



ÉCOLE NATIONALE SUPÉRIEURE DE TECHNIQUES AVANCÉES

---

# Système de Surveillance et de Secours par Drones

---

Auteurs :

Juan RAMIREZ  
Vanessa LOPEZ

Encadrants :

M. Emmanuel BATTESTI  
M. Thibault TORALBA

ROB314 - Architecture matérielle et logicielle pour la robotique

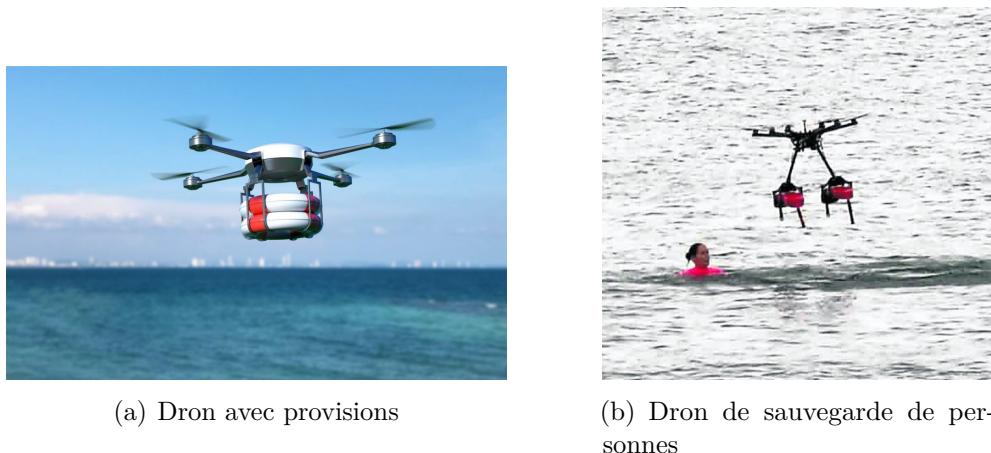
5 avril 2024

# 1 Introduction

## 1.1 Contexte

Ce projet trouve son origine dans la nécessité de localiser les individus après une catastrophe. Lorsque survient un événement de cette envergure, il s'avère essentiel d'allouer un grand nombre de ressources humaines et un temps considérable pour couvrir l'intégralité de la zone sinistrée. Pour remédier à cette problématique, l'idée d'utiliser des drones équipés de provisions afin de repérer les personnes en attente d'assistance, en attendant l'arrivée des équipes de secours sur le terrain, a été envisagée. Ainsi, lorsqu'un drone repère une personne, les autres drones convergent vers ce même point pour fournir des provisions supplémentaires.

Cette approche peut être mise en œuvre dans divers contextes, notamment pour la localisation de personnes en mer ou pour adapter la cible de détection afin de repérer des incendies dans de vastes étendues telles que de grandes forêts.



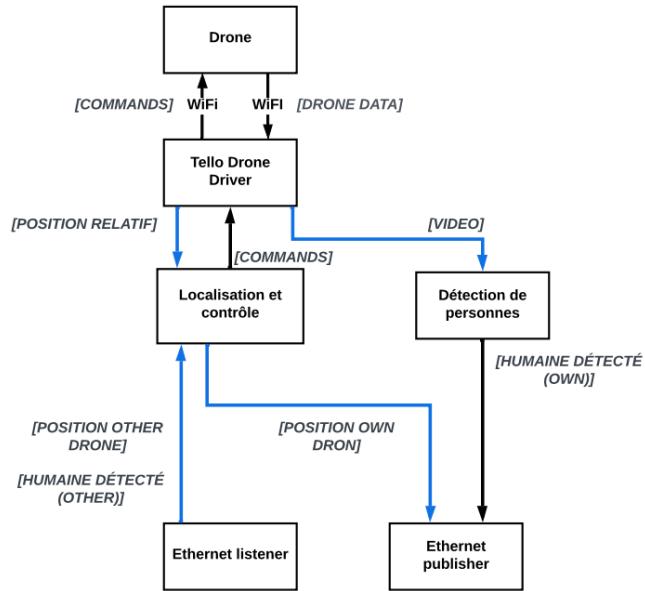
**Figure 1 – Possibles applications du projet**

## 1.2 Description du projet

Dans le cadre de ce projet, l'objectif principal est d'assurer la navigation autonome de deux drones vers des points spécifiques de l'espace (chaque drone ayant pour cible un point différent). Cette démarche vise à simuler l'exploration effectuée par les drones dans un environnement donné.

Durant leur vol, les deux drones opéreront simultanément une surveillance visuelle à l'aide de leur caméra pour détecter la présence d'êtres humains. Si l'un des drones détecte un humain, il interrompra son vol pour se positionner à l'endroit où la détection a été effectuée, tout en signalant sa découverte à l'autre drone. Ce dernier se dirigera alors vers le point où se trouve l'humain, reproduisant ainsi le processus visant à fournir des ressources telles que de la nourriture, des médicaments, dans des zones difficiles d'accès, ressources qui initialement ne pourraient pas être transportées par un seul drone.

Le diagramme de blocs ci-dessous illustre la structure générale du système, avec chaque composant étant détaillé dans la suite de ce rapport.



**Figure 2** – Diagramme de blocs du projet

## 2 Détection des personnes

Dans le cadre du projet, l'objectif principal des drones est de localiser des individus dans des environnements potentiellement dangereux ou difficiles d'accès. Pour accomplir cette tâche, nous utilisons un modèle de détection et de segmentation d'images appelé *Ultralytics Yolov8* [3].

Nous avons choisi ce modèle principalement parce qu'il permet une exécution en temps réel [5], ce qui est critique dans des situations où la recherche constante de personnes doit être effectuée simultanément avec les algorithmes de contrôle et de navigation.

Afin d'utiliser ce modèle pour la détection de personnes, il est nécessaire dans un premier temps d'accéder aux données vidéo du drone. Cette tâche est réalisée dans le nœud **introduction**, où deux actions sont effectuées simultanément après la connexion au drone via WiFi :

- **Threading** : Nous consacrons un thread spécial à la lecture de la vidéo et à sa transformation en une image *OpenCV* pour son traitement ultérieur avec *YoloV8*.

Pour ce faire, nous capturons les images vidéo du flux UDP du drone, qui sont enregistrées dans la variable `vidOut`, accessible depuis n'importe quelle autre fonction ou partie du code.

```

def cam():
    try:
  
```

```

5           global vidOut
6           global stop_cam
7           global cam_error
8           cap = cv2.VideoCapture("udp://@127
9               .0.0.1:5000")
10          if not cap.isOpened():
11              cap.open()
12          while not stop_cam:
13              res, vidOut = cap.read()
14          except Exception as e:
15              cam_error = e
16          finally:
17              cap.release()
18          print("Video Stream stopped.")

camt = threading.Thread(None, cam) # Start thread
camt.start()

```

- **Loopback** : Toujours dans le noeud `integration`, nous nous abonnons au drone via le topic `EVENT_VIDEO_FRAME`, facilité par la bibliothèque *TelloPy* [6]. Le callback de cette fonction se charge simplement d'envoyer les données vidéo via un socket UDP à l'adresse vers laquelle la méthode `cam` est constamment dirigée.

```

def videoFrameHandler(event, sender, data):
    loopback.sendto(data, ('127.0.0.1', 5000))

drone.subscribe(drone.EVENT_VIDEO_FRAME,
                videoFrameHandler)
5

```

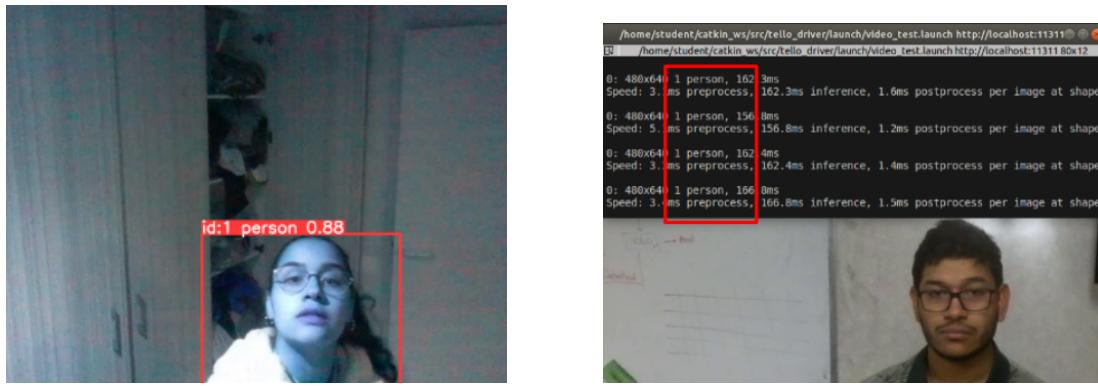
Une fois le flux vidéo accessible en tant que variable appropriée pour le traitement des images (une image *OpenCV*), chaque image est analysée pour déterminer la présence ou non d'une personne.

Le modèle YOLO pré-entraîné détecte initialement des objets parmi un grand nombre de classes, comme le montre la figure suivante.



**Figure 3** – Détection d’objets avec YOLOv8

De cette figure, on peut noter que même la simple présence d’une main humaine est suffisante pour être détectée avec le modèle utilisé. Bien que cela augmente considérablement les capacités de détection du drone d’un point de vue applicatif, cela reste peu pratique dans un environnement de prototypage et lors de un démonstration pour les enseignants du cours. Enfin, le modèle est configuré pour détecter uniquement les personnes au-dessus d’un seuil de confiance, produisant des résultats tels que celui présenté ci-dessous.



(a) Exemple montrant la vidéo

(b) Exemple sans montrer la vidéo

**Figure 4** – Résultat de l’algorithme de détection

Nous avons choisi de ne pas afficher la vidéo ni le cadre de détection (obtenant uniquement le résultat visible dans la console de commande de l’image de droite) pour alléger la charge computationnelle du projet.

### 3 Localisation et contrôle de position

Le positionnement de chaque drone est déterminé à partir des données fournies par son central inertiel, les angles sont donnés sous forme de quaternions ( $q_x, q_y, q_z, q_w$ ) et il n’y a pas de données directes sur la position relative du drone, mais sur sa vitesse linéaire ( $v_x, v_y, v_z$ ).

Cette présentation des données implique qu'une conversion des unités des quaternions en angles d'Euler doit être effectuée, et il est nécessaire d'intégrer les données de vitesse (principalement en  $x$  et  $y$ ) pour obtenir les données de position.

Ainsi, de manière similaire à ce qui a été fait précédemment, nous nous abonnons au drone depuis le noeud `integration` via le topic `EVENT_LOG_DATA`. Dans le callback correspondant, nous effectuons la conversion d'unités décrite précédemment et l'intégration numérique de la vitesse linéaire du drone.

```

def IMUHandler(event, sender, data, **args):
    global imu_data
    imu_data = data.imu

    5      qw = imu_data.q0
    qx = imu_data.q1
    qy = imu_data.q2
    qz = imu_data.q3

    10     vx = imu_data.vg_x
    vy = imu_data.vg_y
    vz = imu_data.vg_z

    15     roll, pitch, yaw = quaternion2euler(qw, qx, qy, qz)
    pos_x, pos_y, pos_z = update_position(vx, vy, vz)

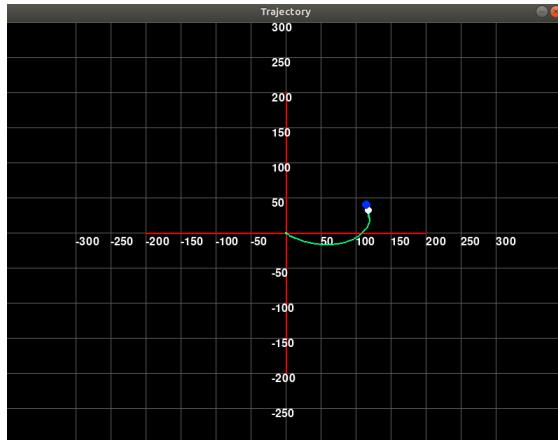
    drone.subscribe(drone.EVENT_LOG_DATA, IMUHandler)

```

Il convient de noter que tant la localisation du drone que la détection des personnes sont réalisées dans le même noeud `integration`. Cela est dû au fait que les données du drone sont transmises via une seule connexion WiFi, et accéder à partir de deux noeuds différents de manière indépendante impliquerait qu'un des deux verrait la connexion occupée. Il serait envisageable de concevoir un noeud chargé de recevoir uniquement les données du drone et d'envoyer des commandes à celui-ci (disons un noeud d'interface avec le drone) pour que celles-ci soient utilisées par un noeud de détection et un noeud de localisation, mais nous considérons que cela complexifierait inutilement le projet sans apporter de nombreux avantages.

En ce qui concerne le contrôle, l'implémentation d'algorithme de planification de trajectoire pour explorer la carte dépasse le cadre de ce projet. Nous cherchons donc à permettre au drone de se rendre à un point arbitraire de la carte (et relatif au drone). Cela implique un contrôle assez simple ; nous avons choisi d'implémenter un régulateur proportionnel plutôt que d'autres régulateurs tels que PI ou MPC en raison de contraintes de temps et du fait que le régulateur proportionnel fonctionnait raisonnablement bien. Le résultat est présenté ci-dessous :

Le contrôle est effectué conjointement avec la localisation dans le même callback, ce qui permet de générer une commande de contrôle à chaque mise à jour de la



**Figure 5** – Control de position

position, évitant ainsi tout retard pouvant déstabiliser la position du drone. La stratégie de ce contrôle consiste donc à calculer l'angle entre la position actuelle et la position souhaitée, à tourner jusqu'à être à une distance  $\epsilon$  de cet angle, et à accélérer ensuite à une vitesse de  $K \cdot (\text{référence} - \text{position})$ .

De plus, nous considérons qu'à une distance  $\epsilon$  de la référence, la vitesse est changée à une vitesse constante pendant de brefs instants pour éviter qu'une faible loi de contrôle due à l'erreur n'arrête effectivement le drone avant d'atteindre la référence.

## ORB-SLAM

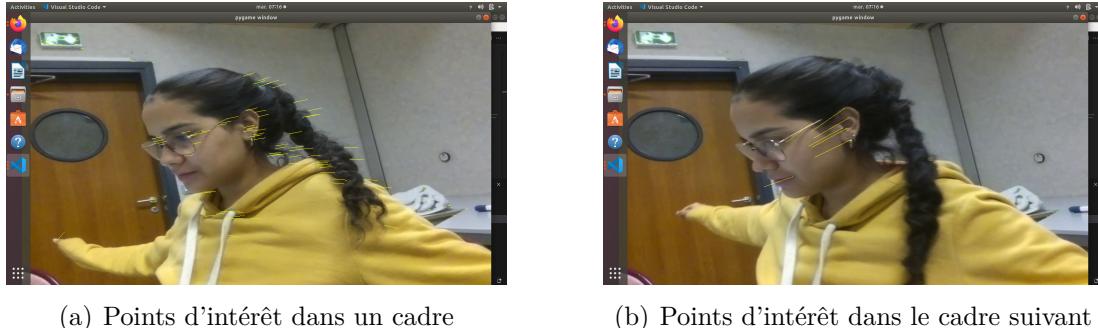
La méthode d'odométrie inertielle sur laquelle nous nous appuyons actuellement présente des problèmes évidents de précision. Cela est dû à la dérive subie lors des vols de drones, que nous ne pouvons pas compenser, ainsi qu'aux problèmes d'intégration numérique et de bruit dans les capteurs, qui accumulent des erreurs au fil du vol.

Il est donc logique de penser à compléter les données inertielles par des données visuelles pour obtenir une mesure de position plus robuste. De plus, les drones sont "aveugles" aux obstacles et sont donc susceptibles de heurter des murs ou d'autres drones s'ils ne sont pas manipulés avec suffisamment de prudence. Par conséquent, l'inclusion d'un algorithme d'évitement d'obstacles viendrait grandement enrichir le projet.

En raison de cela, nous envisageons d'implémenter un algorithme de SLAM, ce qui permettrait de réaliser les deux tâches. D'une part, localiser visuellement le drone, ce qui pourrait être combiné, par exemple, à l'aide d'un filtre de Kalman ou d'un filtre de particules avec les données de l'odométrie. D'autre part, cela permettrait de générer une carte des obstacles afin de pouvoir les éviter et ainsi éviter les collisions avec les murs.

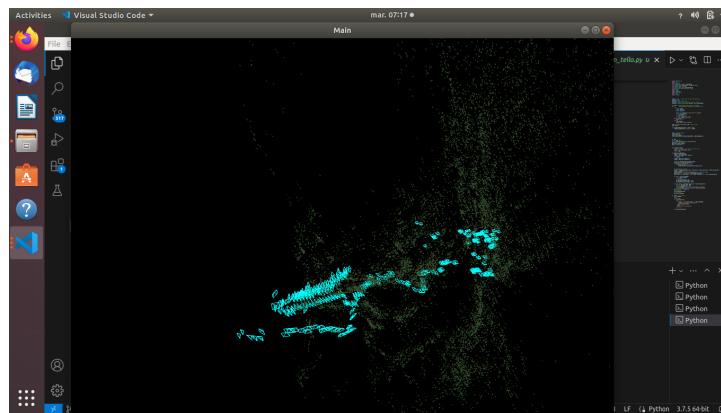
Le choix d'utiliser en particulier ORB comme méthode de correspondance des caractéristiques s'explique par le fait qu'il s'agit d'un algorithme assez rapide

et approprié pour les systèmes en temps réel [4]. Ainsi, grâce à la bibliothèque SLAMPy, nous utilisons ORB-SLAM. [1]



**Figure 6** – Comparaison des points d'intérêt de deux cadres utilisant ORB

En raison de la résolution de la caméra et du fait que le maintien en position des drones n'est pas parfait, ainsi que de la latence entre les cadres due au traitement des autres modules du projet (détection et odométrie), la mise en correspondance des points d'intérêt ne se déroule pas correctement. Il existe des changements très importants pour de très petits mouvements (comme le montre l'image précédente), et il y a une variation très importante du nombre de points d'intérêt à chaque étape. Le résultat de cela est une estimation du mouvement considérablement éloignée de la réalité, comme le montre l'image ci-dessous (le mouvement du drone était une trajectoire circulaire).



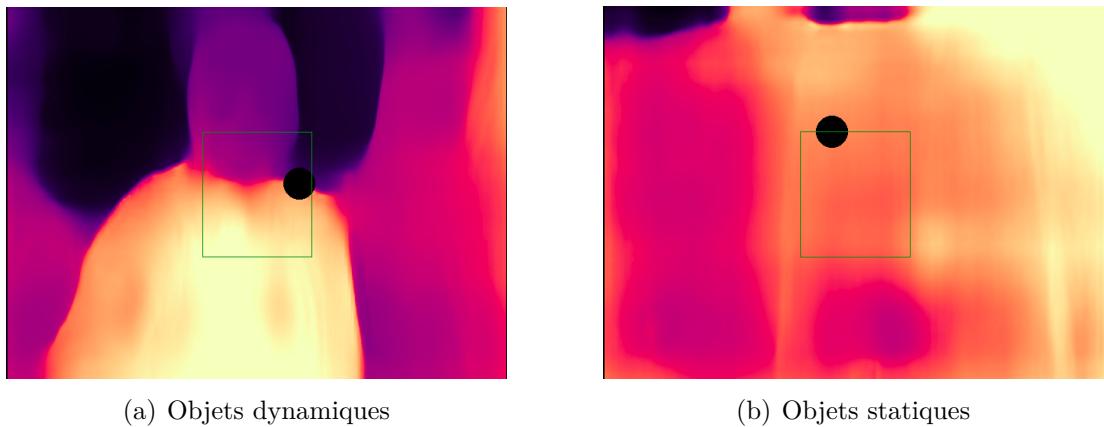
**Figure 7** – Estimation de position avec ORB-SLAM

### ***Estimation de profondeur***

Malgré l'échec de la tentative précédente visant à améliorer la localisation du robot, il existe des alternatives pour résoudre le problème de détection des obstacles. La solution la plus évidente est sans aucun doute la détection de profondeur, qui est agnostique quant au type d'objet en question et se contente simplement d'éviter les collisions avec quelque chose de "proche".

Étant donné que nous disposons d'un système monoculaire, estimer la position est assez compliqué. Les méthodes traditionnelles de stéréoscopie (monoculaire) nécessiteraient de faire correspondre les points pris par la même caméra à deux instants différents. De plus, les erreurs considérables dans l'estimation de la position rendent cette méthode non viable.

Conscients de l'existence de réseaux neuronaux pour estimer la profondeur, nous avons décidé de mettre en œuvre cette solution car elle est compatible avec un système de caméra monoculaire et ne nécessite pas de connaissance a priori des paramètres internes de la caméra. Pour ce faire, nous avons utilisé *MonoDepth2*[2]. Voici les résultats obtenus :



**Figure 8** – Comparaison des performances avec différents types d'objets

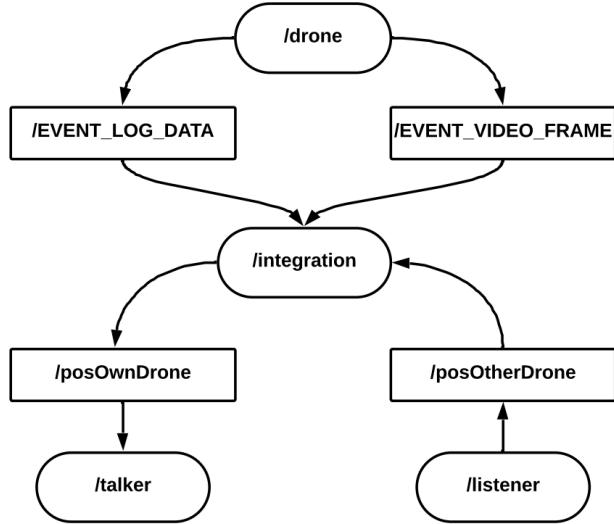
Étant donné que l'ensemble de données est principalement entraîné avec des images tirées de vidéos automobiles, la profondeur est correctement estimée lorsque des personnes ou des objets qui peuvent également être couramment rencontrés dans la rue apparaissent. Cependant, lorsque la tâche est simplement de détecter la distance jusqu'à un mur ou à de petits objets d'intérieur, les performances se détériorent considérablement, rendant l'application inutilisable pour notre cas d'utilisation.

## 4 Communication entre drones

La connexion entre les drones et les ordinateurs se fait via WiFi, ce qui rend complexe l'accès simultané à deux drones à partir d'un seul ordinateur. Pour pallier à cela, nous utilisons un ordinateur distinct pour chaque drone, puis nous relions ces ordinateurs via Ethernet pour transmettre les données de position du drone voisin ainsi que les informations sur la détection d'une présence humaine.

Afin de publier en continu la position d'un drone tout en recevant celle de l'autre, nous avons développé deux nœuds ROS distincts, chacun ayant un unique topic. Ces nœuds sont respectivement nommés **talker** et **listener**, et chacun assure la connexion Ethernet avec l'autre ordinateur en envoyant ou recevant des

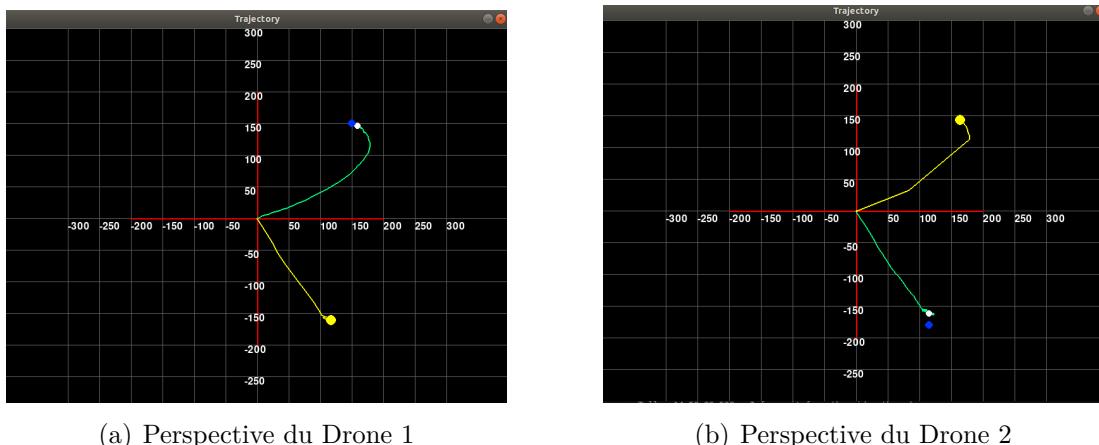
données. Un schéma illustrant les nœuds et topics de ROS implémentés est présenté ci-dessous.



**Figure 9** – Schéma des noeuds et topics ROS

Chacun de ces noeuds `listener` et `talker` est équipé d'un code Python pour permettre la communication via Ethernet avec un délai de 1 seconde afin d'éviter tout problème de synchronisation. Il convient de noter que cette transmission de données inclut également une variable liée à la détection d'une présence humaine en plus de la position du drone.

Le résultat est que dès le départ, lorsqu'un drone se dirige vers sa référence, il intègre dans sa carte la position de l'autre drone. Ainsi, les deux drones sont conscients de la position de l'autre en temps réel, comme illustré ci-dessous.

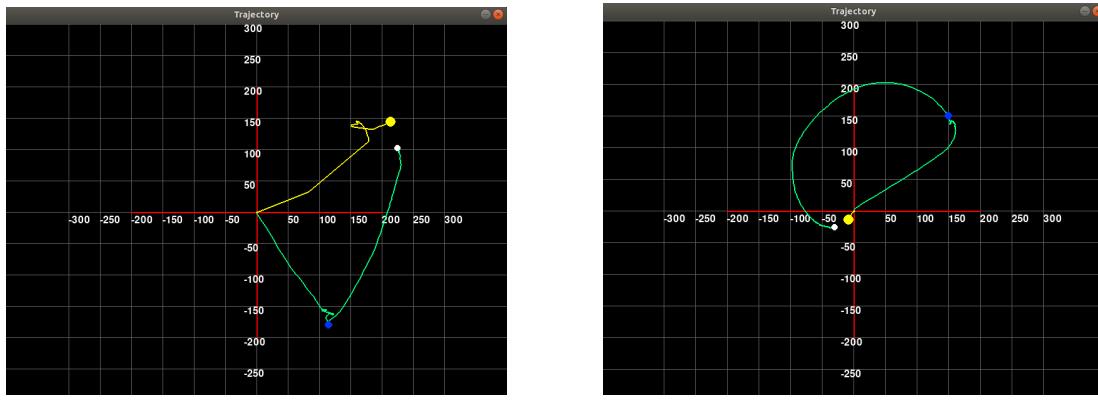


**Figure 10** – Suivi de référence

Dans les deux cas, la ligne verte représente le drone qui trace sa propre position, tandis que la ligne jaune représente celle de l'autre drone. On peut remarquer que

les lignes jaunes sont moins détaillées que les vertes en raison de la latence de la communication.

Dans le cas précédent, aucun des drones n'avait détecté de personne. Si l'un des drones détecte une personne, l'autre doit se rendre à sa position, comme illustré dans la figure suivante.



(a) Suivi de référence et rendez-vous

(b) Interruption de vol

**Figure 11 – Rendez-vous des drones**

Dans la figure de gauche, on remarque que lorsque le drone repère une personne et entre en phase de maintien en position statique, le maintien défectueux du drone entraîne un phénomène de dérive non corrigé par le contrôle.

Dans la figure de droite, nous avons une situation où un drone est parti avant l'autre, mais celui qui est parti ensuite a immédiatement repéré une personne. Par conséquent, il a interrompu son vol et l'autre drone est allé à sa rencontre, mettant en évidence le fonctionnement d'une autre caractéristique de notre projet : si une personne est repérée, interrompre le vol vers la référence et rester avec la personne détectée.

## 5 Conclusion et perspectives

En résumé, ce projet a représenté une opportunité précieuse pour mettre en œuvre et expérimenter diverses techniques apprises tout au long des cours à l'ENSTA, en particulier l'intégration avec ROS, qui s'est révélée être un outil extrêmement utile pour intégrer des modules initialement très différents et sans interface apparente.

Une extension évidente du projet consisterait à implémenter un algorithme de détection d'obstacles et de planification de trajectoire pour permettre l'exploration autonome d'un espace, similaire à ce qui se ferait naturellement dans l'une des applications mentionnées dans l'introduction. De plus, comme le montrent les résultats, il est impératif de renforcer la localisation du robot en fusionnant les données d'odométrie avec une seconde source d'information (potentiellement la caméra ou un autre capteur).

Enfin, en raison de contraintes de temps, le code n'est pas parfaitement optimisé et modularisé, ce qui pourrait améliorer les performances computationnelles du projet ainsi que sa capacité à évoluer en ajoutant de nouvelles fonctionnalités.

## Références

- [1] Akshat BAJPAI : Slampy : Monocular slam implementation in python, 2020. <https://github.com/Akbonline/SLAMPy-Monocular-SLAM-implementation-in-Python>.
- [2] Clément GODARD, Oisin MAC AODHA, Michael FIRMAN et Gabriel J. BROSSTOW : Digging into self-supervised monocular depth prediction. October 2019.
- [3] Glenn JOCHER, Ayush CHAURASIA et Jing QIU : Ultralytics yolov8, 2023. <https://github.com/ultralytics/ultralytics>.
- [4] Ebrahim KARAMI, Siva PRASAD et Mohamed S. SHEHATA : Image matching using sift, surf, BRIEF and ORB : performance comparison for distorted images. *CoRR*, abs/1710.02726, 2017.
- [5] Ivan LAZAREVICH, Matteo GRIMALDI, Ravish KUMAR, Saptarshi MITRA, Shahrukh KHAN et Sudhakar SAH : Yolobench : Benchmarking efficient object detectors on embedded systems. In *2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 1161–1170, 2023.
- [6] Hanya ZHOU : Dji tello drone controller python package, 2020. <https://github.com/hanyazou/TelloPy>.