




# Dossier de définition du projet Cansat

Équipe ENSEA : Cheyenne PEREZ, Rémi GLEMET, Noah DELCOURT, Quentin ROLLAND
2022-2023

Professeurs référents : M. Abdelhak KASBARI et M. Julien SEIGNEURBIEUX

Compétition CanSat, édition 2022-2023

# Table des matières

# Introduction

"Le CanSat est un dispositif autonome simulant l'atterrissage d'une sonde sur une planète étrangère et étant capable de réaliser des missions scientifiques. Le principe du CanSat repose sur l'idée de concevoir dans un volume réduit correspondant à une Canette de soda, une charge utile similaire à celle embarquée dans un Satellite/sonde d'exploration. Originellement de 33cL, notre catégorie de la compétition nous impose un volume de 1L pour une hauteur de 20 cm et un diamètre de 8 cm. L'ensemble des systèmes propres aux satellites, tels que l'alimentation, les dispositifs de mesure et de télémesure, doivent être conçus pour pouvoir intégrer le volume réduit du CanSat.

Ce dernier est largué à l'aide d'un drone ou d'un ballon captif et réalise ses missions à partir du largage, pendant la descente et jusqu'à 5 minutes après l'atterrissage. Les missions proposées impliquent la mise en œuvre d'expériences scientifiques depuis la phase de définition jusqu'à la phase d'analyse des données collectées.

L'objectif de la compétition CanSat France est de réaliser un prototype CanSat capable d'exécuter les missions définies chaque année par l'organisme Planète science. La partie finale du projet est la compétition qui réunit l'ensemble des équipes participantes, durant l'été, lors de la campagne nationale de lancement C'Space, en présence de représentants du CNES, de Planète Sciences et de l'industrie spatiale."

*d'après le règlement Cansat 2023*[1]

Nous avons pris part à la compétition Cansat dans un premier temps dans le cadre de notre projet de dernière année de l'ENSEA. Notre cursus à l'ENSEA s'est terminé en janvier. Nous avons ensuite décidé de poursuivre le projet en dehors du cadre de l'école et de le porter à son terme. Les 4 membres du groupe ont été séparés durant la période de janvier à juin car réalisant des stages ou échanges académiques à l'étranger (Etats Unis, Canada et Espagne), il n'a donc pas été aisé de travailler conjointement. Nous avons travaillé au mieux avec le temps et les moyens disponibles avec pour seul objectif de livrer un prototype fonctionnel dans les délais pour le C'space. Nous avons pris beaucoup de plaisir à réaliser ce projet qui nous a permis de développer de nombreuses compétences et de mettre en pratique notre apprentissage à l'ENSEA.

# Compte-rendu technique

## 1 Cahier des charges

### 1.1 Mission principale : Contrôler le Cansat et atteindre un point précis

Le CanSat doit-être capable de naviguer et contrôler sa trajectoire pour se rapprocher le plus d'une position GPS définie avant le largage.

Position GPS cible : 43°13'18.7"N 0°03'10.0"W

La position GPS peut être modifiée par l'organisation et doit pouvoir être transférée rapidement dans le CanSat le jour du lâcher.

### 1.2 Missions secondaires

#### 1.2.1 Reconnaître un QR code au sol

Un QR Code au format A0 sera placé au sol, notre mission est de reconnaître le QR Code et de récupérer les informations encodées par télémétrie ou par enregistrement à bord (vérifié avant et après le largage). Les QR codes recèlent des segments de texte de taille variable (moins de 50 caractères), pouvant contenir tous types de caractères alphanumériques. Les panneaux sont imprimés au format A0. Les fichiers svg et png sont disponibles sur demande.

#### 1.2.2 Déployer une structure au sol

Le CanSat doit déployer une structure couvrant le maximum de surface au sol. Chaque équipe décide de la nature, forme et taille de la structure.

# 2 Ressources matérielles et logicielles

## 2.1 Ressources matérielles

| Nom du composant | Référence |
|---|---|
| Module GPS | TEL 0132 |
| Parapente miniature | Hacker ARF rouge Parapente RC 1500 mm |
| Centrale inertielle | IMU-10DOF |
| Microcontrôleur | STM32-Nucleo-F411RE |
| 3 servomoteurs | HiTEC HS-422 |
| Module structure gonflable | cf partie 4 |
| Batterie | 1 pile Alkaline 9V |
| Carte de puissance | cf partie 5.1 |
| Détecteur infrarouge | pololu sds01a (GP2Y0D805Z0F) |
| Baromètre | BMP280 |

TABLE 1 – Liste de tous les composants utiles pour notre projet

## 2.2 Ressources logicielles

- Pour la partie software
  - STM32CubeIDE, FreeRTOS
- Pour le design du PCB
  - KiCAD
- Design de la structure
  - Fusion 360

# 3 Mission principale

Dans cette partie, nous allons développer la solution que nous avons imaginée pour effectuer la mission principale, qui, nous le rappelons consiste à contrôler le Cansat et atteindre un point GPS précis.

## 3.1 Détails de notre solution

Dans un premier temps, nous avons étudié les deux possibilités qui s'offraient à nous. Nous pouvions soit orienter le Cansat dans les airs à l'aide d'une aile de parapente, soit faire atterrir le Cansat au sol et ensuite le diriger au sol à l'aide de roues. Cependant, le cahier des charges indique que : "le terrain est inégal avec des herbes hautes et/ou non entretenues. Selon les conditions météorologiques, les déplacements au sol des CanSat peuvent être difficiles". Nous avons donc choisi de nous orienter vers la première solution, à savoir, guider le Cansat à l'aide d'une aile de parapente pendant sa descente.

Pour ce faire, nous utilisons deux bras reliés à deux servomoteurs, placés de part et d'autre de la structure du Cansat (cf figure 1, 2 et 3).

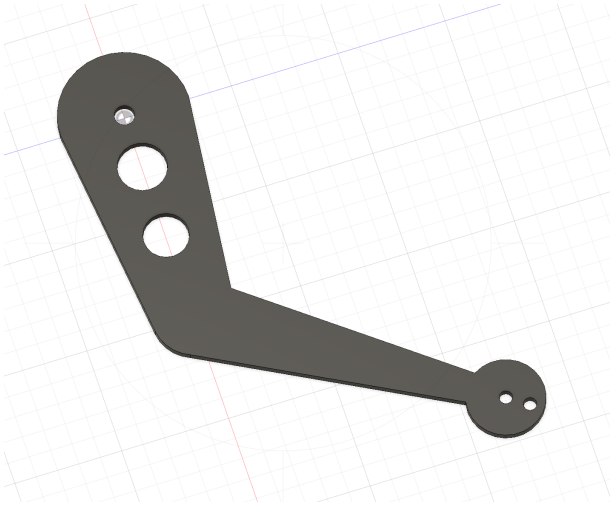

FIGURE 1 – Schéma 3D du bras du Cansat

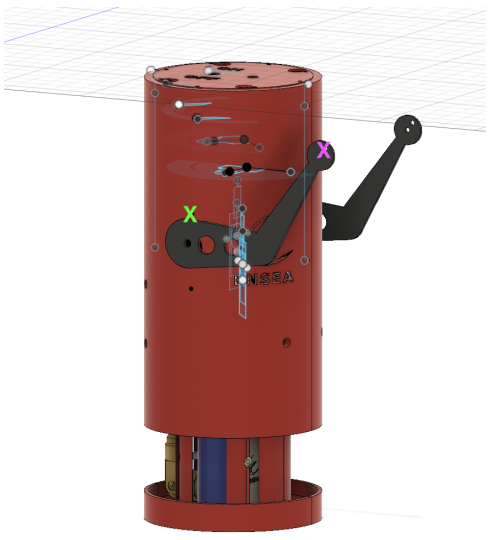

FIGURE 2 – Schéma 3D du Cansat avec bras fixés vue du côté droit

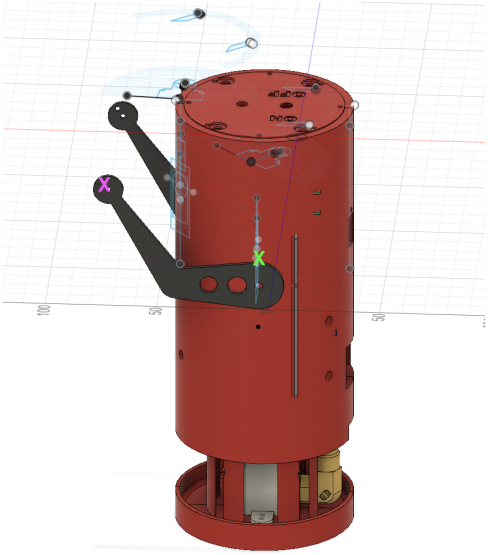

FIGURE 3 – Schéma 3D du Cansat avec bras fixés vue du côté gauche

Les attaches du parachutes sont fixées au niveau des marqueurs verts sur les figures 2 et 3. Les attaches reliées aux freins du parachute sont fixés aux extrémités des bras, au niveau des marqueurs roses sur ces mêmes figures. Ainsi, il nous suffira d'activer les servomoteurs reliés au bras pour tirer sur les freins gauches ou droit du parachute, afin de modifier la trajectoire du système.

Cependant, pour orienter le Cansat et pour décider de la trajectoire à empreinter, il faut connaître son orientation (c'est-à-dire vers quelle direction pointe l'avant du Cansat) et sa position geographique (ou sa position GPS). L'orientation du Cansat sera calculée par un microcontrôleur qui devra faire l'acquisition à intervalle de temps régulier de la position GPS actuelle du système, via un module GPS, ainsi que de son orientation, via un

magnétomètre utilisé comme boussole. En combinant ces deux données, le microcontrôleur sera en mesure de déterminer la bonne trajectoire et de s'orienter dans la bonne direction en actionnant les servomoteurs reliés aux bras eux-même reliés aux freins du parachute.

## 3.2 Calculs et dimensionnement du parachute

### 3.2.1 Dimensionnement du parachute

On considère un modèle simplifié du projet, avec notre module représenté par une masse m (qu'on estime à 1kg, soit la valeur maximale autorisée par le cahier des charges). Les forces qui s'appliquent à notre système sont donc :

- $P$ : le poids appliqué à la masse (on néglige celui du parapente)
- $F_t$ : la force de traînée du parapente
- $F_p$ : la force de portance du parapente

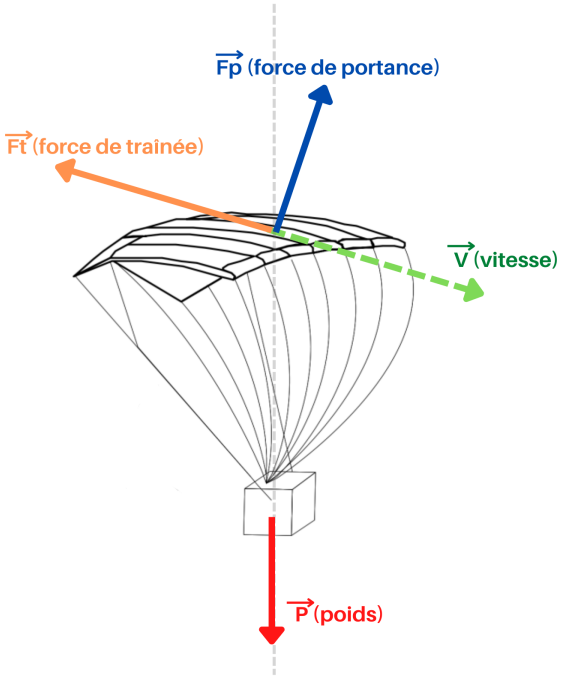


FIGURE 4 – Représentation des forces sur le parachute

Dans un premier temps nous effectuons le choix de notre aile de parapente, sur laquelle repose tout notre système.

La finesse d'une aile de parapente récente (soit le rapport du déplacement horizontal sur le déplacement vertical) se situe en moyenne entre 6 et 8. D'après le cahier des charges, la hauteur minimale à laquelle notre CanSat sera largué est de 80m. Avec une finesse de 6, cela signifie donc que notre module peut théoriquement effectuer un déplacement horizontal de 480m, ce qui est largement supérieur aux 150m de distance maximale à

laquelle sera l'objectif réel. Cette marge nous permettra de compenser les mouvements parasites dûs aux changements de trajectoire, aux erreurs d'asservissements, et autres perturbations.

Compte tenu de la précision nécessaire dans la conception de l'aile afin d'obtenir un système stable et contrôlable, ainsi que de la complexité de celle-ci, nous avons décidé de partir sur une aile utilisée pour des modèles miniatures de paramoteurs de loisir.

Caractéristiques de l'aile :

- **Envergure** : 1.5m
- **Surface projetée** : $0.5m^2$

FIGURE 5 – Aile utilisée

### 3.2.2 Choix de l'utilisation d'une hélice

Compte tenu des incertitudes concernant les paramètres environnementaux le jour du test (sens et intensité du vent, paramètres météorologiques,...) on envisage l'utilisation d'une hélice de propulsion à l'arrière de notre module afin d'assister notre aile et d'assurer un déplacement horizontal suffisamment rapide.

Cependant son utilisation n'est pas sans inconvénient. En effet une hélice de propulsion entraînée par un moteur brushless nécessiterait une batterie de plus grande capacité et pouvant délivrer un courant plus important. De plus, le capteur IMU permettant de faire l'acquisition de l'orientation du module est sensible aux variations de champs électromagnétiques et risquerait d'être perturbé par un tel moteur. On s'interroge donc sur la pertinence d'un tel ajout.

Afin de trancher sur son utilisation ou non, on commence par estimer si la force de poussée apportée par une hélice apporte une différence significative dans la dynamique de notre modèle.

On utilise le modèle suivant :

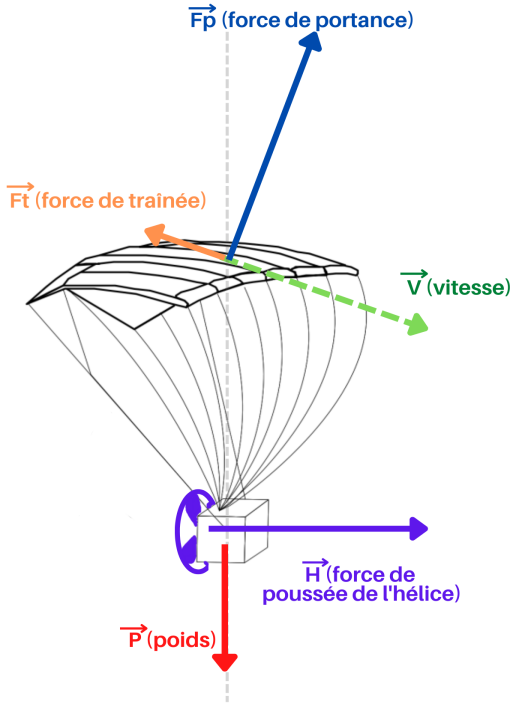


FIGURE 6 – Représentation des forces sur le parachute avec hélice

Les forces appliquées au système sont définies comme suit :

$$P = mg$$

$$F_p = \frac{1}{2} * \rho * V^2 * S * C_z$$

$$F_t = \frac{1}{2} * \rho * V^2 * S * C_x$$

Le calcul de la force de poussée de l'hélice est approximé par la formule d'Abbott :

$$H = 28,35 * Pas * D^3 * N^2 * 10^{-10}$$

Estimation des forces

On suppose :

Masse limite imposée par le cahier des charges

$$m = 1kg$$

Masse volumique de l'air

$$\rho = 1,2kg/m^2$$

Valeur moyenne basse pour des parapentes de cette dimension

$$V = 8m/s$$

Surface projetée de notre aile de parapente

$$S = 0,5m^2$$

Coefficient de portance moyen pour des parapentes de cette dimension

$$C_z = 0,5$$

Coefficient de traînée moyen pour des parapentes de cette dimension

$$C_x = 0,1$$

Valeur maximale du diamètre d'hélice possible avec les restrictions géométriques du cahier des charges

$$D = 2cm$$

Estimation du pas pour une hélice

$$Pas \approx \pi * D * tan(45) = 6,3cm$$

Valeur utilisée pour les modèles réduits de parapente motorisés

$$N = 8000rpm$$

Avec ces valeurs, on obtient une force de traînée $F_t = 2N$ et une force de poussée de l'hélice d'environ $H = 0,05N$ avec ces dimensions. On constate que la force de poussée de l'hélice ne parvient à compenser que 2,5% de la force de traînée de notre parapente.

**Conclusion** : Compte tenu de l'apport limité de l'ajout d'une hélice et des diverses contraintes qu'imposent son utilisation, nous avons pris la décision de ne pas en utiliser.

## 3.3 Récupération des données GPS

### 3.3.1 Présentation du module

Pour ce projet nous avons choisi le module GPS TEL0132 basé sur l'AT6558, une puce de navigation GPS BDS/GNSS multi-mode. Il est capable de fournir des informations de localisation en temps réel en prenant en compte jusqu'à 32 canaux de suivi d'une grande variété de systèmes de navigation par satellite. En outre, le module peut recevoir le signal GNSS de 6 systèmes de navigation par satellite simultanément, dont le BDS chinois (BeiDou navigation Satellite system), le GPS américain, le GLONASS, le système européen GALILEO, Japan QZSS et SBAS satellite augmentation (WAAS, EGNOS, GAGAN MSAS), et réalise le positionnement, la navigation et la synchronisation conjointe.

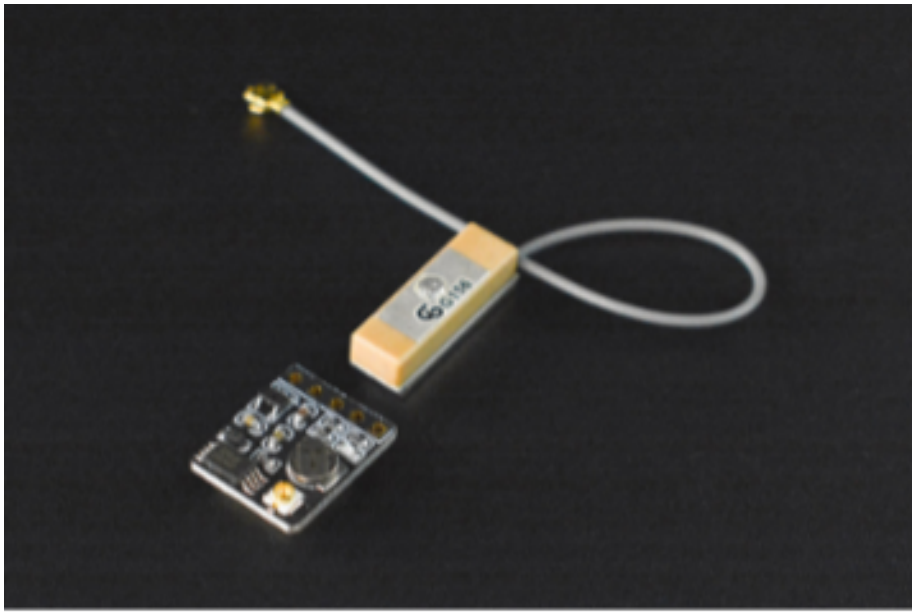

FIGURE 7 – Illustration du GPS

En mode sortie série, le module est compatible avec différents microcontrôleurs équipés de sortie série, comme notamment : Arduino, Raspberry Pi, STM32, ce qui correspond exactement à notre besoin pour ce projet. Sur le papier, l'erreur de précision de positionnement est mesurée à environ 3 m, essentiellement la même que celle des smartphones. La consommation d'énergie du module est aussi faible que 0,1 W (<25mA), et il peut fonctionner en continu pendant une longue période avec une petite alimentation. La petite taille du module lui permet d'intégrer notre système CANSAT facilement.

| Nom | Fonction |
|---|---|
| Le | Entrée d'alimentation (5V) |
| GND | Terre |
| TX | Transmettre |
| RX | Recevoir |
| ..PPS | Sortie d'impulsions par seconde |

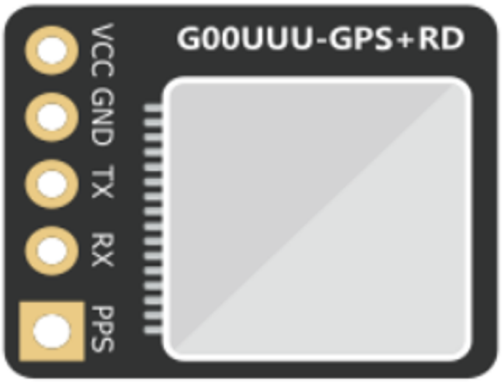


FIGURE 8 – Description des pins du GPS

Il est noté que le module peut être perturbé par les bâtiments environnants et qu'il est recommandé de l'utiliser en extérieur loin de tout éléments perturbateurs. Ce qui correspond exactement à notre utilisation car notre système sera lancé au milieu d'un champ. Cependant lors des TP projet, nous nous trouvons dans un bâtiment de type cage de faraday, ce qui n'est pas très pratique pour vérifier les données reçues. Lorsque la LED embarquée clignote à une fréquence régulière, le positionnement est terminé. Le débit de données du système est fixé à 9600 bauds, nous recevons les trames d'information à la fréquence 1Hz.

### 3.3.2 Analyse des données

Il existe trois types de données dans les trames reçues : GN, GP et BD. Ils représentent respectivement le mode bimode, le mode GPS et le mode Beidou. Le contenu du format de trame de protocole NMEA0318 est de la forme suivante :

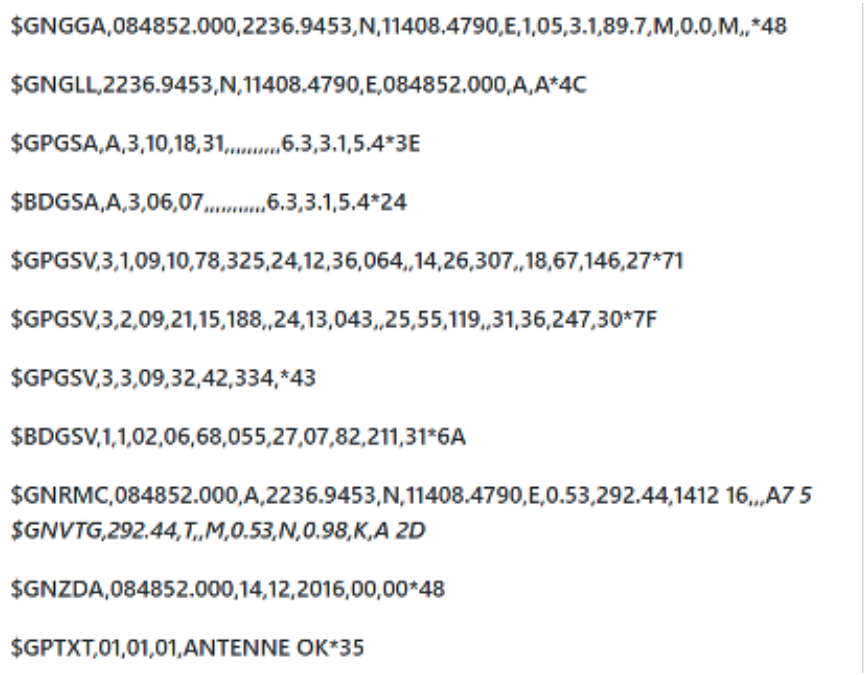


FIGURE 9 – Illustration des trames avec le protocole NMEA0318

Dans la trame récupérée on analyse les informations suivantes pour le mode GPS :

| ID | Protocole | Description |
|---|---|---|
| $GPGGA | GGA | **Informations de localisation GPS** |
| $GPGLL | GLL | Information sur la position géographique |
| $GPGSA | GSA | Information actuelle sur les satellites |
| $GPGSV | GSV | Satellites en vue |
| $GPRMC | RMC | Données GNSS minimales |
| $GPVTG | VTG | Informations sur la vitesse au sol |
| $GPZDA | ZDA | Informations sur l'heure et la date |
| $GPTXT | TXT | Sortie d'état antenne |

FIGURE 10 – Tableau explicatif du protocole NMEA0318

Par la suite nous allons nous baser sur les informations de la première trame d'information du protocole GGA. On peut alors analyser le nombre de satellites détectés, la latitude et la longitude ou encore de nombreuses informations comme la hauteur par rapport au niveau de la mer mais nous nous intéresserons essentiellement aux coordonnées GPS.

Ces données sont transmises au format sexagésimal souvent noté DMS (degrés, minutes, secondes). L'unité de base est le degré d'angle (1 tour complet = 360 degrés), puis la minute d'angle (1 degré = 60'), puis la seconde d'angle (1 degré = 3 600"). Pour la latitude on aura le format suivant : ddmm.mmmm et pour la longitude le format : dddmm.mmmm. On aura simplement à récupérer les degrés puis à additionner la valeur en flottant : mm.mmmm et la diviser par 60.

| Num | Nom | Exemple | Unité | Description |
|---|---|---|---|---|
| | Message ID | $GPGGA | | En-tête du protocole GGA |
| <1> | UTC Position | 161229.487 | | hhmmss.sss |
| <2> | Latitude | 3723.2475 | | ddmm.mmmm |
| <3> | Latitude Direction | N | | N=nord ou S=sud |
| <4> | Longitude | 12158.3416 | | dddmm.mmmm |
| <5> | Longitude | W | | E=est ou W=ouest |
| <6> | Indicateur de correction de position | 1 | | 0: Correction non disponible ou invalide<br>1: Mode GPS SPS, correctif valide 2: GPS différentiel, mode SPS, correctif valide 3: Mode GPS PPS, correctif valide |
| <7> | Nombre de satellites en service | 07 | | Plage 00 à 12 |
| <8> | Dilution horizontale de précision | 1.0 | | Gamme: 0.5-99.9 |
| <9> | Hauteur du niveau de la mer | 9.0 | Mètres | Gamme : -9999.9-9999.9 |
| <10> | Unités | | M | M signifie mètre |
| <11> | Séparation du géoïde | | Mètres | Gamme : -999.9-9999.9 |
| <12> | Unités | | M | M signifie mètre |
| <13> | Données sur l'âge de correction | | Secondes | Vide lorsqu'aucune donnée différentielle n'est présentée |
| <14> | Diff. Réf. ID de la station | 0000 | | Plage : 0000-1023 (vide lorsqu'aucune donnée différentielle n'est présentée) |
| Hh | Somme de contrôle | 18 | | Somme de contrôle de tous les caractères codes ASCII entre $ et *<br>(XOR chaque octet pour obtenir la somme de contrôle, puis la convertir en caractères ASCII en hexadécimal ) |
| | CR LF | | | Fin du cadre du protocole |

FIGURE 11 – Format de l'information GPS

### 3.3.3 Configuration STM32

Pour ce projet nous utilisons la NUCLEO F446RE de chez ST qui nous permet dans un premier temps de réaliser les tests des différents modules que nous souhaitons intégrer au projet. L'UART2 réglé à 115200 Bits/s avec interruption correspond à la liaison série qui nous permet de vérifier les trames et les données reçues dans cette première phase de test. L'UART1 réglé à 9600 bits/s avec interruption permet la communication avec notre module.

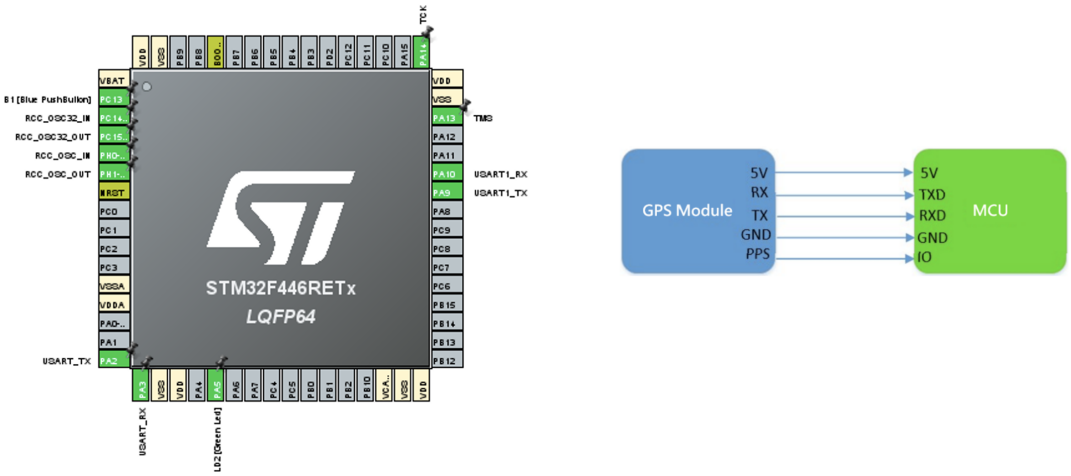


FIGURE 12 – Illustration des pins de configuration

### 3.3.4 Premiers tests

En utilisant simplement la fonction de Callback de l'UART nous pouvons directement afficher la trame reçue toutes les secondes.

```
/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    if(huart == &huart1){
        HAL_UART_Transmit(&huart2, &uart_gps, 1, HAL_MAX_DELAY);
        HAL_UART_Receive_IT(&huart1, &uart_gps, 1);
    }
}
```

FIGURE 13 – Code langage C de l'UART

Concernant le module GPS, les trois situations suivantes sont toutes des démarrages à froid : l'utiliser pour la première fois, si les informations sur les éphémérides sont perdues, si le module est déplacé à plus de 1000 km en état de mise hors tension. Lorsque le GPS a effacé les informations de positionnement interne depuis la dernière opération et que le récepteur GPS a perdu les paramètres du satellite, ou que le navigateur ne peut pas fonctionner correctement parce qu'étant donné que les paramètres existants sont trop différents des paramètres réels reçus du satellite, il est nécessaire que le GPS obtienne les nouvelles données de coordonnées fournies par le satellite. Cela peut prendre plusieurs dizaines de minutes, et c'est pour cela que l'on a eu du mal à le démarrer au début, car nous n'attendions pas assez longtemps. Le démarrage tiède fait référence au démarrage à plus de 2 heures de la dernière heure de positionnement. Le temps de positionnement est compris entre le démarrage à froid et le démarrage à chaud. Si le GPS a été utilisé il y a un jour, le premier démarrage du lendemain est un démarrage tiède et les informations de dernière position seront affichées après le démarrage. Si cela fait trop longtemps alors les éphémérides ont changé et la précédente configuration ne peut pas l'accepter. Ainsi, plusieurs satellites dans les paramètres ont perdu le contact avec le récepteur GPS

et doivent continuer à chercher des informations de position supplémentaires. Par conséquent, le temps de recherche pour le démarrage tiède est plus long que celui du démarrage à chaud et plus court que celui du démarrage à froid. Lorsque le GPS est démarré à l'endroit où il a été arrêté la dernière fois et que le temps écoulé depuis le dernier temps de positionnement est inférieur à 2 heures, il s'agit d'un démarrage à chaud. Ci-dessous voici les informations reçus lorsque nous nous trouvons en salle de projet ( 8 satellites sont détectés) :

```
$GNGGA,104621.000,4902.36368,N,00204.36093,E,1,08,3.7,69.9,M,0.0,M,,*46
$GNGLL,4902.36368,N,00204.36093,E,104621.000,A,A*49
$GNGSA,A,3,05,13,14,30,,,,,,,,4.1,3.7,1.8,1*38
$GNGSA,A,3,07,10,26,40,,,,,,,,4.1,3.7,1.8,4*3A
$GPGSV,3,1,09,05,30,199,23,13,77,105,28,14,48,068,37,15,70,286,,0*68
$GPGSV,3,2,09,17,12,107,,18,06,281,,23,24,315,,24,28,260,,0*62
$GPGSV,3,3,09,30,25,071,35,0*58
$BDGSV,1,1,04,07,23,048,37,10,29,057,38,26,82,203,25,40,22,036,40,0*70
$GNRMC,104621.000,A,4902.36368,N,00204.36093,E,0.00,240.39,060123,,,A,V*0E
$GNVTG,240.39,T,,M,0.00,N,0.00,K,A*2F
$GNZDA,104621.000,06,01,2023,00,00*4C
$GPTXT,01,01,01,ANTENNA OK*35
```

FIGURE 14 – Illustration des informations reçues par le GPS en salle de projet

### 3.3.5 Code de récupération des coordonnées GPS

Afin de récupérer les données UART on commence par créer une variable d'interruption à chaque réception de données dans la variable uart_gps_rx.

```
/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    it_rx_gps = 1;
    HAL_UART_Receive_IT(&huart1, (uint8_t*)&uart_gps_rx, 1);
}
```

FIGURE 15 – Code langage C pour la récupération des données GPS

Ce code a par la suite été modifié en tenant compte de la valeur récupérée par le protocole TXT qui indique le bon fonctionnement de l'antenne. Nous récupérons dans la variable position[24] les coordonnées de latitude et de longitude données par le protocole GGA. Puis dans un deuxième temps nous séparons les données de latitude et longitude dans 2 variables distinctes. On peut ainsi récupérer les valeurs en degrés dans lat2[2] et long3[3] puis les valeurs en minutes dans lat7[7] et long7[7]. On procède ensuite à la conversion en float afin de pouvoir diviser ces deux valeurs respectives et finalement les additionner aux valeurs en degrés avant de les récupérer respectivement dans les variables latitude et longitude.

```
if(it_rx_gps){
    if(data_rdy == 1){
        for(int j = 0; j<=23;j++){
            position[j] = coordonnees[j+18]; //position[] comprend la latitude et la longitude telles q
            //En effet, les coordonnes sont d'abord de cette forme : "$GNGGA,121933.000,4902.36627,N,00
            //On sélectionne donc la partie qui est après l'identifieur et l'heure mais avant le ",E,".
        }

        for(int k = 0; k<=9;k++){
            latitude_data[k] = position[k]; //latitude non convertie
        }
        for(int l = 0; l<=10;l++){
            longitude_data[l] = position[l+13]; //longitude non convertie
        }

        lat2[0] = latitude_data[0];
        lat2[1] = latitude_data[1]; //lat2 -> les deux digits avant la virgue du degré de la lat.

        for(int m = 0; m<=6;m++){
            lat7[m] = latitude_data[m+2]; //lat 7 -> ce qui reste après la virgule de la lat.
        }

        long3[0] = longitude_data[0];
        long3[1] = longitude_data[1];
        long3[2] = longitude_data[2]; //long3 -> les trois digits avant la virgule du degré de la long.

        for(int n = 0; n<=6;n++){
            long7[n] = longitude_data[n+3]; //long7 -> ce qui reste après la virgule de la long.
        }

        float deg_lat = atof(lat2);
        float reste_lat = atof(lat7); //conversion du char[lat] en float[lat]

        float deg_long = atof(long3);
        float reste_long = atof(long7); //conversion du char[long] en float[long]

        latitude = deg_lat + (reste_lat/60); //latitude convertie
        longitude = deg_long + (reste_long/60); //longitude convertie

        data_rdy = 0;
    }
```

FIGURE 16 – Code langage C pour la conversion des données GPS

Toujours lorsque l'interruption est active, on vient remplir le tableau coordonnees[1000] puis lorsque l'on arrive à OK*35 on saute une ligne dans notre terminal puis on fait passer notre variable data ready à 1 afin de pouvoir venir lire nos données comme expliqué précédemment.

```
    if(uart_gps_rx[0]==10){
        HAL_UART_Transmit(&huart2, "\r\n", 2, HAL_MAX_DELAY); //Arrangement de la trame
    }
    else{
        uart_pc_tx[0]=uart_gps_rx[0];

        //affichage trame
        HAL_UART_Transmit(&huart2, uart_pc_tx, 1, HAL_MAX_DELAY);

        coordonnees[i] = uart_pc_tx[0]; //On copie ce qui passe dans l'UART dans un tableau coordonnees[i].
        if(i >= 4){
            if(strncmp("OK*35",&coordonnees[i-4],5) == 0){

                HAL_UART_Transmit(&huart2, "\r\n", 2, HAL_MAX_DELAY); //Arrangement de la trame

                i = 0; //Si on trouve "OK", on réinitialise i à 0 pour synchroniser notre trame.
                if(first_data == 1){
                    data_rdy = 1; //Si la trame est initialisée, le premier caractère du tableau voulu est prêt.
                }
                else first_data = 1; //Permet d'enregistrer le tableau désiré.
            }
            else{
                if(i == GPS_TRAME_SIZE-1){
                    i = 0;
                }
                else{
                    i++;
                }

            }

        }
        else i++;
    }

    it_rx_gps = 0;
}
```

FIGURE 17 – Code langage C GPS

### 3.3.6 Implémentation et précision

Finalement, on peut récupérer les variables de longitude et de latitude qui nous intéressent afin de les comparer avec les coordonnées de la cible à atteindre. Les valeurs récupérées lors des tests nous montrent une précision relative qui peut aller jusqu'à 40m d'erreur lorsque nous nous trouvons à la fenêtre de la salle de projet. Il nous faudra faire des tests en extérieur, loin de tout élément perturbateur afin de vérifier la précision du GPS qui est normalement entre 0 et 10-15m maximum.

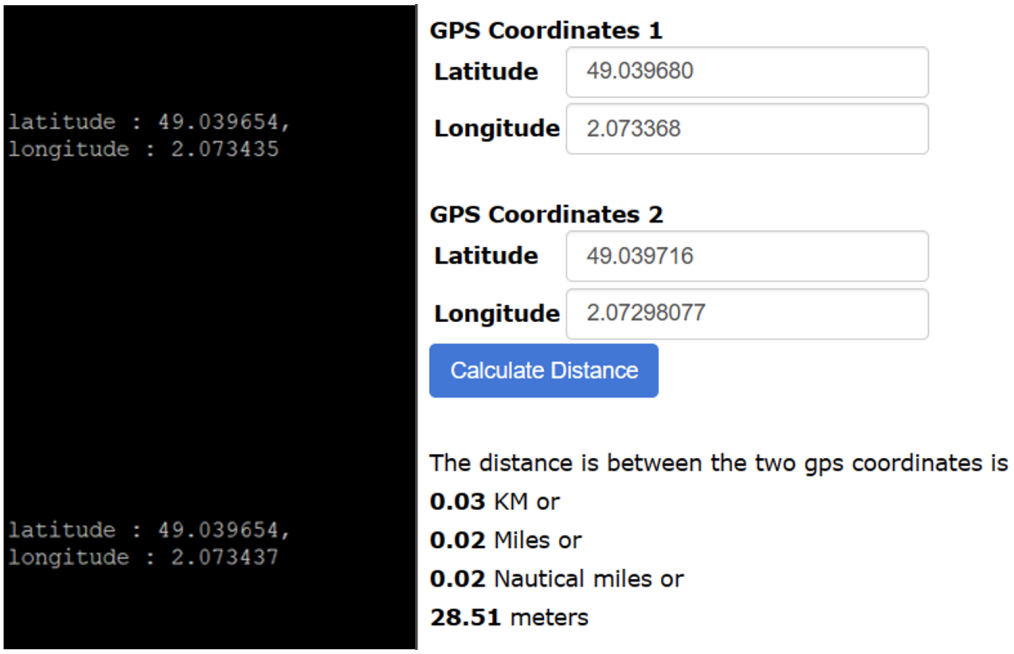


FIGURE 18 – Exemple d'erreur de la part du module GPS

Pour l'implémentation, plusieurs options sont possibles afin de limiter les erreurs :

- Effectuer une moyenne sur 2 ou 3 valeurs avant largage afin de connaître la position approximative du système par rapport à la cible puis une fois le largage fait, venir comparer les valeurs moyennes effectuées précédemment avec les valeurs récupérées en temps réel afin de détecter une potentielle erreur GPS.
- Ne compter que sur les valeurs en temps réel ce qui représente un risque si la précision est aussi mauvaise qu'en salle de projet, ce qui ne devrait pas être le cas.

## 3.4 Récupération des données IMU

Nous utilisons le MPU-9250. Il s'agit d'une centrale inertielle 9 axes. Elle contient un accéléromètre, un gyromètre et un magnétomètre. C'est ce dernier qui va nous intéresser en particulier.

### 3.4.1 ID du capteur

Dans un premier temps, nous devons vérifier l'identité du capteur MPU-9250 afin d'être sûr (car on communique avec plusieurs capteurs) que c'est bien à lui que l'on s'adresse. Pour cela, il existe un registre qui contient un identifiant propre au capteur : registre WHO_AM_I présenté ci-dessous.

| BIT | NAME | FUNCTION |
|---|---|---|
| [7:0] | WHOAMI | Register to indicate to user which device is being accessed. |

This register is used to verify the identity of the device. The contents of *WHO_AM_I* is an 8-bit device ID. The default value of the register is 0x71.

FIGURE 19 – Registre WHO_AM_I

Nous devons lire dans ce registre qui se trouve à l'adresse 0x75. D'après la documentation, nous devrions lire la valeur 0x71. On test donc cette valeur et on rentre dans le Error_Handler si elle ne correspond pas.

### 3.4.2 Initialisation du capteur

Le registre PWR_MGMT_1 permet de réinitialiser l'ensemble des registres du capteur (bit 7 à 1) et il permet aussi de choisir la meilleure horloge possible avec la PLL (bit [2 :0] à 1).

### 3.4.3 Acquisition des données du magnétomètre

Sur le schéma de description interne du MPU-9250 figure 20, nous remarquons que le magnétomètre (entouré en rouge) n'est pas relié au système de la même manière que l'accéléromètre et le gyromètre. Ses données ne sont pas accessibles directement dans les registres. Il faut d'abord désactiver l'interface Master i2c et mettre ses pins en dérivation (ES_CL et ES_DA) afin de permettre aux données du magnétomètre d'atteindre le registre des données.

Pour cela, il faut mettre à 1 le bit 1 du registre 55 : INT_PIN_CFG

**4.19 Register 55 – INT Pin / Bypass Enable Configuration**

**Serial IF: R/W**

**Reset value: 0x00**

| | | |
|---|---|---|
| [1] | BYPASS_EN | When asserted, the i2c_master interface pins(ES_CL and ES_DA) will go into 'bypass mode' when the i2c master interface is disabled. The pins will float high due to the internal pull-up if not enabled and the i2c master interface is disabled. |
| [0] | RESERVED | |

FIGURE 20 – Registre Bypass

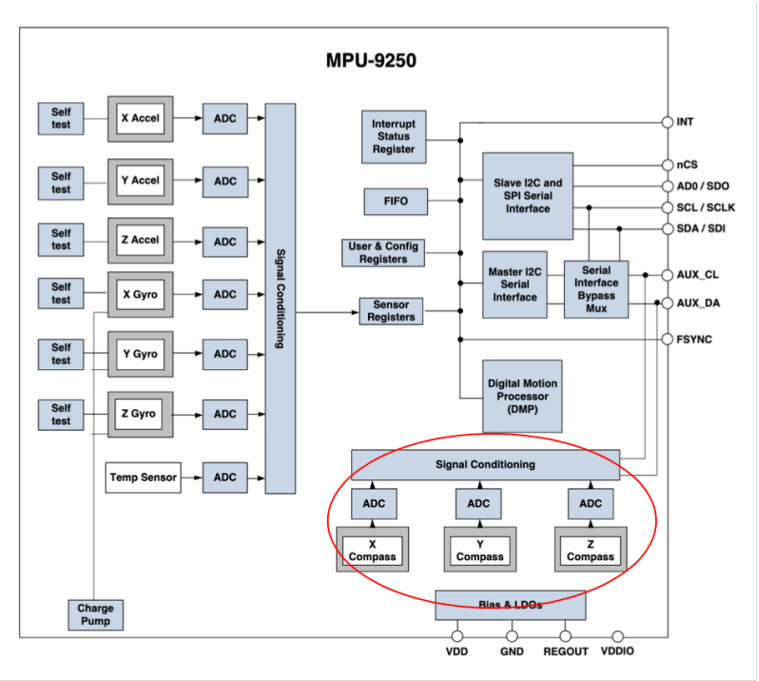


FIGURE 21 – Schéma de la structure de l'IMU

Dans un deuxième temps, il est nécessaire de configurer le magnétomètre afin qu'il effectue des mesures de manière régulière. Pour cela, il faut écrire dans le registre de contrôle du magnétomètre (CNTL1) pour sélectionner le mode qui nous convient. Ici il s'agit du mode : Continuous measurement mode 1 (ou 2). Il faut donc écrire 0110 sur les bit [3 :0]. De plus, nous voulons des données sur 16 bit pour plus de précisions. Il faut donc mettre le bit 4 à 1.

**5.8 CNTL1: Control 1**

| Addr | Register name | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|
| Read-only register | | | | | | | | | |
| 0AH | CNTL1 | 0 | 0 | 0 | BIT | MODE3 | MODE2 | MODE1 | MODE0 |
| Reset | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

MODE[3:0]: Operation mode setting

"0000": Power-down mode
"0001": Single measurement mode
"0010": Continuous measurement mode 1
"0110": Continuous measurement mode 2
"0100": External trigger measurement mode
"1000": Self-test mode
"1111": Fuse ROM access mode
Other code settings are prohibited

BIT: Output bit setting

"0": 14-bit output
"1": 16-bit output

FIGURE 22 – Registre Control 1 du magnétomètre

Pour s'adresser au magnétomètre, il faut désormais utiliser l'adresse MAGNETO_ADD 0x18 et non plus MPU_ADD 0xD0. Dans un premier temps, vérifions que le magnétomètre est oppérationnel et prêt à communiquer sur le bus i2c. Pour cela, nous devons lire la valeur qui se trouve dans le registre WHO_AM_I_AK8963. Si cette valeur est 0x48, la communication est possible.
Une fois acquises par le magnétomètre, les données des mesures de champ magnétique sont disponibles dans les registres HXL ... HZH (à gauche en rouge). Voici le protocole pour y accéder :

1 - Lire le bit DRDY du registre ST1. S'il est à 0, les données ne sont pas prêtes à être lues. S'il vaut 1, les données peuvent être lues.
2 - Lecture des données de champ magnétique
3 - Lire le bit HOFL du registre ST2 pour vérifier qu'il n'y a pas eu d'overflow

| Name | Address | READ/ WRITE | Description | Bit width | Explanation |
|---|---|---|---|---|---|
| WIA | 00H | READ | Device ID | 8 | |
| INFO | 01H | READ | Information | 8 | |
| ST1 | 02H | READ | Status 1 | 8 | Data status |
| HXL | 03H | READ | Measurement data | 8 | X-axis data |
| HXH | 04H | | | 8 | |
| HYL | 05H | | | 8 | Y-axis data |
| HYH | 06H | | | 8 | |
| HZL | 07H | | | 8 | Z-axis data |
| HZH | 08H | | | 8 | |
| ST2 | 09H | READ | Status 2 | 8 | Data status |

FIGURE 23 – Tableau des registres du magnétomètre

La norme du vecteur vaut environ 51 $\mu$T, ce qui correspond à la valeur attendue qui est d'environ 47 $\mu$T à Cergy. Le champ magnétique terrestre étant très faible, dès que l'on approche le capteur d'une source de champ magnétique locale, la norme augmente de manière significative (jusqu'à 100$\mu$T pour l'ordinateur).

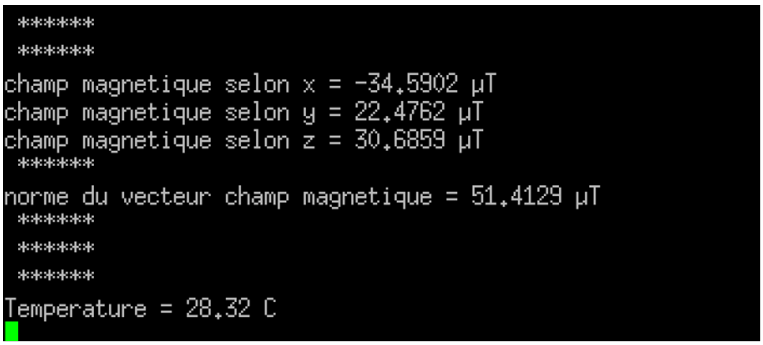


FIGURE 24 – Illustration de l'affichage des données sur l'interface Putty

Il reste ensuite à convertir la donnée de champ magnétique en angle (En cours... explications plus précises dans les prochaines versions).

De plus, il reste les étapes suivantes :
- Calibrer le magnétomètre afin de prendre en compte les caractéristiques de l'environnement. (En cours...)
- Concevoir une boussole robuste multi-capteur afin d'améliorer la précision de la boussole et de limiter au maximum les perturbations de l'acquisition des données de champ magnétique dûes aux mouvements subits par la capsule. Consiste à prendre en compte les données de l'accéléromètre et du gyromètre. (En cours...)

## 3.5 Méthode d'orientation et d'asservissement pour arriver au point GPS attendu

Le GPS est précis à 2.5m près et donne l'information toutes les 1 seconde. Il nous donne une longitude (Ouest) et une latitude (Nord) en degré. Comme le cansat sera parachuté à une distance négligeable devant la taille de la Terre, nous considérerons que nous serons dans le plan. Tous nos calculs se font dans cette hypothèse. Pour trouver quels ordres donner aux servos moteurs, nous avons commencé par chercher toutes les configurations possibles. Nous le rappelons, le but de notre Cansat est d'atteindre un point GPS en se dirigeant dans les airs.

### 3.5.1 Solutions trouvées

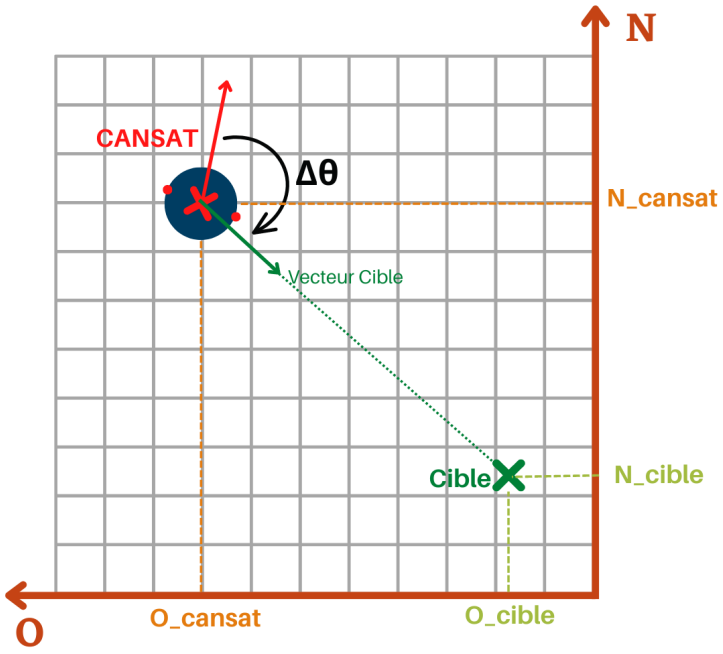


FIGURE 25 – Exemple de placement du Cansat et de la cible

Dans l'exemple ci-dessus, nous avons les coordonnées du Cansat données par le GPS et de la cible. Nous devons définir un vecteur qui donnera la direction dans laquelle

le cansat doit s'orienter. Le nord magnétique sera donné par le magnétomètre étudié précédemment.

Détermination du vecteur donnant la direction à suivre

Nous faisons dans un premier temps les calculs pour l'axe nord :

(N) : $\Delta N = N_{Cible} - N_{Cansat}$

Si $\Delta$N $> 0$, alors nous savons que nous devons orienter notre cansat vers le nord
Si $\Delta$N $< 0$, alors nous savons que nous devons orienter notre cansat vers le sud

Puis, nous effectuons les calculs et le raisonnement identique pour l'axe Ouest.

(O) : $\Delta O = O_{Cible} - O_{Cansat}$

Si $\Delta$O $> 0$, alors nous savons que nous devons orienter notre cansat vers l'ouest
Si $\Delta$O $< 0$, alors nous savons que nous devons orienter notre cansat vers l'est

Maintenant que nous savons comment nous orienter, nous devons trouver un moyen de commander les servomoteurs qui, nous le rappelons, permettent de tirer sur les câbles du parapente et ainsi tourner à droite si l'on tire à droite et d'aller à gauche si l'on tire à gauche.

Ci-dessous nous allons présenter les 4 figures qui représentent les 4 cas auxquels nous pourrions être confrontés. Dans un premier temps, nous ne savons pas comment va être parachuté notre Cansat et donc son orientation de départ. L'objectif de cette partie est de savoir quel servo (droit ou gauche) doit fonctionner pour atteindre la cible. Pour cela nous allons déterminer comment est orienté le vecteur Cible et le vecteur Can par rapport au nord magnétique et ainsi faire une différence d'angle pour pouvoir activer le bon servomoteur.

**Remarque** : L'angle dont nous parlerons dans la suite est $V_{Cible} - V_{Can}$.

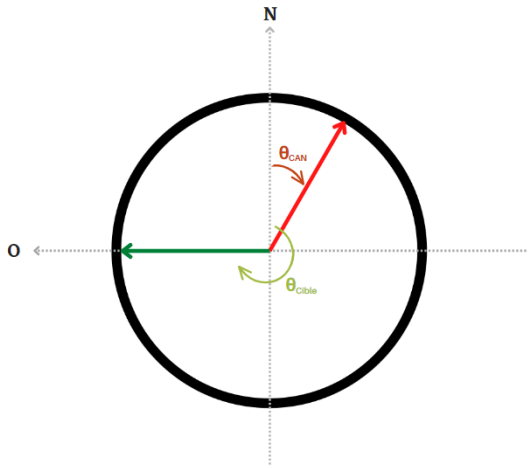


FIGURE 26 – Angle supérieur à 180°

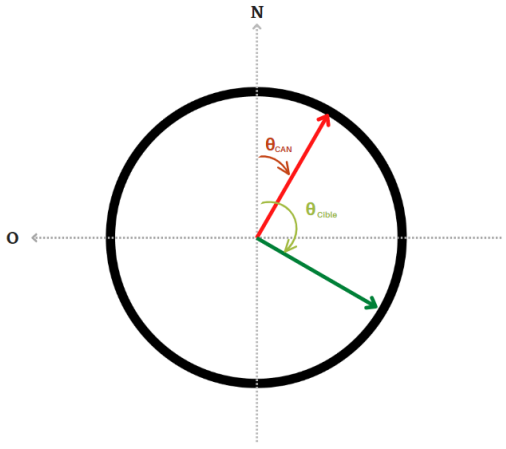


FIGURE 27 – Angle inférieur à 180° et supérieur à 0°

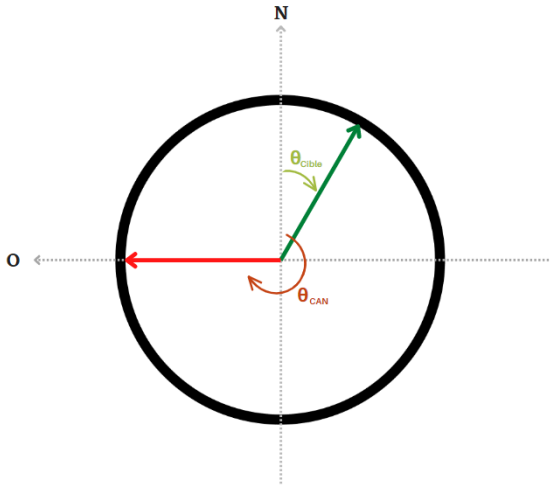


FIGURE 28 – Angle inférieur à -180°

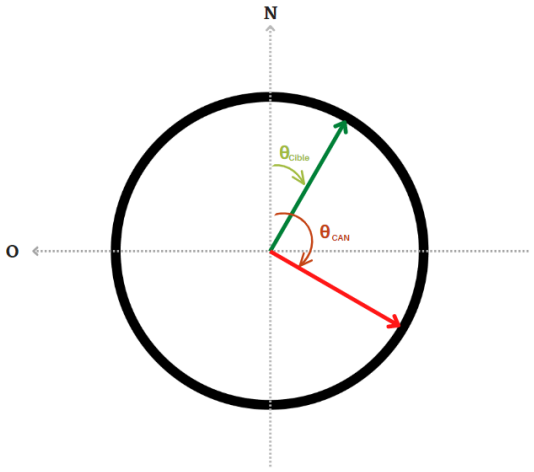


FIGURE 29 – Angle supérieur à -180° et inférieur à 0°

Dans le premier cas, figure 26, nous avons le vecteur Can qui est décalé d'un angle de 30° par rapport au nord et le vecteur cible qui est décalé de 270°. Nous avons donc une différence de 240° qui est supérieur à 180° ce qui signifie que nous devons faire tourner le cansat à gauche pour perdre le moins de temps possible donc faire fonctionner le servomoteur gauche.

Sur la figure 27, nous avons une différence d'angle de 90° $<$ 180° donc d'après notre raisonnement nous devons tourner à droite et donc faire fonctionner le servo de droite.

Avec cette solution, nous nous sommes rendu compte que cela ne fonctionnait pas tout le temps. En effet, pour une différence d'angle négative notre raisonnement ne fonctionne pas. Nous avons donc fait deux exemples contradictoires pour compléter notre réflexion.

Dans le 3ème cas, figure 28, nous inversons les deux vecteurs de la figure 1 , on obtient donc une différence de -250<-180 degrés. Ici, nous devons faire tourner notre servo de droite.

Enfin, dans le 4ème cas, figure 29, nous avons une différence de -90>-180° donc nous devons tourner à gauche.

**Conclusion :**

Si $\Delta\theta \in$ ]-360°$+ \varepsilon$ ; -180°[ alors nous devons aller à droite.
Si $\Delta\theta \in$ [-180° ; - $\varepsilon$ [ alors nous devons aller à gauche.
Si $\Delta\theta \in$ ] $+ \varepsilon$ ; 180° [ alors nous devons aller à droite.
Si $\Delta\theta \in$ [180° ; 360°- $\varepsilon$ [ alors nous devons aller à gauche.

De plus, nous avons décidé de tourner avec des intensités différentes pour optimiser nos déplacements et être le plus précis possible.

En effet, nous avons choisi d'appliquer 3 intensités différentes en fonction de la direction du cansat par rapport à la direction voulue.

Sur la figure ci-dessous sont représentées 4 catégories auxquelles nous avons donné des intensités différentes. Pour la zone verte, nous avons décidé de n'actionner aucun servo et de laisser le cansat dans la même direction. Si la direction du cansat se trouve dans la zone bleue, nous activons le servo concerné avec une intensité faible, c'est-à-dire que le bras descendra légèrement. Puis dans la zone jaune les bras seront encore plus bas et pour la zone rouge le bras sera totalement vers le bas.

**Remarque** : si nous voulons freiner et faire descendre le cansat quasiment à pic, nous devons abaisser les deux bras simultanément.

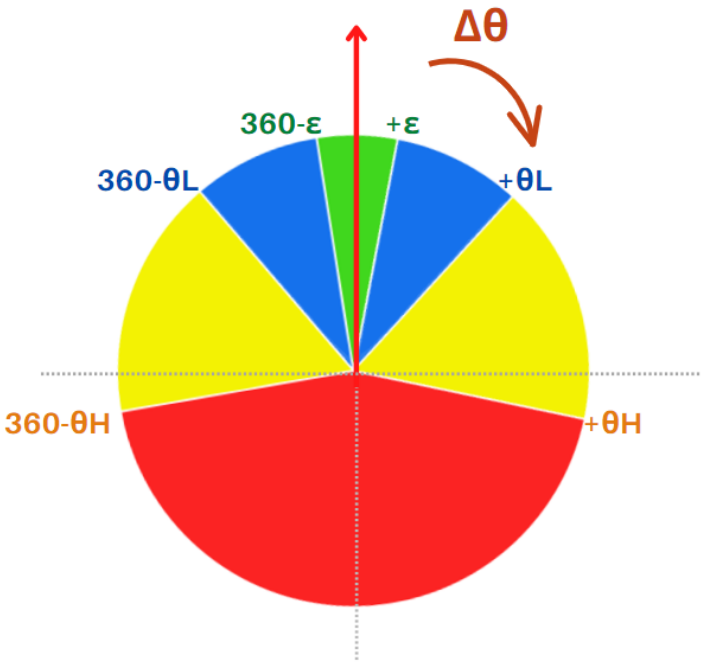


FIGURE 30 – Intensité en fonction de la direction du Cansat par rapport à la direction souhaitée

## 3.6 Solution Software

Concernant le code permettant l'asservissement de la direction du Cansat, voici l'algorigramme récapitulatif :

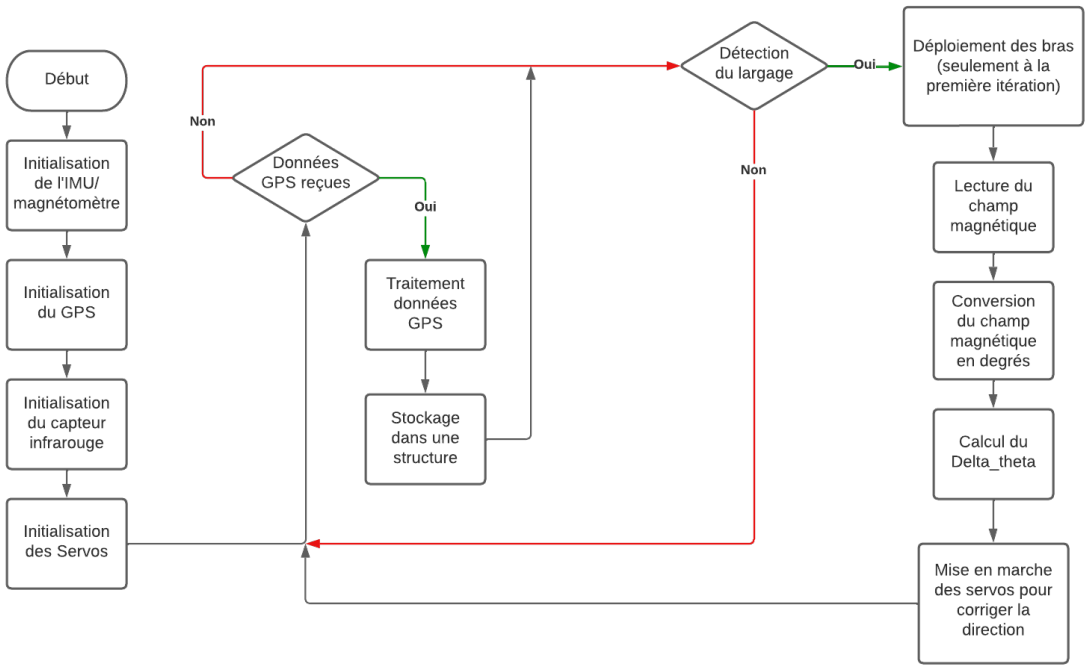


FIGURE 31 – algorigramme pour le code de la mission principale

Lors de la première itération de la fonction main, on commence par initialiser tous les drivers : IMU, GPS, capteur infrarouge et servos. Par initialisation des servos, il s'agit de les placer dans une position qui permette au Cansat de rentrer dans la capsule de largage (en effet, si les bras son déployés, le Cansat ne contient pas dans la capsule). Lorsque l'initialisation est terminée, on rentre dans la boucle while infinie. La première étape d'une itération de cette boucle est de vérifier si des données GPS ont été reçues (en utilisant un flag placé dans la fonction de callback de l'UART, en interruption donc). Si tel est le cas, la chaîne de caractère est traitée, les données utiles (longitude, latitude et altitude) en sont extraites et sont alors stockées dans une structure, cf figure **??**

Pour appliquer la théorie que nous avons abordé précédemment, nous avons créé deux fonctions pour manœuvrer les servos (servoSetPositionLeft() et servoSetPositionRight()). Une fonction pour chaque servo, ainsi qu'une fonction pour choisir la position des servos en fonction de l'angle donné par le magnétomètre (choice_direction_intensity()).

Dans la première fonction, selon la position que l'on donne en entrée, le servo gauche se place. Le choix de la position se fait dans le code de la fonction.

En effet, en entrée de cette fonction, nous avons $\Delta\theta$, fourni par le magnétomètre. Selon la valeur de cette variable, en appliquant la partie théorique que nous avons développé précédemment, nous pouvons savoir quel servo contrôler et à quelle intensité. De plus, chaque intensité correspond à la position d'un servomoteur. Une fois le servo choisi (gauche ou droite) et la position que l'on veut pour le bras, nous utilisons la fonction servoSetPositionLeft() ou servoSetPositionRight() .

# 4 Mission facultative : lecture d'un QR code

La reconnaissance de QR code est effectuée à l'aide d'une caméra reliée à carte Raspberry Pi. Nous avons fait ce choix car la prise en main de la caméra est très simplifiée sur Raspberry. De plus, il est facile d'enregistrer les données issues du QR code sur la carte SD de la Raspberry.

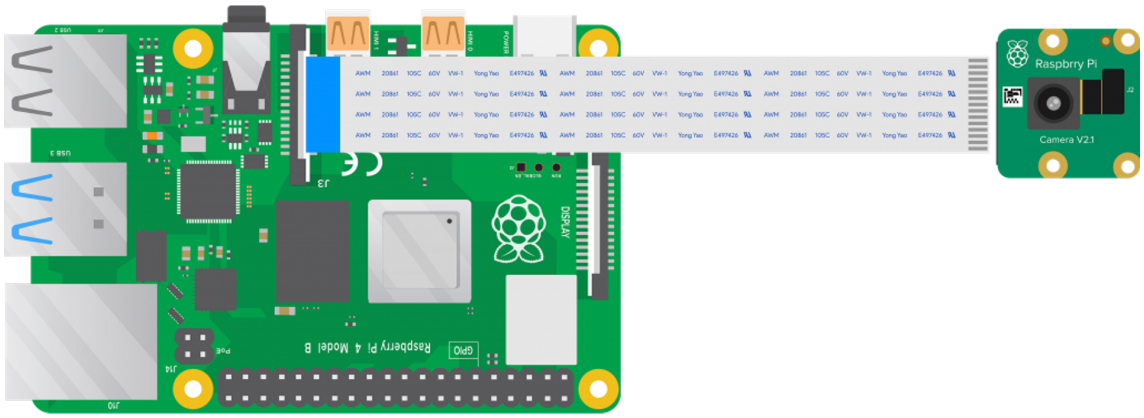


FIGURE 32 – Illustration de la rasperry pi avec camera

→ Raspbery Pi camera V2 OV2640
→ Raspberry pi 3 Model B+

## 4.1 Reconnaissance du QR code et enregistrement

Pour reconnaître les QRcode et déchiffrer les informations qu'ils contiennent, nous utilisons le logiciel openCV qui permet de réaliser du traitement d'image en temps réel. Nous importons l'ensemble des paquets relatifs à openCV dans un script python. Il suffit ensuite d'activer les périphériques de la caméra, de récupérer les images et les fonctions propres à openCV renvoient les informations relatives au QR code. Nous enregistrons automatiquement ces informations sous forme de texte dans un fichier csv qui est donc lui-même enregistré sur la carte micro-SD de la Raspberry.

## 4.2 Lancement automatique du QR code au démarrage de la carte

Il faut ensuite trouver une solution pour pouvoir lancer le code de lecture de QRcode automatiquement, au démarrage de la carte. Sinon, nous sommes contraints de lancer le code manuellement à l'aide d'un écran.

Tout d'abord, il faut créer un fichier shell contenant les commandes à exécuter pour lancer le script python. Ensuite, il faut rendre ce fichier exécutable. Puis on crée un dossier « logs » qui contiendra les différents fichier logs du programme. Ce dernier nous permettra de connaître les erreurs d'exécution de notre programme au démarrage s'il y en a.

Ensuite, nous utilisons le crontab. Il s'agit d'un processus qui permet à l'utilisateur d'exécuter des scripts en tâche de fond à des moments précis (minute, heure, jour...) ou évènements précis. Pour nous, il s'agira de lancer l'exécution de notre shell au démarrage. Avec cette méthode, lorsque nous allumons la Raspberry, le script est lancé automatiquement, et lorsque nous présentons un QR code à la caméra, nous récupérons bien un fichier csv enregistré dans la carte micro-SD.

## 4.3 Test de lecture

Nous réalisons plusieurs tests de lecture avec plusieurs types de QR code au format A4 en réglant l'ouverture de la caméra au maximum. La distance maximale permettant une reconnaissance est d'environ 1m75. Cela n'est pas assez car le Cansat risque de ne pas être exactement au-dessus de la cible à cette hauteur. Cependant, la taille des QR code au moment de la compétition ne sera pas A4 mais A2, donc la distance maximale sera probablement plus grande. Cependant, une taille de cible plus importante n'a pas un impact conséquent car quoi qu'il en soit, les capacités de la caméra que nous utilisons restent insuffisantes. Son ouverture maximale est trop faible pour distinguer des formes et des objets à une distance de plus de quelques mètres. Il serait judicieux de se tourner vers un modèle de caméra plus performant.

## 4.4 abandon de la mission

Nous avons finalement décidé de ne pas réaliser cette mission facultative car elle rendait l'intégration du Cansat impossible. Tout d'abord, la taille de la raspberry est conséquente pour les dimensions du Cansat. L'ensemble du matériel ne contenait que très difficilement et le compactage de l'électronique aurait très probablement entraîné des problèmes de CEM. Mais surtout, la Raspberry consomme beaucoup trop de courant. Avec une caméra branchée, elle peut consommer jusqu'à 3A. Pour fournir un tel courant, et sur la durée qui plus est car la reconnaissance de QRcode tourne en continue, il faut une batterie de taille bien trop importante pour contenir dans le Cansat.

# 5 Mission facultative : déploiement d'une structure au sol

Le cahier des charges étant relativement succinct, le choix est large quant au type de solution envisageable. Afin de maximiser la surface déployée tout en minimisant le volume global, nous avons décidé de nous tourner vers un système gonflable.

## 5.1 Elaboration de la structure du système de gonflage

Le système est constitué d'une vanne pneumatique, d'un percuteur, d'un servomoteur et d'une capsule de CO2 (cf figure 33).

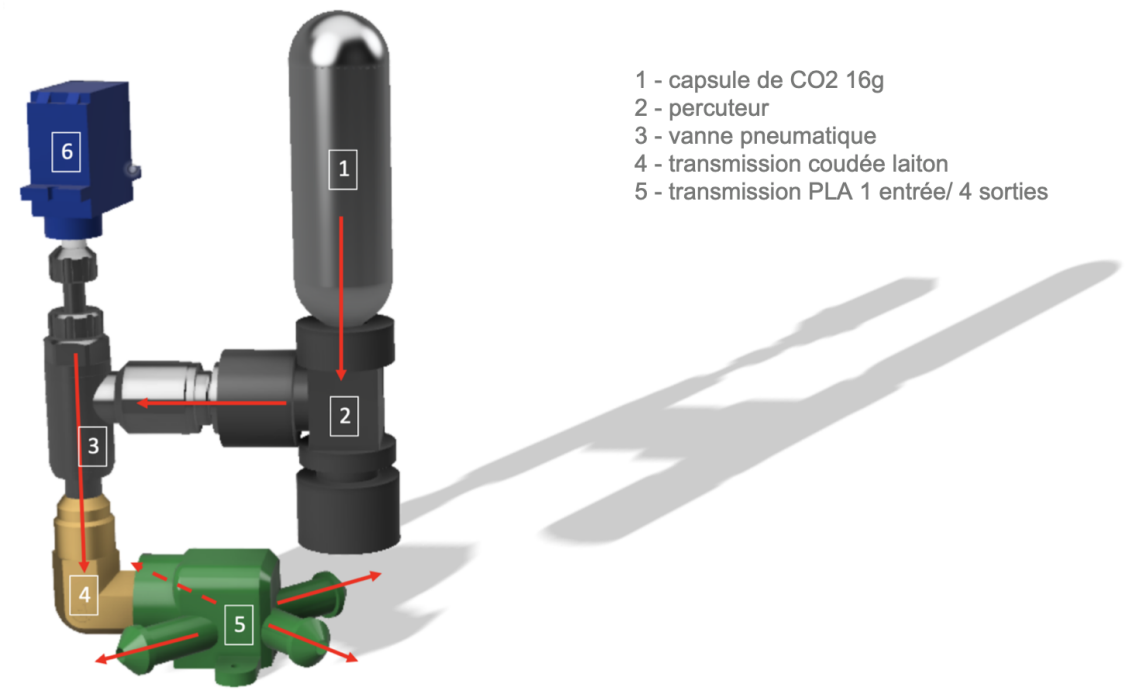


FIGURE 33 – structure de la partie mécanique du sous-système de déploiement gonflable

La capsule de CO2 est vissée au percuteur qui est lui-même soudée à la vanne pneumatique. Lorsque l'on visse la capsule au percuteur, cette dernière est percée et laisse s'échapper le gaz qui est contenu tant que le robinet n'est pas ouvert. Le servomoteur contrôle l'ouverture de la vanne (il est relié à un MCU). Lorsque le servomoteur est activé, la vanne s'ouvre, laissant s'échapper le gaz.

### 5.1.1 élément de transmission vers les ballons

Il nous a été très difficile de trouver une pièce mécanique permettant de redistribuer une entrée de gaz vers 4 sorties afin d'être en mesure de gonfler les 4 ballons. Nous avons pris la décision de concevoir cette pièce nous même en impression 3D. Le PLA étant peu hermétique, nous nous sommes tournés vers un autre procédé utilisant de la

résine polymère, plus laborieux et onéreux mais la pièce obtenue présente une bonne imperméabilité aux gaz.

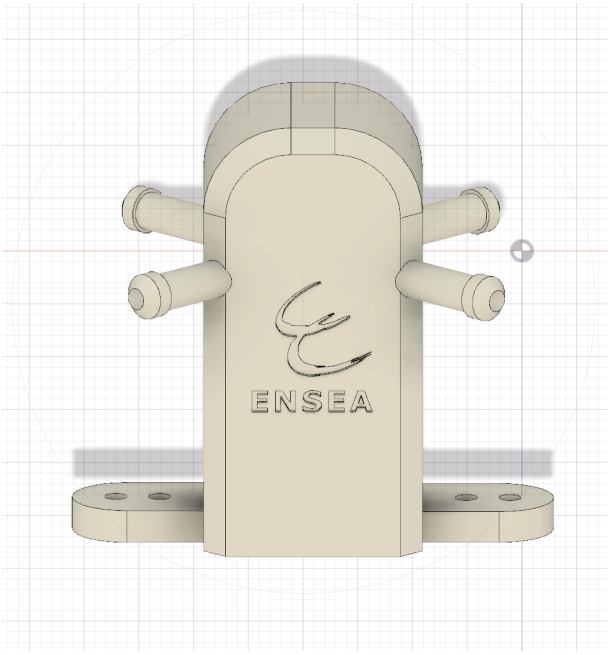


FIGURE 34 – pièce en résine pour la transmission vers les ballons, vue de face

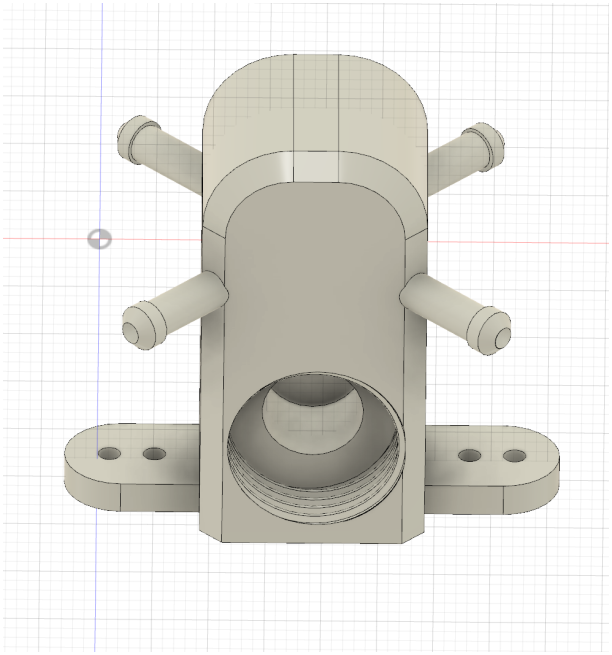

FIGURE 35 – pièce en résine pour la transmission vers les ballons, vue de l'arrière

### 5.1.2 choix des ballons et conception du système déployable

## 5.2 Système embarqué pour le gonflage de la structure

Dans un deuxième temps, il est nécessaire de concevoir un système embarqué afin de déclencher l'ouverture du système gonflable (par l'ouverture du robinet via le servo) lorsque la structure est suffisamment proche du sol. Pour ce faire, nous avons choisi d'utiliser l'altitude fournie par le GPS. Or cette donnée est susceptible d'être très imprécise, ainsi nous la couplons avec les données issues du capteur de pression situé sur le MPU-9250. Le système tourne sur le même MCU que pour la mission principale.

Le MCU reçoit les données d'altitude du GPS toutes les secondes. C'est un temps d'acquisition qui nous paraît acceptable pour l'application présente car le module freiné par le parachute descend à une vitesse de l'ordre de 1 m/s.

### 5.2.1 Utilisation du capteur de pression

- **Identification du BMP280**

L'identification du BMP280 consiste en la lecture du registre ID. Dans un premier temps nous envoyons l'adresse du composant et celle du registre puis nous lisons dans le registre. D'après la datasheet, nous devons obtenir la valeur 0x58, ce qui est bien le cas :

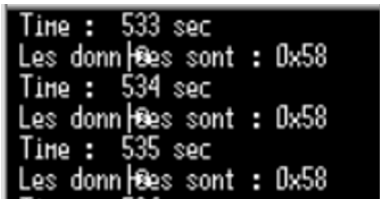


FIGURE 36 – Illustration de l'écran de réception de l'UART

**• Configuration du BMP280 et récupération de l'étalonnage, de la température et de la pression**

- Registres contenant la pression (ainsi que le format) :

| Register 0xF7-0xF9 *"press"* | Name | Description |
|---|---|---|
| 0xF7 | press_msb[7:0] | Contains the MSB part up[19:12] of the raw pressure measurement output data. |
| 0xF8 | press_lsb[7:0] | Contains the LSB part up[11:4] of the raw pressure measurement output data. |
| 0xF9 (bit 7, 6, 5, 4) | press_xlsb[3:0] | Contains the XLSB part up[3:0] of the raw pressure measurement output data. Contents depend on temperature resolution, see table 5. |

FIGURE 37 – Registre des données de pression

Pour faire une mesure il faut configurer le registre ctrl_meas. On écrit donc dans le registre 0xF4 la valeur 0x57. Une fois les données brutes récupérées, il faut les traiter afin d'obtenir les données utiles. Pour cela, on utilise les formules fournies par la datasheet, figure 38.

| | | | |
|---|---|---|---|
| var1 = | 128793,1787 | | var1 = (((double)adc_T)/16384.0 - ((double)dig_T1)/1024.0) * ((double)dic_T2); |
| var2 = | -370,8917052 | | var2 = ((((double)adc_T)/131072.0 - ((double)dig_T1)/8192.0) * (((double)adc_T)/131072.0 - ((double) dig_T1)/8192.0)) * ((double)dig_T3); |
| tfine = | 128422 | | t_fine = (BMP280_S32_t)(var1 + var2); |
| T = | 25,08 | *Temperature [°C]* | T = (var1 + var2) / 5120.0; |
| integer result (**): | 2508 | *Temperature [1/100 °C]* | |
| | | | |
| var1 = | 211,1435029 | | var1 = ((double)t_fine/2.0) - 64000.0; |
| var2 = | -9,523652701 | | var2 = var1 * var1 * ((double)dig_P6) / 32768.0; |
| var2 = | 59110,65716 | | var2 = var2 + var1 * ((double)dig_P5) * 2.0; |
| var2 = | 187120057,7 | | var2 = (var2/4.0)+(((double)dig_P4) * 65536.0); |
| var1 = | -4,302618389 | | var1 = (((double)dig_P3) * var1 * var1 / 524288.0 + ((double)dig_P2) * var1) / 524288.0; |
| var1 = | 36472,21037 | | var1 = (1.0 + var1 / 32768.0)*((double)dig_P1); |
| p = | 633428 | | p = 1048576.0 - (double)adc_P; |
| p = | 100717,8456 | | p = (p - (var2 / 4096.0)) * 6250.0 / var1; |
| var1 = | 28342,24444 | | var1 = ((double)dig_P9) * p * p / 2147483648.0; |
| var2 = | -44875,50492 | | var2 = p * ((double)dig_P8) / 32768.0; |
| p = | 100653,27 | *Pressure [Pa]* | p = p + (var1 + var2 + ((double)dig_P7)) / 16.0; |
| int32 result (**): | 100653 | *Pressure [Pa]* | |
| int64 result (**): | 25767236 | *Pressure [1/256 Pa]* | |

(**) The actual result of the integer calculation may deviate slightly from the values shown here due to integer calculation rounding errors

FIGURE 38 – Calcul de la pression

Les fonctions permettant le calcul de la température et de la pression compensées, en format entier 32 bits sont utilisées pour convertir les données brutes en données utiles.

- Dans un deuxième temps, il s'agit de transformer les données brutes en données utiles. Pour cela, nous créons un ensemble de structures pour stocker les valeurs brutes des données ainsi que les valeurs d'étalonnage et les valeurs finales utiles.
- Ensuite, nous envoyons des requêtes de lecture pour récupérer ces données puis nous les stockons dans les structures créées précédemment.

## 5.3 Conclusion partie structure gonflable

Une fois le système conçu et assemblé, nous avons procédé à des essais techniques. Certains problèmes récurrents nous ont forcé à repenser la structure :

Tout d'abord, le système mécanique n'est pas suffisament hermétique, il laisse s'échapper trop de gaz. Pour faire face à ce problème, nous appliquons du mastic de polymère autour des zones critiques, ce qui évite en grande partie les fuites.

# 6 Intégration

## 6.1 Structure du Cansat

Afin de concevoir la structure du Cansat, nous avons dû penser à chaque éléments du système et à l'emplacement le plus approprié en prenant en compte leur volume, leur propriétés et les exigences intrinssèques à leur fonctionnement.

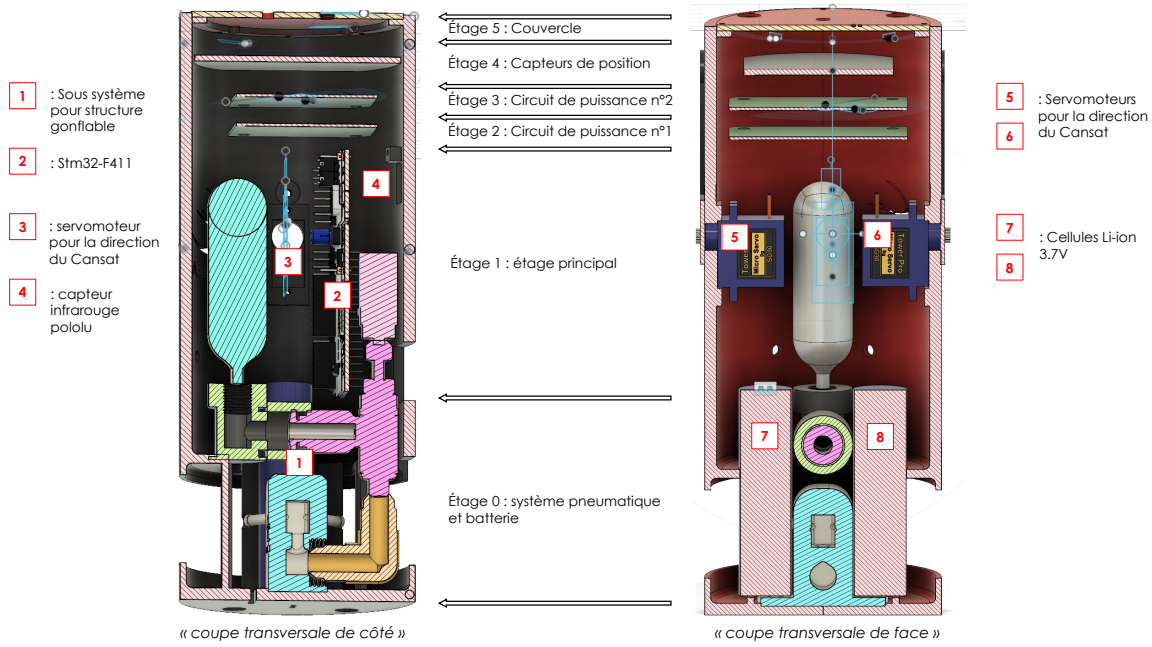


FIGURE 39 – vues en coupe du Cansat

## 6.2 software

## 6.3 Puissance

Notre projet mettant en oeuvre un système voué à être embarqué, nous allons devoir l'alimenter grâce à une batterie. Nous avons alors besoin de concevoir un PCB de puissance qui aura comme sortie des tensions d'alimentation permettant d'alimenter notre carte ainsi que nos composants. Pour cela, nous aurons besoin de régulateurs de tensions qui renvoient 5 et 3.3V.

### 6.3.1 Choix des composants

- **Batterie** : Lipo 11.1V ; 3 cellules ; 1600 mAh
- **Interrupteur** : Le cahier des charges nous impose d'avoir un interrupteur pour l'alimentation de tout le cansat pour que l'on puisse mettre tout hors tension s'il y a un problème. Ce dernier doit être accessible facilement depuis l'extérieur du cansat.
- **Régulateurs de tenion** :

– **L7805** : Pour la STM et les 2 servomoteurs
– **L78L33** : Pour l'IMU

En effet, nous avons besoin d'un régulateur L7805 qui renvoie du 5V car la carte et les deux servos sont alimentés en 5V et d'un régulateur L78L33 qui renvoie du 3.3V pour l'IMU. Le GPS sera directement alimenté depuis la carte STM.

### 6.3.2 Conception du PCB

La figure ci-dessous, représente globalement ce que nous voulons faire sur notre PCB de puissance.

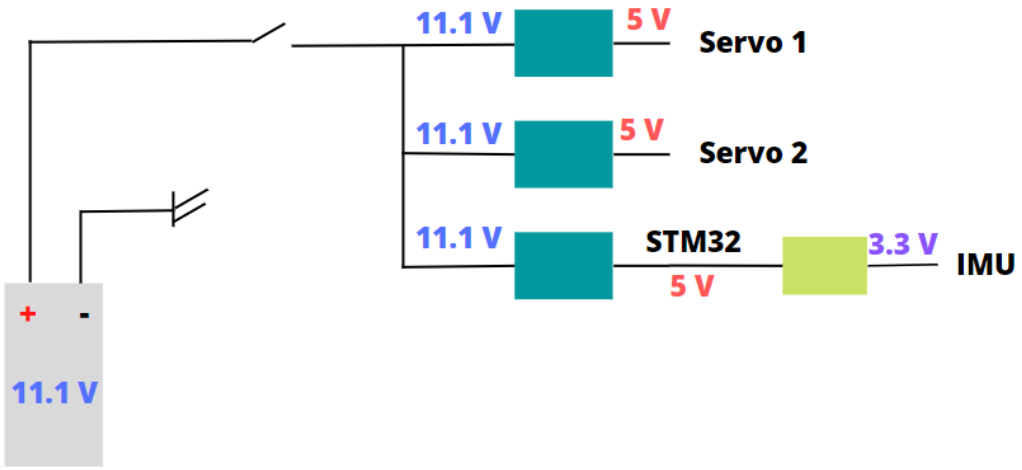


FIGURE 40 – Schéma global pour la conception du PCB de puissance

Nous avons choisi de placer le régulateur de l'IMU après le régulateur de la STM32 qui renvoie 5V pour ne pas faire trop chauffer le régulateur L78L33 afin qu'il ne soit pas endommagé.

D'après la documentation du régulateur L78, nous devons ajouter des condensateurs à notre montage comme montré ci-dessous :

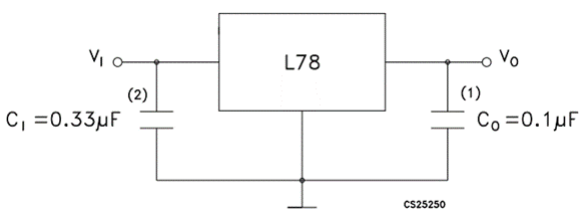


FIGURE 41 – Montage à réaliser d'après la documentation

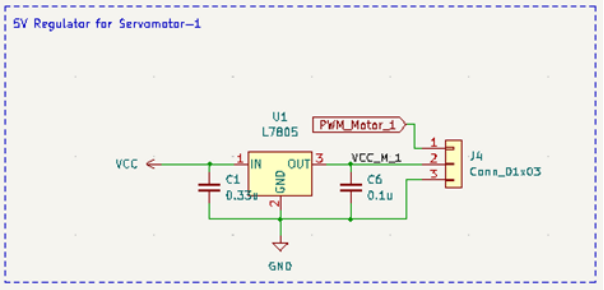


FIGURE 42 – Exemple du montage réalisé pour le servo 1

Finalement, nous obtenons le PCB de puissance suivant :

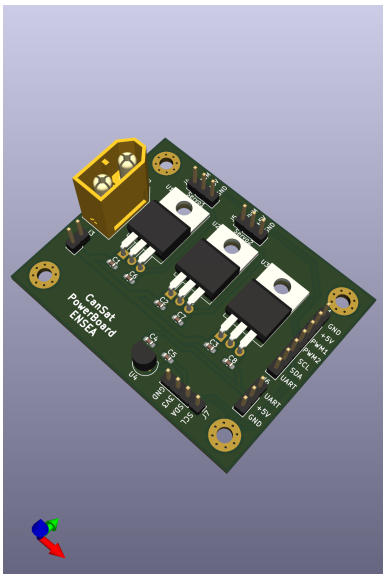


FIGURE 43 – PCB pour l'alimentation du Cansat

# Conclusion

Nous avons eu la chance de participer à la campagne 2023 du C'space à l'issu de laquelle nous avons qualifié notre Cansat qui a donc pu être largué. Malheureusement, certain aléas ont dégradé les conditions de lancement, le drone de largage a été détruit le matin des lancements, nous avons cependant pu larger notre sonde mais a une hauteur relativement dérisoire (50m au lieu de 180m) et à seulement 10m de la cible. Nous n'avons donc pas pu vérifier le bon fontionnement du système de direction autonome. Et le système de gonflage s'est brisé à l'atterrissage. Cependant nous avons tout de même pu admirer notre Cansat en plein vol et ce fut un réel plaisir.

# Références

[1] Planète science, 2023 : Règlement du concours CanSat France 2023 [Équipe Planète Sciences : Alain Ravissot, Alexia Le Gall, Flavien Denis, Francis Qerimi, Henri Le Barbenchon, Sanduni Mataraarachchige. Équipe CNES : Christian Planes.].

[2] L. Giraudeau, Georges Perrot, S. Petit, Bernard Tourancheau. Simulation d'écoulement tridimension- nel autour d'une aile de parapente.. [Research Report] LIP RR-1996-35, Laboratoire de l'informatique du parallélisme. 1996, 2+29p. hal-02101920

[3] Les parachutes des fusées expérimentales - Edition Février 2002 - Note technique ANSTJ - Patrick ROMMELUERE, Arnaud COLMON, Guy PREAUX, CRM - Département Education-Jeunesse du CNES - ANSTJ Secteur Espace

[4] Documentation technique ICM-20948 - TDK - World's Lowest Power 9-Axis MEMS MotionTracking Device - Document Number : DS-000189 Revision : 1.5 Release Date : 09/02/2021

# Annexes

# Structure globale du Cansat

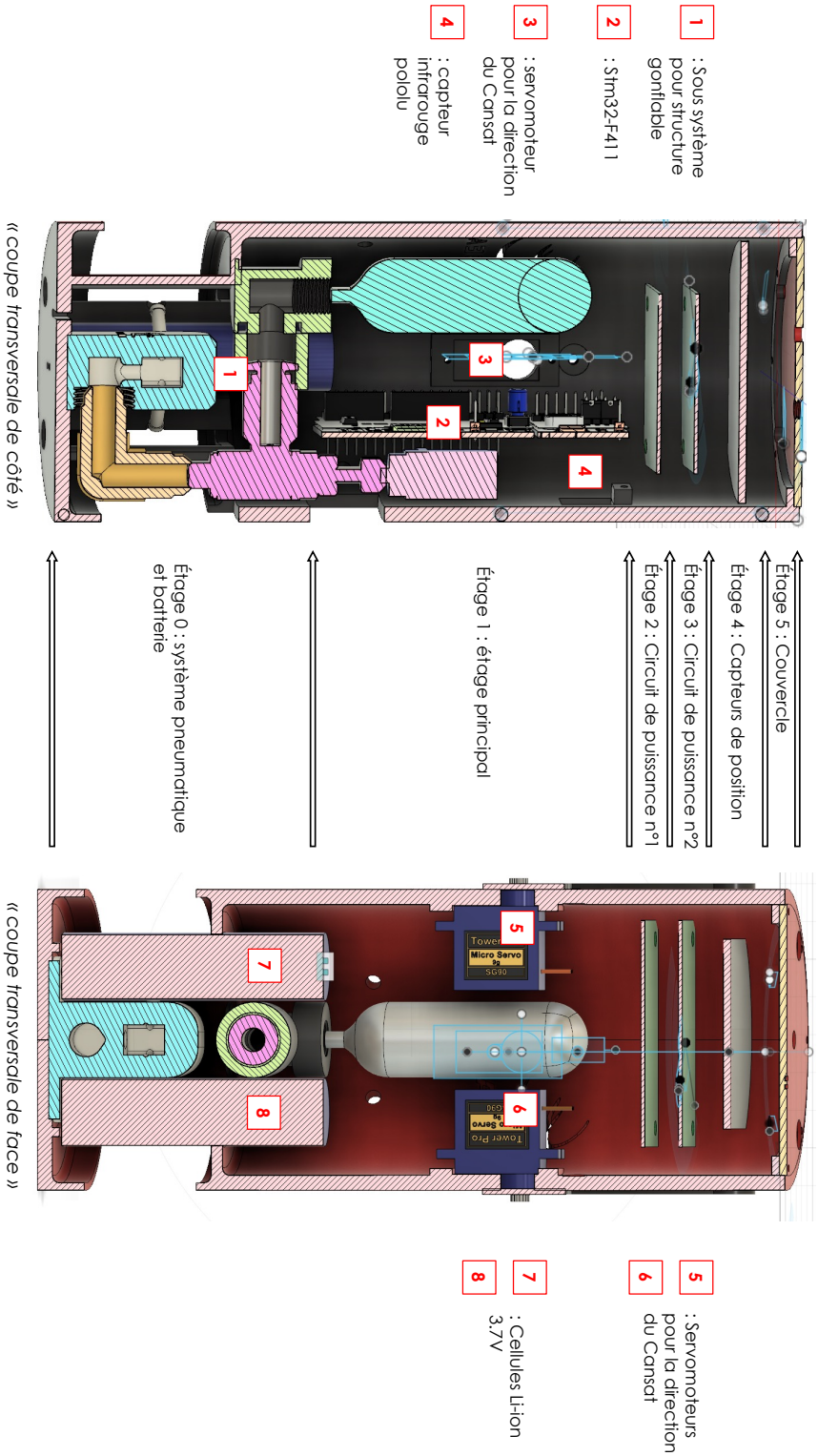


FIGURE 44 – vues en coupe du Cansat

# Plan de la partie électronique du Cansat

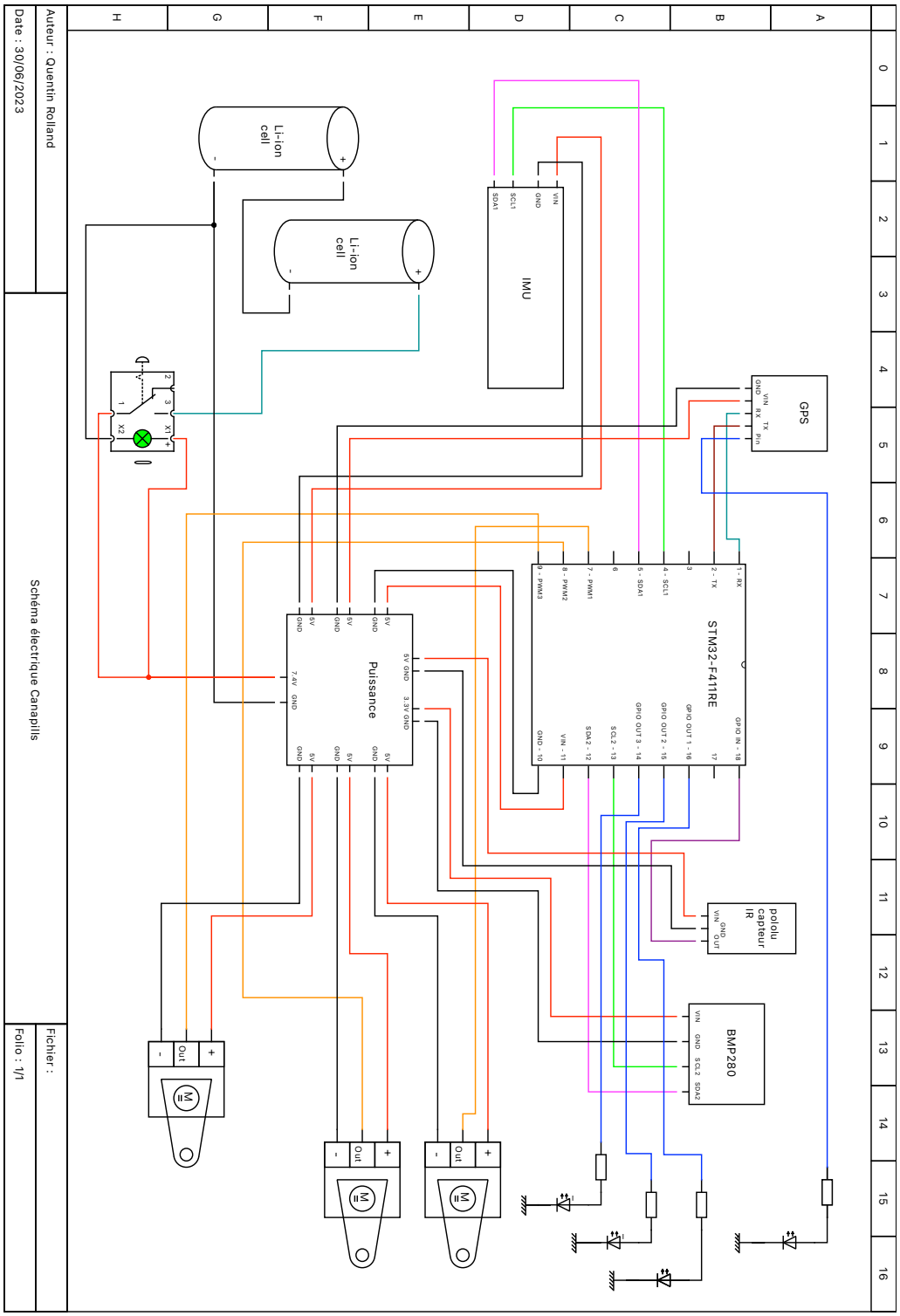


FIGURE 45 – Schema des connexions electriques du Cansat

# Plan de la structure du sous-système de gonflage

# Puissance

### 6.3.3 empreinte du pcb de puissance

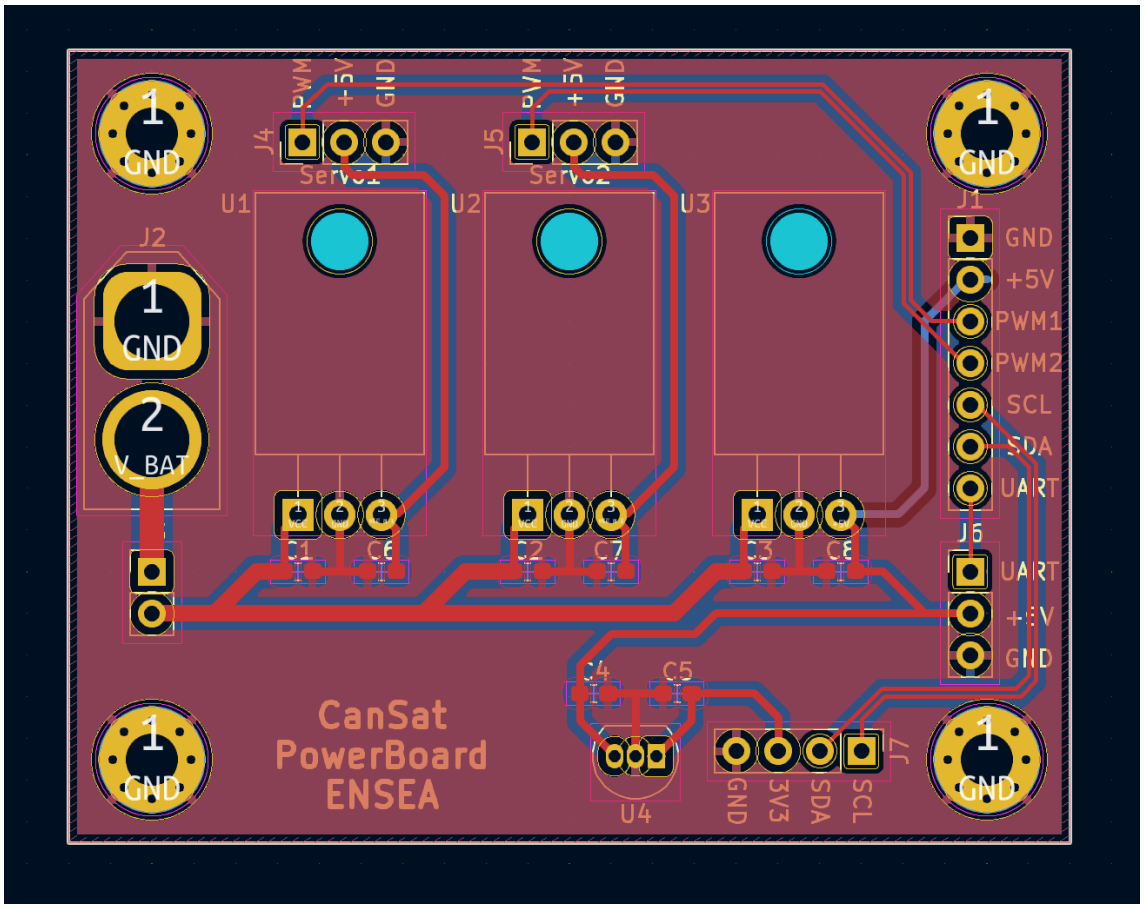


FIGURE 46 – Image de l'empreinte du Cansat powerboard

### 6.3.4 composants de la partie puissance

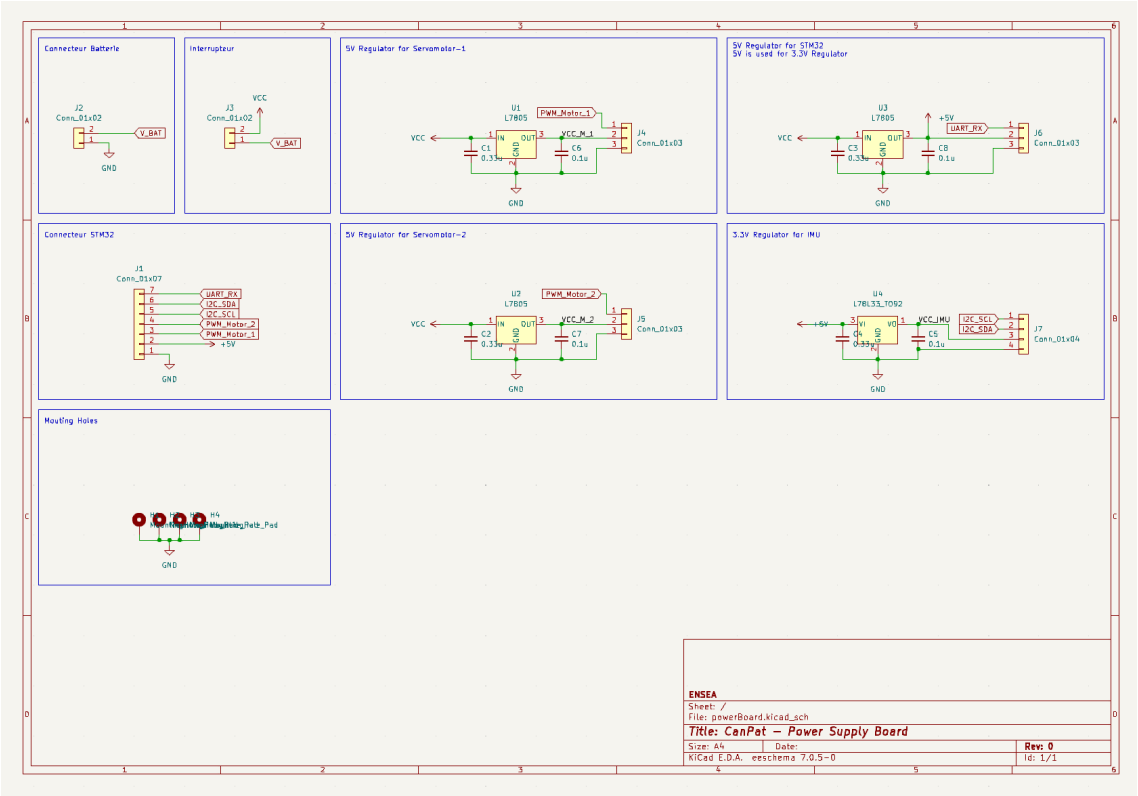


FIGURE 47 – différents éléments composant le pcb puissance