UNIVERSITÉ DE BORDEAUX

SCIENCES ET TECHNOLOGIES

MASTER 2 INFORMATIQUE ASPIC

PROJET DE FIN D'ÉTUDES

Rapport : Rendu PFE

FERNANDES LOPES João Manuel, Cantelobre Gaetan, Aabo-Aljaloo Taif

24 Février 2025

Table des matières

1 Introduction

Au cours du dernier semestre de l'année universtiaire Master 2 ASPIC de l'Université de Bordeaux, nous devons faire la validation de l'UE PFE et MOCPI, un projet de fin d'études. Plusieurs groupes de 2 à 3 personnes doivent se répartir plusieurs sujets présentés par différents *clients*. Au cours du semestre deux UEs se partagent le sujet ; d'une part MOCPI qui est l'UE qui va superviser l'organisation et la forme du projet, en surveillant la répartition des tâches, leur organisation, l'harmonie du code etc... D'une autre part l'UE PFE qui va être l'aspect "*professionnel*" du projet, où l'important du rendu va être le résultat présenté au client, la prise de rendez-vous avec le client, ainsi que la démonstration finale présentée.
L'évaluation sera faites principalement sur deux rendus ; un premier qui est le présent rapport abordant chaque aspect du projet, ainsi qu'une présentation orale d'environ 15 minutes accompagnée d'un support.

2 Contexte

2.1 Sujet et Objectifs

Notre sujet s'intitule *Drone Porteur*, un sujet qui vise à faire les aménagements nécessaires sur le *DJI Matrice300* et lui permettre de porter en dessous de lui des drones plus petits comme le *DJI Mini 3*, ou le *DJI Tello Edu*. Un travail de réécriture du sujet à été fait lors des premiers rapport, et peut être présenté de cette façon : L'objectif est d'avoir le *Matrice300* qui décolle avec plusieurs drones plus petits comme le *Tello Edu*. Dans ce but, nous devons alors concevoir une nacelle attachée au *Matrice300* sur laquelle nous pouvons attacher les plus petits drones. Une fois le décollage fait, nous devons pouvoir amorcer un largage des petits drones en plein vol depuis le *Matrice300*, sans l'interrompre. Une fois le largage fait, les *Tello Edu* commencent alors leur vol, s'engagent dans une mission prédéfinie, et commencent leur atterrissage.

2.2 Etat de l'Art

De nos jours, il existe très peu de solutions adoptant le concept de *carrier drone*. On peut cependant citer le *Loyal Wingman* un drone collaboratif conçu pour accompagner des avions de chasse pilotés, comme le *F-35*. Son objectif est d'améliorer la capacités des avions en matière de détection de cibles, reconnaissance et d'attaque.

Un autre exemple est celui des drones utilisés pour les attaques aériennes, comme le *MQ-1B*, qui peut larguer des missiles guidés sur des cibles précises. Dans ce cas, on peut considérer le *MQ-1B* comme un drone porteur, car ses missiles sont automatisés et programmés pour atteindre une cible spécifique.

3 Résultats Obtenus

Au cours de l'avancée du projet, nous avons pu obtenir plusieurs résultats, certains qui nous ont rapprochés de l'objectif du projet et d'autres qui nous en ont éloignés. Dans cette section, nous aborderons spécifiquement ceux qui correspondaient à l'objectif fixé, accompagnés d'une démonstration montrant l'essentiel du fonctionnement de notre travail.

3.1 Objectifs Accomplis

Lors de la structuration des tâches à accomplir au cours de notre projet, nous nous sommes fixés plusieurs objectifs majeurs à atteindre, qui sont les suivants :

- L'un des premiers objectif lors de l'acquisition du *Tello Edu*, était de comprendre son fonctionnement et d'avoir de premiers mouvements basiques pour contrôler le drone. Objectif atteint rapidement avec l'étude de l'API du drone.
- Un autre défi majeur du projet était le design des différents PCB qui intégreront le *Matrice300*, faire le fonctionnement du servomoteur qui permettra le largage du drone ainsi que le design des différentes pinces, accroches, etc. Ce deuxième défi a pu être complété après quelques semaines d'essais et de travail.
- Après avoir terminé le design des PCB et des différentes parties, il était important de les faire communiquer entre elles et veiller à la robustesse de leurs échanges. Objectif rapidement atteint après avoir fini le design.
- L'un des derniers objectif était maintenant de vérifier si l'ensemble fonctionne bien, et de faire alors un premier essai qui montrerait que le cœur du projet était accomplit. Après assemblage d'une première démonstration, nous avons accompli l'un de nos derniers objectifs fixés, avoir le chargement du drone effectué correctement, une commande de largage donnée à la nacelle, suivi du drone qui est largué et qui décolle en vol.

3.2 Démonstration

Au cours du projet, une première démonstration à été faites le 11 Mars 2025. Dans cette démonstration on peut observer plusieurs éléments importants. On peut voir sur la démonstration un pilier qui supporte le montage et simule une altitude de vol pour le *Matrice300*, et sur lequel est attaché le reste du montage.

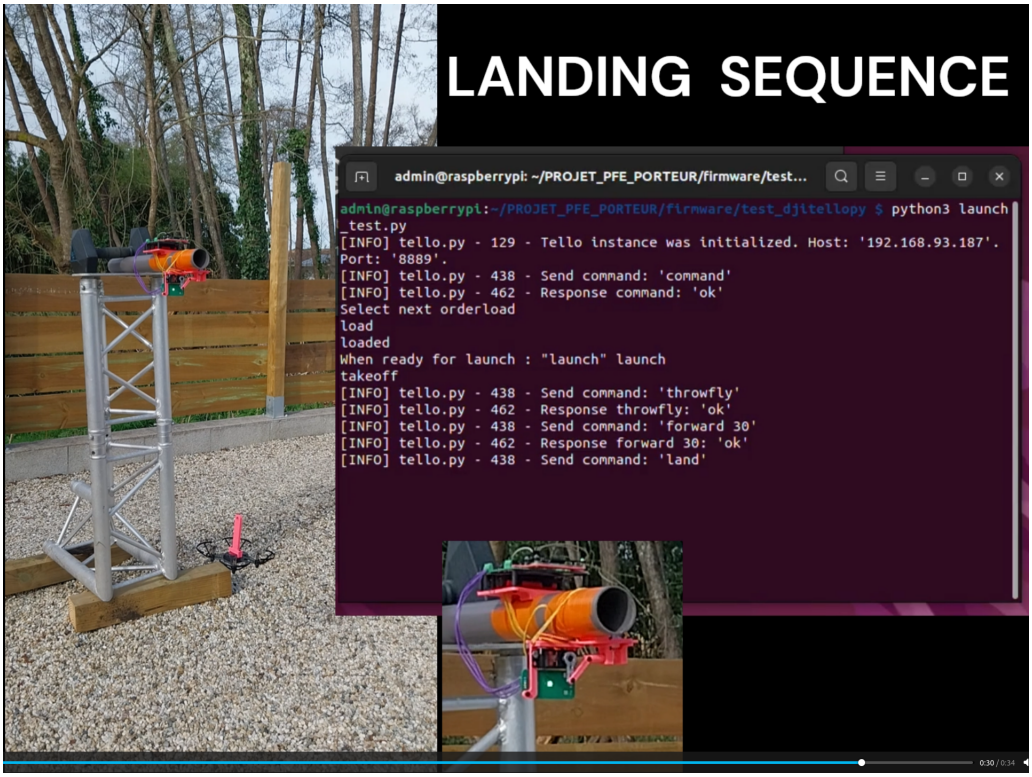

FIGURE 3.1 – Démonstration fonctionnelle du projet v1

Ci-dessus, on peut observer la mise en place de la démonstration. L'ensemble des PCB, la pince, le servo et le reste du matériel est monté sur un tube PVC au sommet du pilier. On voit sur la console que l'on peut donner différentes commandes, comme initier le chargement du drone, ou encore son largage. Après avoir chargé le drone, il est en attente d'ordre, on lui donne alors la commande de largage, et en autonomie le drone initie le vol, avance de 30 centimètres et atterrit en envoyant un message de confirmation que la mission est terminée correctement.

La version précédente est fonctionnelle, et met bien en valeur l'objectif du projet en finalité. Cependant, certaines pièces à ce moment là pouvaient encore être peaufinées pour assurer un fonctionnement encore meilleur lors des démonstrations, comme par exemple le format de la goupille qui maintient le drone en place.

FIGURE 3.2 – Démonstration fonctionnelle finale du projet

Ci-dessus, on peut notamment observer un changement sur la goupille, comme mentionné précédemment, ainsi que l'attache placée sur le drone, réduite en taille pour intérférer le moins possible avec l'aerodynamisme du drone par exemple. Sur cette démonstration finale, en plus d'avancer et d'attérrir, le drone fait un flip latéral gauche et latéral droit en plus.

Toutes ces différentes démonstrations sont consultables en vidéo à l'adresse suivante : Site PFE Drone Porteur (https ://joao-fernandes-lopes.emi.u-bordeaux.fr/ASPIC/PFE/website/)

4 Développement

Au cours de ce projet, nous avons dû travailler sur divers domaines, qu'il s'agisse du logiciel, du matériel ou même de la conception.

4.1 Electronique

4.1.1 Définition du materiel nécessaire

Dans l'optique de répondre correctement à la problématique et de présenter un "PROOF OF CONCEPT" plus portable, il était dans notre intérêt de trouver une solution semi-permanente qui peut être manipulée et modifiée facilement ainsi qu'à être portée par un drone. Lors des discussions avec le client, nous avons pu définir les spécificités des attentes : porter, lâcher et contrôler 4 drones depuis un drone porteur. Pour cela, nous avons développé une carte de contrôle principale ainsi que des cartes d'alimentation pour chaque mécanisme de porter/lâcher.

4.1.2 Spécifications carte de contrôle et cartes d'alimentation

La carte de contrôle principale doit être capable de piloter 4 servo moteur afin de contrôler les différents systèmes d'accroche des drones, et pour cela, nous avons utilisé un RP2040 sur un Pi Pico. Il était aussi nécessaire de communiquer avec en serial à l'aide d'un Pi Zero W 2 afin d'assurer la communication via Wi-fi (SSH) avec la nacelle.

La communication visuelle étant un point important afin de pouvoir observer et déterminer l'état et le fonctionnement de la nacelle sans être obligé de passer par une communication type STDOUT, nous avons opté pour l'utilisation de LED adressable WS2812B sur les cartes d'alimentation des servos.

Un convertisseur de niveau de logique TXS0108E peut être nécessaire en fonction des servos et des microcontrôleurs utilisé afin de passer d'une logique 3.3 V aux 5 V et vice-versa. Dans notre cas, il n'est pas utilisé, mais dans la logique d'une plus grosse compatibilité et pour s'adapter au fur et à mesure du développement ce choix a été adopté.

Concernant les cartes d'alimentation et de la carte de contrôle principale, leur alimentation est faite par 2 cellules Li-Ion 18650 3000 mAh donnant un voltage nominal de 7.4 V. L'alimentation du RP2040 étant du 3.3 V et l'alimentation du Pi Zero 2 W étant du 5 V, deux LDO AMS 1117 de 5 V et 3.3 V, respectivement, sont nécessaire.

Les servos utilisés pour le système d'accrochage des drones utilisent un courant élevé de 300 mA à 1 A, donnant une consommation potentielle au minimum de 1,2 A ou 4 A au maximum, des LDO ne sont pas adaptés à cette utilisation. Un convertisseur DC-DC LM2596 pouvant convertir des hautes tensions à du 5 V pour des courants jusqu'à 3 A sont utilisé sur les cartes d'alimentation des servos. Ces cartes sont aussi munies de connectiques

pour placer une touche de clavier de type CHERRY MX afin de servir de détecteur de présence dans le système d'accroche.

Un emplacement pour un module radio 2.4GHz nrf24l01 est prévu, mais pas utilisé dans notre version du prototype pour des raisons de stabilité de connexion. Un ensemble de composants passifs est utilisé pour la filtration du courant ainsi que des connecteurs afin de faciliter le montage et l'utilisation.

4.1.3 Design et production des PCB via KICAD

Les deux types de PCB on été dessinés sur KICAD. Le "workflow" KICAD est fait en 2 étapes :

— La première étant de faire un schéma élèctrique de nôtre système puis de faire le design physique de la carte en tracant les traces du PCB. Ci-dessous un exemple de schéma élèctrique (carte de contrôle principale) :

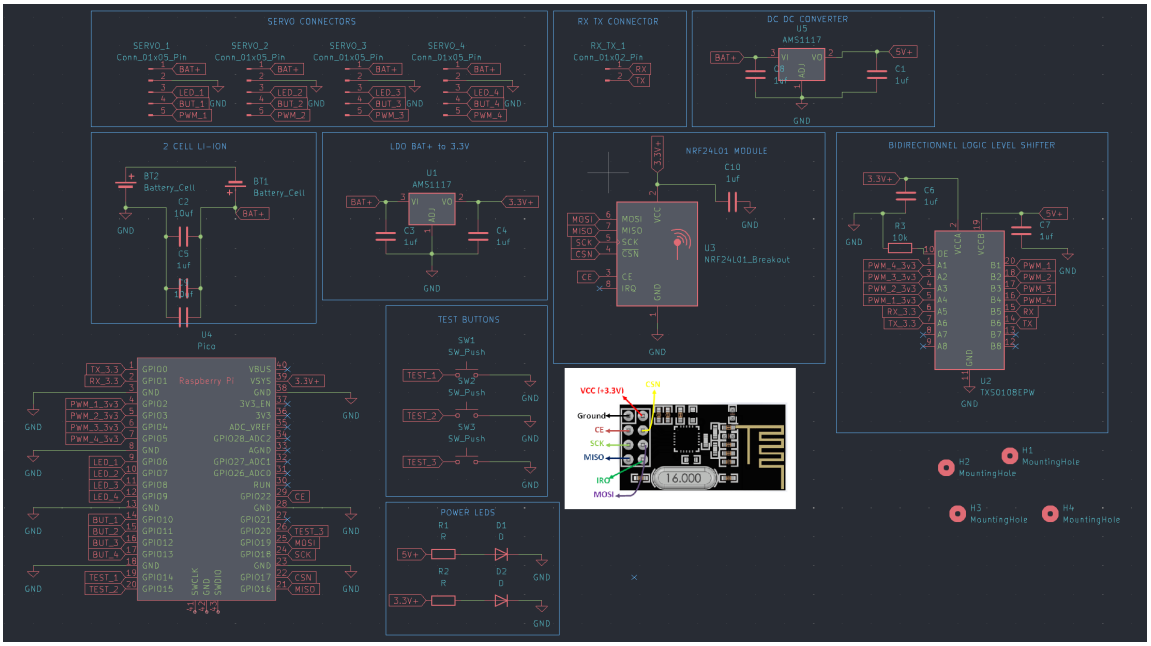

FIGURE 4.1 – Capture d'écran du logiciel KICAD du schéma de la carte de contrôle principale.

Voici un exemple de la fenetre de design de PCB de KICAD (carte de contrôle principale) :

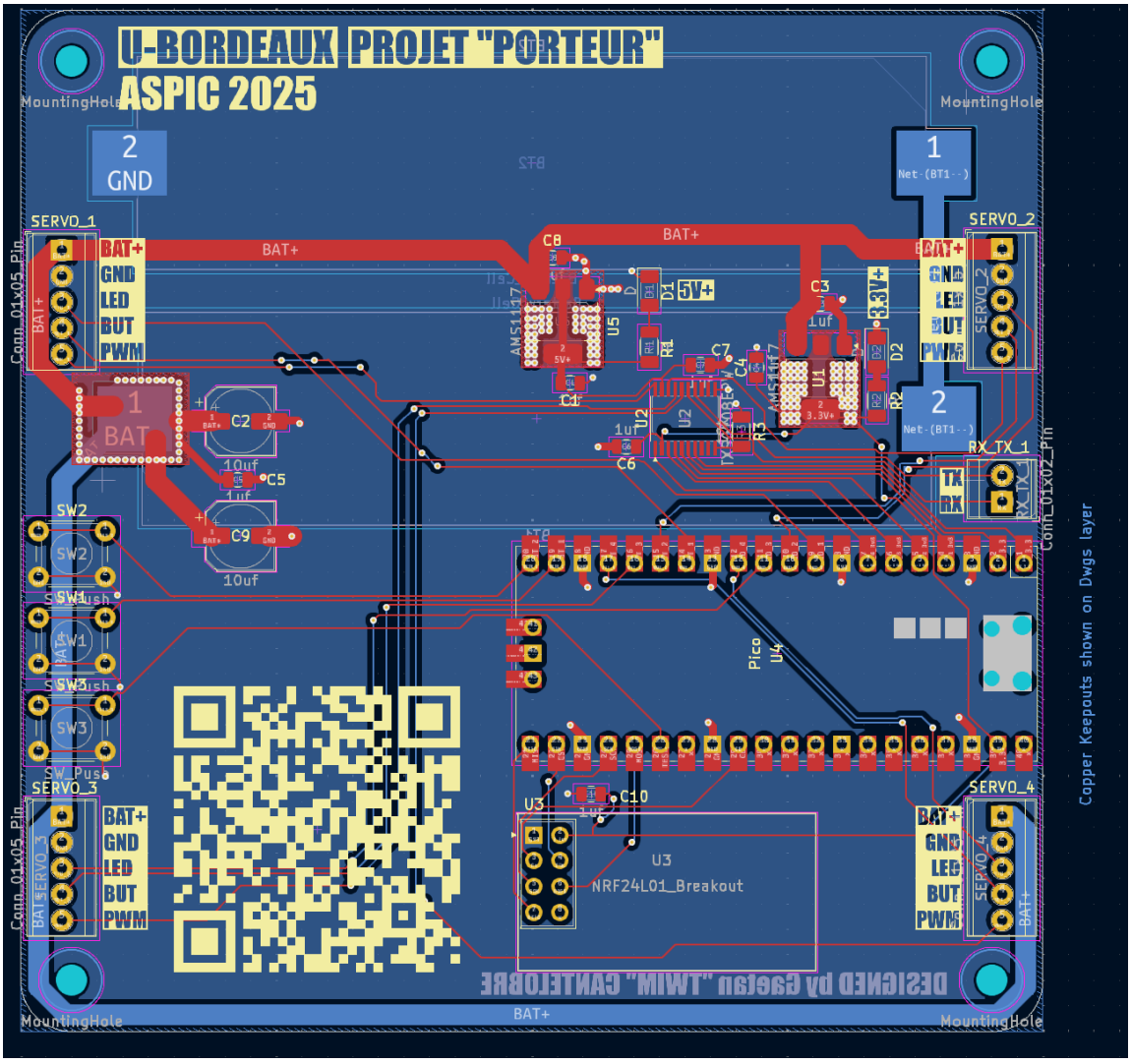

FIGURE 4.2 – Capture d'écran du logiciel KICAD lors du design de PCB de la carte de control principal.

Pour ce projets 2 cartes ont été dessiné sur KICAD la carte de contrôle et la deuxieme est la carte d'alimentation des modules d'accroche de la nacelle.

4.1.4 Production des PCB via JLBPCB et montage

Les PCB sont produits par JLBPCB, les composants sont soudés via des techniques de "reflow soldering" qui consiste à avoir l'ensemble du PCB qui est chauffé sur une courbe de

température afin de contrôler la fonte de l'étain et du flux. Les composants sont placés à leur place avant la fonte de l'étain et se repositionnent correctement par la fonte d'étain. Le PCB est ensuite refroidi en fonction d'une courbe afin d'éviter de soumettre l'étain, le PCB et le composant à des changements de température soudain. Les composants électroniques sont très sensibles aux changements de température, et un non-respect des courbes de températures indiqué par les producteurs peut engendrer des comportements imprévisibles. Le PCB est nettoyé et les composants THT ainsi que les connecteurs sont soudés. Voici des exemples de PCB après toutes les étapes de production :

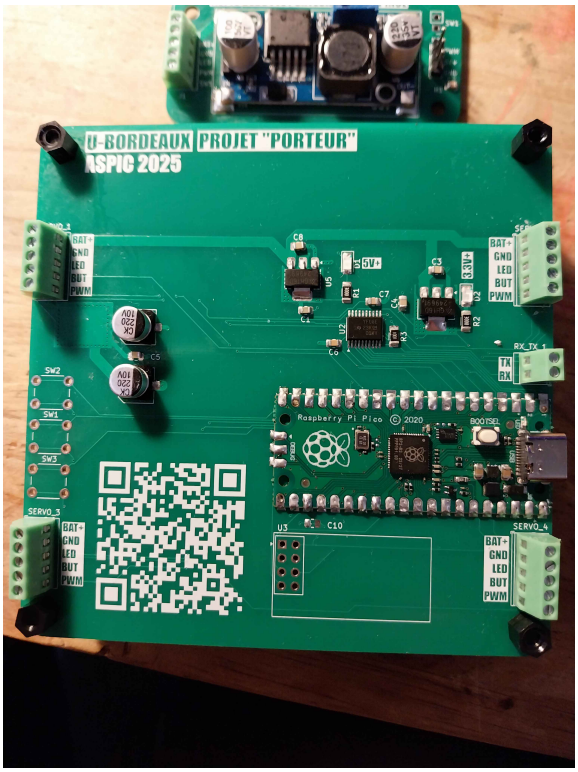

FIGURE 4.3 – Carte de contrôle principale prête à l'utilisation.

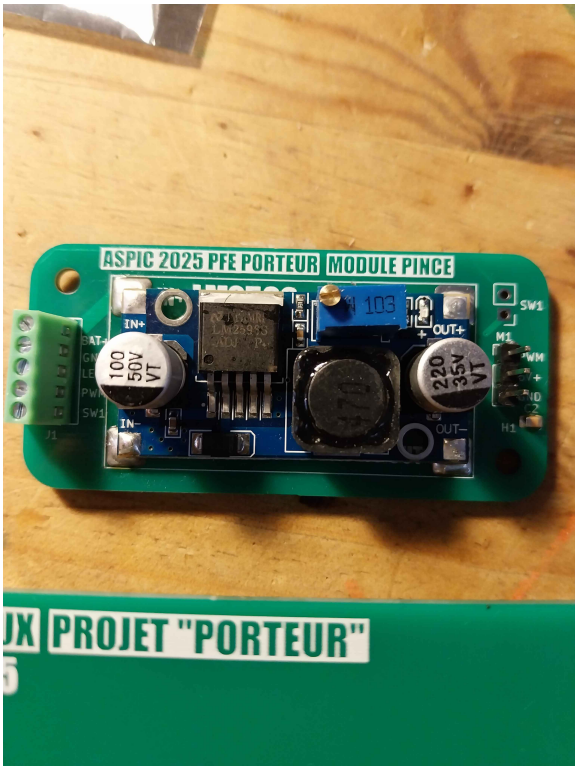

FIGURE 4.4 – Module "Pince" prêt à l'utilisation.

4.1.5 Considération et piste d'amélioration sur le design du PCB

La conception et la production de nos PCB ne font pas partie de notre parcours universitaire, ce qui signifie que leur qualité et leur design ne répondent pas aux standards industriels ni aux normes de sécurité. Cependant, notre équipe a su développer une solution fonctionnelle et durable, répondant efficacement à la problématique posée. La production et l'utilisation de ces PCB ont joué un rôle essentiel dans l'avancement et la pertinence du résultat fourni.

Certains efforts ont été fournis afin d'assurer la longévité de ces PCB, comme par exemple des zones de dissipation thermique pour les connecteurs de batteries et pour les LDO en utilisant les différentes couches de cuivre afin d'avoir une grande zone de cuivres. Sans ces zones, on augmente les chances d'endommager les composants.

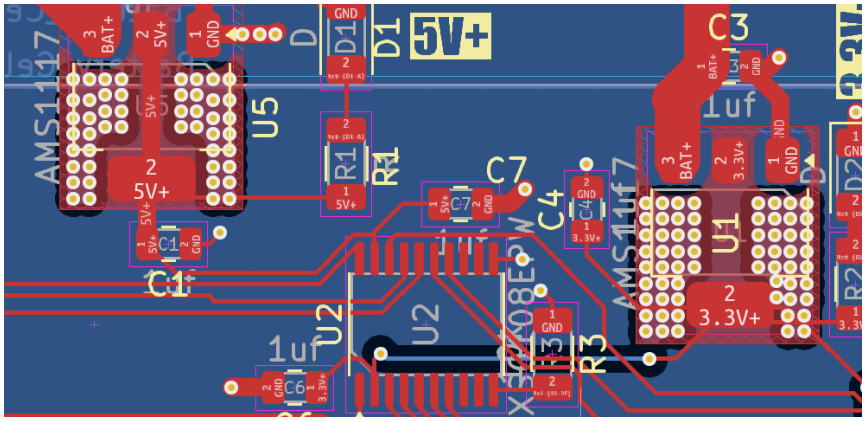

FIGURE 4.5 – Exemple de zone dissipation thermique pour LDO.

L'alimentation du Pi Zero W 2 était initialement prévue comme externe a la carte, dans notre cas nous avions un LDO 5 V 300 mA a disposition sur la carte car nous nous servons pas du convertisseur de niveau de logique, qui sert donc a alimenter le PI Zero W 2, son alimentation n'est pas correcte car 300 mA est la consommation max du Pi Zero W 2 donc un LDO avec des spécification plus élevées serait nécessaire.

Lors du design de la carte de contrôle, nous avons fait l'erreur de ne pas mettre un GND commun avec la carte Pi Zero 2 W, car sans elle, nous ne pouvons pas faire de communication seriale.

4.2 Modélisation CAO

4.2.1 Système d'accroche du drone

Une partie majeure de la problématique était de charger et de décharger des drones. La première approche pour faire ce montage était un système de pince pour venir pincer un support monté sur le drone. Cette version relève des problèmes de sécurité, car pour assurer la sécurité des drones, une source d'énergie doit être activée pour tenir le drone, et dans le cas d'un problème d'alimentation électrique une chute du drone et les dégâts associés sont un risque majeur. Pour éviter ces problèmes, nous avons développé un système de goupille qui s'insère dans une monture sur le drone et qui s'appuie sur la nacelle. Ce système assure que dans les 2 positions, ouverte et fermée, nous ne nécessitons pas de source d'énergie pour garder cette position. Assurer la sécurité du matériel embarqué a été rapidement identifié comme une priorité, car en utilisant du matériel prêté par l'université, nous voulons en assurer l'intégrité.

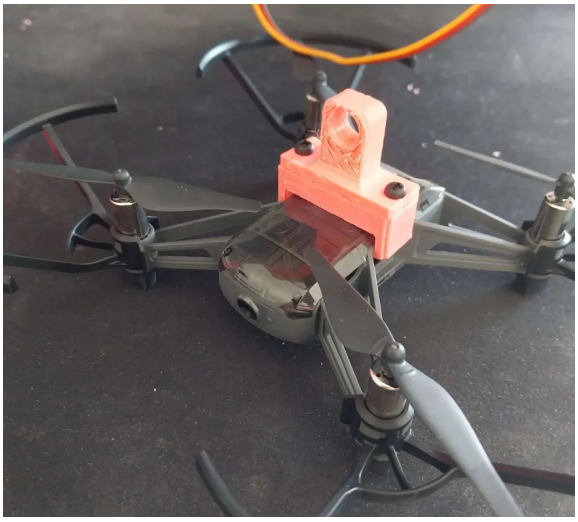

FIGURE 4.6 – Element de montage sur le Tello Edu.

L'ensemble des pièces pour la nacelle ont été dessinées sur Fusion360 et imprimées en 3D via des imprimantes FDM en PLA+.

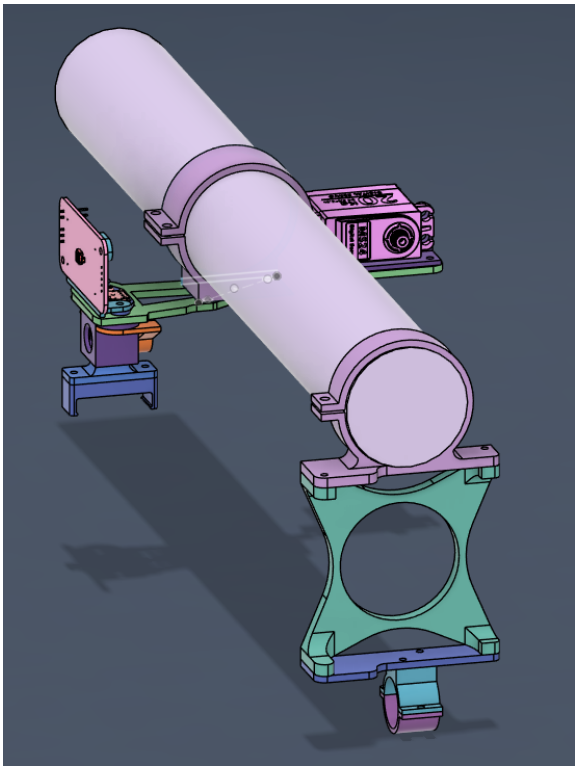

FIGURE 4.7 – Représentation 3D du support de la nacelle

Le système d'accroche est contrôlé par un servo "20kg" ayant un torque très haut, dans le cadre d'utilisation du Tello Edu, le servo moteur est largement assez performant, mais nous voulions avoir un système capable de porter des système embarqué (drone, charge utile) comme le drone qui devait être porté initialement qui était un DJI mini 3 (drastiquement plus lourd), alors ce choix a été judicieux.

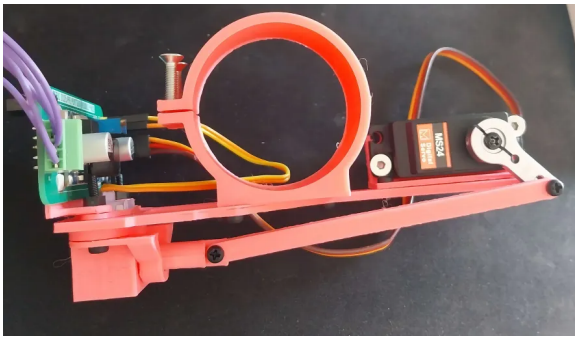

FIGURE 4.8 – Vue latérale du systeme de montage de drone

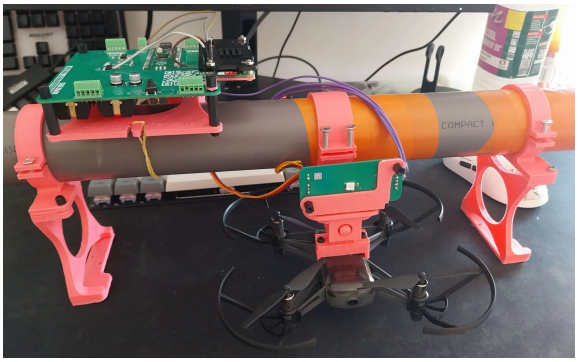

FIGURE 4.9 – Photo de la nacelle entière.

La structure principale de la nacelle est un tube de PVC de 50mm de diamètre. L'ensemble des éléments sont montés sur cette structure principale. La nacelle est composée de 3 composantes ; la carte de commande centrale, un (ou plusieurs) module porteurs et les éléments pour l'interfaçage du DJI Matrice 300.

4.3 Structure Informatique

4.3.1 Firmware embarqué sur RP2040

Dans un effort de facilité et de compatibilité, nous avons utilisé comme logiciel Arduino IDE, permettant de modifier et flasher le code sur la carte de contrôle. Cela permet d'éviter tout problème de compilation et de dépendances. Ce firmware comporte uniquement le strict minimum afin d'éviter d'avoir deux systèmes complexes qui s'interfacent, il est plus raisonnable d'avoir un seul système complexe. L'utilité du RP2040 dans ce projet est de créer des signaux PWM pour contrôler les servos, des modules d'attaches des drones, détecter la présence de drone via les switches cherry mx et contrôler les LEDs WS2812B pour donner du feedback visuel.

Dans notre git nous avons deux firmware différents.

— Un premier étant le firmware principal avec une fonctionnalité complète : contrôle de 4 drones possibles, ajustable et complétement modulable.

— Et un deuxième qui sert de "démo" dans lequel un scénario est possible. Cela est un effort pour éviter "l'effet demo" lors des tests avec le client. La fonctionnalité du code est réduite mais nous utilisons ce rendez-vous client comme des *showcase* et des *proof of concept*.

Ces scripts "démo" sont constitués de trois parties majeures. La première étant la partie configuration, dans laquelle nous mettons nos paramètres comme le nombre de drones, la connectique associée et les pins a utiliser.

```
[caption=Exemple configuration d mo]
    #define LED_PIN 6     // Pin for WS2812B LED
    #define NUM_LEDS 1    // Number of LEDs
    #define SERVO_PIN 2   // Pin for the servo
    #define ONBOARD_LED 25 // Onboard LED for Pi Pico
    #define SWITCH1 10
    #define OPEN_ANGLE 90
    #define CLOSE_ANGLE 5
    #define RESTART_BUT 14
    int loading_mode = 0;
```

La deuxième partie est la configuration initiale de la démo. Cette partie vise à mettre la nacelle dans une configuration particulière, généralement vide avec les modules d'accroche ouverts, ainsi qu'à ouvrir la communication serial avec le Pi Zero W 2 et configurer les GPIO correctement. Un protocole de détection d'erreur de configuration est systématiquement mis en place. Un seul module d'accroche est généralement mis en service pour les démos. Voici un exemple de configuration initiale :

```
void setup() {
loading_mode = 0;
closed = 0;
myServo.attach(SERVO_PIN);
pinMode(ONBOARD_LED, OUTPUT); // Set onboard LED as output
ledStrip.begin();
ledStrip.show(); // Initialize LED strip

set_servo_angle(OPEN_ANGLE); // Set initial servo position

Serial1.begin(115200); // Initialize UART (RX: GP1, TX: GP0 by default)
ledStrip.setPixelColor(0, ledStrip.Color(0, 255, 0));
ledStrip.show();
pinMode(SWITCH1, INPUT_PULLUP);
pinMode(RESTART_BUT, INPUT_PULLUP);

// Check if Serial1 starts properly, blink onboard LED if it fails
if (!Serial1) {
    while (true) {
        digitalWrite(ONBOARD_LED, HIGH);
        delay(500);
        digitalWrite(ONBOARD_LED, LOW);
        delay(500);
    }
}
ledStrip.setPixelColor(0, ledStrip.Color(255, 255, 255));
ledStrip.show(); }
```

Le comportement de la nacelle est guidé afin d'éviter des erreurs de pratique lors des démos. On attend des commandes spécifiques, et les actions ne sont possibles que dans un certain ordre. Une fois que la démo est faite et que toutes les parties de la démo sont sécurisées (drones et nacelles), la nacelle se "brick" et attends d'être flashé à nouveau ou redémarrée. Cela est une mesure de sécurité afin d'éviter toute action non voulue. Voici un exemple d'un scénario de démonstration :

```
 void loop() {
check_restart();
if(!closed && !digitalRead(SWITCH1) && loading_mode) {
  closed = 1;
  set_servo_angle(CLOSE_ANGLE);
  Serial1.write("loaded\n");
  ledStrip.setPixelColor(0, ledStrip.Color(255, 0, 0));
  ledStrip.show(); }

  if (Serial1.available()) {
    digitalWrite(ONBOARD_LED, HIGH); // Turn on onboard LED when receiving data
    String received = Serial1.readStringUntil('\n');
    received.trim(); // Remove extra spaces and newlines

    if(received == "load") {
      loading_mode = 1;
      Serial.println(" entered loading mode");
      ledStrip.setPixelColor(0, ledStrip.Color(0, 0, 255));
      ledStrip.show(); }

    if(received == "launch") {
      launch_procedure();
      while(1){
        check_restart();}}
    ledStrip.show();}
}
```

Nous pouvons observer qu'une fois la fonction *launch_procedure()* éxecutée (une fonction permettant de lâcher le drone qui est prêt à voler) est terminée, nous ne pouvons plus communiquer avec la nacelle car la seule action possible est de redémarrer la carte pour recommencer le scénario.

4.3.2 Script démo Pi Zero 2 W

En parallèle de ce script dédié à la démonstration du drone, nous avons développé une structure de code symétrique par laquelle on interagit avec le contrôleur de la nacelle par l'intermédiaire de ce code de démonstration. Dans ce script Python, nous utilisons l'API DJITELLOPY qui permet de contrôler les drones Tello Edu. Ce script se limite à la connexion d'un seul drone et se présente comme une interface CMD classique. Cependant, il n'est possible de donner qu'un seul ordre spécifique de commandes dans l'optique d'assurer un déroulement favorable des démonstrations. Ce script se limite au chargement des paramètres de contrôle d'un fichier JSON, la connexion d'un drone, le chargement d'un

seul module porteur, recevoir le message serial de confirmation et la coordination du lâcher du drone et son décollage. Nous avons ajouté un paramètre optionnel de lancement qui fait faire des "flip" au drone en avant et en arrière afin d'ajouter un peu plus d'engagement de la part des spectateurs.

Voici un example d'un de ces scripts de démonstration :

```
        from djitellopy import Tello
        import time
        import serial
        import json

        with open("demo_settings.json", "r") as file:
            config = json.load(file)

        # Assign values to variables
        DRONE_IP = config["drone_ip"]
        ADD_FLIP = config["add_flip"]

        ser = serial.Serial('/dev/serial0', 115200)  # Default UART on Pi is /dev/serial0 (or /dev/ttyAMA0)
        tello = Tello(host=DRONE_IP)
        tello.connect()
        time.sleep(1)

        answer = ""
        while(answer != "load"):
            answer = input("Select next order ? \n")

        print("You can load the drone now.\n")
        loaded = False
        # Connect to the drone
        ser.write("load\n".encode())

        while(loaded is False):
           msg = ser.readline().decode().strip()
           print(msg) if msg is not None else None
           loaded = msg.strip() == "loaded"

        print("Drone is loaded.")

        answer = ""
        while(answer != "launch"):
            answer = input("When ready for launch : \"launch\" ")

        print("takeoff")
        tello.initiate_throw_takeoff()
        time.sleep(1)
        ser.write("launch\n".encode())
```

```
        time.sleep(3)

        # Start flight
        tello.move_forward(30)
        if(ADD_FLIP):
            tello.flip_forward()
            tello.flip_back()


        ser.close()

        tello.land()
        # Disconnect
        tello.end()
```

Ces deux types de script de démo nous permettent d'assurer une exécution contrôlée sans surprise lors des démonstrations avec le client.

4.3.3 Classe Python sur Pi Zero 2 W

Une structure de code plus complexe est utilisée dans le cadre d'une utilisation libre des fonctionnalités complète du système. Le mode d'utilisation est le même que dans les scripts de démonstration, c'est-à-dire une entrée de type terminal de commande. Une classe python est utilisée pour encapsuler l'ensemble des actions du drone et de la nacelle. Chaque action possible de la nacelle est contenue dans un dictionnaire dans lequel on trouve le nom de la commande à entrer dans le STDIN, une fonction associée, le nombre de paramètres attendus, et un message d'usage. Cela permet à n'importe quel utilisateur de contrôler la nacelle sans formation en lisant la description des actions et leurs usages. Voici une liste des actions contenues dans la classe :

```
        self.actions = {
                "status": [self.get_nacelle_status,0,"Asks and waits for a status update frome the nacelle."],

                "reset": [self.reset_nacelle,0,"Reset the pi pico on the nacelle, closing all grabbing modules."],

                "load" : [self.load_nacelle,1,"Loads grabber modules. Takes the target grabber id as a parameter or \"ALL\" to load all, waits for confirmation and a list of the load grabber modules"],

                "launch" : [self.launch, 1,"Unloads grabber module. Takes the target grabber id or \"ALL\""],

                "force_open" : [self.force_open, 1,"Opens all grabbing modules"],
```

```
11
12              "force_close" : [self.force_close, 1,"Closes all grabbing modules"],
13
14              "help" : [self.help,0,"Prints a list of commands and their parameters"]
15          }
```

Cette structure de code permet d'ajouter facilement de nouvelles actions, comme des scénarios ou des commandes spécifiques en suivant le modèle des autres actions. Cependant, il faut ajouter un code symétrique sur la carte de contrôle de la nacelle afin de définir les messages ACK et les comportements qui sont associés.

Chaque action est en attente d'un message ACK afin de pouvoir contribuer au système d'estimation d'état de la nacelle. Pour simplifier les choses, la nacelle attend toujours de recevoir des messages de la part du Pi Zero. Suite aux actions que le Pi Zero fait, il attend une réponse serial. Ces messages ont pour objectif d'informer l'ordinateur embarqué sur l'état d'avancement des actions comme par exemple, indiquer que la nacelle a fini d'être chargée, et indiquer quelles parties de la nacelle sont chargées.

Cet effort de développement de code peut permettre une facilité de poursuite de développement d'un projet.

4.3.4 Code "Legacy"

Sur notre Git nous avons du code faiblement développé, car ce sont des snippets de code de test pour certaines technologies que nous n'avons pas utilisé. Dans le nôtre cas l'abandon des modules de communication radio NRF24L01 nous a forcé a renouveler une nouvelle base de code.

4.4 Intégration

4.5 Développement drone

Pendant ce projet, nous devions automatiser certaines tâches sur un drone afin qu'il puisse décoller après avoir été lâché en vol, exécuter une mission spécifique, puis atterrir au sol. Pour cela, nous avions plusieurs drones à notre disposition. Nous allons détailler l'approche adoptée pour chacun d'eux.

4.5.1 DJI Mini3

Au début, nous avions prévu d'utiliser le drone Mini 3 de DJI. L'objectif principal était de fixer un ou plusieurs Mini 3 sur le drone porteur, puis d'automatiser certaines tâches.

Pour ce faire, nous devions développer une application Android spécifique au Mini 3. Cependant, le manque de documentation, la complexité du SDK, ainsi que l'absence de

travaux de référence sur ce dernier nous ont contraints à abandonner cette approche.

Malgré ces difficultés, nous avons réussi à accomplir certaines avancées, notamment :

— La récupération du flux vidéo du drone.

— L'acquisition de certains états vitaux.

— L'automatisation du décollage et de l'atterrissage.

Nous avons également commencé à développer des joysticks virtuels pour contrôler les axes *PITCH*, *ROLL* et *YAW*. Toutefois, seule la partie graphique a été réalisée, car la mise en œuvre de la logique nécessitait un module séparé que nous n'avons pas réussi à importer et à faire fonctionner correctement. Ci-dessous des images représentant le mini3 ainsi que son application.

FIGURE 4.10 – Communication entre le Mini3 et l'application Android développée

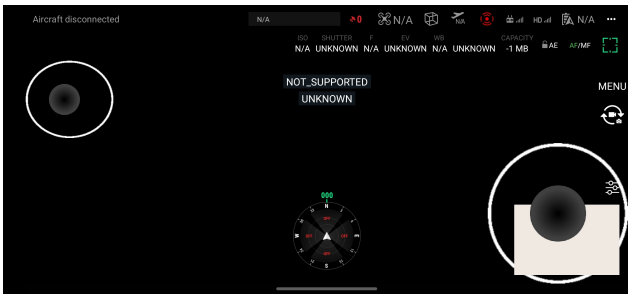

FIGURE 4.11 – Interface de l'application implémentant les joysicks

4.5.2 Tello Edu

Au cours du projet, nous avons principalement fait usage du drone *DJI Tello Edu*, qui a été utilisé comme drone largué par le *Matrice300*. Au départ, il nous était demandé de faire fonctionner le projet avec plusieurs drones, dans l'idée 4 drones largués par le Matrice300. Nous avons étudié les différentes options en faisant des recherches sur des essaims qui ont été faits avec les mêmes drones, notamment ce post sur le forum *TelloPilots*, ou encore cette API existante pour faire du swarming de Tello. Cependant, pour faire des avancées

significatives, et pouvoir rapidement produire un résultat visualisable, nous nous sommes concentrés sur l'utilisation d'un seul drone, et ce jusqu'à la fin du projet.

L'implémentation des commandes de contrôle du Tello Edu a été faite grâce à l'API suivante, dans laquelle nous pouvons avoir plusieurs commandes à donner au drone, notamment décoller, atterrir, accéder à la vidéo du drone et autre. Nous aborderons plus précisément les fonctions utilisées dans l'API plus bas après avoir détaillé les différentes préparations du drone.

```
1   with open("demo_settings.json", "r") as file: // Configuration des
    settings utilises pour la suite de la demonstration
2   config = json.load(file)
3
4   # Assign values to variables
5   DRONE_IP = config["drone_ip"]
6   ADD_FLIP = config["add_flip"]
7
8   //Connection a la nacelle via port serial
9   ser = serial.Serial('/dev/serial0', 115200)  # Default UART on Pi is /
    dev/serial0 (or /dev/ttyAMA0)
10  tello = Tello(host=DRONE_IP)
11  tello.connect()
12  time.sleep(1)
```

Ci-dessus, nous avons la partie configuration du drone et de son environnement avant de lui donner des commandes. On utilise ici un fichier JSON pour faciliter l'utilisation de paramètres et les avoir accessibles sur d'autres scripts. Après avoir initialisé les différents objets que l'on utilise, on établit une connexion entre le drone et la nacelle et nous pouvons maintenant commencer à lui donner des commandes.

```
1   // Wi-Fi SSID and Password
2   WIFI_SSID = "Projet_porteur"
3   WIFI_PASSWORD = "aspic2025"
4
5   // Connect to the drone
6   tello = Tello()
7   tello.connect()
8
9   // Send the STA mode command
10  tello.connect_to_wifi(WIFI_SSID, WIFI_PASSWORD)
11
12  print("Tello is switching to Wi-Fi network:", WIFI_SSID)
```

Pour faciliter le travail sur le drone, nous avons fait principalement deux scripts qui permettent de configurer le drone, son réseau, et le trouver sur le réseau. Ici, nous avons

un premier script qui permet de changer le réseau du Tello Edu. Le drone par défaut se configure lui-même comme un *"access point"* et créer un réseau sur lequel nous pouvons nous connecter pour contrôler le drone. Seulement, dans notre cas, nous voulons que le drone se connecte lui sur un réseau à part, car nous voulons pouvoir communiquer non seulement avec la nacelle, mais aussi pour prévoir potentiellement plusieurs drones qui pourraient tous se connecter à un même réseau pour faire un essaim. Nous pouvons voir qu'une fois connecté au drone, nous lui donnons la commande *connect_to_wifi* pour qu'il se connecte au réseau Wi-fi de la nacelle.

```
    def scan_network():
    nm = nmap.PortScanner()

    # Get the local network IP range
    hostname = socket.gethostbyname(socket.gethostname())
    network_prefix = ".".join(hostname.split(".")[:-1]) + ".0/24"

    print(f"Scanning network: {network_prefix}...")

    # Scan for devices with port 8889 open (Tello default port)
    nm.scan(hosts=network_prefix, arguments='-p 8889 --open')

    tello_ips = [host for host in nm.all_hosts() if '8889/tcp' in nm[host
    ]['tcp']]

    if tello_ips:
        print(f" Tello EDU Drone Found! IP: {tello_ips[0]}")
        return tello_ips[0]
    else:
        print(" No Tello EDU Drone found on the network.")
        return None

# Run the scanner
tello_ip = scan_network()
```

Un deuxième script ici permet de faire un scan du réseau Wi-fi pour trouver la ou les adresses IP des potentiels drones Tello Edu connectés sur le réseau. Grâce aux documentations sur le drone, nous avons pu constater que le port ouvert par défaut sur les Tello Edu est le port *"8889"*, il nous suffit alors de regarder ce port-là pour avoir les différentes adresses IP de Tello Edu connectés au réseau.

Maintenant, que toute la configuration du drone est faite, nous pouvons pour la démonstration donner des commandes à notre drone. Comme montré dans la démonstration donnée au début du rapport, le drone commence avec la commande *"initiate_throw_takeoff()"* après que le largage a été effectué. Le drone commence ensuite à faire du vol stationnaire, et nous lui donnons ensuite une commande pour avancer de 30 centimètres, et finalement

de faire un flip avant puis un flip arrière avant de se poser au sol.

Concernant certains soucis avec le drone, nous avons par exemple des décollages irréguliers quand le Tello Edu ne décolle pas sur une surface parfaitement plate ; si ce n'est pas le cas, il décollera en dérivant sur un côté au lieu de décoller droit verticalement, les secousses présentes notamment quand il est accroché au Matrice300 feront le même effet. Un problème supplémentaire remarqué est que si le drone reçoit une erreur au cours de son vol, par exemple le décollage qui n'est pas droit, ou autre, il se mettra à faire un long vol stationnaire avant d'atterrir sans traiter la suite du script.

4.5.3 Matrice300

Pour une première approche, nous avions envisagé de développer un logiciel embarqué sur le Matrice 300 afin d'envoyer des signaux et de le contrôler à distance. Cependant, nous avons rapidement constaté que cette approche nécessitait l'achat de matériel supplémentaire, notamment le Manifold 2, indispensable pour embarquer un logiciel personnalisé sur le drone.

En raison du coût élevé de ce matériel et de la non trivialité de cette solution, nous avons décidé d'opter pour une approche plus simple. Nous avons donc contrôlé le drone manuellement et fixé les contrôleurs directement dessus.

4.6 Tests

4.6.1 Tests DJI Mini3

L'avancement sur le Mini 3 a été limité, mais nous avons tout de même pu réaliser quelques tests. Nous avons réussi à récupérer le flux vidéo du drone ainsi qu'à automatiser son décollage et son atterrissage.

Nos tests et tentatives d'amélioration sur le Mini 3 ont été interrompus en raison de son abandon au profit du Tello Edu.

Nous avions également commencé à développer certains éléments pour le Mini 3, notamment un PIN de fixation permettant de l'attacher au Matrice 300, que vous pouvez voir sur la figure ci-dessous :

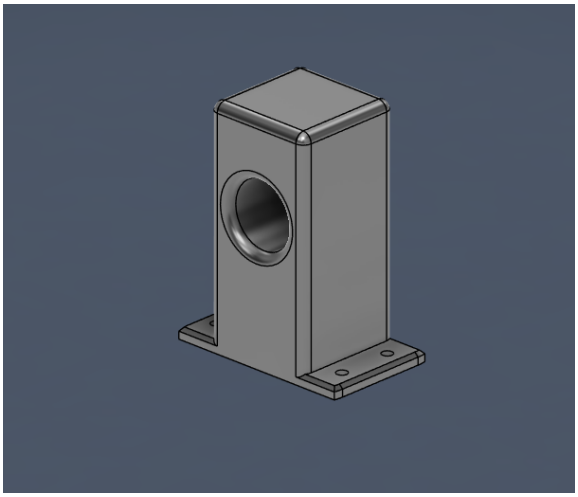

FIGURE 4.12 – PIN de fixation de Mini 3 sur la nacelle

4.6.2 Tests Tello Edu

Pour le tello Edu nous avons vu dans l'application mobile et le SDK de DJI que le Tello Edu est capable de faire des décollages par lancement à la main, nous avons donc décidé de faire des tests pour voir si le Tello Edu pouvait décoller en étant lâché à la vertical depuis une position fixe, car cette capacité est très importante pour faire décoller le drone dans les airs. D'après nos tests, nous avons trouvé qu'il était possible de faire cela, mais qu'il fallait quand même optimiser quelques paramètres comme la hauteur de lancement (qui doit être d'environ 1m40) mais aussi la charge de la batterie qui doit être assez haute pour que le drone puisse décoller. Nous avons aussi fait des tests pour voir si le Tello Edu pouvait atterrir en toute sécurité après avoir été lâché en vol, et nous avons trouvé que cela était possible, mais qu'il fallait être vigilant à la dérive des mouvements du drone qui pouvait alors le faire bouger dangereusement. Nous avons fait une démo où le drone décolle depuis la nacelle et atterrit en toute sécurité, vous pouvez voir la vidéo ici.

4.6.3 Tests Matrice300

Concernant le *Matrice300*, nous avons réalisé plusieurs tests afin de vérifier la *stabilité du drone en vol*. Ce modèle est équipé de *capteurs infrarouges (IR)* situés sous l'appareil, qui lui permettent de détecter le sol et de maintenir une altitude constante. Pour fixer notre nacelle sur le Matrice300, nous avons essayé d'obstruer ses capteurs IR à l'aide de carton, afin de déterminer si le drone pouvait toujours voler normalement. Vous pouvez voir cette manipulation sur la figure ci-dessous :

FIGURE 4.13 – Capteurs bloqués par un morceau de carton

Nous avons constaté que le Matrice300 pouvait voler sans problème malgré l'obstruction des capteurs IR. Cependant, l'atterrissage était plus difficile, car les calculs d'altitude étaient faussés par l'absence de détection du sol.

Suite à ces essais, nous avons testé le décollage du Matrice300 avec le système de nacelle attaché, vous pouvez voir le résultat ici.

5 Organisation de Projet

5.1 Github

Pour la gestion des différentes versions de notre projet, nous avons logiquement opté pour *Github*, la plateforme de versioning la plus populaire et facile d'utilisation. Grâce à Github avons pu maintenir une cohésion des différentes versions du code de chacun, et pouvoir ensuite faire l'ajout de plusieurs travaux différents sans problèmes.
En ce qui concerne l'architecture de projet que nous avons mise en place consiste en trois points principaux :

— La branche **main**, qui est notre branche dite de *production*, la version finale et fonctionnelle de chaque ajout qui y est fait.
— La branche **main_WIP**, pour *work in progress*, qui est un jumeau de la branche finale main, mais où nous n'avons pas encore testé la robustesse des dernier ajouts qui ont été fait, et que nous décidons de ne pas encore passer sur la branche main pour ne pas mettre en péril son fonctionnement.
— Les branches pour chaque **tâche**, correspondent aux branches créées quand un membre commence à travailler sur une tâche en particulier, comme par exemple pour lors du travail sur le contrôle du Tello Edu, nous avons créé la branche *Tello_Controller*, ou encore *M3_Controller* pour travailler sur la manette du Mini3.

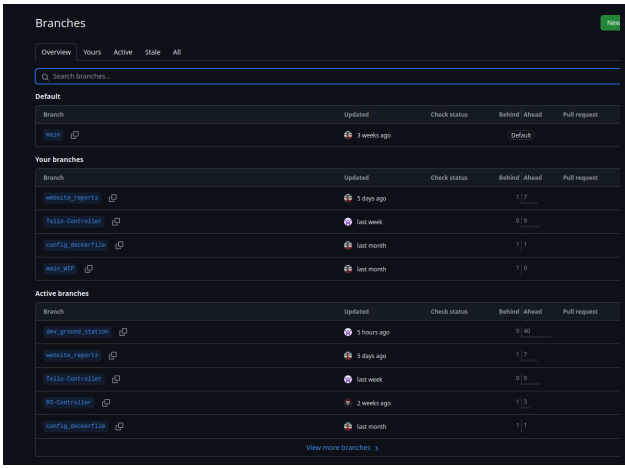

FIGURE 5.1 – Branches Git

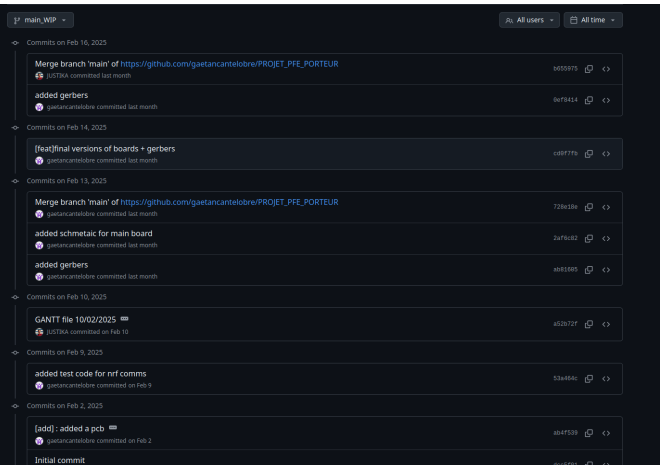

FIGURE 5.2 – Commits Git

L'organisation se fait ensuite comme suit, on créer d'abord notre branche correspondant à une tâche, par exemple *Tello_Controller*. Une fois que nous avons terminé le travail sur cette branche, on peut faire un premier merge sur la branche *main_WIP*, et vérifier les premiers potentiels problèmes de merge. Une fois fait, on peut maintenant faire plusieurs

tests pour vérifier l'intégrité du mélange des deux codes et être certains que ça n'a pas créé de nouveaux bugs, et si tout est fonctionnel, faire le merge de ces changements à la branche de production *main*.

5.2 Trello

Dans le cadre de l'UE **MOCPI**, il nous a été demandé d'utiliser la plateforme *Trello*, qui permet de mettre en place un planning de tâche type Kanban. Avec Trello, nous avons pu créer plusieurs *user stories* qui nous ont ensuite permises de faire les tâches à suivre au cours du projet. Sur la plateforme, nous pouvons ajouter des commentaires aux tâches pour tenir au courant de l'avancée, assigner plusieurs personnes aux tâches, les classer par catégories, ou encore donner une estimation de temps que la tâche prendra à être faites. L'organisation de notre Trello est comme suis :

1. User Stories
2. All Tasks
3. Weekly Tasks
4. Doing
5. Code Review
6. Testing
7. Done

Dans l'ordre, nous avons créé les user stories pour pouvoir ensuite organiser le projet. Après ça, nous avons créé toutes les différentes *tasks* faites en fonction des user stories. Nous nous sommes enfin partagé les tasks à faire au fil de la semaine.
On classe ensuite les tâches qui sont en cours, celles qui sont finies et qui doivent être vérifiée par un enseignant vont dans *Code Review*, et dans *Testing* si l'implémentation est finie, mais qu'on doit maintenant faire des essais. Et enfin dans *Done* quand la tâche est complétement terminée.

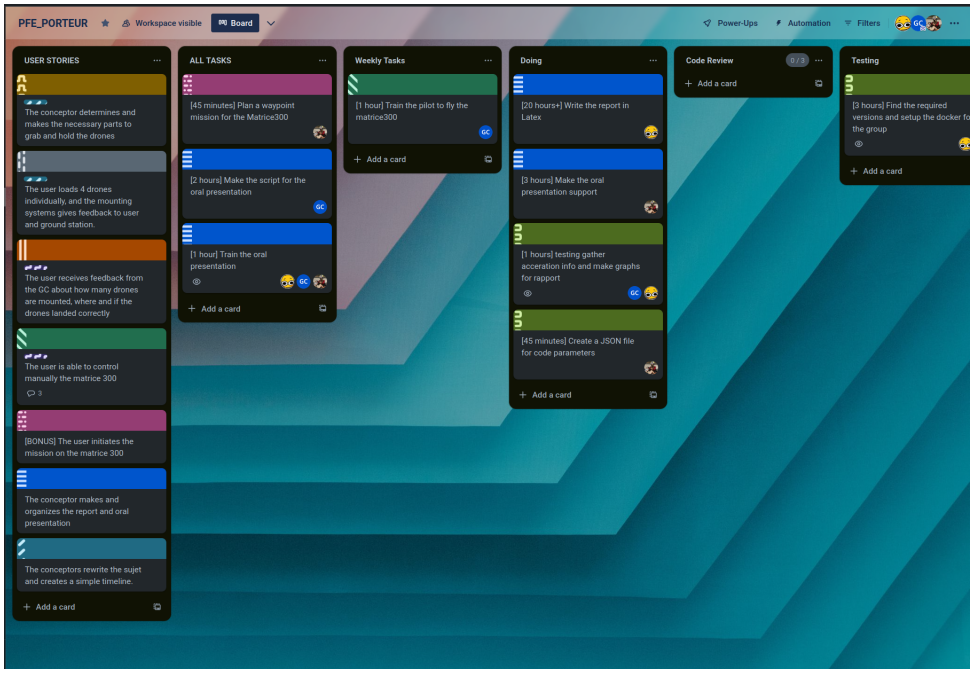

FIGURE 5.3 – Capture d'écran du Trello

Ci-dessus on peut voir l'ensemble de l'organisation du Trello de notre groupe comportant les différents groupes de tâches.

5.3 Gantt

Au cours du projet, le client pouvait nous demander certaines fonctionnalités, organiser des meetings, ou encore exiger certains rendus. L'un d'eux est la création d'un planning type *Gantt* pour superviser l'organisation et la répartition du travail sur court terme. Un planning Gantt nous a permis de planifier certains objectifs importants et apporter une notion de temps à quand ces fonctionnalités seront présentes, sous forme de frise chronologique. Cependant, il est important d'actualiser régulièrement ce planning, pour cause les différents soucis que nous avons pu rencontrer, ou encore avoir mis plus de temps que prévu pour certaines tâches. Ce planning était actualisé toutes les semaines en fonction de notre avancement. Les changements sur notre planning Gantt n'étaient pas nombreux, nous nous sommes plutôt bien tenu à l'organisation faite au début du projet, les changements fait sont majoritairement de légers retards par rapport aux prédictions faites.

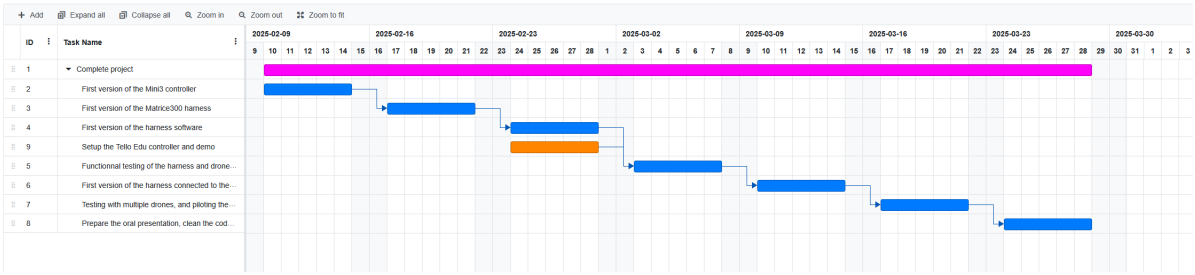

FIGURE 5.4 – Screenshot du planning Gantt

5.4 RocketChat

Pour la communication officielle avec notre client, nous avons utilisé *RocketChat*, car elle offre un serveur qui est hébergé par l'université et une application open source. Sur cette plateforme, nous avons pu partager souvent quotidiennement les différentes avancées qui ont été faites, ou les tâches prévues ainsi que les démonstrations faites. Grâce à ce canal de communication, le client pouvait lui aussi organiser des réunions ou des meetings pour discuter de certaines fonctionnalités qu'il voulait dans le projet.

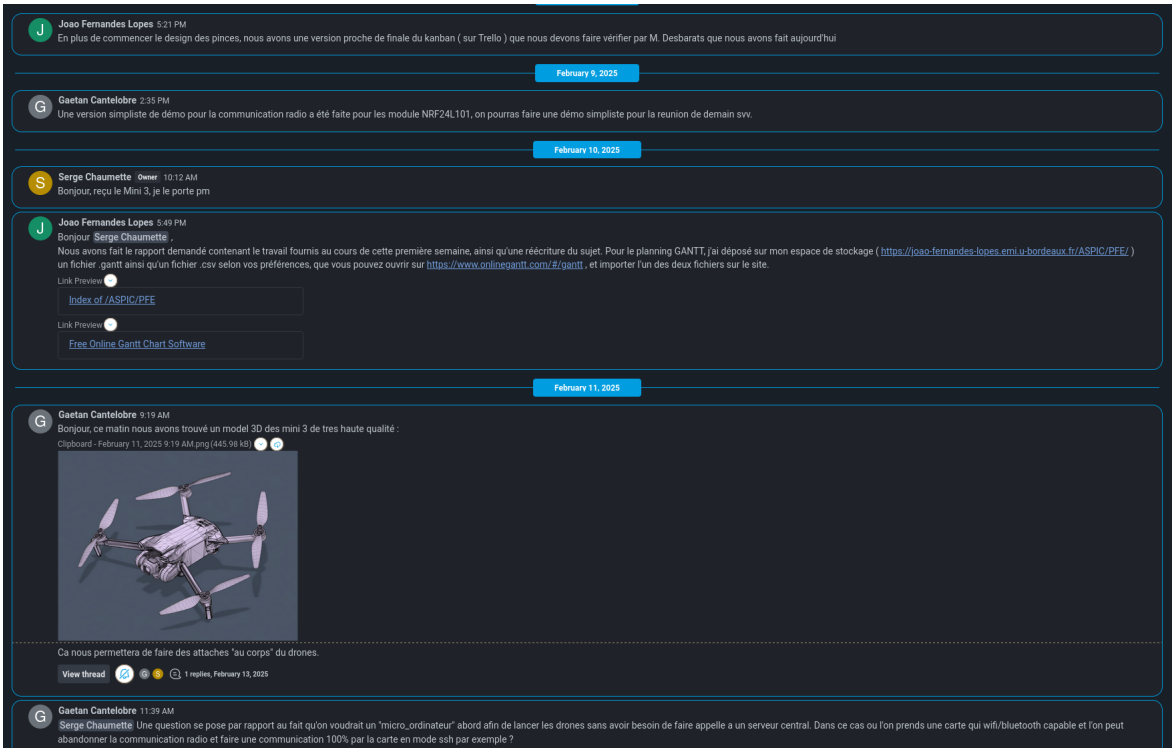

FIGURE 5.5 – Capture d'écran du RocketChat

5.5 Discord

Pour la communication au sein des membres du groupe, nous avons opté pour une plateforme avec laquelle nous sommes familiers depuis plusieurs années, *Discord*. Grâce à Discord, nous avons une interface de communication simple d'utilisation, où nous pouvons créer des salons à volonté, organiser des discussions vocales au cours desquelles nous pouvons partager nos écrans et notre caméra, ce qui permet une avancée simple et fluide de l'avancement de chacun. Sur Discord, nous avons pu organiser des meetings pour discuter des objectifs fixés, nous aider quand nous ne pouvions pas avancer, ou simplement partager des informations comme des dates de rendus, des documents, des photos, etc.

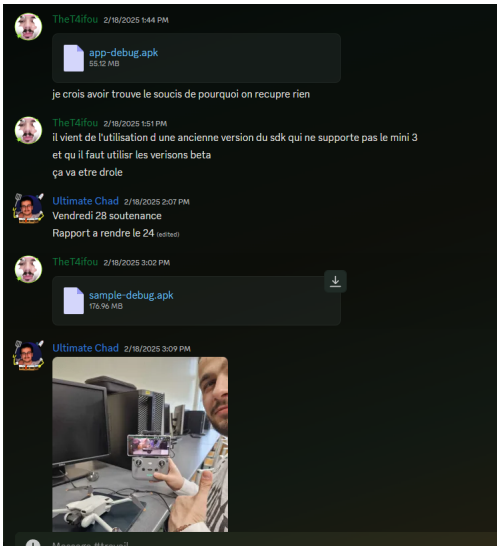

FIGURE 5.6 – Capture d'écran d'un salon Discord

5.6 Organisation annexe

5.6.1 Rapports hebdomadaires

En plus des choix d'organisation présentés précédemment, il nous a aussi été demandé de fournir des rapports hebdomadaires au cours du projet. Des comptes-rendus fait toutes les semaines, dans lequel nous présentons l'avancée faites au fil de la semaine au niveau implémentation, organisation, électronique et autre.

Rapport semaine 2 PFE "Porteur"

CANTELOBRE Gaetan, AABO ALJALOO Taif , FERNANDES LOPES João

PARTIE ÉLECTRONIQUE :

Communication radio 2.4 GHZ avec module nrf24l01 entre 2 pi pico avec le servo moteur 20kg. Afin de simuler la nacelle et la ground station.

Suite à la réunion PFE de la première semaine, il est donc convenu d'utiliser un MCU de type Pi Zero W afin d'avoir un système capable de contrôler la nacelle et de contrôler les drones lâchés.

PARTIE PCB :

Dans l'optique d'un produit plus concret et fini, nous avançons sur le design de PCB. Chaque module pour porter un drone doit pouvoir alimenter un servo 20kg (convertisseur DC DC type LM2596), ainsi qu'un avoir la connectique nécessaire pour le capteur de présence et le retour visuel (LED WS2812B). Utilisant des connecteurs PH 2.0 mm. Ci joint : Rendu 3D du PCB + schéma électrique.

FIGURE 5.7 – Capture d'écran du rapport n°2

5.6.2 Site Web

En plus des rapports hebdomadaires, nous avons aussi mis en place une application web sur laquelle est consultable plusieurs éléments importants de notre projet, comme par exemple les différents sites utile pour accéder au projet, le Github, le Trello, ou encore le site à utiliser pour ouvrir les fichiers Gantt. Sur le site, sont aussi présents les différents rapports hebdomadaires accompagnés du planning Gantt mis à jour régulièrement. Et puis nous avons aussi différentes versions du présent rapport, ainsi qu'une vidéo de démonstration de notre projet.

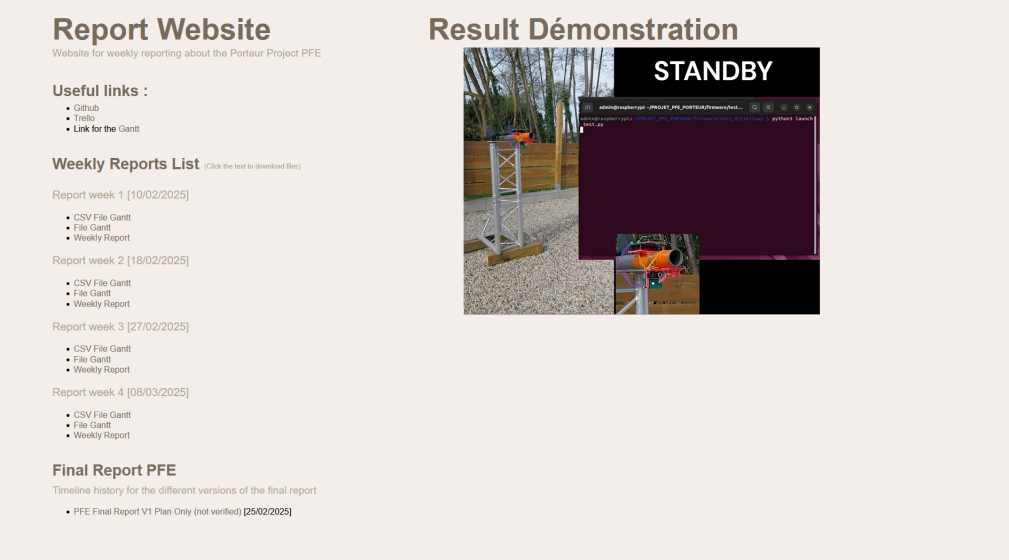

FIGURE 5.8 – Capture d'écran de notre site web

6 Problèmes rencontrés

Lors de la réalisation de ce projet, nous avons rencontré plusieurs problèmes. Nous allons les détailler dans cette section.

6.1 Objectifs non atteints

- L'utilisation du DJI Mini 3 comme drone secondaire fixé sur le drone porteur n'a pas été possible. En effet, le SDK du *Mini 3* s'est révélé peu adapté aux développeurs, ce qui nous a contraints à changer d'approche. Nous avons finalement opté pour un autre drone, le DJI Tello.

6.2 Problèmes Techniques Rencontrés

- L'initialisation du SDK du *Mini 3* était particulièrement complexe, ce qui nous a fait perdre beaucoup de temps.
- L'ajout de nouvelles fonctionnalités à l'application Android du *Mini 3* s'est avéré difficile en raison du manque de documentation.
- Peu de travaux réalisés par d'autres développeurs sur le *Mini 3* SDK étaient disponibles, ce qui a limité nos ressources et solutions possibles.
- Les module radio NRF24L01 incompatible avec l'environnement a haute EMF : les modules radios sont extrêmement sensible au bruit électromagnétique et nécessite un architecture d'alimentation "propre" qui n'a pas pu être atteint avec les PCB fait par l'équipe. De plus, l'environnement pourrait contenir en théorie plusieurs drones, et plusieurs servos moteur donc la liaison radio n'est pas garantie.
- Redesign multiple du système d'arroche des drones sur la nacelle : la première version du design de goupille pouvait se bloquer sur les parois du module au lieu de se placer correctement. La deuxième version à solution ce problème avec une cinématique plus simple des pièces dédie au guidage de la goupille.
- Commentaire sur la bilbioteque DJITELLOPY : Le moyen de fonctionnement du drone Tello Edu avec la bibliothèque DJITELLOPY ne permet pas un contrôle totalement réactif du drone, car son fonctionnement n'est pas fiable, des erreurs et un manque de réactivité sont souvent obtenues à cause des faiblesses de stabilité de réseau. De plus, chaque action demandée à l'API nécessite une stabilisation de plusieurs secondes entre chaque action du drone sous peine de ne pas pouvoir contrôler précisément les actions du drone, dû au manque d'un GPS RTK qui rend la navigation d'autant moins stable et moins précise.

7 Bilan Et Perspectives

7.1 Travail Futur

Il existe plusieurs axes d'amélioration pour ce projet.

— Fusion avec d'autres projets : lors d'un dernier meeting avec notre client et un autre groupe (projet Alvéoles), nous avons discuté de la possibilité de fusionner nos deux projets. En effet, notre projet permet de faire décoller un ou plusieurs drones depuis le Matrice 300 RTK, ainsi que de faire quelques actions, on peut donc imaginer un scénario ou après l'exécution de nos actions, on peut passer la main de contrôle au projet Alvéoles pour faire atterrir et ranger le drone dans un endroit précis.

— Le rajout d'un serveur de communication : le PI PICO à bord de la nacelle est déjà utilisé pour faire actionner les servo moteurs,et le Pi Zero 2 W est dedié a la sommunication seriale est n'est pas capable de faire du traitement d'image qui permetterais de faire une fusion avec le projet Alvéoles. Il serait donc intéressant de rajouter un serveur ou une carte embarquée, avec une puissance de calcul plus importante pour faire des taches plus complexes.

— Rajouter la capacité d'utiliser le MINI 3 : nous avons vu que l'utilisation du MINI 3 n'est pas possible pour le moment, mais il serait intéressant de rajouter cette capacité pour augmenter la comptabilité de notre projet avec d'autres types de drones. De plus avoir plusieurs drones "lourds" accrochés au Matrice 300 permettrait d'exploiter au maximum la capacité de transport du Matrice 300.

— Rajouter la capacité de faire des actions en parallèle : pour le moment, notre projet ne permet pas de faire des actions en parallèle, il serait intéressant de rajouter cette capacité pour augmenter la vitesse d'exécution des actions. Comme par exemple faire décoller un drone pendant que l'autre est en train de se poser, ou décoller plusieurs drones en même temps.

— Avoir plusieurs drones sur le *Matrice300* : notre implémentation ne permet d'attacher qu'un seul drone au *Matrice300*. Cependant, par manque de temps — notamment pour l'impression d'autres systèmes de fixation, l'acquisition de servomoteurs supplémentaires, et leur intégration — nous n'avons pas pu concrétiser cette amélioration.

— Avoir un environnement portable de code : avoir un environnement portable de code Pour le moment, notre code tourne nativement sur un ordinateur, il serait intéressant de faire un conteneur docker pour pouvoir lancer notre code sur n'importe quel ordinateur.

— La conception d'un système de contrôle à distance via radio ou IP pourrait être une large piste d'amelioration, ou l'on pourrait imaginer de vrai "use-case" sur de grandes distances.

7.2 Bilan

Ce projet nous a permis de mettre en pratique plusieurs notions acquises tout au long de notre parcours. Nous avons pu accomplir la majorité des objectifs fixés, notamment :

— La fixation d'une nacelle portant un drone sur le Matrice 300

— La conception d'un système de largage

— L'exécution de commandes à distance sur le (s) drone (s) porté (s) ou largué (s)

Cependant, ces réalisations peuvent encore être améliorées et perfectionnées grâce à l'ajout de nouvelles fonctionnalités et à l'optimisation de celles existantes, aussi bien sur le plan logiciel que matériel.

Le but principal du projet a été atteint avec succès, bien que certains objectifs aient dû être adaptés en cours de route. Par exemple, nous avons dû remplacer le Mini 3 par le Tello Edu en raison des contraintes techniques rencontrées ou alors l'abandon de l'utilisation des modules radios NRF24L01.

Il existe plusieurs pistes d'amélioration pour ce projet, que ce soit :

— Dans le domaine logiciel (optimisation et développement du code).

— Dans le domaine matériel (amélioration des systèmes électroniques et de fixations).

8 Conclusion

Ce projet de drone porteur nous a permis d'explorer divers aspects techniques, allant de la conception mécanique et électronique à l'automatisation logicielle. Malgré certaines difficultés rencontrées, comme les limitations du Mini3 ou la nécessité d'adaptations sur le Matrice300, nous avons pu atteindre nos objectifs principaux. Nous avons conçu et fixé une nacelle fonctionnelle, développé un système de largage efficace et démontré la capacité du Matrice300 à transporter et larguer des drones plus petits. Le Tello Edu ainsi que le Mini3 nous ont aussi permis d'étudier et implémenter des systèmes qui permettent de contrôler et communiquer avec nos premiers drones.

Toutefois, plusieurs pistes d'amélioration restent ouvertes, notamment l'optimisation du code et du matériel, l'intégration de nouveaux drones, ainsi que la possibilité d'exécuter plusieurs actions en parallèle.

Ce projet représente une base solide pour des développements futurs et pourrait être fusionné avec d'autres initiatives, renforçant ainsi son utilité et son applicabilité dans des scénarios réels.

9 Bibliographie

— DJI MINI 3 SDK - DJI Mobile SDK.
— DJI MATRICE 300 SDK - DJI Onboard SDK.
— DJI OnBoard Computer - Manifold 2.
— Drone militaire - MQ-1B.
— DJI Tello Edu API - Tellu Edu API.
— Le code source du projet - Github.
— Le site du projet - Drone Porteur.
— Sujet en rapport - UAV Payload drop mission