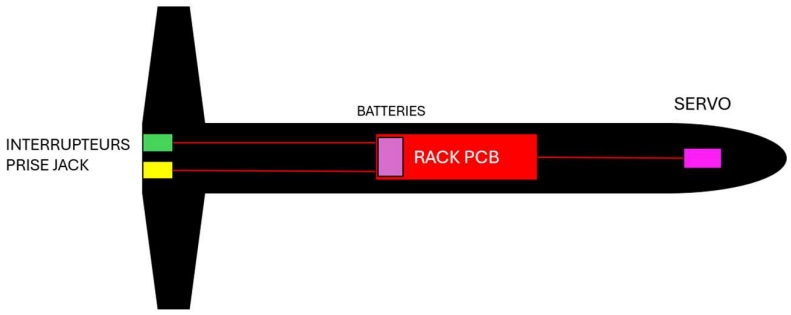


Fig. 7 Schéma général et simplifié de l'électronique

Les batteries utilisées sont des batteries 7,4V 450mah.

3.2 Schémas de l'électronique

3.2.1 Système d'éjection

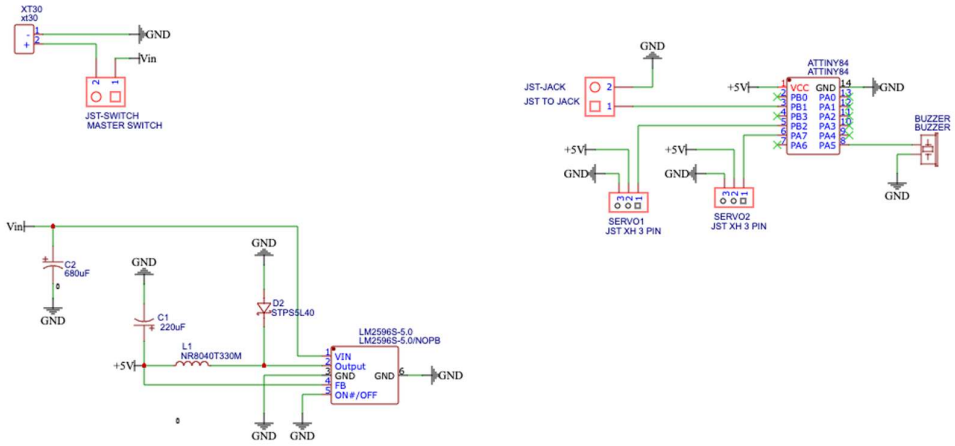


Fig. 8 Schémas électroniques des composants du PCB du système d'éjection

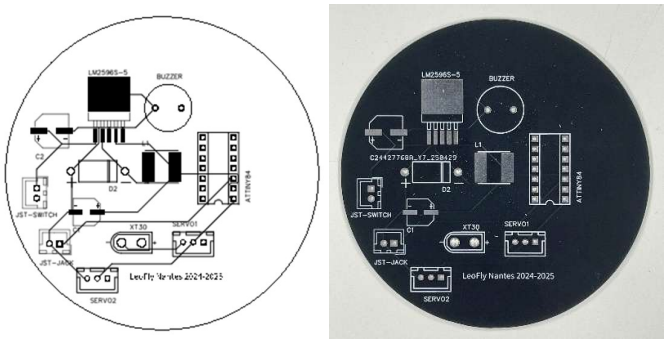


Fig. 9 PCB du système d'éjection

Concernant l'attache qui relie ce PCB au servomoteur placé dans la coiffe, nous avons utilisé un câblage classique Dupont, ce qui permet une connexion assez forte avant la séparation entre les câbles, mais cela permet aussi à cette attache de se défaire facilement lors de la séparation. En effet, ces câbles sont plus courts que la cordelette reliant le tube à la coiffe, car évidemment c'est la cordelette qui doit absorber le choc de la séparation. Utiliser des câbles électroniques trop long entrainerait un risque d'arrachage, et ils pourraient s'emmêler avec la cordelette à l'intérieur du tube.

3.2.2 Expérience

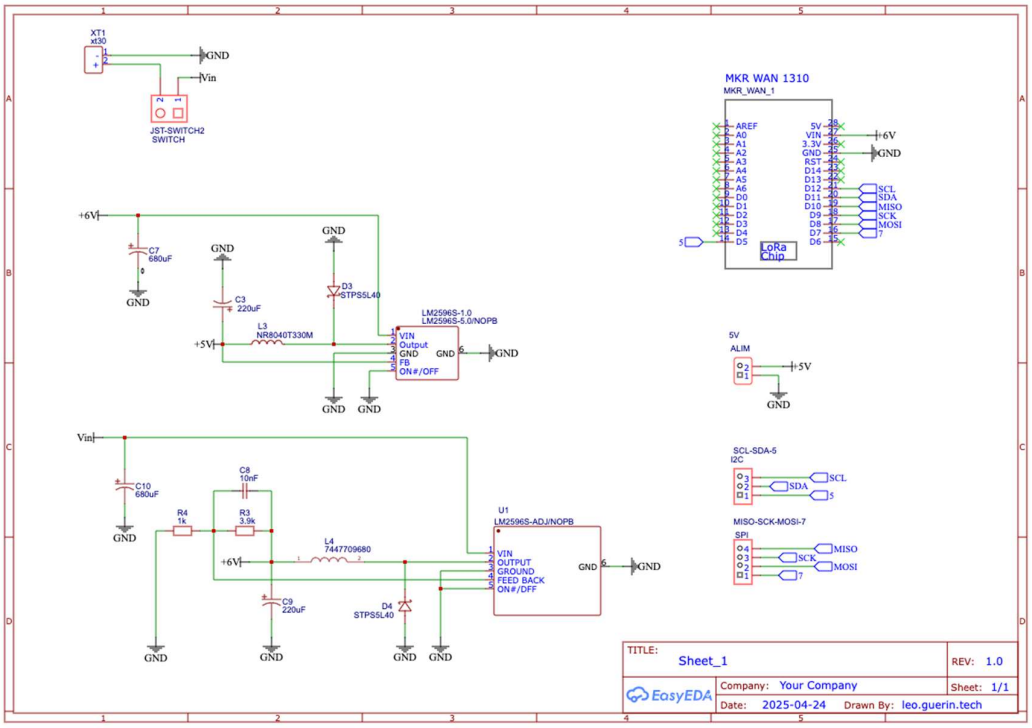


Fig. 10 Schéma électronique de l'étage 1 du système d'expérience

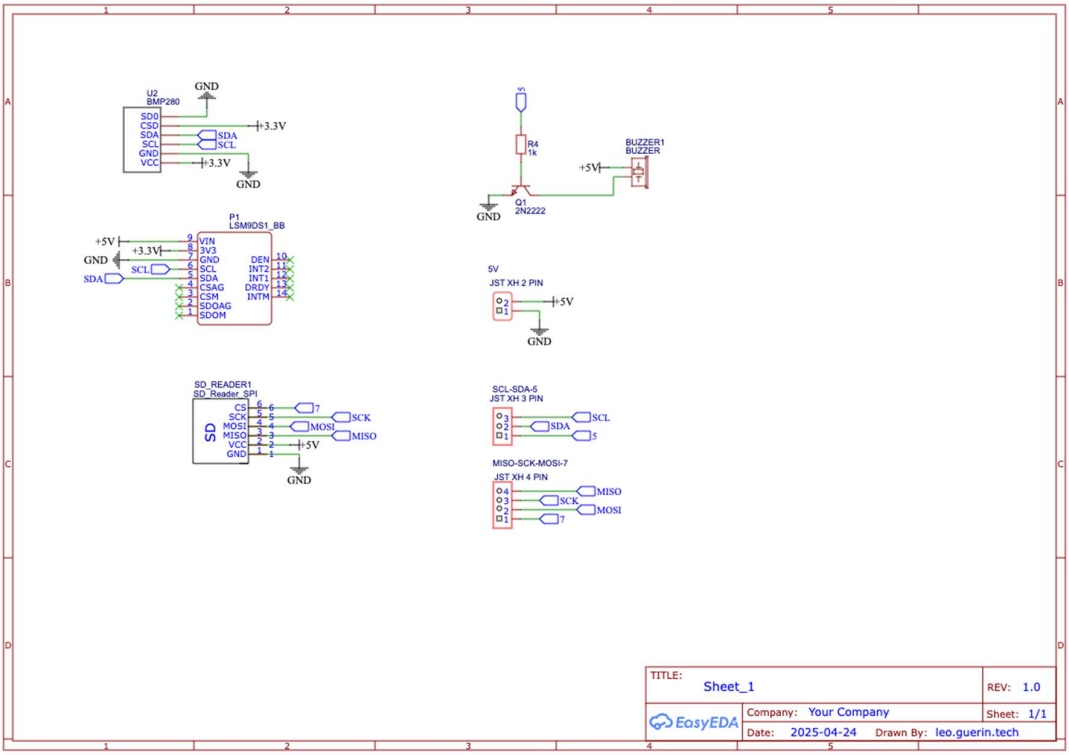


Fig. 11 Schéma électronique de l'étage 2 du système d'expérience

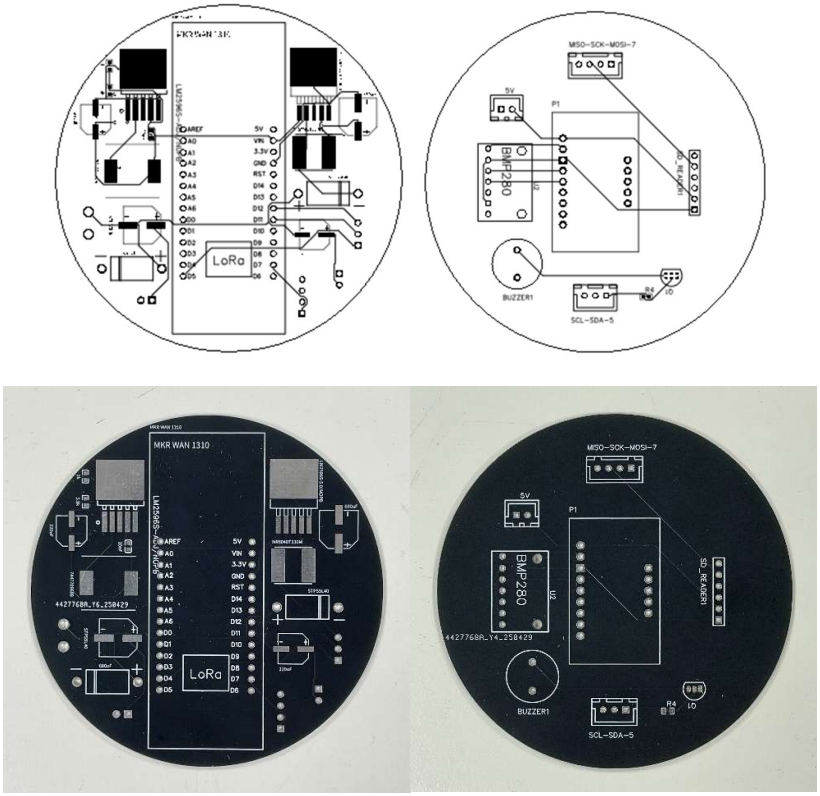


Fig. 12 PCBs de l'expérience/récolte de données

Les paramètres que nous mesurons lors du vol de la miniF sont les suivants : température, pression, positions et accélérations selon $x\ y\ z$.

La collecte de données sur la carte SD se fait toutes les 10 millisecondes, ce qui nous donne donc 100 données par seconde. Avec un temps de vol jusqu'à l'apogée de 7,5 secondes, cela nous donne 750 mesures distinctes, ce qui est largement suffisant pour que l'on fasse ensuite des analyses intéressantes.

Quant à la télémétrie de l'antenne LoRa envoie ces mêmes données à une antenne au sol toutes les 500 millisecondes. Cela nous donne moins de données, mais au moins cela nous confirme bien que la lecture de données marche correctement pendant le vol. Ce choix vient aussi du fait que l'envoi des données peut éventuellement prendre du temps au niveau du temps d'exécution du code, et ainsi cette mesure permet d'éviter une surcharge. Nous avons aussi mis un delay dans le code afin d'éviter justement que le processeur ait trop de choses à faire à la fois.

```
      // Vérification du timing pour l'envoi LoRa
      if (currentTime - lastLoRaTime >= LORA_INTERVAL_MS) {
        sendLoRa(data);
        lastLoRaTime = currentTime;
        bip(20); // Bip court pour l'envoi LoRa
      }
    }

    lastSDTime = currentTime;
  }

  // Micro-délai pour éviter de surcharger le processeur
  delayMicroseconds(100);
```

3.3 Rack électronique

Le rack électronique a été conçu de façon qu'il puisse s'insérer dans le tube de la miniF, tout en tenant l'ensemble des composants en place. Il fait une longueur de 194mm, et son diamètre est de 79mm ce qui laisse un peu de jeu à l'intérieur pour qu'il ne se coince pas. Les surfaces supérieures et inférieures du rack sont une découpe en forme de nid d'abeilles permettant de laisser passer le câblage et d'aérer la structure, et la face inférieure contient une surface pleine où peuvent reposer les batteries.

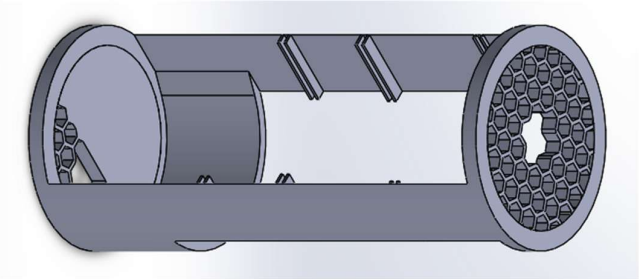


Fig. 13 CAO du rack PCB

Sur le long du rack, il y a trois insères permettant de tenir en place chacun des PCBs, afin qu'ils soient proprement fixés, et afin de minimiser le mouvement du câblage qui peut être fragile.

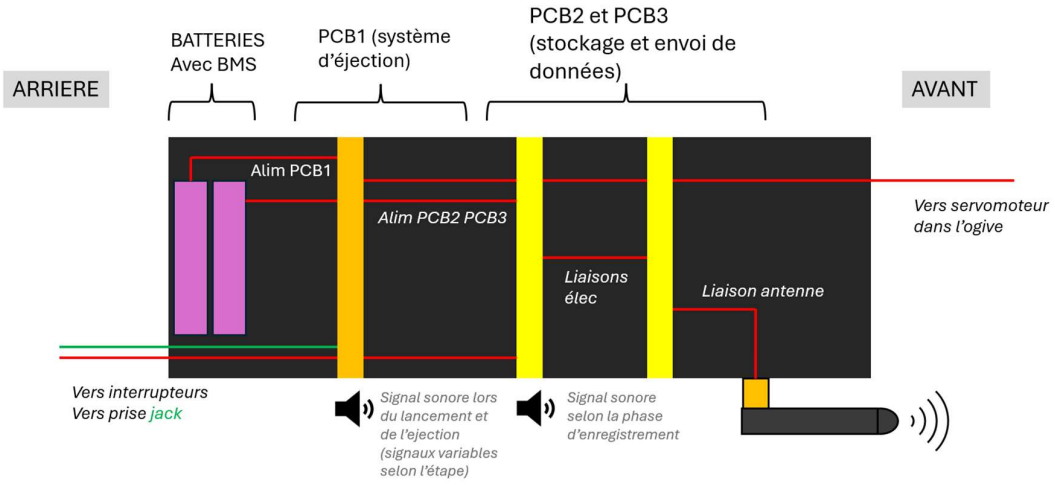


Fig. 14 Composants et branchements du rack électronique

4 – Aérodynamique

Voici le diagramme de stabilité de la miniF :

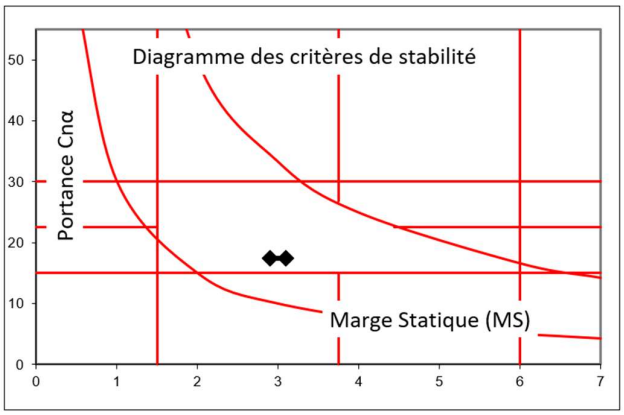


Fig. 15 Diagramme des critères de stabilité

Ce diagramme montre que la portance est légèrement faible. En effet, le $Cn\alpha$ est faible par rapport au centre de la zone de stabilité, mais il reste acceptable. Cela souligne que les ailerons sont éventuellement trop petits, et donc que la surface stabilisatrice n'est pas forcément assez importante. La miniF manque donc un peu de stabilité. Cependant, elle rentre toujours dans les critères de stabilité, donc le vol devrait se passer comme prévu.

5 – Support de propulsion / Plaque de poussée

Le(s) support(s) de propulsion est le support sur lequel la force de poussée lors de la combustion du Pro-24 est appliquée. Selon l'assemblage de la miniF, cette force peut être appliquée différemment, et cela se choisit notamment par rapport à comment le propulseur est attaché structurellement.

Dans notre cas, la poussée se fait en même temps par le haut et par le bas, c'est-à-dire que le Pro-24 est en contact avec la miniF en deux points. Un épaulement en bas du tube sur la bague tenant le Pro-24 permet de le fixer. Puis, le Pro-24 est directement en contact avec la plaque de poussée située plus haut dans le tube. Ainsi, la force de poussée est répartie entre ces deux points de la miniF. La bague de centrage quant à elle permet simplement de centrer le Pro-24 par rapport au tube, pour que la propulsion soit bien uniforme et que la fusée ne soit pas en déséquilibre au lancement.

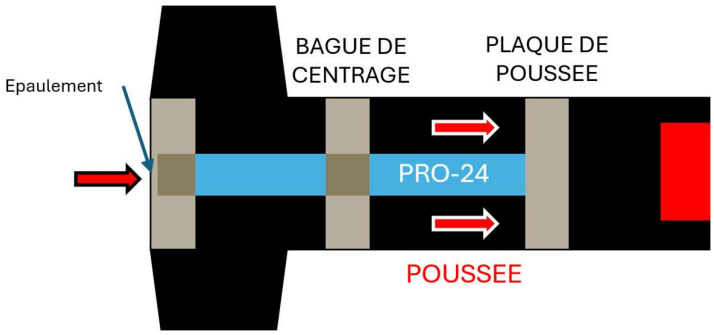


Fig. 16 Principe du support de propulsion

La fixation des différentes bagues et leurs caractéristiques sont cruciales, car ce sont elles qui emmagasinent le plus de forces lors du lancement. Dans le cas où elles sont trop faibles ou fines, elles peuvent se casser, ce qui ferait que le propulseur traverserait l'entièreté de la fusée très rapidement, n'y étant donc plus attaché. Nos bagues ont été découpées d'une planche de bois de 25mm d'épaisseur, et leur diamètre leur permet de se serrer précisément à l'intérieur du tube. Puis, après avoir fait 4 trous dans le tube, elles sont vissées en 4 points, ce qui assure une rigidité structurelle. *D'après les calculs du trajecto, le propulseur apporte une force (maximale pendant le lancement) de 240 newtons seulement 0,03s après sa mise à feu.*

6 – Calculs et estimations de trajectoire et vitesse

Simulation de vol par rapport au trajecto

Apogée :	$317.42\ m$
Vitesse de pointe :	$94\ ms^{-1}$
Equivalent Mach :	$M\ 0.276$
Accélération maximale :	$173\ ms^{-2}$
Durée du vol jusqu'à l'apogée :	$7.6\ s$
Vitesse au lancement (sortie de rampe) :	$26.47\ ms^{-1}$

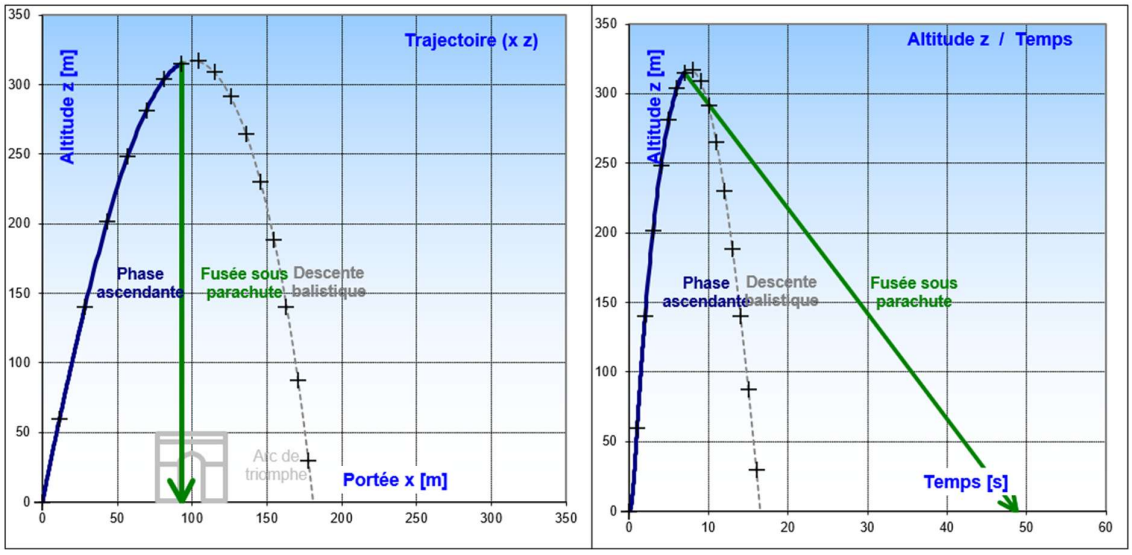


Fig. 17 Trajectoire estimée et altitude de la miniF par rapport au temps

La propulsion dure 1,09 secondes sur les 7,6 secondes de vol jusqu'à l'apogée. A la fin de la propulsion, la miniF est à une altitude de 67.87 mètres, ce qui veut dire que la montée jusqu'aux 317 mètres de l'apogée se fait uniquement grâce à l'inertie de la fusée.

7 – Rapport de vol

7.1 Electronique

Des incidents de carte PCB concernant la partie collecte et envoie de données a fait que nous n'avons pas pu collecter de données lors du vol. Cependant, tous ces systèmes ont fonctionné au sol lors des nombreux tests et expériences, donc leur développement a tout de même a été très formateur et nous a appris à expérimenter avec la télémétrie et l'émission à distance. Pour plus facilement débugger les problèmes de cartes électroniques, nous aurions pu ajouter des toutes petites LEDs (SMD LED) intégrées au circuit pour avoir des indications lumineuses, plus parlantes que de tester les incidents de carte au multimètre.

La présence de LEDs aurait été avantageuse pour mieux repérer l'état de la fusée selon l'étape du vol. En effet, selon le type de buzzer et le voltage circulant dans les systèmes électroniques, le buzzer n'a pas été assez fort en volume pour que l'on reconnaisse dans quel état était la fusée au sol selon la distance.

De plus, le fait de placer la connectique sur l'arrière de la miniF a eu un point positif et négatif :

- Point positif : placer les interrupteurs et le jack sur l'arrière de la miniF permet une certaine esthétique car cela enlève le besoin d'ajouter une trappe sur le côté du tube (surtout quand le tube est en carbone et que l'usinage devient donc compliqué et technique). Le tube est donc net et n'a pas de parties modifiées.
- Point négatif : l'accès est difficile pour les pyrotechniciens, qui lors du lancement ont accidentellement actionné l'un des interrupteurs. Les mettre sur le côté de la miniF permet une bien meilleure accessibilité en cage. De plus, il y a des risques que la chaleur de la propulsion brûle certains câbles (ce qui n'a pas été le cas lors de notre lancement mais l'éventualité n'est pas à négliger dans l'absolu). Enfin, le fait de faire cela implique une longueur importante de câbles qui doivent parcourir le long du tube. Ainsi le rack a été difficile à placer correctement sans écraser le câblage sur la bague de poussée.

7.2 Performances de vol

La trajectoire de la fusée a été très nette lors du lancement. Malgré l'absence d'une trajectoire tracée avec des données, visuellement le vol a été performant.

7.3 Incidents potentiels ayant mené au vol balistique

A l'apogée du vol, nous avons constaté qu'aucune séparation n'a eu lieu, même si les nombreux tests au sol ont été performants et efficaces. On a eu l'impression que la coiffe ne s'est pas séparée du tout de la miniF. Dans cette partie, nous dresserons la liste

des incidents potentiels ayant mené au vol balistique de la miniF. Il y a plusieurs facteurs qui ont pu participer à ce scénario :

- (Le plus probable) Nous avons eu besoin de changer la connectique entre le servomoteur dans la coiffe et le PCB, afin que lors de la séparation de la coiffe, celle-ci puisse s'éjecter suffisamment loin sans avoir besoin de câbles très longs et sans risquer que ces câbles s'arrachent.

Fig. 18 Câbles du servomoteur avec connectique Dupont

Ainsi, en utilisant des câbles Dupont, les câbles peuvent facilement se détacher lors de la séparation. Le problème est que les forces exercées lors du lancement ont potentiellement défait cette connexion, ce qui peut expliquer pourquoi la séparation ne s'est pas faite du tout lors du vol.

- Les forces du vent ont empêché l'éjection de la coiffe vers l'avant. Cette option a été assez prise en compte lors des tests avant le lancement. Cependant, considérant que la coiffe n'a absolument pas bougé, que la séparation fonctionnait correctement, et que la fusée a eu un vol très lent à l'apogée (et donc a fait face à un vent relatif très faible), c'est étonnant qu'aucune séparation, même partielle, n'ai eu lieu.

Fig. 19 La miniF en vol

Fig. 20 La miniF en fin de production

Chronologie Zenith – MF31 – C'Space 2025

Quand?	Où?	Qui?	Quoi?
T-60 min	Station de travail	Alexandre	• Brancher piles sur cartes élec • Insérer carte élec dans le rack • Visser bague corde parachute • Visser le support de la coiffe • Plier parachute, placer dans tube • Placer correctement les pins • Secouer coiffe pour assurer qu'ils sont bien placés • Visser le cache de l'ogive • Attacher élec servo • Pousser a coiffe à son emplacement de vol
		Léopold	• Allumer l'interrupteur **EJ** pour armer les pins avec le jack inséré **actions consécutives* • Eteindre l'interrupteur **EJ** et débrancher le jack
T-30 min	Public	Alexandre, Léopold, accompagnateurs	Arrivée sur l'aire de lancement
T-20 min	Zone rampe	Alexandre, Léopold, lanceur, accompagnateurs	Rejoindre la zone de rampe
T-15 min	Zone rampe	Alexandre, lanceur	• Mettre la fusée en rampe • Effectuer le test de compatibilité rampe
		Alexandre	• Brancher l'initialiseur dans le port jack **SEPA TRIG**. • Allumer l'interrupteur **EJ**. S'assurer que l'on entend le buzzer continu.

T-10 min			
	Zone rampe	Lanceur	Eriger la rampe à 80°
		Alexandre	Attacher l'initialiseur/jack à la rampe. Vérifier que celui-ci est bien attaché
T-5 min	Pupitre	Alexandre, Léopold, accompagnateurs	Rejoindre le pupitre de lancement
	Zone rampe	Lanceur	• Aller chercher le propulseur en zone de stockage • Mettre en place le propulseur • Mettre en place l'inflammateur
T-2 min	Pupitre	Lanceur	Rejoindre le pupitre
		Alexandre, Léopold, accompagnateurs	Ecouter les consignes de sécurité
T		Lanceur	Décompte final
		Alexandre	Appuyer sur le bouton de mise à feu

STABILITO

Stabilité de fusée à ailerons

Language/Langue	Français

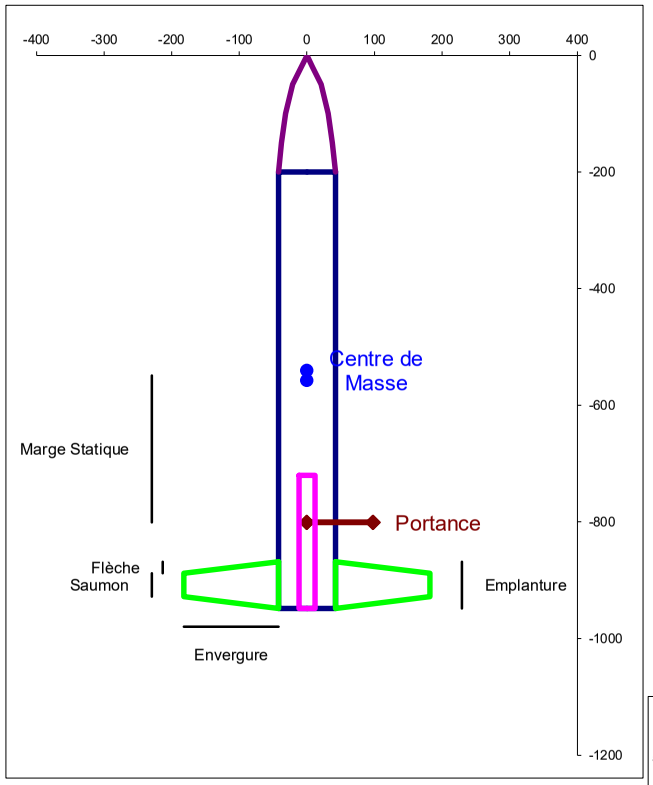
Remplir les cases jaunes

Fusée		
Nom	Zenith	
Club	Léofly	
Type	Minifusée	
Masse	1158 g	sans propu
Centre de Masse	519 mm	sans propu
Longueur totale	948 mm	

Propulseur	
Type	Pandora (Pro24-6G BS)
Position du bas	948 mm

Coiffe	
Forme	Ogivale (pointue)
Hauteur	200 mm
Diamètre	84 mm

Ailerons	Mono-empennage
Emplanture 'm'	80 mm
Saumon 'n'	40 mm
Flèche 'p'	20 mm
Envergure 'E'	140 mm
Epaisseur 'ep'	3 mm
Nombre	4
Position du bas	948 mm

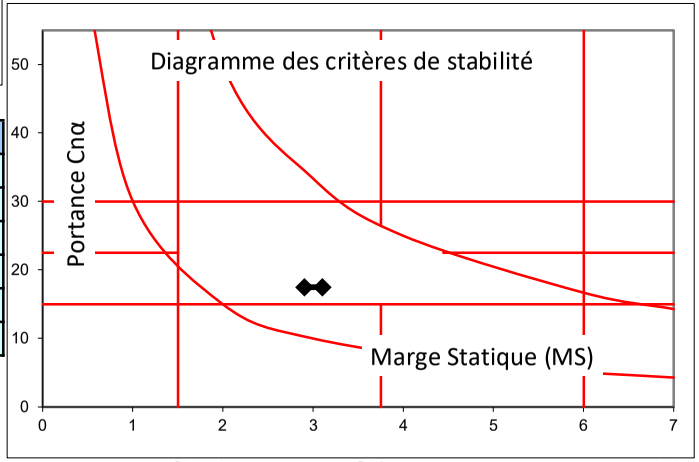


Fusée mono-diamètre,

	Propu plein	Propu vide	Sans propu
Masse propu	0.16 kg	0.084 kg	-
CdM propu	114 mm	114 mm	-
Masse fusée	1.318 kg	1.242 kg	1.158 kg
CdM fusée	557 mm	540 mm	519 mm

	XCp	Cnα
Coiffe	93 mm	2.0
Ailerons	892 mm	15.5

7/4/2025	Min	Résultats		Max
Finesse	10	11.3		20
Portance	15	17.5	17.5	30
MargeStat.	1.5 D	2.90 D	3.10 D	6 D
Couple	30	50.6	54.1	100
XCp		801 mm	801 mm	
MS /L		26% L	27% L	
		STABLE		



Commentaire libre :

Checksum : propu OK

v3.4.2

TRAJECTO

Trajectographie de fusée

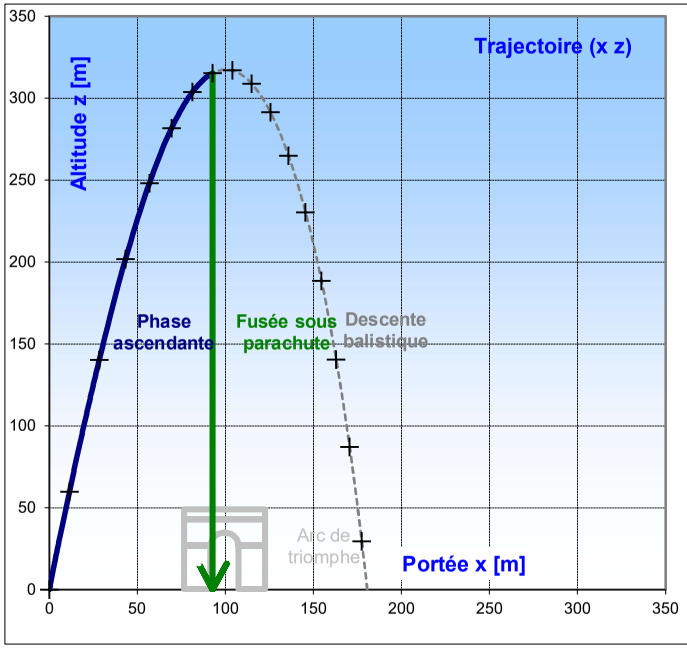
Remplir les cases jaunes

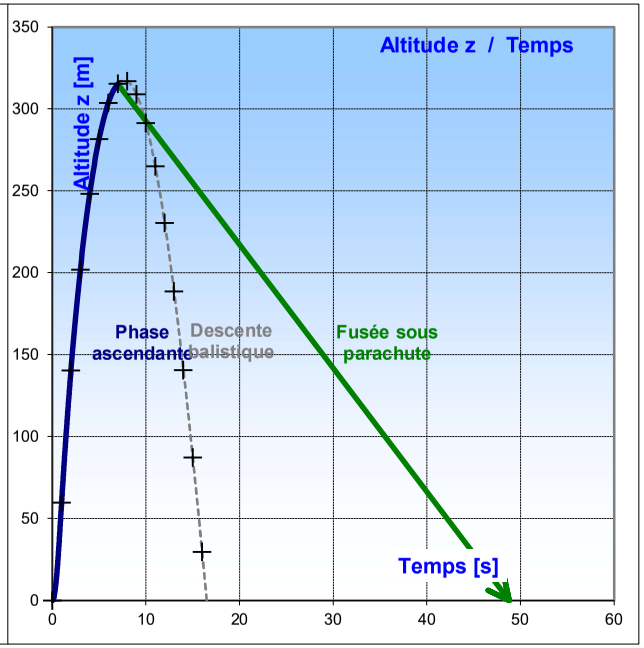
	Fusée
Nom	Zenith
Club	Léofly
Masse totale	1.3179 kg
Propulseur	Pandora (Pro24-6G BS)

	Traînée Aérdynamique
Surface Réf.	0.007222 m²
Cx	0.5

	Rampe de Lancement
Longueur	2.5 m
Élévation	80 °
Altitude	0 m

	DescenteSousParachute	
	Fusée	0 satellite
Masse	1.2423 kg	
Dépotage	N/A	
Ouverture para	7 s	
Surface para	0.35 m²	
Cx parachute	1	
Vitesse du vent	5 m/s	
Vitesse descente	7.5 m/s	
Durée descente	42 s	
Durée du vol	49 s	
Déport latéral	± 209 m	





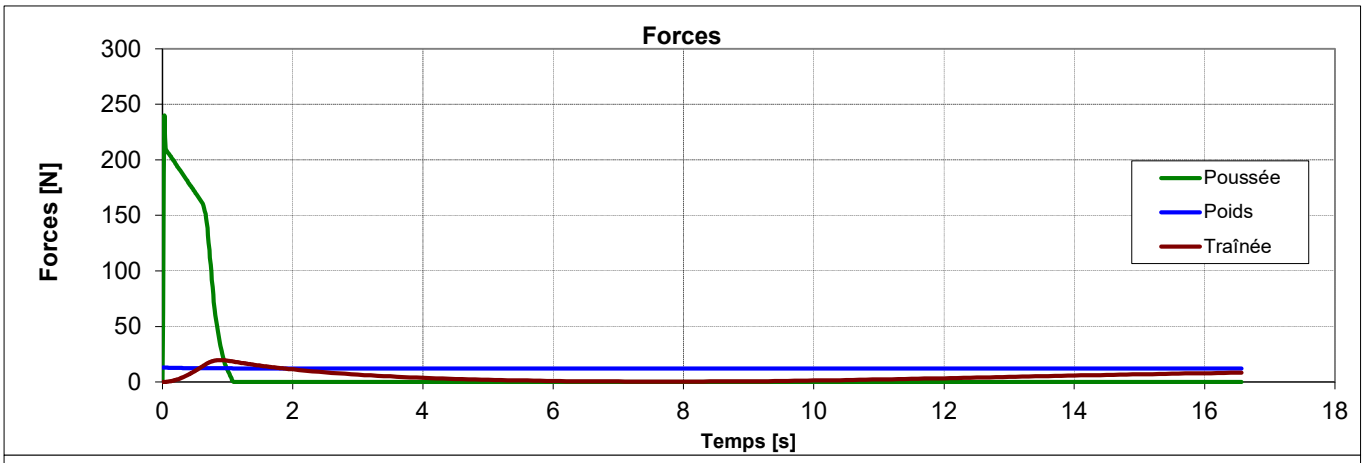
7/4/2025	Temps	Altitude z	Portée x	Vitesse	Accélération	Efforts
Sortie de Rampe				26.9 m/s		
Vit max & Acc max				94 m/s	173 m/s²	
Culmination, Apogée	7.6 s	317 m	100 m	11 m/s		
Ouverture parachute fusée	7.0 s	315 m	93 m	13 m/s		36.8 N
Impact balistique	16.5 s	~0 m	181 m	61.5 m/s		2351 J

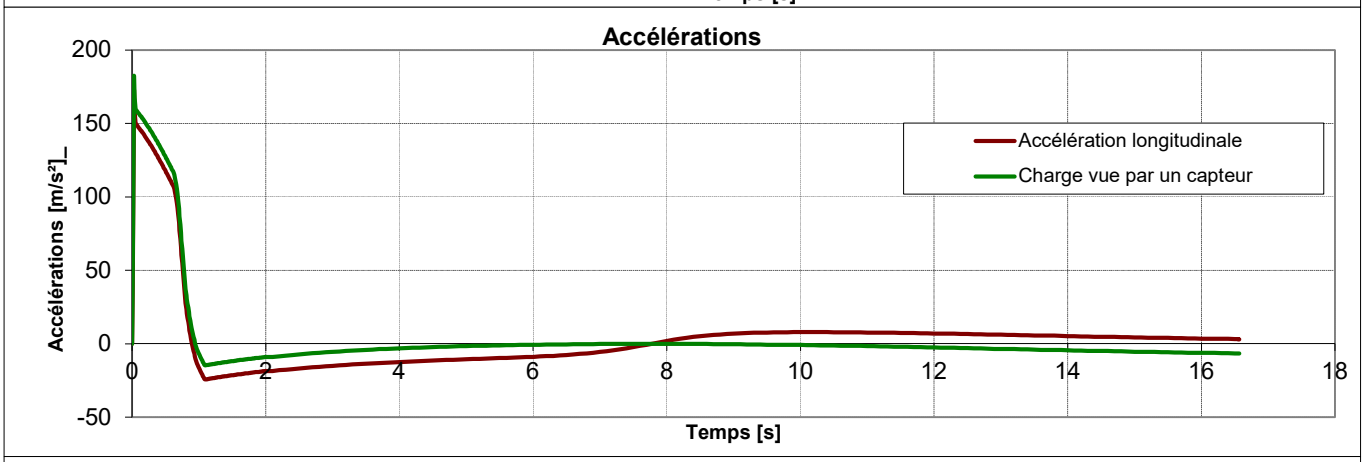
	Pour localiser la fusée
Couleur fuselage/coiffe	noir
Couleur parachute fusée	rouge

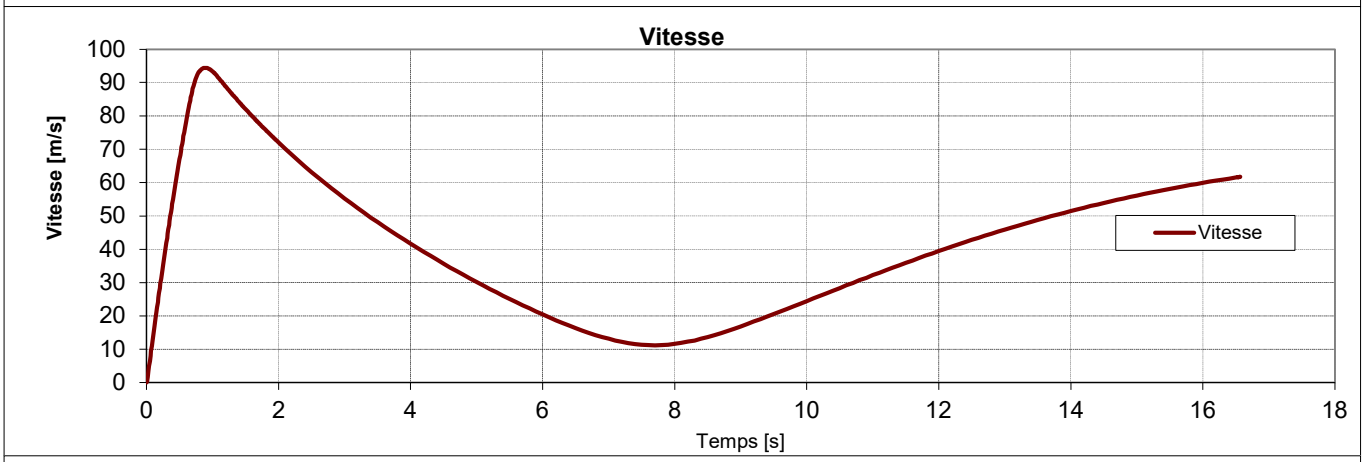
Commentaire libre :

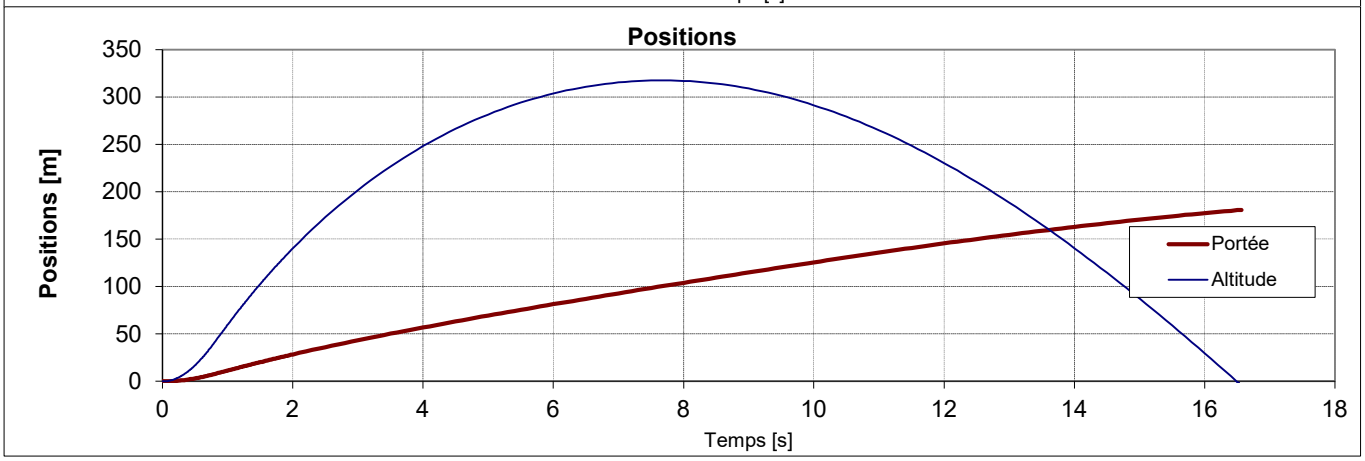
propu OK

v3.4.2









Notes :
Ces courbes représentent la trajectoire de la fusée dans l'hypothèse d'une descente balistique (sans ouverture du parachute).
L'accélération longitudinale gravitationnelle définit le mouvement (dérivée de la vitesse) : Acc = (Poussee - Traînée ± Poids) / m
La charge "non-gravitationnelle" vue par un capteur d'accélération (masse-ressort) est : Charge = (Poussée - Traînée) / m

```
extern const byte SERVOS = 1;
#include <TinyServo.h>

const int buzzerPin = 5;         // PA5
const int cablePin = 1;          // PA1
const byte servoPin[] = {2};     // PA2 : un seul servo utilisé
#define SERVO1 0

const int angleLibre = 70;       // À ajuster selon le montage
const int angleLibreViolent = 40;       // À ajuster selon le montage
const int angleInsertion = 85;   // À ajuster selon le montage

bool enAngleLibre = true;
bool cableArracher = false;
unsigned long cableArracheTime = 0;
unsigned long lastBeepTime = 0;

void setup() {
  pinMode(buzzerPin, OUTPUT);
  pinMode(cablePin, INPUT_PULLUP);
  setupServos();

  int cableState = digitalRead(cablePin);

  if (cableState == HIGH) {
      moveServo(SERVO1, angleLibre);
      enAngleLibre = true;
      delay(10000);
  } else {
    moveServo(SERVO1, angleInsertion);
  }

  if (digitalRead(cablePin) == LOW) {
    moveServo(SERVO1, angleInsertion);
    enAngleLibre = false;
  }
}

void loop() {

  int cableState = digitalRead(cablePin);
  unsigned long currentTime = millis();

  if (cableState == HIGH) {
    if (!cableArracher) {
      cableArracher = true;
      cableArracheTime = currentTime;
    }
```

```
    if ((currentTime - cableArracheTime) >= 8000) {
      if (!enAngleLibre) {
        moveServo(SERVO1, angleLibreViolent);
        enAngleLibre = true;
        beepSepa(buzzerPin, 1500, 100, 400, 200, 10000);
      }

    } else if (currentTime - lastBeepTime > 300) {
      beep(buzzerPin, 1000, 100);
      lastBeepTime = currentTime;
    }
  } else {
        cableArracher = false;

    if (enAngleLibre) {
      moveServo(SERVO1, angleInsertion);
      enAngleLibre = false;
    }

    if (currentTime - lastBeepTime > 1000) {
      beep(buzzerPin, 700, 100);
      lastBeepTime = currentTime;
    }
  }
}

void beep(int pin, int frequency, int duration) {
  long period = 1000000L / frequency;
  long cycles = (long)frequency * duration / 1000;

  for (long i = 0; i < cycles; i++) {
    digitalWrite(pin, HIGH);
    delayMicroseconds(period / 2);
    digitalWrite(pin, LOW);
    delayMicroseconds(period / 2);
  }
}

void beepSepa(int pin, int frequency, int shortDuration, int longDuration, int ↵
  pause, int totalDuration) {
  long period = 1000000L / frequency;
  long shortCycles = (long)frequency * shortDuration / 1000;
  long longCycles = (long)frequency * longDuration / 1000;
  unsigned long startTime = millis();

  while (millis() - startTime < totalDuration) {
    for (long i = 0; i < shortCycles; i++) {
      digitalWrite(pin, HIGH);
      delayMicroseconds(period / 2);
```

```
      digitalWrite(pin, LOW);
      delayMicroseconds(period / 2);
    }
    delay(pause);

    for (long i = 0; i < longCycles; i++) {
      digitalWrite(pin, HIGH);
      delayMicroseconds(period / 2);
      digitalWrite(pin, LOW);
      delayMicroseconds(period / 2);
    }

    delay(pause);
  }
}
```

```cpp
#include <SPI.h>
#include <SD.h>
#include <Wire.h>
#include <LoRa.h>
#include <Adafruit_BMP280.h>
#include <Adafruit_LSM9DS1.h>
#include <Adafruit_Sensor.h>

/*************** Définition des broches ****************/
#define SD_CS 7
#define LORA_CS 4
#define BUZZER_PIN 5

/*************** Timing constants ****************/
#define SD_INTERVAL_MS 10      // Sauvegarde SD toutes les 10ms
#define LORA_INTERVAL_MS 500  // Envoi LoRa toutes les 5000ms (demi seconde)

/*************** Macros debug ****************/
#define debugPrint(x) if (Serial) Serial.print(x)
#define debugPrintln(x) if (Serial) Serial.println(x)

/*************** Variables de timing ****************/
unsigned long lastSDTime = 0;
unsigned long lastLoRaTime = 0;
unsigned long currentTime = 0;

/*************** Buffer pour optimiser les écritures SD ****************/
String dataBuffer = "";
const int BUFFER_SIZE = 50; // Nombre de lignes à accumuler avant écriture
int bufferCount = 0;

void bip(int duration) {
  digitalWrite(BUZZER_PIN, HIGH);
  delay(duration);
  digitalWrite(BUZZER_PIN, LOW);
  delay(200);
}

void bipOK() {
  bip(50);
  delay(100);
  bip(50);
}

void bipErreur() {
  while (true) {
    bip(400);
    delay(600);
  }
```

```
}

/*************** Capteurs ***************/
Adafruit_BMP280 bmp;
Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1();

/*************** Setup IMU ***************/
void setupIMU() {
  lsm.setupAccel(lsm.LSM9DS1_ACCELRANGE_2G, lsm.LSM9DS1_ACCELDATARATE_119HZ);
  lsm.setupGyro(lsm.LSM9DS1_GYROSCALE_245DPS);
  lsm.setupMag(lsm.LSM9DS1_MAGGAIN_4GAUSS);
}

/*************** Fonction pour lire les capteurs ***************/
String readSensors() {
  // Lecture BMP280
  if (!bmp.takeForcedMeasurement()) {
    debugPrintln("BMP280 erreur mesure");
    return "";
  }

  float temp = bmp.readTemperature();
  float press = bmp.readPressure();
  float alt = bmp.readAltitude(1013.25);

  // Lecture LSM9DS1
  lsm.read();
  sensors_event_t a, m, g, tempIMU;
  lsm.getEvent(&a, &m, &g, &tempIMU);

  // Formatage des données avec timestamp
  String data = String(millis()) + "," + String(temp, 1) + "," + String(press,
    0) + "," + String(alt, 1) + ",";
  data += String(a.acceleration.x, 2) + "," + String(a.acceleration.y, 2) + ","
     + String(a.acceleration.z, 2) + ",";
  data += String(g.gyro.x, 2) + "," + String(g.gyro.y, 2) + "," + String
    (g.gyro.z, 2);

  return data;
}

/*************** Fonction pour sauvegarder sur SD avec buffer ***************/
void saveToSD(String data) {
  if (data.length() == 0) return;

  dataBuffer += data + "\n";
  bufferCount++;

  // Écriture quand le buffer est plein ou toutes les 5 secondes
```

```
  if (bufferCount >= BUFFER_SIZE || (millis() - lastSDTime) > 5000) {
    File file = SD.open("log.csv", FILE_WRITE);
    if (file) {
      file.print(dataBuffer);
      file.close();
      dataBuffer = "";
      bufferCount = 0;
    } else {
      debugPrintln("Erreur ouverture fichier SD");
      bip(150);
    }
  }
}

/*************** Fonction pour envoyer par LoRa ***************/
void sendLoRa(String data) {
  if (data.length() == 0) return;

  LoRa.beginPacket();
  LoRa.print(data);
  LoRa.endPacket();
}

void setup() {
  Serial.begin(115200);
  delay(1000);

  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, LOW);

  debugPrintln("=== Démarrage ===");

  /***** LSM9DS1 (I2C) *****/
  debugPrintln("Initialisation LSM9DS1...");
  if (!lsm.begin()) {
    debugPrintln("Erreur LSM9DS1");
    bipErreur();
  }
  setupIMU();

  /***** BMP280 (I2C) *****/
  debugPrintln("Initialisation BMP280...");
  if (!bmp.begin(0x76)) {
    debugPrintln("Erreur BMP280 (I2C)");
    bipErreur();
  }
  bmp.setSampling(Adafruit_BMP280::MODE_FORCED,
                  Adafruit_BMP280::SAMPLING_X2,
                  Adafruit_BMP280::SAMPLING_X16,
```

```
                  Adafruit_BMP280::FILTER_X16,
                  Adafruit_BMP280::STANDBY_MS_500);

  /***** SD Card (SPI) *****/
  debugPrintln("Initialisation carte SD...");
  if (!SD.begin(SD_CS)) {
    debugPrintln("Erreur carte SD");
    bipErreur();
  }

  // Création du header CSV si le fichier n'existe pas
  if (!SD.exists("log.csv")) {
    File file = SD.open("log.csv", FILE_WRITE);
    if (file) {
      file.println
        ("timestamp,temp,press,alt,acc_x,acc_y,acc_z,gyro_x,gyro_y,gyro_z");
      file.close();
    }
  }

  /***** LoRa (SPI) *****/
  debugPrintln("Initialisation LoRa...");
  if (!LoRa.begin(868E6)) {
    debugPrintln("Erreur LoRa");
    bipErreur();
  }

  debugPrintln("Setup terminé !");
  bipOK();

  // Initialisation des timers
  lastSDTime = millis();
  lastLoRaTime = millis();
}

void loop() {
  currentTime = millis();

  // Vérification du timing pour la sauvegarde SD (toutes les 10ms)
  if (currentTime - lastSDTime >= SD_INTERVAL_MS) {
    String data = readSensors();

    if (data.length() > 0) {
      // Sauvegarde sur SD
      saveToSD(data);

      // Debug (optionnel, peut ralentir)
      // debugPrintln(data);
```

```
      // Mini bip pour indiquer l'activité (optionnel)
      digitalWrite(BUZZER_PIN, HIGH);
      delayMicroseconds(500);
      digitalWrite(BUZZER_PIN, LOW);

      // Vérification du timing pour l'envoi LoRa
      if (currentTime - lastLoRaTime >= LORA_INTERVAL_MS) {
        sendLoRa(data);
        lastLoRaTime = currentTime;
        bip(20); // Bip court pour l'envoi LoRa
      }
    }

    lastSDTime = currentTime;
  }

  // Micro-délai pour éviter de surcharger le processeur
  delayMicroseconds(100);
}

/*************** Fonction d'urgence pour vider le buffer ***************/
void flushSDBuffer() {
  if (bufferCount > 0) {
    File file = SD.open("log.csv", FILE_WRITE);
    if (file) {
      file.print(dataBuffer);
      file.close();
      dataBuffer = "";
      bufferCount = 0;
    }
  }
}
```