# PDP - TWIN

**Rapport**

Paulin Bardon Marcel Ezedine Melvin Bertrand
Mahamat Younous Abdraman

Printemps 2023

# Remerciements

Nous souhaitons remercier monsieur Chaumette et monsieur Cubilier de nous avoir accompagnés pendant le projet ainsi que de nous avoir fourni un drone s500 afin de faire des tests sur du matériel réel.

# Table des matières

# 1 Introduction

## 1.1 Contexte

Les drones et essaims de drones sont des sujets d'actualités à cause de leurs utilisations massives dans les domaines militaires et publics depuis quelques années.

## 1.2 Présentation du sujet

L'article « aMDH and TWIN : Two original honeypot-based approaches to protect swarms of drones »[CC21] écrit par Serge Chaumette et Titien Cubilier présente deux approches originales basées sur des honeypots pour protéger les essaims de drones contre les accès externes non autorisés. Nous résumons ces deux approches dans les sections suivantes.

Ce projet a pour but d'implémenter une preuve de concept du honeypot TWIN présenté dans cet article.

### 1.2.1 TWIN

TWIN consiste à avoir un second drone dans un environnement contrôlé, invisible pour l'utilisateur final. Lorsqu'un accès non autorisé est détecté sur le drone en opération, cet accès est redirigé du drone en opération vers le drone en environnement contrôlé sans que l'utilisateur s'en rende compte.

Les opérations sensibles demandées par l'attaquant sont effectuées sur le drone en environnement contrôlé tandis que les actions jugées non sensibles sont effectuées sur le drone en opération (par exemple demander les informations du capteur de température). Cela permet d'avoir des données cohérentes et de renforcer l'impression pour l'attaquant qu'il contrôle le drone en opération.

### 1.2.2 MDH

MDH consiste à multiplier le nombre de drones détectés par un attaquant en équipant les drones en opération avec plusieurs microcontrôleurs simulant un drone.

Les drones simulés n'ayant pas besoin d'informations provenant du vrai drone, cette approche peut être mise en place très facilement en équipant simplement des drones avec les microcontrôleurs sans se soucier de les connecter au drone hôte.

Pour tromper l'attaquant, les drones simulés peuvent altérer certaines informations renvoyées, par exemple bouger les coordonnées GPS de quelques mètres pour qu'un missile lancé vers le drone ne le touche pas.

### 1.2.3 Utilisation simultanée des deux approches

Une opération étant limitée dans le temps, le but est de faire perdre le plus de temps possible à l'attaquant pour que l'opération se termine avant qu'il n'ait eu le temps de contrôler un drone en opération.

La multiplication des drones avec MDH réduit les chances de se connecter à un drone réel. Même si l'attaquant se rend compte qu'il est sur un drone simulé, rien ne garantit que sa prochaine tentative de connexion sera sur un drone réel. Si l'attaquant trouve un vrai drone, l'utilisation de TWIN permet de l'empêcher de contrôler le drone sensible tout en lui faisant croire que c'est le cas.

## 1.3 Analyse de l'existant

Un honeypot (leurre) est un dispositif qui doit attirer le regard de l'attaquant et lui faire perdre un certain temps pour nous permettre de réagir.

Les leurres actuels sont en majorité utilisés dans les serveurs, il en existe peu pour les véhicules autonomes, et encore moins pour les véhicules aériens[CC21].

### 1.3.1 PX4

Le projet PX4 Autopilot est un système open source de pilote automatique pour les véhicules autonomes. Il sert à la conception, la construction et le contrôle de véhicules autonomes pour une grande variété d'applications.

PX4 permet la communication entre un véhicule autonome et une station de contrôle via le protocole MAVLink.

### 1.3.2 MAVLink

Le protocole MAVLink[Mav] (Micro Air Vehicle Link) est un protocole de communication conçu pour les véhicules autonomes et télécommandés, qu'ils soient aériens, terrestres ou maritimes. Il a été créé en 2009 par Lorenz Meier, Kevin Sartori et d'autres membres de l'équipe du projet PX4 dans le but de fournir une

interface standardisée et extensible pour la communication entre les véhicules aériens télécommandés et les stations terrestres.

Il permet d'envoyer des commandes à un véhicule ainsi que de transmettre des données télémétriques comme la position, la vitesse, l'altitude et l'état des capteurs, des moteurs et d'autres informations sur le véhicule.

Le protocole MAVLink est largement utilisé pour les drones.

# 2 Approche

## 2.1 Proxy

Notre approche consiste à mettre un Raspberry Pi avec un proxy entre les clients et le drone en opération qui triera les commandes reçues. Les commandes sensibles sont renvoyées vers le drone en environnement contrôlé tandis que les commandes non sensibles sont envoyées au drone en opération.

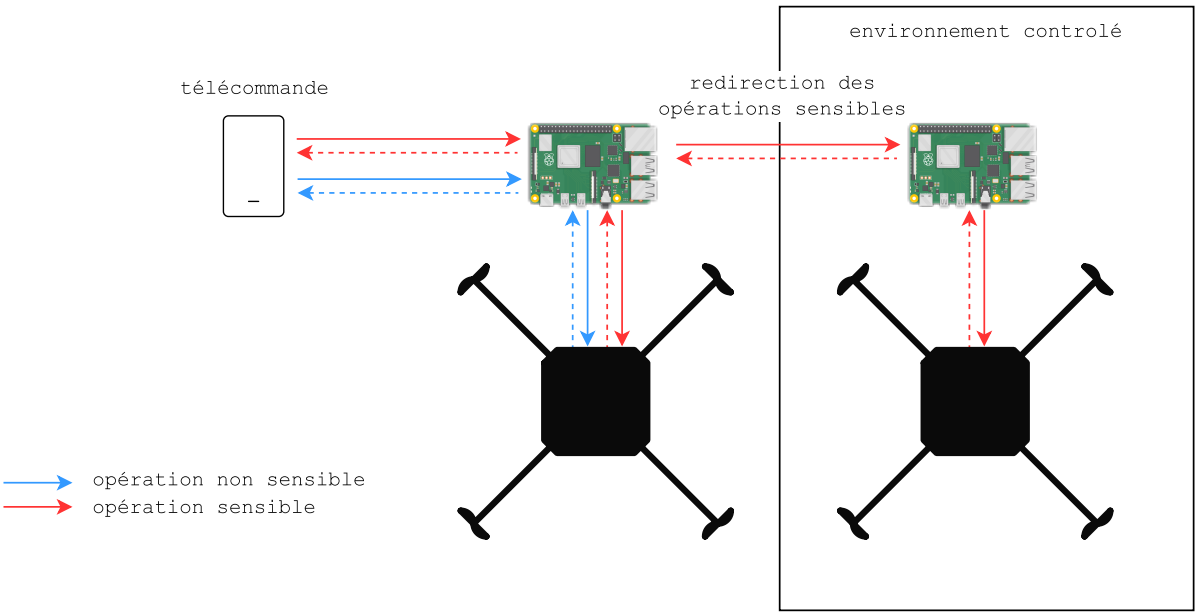



FIGURE 1 – Fonctionnement de TWIN

Pour cela, le proxy décode les commandes reçues puis recréé un paquet avec la même commande pour l'envoyer au drone choisi.

Le choix des commandes sensibles et non sensibles est laissé à l'utilisateur. Certaines commandes sensibles telles que l'activation de la caméra peuvent difficilement être

simulées (la simulation d'une caméra pourrait être un projet à part entière) et devront donc être effectuées sur le drone en opération.

## 2.2 Application pour communiquer avec le drone

Afin de tester le système que nous avons créé, nous avons besoin d'une application permettant de piloter un drone plus facilement. Dans le contexte de notre projet, nous n'avons pas besoin de piloter complètement le drone, on veut simplement pouvoir lui envoyer certaines instructions comme "décoller", "atterrir", et pouvoir récupérer et afficher des données comme les coordonnés GPS du drone, ou même l'affichage de sa caméra.

Nous avons pensé à une application Android avec une interface décrite dans la figure 2.

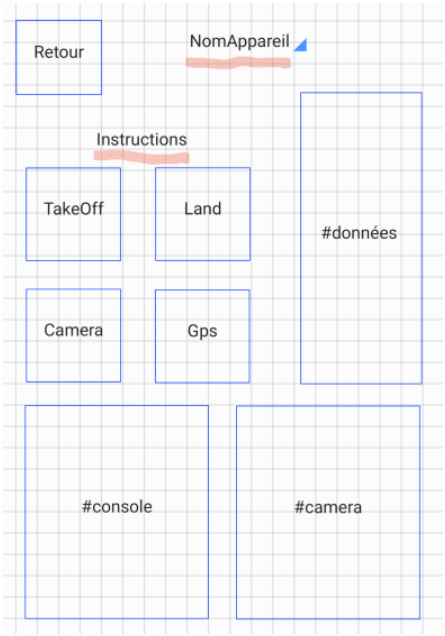



FIGURE 2 – Schéma de l'infrastructure de l'application

Elle est composée de 4 boutons pour les instructions à envoyer au drone. On a également une zone de texte dans laquelle on affichera les données que l'on reçoit, comme les coordonnées GPS, ou si on veut rajouter par la suite d'autres instructions qui permettrait de récupérer des informations issues de capteurs supplémentaires comme la température ou la pression de l'air. En dessous, une zone d'affichage pour le flux vidéo transmis par le drone. Tout en bas, la console permet d'avoir les détails des transactions entre le drone et l'application.

L'application ne sait pas quel drone elle contrôle, il s'agit simplement d'une

télécommande.

On peut ainsi établir une liste de besoins fonctionnels et non fonctionnels comme ceci :

### 2.2.1 Besoins fonctionnels

— L'utilisateur doit pouvoir choisir le drone qu'il veut contrôler.
— La connexion au drone doit s'effectuer lorsque l'utilisateur clique dessus.
— L'utilisateur doit pouvoir faire décoller le drone à l'aide d'un bouton.
— On pourra décider à quelle hauteur le drone décolle dans un certain intervalle (ex : 0-10 mètre).
— L'utilisateur doit pouvoir faire atterrir le drone lorsqu'il est dans les airs.
— L'utilisateur doit pouvoir faire atterrir le drone lorsqu'il est dans les airs.
— L'utilisateur doit pouvoir afficher les coordonnées GPS du drone en temps réel.
— L'utilisateur doit pouvoir afficher le flux vidéo de la caméra en temps réel.

### 2.2.2 Besoins non fonctionnels

— La liste des drones pilotables doit s'afficher sur l'écran d'accueil.
— L'application doit être lisible.
— L'application doit être facile à utiliser.
— L'application doit être rapide et réactive.
— L'application doit être fiable pour ne jamais envoyer de mauvaises instructions au drone.

# 3 Réalisation

## 3.1 Outils utilisés

### 3.1.1 MAVSDK

MAVSDK[MAV21] est un kit de développement logiciel open-source qui fournit une API de haut niveau pour le protocole MAVLink.

### 3.1.2 Pymavlink

Pymavlink est une bibliothèque Python qui implémente le protocole MAVLink. Elle contient des fonctions de plus bas niveau que MAVSDK qui permettent de créer nos

propres paquets MAVLink ainsi que de décoder les paquets entrants.

Nous nous en servons pour communiquer entre notre proxy (sur le Raspberry Pi qui contrôle le drone) et l'application ou le programme de commande.

La version actuelle de PX4 et des simulations semble avoir un problème de compatibilité avec certaines fonctions. Nous utiliserons donc également MAVSDK pour éviter les bugs.

### 3.1.3 Drone s500

Le drone s500 est un drone en kit disponible à l'achat. Il est facile à monter et permet une grande flexibilité des composants, ce qui est parfait pour des développeurs. Le drone que nous avons est équipé d'un contrôleur Pixhawk.

### 3.1.4 Pixhawk

Pixhawk est un contrôleur de vol ouvert conçu pour les drones et les véhicules autonomes. Il est basé sur une carte électronique qui intègre un processeur, des capteurs, des ports d'entrée/sortie et d'autres composants nécessaires pour contrôler le vol d'un drone. Il nous sert à contrôler le drone s500.

PX4 peut être installé directement sur le Pixhawk ou sur un Raspberry Pi.

### 3.1.5 Raspberry Pi

Le Raspberry Pi est un mini ordinateur qui nous permet d'embarquer le proxy directement sur le drone.

### 3.1.6 Android Studio

Android Studio est un environnement de développement pour créer des applications Android. Il prend en charge Java, le langage de programmation orienté objet le plus utilisé pour développer des applications Android. L'interface de l'application est définie avec des fichiers XML, donc très simple à manipuler. En plus de fournir un éditeur de code, Android Studio permet d'émuler un appareil Android pour tester directement l'application, un outil de conception d'interface utilisateur très ergonomique, et bien plus encore.

## 3.2 Proxy simple

### 3.2.1 Serveur udp-mavsdk

Nous avons commencé par implémenter un premier serveur. Il utilise le module `socket` pour recevoir des chaînes de caractères depuis un client. Ensuite, en fonction de la chaîne reçue, il va envoyer différentes instructions avec le protocole MAVLink vers la simulation en utilisant la bibliothèque MAVSDK. Il sert d'intermédiaire pour contacter le drone dans la simulation.

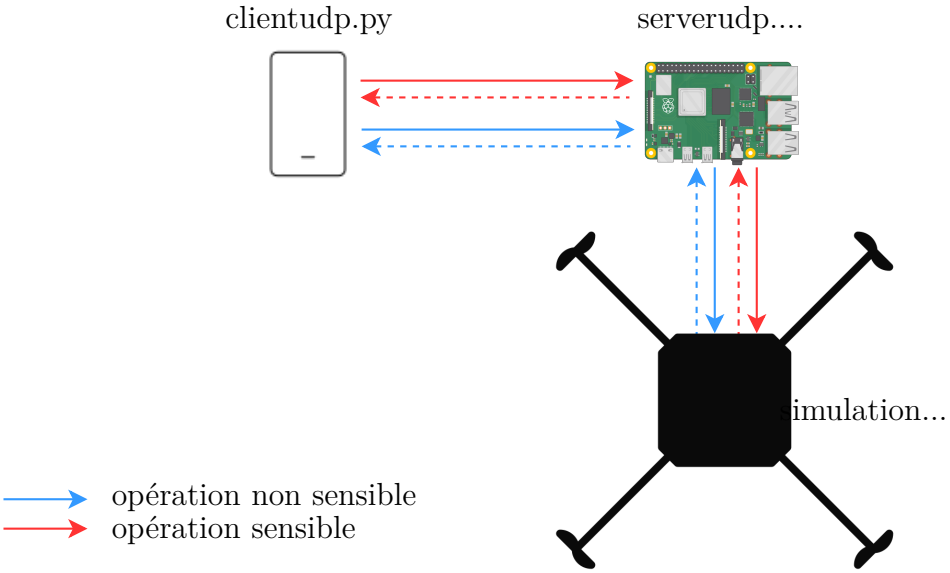



FIGURE 3 – Fonctionnement d'un serveur proxy

Ce serveur permet d'occulter le contrôle du drone pour l'utilisateur et de prendre en main l'architecture de réception et de redirection.

```
while(True):
    bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)

    message = bytesAddressPair[0]

    address = bytesAddressPair[1]

    clientMsg = "Message from Client:{}".format(message)
    clientIP = "Client IP Address:{}".format(address)
    clientDrone= message.decode()
    print(clientMsg)
    print(clientIP)
    print(clientDrone)

    if(clientDrone == "connect"):
        asyncio.run(connect())
        msg = "drone connect"
        bytesToSend = str.encode(msg)
        #message back
        UDPServerSocket.sendto(bytesToSend, address)
```

Listing 1 – Code du serveur, boucle principale montrant la réception en UDP et la transmission en MAVLink

### 3.2.2 Pour aller plus loin

Pour améliorer ce projet, nous pouvons rajouter l'architecture TWIN en ajoutant un deuxième drone connecté dans la simulation et en exécutant les takeoffs dessus. Cependant, la simulation plante si nous lançons deux drones à l'intérieur, nous empêchant de tester ce procédé.

Il faudra penser à calculer le décalage de la position et des autres instruments pour masquer l'utilisation de deux drones.

Nous allons également utiliser le protocole MAVLink de bout en bout pour permettre le contrôle du drone depuis une télécommande ou une application compatible MAVLink.

## 3.3 Serveur proxy TWIN

### 3.3.1 Fonctionnement

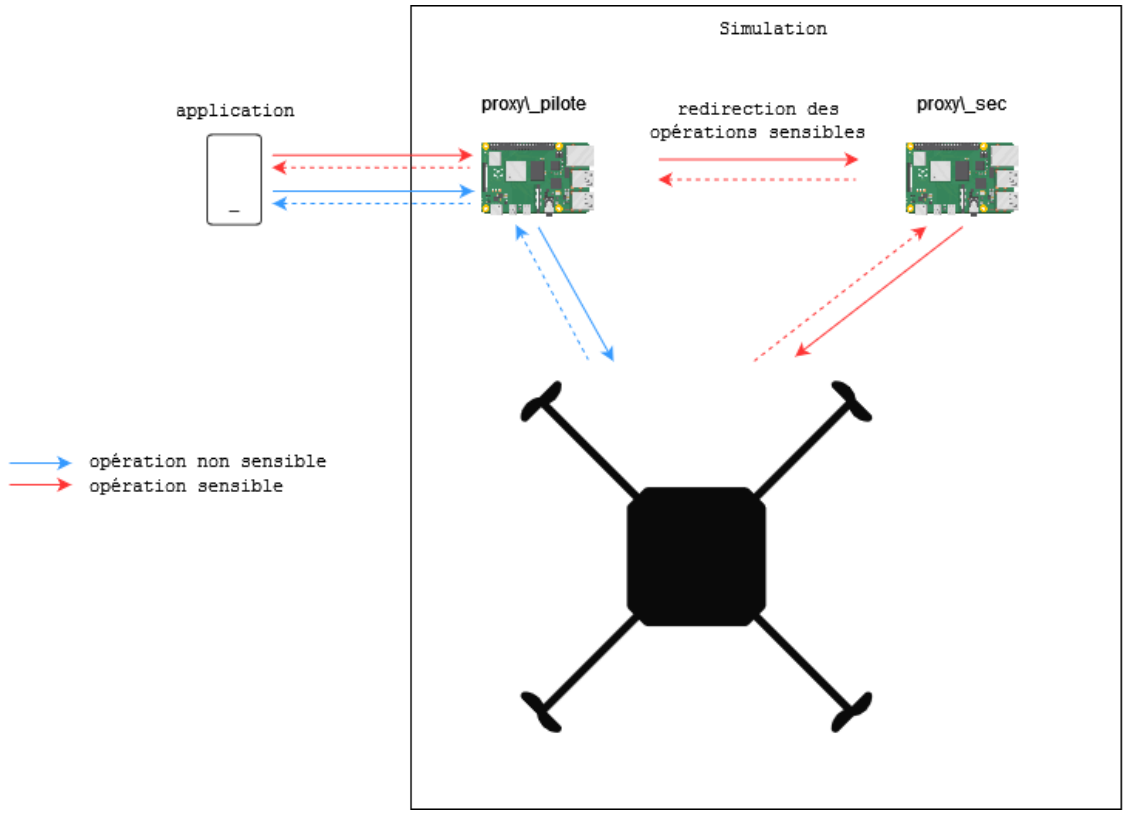



FIGURE 4 – Fonctionnement de l'exemple TWIN implémenté

Pour implémenter la solution TWIN, nous nous sommes confrontés à de nombreux problèmes provenant de l'architecture PX4.

Nous n'avons pu utiliser qu'un seul drone dans la simulation. Ce drone reçoit des messages MAVLink depuis deux serveurs différents qui représentent le Raspberry Pi du drone cible et celui du drone sécurisé.

Depuis l'application, les commandes sont toutes envoyées avec Pymavlink vers le serveur pilote**??** (celui du drone cible). Si une instruction n'est pas sensible, le serveur pilote va exécuter l'action sur le drone de la simulation.
Si une instruction est sensible, le serveur pilote va envoyer la commande au serveur secondaire**??** (celui du drone sécurisé) qui va effectuer l'action sur le drone de la simulation.

On reconnaît l'architecture TWIN dans la communication entre les deux serveurs malgré l'impossibilité de l'exécuter sur deux drones dans la simulation.

Les figures 5 et 6 montrent le proxy en fonctionnement.

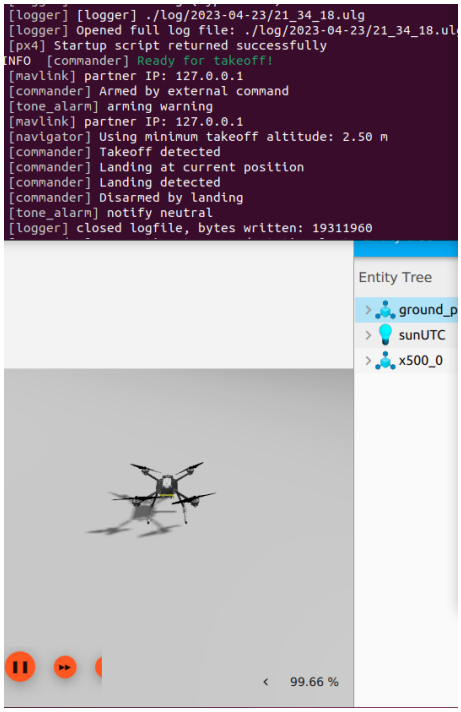



FIGURE 5 – Drone en fin de ”land” dans la simulation

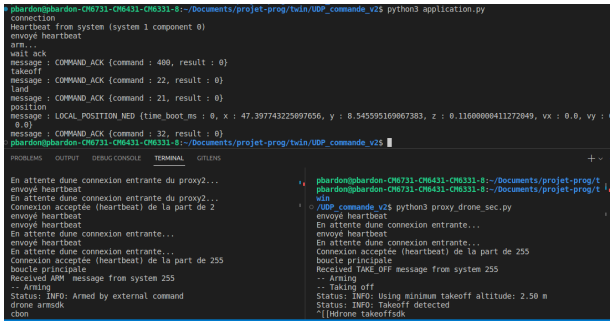



FIGURE 6 – affichage de fonctionnement dans le terminal application / proxy pilote / proxy sec

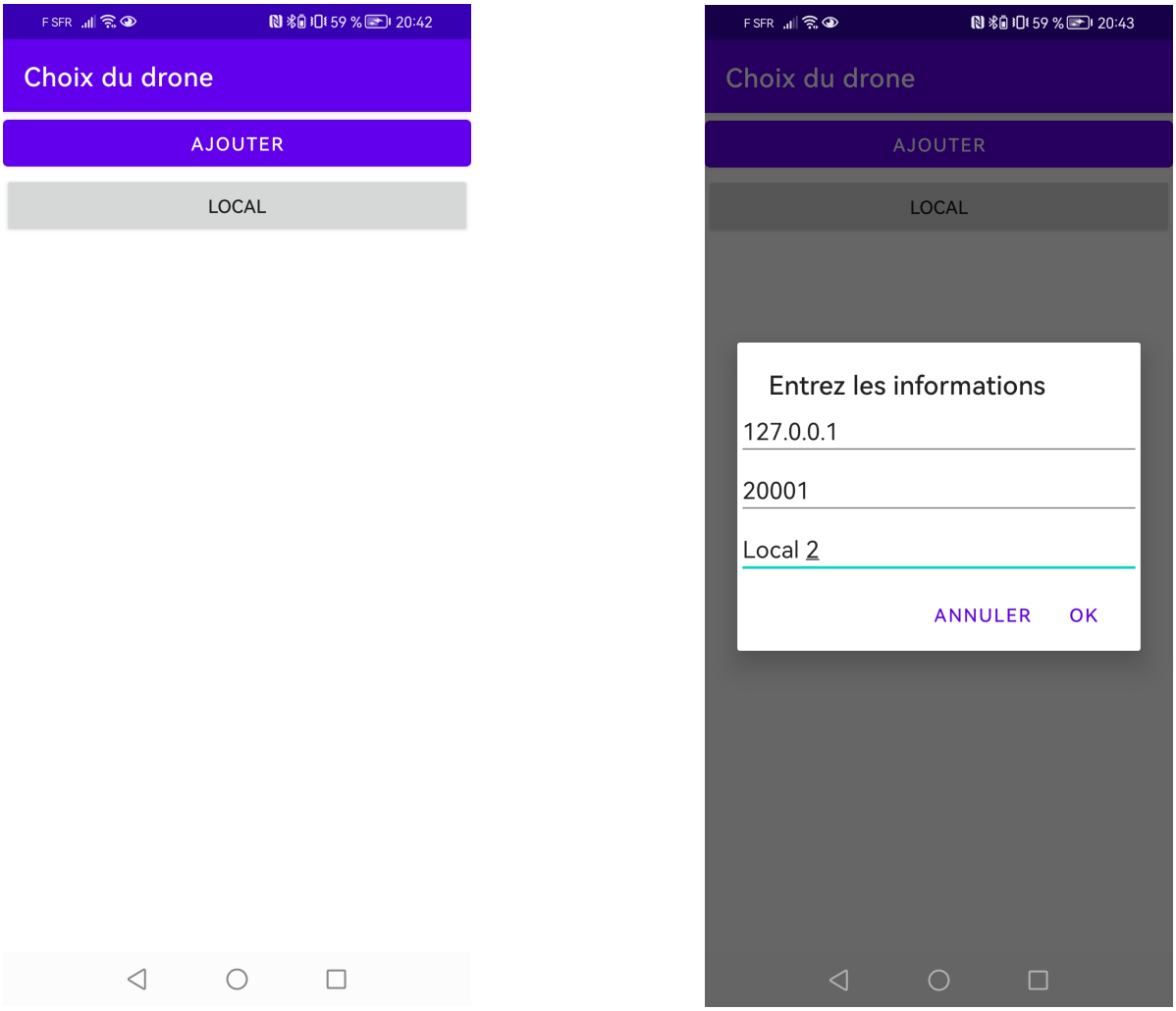
## 3.4 Application Android

Le développement de cette application s’est fait principalement en 2 étapes.

En premier lieu, nous avons créé l’infrastructure comme prévu dans le schéma. Nous avons ensuite implémenté les boutons pour qu’ils envoient les bonnes instructions au serveur dont l’adresse IP et le port d’écoute ont été entrés au préalable par l’utilisateur.

Finalement, l'infrastructure de l'application est composée de la manière suivante : au démarrage de l'application, nous sommes sur une fenêtre qui nous propose d'ajouter des drones à la liste, ou de choisir un des drones de la liste à piloter.

L'appui sur le bouton « ajouter »ouvre une fenêtre de dialogue qui propose d'entrer les informations nécessaires pour piloter le drone.



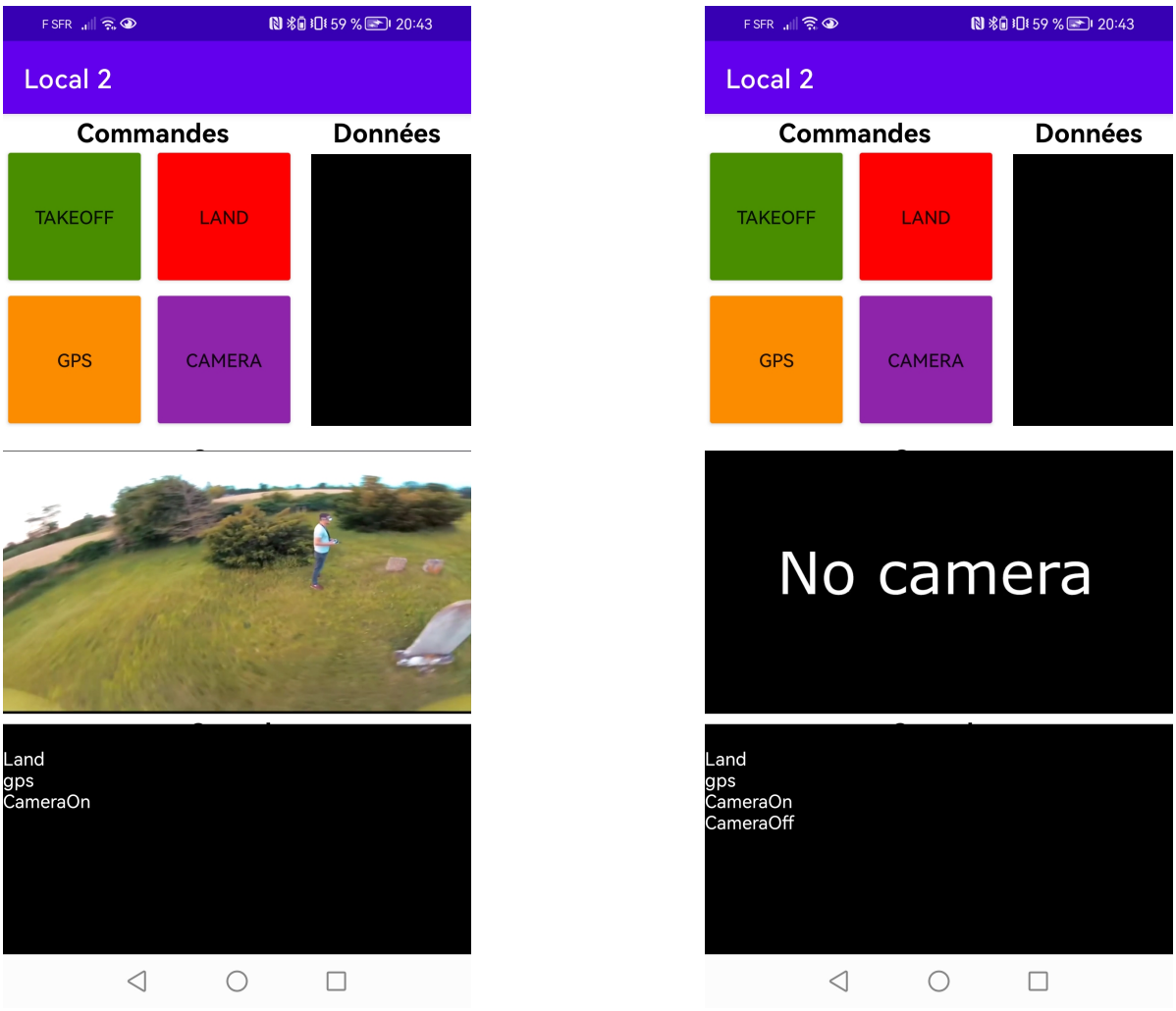

(a) Écran d'accueil

(b) Fenêtre de dialogue

En cliquant sur un des drones de la liste, l'application va entrer en contact avec le drone et nous sommes redirigés sur l'écran de pilotage du drone, qui est très similaire à l'idée d'infrastructure que nous avons eu à la base.

Ainsi, le fonctionnement de notre application pourrait être résumé avec le diagramme de séquence suivant :

La création de l'infrastructure s'est passé sans trop de problème. C'est en implémentant les boutons que nous avons rencontré des difficultés. Premièrement, nous




(a) Caméra activée

(b) Caméra désactivée

voulions qu'un clic sur le bouton `land`, par exemple, envoie au serveur, une chaîne de caractères avec la valeur `land`. Nous avons implémenté la fonction « `onClick` »qui est appelée lors d'un appui sur le bouton, et envoie la chaîne au serveur. À ce moment-là, nous avons eu une erreur « `android.os.NetworkOnMainThreadException` », ce qui signifie que nous avons effectué une opération réseau sur le thread principal au lieu d'utiliser un thread dédié. Pour régler ce problème, nous avons utilisé une `asynchTask`, qui est une classe permettant de réaliser des tâches en arrière-plan. Ainsi, nous avons une classe `UDPClient` qui étend la classe `asynchTask`, et qui contient une fonction `doInBackGround` qui envoie un message donné dans le constructeur au serveur, également donné dans le constructeur. On instancie ensuite dans le bouton un `UDPClient`, sur lequel on appelle la fonction `execute`, qui va exécuter, en arrière-plan, le contenu de la fonction `doInBackground`. Par exemple, un clic sur le bouton « `land` »enverra une chaîne de caractères « `land` »au serveur comme ceci :

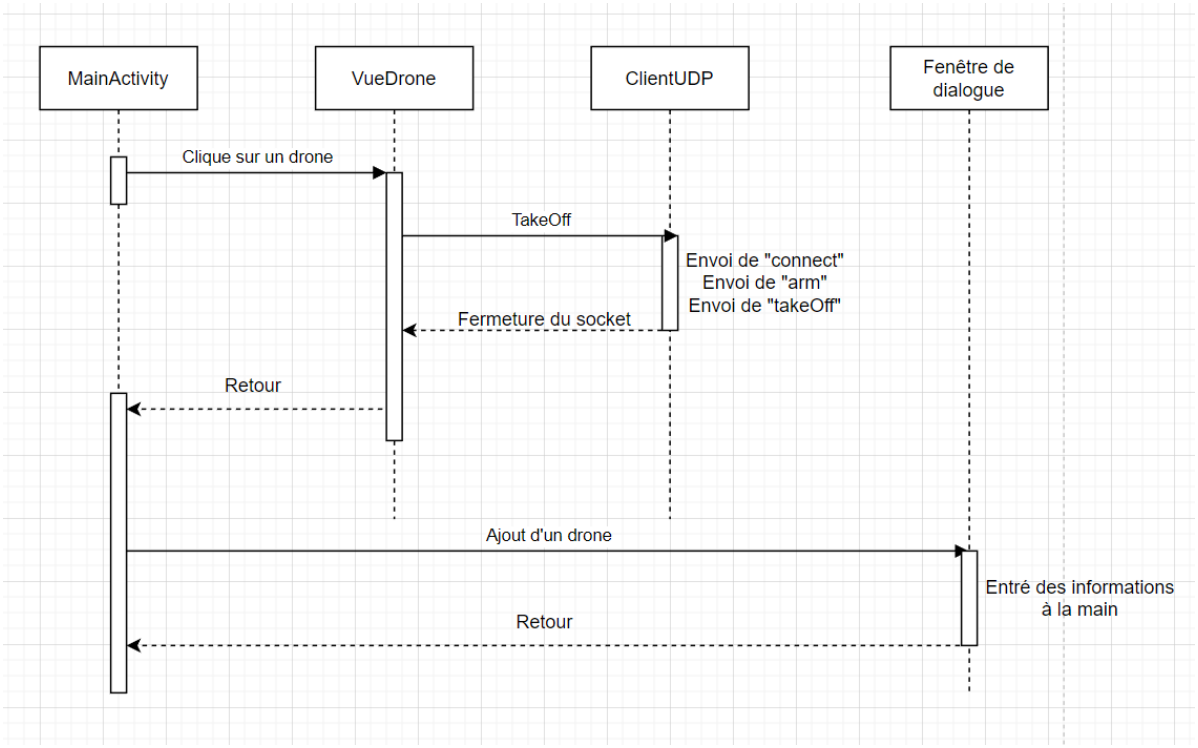



FIGURE 9 – Diagramme de séquence

```
UdpClient udpClientTask =
new UdpClient(finalServerAddress, serverPort, sendData);
udpClientTask.execute();
```

Avec `finalServerAddress`, l'adresse du serveur, `serverPort`, le port du serveur, et `sendData`, le message que l'on souhaite envoyer.

Ainsi, le problème de thread est réglé. Cependant, bien que le paquet s'envoie sans problème, le serveur ne le réceptionne pas.

# 4 Conclusion

Nous avons réussi à implémenter partiellement la solution TWIN proposée. Les limitations de la simulation ne nous ont pas permis de tester avec deux drones séparés. Le but étant de proposer une preuve de concept, le proxy n'est pas prêt à être utilisé en situation réelle, mais il peut être amélioré dans ce but.

## 4.1 Difficultés et contraintes

Le sujet de notre projet était intéressant et nouveau pour nous. Il nous était difficile de progresser rapidement, car les outils utilisés ne nous étaient pas familiers. En particulier, nous avons eu du mal à nous familiariser avec le protocole MAVLink. Des problèmes supplémentaires ont été posés par la documentation incomplète ou obsolète de certains outils.

# Références


[CC21] Serge Chaumette et Titien Cubilier. « aMDH and TWIN : Two original honeypot-based approaches to protect swarms of drones ». In : *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE. 2021, p. 783-787.

[Mav] *MAVLink Common Message Set*. MAVLink. URL : https://mavlink.io/en/ (visité le 23/04/2023).

[MAV21] MAVSDK Development Team. *MAVSDK*. https://mavsdk.mavlink.io/main/en/index.html. Accessed April 23, 2023. 2021.