



# Drone Autonome pour l'inspection des tunnels de métro

Rapport de Stage Stage Ingénieur (S10)

Rédigé par

**Jerónimo Pérez**

Etudiant 5eme année (S10) Ecole Nationale  
d'ingénieurs de Brest (ENIB)

Supervisé par  
**Vincent Kerohas**  
ENIB  
**Tarek Hemia**  
CDSI

# Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidées tout au long de cette expérience enrichissante. Tout d'abord, je remercie Maria Caprian (Directrice de la société) et François Arnoul (Directeur technique) pour me faire confiance dans le développement de ce projet. J'adresse mes remerciements particuliers aux mes tuteurs de stage, Tarek Hemia dans l'entreprise et Vincent Kerohas à l'ENIB.

Je tiens à remercier toute l'équipe chez CDSI pour leur gentillesse et la bonne ambiance. Je tiens également à remercier le personnel de l'ENIB ayant permis la mise en place de ce stage, en particulier Vincent Kerohas, mon tuteur de stage, Thierry LE MAGUERESSE (Directeur Relations Entreprises) et Chantal CALVES (Service Relations Entreprises).

# Résumé

Pour compléter ma formation comme ingénieur généraliste spécialisé en mécatronique dans l'École Nationale d'ingénieurs de Brest, école dans lequel j'ai continué mes études initiés en Argentine avec une bourse de double diplôme, j'ai effectué mon stage de fin d'études dans l'entreprise CDSI à Paris.

Crée en 2016, Centre Drone Systèmes industrielles est une société qui conçoit des systèmes-drones techniques sur mesure et travaille sur la R&D depuis 2003. Le travail sur lequel je me suis concentré c'est le développement d'un drone Autonome pour l'inspection des endroits fermes ou nous n'avons pas accès au réseau GPS et les conditions de vol peuvent ne pas être optimales (basse luminosité, poussière, courants d'air, etc). Ce drone doit aussi être capable de porter différentes charges utiles selon le type d'inspection nécessaire.

Le défi technique peut être divisé en trois parties, premièrement il nous faut une plate-forme robuste sur laquelle développer notre intelligence, cela veut dire un drone qui vole bien avec une autonomie et portance suffisante et une construction mécanique solide. Deuxièmement le drone doit avoir une référence de position globale pour pouvoir lui donner des commandes de vitesse ou position. Troisièmement il doit percevoir et comprendre son environnement pour générer une commande pertinente à chaque situation.

Cet rapport présente les travaux réalisés pour atteindre ces objectifs et les problématiques rencontrées. L'ordre utilisé ici n'est pas l'ordre chronologique étant donné que cela ne serait pas idéale pour répliquer les résultats obtenus .

# Abstract

To complete my training as a general engineer specialized in mechatronics in the National School of Engineers of Brest, school in which I continued my studies initiated in Argentina with a scholarship of double degree, I completed my graduate internship at the company CDSI in Paris.

Established in 2016, Centre Drone Industrial Systems is a company that designs custom technical drone systems and works on RD since 2003. The work I focused on is the development of an autonomous drone for the inspection of closed places where we do not have access to the GPS network and the flight conditions may not be optimal (low light, dust, drafts, etc). This drone must also be able to carry different payloads depending on the type of inspection required.

The technical challenge can be divided into three parts, first we need a robust platform on which to develop our intelligence, that means a drone that flies well with sufficient autonomy and lift and a solid mechanical construction. Secondly, the drone must have a global position reference to be able to give it speed or position controls. Third, it must perceive and understand its environment to generate a command relevant to each situation.

This report presents the work carried out to achieve these objectives and the problems encountered. The order used here is not chronological since it would not be ideal to replicate the obtained results .

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>10</b>
1.1	Structure générale d'un Drone . . . . .	10
1.1.1	Principales composants et logiciels que nous utilisons . . . . .	11
<b>2</b>	<b>Conception Mécanique</b>	<b>16</b>
2.1	Point de départ . . . . .	16
2.2	Premières modifications . . . . .	17
2.3	Un nouveau départ : reconception . . . . .	18
2.3.1	Impresion 3D CCF (Continuous Carbon Fiber) . . . . .	21
2.3.2	Boîtier pour la Jetson Nano . . . . .	22
2.4	Problèmes rencontrés . . . . .	22
<b>3</b>	<b>Navigation sans GPS</b>	<b>26</b>
3.1	Intel T265 . . . . .	26
3.1.1	Intégration avec Ardupilot . . . . .	26
3.1.2	Tests . . . . .	27
3.1.3	Conclusion . . . . .	29
3.1.4	Alternatives possibles . . . . .	30
3.2	RTAB-Map . . . . .	31
3.2.1	ROS TF . . . . .	31
3.2.2	Fermerture de boucle et correction de position globale . . . . .	35
3.2.3	Intégration avec Ardupilot . . . . .	36
3.2.4	Problèmes rencontrés . . . . .	39
3.2.5	Conclusion . . . . .	40
<b>4</b>	<b>Lidar</b>	<b>41</b>
4.1	VL53L1X . . . . .	41
4.1.1	Intégration sur le drone . . . . .	43
4.1.2	Intégration avec Ardupilot . . . . .	46
4.1.3	Tests . . . . .	48
<b>5</b>	<b>Mode Automatique</b>	<b>49</b>
5.1	SITL (Software In The Loop) . . . . .	49
<b>6</b>	<b>Machine Vision</b>	<b>52</b>
6.1	Gradient . . . . .	52
6.1.1	Démonstration de faisabilité . . . . .	53
6.1.2	Intégration avec la caméra D435 . . . . .	54
6.2	Détection d'obstacles et planification de trajectoire . . . . .	58

6.2.1	Détection d'espace libre . . . . .	58
<b>7</b>	<b>Banc d'essai</b>	<b>61</b>
7.1	Théorie . . . . .	61
7.1.1	Équations . . . . .	62
7.2	Conception . . . . .	62
7.2.1	Capteurs utilisés . . . . .	62
7.2.2	Conception 3D . . . . .	63
7.2.3	Schéma des connections électriques . . . . .	63
7.2.4	Produit final . . . . .	64
7.3	Essai de démonstration . . . . .	67
7.4	Problèmes rencontrés . . . . .	68
7.4.1	Temporisation . . . . .	68
7.4.2	Problèmes mécaniques . . . . .	69
<b>8</b>	<b>Conclusion</b>	<b>70</b>
8.1	Diagramme de Gantt . . . . .	71
8.2	Travail future . . . . .	71
8.3	Rapport d'émerveillement . . . . .	71
<b>A</b>	<b>Environment setup</b>	<b>77</b>
A.1	Real Sense SDK . . . . .	77
A.2	OpenCV + CUDA support . . . . .	78
A.3	ROS Melodic . . . . .	79
A.3.1	Configure a catkin workspace . . . . .	80
A.3.2	RealSense ROS . . . . .	80
A.3.3	MAVROS . . . . .	80
A.3.4	vision_to_mavros . . . . .	80
A.4	pymavlink . . . . .	81
A.4.1	RTAB-Map . . . . .	81
A.4.2	SITL . . . . .	81
A.5	Test environment . . . . .	82
A.5.1	ROS RealSense . . . . .	83
A.5.2	Serial communication . . . . .	83
A.5.3	TCP Telemetry bridge . . . . .	83
A.5.4	Vision_to_mavros . . . . .	83
A.5.5	pymavlink . . . . .	84
A.6	All nodes together . . . . .	84
A.7	RTAB-Map . . . . .	84
A.7.1	SITL . . . . .	84
<b>B</b>	<b>Optional steps</b>	<b>86</b>
B.1	On/Off button . . . . .	86
B.2	Remote desktop . . . . .	87
<b>C</b>	<b>Code</b>	<b>88</b>
C.1	Modified vision_to_mavros . . . . .	88
C.2	Lidar manager . . . . .	95
C.3	Lidar to Mavros . . . . .	97

C.4	Automatic flight . . . . .	98
C.4.1	Take Off and Landing . . . . .	98
C.4.2	Yaw control . . . . .	102
C.4.3	Keyboard Control . . . . .	106
C.5	Machine Vision . . . . .	110
C.5.1	Gradient . . . . .	110
C.5.2	Interpreter . . . . .	117
C.5.3	Évasion d'obstacles . . . . .	118

# Table des figures

1.1	Structure générale d'un Drone . . . . .	10
1.2	Piwhawk 4 (Holybro) [11] . . . . .	11
1.3	Moteur Sunnysky X2212 V3 . . . . .	12
1.4	Carte de développement Nvidia Jetson Nano . . . . .	12
1.5	VL53L1X breakout board . . . . .	13
1.6	Camera RealSense T265 . . . . .	13
1.7	Camera RealSense D435 . . . . .	14
1.8	Mission Planner GCS . . . . .	15
2.1	Premier prototype . . . . .	16
2.2	Premier prototype . . . . .	17
2.3	Inspector V2 . . . . .	18
2.4	Inspector V3 render . . . . .	19
2.5	Inspector V3 render . . . . .	19
2.6	Inspector V3 render vue explosé . . . . .	20
2.7	Inspector render bras vue explosé . . . . .	20
2.8	Illustration des pièces imprimées en 3D . . . . .	21
2.10	Troues avec le cuivre exposé . . . . .	23
2.11	Fixations des bras . . . . .	24
2.12	État du drone après la défaillance mécanique . . . . .	25
3.1	Flux des données [21] . . . . .	27
3.2	Transformations matricielles [21] . . . . .	27
3.3	Test d'échelle . . . . .	28
3.4	Test en mode Loiter . . . . .	28
3.5	Problèmes de mesure . . . . .	29
3.6	Message du support d'Intel dans le GitHub de l'API RealSense . . . . .	30
3.7	Arbre des systèmes de référence T265 et Mavros . . . . .	31
3.8	Arbre des systèmes de référence T265, D435 et Mavros . . . . .	32
3.9	Configuration matérielle et cadres de coordonnées [13] . . . . .	32
3.10	Support D435 et T265 . . . . .	33
3.11	T265 et D435 reliés via la transformation statique . . . . .	34
3.12	Arbre de TF final . . . . .	35
3.13	Visualisation de DB avec reconnaissance de points caractéristiques (Tunnel de métro) . . . . .	36
3.14	Trajectoires de T265 (vert) et RTAB-Map (bleue) avec décalage. . . . .	37
3.15	Trajectoires recalées de T265 (vert) et RTAB-Map (bleue). . . . .	38
3.16	Trajectoire avec reconstruction 3D de l'environnement. . . . .	38
3.17	Correction de dérive via grâce à fermeture de boucle. . . . .	39

4.1	Capteur SMT VL53L1X . . . . .	41
4.2	Breakout Board utilisé . . . . .	42
4.3	Distribution des capteurs . . . . .	43
4.4	Schéma de connections pour les capteurs LIDAR . . . . .	44
4.5	Résistances a enlever . . . . .	45
4.6	Capteurs LIDAR avec support et connecteur JST-GH . . . . .	45
4.7	Prototype du faisceau électrique . . . . .	46
4.8	Vérification de réception de message via Mavlink . . . . .	47
4.9	Visualisation des capteurs de distance sur Mission Planner . . . . .	48
5.1	Architecture du simulateur [6] . . . . .	50
5.2	Exemple de simulation [6] . . . . .	50
6.1	Alternatives de visualisation . . . . .	52
6.2	Idée générale pour la détection du point de fuite . . . . .	53
6.3	Test dans un cas plus complexe . . . . .	54
6.4	Importance du pré-traitement de l'image . . . . .	55
6.5	Pipeline de l'algorithme du gradient. . . . .	55
6.6	Visualisation de l'algorithme dans un couloir . . . . .	56
6.7	Détection de bifurcation . . . . .	57
6.8	Simplification du comportement attendu . . . . .	57
6.9	Détection de chemin libre et obstacles à 1m . . . . .	59
6.10	Détection de chemin libre et obstacles à 1m . . . . .	59
6.11	Comportement attendu . . . . .	59
6.12	Comportement attendu . . . . .	60
7.1	Flux énergétique du système de puissance . . . . .	61
7.2	Conception 3D . . . . .	63
7.3	Schéma des connections électriques . . . . .	64
7.4	Banc d'essais . . . . .	64
7.5	Banc d'essais . . . . .	65
7.6	Contrôleur . . . . .	65
7.7	Interface du contrôleur . . . . .	66
7.8	Relation linéaire entre la consigne PWM et la vitesse . . . . .	67
7.9	Rendement général . . . . .	68
7.10	Rendement de l'hélice . . . . .	68
7.11	Problèmes mécaniques . . . . .	69
8.1	Distribution temporelle des travaux réalisés . . . . .	71

# Chapitre 1

## Introduction

### 1.1 Structure générale d'un Drone

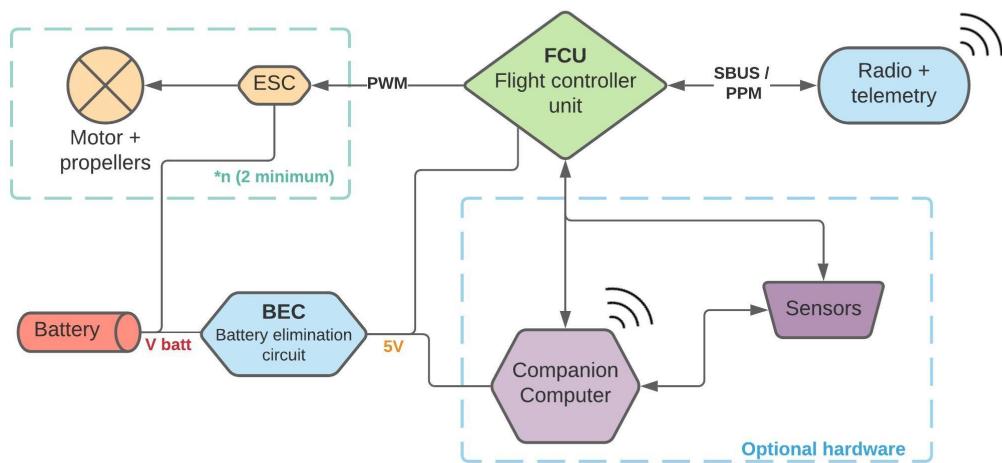


FIGURE 1.1 – Structure générale d'un Drone

### 1.1.1 Principales composants et logiciels que nous utilisons

Composants :

— **FCU** : Pixhawk 4

Cet contrôleur de vol open source est une des meilleures plateformes de développement des drones actuels grâce à ça flexibilité, ces nombreux ports de communication et facilité d'intégration sur des projets robotiques. Entre les composants utilisés dans tous les contrôleurs de vols nous trouvons un ou plusieurs Inertial Measurement Unit (IMU), magnétomètres et baromètres.

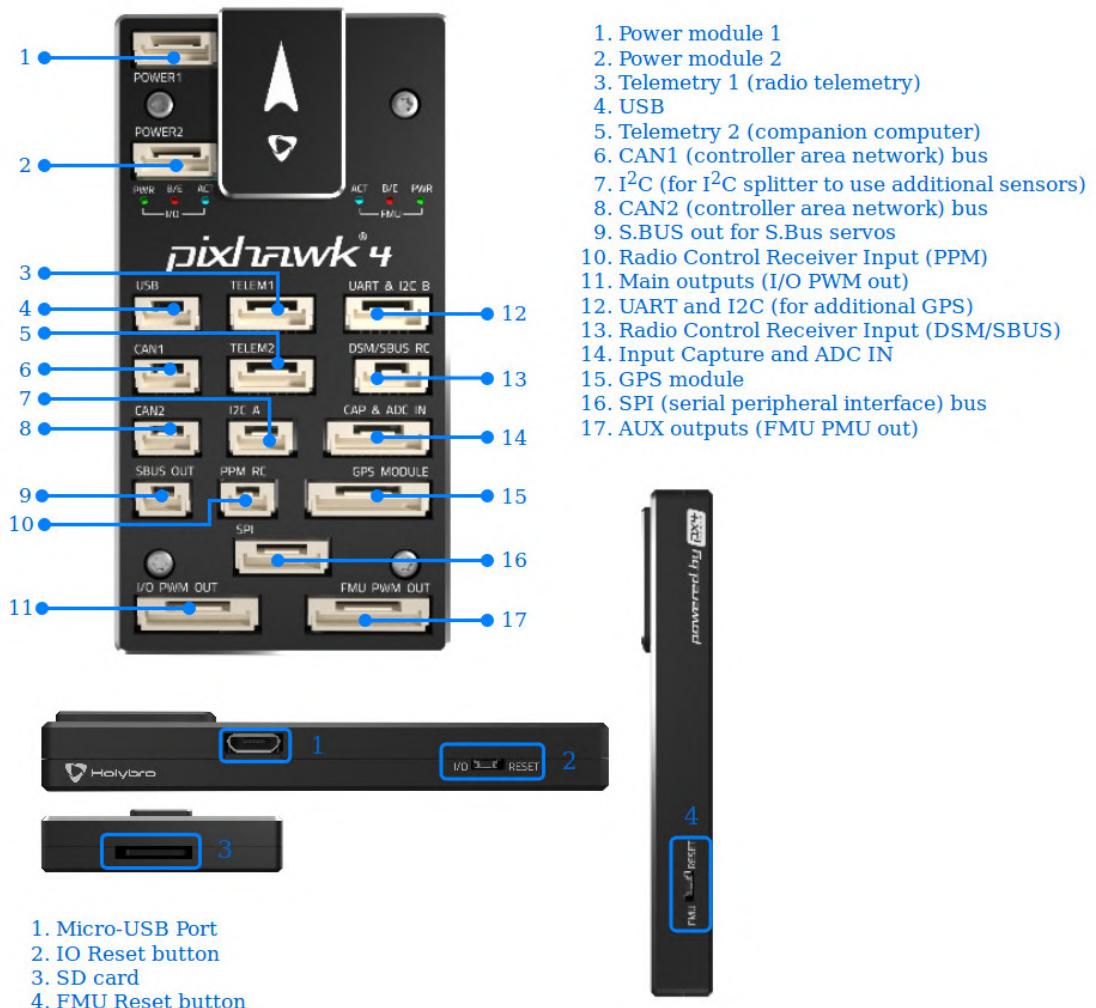


FIGURE 1.2 – Piwhawk 4 (Holybro) [11]

— **ESC** : Hobbyking 20 A Il génère la signal triphasé pour contrôler le moteur brushless.

— **Moteurs** : Sunnysky X2212 V3 980 kV

Nous avons choisi ces moteurs en raison de la qualité mécanique et le haut rendement énergétique.<sup>1</sup>



FIGURE 1.3 – Moteur Sunnysky X2212 V3

— **Hélices** :

Pour avoir un meilleur rapport surface/portance nous avons choisi d'utiliser des hélices à 3 pales, cela nous permet d'utiliser hélices de 8 puces on lieu de 9, en sacrifiant un peu de rendement à cause d'une majeure traînée aérodynamique.<sup>2</sup>

— **Companion Computer** : Nvidia Jetson Nano

Cet ordinateur accompagnant est en charge de toutes les opérations "intelligentes" grâce à la puissance de calcul donné par son GPU. Il utilise comme système d'exploitation une version modifiée d'Ubuntu Linux et il se communique avec le FCU parmi port série en utilisant le protocole *Mavlink* (1.1.1)



FIGURE 1.4 – Carte de développement Nvidia Jetson Nano

1. Pour l'instant nous faisons confiance aux valeurs publiées par le fabricant mais des tests avec le banc de test présenté dans le chapitre 7 sont prévues.

2. Il faut vérifier la performance des hélices avec le banc de test présenté dans le chapitre 7.

— **Capteurs :**

— **VL53L1X** : Time-of-Flight Lidar

Ce capteur développé par STMicroelectronics a été choisi grâce à ses dimensions et ça plage de mesures (0.04m - 4m). Il se communique par bus I2C.



FIGURE 1.5 – VL53L1X breakout board

— **Realsense T265** : Tracking Camera

Cette caméra stéréo nous donne une estimation de ça position et orientation dans l'espace (6 DoF) grâce aux algorithmes de SLAM embarqués. Nous rentrerons en détail sur cette caméra dans la section 3.1



FIGURE 1.6 – Camera RealSense T265

— **Realsense D435** : Depth Camera

Cette caméra stéréo nous donne une image RGB ainsi que une nuage de points, pouvant être fusionnés pour générer une image RGBD (couleur + profondeur pour chaque pixel).



FIGURE 1.7 – Camera RealSense D435

**Firmware et logiciels :**

— **Ardupilot** : Firmware open source pour le contrôleur de vol

ArduPilot est une suite logicielle de pilotage automatique de véhicules sans pilote open source capable de contrôler de manière autonome Drones multirrotors, Aéronefs à voilure fixe et VTOL, Hélicoptères, Rovers, etc. Nous utilisons ce firmare parce-qu'il a une communauté énorme qui développe et contribue à sa amélioration.

— **Mission Planner**Ground Control Station (GCS) :

Ce logiciel nous permet de régler tous les paramètres du contrôleur, calibrations, création des missions automatiques et certaines fonctionnalités de télécommande et surveillance de vol.

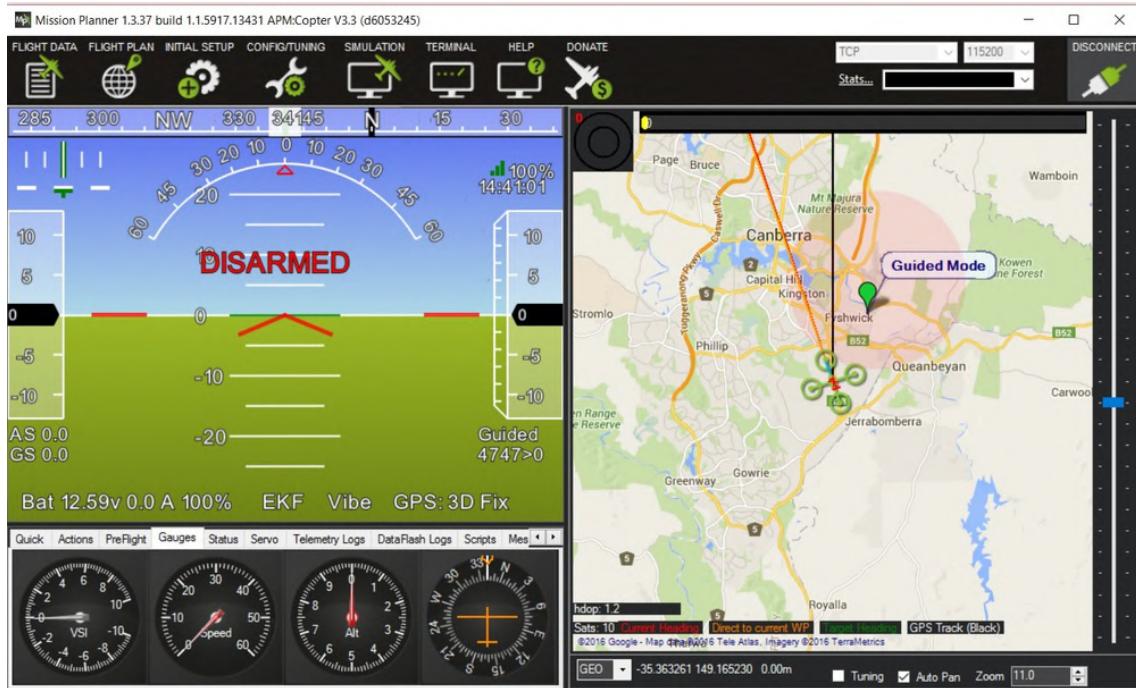


FIGURE 1.8 – Mission Planner GCS

- **Micro Air Vehicle Link (Mavlink)** : Protocole pour communiquer avec des petits véhicules inhabités. Il est principalement utilisé pour la communication entre un poste de contrôle au sol (GCS) et des véhicules sans pilote, et dans l’inter-communication entre sous-systèmes du véhicule. Il peut être utilisé pour transmettre l’orientation du véhicule, sa position GPS et sa vitesse. Un contrôle de redondance cyclique est aussi implémenté permettant de détecter les erreurs de communication.

# Chapitre 2

## Conception Mécanique

### 2.1 Point de départ

Au moment de commencer ce projet, un drone avait déjà été construit avec une configuration classique QuadRotor, hélices de 9 puces et moteurs génériques. Nous avons fait des adaptations qui nous ont permis de monter les composants nécessaires.



FIGURE 2.1 – Premier prototype



FIGURE 2.2 – Premier prototype

## 2.2 Premières modifications

Suite aux premiers essais du système de positionnement visuel nous avons constaté que avec la configuration originale et le poids rajouté nous avions un temps de vol trop court d'environ 7 minutes, c'est pour cela que nous avons choisi de passer à une configuration OctaQuad, ceci implique de rajouter 4 moteurs, en ayant ainsi 2 moteurs contrarotatifs par bras.

Cette modification nous permet d'avoir un incrément estimé dans la capacité de portée d'un %60 [2], qui nous permet d'installer une deuxième batterie et augmenter le temps de vol même si le rendement énergétique total est mineur. Vue que nous devions désassembler le drone, nous avons profité pour faire des nouvelles protections légères construits avec des tiges de fibre de carbone et pièces imprimés en 3D.



FIGURE 2.3 – Inspector V2

## 2.3 Un nouveau départ : reconception

Suite à une succession de défaillances catastrophiques voir figure 2.12 section 2.4, une reconception totale du drone a été mise en place pour le rendre plus robuste mécaniquement et pour optimiser son câblage. La nouvelle version a, été conçue entièrement en 3D avant sa construction pour éviter les surprises et inconvénients de l'ancienne méthode de développement "au fur et à mesure".

La configuration mécanique est similaire mais sur cette nouvelle version nous utilisons une plaque de distribution de puissance développé par le fabricant du contrôleur de vol, et pour la structure, des plaques de fibre de carbone et des pièces imprimés en 3D.

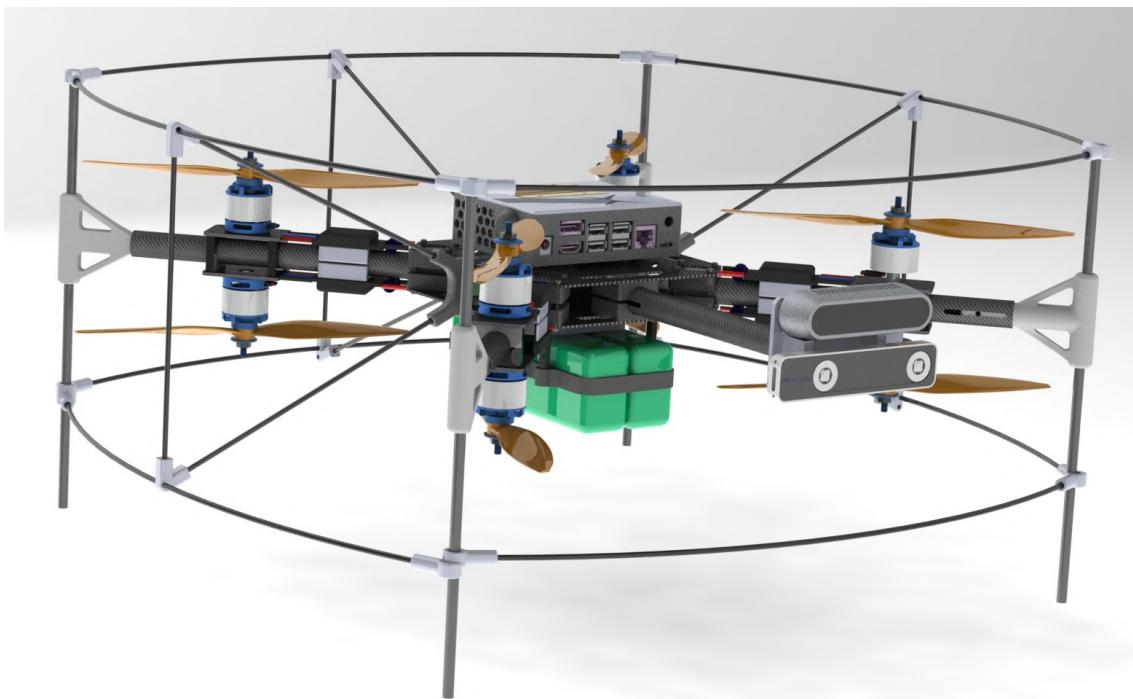


FIGURE 2.4 – Inspector V3 render

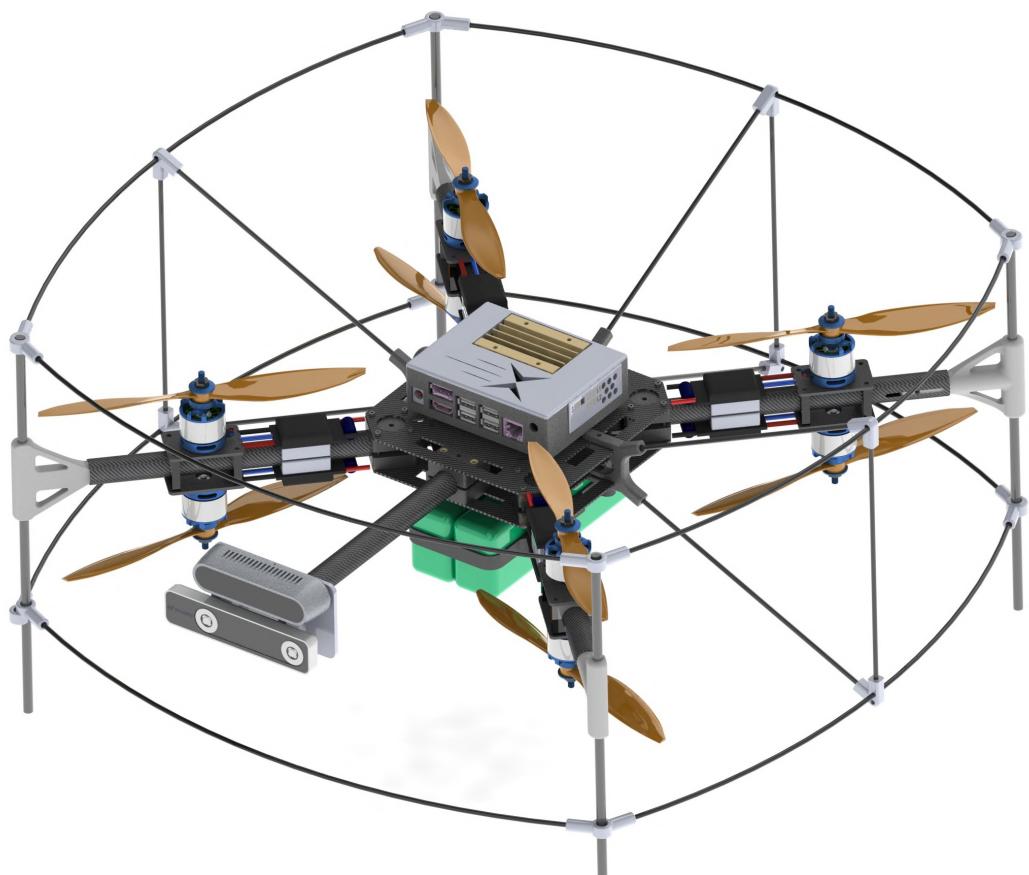


FIGURE 2.5 – Inspector V3 render

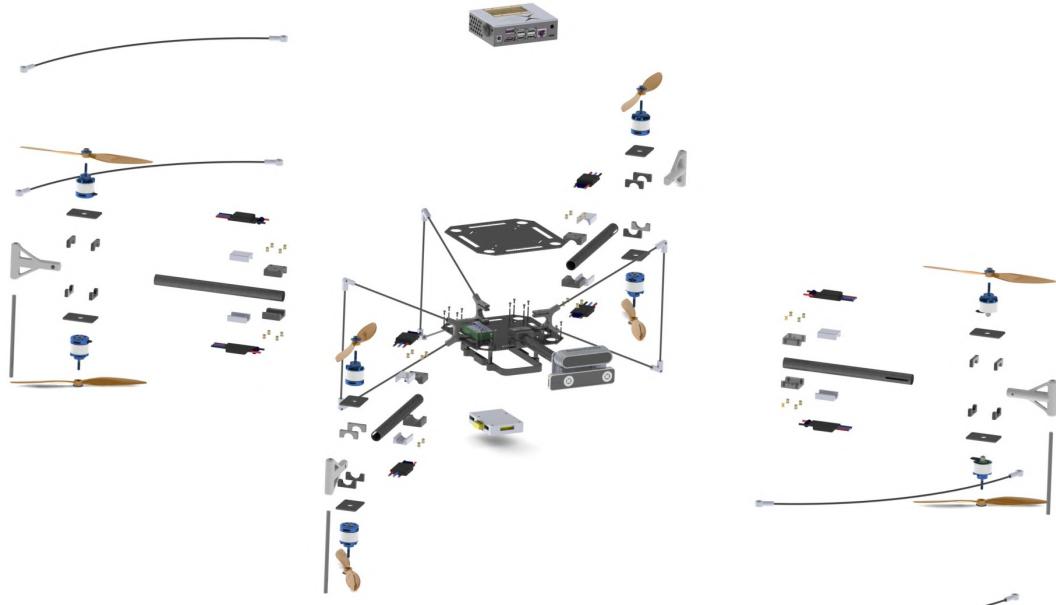


FIGURE 2.6 – Inspector V3 render vue explosé

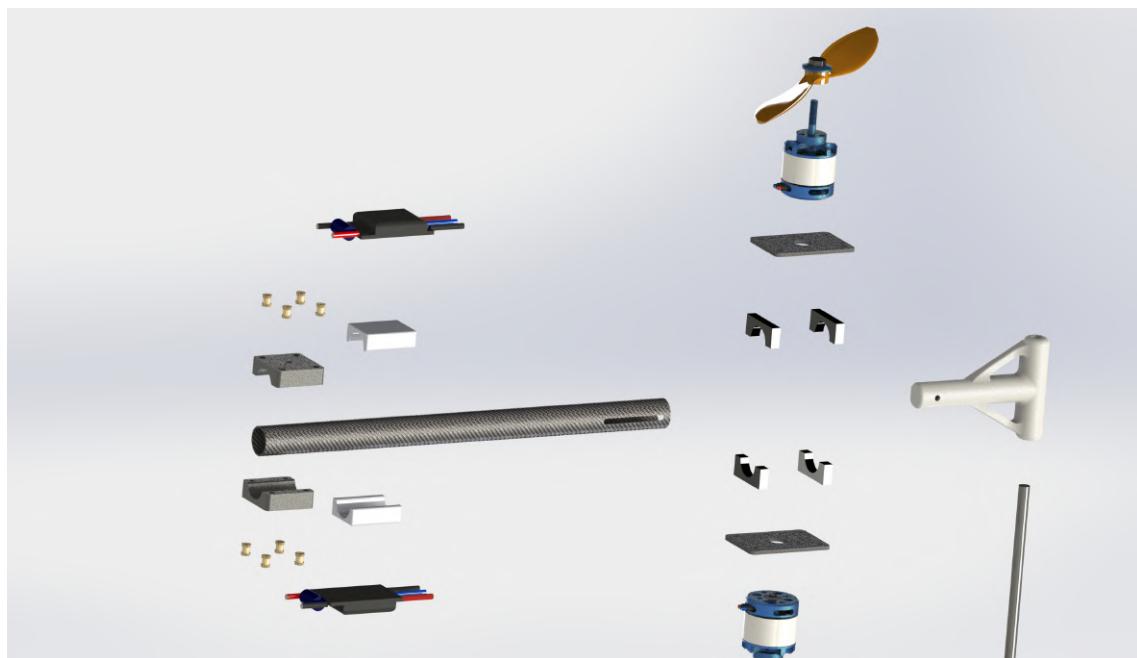
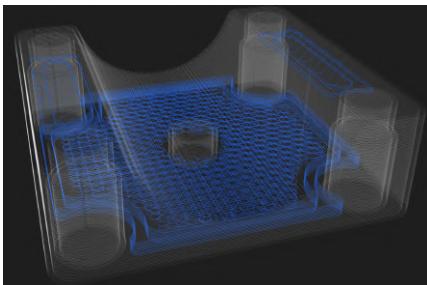


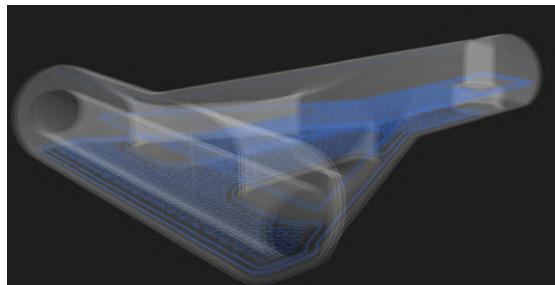
FIGURE 2.7 – Inspector render bras vue explosé

### 2.3.1 Impression 3D CCF (Continuous Carbon Fiber)

La conception de pièces pour imprimer en 3D a été optimisé pour la technologie CCF implémenté par Markforged® dans ses imprimantes 3D. Cette technologie nous permet d'intégrer des couches de fibre de carbone, Kevlar ou verre dans une matrice de nylon renforcé avec des fragments microscopiques de fibre de carbone.



(a) Fixation des bras et renforcements de fibre (bleu)



(b) Fixation des pieds et renforcements de fibre (bleu)



(c) Fixation des bras avec inserts filetés



(d) Fixations des pieds

FIGURE 2.8 – Illustration des pièces imprimées en 3D

### 2.3.2 Boîtier pour la Jetson Nano

Nous avons conçu un boîtier pour protéger l'ordinateur, rendre ses ports de communication plus accessibles et faciliter son montage sur le drone étant donné que pour le développement des logiciels c'est plus pratique de la démonter.



(a) Jetson Nano



(b) Connecteurs JST-GH

Nous avons un accès au port série pour la connexion via Mavlink avec le contrôleur de vol Pixhawk et deux connecteurs pour les capteurs Lidar exposés dans la section 4. Les connecteurs JST-GH sont utilisés dans la Pixhawk et dans plusieurs capteurs dans le marché.

## 2.4 Problèmes rencontrés

Nous avons rencontré deux problèmes majeurs avec la structure originale du drone, premièrement la plaque principale sur laquelle tous les composants sont fixés été une plaque PCB avec distribution de puissance intégrée, cela veut dire qu'il y a deux couches de cuivre isolées avec des points de soudure qui permettent de connecter la batterie batterie aux ESC avec moins des câbles.

Normalement ce dernier point est une avantage à l'heure de monter le drone, sauf que les trous originales (et isolés) que nous trouvons sur la plaque ne permettaient pas de fixer les bras avec les composants disponibles, c'est pourquoi la plaque a été percée en laissant exposé le cuivre connecté à la batterie (16 V) provoquant des court-circuits entre différents composants du drone. Il faut remarquer aussi que les plaques et tubes de fibre de carbone utilisés pour la construction conduisent l'électricité .

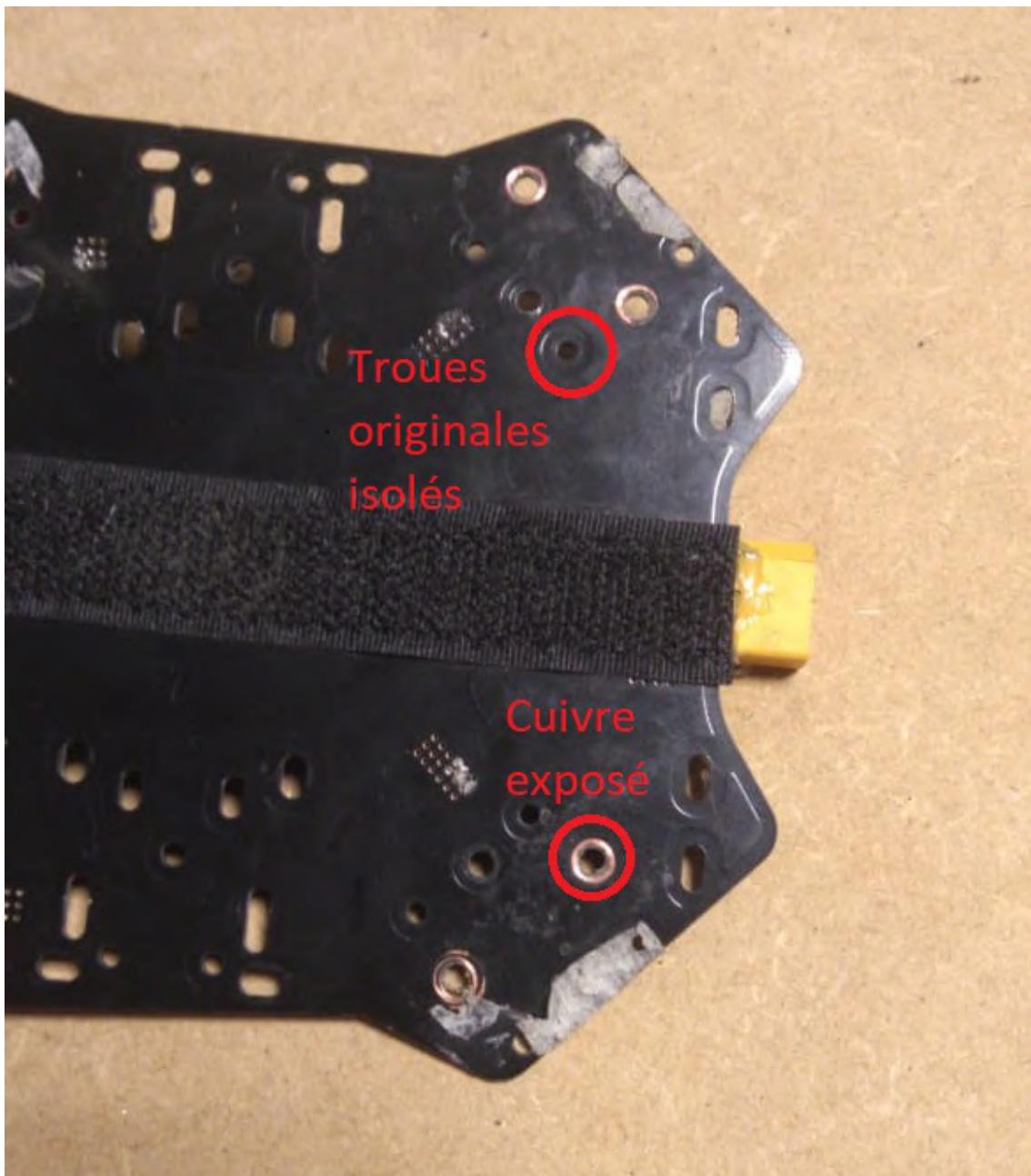


FIGURE 2.10 – Troues avec le cuivre exposé

Le deuxième problème a été au niveau des fixations des bras du drone :



FIGURE 2.11 – Fixations des bras

Les pièces en aluminium qui serrent ne sont pas suffisantes pour tenir les bras qui sont tirés vers l'extérieur par les protections élastiques, cela a provoqué un désassemblage en vol (RUD) pendant les tests du système de positionnement.



FIGURE 2.12 – État du drone après la défaillance mécanique

# Chapitre 3

## Navigation sans GPS

Ardupilot nous permet d'utiliser plusieurs modes de vol, mais nous allons nous intéresser sur trois modes principales :

- **Altitude Hold (AltHold)** : Ce mode nous permet de stabiliser le drone en altitude grâce au baromètre intégré et en tangage, roulis et lacet grâce à l'IMU. Le drone restera stable mais sans pouvoir maintenir sa position car il ne la connaît pas.
- **GPS (Loiter)** : Ce mode rajoute aux fonctionnalités du mode AltHold le contrôle de position, en permettant de corriger la dérive provoqué par des perturbation externes.
- **Guided** : Ce mode rajoute au mode GPS une couche d'automatisation en permettant donner des ordres de déplacement en position, avec contrôle de vitesse et temporisations.

Le problème avec les drones en intérieur c'est que les modes qui se servent du GPS ne sont pas disponibles donc nous ne pouvons pas donner des ordres de déplacement en position et le drone n'a aucun moyen de connaître sa position globale.

Dans ce chapitre nous allons voir comment remplacer la source de position globale (GPS) par un système visuel-inertiel qui nous permet de connaître notre position par rapport au point de démarrage.

### 3.1 Intel T265

Comme nous avons dit dans l'introduction, cette caméra nous donne une estimation de la position et orientation (6 DOF) et tous les calculs nécessaires pour le V-SLAM sont faites dans le processeur embarqué Intel® Movidius™ Myriad™ 2 VPU. L'API RealSense est disponible sur les systèmes d'exploitation Linux, Windows et Android et nous avons besoin d'un d'entre eux pour faire l'interface avec le contrôleur de vol.

#### 3.1.1 Intégration avec Ardupilot

Pour envoyer l'estimation de position de la caméra vers le FCU nous avons suivi une série des tutoriels sur le blog d'Ardupilot [21] où nous trouvons deux alternatives, une mise en oeuvre sur Robot Operating System (ROS) en utilisant le module Mavros ou directement avec un script python et pyMavlink. Nous avons choisi d'utiliser ROS parce que c'est une plateforme de développement modulaire

et robuste, avec une communauté énorme qui contribue à son développement. Pour une meilleure compréhension de cette section nous recommandons fortement d'avoir une connaissance basique du fonctionnement de ROS.

La mise en place de l'environnement de travail sur Jetpack (Ubuntu Linux modifié par Nvidia) est expliquée dans l'annexe A. Nous avons créé aussi une image du système d'exploitation avec tous les modules et logiciels déjà installés prête à être gravé dans une carte SD.

L'intégration consiste d'un programme (Node ROS), appelé ***vision\_to\_mavros*** qui lit les valeurs d'odométrie depuis le topic ROS correspondant, il fait la transformation du système de référence de la caméra vers le système de référence utilisé par le FCU parmi le module ROS TF de transformations matricielles et il les envoie vers le FCU via un message Mavlink. Ce node publie aussi un topic `body_frame\path` avec la trajectoire parcourue par le drone pour la visualiser dans Rviz.

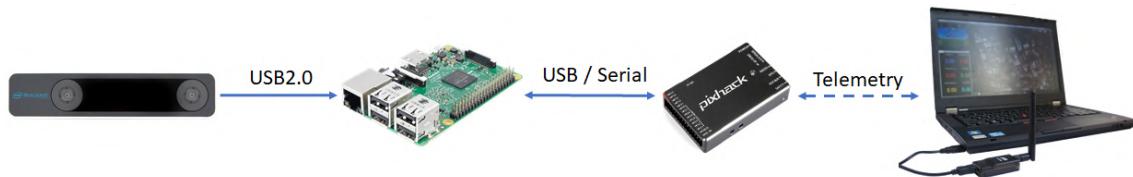


FIGURE 3.1 – Flux des données [21]

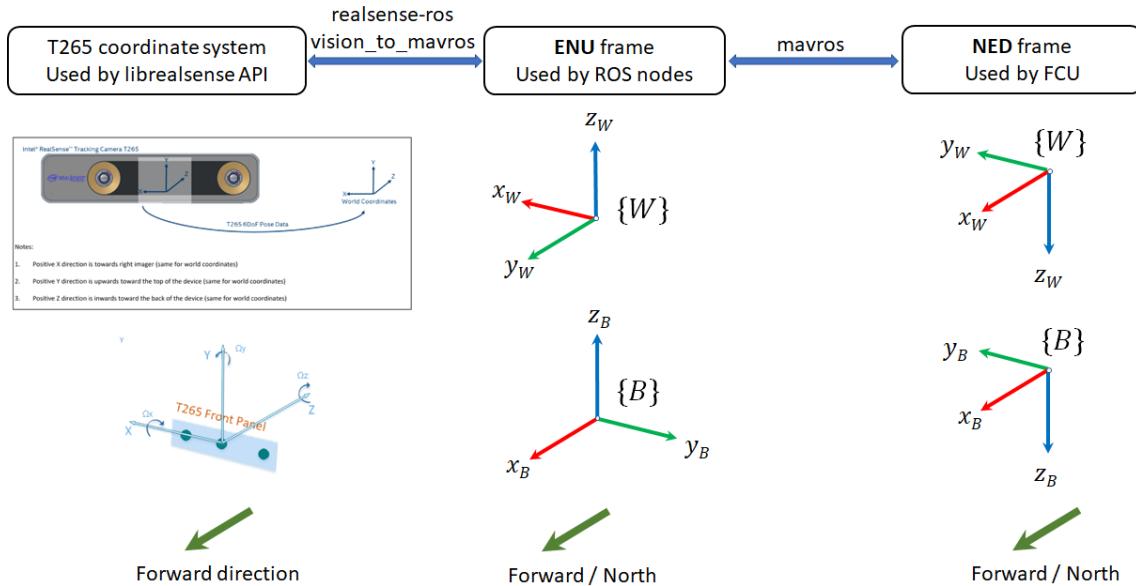


FIGURE 3.2 – Transformations matricielles [21]

### 3.1.2 Tests

#### Procédure de test

Dans une première étape nous avons suivi les test de calibration d'échelle recommandés dans le tutoriel [4] pour vérifier les mesures de position. Nous avons placé des marques à 4 mètres de distance et nous avons fait plusieurs vols en mode manuel (AltHold) pour vérifier si la caméra mesure correctement le déplacement.

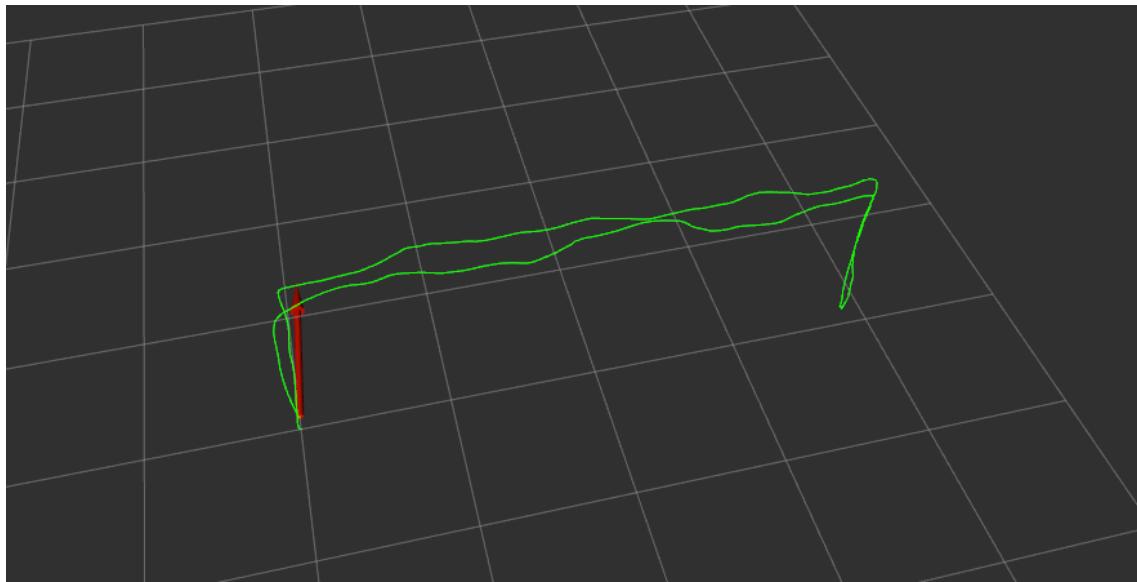


FIGURE 3.3 – Test d'échelle

Nous pouvons voir dans la figure 8.1 que le déplacement mesuré correspond à la vrai valeur de déplacement (Chaque section dans la grillé à 1m de longueur).

En suite nous avons testé la performance en mode Loiter, où le drone doit corriger sa position face aux perturbations externes.

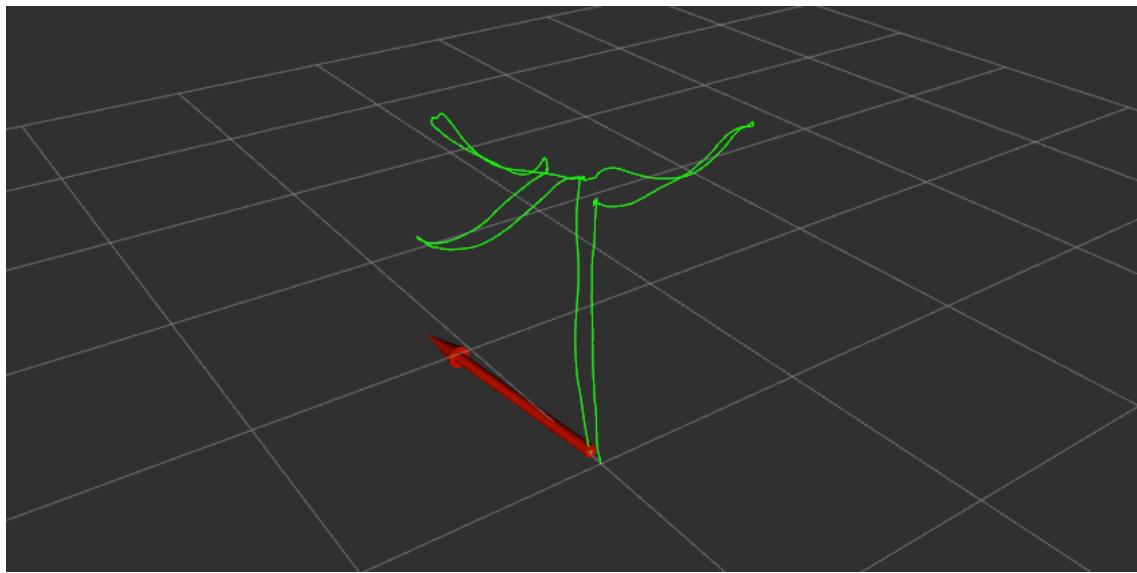


FIGURE 3.4 – Test en mode Loiter

Dans ce cas nous avons décollé et déplacé le drone en mode Loiter plusieurs fois dans des environnements différents en réalisent des trajectoires de complexité variée pour vérifier si le drone perds son repère ou s'il y a une dérive dans la mesure. La performance générale est bonne pendant les premières minutes de vol mais après nous commençons a voir des erreurs dans l'estimation.

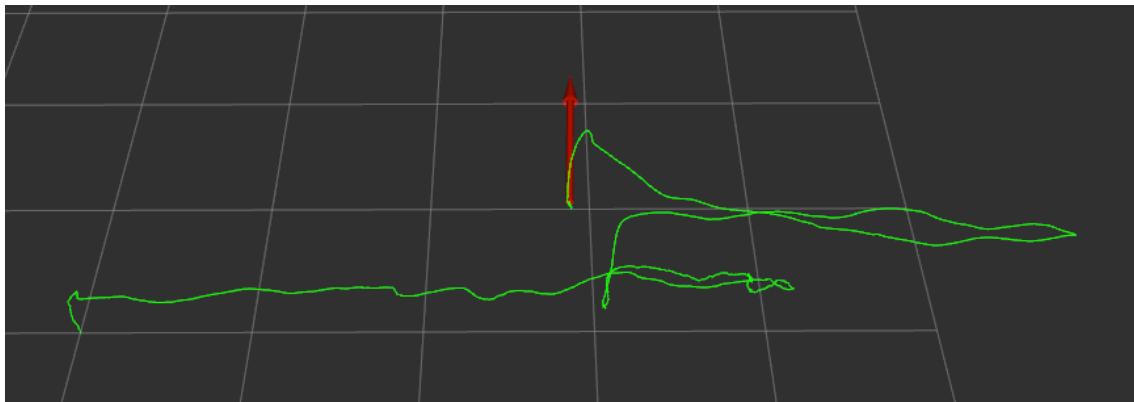


FIGURE 3.5 – Problèmes de mesure

Dans la figure 3.5 le drone est rentré au point de décollage mais la trajectoire mesuré montre un décalage d'environ 3 mètres. Ce comportement peut être causé par une mauvaise estimation de vitesse de déplacement ce qui génère une mesure plus courte que la vrai valeur.

Dans autres tests, la caméra croyait qu'elle était en train de se déplacer quand en vrai le drone restait en vol statique. Cette mesure faussée génère une correction de la part du drone qui n'est pas nécessaire et il commence à se déplacer sans jamais attendre sa consigne de position en obligeant au pilote à reprendre le contrôle pour éviter un accident.

### 3.1.3 Conclusion

Nous avons fait nombreux tests du système dans différentes conditions de luminosité en extérieur et en intérieur et nous avons trouvé que quand tout marche bien, la précision du positionnement est remarquable avec un erreur de quelques centimètres, en donnant une réponse aux perturbations en vol très bonne. Malheureusement nous avons trouvé aussi quelques modes de défaillance qui rendent le système très imprévisible et pourtant, pas assez robuste pour notre application.

- **Dérive :** Dans certaines situations, l'algorithme de v-slam peut se perdre et envoyer des valeurs faussées de position à le FCU, en disant par exemple que le drone est en train de bouger quand en réalité il n'y a pas des déplacements. Le FCU essaiera de corriger sa position en donnant une ordre de vitesse de translation et le drone bougera imprévisiblement et nous avons aucun moyen de détecter cette situation sans un observateur externe. Les causes de cet erreur ne sont pas claires car cette caméra fonctionne comme une boîte noir, nous avons accès aux valeurs de sortie et quelques configurations mais pas aux algorithmes utilisés pour l'estimation.
- **NaN :** D'une façon similaire à la dérive, dans certaines situations apparemment aléatoires la caméra peut envoyer des valeurs "Not a Number" (*NaN*) à le FCU. Dans la dernière version stable d'Ardupilot au moment de rédiger ce rapport (4.0.5), cet erreur provoque un comportement imprévisible et sou-

vent la perte du véhicule à cause de l'impact contre un mur. Dans la version beta 4.1.0-dev plusieurs améliorations ont été implémentées pour avoir une meilleure intégration de cette caméra, entre elles le filtrage des valeurs NaN dans la nouvelle version du filtrage de Kalman utilisé (EKF3). Ce filtrage nous permet de éviter les déplacements imprévisibles mais nous perdons toujours le positionnement. Ce problème est reconnu par Intel mais dans plusieurs "Issues" sur le GitHub de RealSense ils sont déclarés qu'il ne sera pas abordé car ils concentreront leurs efforts sur la nouvelle génération de produits.

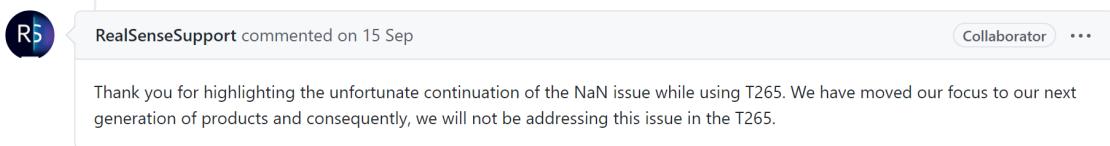


FIGURE 3.6 – Message du support d’Intel dans le GitHub de l’API RealSense

En raison de ces problèmes, nous ne pouvons pas utiliser cette caméra comme source de position unique et nous sommes obligés de analyser différentes alternatives.

### 3.1.4 Alternatives possibles

- **Détection de dérive et réinitialisation :**  
Comparaison de l'estimation donnée par la caméra avec les valeurs d'accélération mesurées par le FCU pour détecter quand il y a une dérive et réinitialiser la caméra. Les problèmes avec ce rapprochement sont la difficulté de cette détection et la perte de continuité dans le positionnement global. Nous aurions aussi quelques secondes entre la détection de la dérive et le redémarrage de la caméra sans source de position.
- **Fusion de plusieurs T265 :**  
Comparaison de différentes sources de position pour la détection de dérive. Si nous détectons une variation grande dans une caméra nous pouvons l'ignorer et continuer avec l'estimation d'une autre ou de la fusion de données fait avec un filtre de Kalman. L'inconvénient avec cet approche c'est que l'API ROS RealSense ne supporte pas plus d'une T265 connecté, il faudrait développer une implémentation de l'API personnalisé.
- **Fusion de multiples capteurs :**  
Fusion de l'estimation de position différents capteurs (IMU, T265, RGBD SLAM) parmi un filtre de Kalman et une stratégie de commande en bas niveau pour la détection de dérive. La problématique ici c'est le temps de développement.
- **Intégration avec RTAB-Map :**  
Rtabmap nous permet d'utiliser l'estimation de position de la T265 comme source d'odométrie et l'image RGBD générée par la D435 pour l'algorithme de RGBD SLAM, qui nous permet de faire la cartographie 3D avec la reconstruction des nuages de points et la fermeture de boucle grâce à la reconnaissance des points caractéristiques de l'image. Un des possibles obstacles pour ça mise en œuvre c'est la puissance de calculs requise. Les instructions d'installation sont disponibles dans l'annexe A.4.1

## 3.2 RTAB-Map

Nous nous sommes intéressés par RTAB-Map à cause de ça capacité de faire le RGBD SLAM avec la caméra D435, qui nous permet de faire la correction de position parmi la fermeture de boucle grâce à leur système "bag-of-words". Ce système reconnaît des points caractéristiques dans l'image et les associe avec une position dans la carte. RTAB-Map et nous donne aussi la possibilité de faire l'estimation de position en utilisant seulement la D435 au prix d'un complexité de calcul plus élevée.

Pour mieux comprendre le fonctionnement de la fermeture de boucle il faut avant comprendre la gestion des systèmes de coordonnées utilisé par ROS et le module TF.

### 3.2.1 ROS TF

TF est un paquet qui permet à l'utilisateur de suivre plusieurs systèmes de coordonnées en maintenant la relation entre eux dans une arborescence tamponnée dans le temps. Il permet à l'utilisateur de transformer des points, des vecteurs, etc entre deux systèmes de coordonnées à n'importe quel moment désiré.

Par exemple, nous pouvons voir l'arbre de TF généré quand nous lançons le node ***vision\_to\_mavros*** avec la commande suivante :

```
1 $ rosrun tf view_frames
```

L'arbre est enregistré dans un PDF dans le dossier où nous avons lancé la commande.

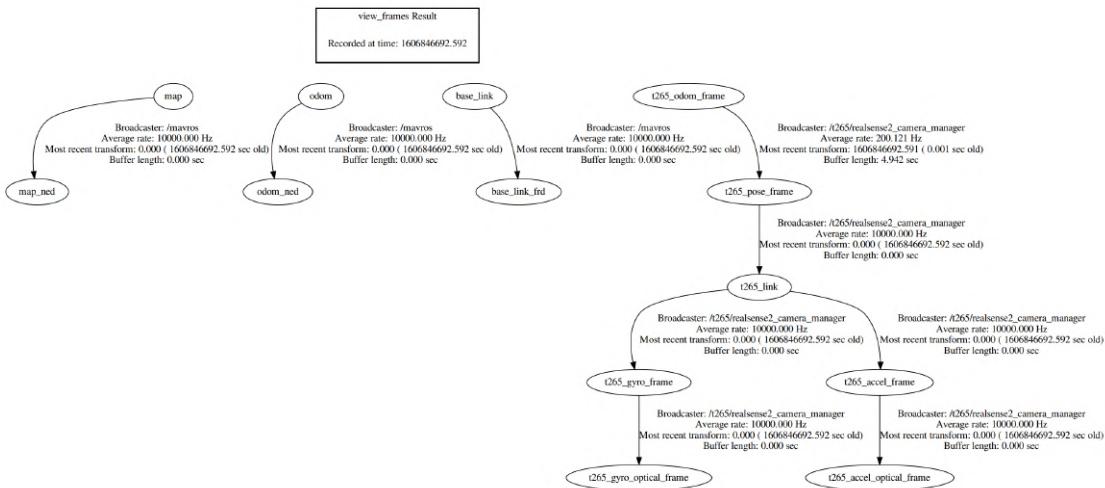


FIGURE 3.7 – Arbre des systèmes de référence T265 et Mavros

Nous pouvons voir ici que nous avons plusieurs arbres publiés par différentes sources. Idéalement nous devrions avoir un seul arbre avec des nœuds reliés par des transformations qui peuvent être statiques ou dynamiques (qui changent dans le temps).

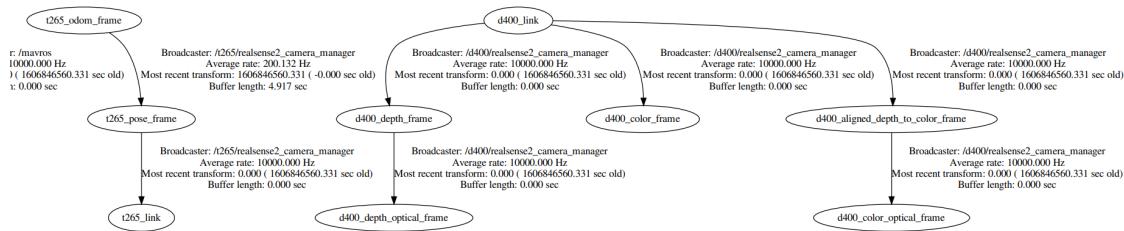
Maintenant nous pouvons modifier le launch file du node ***vision\_to\_mavros*** pour lancer les deux caméras au même temps en changeant la ligne :

```
1 <include file="$(find realsense2_camera)/launch/rs_t265.launch"
" />
```

pour :

```
1 <include file="$(find realsense2_camera)/launch/
rs_d400_and_t265.launch"/>
```

Nous pouvons maintenant voir une section du nouveau arbre de TF :



Nous constatons que nous avons deux arbres séparés pour chaque caméra, mais RTAB-Map à besoin de connaître la relation physique entre la source d'odométrie (T265) et la source RGBD (D435) pour faire la reconstruction de la carte 3D donc nous avons besoin de relier ces deux arbres parmi un transformation matricielle statique.

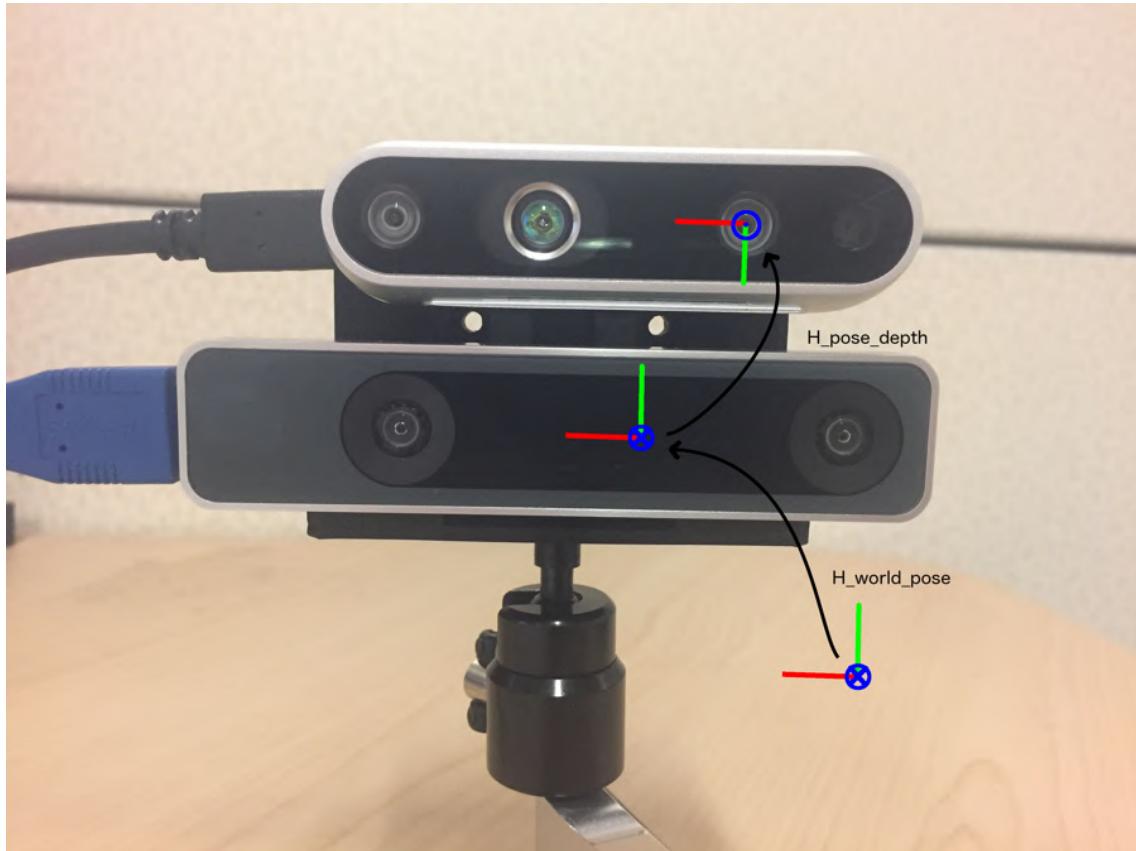


FIGURE 3.9 – Configuration matérielle et cadres de coordonnées [13]

Pour connaître la transformation nécessaire exacte, nous avons désigné le support pour les caméras en utilisant comme guide le support fourni dans le GitHub de librealsense [13] ou nous trouvons aussi la matrice de transformation qui lui correspond.

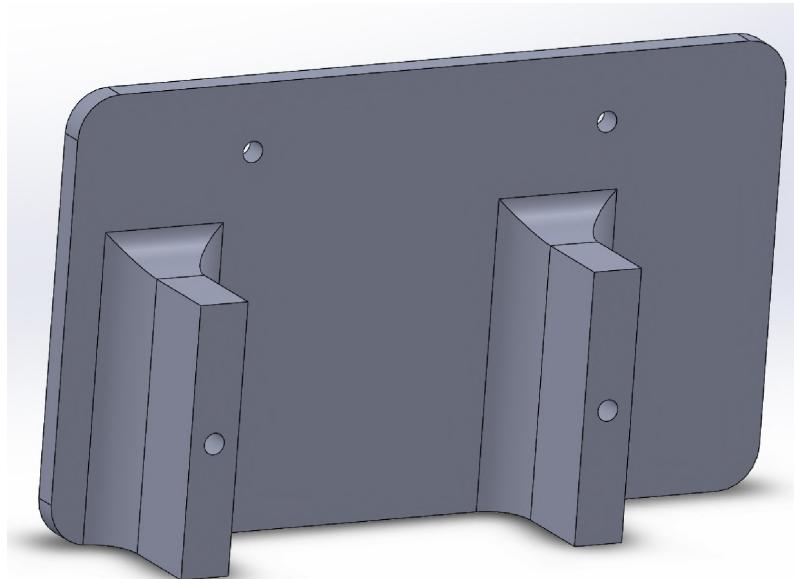


FIGURE 3.10 – Support D435 et T265

La transformation statique peut être rajouté grâce à la fonction *static transform publisher* du module TF. Il suffit de rajouter la ligne suivante dans le launch file *rs\_d400\_and\_t265.launch* :

```

1   <node pkg="tf" type="static_transform_publisher" name =
2     t265_to_d400" args="0.009 0.021 0.027 0.005 -0.018 0 /$(arg
      tf_prefix_camera1)_link /$(arg tf_prefix_camera2)_link 100"/>

```

Nous pouvons constater dans le nouveau arbre que la connexion est publié :

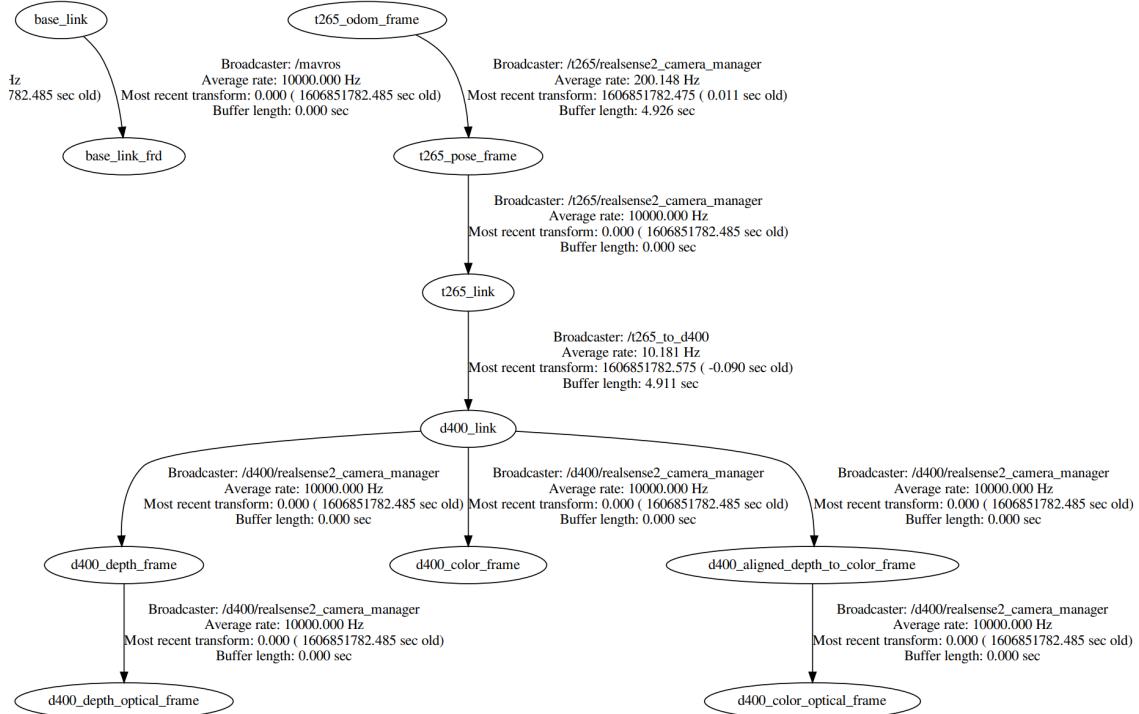


FIGURE 3.11 – T265 et D435 reliés via la transformation statique

Maintenant, en démarrant RTAB-Map nous pouvons voir comment la liaison avec la carte globale est générée :

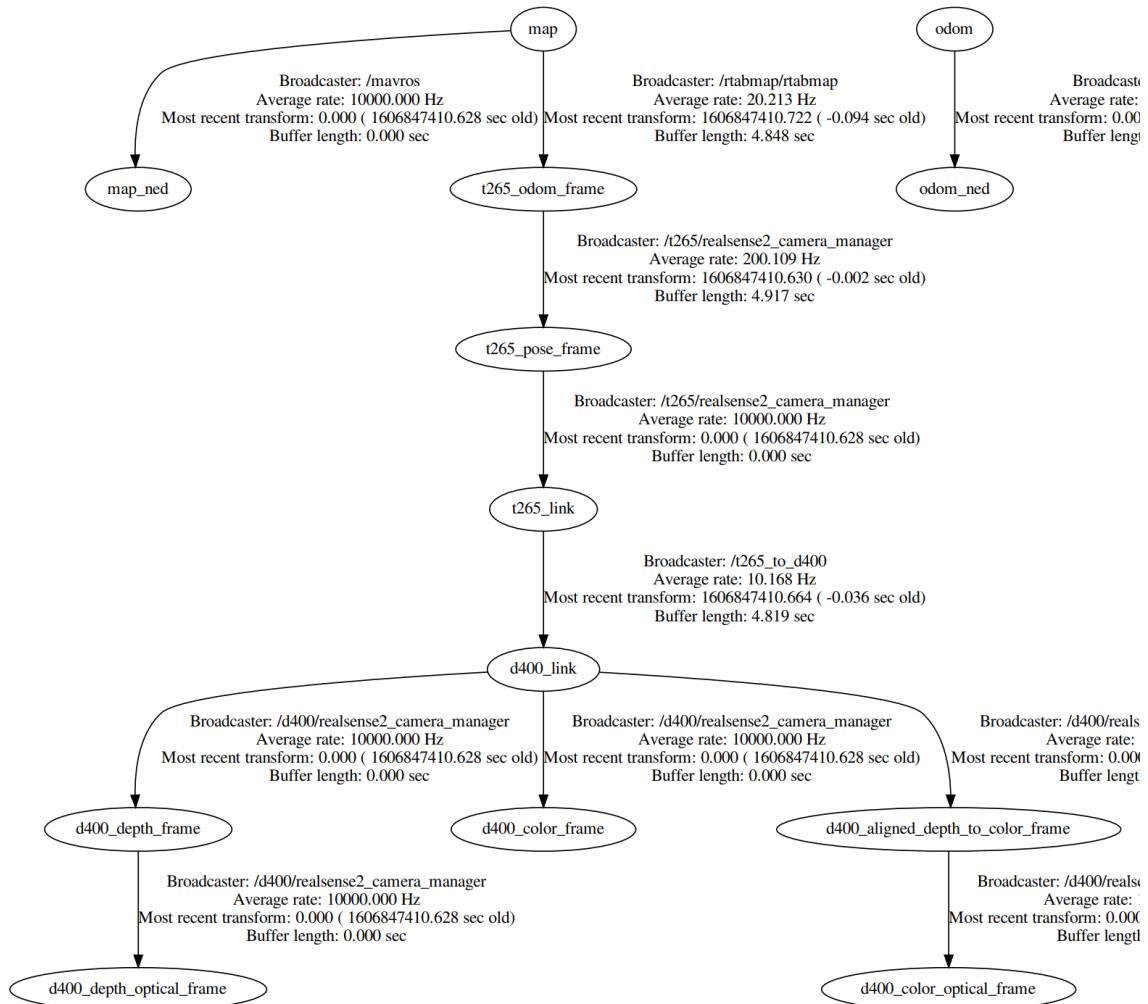


FIGURE 3.12 – Arbre de TF final

### 3.2.2 Fermeture de boucle et correction de position globale

Maintenant que nous avons une compréhension générale de la gestion des systèmes de cordonnées nous pouvons expliquer le fonctionnement du système de correction de RTAB-Map. Comme nous pouvons voir dans la figure 3.12, la transformation entre le TF de la carte globale (Map) et le TF utilisé par la T265 (t265\_odom\_frame) est publié par RTAB-Map, cela veut dire que quand il détecte une fermeture de boucle cette transformation sera modifiée pour recalculer l'estimation de la position générée par la T265 et la faire coïncider avec sa propre estimation, en éliminant ainsi la dérive.

Pour pouvoir faire la fermeture de boucle, RTAB-map a besoin de créer une base de données pour relier des images et caractéristiques dans les mêmes avec une position dans l'espace, cela veut dire qu'il faut faire une première exploration de l'environnement pour enregistrer ces valeurs. Pour visualiser la base de données nous pouvons utiliser l'outil suivant :

```
1 $rtabmap -databaseViewer <path-to-DataBase>
```

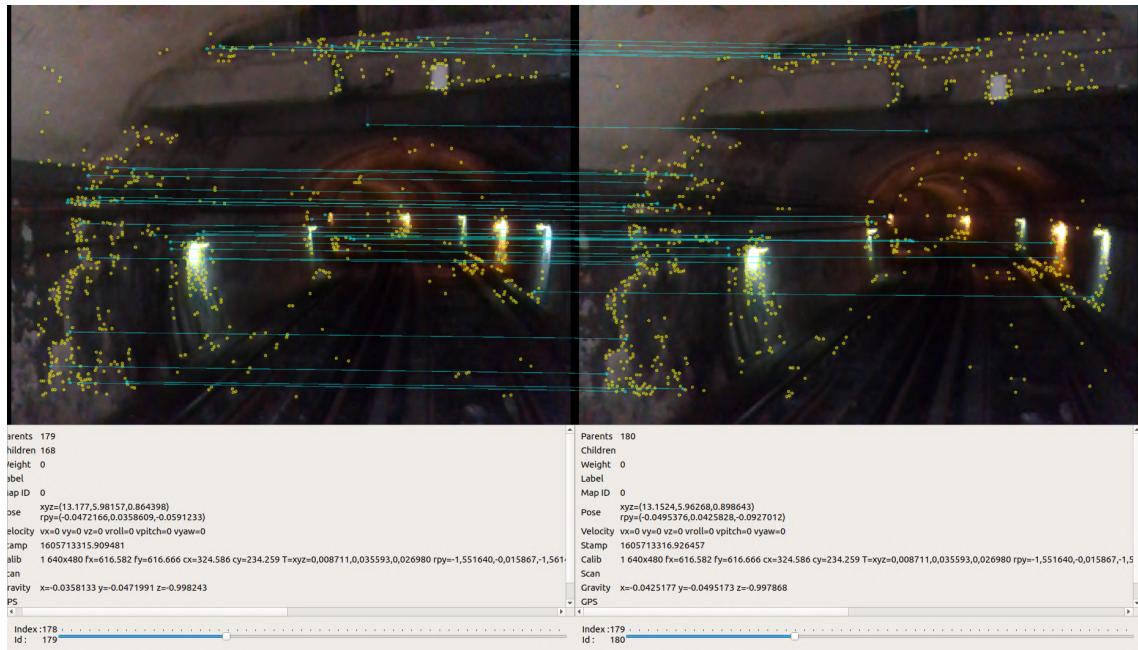


FIGURE 3.13 – Visualisation de DB avec reconnaissance de points caractéristiques (Tunnel de métro)

Nous pouvons choisir si nous voulons effacer la base de données et créer une nouvelle avec l'argument "**-d**" au moment de lancer RTAB-map.

### 3.2.3 Intégration avec Ardupilot

Maintenant que nous avons notre position dans un système de référence global corrigé, nous pouvons remplacer la source de position envoyé vers le contrôleur de vol parmi le node **vision\_to\_mavros**.

Dans la section 3.1 nous avons expliqué comment nous envoyons l'estimation de position généré par la caméra, cette estimation est référencé dans le système **t265\_odom\_frame** qui n'est pas corrigé. Pour envoyer une position corrigé nous avons besoin de la transformation correspondante à **map → t265\_pose\_frame** et pour l'obtenir nous devons faire la multiplication matricielle entre la matrice de transformation **map → t265\_odom\_frame** et **t265\_odom\_frame → t265\_pose\_frame**

$$[TF_{map \rightarrow pose}] = [TF_{map \rightarrow t265\_odom\_frame}] * [TF_{t265\_odom\_frame \rightarrow t265\_pose\_frame}] \quad (3.1)$$

Une fois que nous avons obtenu la matrice correspondante nous pouvons faire la rotation expliquée dans la section 3.1 3.2 et envoyer la nouvelle estimation.

Nous pouvons maintenant lancer les deux programmes ensemble avec les commandes présentés dans l'annexe A.7.

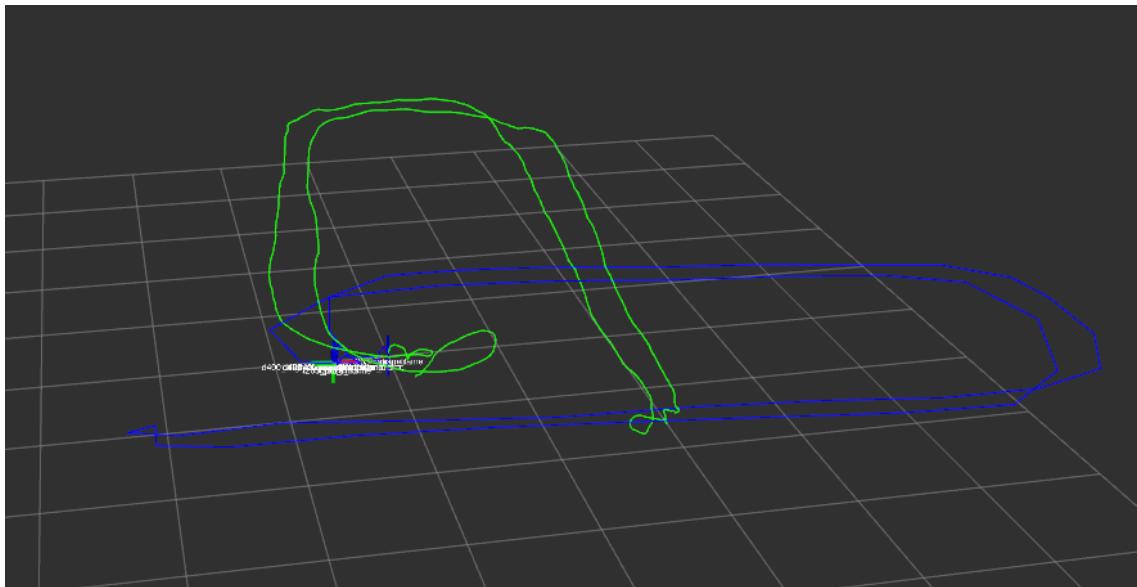


FIGURE 3.14 – Trajectoires de T265 (vert) et RTAB-Map (bleue) avec décalage.

Dans la figure 3.14 nous voyons un décalage de  $90^\circ$  entre les deux trajectoires, même si c'est juste un problème de visualisation nous pouvons le corriger en modifiant le message publié dans le topic "/body\_frame/path" par le node vision\_to\_mavros C.1.

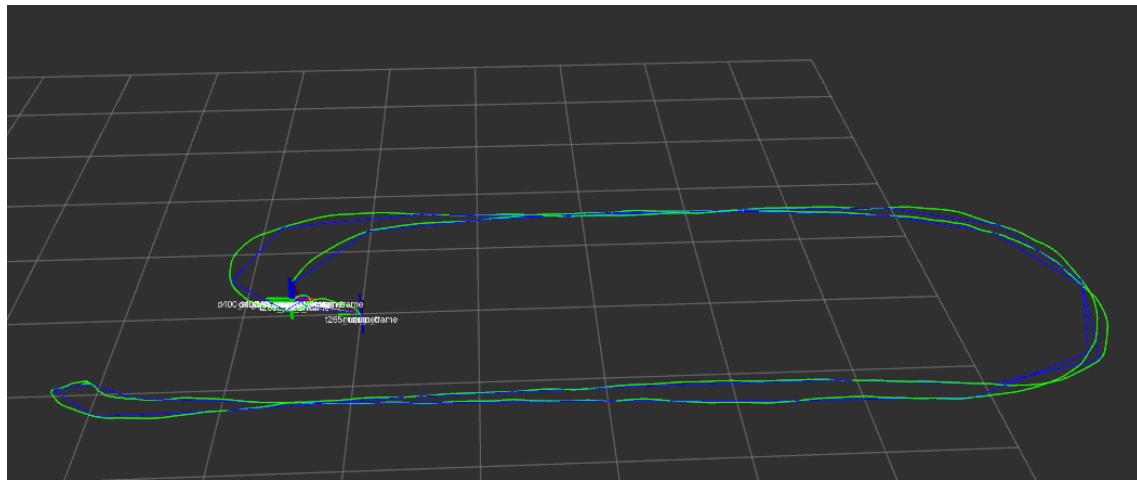


FIGURE 3.15 – Trajectoires recalés de T265 (vert) et RTAB-Map (bleue).

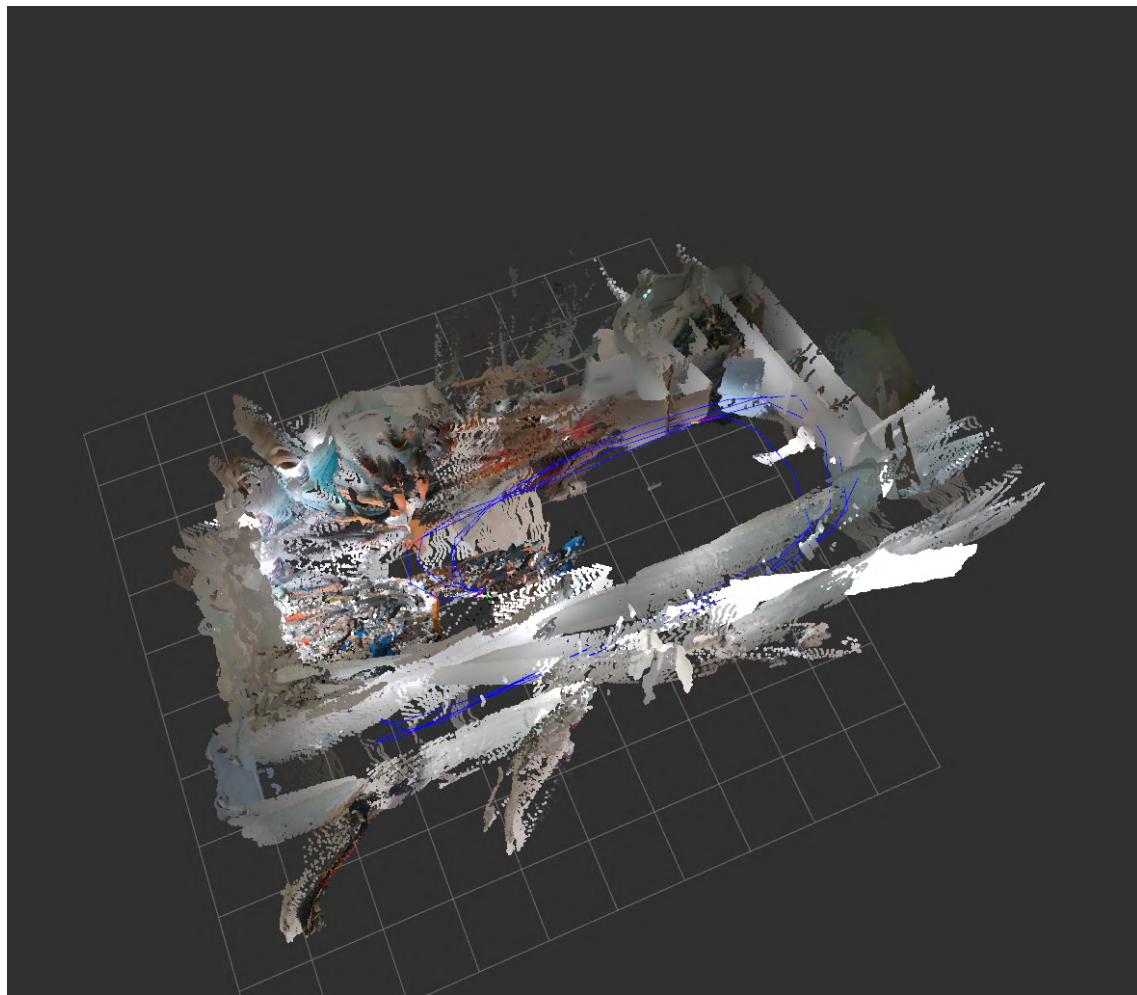


FIGURE 3.16 – Trajectoire avec reconstruction 3D de l'environnement.

### 3.2.4 Problèmes rencontrés

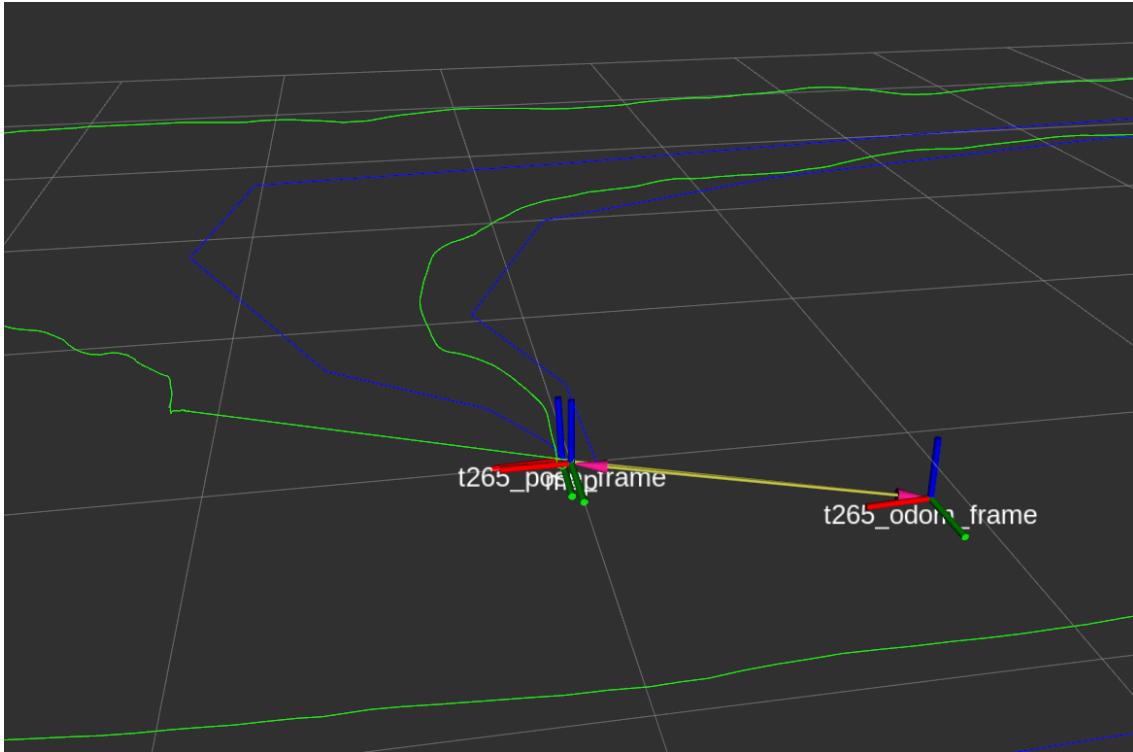


FIGURE 3.17 – Correction de dérive via grâce a fermeture de boucle.

Nous avons constaté pendant les test qu'il y a une discontinuité dans l'estimation de la position générée par RTAB-map à cause de la fermeture de boucle. Nous pouvons voir dans la figure 3.17 que quand l'algorithme trouve une fermeture de boucle il y a un saut discret dans la position à cause du déplacement du système **t265\_odom\_frame** par rapport au système **map**. Ce comportement est expliqué dans le REP 105 [19] (ROS Enhancement Proposal) "*Coordinate Frames for Mobile Platforms*" :

#### — Odom :

Le cadre de coordonnées appelé "odom" est un cadre fixe global. La position d'une plateforme mobile dans le cadre odom peut dériver au fil du temps, sans aucune limite. Cette dérive rend le cadre odom inutile comme référence globale à long terme. Cependant, la pose d'un robot dans le cadre odom est garantie d'être continue, ce qui signifie que la pose d'une plateforme mobile dans le cadre odom évolue toujours de manière fluide, sans sauts discrets.

Dans une configuration typique, le cadre odométrique est calculé en fonction d'une source odométrique, comme l'odométrie des roues, l'odométrie visuelle ou une unité de mesure inertielles. **Le cadre odom est utile comme référence locale précise à court terme, mais la dérive en fait un piètre cadre de référence à long terme.**

#### — Map :

Le cadre de coordonnées appelé "map" est un cadre fixe global, avec son axe

Z pointant vers le haut. La pose d'une plateforme mobile, par rapport au cadre map, ne devrait pas dériver de façon significative au fil du temps. Le cadre de carte n'est pas continu, ce qui signifie que la pose d'une plateforme mobile dans le cadre map peut changer en sauts discrets à tout moment.

Dans une configuration typique, un composant de localisation recalcule constamment la pose du robot dans le cadre map en fonction des observations des capteurs, éliminant ainsi la dérive, mais provoquant des sauts discrets lorsque de nouvelles informations du capteur arrivent.

**Le cadre map est utile comme référence globale à long terme, mais les sauts discrets dans les estimateurs de position en font un mauvais cadre de référence pour la détection et la génération de commandes locales.**

Cette discontinuité dans l'estimation de la position provoque des mouvements brusques sur la drone car il détecte qu'il y a eu un décalage par rapport à la consigne et il essaie de se corriger, cela démontre pourquoi le système de référence map n'est pas adapté pour la génération de commandes locales.

### 3.2.5 Conclusion

Ce rapproche présente des points très intéressantes comme la cartographie 3D, la fermeture de boucle et la localisation basé sur des basses données déjà existantes, mais il n'est pas adapté pour utiliser comme source d'odometrie à cause des discontinuités dans l'estimation. Une possible solution serait d'intégrer un filtrage de Kalman pour réaliser la fusion avec plusieurs capteurs avant d'envoyer l'estimation au contrôleur de vol comme mentionné dans la section 3.1.4.

# Chapitre 4

## Lidar

Pendant les test du système de positionnement nous avons vu qu'il faut aussi implémenter un système de détection d'obstacles dans plusieurs directions pour éviter des accidents et générer les commandes d'une façon plus intelligente. Nous avons choisi d'utiliser le capteur VL53L1X développé par ST Microelectronics [20] grâce à sa taille et sa capacité de mesurer des distances entre 10 et 400 cm en intérieur.



FIGURE 4.1 – Capteur SMT VL53L1X

### 4.1 VL53L1X

Ce capteur est très puissant et il présente plusieurs fonctionnalités avancées si nous utilisons l'API originale écrite en C++, mais nous avons choisi d'utiliser une librairie implémenté sur python [8] pour rendre plus simple l'intégration. Nous pouvons nous communiquer avec ce capteur par SPI ou I2C et Ardupilot le supporte comme source de mesure d'altitude pour l'atterrissement ou comme capteur de proxi-

mité pour la détection d'obstacles, avec une limitation : l'adresse I2C du capteur se réinitialise à chaque redémarrage.

Pour pouvoir connecter plusieurs capteurs en I2C sur Ardupilot il faut spécifier l'adresse de chacun, mais nous n'avons pas un moyen de changer les adresses de façon permanente, cela veut dire que à chaque redémarrage nous avons besoin d'allumer un capteur à la fois pour changer son adresse I2C. Cette gestion des capteurs doit être réalisée par la Jetson Nano et une fois que chaque capteur a été configuré nous pouvons lire les données indépendamment et les envoyer vers le contrôleur de vol via Mavros.



FIGURE 4.2 – Breakout Board utilisé

Pour réaliser cette configuration nous utilisons le pin "*xshut*" qui nous permet d'éteindre le capteur quand il est connecté à 3,3 V. Pour lire les valeurs mesurées et réaliser la configuration initiale nous avons créé une librairie python (C.2) dont nous utilisons dans un node ROS (C.3) qui est en charge de publier les valeurs dans les topics correspondants.

### 4.1.1 Intégration sur le drone

Nous avons choisi d'installer huit capteurs avec un décalage de 45 degrés entre eux.

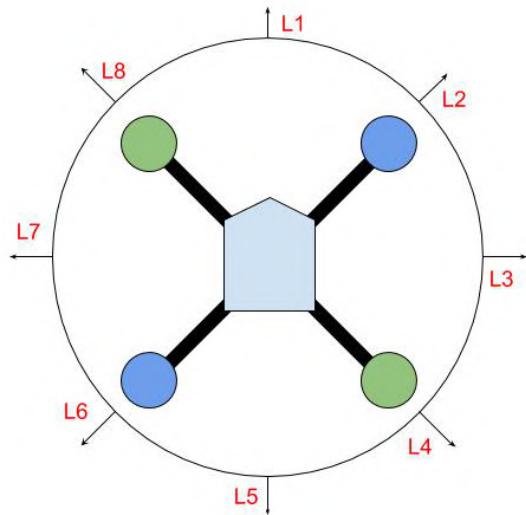


FIGURE 4.3 – Distribution des capteurs

Chaque capteur partage 4 câbles (Vin, GND, SDA, SCL) et il faut rajouter un câble de contrôle supplémentaire pour la configuration (XSHUT) donc nous avons 12 câbles à connecter dans la Jetson Nano. Pour faciliter le montage/démontage de la même nous avons installé les connecteurs JST-GH présentés dans la section 2.3.2 figure 2.9b.

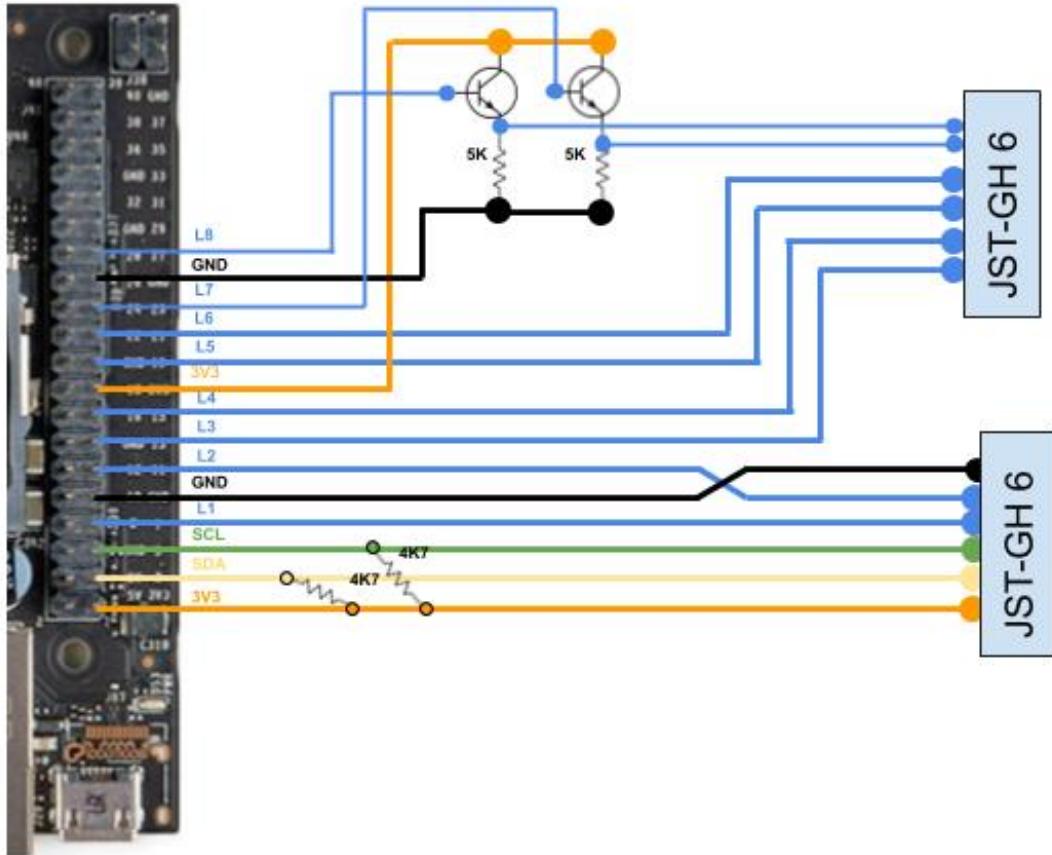


FIGURE 4.4 – Schéma de connections pour les capteurs LIDAR

Dans la figure 4.4 nous pouvons voir quelques composants supplémentaires (Résistances et Transistors), le bloc supérieur serve à commander le pin XSHUT des deux capteurs situés dans les extrémités du faisceau électrique (4.7). Cette méthode de commande est nécessaire car la carte de développement de la Jetson Nano utilise comme convertisseur de niveau logique bidirectionnel le TXB0108 qui est assez limité en courant de sortie ( $\pm 1\text{mA}$ ) et en charge capacitive ( $70\text{pF}$ ). Nous ne sommes pas sûres si la capacitance de ces câbles dépasse la limite mais sans les transistors nous n'arriverons pas à établir la connexion avec les capteurs.

Les deux résistances qui relient les câbles SDA et SCL à l'alimentation 3,3V sont nécessaires pour le bus I2C car c'est un protocole logique TTL Open-Drain. Il faut faire attention aux breakout boards des capteurs utilisés car ils ont déjà intégrés des résistances qu'il faut enlever (voir figure 4.5).

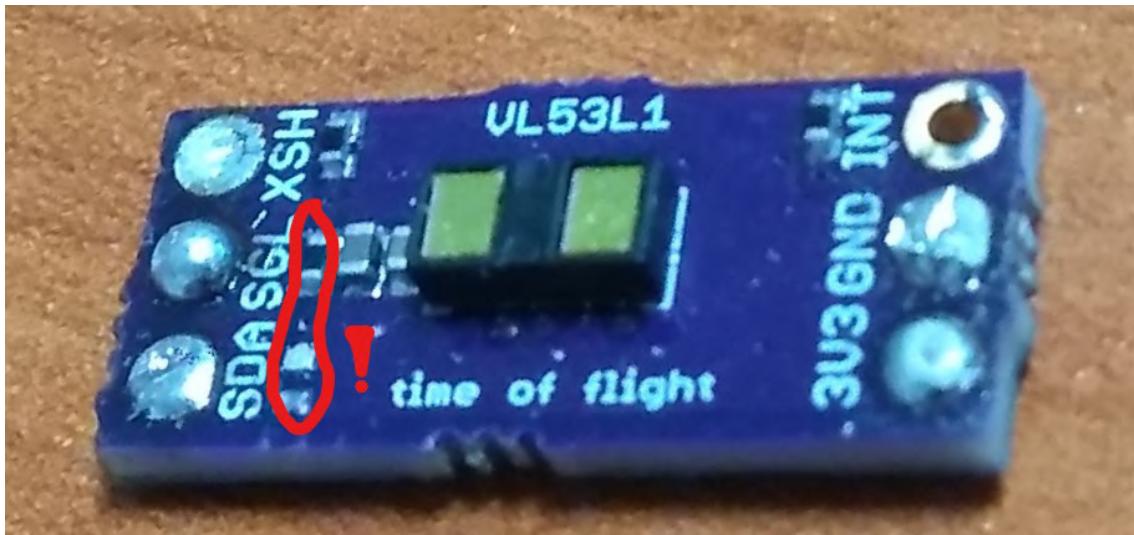


FIGURE 4.5 – Résistances à enlever

Pour faciliter l'installation sur le drone nous avons imprimé en 3D des supports qui intègrent des connecteurs JST-GH.

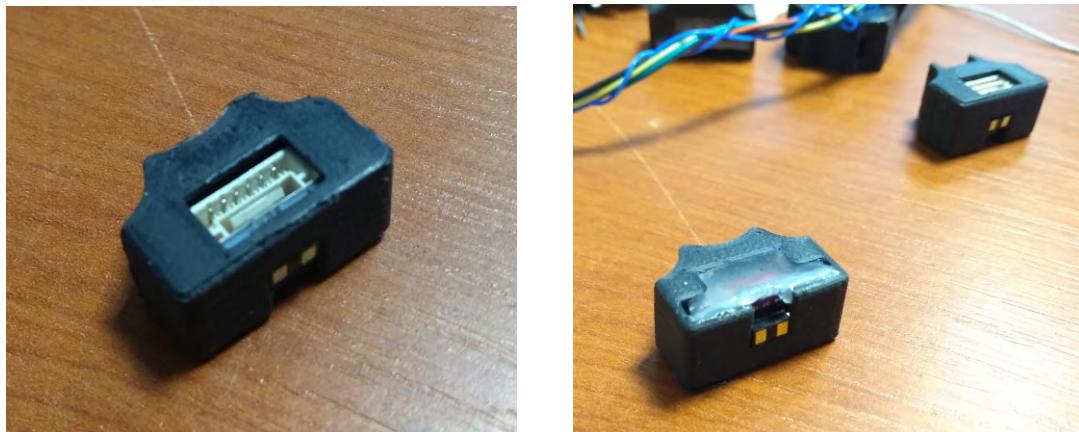


FIGURE 4.6 – Capteurs LIDAR avec support et connecteur JST-GH

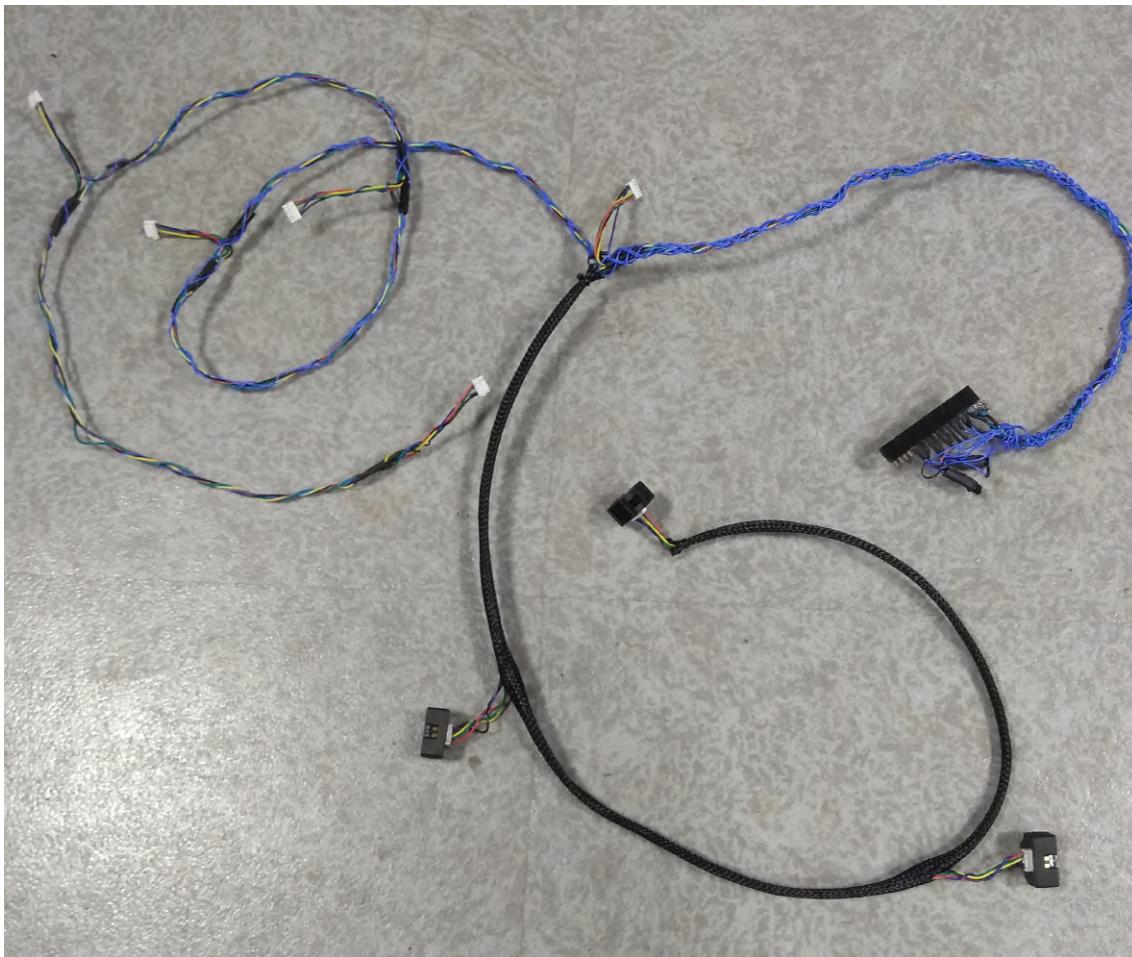


FIGURE 4.7 – Prototype du faisceau électrique

#### 4.1.2 Intégration avec Ardupilot

Jusqu'à ici nous avons les mesures disponibles sur la Jetson Nano, nous pourrions les utiliser pour améliorer notre estimation de position avec la fusion des capteurs mentionné dans le chapitre 3 section 3.1.4, mais nous avons choisi de tester une fonctionnalité d'Ardupilot qui est l'évasion d'obstacles automatique [5].

Il y a deux options pour envoyer les données vers Ardupilot, via le message Mavlink OBSTACLE\_DISTANCE ou via le message DISTANCE\_SENSOR. Le dernier est un message de l'ancienne version de Mavlink, et il nous donne moins de flexibilité mais il es le seul a être implémenté sur Mavros donc c'est lui qui nous allons utiliser.

Pour envoyer les distances via Mavros nous devons les publier chaque un dans un topic depuis notre node de gestion des capteurs, une fois que les valeurs sont publiés, c'est le node Mavros qui va s'abonner aux topics et les renvoyer en forme de message Mavlink.

Pour configurer cette fonctionnalité de Mavros nous devons modifier le fichier **apm\_config.yaml** pour configurer le plugin "distance sensor". Par exemple pour ajouter un seul capteur la procédure serait la suivante :

```
1 $ cd /opt/ros/melodic/share/mavros/launch
2 $ sudo gedit apm_config.yaml
```

maintenant il faut chercher la configuration du plugin **distance sensor** et la modifier de la façon suivante :

```
1     distance_sensor:          <== plugin name
2         rangefinder_sub: <YOUR TOPIC NAME> <== ROS subscriber name
3         subscriber: true    <== set subscriber type
4         id: 1    <== sensor instance number must match the one set
5             on ArduPilot side
6         orientation: YAW_0  <== facing forward
```

Une fois que nous avons configuré Mavros nous pouvons lancer notre programme de gestion des capteurs et vérifier si Ardupilot reçoit le message via Mission planner. Nous avons deux outils pour cette vérification, Mavlink Inspector et Proximity, nous pouvons les lancer depuis le panneau de commandes (Ctrl + F).

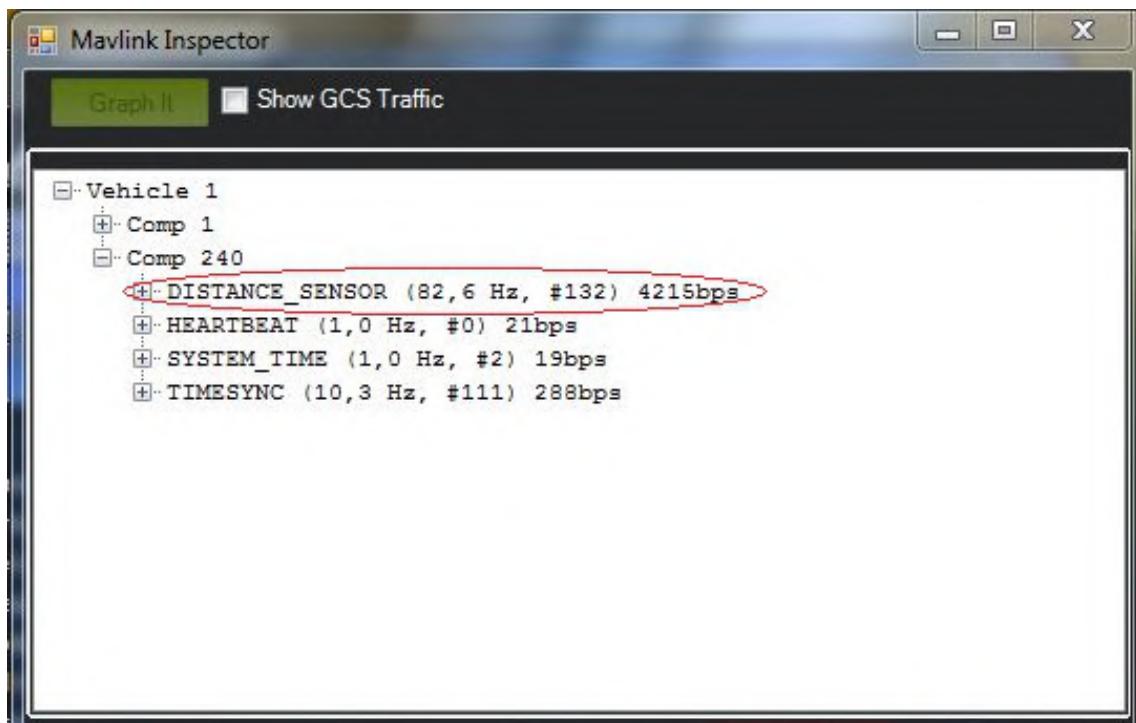


FIGURE 4.8 – Vérification de réception de message via Mavlink

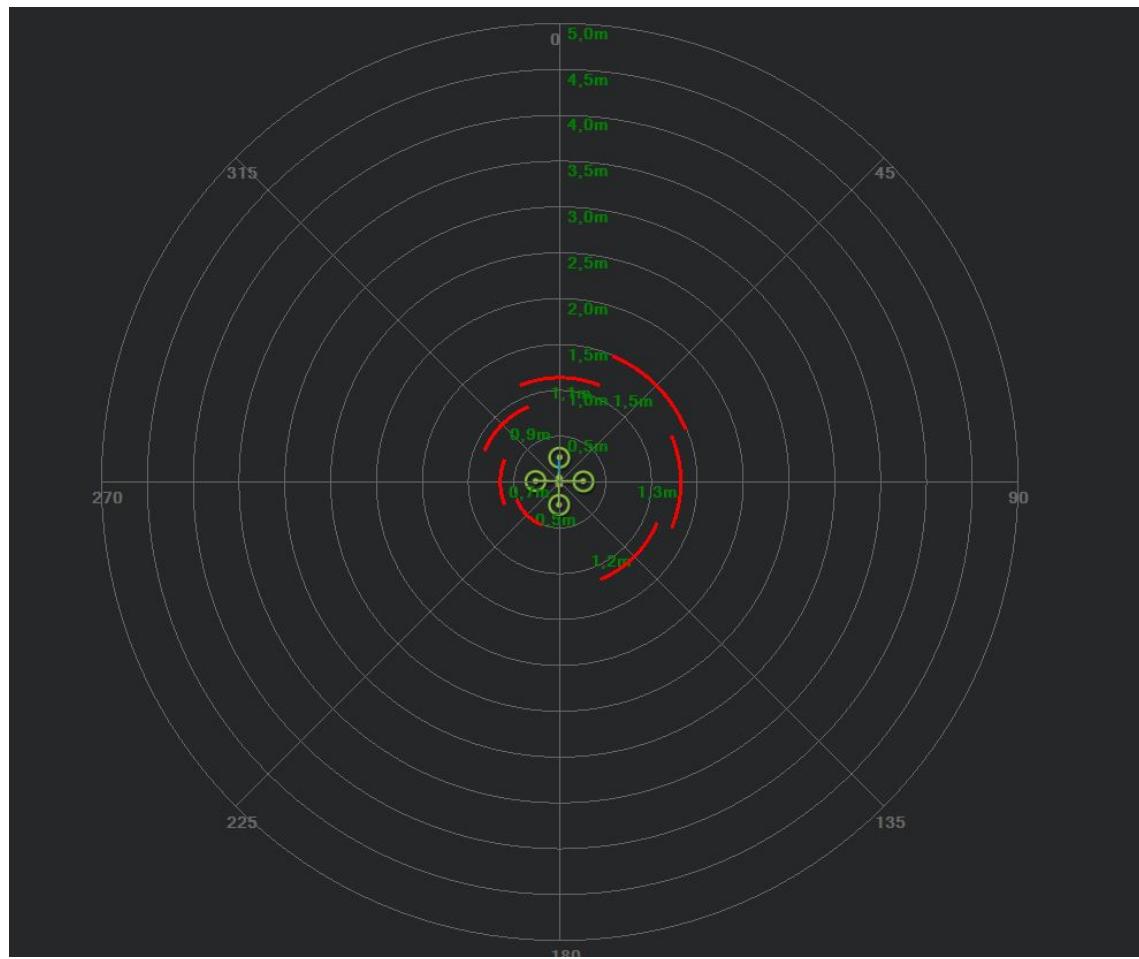


FIGURE 4.9 – Visualisation des capteurs de distance sur Mission Planner

### 4.1.3 Tests

Nous avons plusieurs configurations et modes de vol à tester qui sont détaillées dans l'article [5], mais nous n'avons pas eu le temps de tester correctement le système.

# Chapitre 5

## Mode Automatique

Nous avons créé plusieurs programmes de contrôle du drone basés sur les scripts développés par Andrea Belloni qui sont disponibles dans son GitHub [7], qui nous permettent d'automatiser les trajectoires de vol. Cette fonctionnalité est indispensable pour notre application car nous avons besoin de donner des ordres au drone basés sur une stratégie de commande intelligente. Toutes les commandes utilisées dans le script partent du fait que le drone à un moyen de connaître ça position et vitesse dans tout moment.

Le code commenté est disponible dans l'annexe C.4, ou nous trouvons des commandes automatisées pour des trajectoires simples (Décollage, déplacement et atterrissage), commandes par clavier et commandes reliées aux algorithmes de vision par ordinateur qui seront présentés dans le chapitre 6.

### 5.1 SITL (Software In The Loop)

?? Pour éviter les accidentes reliés a la commande automatique et diminuer le risque de casse du drone nous avons utilisé un simulateur intégré dans Ardupilot qui profite de ça portabilité et nous permet de le faire tourner dans notre ordinateur Linux (Dans ce cas, l'ordinateur accompagnant Jetson Nano).[6] Les instruction d'installation sont disponibles dans l'annexe A.4.2

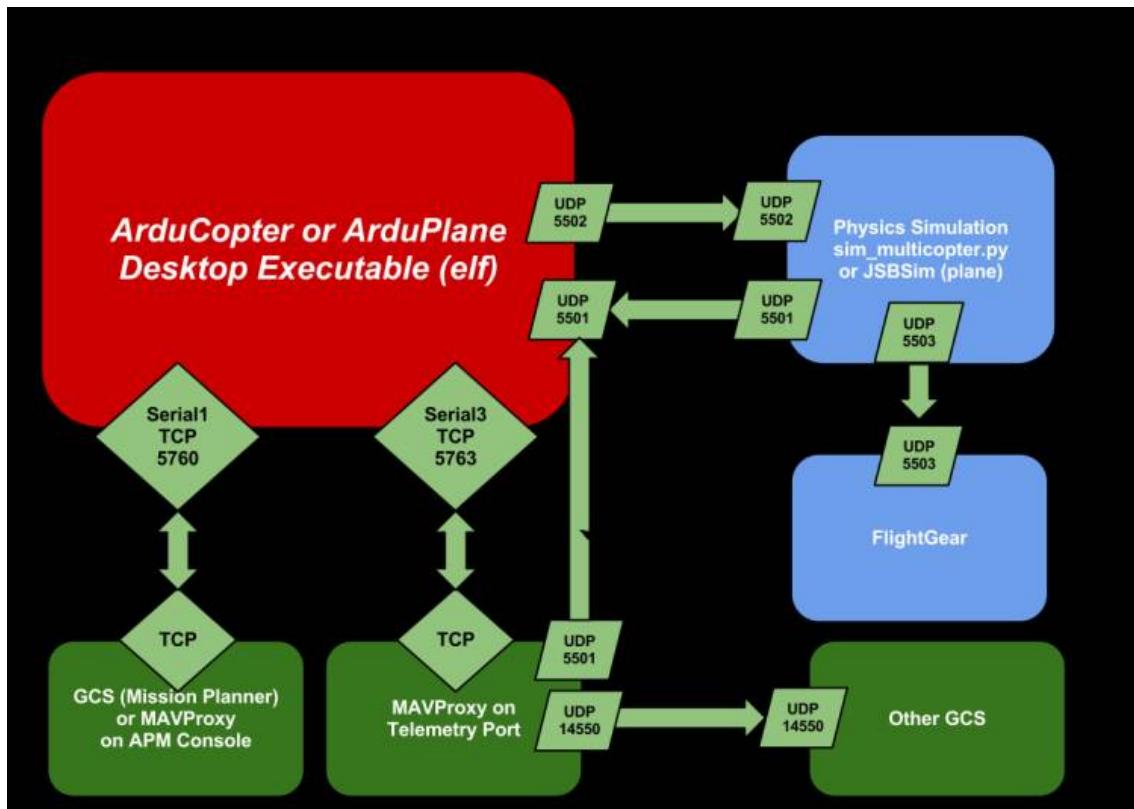


FIGURE 5.1 – Architecture du simulateur [6]

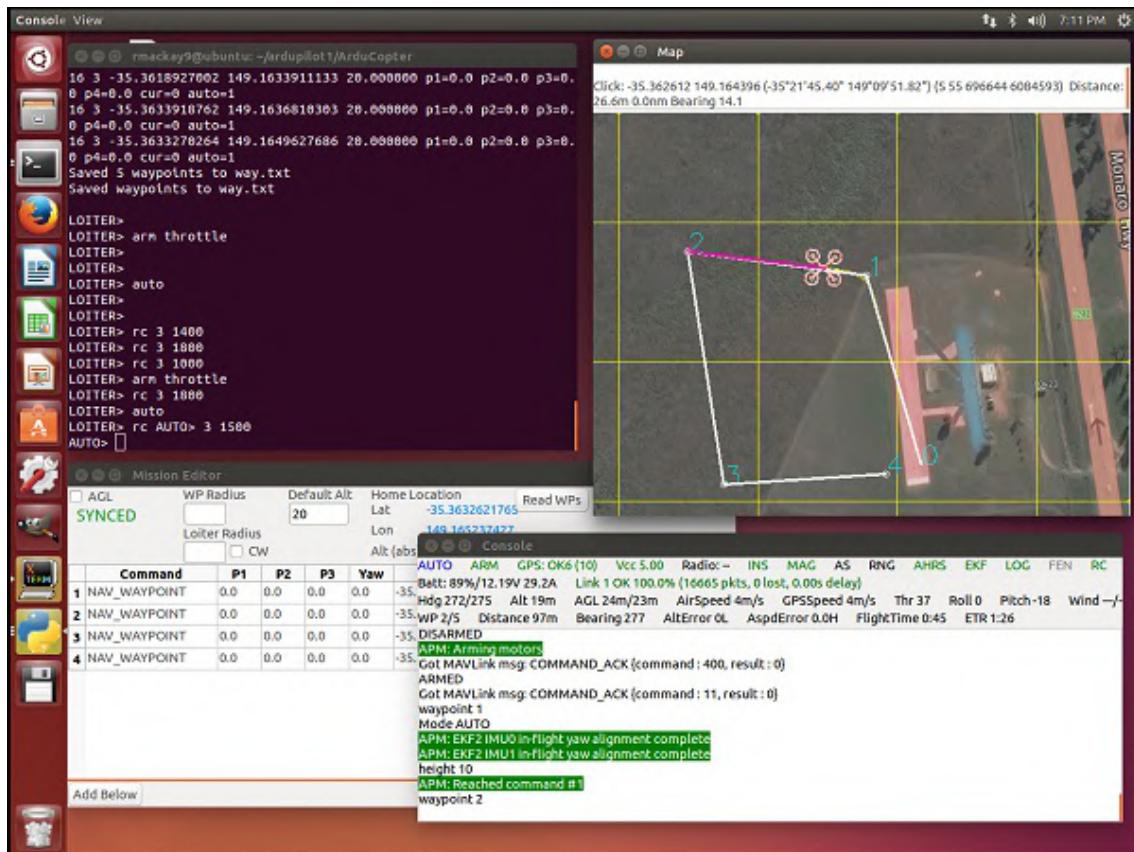


FIGURE 5.2 – Exemple de simulation [6]

Pour envoyer les commandes au simulateur nous devons configurer Mavros pour se connecter sur un des ports TCP disponibles qui sont créés via Mavproxy. Les ports disponibles sont affichés au moment de lancer SITL avec la commande suivante :

```
1 $ cd /ardupilot/Tools/autotest
2 $ sim_vehicle.py -v ArduCopter --map --osd --console
3
```

Avec le simulateur démarré nous pouvons utiliser toutes les fonctionnalités de Mavros.

# Chapitre 6

## Machine Vision

Au moment de commencer ce projet, nous croyons que le défi allait être dans la génération d'une commande intelligente et que le système de positionnement visuel était déjà suffisamment robuste, c'est pour ce la que nous avons commencé par le développement des algorithmes de traitement d'images en utilisant comme source la caméra RGBD Intel RealSense D435.

### 6.1 Gradient

Avec la caméra RGBD nous avons comme sortie une image à couleur alignée avec une image de profondeur, cette dernière nous donne pour chaque pixel la distance entre le point dans l'espace qui lui correspond et la caméra en millimètres. Pour une visualisation plus intuitive nous pouvons choisir différents "color maps", couleurs qui nous allons donner à certaines plages de valeurs de distance.

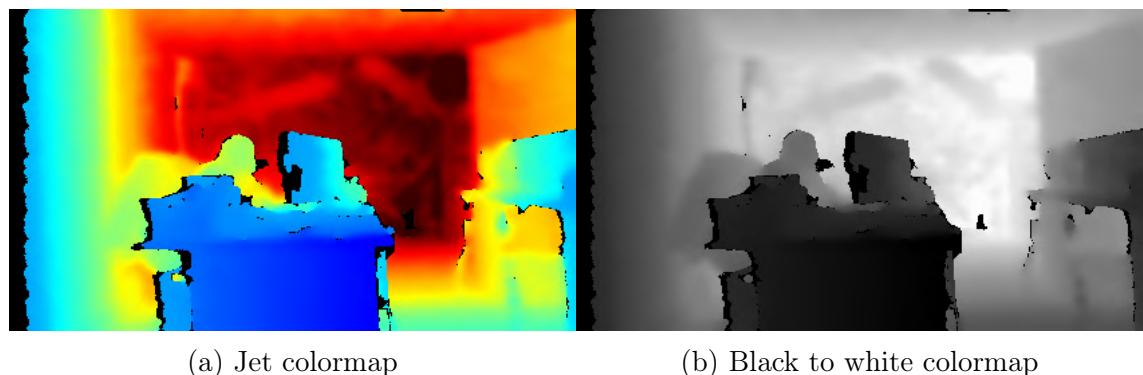


FIGURE 6.1 – Alternatives de visualisation

Comme nous pouvons voir dans la figure 6.1b, avec ce type de visualisation le gradient de couleur est linéairement proportionnel au gradient de profondeur, et c'est là où nous trouvons la base logique pour notre algorithme.

L'idée c'était d'utiliser OpenCV pour filtrer l'image et calculer la dérivée (gradient) dans différents points de l'image pour ainsi avoir une moyenne de tous les vecteurs de dérivée dans le sens verticale et horizontale, si la moyenne est presque nulle nous pouvons dire que la caméra est centrée par rapport au point de fuite de l'image.

### 6.1.1 Démonstration de faisabilité

Pour les premiers essais nous avons utilisé l'outil "Google Collab" qui nous permet de programmer en Python avec un format très utile pour la recherche, les "Notebooks" (nous pouvons mélanger lignes de code avec graphiques, commentaires et autres ressources) sans avoir besoin d'installer toutes les librairies nécessaires dans un ordinateur.

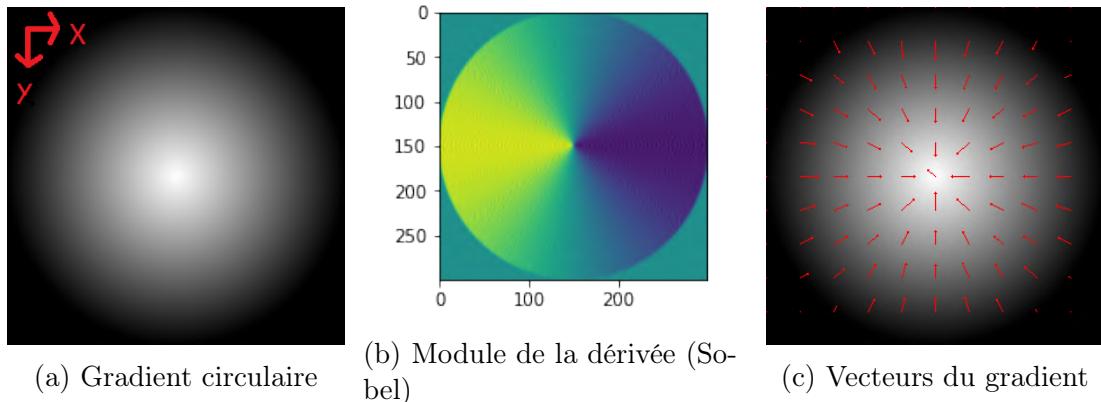


FIGURE 6.2 – Idée générale pour la détection du point de fuite

Dans la figure 6.2a nous voyons une simplification de l'image d'un tunnel éclairé à la fin ou d'une image de profondeur où les points plus proches sont représentés par la couleur noire et le blanc absolu c'est l'infini (ou tous les points hors le range du capteur). Si nous utilisons l'opérateur de Sobel intégré dans OpenCV elle nous donne comme résultat deux matrices dans lesquelles nous trouvons le module de la dérivée dans le sens "x" (6.2b) et "y" et parmi un simple calcul pythagorique (ou la fonction "cartToPolar") nous pouvons obtenir le gradient total dans chaque pixel. Dans la figure 6.2c nous voyons les vecteurs correspondants à une grille de 100 points. À continuation vous pouvez voir le même traitement appliqué dans une image réel d'un tunnel de métro.

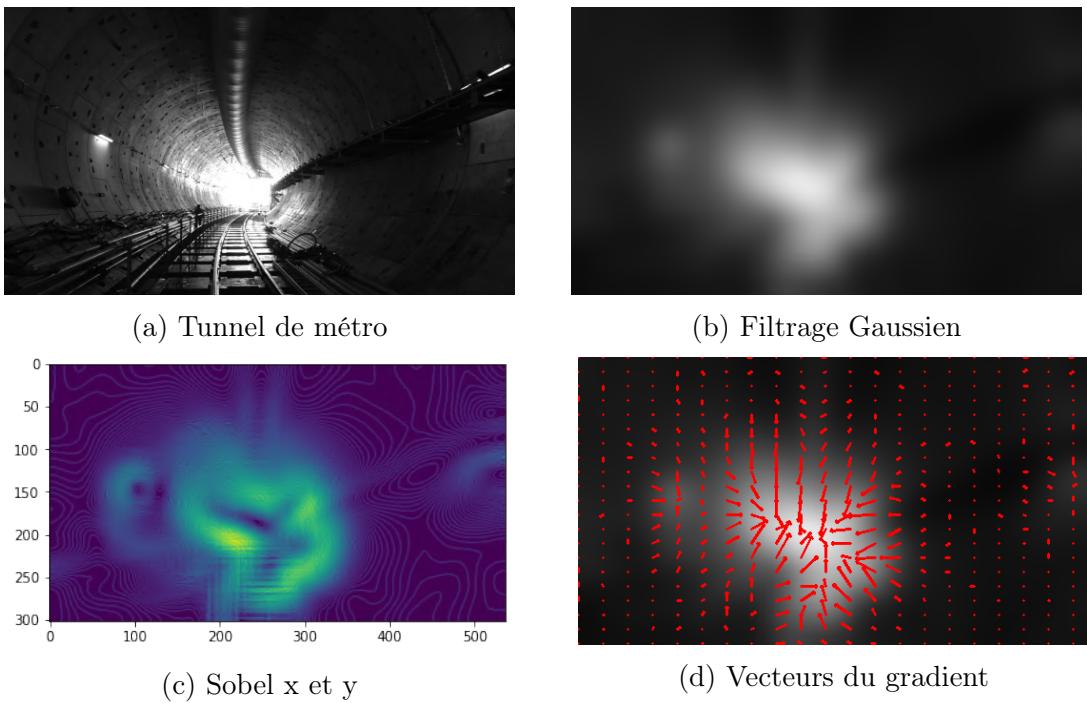


FIGURE 6.3 – Test dans un cas plus complexe

### 6.1.2 Intégration avec la caméra D435

Pour l'intégration de l'algorithme avec la caméra sur la Jetson Nano nous avons choisi de rester avec Python pour la simplicité au moment de développer, et si l'installation d'OpenCV est fait avec support de CUDA nous ne devrions pas noter des différences de performance si grandes avec C++. Dans une première étape le développement a été réalisé avec le Wrapper Python de l'API RealSense mais après nous l'avons intégré aussi sur RealSenseROS.

# Prétraitement d'images

Pour avoir une bonne performance nous allons utiliser le filtrage développé par Intel et intégré sur l'API RealSense ROS. Nous avons choisi d'activer seulement 3 filtres car les autres introduisent un retard dans l'image.

- Decimation filter
  - Spatial Edge-Preserving filter
  - Holes Filling filter

La documentation pour les filtres est disponible dans le site d'Intel Realsense [12] et il peuvent être activés depuis le launch file correspondant.

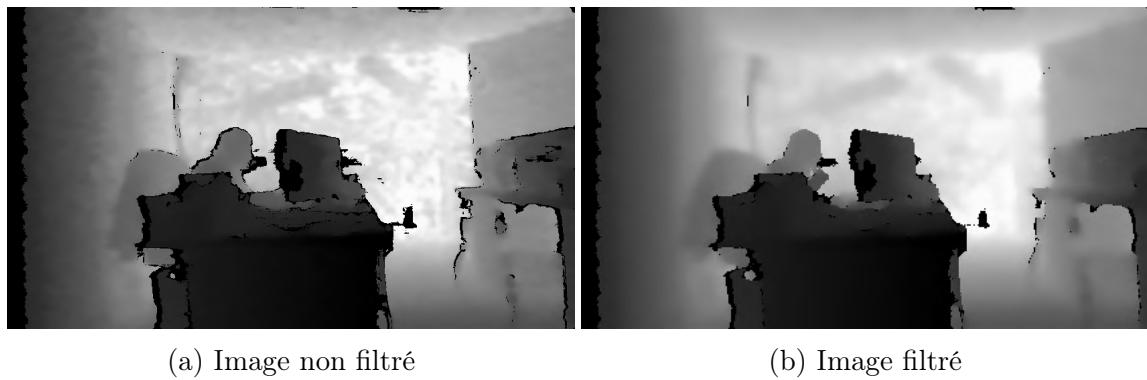


FIGURE 6.4 – Importance du pré-traitement de l'image

### Calcul du gradient

Comme nous avons expliqué au début de ce chapitre, l'idée est de pouvoir détecter un décalage entre le centre de l'image et le point de fuit du tunnel et ainsi générer une commande pertinente pour le drone. Pour les premiers tests nous avons choisi de faire les calculs seulement en direction horizontale, cela veut dire dans l'axe X de l'image car nous pouvons le traduire facilement à une commande en lacet pour le drone.

Nous avons deux programmes principales, **depthgradient\_ROS.py** qui peut récupérer les images de profondeur depuis le topic ROS correspondant, faire le traitement et nous donner une valeur de gradient moyen pour chaque moitié de l'image et une moyenne totale et aussi afficher une représentation graphique de ces valeurs (Nous allons voir quel est l'intérêt de le faire en deux moitiés dans la section 6.1.2). Le fonctionnement générale du code (disponible dans l'annexe C.5.1) est illustré dans la figure 6.5.

### Suivi de direction de tunnel automatique

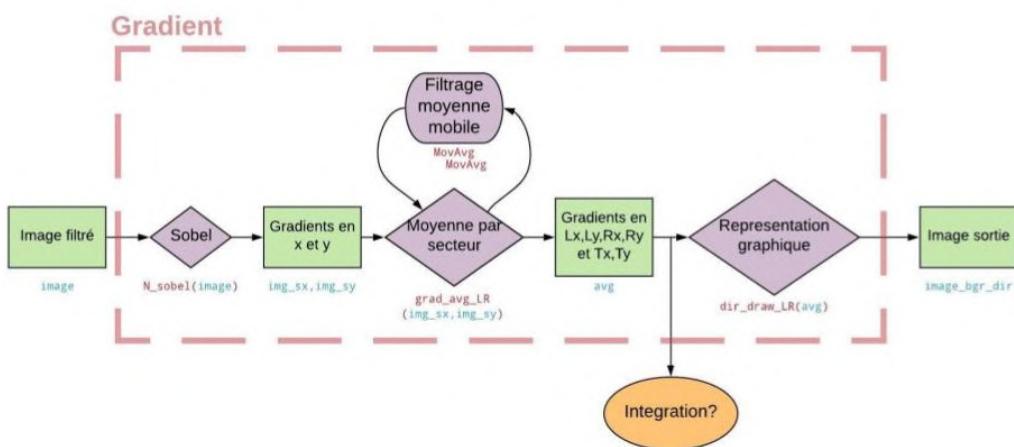


FIGURE 6.5 – Pipeline de l'algorithme du gradient.

Le deuxième programme **interpreter\_ROS.py** (C.5.2) est lequel est sensé à être utilisé pour la génération de commandes, il à la fonction du callback qui est appelé quand une nouvelle image est disponible, et c'est là où nous utilisons les

fonctions du module `depthgradient_ROS.py` pour faire le processus et récupérer la valeur moyenne du gradient. Une fois que nous avons cette valeur nous pouvons créer un contrôleur proportionnel pour donner des commandes en lacet avec les principes présentés dans le chapitre 5.

Une alternative très simple mais pratique pour comprendre le fonctionnement c'est une commande à vitesse angulaire constante, cela veut dire que si le gradient nous dit que nous sommes décalés vers la droite (valeur négative), nous allons tourner en sens antihoraire jusqu'à que le module du gradient totale soit plus petit que une tolérance prédéfinie, et le même principe s'applique dans le sens inverse.

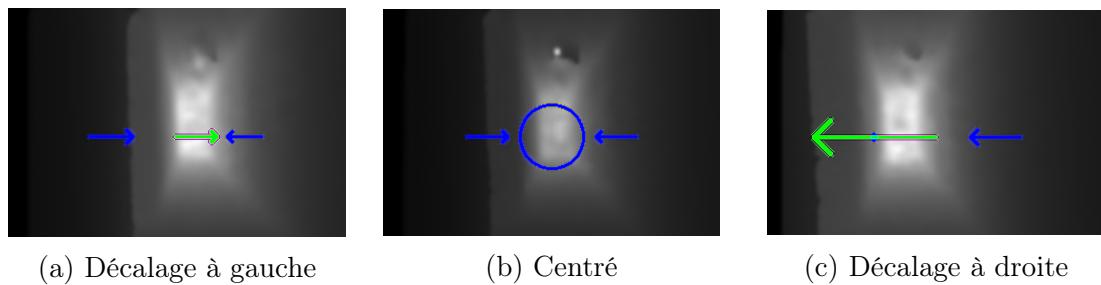


FIGURE 6.6 – Visualisation de l'algorithme dans un couloir

## Détection de bifurcations

Nous avons dit que nous sommes centrés dans le tunnel quand le gradient total a un module suffisamment petit car les vecteurs du gradient de chaque moitié sont similaires en magnitude mais de signe opposé, mais nous avons deux autres situations dans lesquelles le gradient total peut devenir  $\approx 0$ . La situation plus évidente c'est lorsque les deux gradients sont  $\approx 0$ , ce qui pourrait arriver si nous sommes alignés perpendiculairement à un mur. L'autre situation nous la trouvons dans une bifurcation dans laquelle on est parfaitement centrés.

Nous pouvons détecter facilement la première situation en regardant le module de chaque gradient et la deuxième en regardant ses directions, car dans une bifurcation nous avons une divergence (les deux vecteurs se dirigent vers l'extérieur de l'image).

Une fois que nous avons trouvé un de ces situations la commande à réaliser sera la même, il faut choisir une direction préférentielle pour tourner.

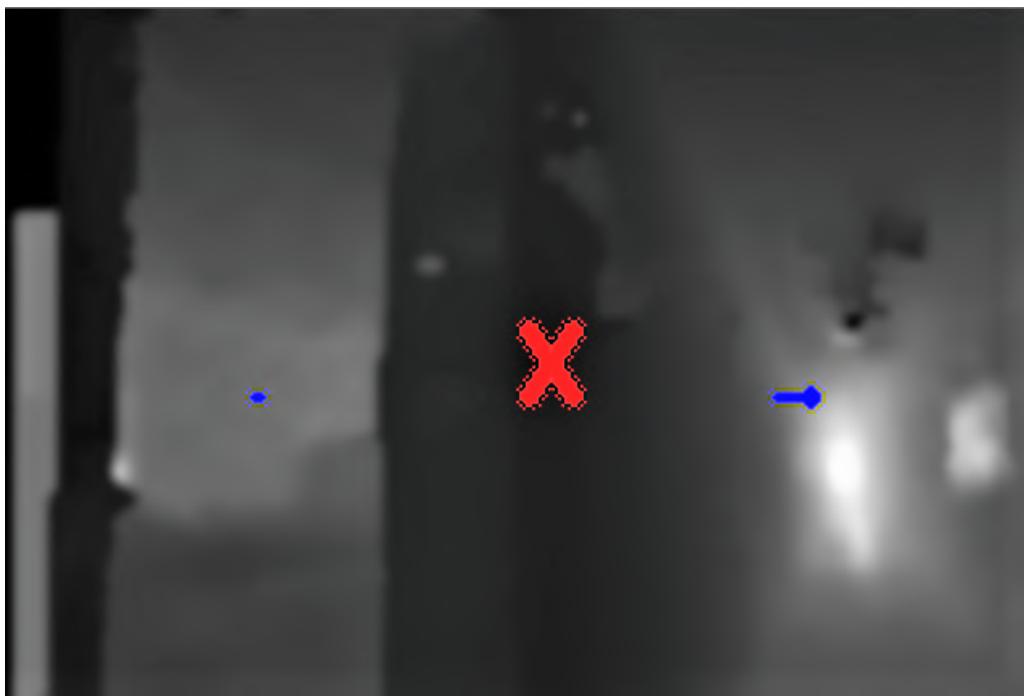


FIGURE 6.7 – Détection de bifurcation

### Comportement attendu

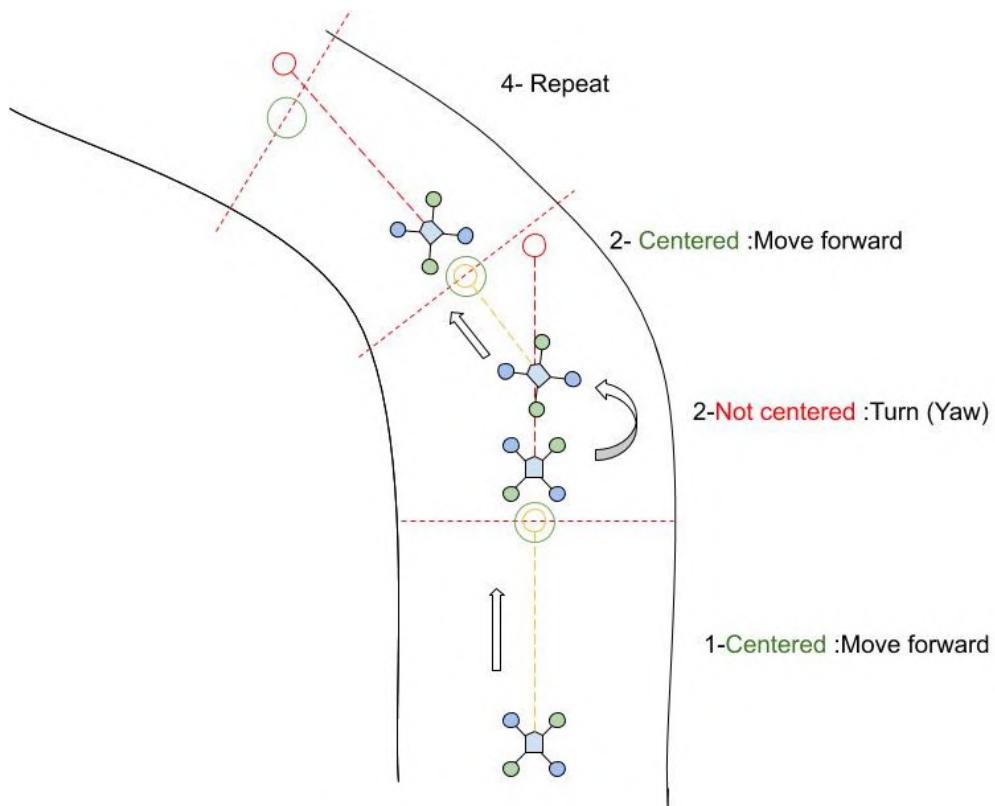


FIGURE 6.8 – Simplification du comportement attendu

## 6.2 Détection d'obstacles et planification de trajectoire

Dans cette section nous allons présenter une deuxième application du traitement d'images RGBD pour générer une stratégie de commande qui nous permet de trouver des chemins sans obstacles pour se déplacer.

### 6.2.1 Détection d'espace libre

Dans le calcul du gradient nous avons utilisé les valeurs de profondeur reliés à chaque pixel  $z = \text{profondeur}(u, v)$  en étant "u" la colonne et "v" la file de la matrice de pixels , mais nous pouvons aussi transformer un point  $P(u,v,z)$  dans un point  $P(X,Y,z)$  en réalisant la déprojection parmi des valeurs intrinsèques de la caméra comme la distance focale et l'ouverture. Nous n'allons pas entrer en détail sur ces calculs car l'API de RealSense nous fournit une méthode pour faire cette conversion, mais c'est important de comprendre que nous pouvons changer entre la projection 2D et l'espace 3D facilement.

Grâce à la fonctionnalité que nous venons de présenter nous pouvons aussi mesurer la distance entre points de l'image et détecter si il y a assez de place pour que le drone passe entre deux obstacles ou dans un trou. Pour la mise en place de cet algorithme nous avons réalisé une discréttisation de l'espace 3D en profondeur en prennent des couches dans l'axe de profondeur "z" dans lesquelles nous faisons une binarisation de l'image, et ainsi nous pouvons filtrer tous les obstacles qui sont entre deux valeurs de distance déterminées. Une fois que nous avons la binarisation nous pouvons designer le profil de notre drone dans l'image, avec les dimensions adaptés à la distance filtré dans la binarisation. Si dans ce profil nous n'avons pas des parties en rouge cela veut dire que notre drone peut passer, et pour trouver tous les chemins possibles nous pouvons parcourir toute l'image et vérifier toutes les parties où le drone entrerais. Le code est disponible dans l'annexe C.5.3.

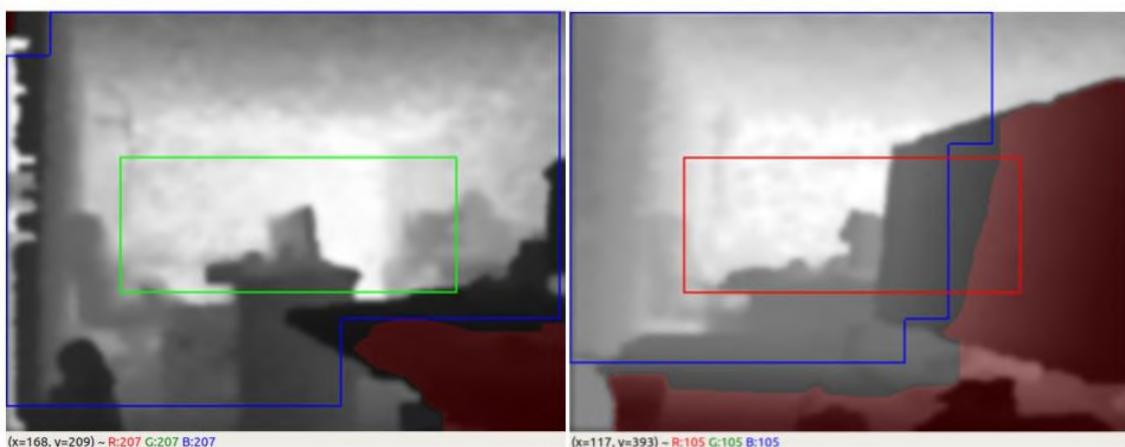


FIGURE 6.9 – Détection de chemin libre et obstacles à 1m

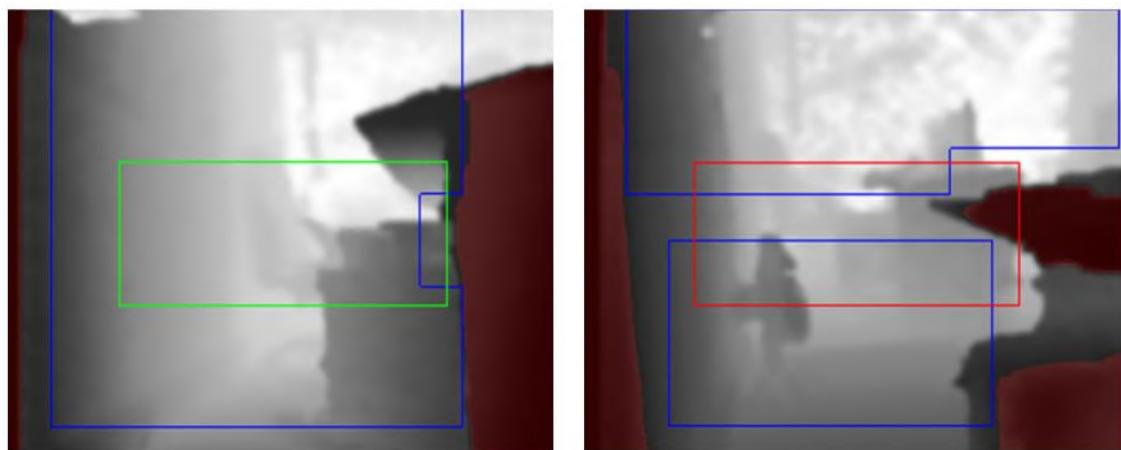


FIGURE 6.10 – Détection de chemin libre et obstacles à 1m

### Comportement attendu

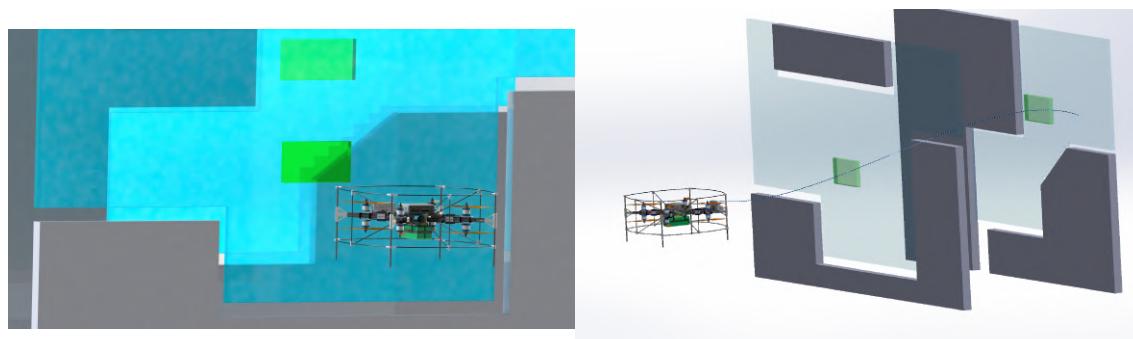


FIGURE 6.11 – Comportement attendu

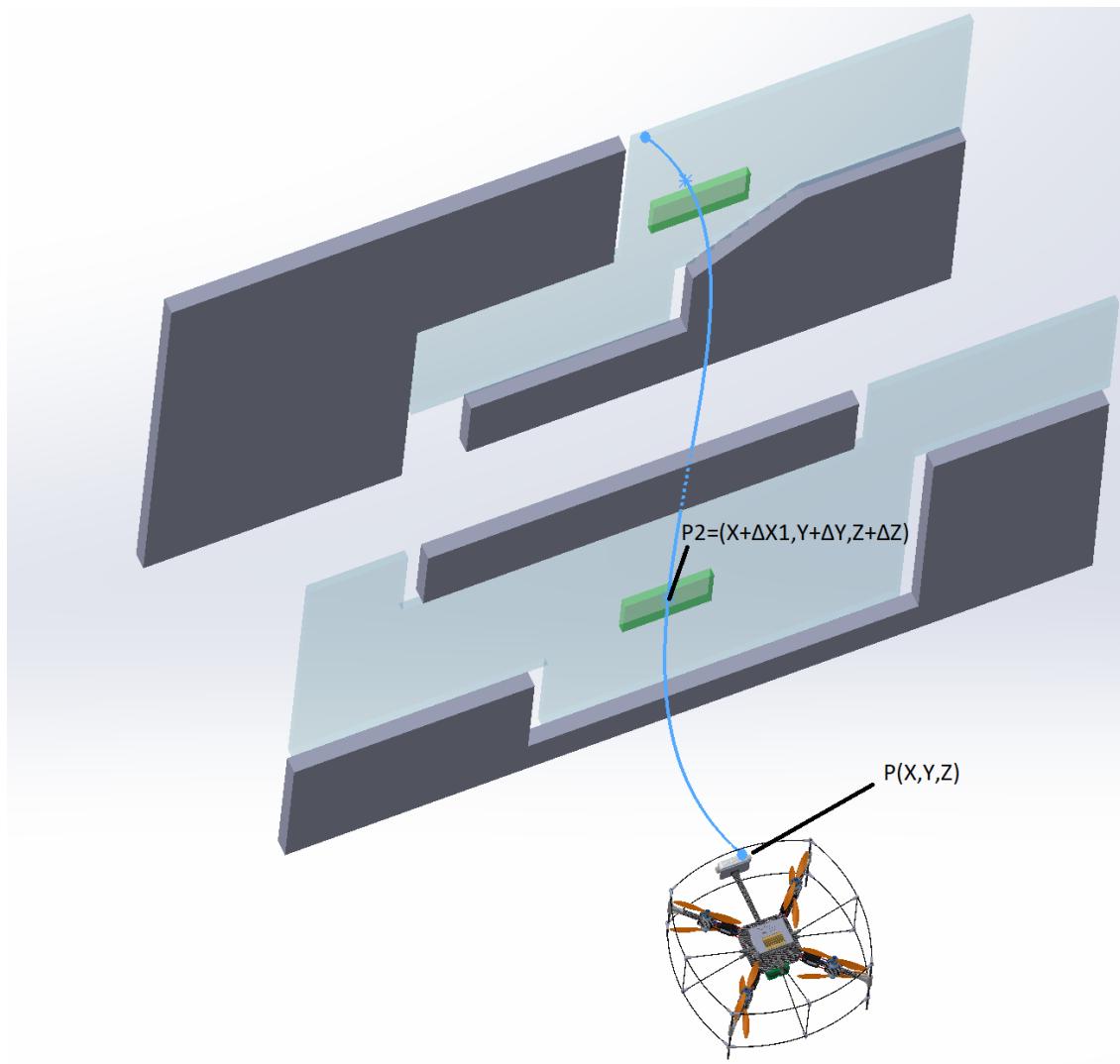


FIGURE 6.12 – Comportement attendu

Dans les figures 6.11 et 6.12 nous voyons une représentation 3D du système de évasion d'obstacles et planification de trajectoire. En gris nous voyons les obstacles, en bleue les zones libres ou le drone peut passer et en vert le chemin choisi pour chaque couche. Nous connaissons la position de notre cible par rapport au drone, donc nous pouvons générer une commande de déplacement correspondante.

# Chapitre 7

## Banc d'essai

Après d'avoir fait la reconception mécanique du drone, nous nous sommes dit que pour continuer à améliorer le dessin et vérifier la théorie appliquée nous avons besoin d'un banc d'essais qui nous permettra de caractériser le système de propulsion. **Comme ce projet se dévie un peu de l'objective principale nous nous avons mis un temps de développement maximal de 5 jours.**

### 7.1 Théorie

Pour faire un analyse énergétique complet les valeurs qui nous devons mesurer sont les suivantes :

- Courant d'entrée (I [A])
- Tension d'entrée (V [V])
- Vitesse du moteur (w [rad/s])
- Couple du moteur (T [Nm])
- Poussée (F [kgf])

Nous allons étudier les pertes dans le **sous système "A"** (Moteur + ESC) et du **système "B"** (Hélice) .

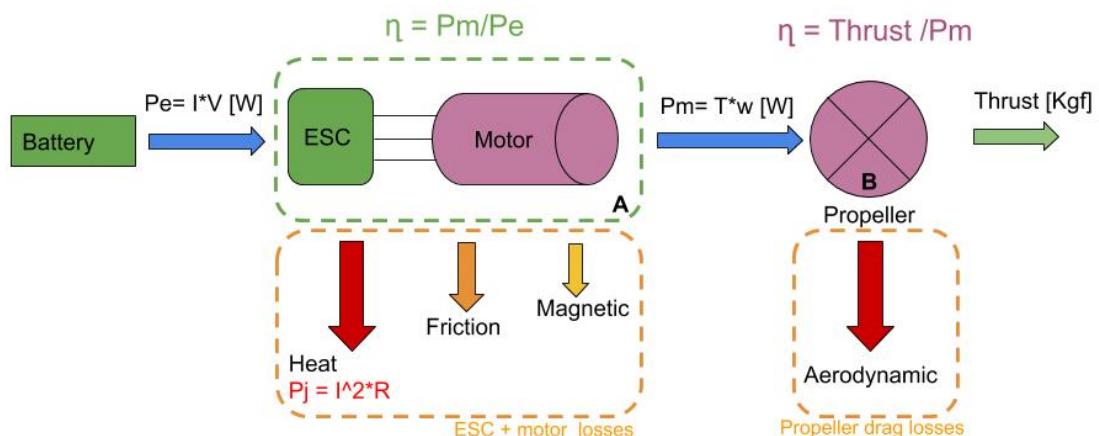


FIGURE 7.1 – Flux énergétique du système de puissance

### 7.1.1 Équations

$$\begin{aligned}
 P_e &= I * V = \text{Puissance électrique d'entrée} \\
 P_m &= T * w = \text{Puissance mécanique de sortie} \\
 P_p &= \text{Puissance perdue dans le sous système ESC + Moteur} \\
 P_{pA} &= P_{Joule} + P_{magnetic} + P_{motor friction} \\
 P_{pB} &= \text{Pertes aérodynamiques dans l'hélice} \\
 T_m &= \text{Couple du moteur} \\
 T_l &= \text{Couple de charge généré par l'hélice} \\
 F &= \text{Poussée générée par l'hélice}
 \end{aligned} \tag{7.1}$$

Et nous nous intéressons spécifiquement aux relations suivantes :

$$\eta_A = P_e - P_m \tag{7.2}$$

$$\eta_B = F/P_m \tag{7.3}$$

$$\eta_T = F/P_e \tag{7.4}$$

Pour la conception d'un drone un des premiers valeurs que nous devons déterminer c'est la poussée totale du drone, qui déterminera avec le poids des composants minimales la capacité de charge utile. Pour avoir un drone suffisamment réactif en vol, son poids total au décollage ne doit pas dépasser le 80% de sa capacité totale de charge. Sont ces valeurs qui nous serviront de guide pour le dimensionnement des moteurs et hélices nécessaires.

Si nous voulons maximiser le temps de vol, les moteurs doivent tourner à son régime de rendement maximal, et pour le connaître nous pouvons utiliser la courbe Rendement-Couple de charge. Nous pouvons aussi sacrifier un peu de rendement énergétique pour augmenter la charge maximale, en changeant par exemple des hélices bipale pour tripale.

Tous ces dimensionnements peuvent être vérifiés et modifiés avec les valeurs données par le banc d'essais.

## 7.2 Conception

### 7.2.1 Capteurs utilisés

Pour la mesure nous avons utilisé les capteurs suivantes :

- **Module BEC pour Pixhawk** : C'est un module utilisé dans le drone pour mesurer la courant consommé et tension de la batterie, ils nous fournissent deux valeurs analogiques et aussi une alimentation de 5V qui nous utilisons pour alimenter les autres composants électroniques.
- **Capteur de distance IR** : Nous utilisons ce capteur infrarouge et une bande adhésive réfléchissante pour mesurer les tours par minute du moteur.[10]
- **Capteurs de force** : Des jauge de déformation en configuration "Pont de Wheatstone" avec un amplificateur différentiel HX711[9]. Nous utilisons un capteur de 20 kgf pour la poussée et un de 2 kgf pour le couple du moteur.

### 7.2.2 Conception 3D

Nous avons désigné en 3D la structure d'un banc d'essais minimal pour des configurations simples mais il pourraient être modifiés facilement pour tester des configurations coaxiales ou avec des hélices canalisées.



FIGURE 7.2 – Conception 3D

### 7.2.3 Schéma des connections électriques

Les cartes électroniques ont été intégrées dans une plaque de prototypage pour les installer après dans un boîtier. Les capteurs de force sont connectés parmi des connecteurs JST-GH, la sortie PWM et l'entrée du tachymètre utilisent des connecteurs de servo "Hitec 'S'", l'alimentation utilise un connecteur XT-60 et la sortie d'alimentation pour l'ESC utilise un bornier.

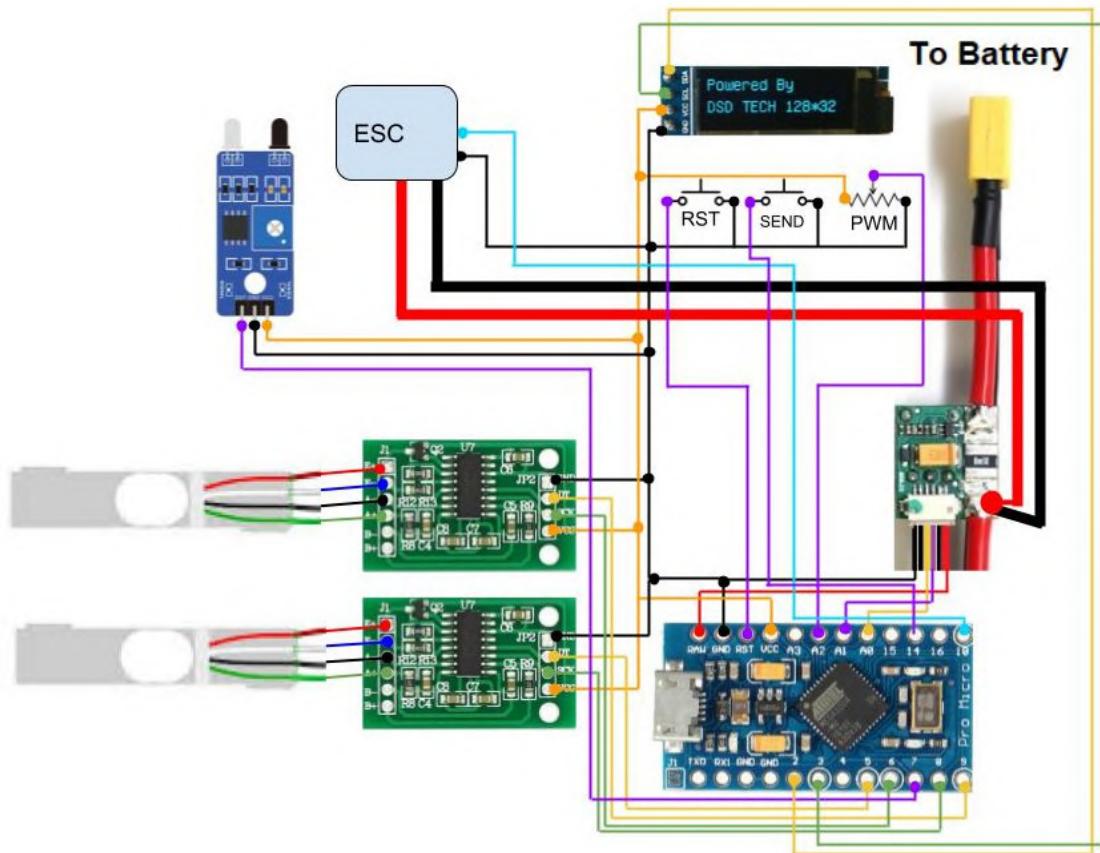


FIGURE 7.3 – Schéma des connections électriques

#### 7.2.4 Produit final

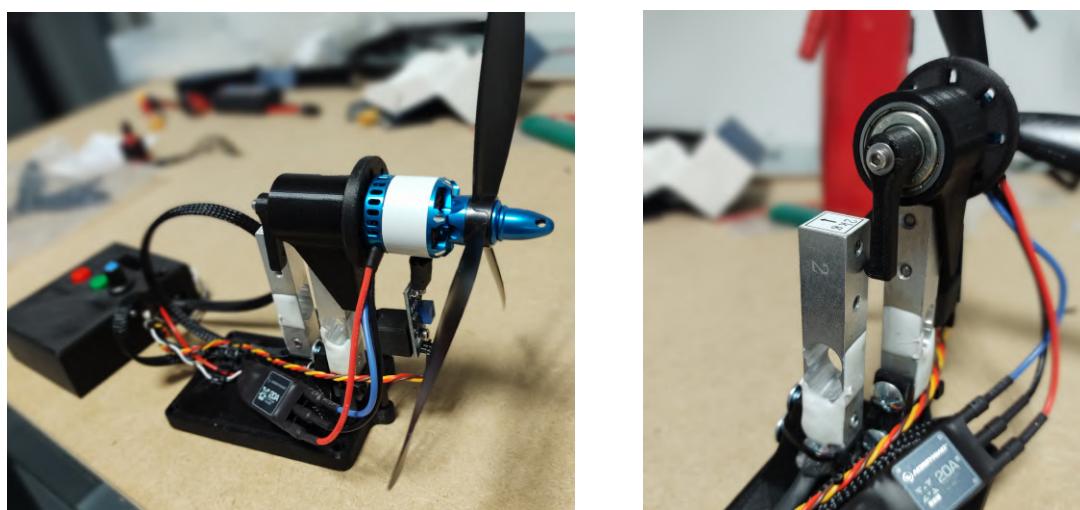


FIGURE 7.4 – Banc d'essais

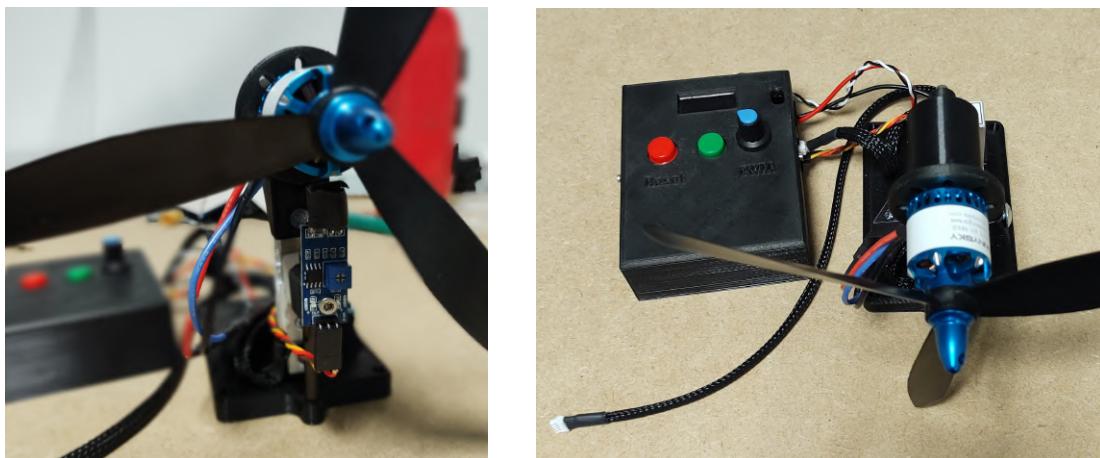


FIGURE 7.5 – Banc d'essais

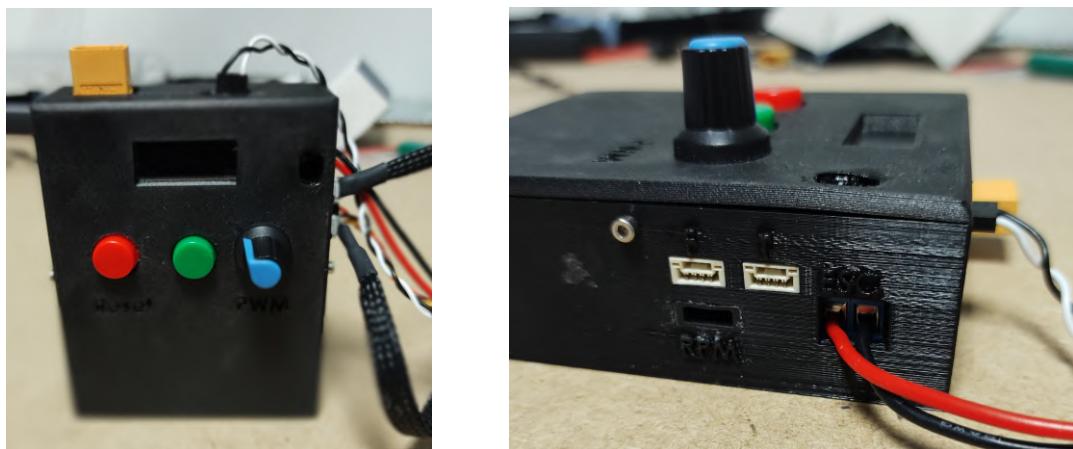


FIGURE 7.6 – Contrôleur



FIGURE 7.7 – Interface du contrôleur

Dans la figure 7.7 nous voyons une description des informations affichées dans l'écran ainsi que une démonstration du système de sécurité qui empêche au moteur de démarrer quand nous allumons le contrôleur avec une consigne de PWM non nulle.

### 7.3 Essai de démonstration

Nous avons fait un essai de rendement énergétique avec les moteurs et hélices utilisés dans notre drone. L'essai a été fait en envoyant plusieurs valeurs pour chaque consigne PWM parmi le bouton vert, qui envoie des valeurs par port série et ils sont récupérés parmi un script python qui les enregistre dans un fichier Comma Separated Value (csv). Une fois que nous avons récupéré toutes les mesures nous pouvons les filtrer et faire un analyse énergétique . Nous avons utilisé le mode manuel car le mode automatique ne marche pas pour l'instant à cause d'un problème présenté dans la section 7.4.1. C'est test a comme objectif démontrer les possibilités de ce système et les valeurs et graphiques sont données à titre indicatif. Nous avons pris plusieurs échantillons correspondants à différentes consignes PWM. Les données récupérées en format csv ont été traités et analysées dans un tableau Excel.

#### Résultats

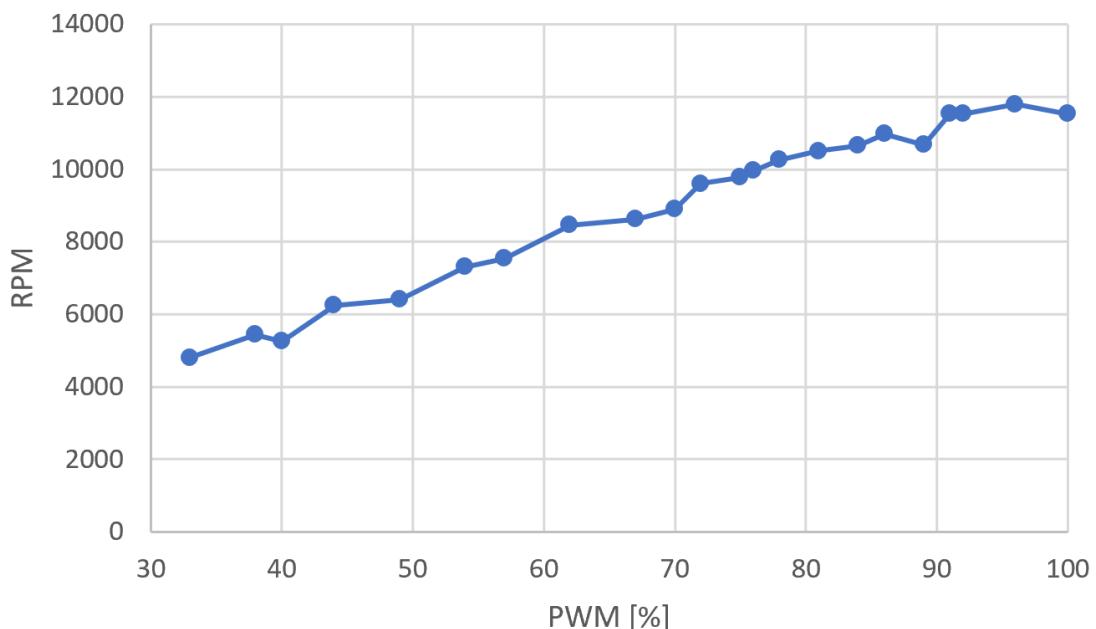


FIGURE 7.8 – Relation linéaire entre la consigne PWM et la vitesse

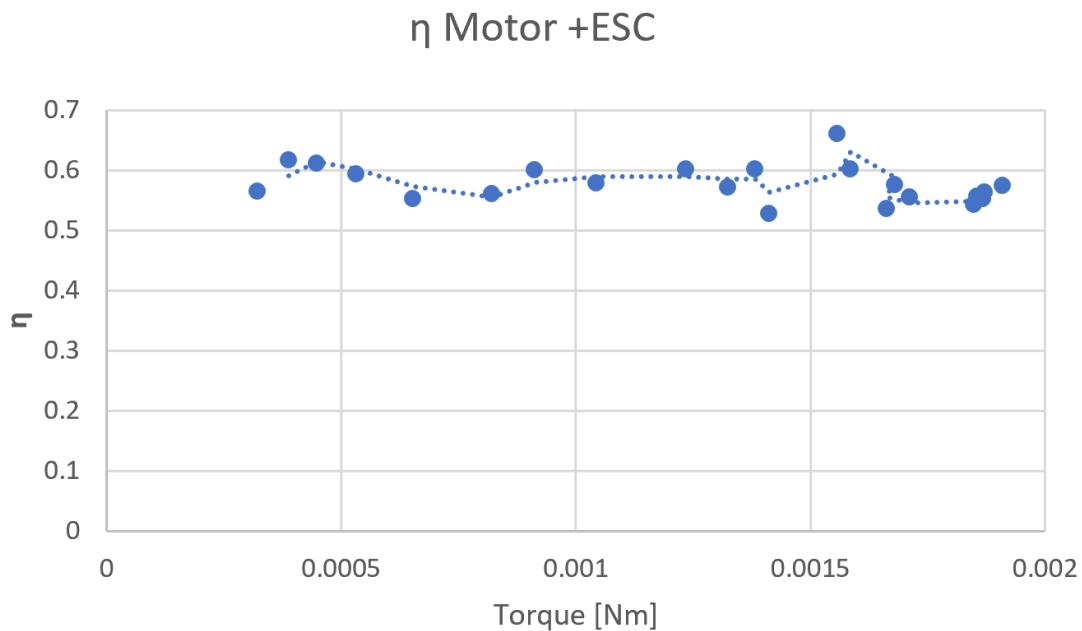


FIGURE 7.9 – Rendement général

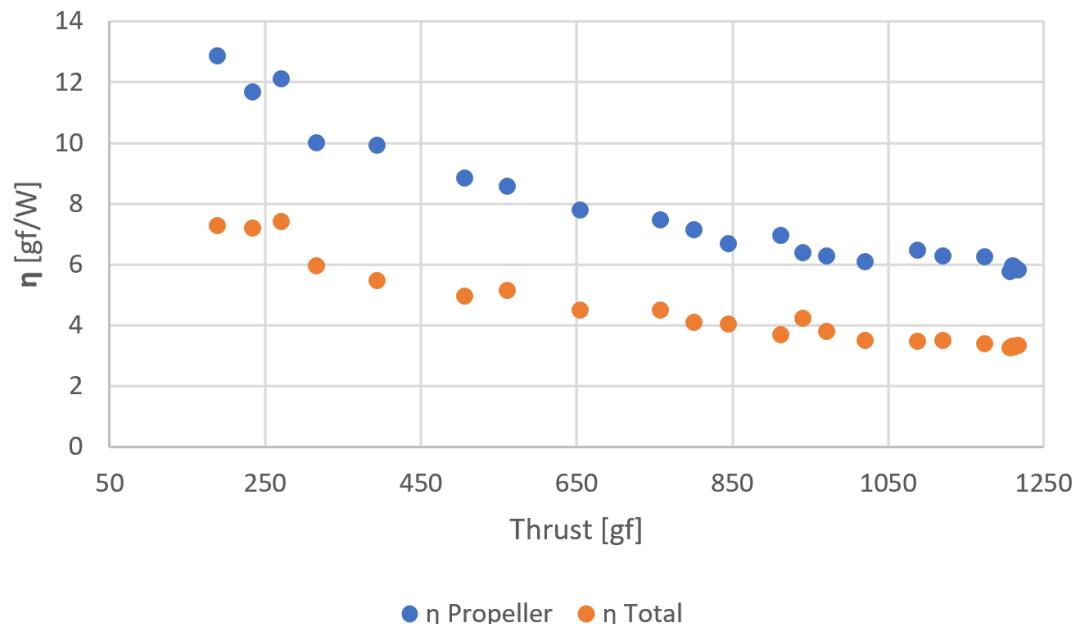


FIGURE 7.10 – Rendement de l'hélice

## 7.4 Problèmes rencontrés

### 7.4.1 Temporisation

Nous avons utilisé comme microcontrôleur un ATMega32U4 car il était disponible au moment de faire la carte électronique, mais au moment de programmer toutes les fonctionnalités nous avons trouvé un problème. Ce microcontrôleur a un seul horloge disponible et nous avons plusieurs tâches qui ont besoin d'une temporisation

comme la génération de la signal PWM ou la mesure de vitesse, cela génère des distorsions dans le signal PWM en produisant des changements de vitesse brusques périodiquement et nous empêche d'utiliser le mode automatique.

### 7.4.2 Problèmes mécaniques

Le dessin utilisé génère des efforts transversales dans le capteur de force qui mesure la poussée générée et les données peuvent fausser les mesures.

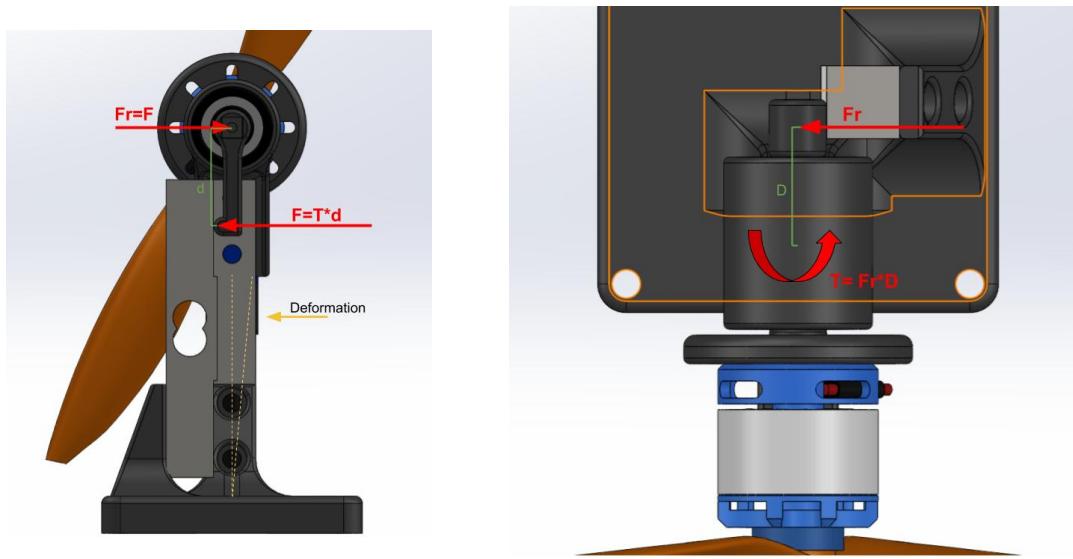


FIGURE 7.11 – Problèmes mécaniques

# Chapitre 8

## Conclusion

Nous avons commencé cet travail de recherche et développement avec l'idée d'avancer le plus possible sur la navigation sans GPS dans un drone, avec une feuille de route très floue et un objectif très ambitieux : le vol autonome dans des endroits fermés.

Nous avons choisi d'essayer une solution relativement développée avec des composants disponibles dans le marché comme la caméra T265 pour avoir un produit ou démonstrateur technologique disponible à court terme avec des ressources limitées (une seule personne dans le développement). Après 6 mois de travail, nombreux essais et beaucoup d'apprentissage je peux conclure que pour le développement d'un drone avec les compétences nécessaires pour le vol automatique il faudrait travailler à bas niveau, cela veut dire faire une intégration de plusieurs capteurs parmi un filtre de Kalman ou une autre stratégie de commande mais toujours en comprennent le fonctionnement de la boucle principale du drone.

Personnellement, j'ai vécu cet stage comme une expérience très enrichissante car j'ai eu l'opportunité d'apprendre des nouveaux outils qui vont me servir comme ingénieur et aussi dans la vie quotidienne. A continuation je vous présente une petite liste avec les principales compétences acquises ou améliorées :

- Gestion des projets
- Conception d'un produit
- Fonctionnement générale d'un drone
- ROS
- Python
- Conception et fabrication 3D
- Machine Vision et OpenCV
- Pilotage des drones

## 8.1 Diagramme de Gantt

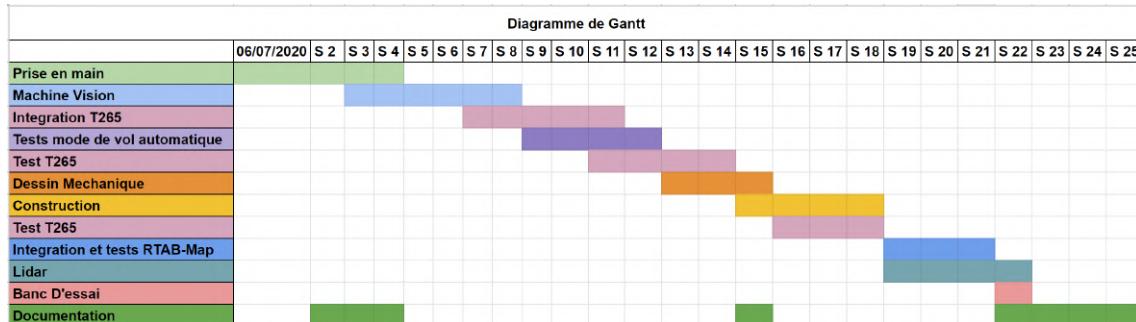


FIGURE 8.1 – Distribution temporelle des travaux réalisés

## 8.2 Travail future

A continuation je présente quelques idées que, à mon avis, il faudrait réaliser pour la continuation de ce projet :

- **Construction d'un banc d'essai générique** : Cet outil permettrait de trouver le dessin optimal pour chaque application d'un drone, donc il serait utile pour tous les projets de l'entreprise.
- **Intégration des multiples capteurs** : Création d'un node ROS personnalisé pour l'intégration de plusieurs caméras T265.
- **Création d'une stratégie de commande en bas niveau** : Pour avoir une meilleure compréhension du comportement du drone il faudra générer une stratégie de commande adapté à certains scénarios spécifiques. Une combinaison des capteurs sera nécessaire pour percevoir l'environnement.

## 8.3 Rapport d'émerveillement

**L'ambiance sur le lieu travail** : Il y a une atmosphère agréable car tout le monde aime son travail et s'intéresse à amener au bout chaque projet. Les responsables sont toujours disponibles pour discuter et écouter tes idées.

**L'accueil et l'intégration** : Lors de mon premier jour dans l'entreprise certains détails m'ont difficilement l'adaptation, comme ne pas avoir un ordinateur disponible au moment de commencer ou un projet bien défini.

**Le poste en lui-même, environnement et moyens mis à disposition** : L'endroit est spacieux et pratique pour le travail car il y a de la place pour voler et tester les drones. Par rapport aux moyennes mis en disposition le support donne a été remarquable, j'ai eu accès à tous les matériaux nécessaires pour réaliser mon travail confortablement. J'ai reçu aussi un ordinateur portable que j'avais pendant tout le stage et que je pouvais emporter chez moi si j'en avais besoin.

**Le processus de recrutement** : C'était un processus rapide mais pas trop clair par rapport au déroulement du stage, objectifs et conditions du travail.

De manière générale, ce fut une expérience très positive avec beaucoup d'apprentissage et vraiment motivante et je remercie d'avoir eu ma première expérience du travail dans une petite entreprise (7 salariés) où le fonctionnement de la même est

très clair et accessible. Je suis très content aussi d'avoir pu gérer la totalité de mon projet avec un liberté créative absolue .

# Glossaire

**brushless** Un moteur sans balais, ou « moteur brushless », ou machine synchrone auto-pilotée à aimants permanents, est une machine électrique de la catégorie des machines synchrones<sup>1</sup>, dont le rotor est constitué d'un ou de plusieurs aimants permanents et pourvu d'origine d'un capteur de position rotorique. 11

**callback** En informatique, une fonction de rappel (callback en anglais) ou fonction de post-traitement est une fonction qui est passée en argument à une autre fonction. Cette dernière peut alors faire usage de cette fonction de rappel comme de n'importe quelle autre fonction, alors qu'elle ne la connaît pas par avance. 55

**firmware** Dans un système informatique, un firmware (ou micrologiciel, micro-code, logiciel interne, logiciel embarqué ou encore microprogramme) est un programme intégré dans un matériel informatique (ordinateur, photocopieur, automate (API, APS), disque dur, routeur, appareil photo numérique, etc.) pour qu'il puisse fonctionner. 14

**mavproxy** MAVProxy est un GCS entièrement fonctionnel pour les UAV, conçu comme un GCS minimaliste, portable et extensible pour tout système autonome prenant en charge le protocole MAVLink (comme celui utilisant ArduPilot).. 51

**Mavros** Mise en place du protocole Mavlink dans ROS qui permet une connexion proxy avec le GCS. 26

**OpenCV** (pour Open Computer Vision) est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel. 53

**Pipeline** Processus de traitement. 9, 55

**Rviz** Outil de visualisation 3D utilisé par ROS. 27

**Wrapper** En programmation informatique, une fonction wrapper (de l'anglais « wrapper function ») est un programme dont la fonction principale est d'appeler une autre fonction. 54

# Acronyms

**API** Application programming Interface. 26

**csv** Comma Separated Value. 67

**CUDA** Compute Unified Device Architecture. 54

**DOF** Degrees Of Freedom. 26

**FCU** Flight Controller Unit. 11

**GCS** Ground Control Station. 15

**GPS** Global Positioning System. 15

**IMU** Inertial Measurement Unit. 11

**Mavlink** Micro Air Vehicle Link. 15

**PWM** Pulse With Modulation. 63

**ROS** Robot Operating System. 26

**RUD** Rapid Unscheduled Dissassembly. 24

**V-SLAM** Visual Simultaneous Localization And Mapping. 26

**VPU** Visual Processing Unit. 26

# Bibliographie

- [1] @KARAAGE0703. *Jetson Nano Realsense Python-PyRealsense*. 2020. URL : <https://qiita.com/karaage0703/items/39040f23cbb0611c0a0e> (visité le 19/01/2020).
- [2] Inês Silva AMADO. « Experimental Comparison of Planar and Coaxial Rotor Configurations in Multi-rotors ». Thèse de doct. Técnico Lisboa, 2017.
- [3] ARDUPILOT. *Installing MAVROS*. 2020. URL : <https://ardupilot.org/dev/docs/ros-install.html#installing-mavros> (visité le 24/08/2020).
- [4] ARDUPILOT. *ROS and VIO tracking camera for non-GPS Navigation*. 2020. URL : <https://ardupilot.org/dev/docs/ros-vio-tracking-camera.html> (visité le 24/08/2020).
- [5] ARDUPILOT. *Simple Object Avoidance*. 2020. URL : <https://ardupilot.org/copter/docs/common-simple-object-avoidance.html> (visité le 10/12/2020).
- [6] ARDUPILOT. *SITL Simulator (Software in the Loop)*. 2020. URL : <https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html> (visité le 11/12/2020).
- [7] Andrea BELLONI. *Mavros Control*. 2019. URL : [https://github.com/anbello/aruco\\_gridboard/tree/master/script](https://github.com/anbello/aruco_gridboard/tree/master/script) (visité le 11/12/2020).
- [8] SparkFun ELECTRONICS. *QwiicVL53L1XPy*. 2020. URL : <https://qwiic-vl53l1x-py.readthedocs.io/en/latest/> (visité le 10/12/2020).
- [9] ELECTROPEAK. *Build a Digital Weight Scale Force Guage W/ Arduino Load-cell*. 2020. URL : <https://www.instructables.com/Build-a-Build-a-Digital-Weight-Scale-Force-Guage/> (visité le 16/12/2020).
- [10] GREATSCOTTLAB. *DIY Tachometer (RPM Meter)*. 2020. URL : <https://www.instructables.com/DIY-Tachometer-RPM-Meter/> (visité le 16/12/2020).
- [11] HOLYBRO. *Pixhawk Holybro*. 2020. URL : <https://docs.px4.io/> (visité le 27/11/2020).
- [12] INTEL. *Post-processing filters*. 2018. URL : <https://dev.intelrealsense.com/docs/post-processing-filters> (visité le 14/12/2020).
- [13] INTEL. *rs-tracking-and-depth Sample*. 2020. URL : <https://github.com/IntelRealSense/librealsense/tree/master/examples/tracking-and-depth> (visité le 12/03/2020).
- [14] INTROLAB. *RTAB-Map ROS installation*. 2020. URL : [https://github.com/introlab/rtabmap\\_ros](https://github.com/introlab/rtabmap_ros) (visité le 12/03/2020).

- 
- [15] KANGALOW. *Jetson Nano – Even More Swap*. 2019. URL : <https://www.jetsonhacks.com/2019/11/28/jetson-nano-even-more-swap> (visité le 24/08/2020).
  - [16] KANGALOW. *OpenCV 4 + CUDA on Jetson Nano*. 2019. URL : <https://www.jetsonhacks.com/2019/11/22/opencv-4-cuda-on-jetson-nano> (visité le 24/08/2020).
  - [17] Stereo LABS. *Getting Started with ROS on Jetson Nano*. 2020. URL : <https://www.stereolabs.com/blog/ros-and-nvidia-jetson-nano> (visité le 24/08/2020).
  - [18] MATHIEULABBE. *RGB-D Handheld Mapping*. 2020. URL : [http://wiki.ros.org/rtabmap\\_ros/Tutorials/HandHeldMapping](http://wiki.ros.org/rtabmap_ros/Tutorials/HandHeldMapping) (visité le 30/07/2020).
  - [19] Wim MEEUSSEN. *Coordinate Frames for Mobile Platforms*. 2010. URL : <https://www.ros.org/reps/rep-0105.html> (visité le 12/07/2020).
  - [20] ST MICROELECTRONICS. *Time-of-Flight ranging sensor based on ST’s Flight-Sense technology*. 2020. URL : <https://www.st.com/en/imaging-and-photonics-solutions/vl53l1x.html> (visité le 10/12/2020).
  - [21] Thien NGUYEN. *Integration of ArduPilot and VIO tracking camera (Part 2) : Complete installation and indoor non-GPS flights*. 2019. URL : <https://discuss.ardupilot.org/t/integration-of-ardupilot-and-vio-tracking-camera-part-2-complete-installation-and-indoor-non-gps-flights/43405> (visité le 24/08/2020).
  - [22] NOMACHINE. *Tips for using NoMachine on NVIDIA Jetson Nano*. 2020. URL : <https://www.nomachine.com/AR02R01074> (visité le 01/09/2020).

# Annexe A

## Environment setup

The following list indicates the packages that we must install if we don't have access to the "ready to go" image :

- Real Sense SDK
- OpenCV
- ROS + packages
- pymavlink

### A.1 Real Sense SDK

The following section is based on the installation guide found at the Qiita website.  
[1]

*Cmake* should be installed by default, we can verify it running :

```
1 $ sudo apt-get update
2 $ sudo apt-get install -y cmake
```

Then we have to get the source files from the GitHub repository :

```
1 $ cd && git clone https://github.com/IntelRealSense/
2 librealsense.git
3 $ cd librealsense
4 $ mkdir build
5 $ cd build
```

Before proceeding to compile we have to add the CUDA compiler path to the *bashrc* file as we will need it for the next step :

```
1 $sudo gedit ~/.bashrc
```

and add to the end of the file the following lines :

```
1 export CUDA_HOME=/usr/local/cuda
2 export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64:/
3 /usr/local/cuda/extras/CUPTI/lib64
4 export PATH=$PATH:$CUDA_HOME/bin
```

Next, we have to set the flags for the *cmake* compilation, here we are going to specify some compilation parameters to build with *CUDA* support and to build the *Python* bindings.

```
1 $cmake ../ -DBUILD_PYTHON_BINDINGS:STRING=true
2 $cmake ../ -DBUILD_WITH_CUDA:BOOL=ON
```

If you get an error like "Xinerama" or "xcursor" not found you have to install them :

```
1 $sudo apt-get install libxcursor-dev libxkbcommon-x11-dev
2 $sudo apt-get install libxinerama1 libxinerama-dev
```

to see all the available Cmake flags :

```
1 $cmake -LA | awk '{if(f)print} /-- Cache values/{f=1}'
```

finally we can proceed to compile with the following command :

```
1 $make -j1
```

the -j1 option is to only use one core of the processor, if we use more the jetson will freeze.

The last step is to put the compiled python library shared object in the correct directory :

```
1 $cd ~/librealsense/build/wrappers/python
2 $sudo cp pyrealsense2.so /usr/local/lib/python2.7/dist-packages
```

Now we can test it by running a Python example from the wrappers folder :

```
1 $cd ~/librealsense/wrappers/python/examples
2 $python align-depth2color.py
```

## A.2 OpenCV + CUDA support

The following section is based on the [15] and [16] articles of the Jetson Hacks forum.

We need to change the amount of swap memory to accelerate the OpenCV compilation. *"The Jetson Nano by default has 2GB of swap memory. The swap memory allows for "extra memory" when there is memory pressure on main (physical) memory by swapping portions of memory to disk."* [15]

You can examine the swap memory information :

```
1 $ zramctl
```

On the JetsonHacksNano account on Github, there is a repository named resizeSwapMemory. The repository contains a convenience script to set the size of the swap memory.

```
1 $ git clone https://github.com/JetsonHacksNano/resizeSwapMemory
2 $ cd resizeSwapMemory
```

To use the script in the repository :

```
1 $ git clone https://github.com/JetsonHacksNano/resizeSwapMemory
2 $ cd resizeSwapMemory
```

Example usage :

```
1 $ ./setSwapMemorySize.sh -g 4
```

will set the entire swap memory size to 4GB. You will need to reboot for the change to take effect.

Now lets build OpenCV :

```
1 $ git clone https://github.com/JetsonHacksNano/buildOpenCV
2 $ cd buildOpenCV
3 $ ./buildOpenCV.sh |& tee openCV_build.log
```

The last command will build OpenCV, and send the results of the build to the file *openCV\_build.log*

To check if OpenCV was successfully installed with CUDA support we can run the following lines :

```
1 $python
```

Then importing opencv :

```
1 >>import cv2
2 >>print(cv2.getBuildInformation())
```

and look for CUDA and related GPU information.

Now that OpenCV was successfully installed we can reconfigure the swap memory to the original size :

```
1 $ cd buildOpenCV
2 $ ./setSwapMemorySize.sh -g 2
```

## A.3 ROS Melodic

The following section is based on the stereolabs tutorial [17]

Set up the Jetson Nano to accept software from packages.ros.org :

```
1 $ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list
,'
```

Add a new apt key :

```
1 $sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80'
--recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

1 $ sudo apt update
2 $ sudo apt install ros-melodic-desktop
```

Initialize rosdep. This enables you to easily install system dependencies for source code you want to compile and is required to run some core components in ROS :

```
1 $ sudo rosdep init
2 $ rosdep update
```

It is recommended to load the ROS environment variables automatically when you execute a new shell session. Update your .bashrc script :

```
1 $ echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
2 $ source ~/.bashrc
```

### A.3.1 Configure a catkin workspace

To start running your own ROS packages or install other packages from source (such as the vision\_to\_mavros module), you must create and configure a catkin workspace. Install the following dependencies :

```
1 $ sudo apt-get install cmake python-catkin-pkg python-empy
  python-nose python-setuptools libgtest-dev python-rosinstall
  python-rosinstall-generator python-wstool build-essential git
```

Create the catkin root and source folders :

```
1 $ mkdir -p ~/catkin_ws/src
2 $ cd ~/catkin_ws/
```

It is recommended using caktin\_tools instead of the default catkin\_make as it is more powerful

```
1 $ sudo apt-get install python-catkin-tools
```

Configure the catkin workspace by issuing a first “empty” build command :

```
1 $ catkin build
```

Finally, update your .bashrc script with the information about the new workspace :

```
1 $ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
2 $ source ~/.bashrc
```

### A.3.2 RealSense ROS

```
1 $ export ROS_VER=melodic
2 $ sudo apt-get install ros-$ROS_VER-realsense2-camera
```

### A.3.3 MAVROS

The following section is based on the Ardupilot wiki page [3]

```
1 $sudo apt-get install ros-melodic-mavros ros-melodic-mavros-
  extras
2 $wget https://raw.githubusercontent.com/mavlink/mavros/master/
  mavros/scripts/install_geographiclib_datasets.sh
3 $chmod a+x install_geographiclib_datasets.sh
4 $./install_geographiclib_datasets.sh
```

For ease of use on a desktop computer, please also install RQT

```
1 $sudo apt-get install ros-melodic-rqt ros-melodic-rqt-common-
  plugins ros-melodic-rqt-robot-plugins
```

### A.3.4 vision\_to\_mavros

The following section is based on an article from ardupilot wiki.[4]

To clone and build the package, navigate to catkin workspace :

```
1 $cd ~/catkin_ws/src
2 $git clone https://github.com/hoangthien94/vision_to_mavros.git
3 $cd ..
4 $catkin build
5 $source ~/.bashrc
```

## Possible errors

If *catkin build* gives opencv bridge error, you have to change the include lib path

```
1 $ sudo gedit /opt/ros/melodic/share/cv_bridge/cmake/
  cv_bridgeConfig.cmake
```

and change */usr/include/opencv* to */usr/include/opencv4*

If *catkin build* gives error : ‘*CV\_CALIB\_ZERO\_DISPARITY*’ was not declared in this scope we have to edit the file :

```
1 $ sudo gedit /home/jetson/catkin\_ws/src/vision\_to\_mavros/src/
  t265\_fisheye\_undistort.cpp
```

And add this lines :

```
1 #include <opencv2/calib3d/calib3d.hpp>
2 #include <opencv2/calib3d.hpp>
3 #include <opencv2/calib3d/calib3d_c.h>
```

## A.4 pymavlink

To be able to send Mavlink commands to the FCU we need the pymavlink library.

```
1 $ sudo apt-get install libxml2-dev libxslt-dev python-dev
2 $ pip install pymavlink
```

### A.4.1 RTAB-Map

To install RTAB-map ROS we can simply use the following command :

```
1 $ sudo apt install ros-melodic-rtabmap-ros
```

But as explained in the Introlab GitHub repository [14], if we want the support of OpenCV built for Tegra on the Jetson, we could build the package from source following their instructions.

### A.4.2 SITL

To have Ardupilot’s SITL running we need to set up the build environment. We are going to clone the Ardupilot repository in the home directory.

```
1 $cd
2 $git clone https://github.com/ArduPilot/ardupilot.git
3 $cd ardupilot
4 $git submodule update --init --recursive
```

There is a script in the folder that installs the required dependencies :

```
1 Tools/environment_install/install-prereqs-ubuntu.sh -y
```

Reload the path (log-out and log-in to make permanent) :

```
1 $ . ~/profile
```

## Build Arducopter

```
1 $ ./waf configure --board=linux
2 $ ./waf
```

## A.5 Test environment

We have 4 features to test :

- ROS RealSense
- Serial communication
- TCP telemetry bridge
- vision\_to\_mavros node
- pymavlink

I recommend to follow the testing procedure detailed in this ardupilot blog post[21].

The first step is to configure our FCU connecting to it via Mission Planner.

```
1
2 # For EKF2
3
4 AHRS_EKF_TYPE = 2
5
6 EK2_ENABLE = 1
7
8 EK3_ENABLE = 0
9
10 EK2_GPS_TYPE = 3
11
12 EK2_POSNE_M_NSE = 0.1
13
14 EK2_VELD_M_NSE = 0.1
15
16 EK2_VELNE_M_NSE = 0.1
17
18 # Similar for both EKFs
19
20 BRD_RTC_TYPES = 2
21
22 GPS_TYPE = 0
23
24 COMPASS_USE = 0
25
26 COMPASS_USE2 = 0
27
28 COMPASS_USE3 = 0
29
30 SERIAL1_BAUD = 57600 (the serial port used to connect to the Jetson
   Nano)
31
32 SERIAL1_PROTOCOL = 1
33
34 SYSID_MYGCS = 1 (to accept control from mavros)
```

then we can proceed to configure the `apm.launch` file from MAVROS

```
1 $ cd/opt/ros/melodic/share/mavros/launch
2 $ sudo gedit apm.launch
```

and change the *fcu\_url* to `/dev/ttyTHS1:57600` and add to *gcs\_url* `tcp-1://YOUR IP` then we can run each node separately to check them :

### A.5.1 ROS RealSense

```
1 $ roslaunch realsense2_camera rs_t265.launch
2 $ rostopic -hz /camera/odom/sample/
```

The topic `/camera/odom/sample/` and `/tf` should be published at 200Hz.

### A.5.2 Serial communication

MAVROS node :

```
1 $ roslaunch mavros apm.launch
```

We may have to give permission o the serial port with :

```
1 $ systemctl stop nvgetty
2 $ systemctl disable nvgetty
3 $ udevadm trigger
4 $ sudo chmod 666 /dev/ttyTHS1
5 $ sudo reboot
```

and if the connection is working, the communication data will be prompted in the terminal. We can also check that `/mavros/state` should show that FCU is connected.

```
1 $ rostopic echo /mavros/state
```

### A.5.3 TCP Telemetry bridge

In a device connected to the same LAN launch Mission Planner (or QGround Control) and start a new TCP connection to the ip address specified in section A.5 If everithing works fine we should see the new connection in the terminal and we should receive all the telemetry values on the GCS software.

### A.5.4 Vision\_to\_mavros

```
1 $ roslaunch vision_to_mavros t265_tf_to_mavros.launch
2 $ rostopic echo /mavros/vision_pose/pose
```

And in another terminal :

```
1 $ rostopic echo /mavros/vision_pose/pose
2 $ rostopic hz /mavros/vision_pose/pose
```

We should see the pose data from the T265 and that the topic is being published at 30Hz.

### Verify that ArduPilot is receiving data from MAVROS

Check that ArduPilot is receiving position data by viewing the topic `VISION_POSITION_ESTIMATE` on GCS. For Mission Planner, press `Ctrl+F` and click on “Mavlink Inspector”, you should be able to see data coming in.

### A.5.5 pymavlink

To work properly, the FCU needs to know its position as Latitude and longitude values, and we can send them via mission planner or via the *set\_origin.py* script situated in the *vision\_to\_mavros* folder.

```
1 $ rosrun vision\_\_to\_\_mavros set\_origin.py
```

Later, we can modify it to send the coordinates of our choice.

## A.6 All nodes together

If everything is working, we can launch all the nodes together (the *set\_position* script is not a node) with the following line :

```
1 $ vision\_to\_mavros t265\_all\_nodes.launch
```

## A.7 RTAB-Map

To launch a basic test of the system we will follow the tutorial from the ROS Wiki [18].

First of all we need to launch our cameras with the ROS RealSense API so we can get the right ROS topics published.

```
1 $ roslaunch realsense2_camera rs_d400_and_t265.launch
```

Once that we have that running we can start the RTAB-Map module in mapping mode (A localization mode based on an already existent map is available and detailed in the tutorial).

Because of compatibility problems with the Jetson Nano we can only use Rviz as visualizer and not rtabmapviz, to configure this we have to modify the launch file **rtabmap.launch**, set **rviz :=true** and **rtabmapviz :=false**

```
1 $ sudo gedit opt/ros/melodic/share/rtabmap_ros/launch
```

Then we can run the following command to start RTAB-Map :

```
1 $ roslaunch rtabmap_ros rtabmap.launch \
2 args:="-d --Mem/UseOdomGravity true --Optimizer/GravitySigma 0.3
" \
3 odom_topic:=/t265/odom/sample \
4 frame_id:=t265_link \
5 rgbd_sync:=true \
6 depth_topic:=/d400/aligned_depth_to_color/image_raw \
7 rgb_topic:=/d400/color/image_raw \
8 camera_info_topic:=/d400/color/camera_info \
9 approx_rgbd_sync:=false \
10 visual_odometry:=false
```

We can experiment with the different visualization options available in the *Displays* menu of Rviz

### A.7.1 SITL

Once that we have compiled Ardupilot as explained in the section A.4.2 we can proceed to run the simulator with the following commands :

```
1 $ cd/ardupilot/Tools/autotest  
2 $ sim_vehicle.py -v ArduCopter --map --osd --console
```

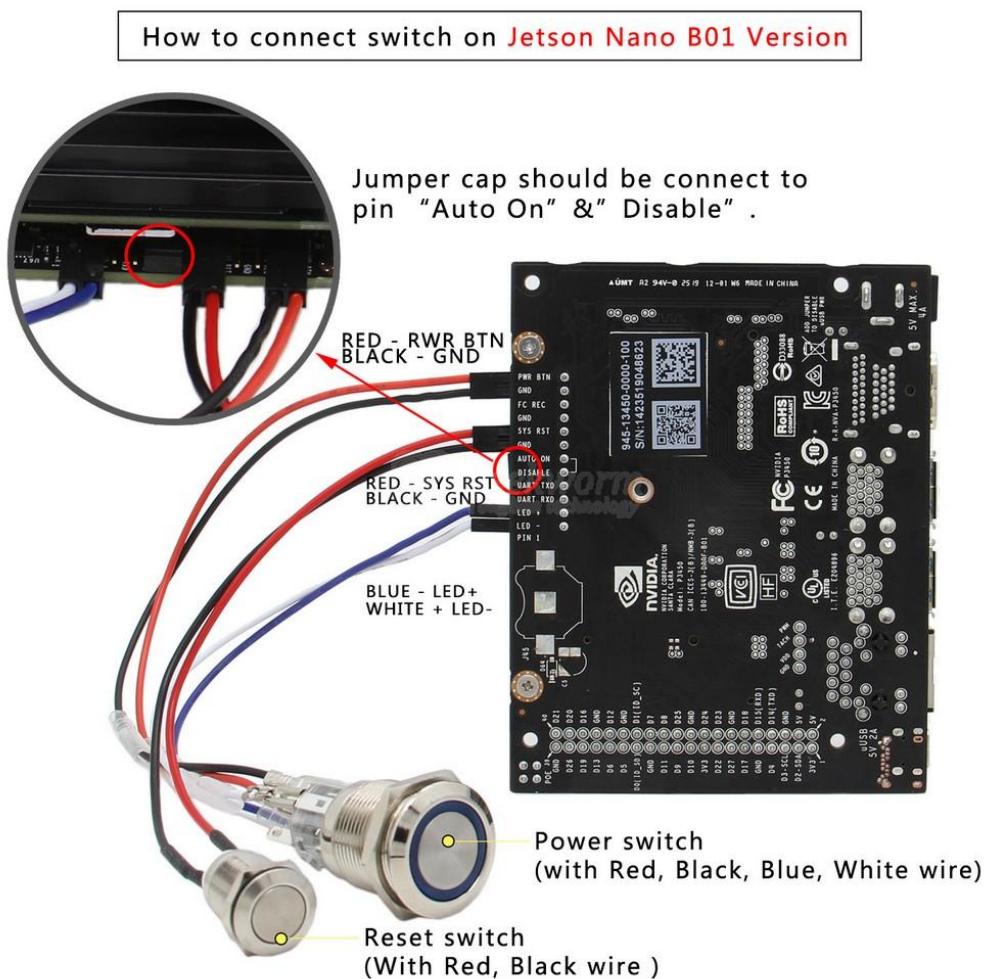
Pour se connecter avec un GCS ou avec Mavros nous devons utiliser un des ports de sortie UDP ouverts par Mavproxy au moment de lancer le simulateur.

# Annexe B

## Optional steps

### B.1 On/Off button

We can install a temporary switch to turn on and off the Jetson Nano easily following the instructions from this image :



By default, pushing the power button in Ubuntu will open a shutdown menu with many options. If we want to set the power down option we have to run the following lines in a terminal :

```
1 $ hostnamectl set-chassis vm
2 $ gsettings set org.gnome.settings-daemon.plugins.power button-
power 'shutdown'
```

## B.2 Remote desktop

To acces to the jetson nano while mounted on the done, we will need a remote descktop client. We are goint to use NoMachine as it is fast and easy to install. This software will allow us to connect over WLAN but if we want a connection over the internet (4G mobile connection) we will need a VPN server. We will follow the instructions from the official website [22].

Go to the NoMachine website and download the package for for ARMv8 (arm64-aarch64) from : <https://www.nomachine.com/download/download&id=111&s=ARM> and then install NoMachine with the `dpkg` command. For example if you downloaded package '`nomachine_6.9.2_1_arm64.deb`' :

```
1 $ sudo dpkg -i nomachine_6.9.2_1_arm64.deb
```

If we want a headless display with high resolution over remote desktop we will need an HDMI dummy-plug.

# Annexe C

## Code

### C.1 Modified vision\_to\_mavros

```
1 #include <ros/ros.h>
2 #include <tf/transform_listener.h>
3 #include <tf/transform_broadcaster.h>
4 #include <geometry_msgs/Pose.h>
5 #include <geometry_msgs/PoseStamped.h>
6 #include <nav_msgs/Path.h>
7 #include <mavros_msgs/LandingTarget.h>
8
9 #include <string.h>
10 using namespace std;
11 tf::Transform get_tf_from_stamped_tf(tf::StampedTransform sTf) {
12     tf::Transform tf(sTf.getBasis(), sTf.getOrigin()); //construct a
13     transform using elements of sTf
14     return tf;
15 }
16
17 bool multiply_stamped_tfs(tf::StampedTransform A_stf,
18     tf::StampedTransform B_stf, tf::StampedTransform &C_stf) {
19     tf::Transform A,B,C; //simple transforms--not stamped
20     std::string str1 (A_stf.child_frame_id_); //want to compare
21     strings to check consistency
22     std::string str2 (B_stf.frame_id_);
23     if (str1.compare(str2) != 0) { //SHOULD get that child frame of A
24         is parent frame of B
25         std::cout << "can't multiply transforms; mismatched frames" <<
26         endl;
27         std::cout << str1 << " is not " << str2 << '\n';
28         return false;
29     }
30     //if here, the named frames are logically consistent
31     A = get_tf_from_stamped_tf(A_stf); // get the transform from the
32     stamped transform
33     B = get_tf_from_stamped_tf(B_stf);
34     C = A*B; //multiplication is defined for transforms
35     C_stf.frame_id_ = A_stf.frame_id_; //assign appropriate parent
36     and child frames to result
37     C_stf.child_frame_id_ = B_stf.child_frame_id_;
38     C_stf.setOrigin(C.getOrigin()); //populate the origin and
39     orientation of the result
40     C_stf.setBasis(C.getBasis());
```

```

34     C_stf.stamp_ = ros::Time::now(); //assign the time stamp to
35     // current time;
36     // alternatively, could assign this to the OLDER of A or B
37     // transforms
38     return true; //if got here, the multiplication is valid
39 }
40
41 int main(int argc, char** argv)
42 {
43     ros::init(argc, argv, "vision_to_mavros");
44
45     ros::NodeHandle node;
46
47     /////////////////////////////////
48     // Variables for precision navigation
49     /////////////////////////////////
50     ros::Publisher camera_pose_publisher = node.advertise<
51         geometry_msgs::PoseStamped>("vision_pose", 10);
52
53     ros::Publisher body_path_publisher = node.advertise<nav_msgs::Path
54         >("body_frame/path", 1);
55
56     tf::TransformListener tf_listener;
57
58     tf::TransformListener tf_map_listener; //map to odometry
59     // transform listener (thats given by rviz)
60
61     tf::TransformBroadcaster br; //broadcaster of a tf FCU
62
63     tf::StampedTransform transform; //original transform from
64     // odometry frame to t265_link
65
66     tf::StampedTransform transform_map_odom ,transform_map_link; ////
67     // map to odometry transform and total transform map to t265_link
68
69     geometry_msgs::PoseStamped msg_body_pose;
70
71     nav_msgs::Path body_path;
72
73     std::string target_frame_id = "/camera_odom_frame";
74
75     std::string source_frame_id = "/camera_link";
76
77     double output_rate = 20, roll_cam = 0, pitch_cam = 0, yaw_cam =
78     1.5707963, gamma_world = -1.5707963;
79
80     // Read parameters from launch file, including: target_frame_id,
81     // source_frame_id, output_rate
82     {
83         // The frame in which we find the transform into, the original
84         // "world" frame
85         if(node.getParam("target_frame_id", target_frame_id))
86         {
87             ROS_INFO("Get target_frame_id parameter: %s", target_frame_id
88             .c_str());
89         }

```

```
81     else
82     {
83         ROS_WARN("Using default target_frame_id: %s", target_frame_id
84             .c_str());
85     }
86
87     // The frame for which we find the transform to target_frame_id,
88     // the original "camera" frame
89     if(node.getParam("source_frame_id", source_frame_id))
90     {
91         ROS_INFO("Get source_frame_id parameter: %s", source_frame_id
92             .c_str());
93     }
94     else
95     {
96         ROS_WARN("Using default source_frame_id: %s", source_frame_id
97             .c_str());
98     }
99
100    // The rate at which we wish to publish final pose data
101    if(node.getParam("output_rate", output_rate))
102    {
103        ROS_INFO("Get output_rate parameter: %f", output_rate);
104    }
105    else
106    {
107        ROS_WARN("Using default output_rate: %f", output_rate);
108    }
109
110    // The rotation around z axis between original world frame and
111    // target world frame, assuming the z axis needs not to be changed
112    // In this case, target world frame has y forward, x to the
113    // right and z upwards (ENU as ROS dictates)
114    if(node.getParam("gamma_world", gamma_world))
115    {
116        ROS_INFO("Get gamma_world parameter: %f", gamma_world);
117    }
118    else
119    {
120        ROS_WARN("Using default gamma_world: %f", gamma_world);
121    }
122
123    // The roll angle around camera's own axis to align with body
124    // frame
125    if(node.getParam("roll_cam", roll_cam))
126    {
127        ROS_INFO("Get roll_cam parameter: %f", roll_cam);
128    }
129    else
130    {
131        ROS_WARN("Using default roll_cam: %f", roll_cam);
132    }
133
134    // The pitch angle around camera's own axis to align with body
135    // frame
136    if(node.getParam("pitch_cam", pitch_cam))
137    {
138        ROS_INFO("Get pitch_cam parameter: %f", pitch_cam);
```

```

131     }
132     else
133     {
134         ROS_WARN("Using default pitch_cam: %f", pitch_cam);
135     }
136
137     // The yaw angle around camera's own axis to align with body
138     // frame
139     if(node.getParam("yaw_cam", yaw_cam))
140     {
141         ROS_INFO("Get yaw_cam parameter: %f", yaw_cam);
142     }
143     else
144     {
145         ROS_WARN("Using default yaw_cam: %f", yaw_cam);
146     }
147
148 ///////////////////////////////////////////////////////////////////
149 // Variables for precision landing (optional)
150 ///////////////////////////////////////////////////////////////////
151 bool enable_precland = false;
152
153 std::string precland_target_frame_id = "/landing_target";
154
155 std::string precland_camera_frame_id = "/"
156     "camera_fisheye2_optical_frame";
157
158 ros::Publisher precland_msg_publisher;
159
160 if(node.getParam("enable_precland", enable_precland))
161 {
161     ROS_INFO("Precision landing: %s", enable_precland ? "enabled" :
162             "disabled");
163 }
164 else
165 {
165     ROS_INFO("Precision landing disabled by default");
166 }
167
168 if (enable_precland)
169 {
170     // The frame of the landing target in the camera frame
171     if(node.getParam("precland_target_frame_id",
172         precland_target_frame_id))
173     {
173         ROS_INFO("Get precland_target_frame_id parameter: %s",
174         precland_target_frame_id.c_str());
175     }
176     else
177     {
177         ROS_WARN("Using default precland_target_frame_id: %s",
178         precland_target_frame_id.c_str());
179     }
180
181     if(node.getParam("precland_camera_frame_id",
182         precland_camera_frame_id))
183     {

```

```

182     ROS_INFO("Get precland_camera_frame_id parameter: %s",
183     precland_camera_frame_id.c_str());
184     }
185     else
186     {
187         ROS_WARN("Using default precland_camera_frame_id: %s",
188         precland_camera_frame_id.c_str());
189     }
190 }

191 //////////////////////////////////////////////////////////////////
192 // Wait for the first transform to become available.
193 //////////////////////////////////////////////////////////////////
194 tf_listener.waitForTransform(target_frame_id, source_frame_id,
195     ros::Time::now(), ros::Duration(3.0));

196 ros::Time last_tf_time = ros::Time::now();
197 ros::Time last_precland_tf_time = ros::Time::now();

198 // Limit the rate of publishing data, otherwise the other
199 // telemetry port might be flooded
200 ros::Rate rate(output_rate);

201 while (node.ok())
202 {
203     // For tf, Time(0) means "the latest available" transform in
204     // the buffer.
205     ros::Time now = ros::Time(0);

206     //////////////////////////////////////////////////////////////////
207     // Publish vision_position_estimate message if transform is
208     // available
209     //////////////////////////////////////////////////////////////////
210     try
211     {
212         // lookupTransform(frame_2, frame_1, at_this_time,
213         this_transform)
214         // will give the transform from frame_1 to frame_2
215         tf_listener.lookupTransform(target_frame_id, source_frame_id,
216             now, transform); //get the odom to link transform
217         tf_map_listener.lookupTransform("map", target_frame_id, now,
218             transform_map_odom); //get the map to odom transform

219         if(!multiply_stamped_tfs(transform_map_odom, transform,
220         transform_map_link)){ //get the map to link transform
221             ROS_WARN("Multiplication not valid");
222         }
223         // Only publish pose messages when we have new transform data
224         .
225         if (last_tf_time < transform.stamp_)
226         {
227             //last_tf_time = transform.stamp_;
228
229             static tf::Vector3 position_orig, position_body;

```

```

228     static tf::Quaternion quat_cam, quat_cam_to_body_x,
229     quat_cam_to_body_y, quat_cam_to_body_z, quat_rot_z, quat_body,
230     quat_body_NED;
231
232     // 1) Rotation from original world frame to world frame
233     // with y forward.
234     // See the full rotation matrix at https://en.wikipedia.org/wiki/Rotation\_matrix#Basic\_rotations
235
236     //position_orig = transform.getOrigin();
237     position_orig = transform_map_link.getOrigin();
238
239     position_body.setX( cos(gamma_world) * position_orig.getX()
240 + sin(gamma_world) * position_orig.getY());
241     position_body.setY(-sin(gamma_world) * position_orig.getX()
242 + cos(gamma_world) * position_orig.getY());
243     position_body.setZ(position_orig.getZ());
244
245
246
247     quat_cam_to_body_x = tf::createQuaternionFromRPY(roll_cam,
248 0, 0);
249     quat_cam_to_body_y = tf::createQuaternionFromRPY(0,
250 pitch_cam, 0);
251     quat_cam_to_body_z = tf::createQuaternionFromRPY(0, 0,
252 yaw_cam);
253
254     // 3) Rotate body frame 90 degree (align body x with world
255     // y at launch)
256     quat_rot_z = tf::createQuaternionFromRPY(0, 0, -gamma_world
257 );
258
259     quat_body = quat_rot_z * quat_cam * quat_cam_to_body_x *
260     quat_cam_to_body_y * quat_cam_to_body_z;
261     quat_body_NED = quat_cam_to_body_x * quat_cam_to_body_y *
262     quat_cam_to_body_z;
263     quat_body.normalize();
264
265     //trying to add tf in NED frame
266     tf::Transform transform_NED;
267     transform_NED.setRotation( quat_body );
268     transform_NED.setOrigin( position_body );
269
270     //br.sendTransform(tf::StampedTransform(transform_NED, ros
271 ::Time::now(),target_frame_id,"FCU"));
272     br.sendTransform(tf::StampedTransform(transform_map_link,
273 ros::Time::now(),"map","FCU"));
274
275     // Create PoseStamped message to be sent
276     msg_body_pose.header.stamp = transform.stamp_;
277     msg_body_pose.header.frame_id = transform.frame_id_;
278     msg_body_pose.pose.position.x = position_body.getX();
279     msg_body_pose.pose.position.y = position_body.getY();

```

```

271     msg_body_pose.pose.position.z = position_body.getZ();
272     msg_body_pose.pose.orientation.x = quat_body.getX();
273     msg_body_pose.pose.orientation.y = quat_body.getY();
274     msg_body_pose.pose.orientation.z = quat_body.getZ();
275     msg_body_pose.pose.orientation.w = quat_body.getW();
276
277     // Publish pose of body frame in world frame
278     camera_pose_publisher.publish(msg_body_pose);
279
280     // Publish trajectory path for visualization
281     body_path.header.stamp = msg_body_pose.header.stamp;
282     body_path.header.frame_id = msg_body_pose.header.frame_id;
283     body_path.poses.push_back(msg_body_pose);
284     body_path_publisher.publish(body_path);
285 }
286 }
287 catch (tf::TransformException ex)
288 {
289     ROS_WARN("%s",ex.what());
290     ros::Duration(1.0).sleep();
291 }
292
293 //////////////////////////////////////////////////////////////////
294 // Publish landing_target message if option is enabled and
295 // transform is available
296 //////////////////////////////////////////////////////////////////
297 if (enable_preland)
298 {
299     if (tf_listener.canTransform(preland_camera_frame_id,
300         preland_target_frame_id, now))
301     {
302         // lookupTransform(frame_2, frame_1, at_this_time,
303         this_transform)
304         // will give the transfrom from frame_1 to frame_2
305         tf_listener.lookupTransform(preland_camera_frame_id,
306         preland_target_frame_id, now, transform);
307
308         // Only publish when we have new data
309         if (last_preland_tf_time < transform.stamp_)
310         {
311             last_preland_tf_time = transform.stamp_;
312
313             mavros_msgs::LandingTarget msg_landing_target;
314
315             // Setup the landing target message according to the
316             // relative protocol: https://mavlink.io/en/services/landing_target
317             // .html#camera_image_relative
318             msg_landing_target.header.frame_id = transform.frame_id_;
319             msg_landing_target.header.stamp = transform.stamp_;
            msg_landing_target.target_num = 0;
            msg_landing_target.frame = mavros_msgs::LandingTarget::LOCAL_NED;
            msg_landing_target.type = mavros_msgs::LandingTarget::VISION_FIDUCIAL;
320
            msg_landing_target.angle[0] = std::atan(transform.
getOrigin().getX() / transform.getOrigin().getZ());
            msg_landing_target.angle[1] = std::atan(transform.

```

```

320     getOrigin().getY() / transform.getOrigin().getZ());
321     msg_landing_target.distance = transform.getOrigin().
322     length();
323
324     // Publish the message
325     precland_msg_publisher.publish(msg_landing_target);
326
327     ROS_INFO("Landing target detected");
328 }
329
330 ///////////////////////////////////////////////////
331 // Repeat
332 ///////////////////////////////////////////////////
333 rate.sleep();
334 }
335 return 0;
336 }
```

## C.2 Lidar manager

```

1 import qwiic_vl53l1x
2 import time
3 import sys
4 import Jetson.GPIO as GPIO
5
6 import os
7
8 #Cable connection : 29,21,15,M19M,13,11,7,23      M marks main branch
9 class lidarManager:
10
11     def __init__(self,XSHUT = [15,21,29,19,23,11,13,7]):  #
12         15,21,29,19,23,11,13,7
13         self.XSHUT = XSHUT # 13,15,21,23      pins of connected
14         sensors, it also defines the number of sensors to be used
15
16         self.first_address = 0x30 #first I2C address to assign
17
18         GPIO.setwarnings(False)
19         GPIO.setmode(GPIO.BCM) #numbering of ports
20         self.init_outputs()
21         #instantiate sensor objects
22         self.ToF = []
23         for sensor in range(0,len(self.XSHUT)):
24             self.ToF.append(qwiic_vl53l1x.QwiicVL53L1X())
25
26     def init_outputs(self):
27         for pin in self.XSHUT:
28             GPIO.setup(pin, GPIO.OUT) #put as output
29             GPIO.output(pin, False) #turn off
30
31     def set_input(self):
32         for pin in self.XSHUT:
33             GPIO.setup(pin, GPIO.IN) #put as output
```

```

35
36     def start_sensors(self):
37         #turn on and set addresses
38         for sensor in range(0,len(self.XSHUT)):
39             GPIO.output(self.XSHUT[sensor], True) #turn on
40             # print("sensor on")
41
42             selfToF[sensor].sensor_init() #initialize
43             time.sleep(.005)
44             selfToF[sensor].set_distance_mode(2)
45             selfToF[sensor].set_i2c_address( self.first_address +
46             sensor*2) #change address
47
48     def stop_sensors(self):
49         #turn off
50         for sensor in range(0,len(self.XSHUT)-1):
51             GPIO.output(self.XSHUT[sensor], False) #turn off
52
53     def start_ranging(self):
54         #start ranging
55         for sensor in range(0,len(self.XSHUT)):
56             selfToF[sensor].start_ranging()
57
58     def stop_ranging(self):
59         #start ranging
60         for sensor in range(0,len(self.XSHUT)):
61             selfToF[sensor].stop_ranging()
62
63     def get_range(self,sensor):
64         return selfToF[sensor].get_distance()
65
66
67 def main():
68     while True:
69         Manager = lidarManager()
70         Manager.init_outputs()
71         Manager.start_sensors()
72         Manager.set_input()
73         Manager.start_ranging()
74         print("while")
75
76
77     while True:
78         try:
79             string = ""
80             start_time = time.time()
81
82             for sensor in range(0,len(Manager.XSHUT)):
83                 value = Manager.get_range(sensor)
84                 string = string + str(value)+","
85             time.sleep(.05)
86             print("Distance(mm): %s" % string)
87             print("--- %f Hz ---" % (1/(time.time() -
88             start_time)))
89
90         except:

```

```

91             GPIO.cleanup()
92             break
93
94
95         print("Error")
96         break
97     # GPIO.setmode(GPIO.BOARD)
98     # GPIO.setup(7, GPIO.OUT)
99     # GPIO.output(7, False)
100
101 if __name__ == '__main__':
102     try:
103         main()
104     except Exception as e:
105         print(e)

```

## C.3 Lidar to Mavros

```

1#!/usr/bin/env python
2import roslib; roslib.load_manifest('rviz')
3import rospy
4from sensor_msgs.msg import Range
5
6
7
8def talker():
9    pub0 = rospy.Publisher('/mavros/distance_sensor/
10                           rangefinder_0_sub', Range, queue_size = 10)
11    pub45 = rospy.Publisher('/mavros/distance_sensor/
12                           rangefinder_45_sub', Range, queue_size = 10)
13    rospy.init_node('talker')
14    # ranges = [float('NaN'), 1.0, -float('Inf'), 3.0, float('Inf')]
15    ranges = [float(1)]
16    min_range = 0.04
17    max_range = 4
18    while not rospy.is_shutdown():
19        r = Range()
20        r.header.stamp = rospy.Time.now()
21        r.header.frame_id = "/base_link"
22        r.radiation_type = 1 # 0 ultrasound - 1 infrared
23        r.field_of_view = 0.349066 #radian
24        r.min_range = min_range #[m]
25        r.max_range = max_range #[m]
26        r.range = rg
27        pub.publish(r)
28        rospy.sleep(1.0)
29
30if __name__ == '__main__':
31    try:
32        talker()
33    except rospy.ROSInterruptException: pass

```

## C.4 Automatic flight

### C.4.1 Take Off and Landing

```

1  #!/usr/bin/env python
2
3  ##
4  #
5  # Control a MAV via mavros
6  #
7  ##
8
9  import rospy
10 import tf
11 from geometry_msgs.msg import Pose, PoseStamped, Twist, Quaternion
12 from mavros_msgs.msg import OverrideRCIn
13 from mavros_msgs.msg import RCIn
14 from mavros_msgs.srv import CommandBool
15 from mavros_msgs.srv import SetMode
16 from mavros_msgs.srv import CommandTOL
17
18 pi_2 = 3.141592654 / 2.0
19
20 class MavController:
21     """
22         A simple object to help interface with mavros
23     """
24     def __init__(self):
25
26         rospy.init_node("mav_control_node")
27         rospy.Subscriber("/mavros/local_position/pose", PoseStamped,
28             self.pose_callback)
29         rospy.Subscriber("/mavros/rc/in", RCIn, self.rc_callback)
30
31         self.cmd_pos_pub = rospy.Publisher("/mavros/
32             setpoint_position/local", PoseStamped, queue_size=1)
33         self.cmd_vel_pub = rospy.Publisher("/mavros/
34             setpoint_velocity/cmd_vel_unstamped", Twist, queue_size=1)
35         self.rc_override = rospy.Publisher("/mavros/rc/override",
36             OverrideRCIn, queue_size=1)
37
38         # mode 0 = STABILIZE
39         # mode 4 = GUIDED
40         # mode 9 = LAND
41         self.mode_service = rospy.ServiceProxy('/mavros/set_mode',
42             SetMode)
43         self.arm_service = rospy.ServiceProxy('/mavros/cmd/arming',
44             CommandBool)
45         self.takeoff_service = rospy.ServiceProxy('/mavros/cmd/
46             takeoff', CommandTOL)
47
48         self.rc = RCIn()
49         self.pose = Pose()
50         self.timestamp = rospy.Time()
51
52     def rc_callback(self, data):
53         """
54             Keep track of the current manual RC values
55         """

```

```

48     """
49     self.rc = data
50
51     def pose_callback(self, data):
52         """
53             Handle local position information
54         """
55         self.timestamp = data.header.stamp
56         self.pose = data.pose
57
58     def goto(self, pose):
59         """
60             Set the given pose as a the next setpoint by sending
61             a SET_POSITION_TARGET_LOCAL_NED message. The copter must
62             be in GUIDED mode for this to work.
63         """
64         pose_stamped = PoseStamped()
65         pose_stamped.header.stamp = self.timestamp
66         pose_stamped.pose = pose
67
68         self.cmd_pos_pub.publish(pose_stamped)
69
70     def goto_xyz_rpy(self, x, y, z, ro, pi, ya):
71         pose = Pose()
72         pose.position.x = x
73         pose.position.y = y
74         pose.position.z = z
75
76         quat = tf.transformations.quaternion_from_euler(ro, pi, ya
77 + pi_2)
78
79         pose.orientation.x = quat[0]
80         pose.orientation.y = quat[1]
81         pose.orientation.z = quat[2]
82         pose.orientation.w = quat[3]
83         self.goto(pose)
84         #print(quat)
85
85     def set_vel(self, vx, vy, vz, avx=0, avy=0, avz=0):
86         """
87             Send comand velocities. Must be in GUIDED mode. Assumes
88             angular
89             velocities are zero by default.
90         """
91
92         cmd_vel = Twist()
93
93         cmd_vel.linear.x = vx
94         cmd_vel.linear.y = vy
95         cmd_vel.linear.z = vz
96
96         cmd_vel.angular.x = avx
97         cmd_vel.angular.y = avy
98         cmd_vel.angular.z = avz
99
100        self.cmd_vel_pub.publish(cmd_vel)
101
102    def arm(self):
103        """

```

```
104     Arm the throttle
105     """
106     return self.arm_service(True)
107
108 def disarm(self):
109     """
110     Disarm the throttle
111     """
112     return self.arm_service(False)
113
114 def takeoff(self, height=1.0):
115     """
116     Arm the throttle, takeoff to a few feet, and set to guided
117     mode
118     """
119     # Set to stabilize mode for arming
120     #mode_resp = self.mode_service(custom_mode="0")
121     mode_resp = self.mode_service(custom_mode="4")
122     self.arm()
123
124     # Set to guided mode
125     #mode_resp = self.mode_service(custom_mode="4")
126
127     # Takeoff
128     takeoff_resp = self.takeoff_service(altitude=height)
129
130     #return takeoff_resp
131     return mode_resp
132
133 def land(self):
134     """
135     Set in LAND mode, which should cause the UAV to descend
136     directly,
137     land, and disarm.
138     """
139     resp = self.mode_service(custom_mode="9")
140     self.disarm()
141
142 def simple_demo():
143     """
144     A simple demonstration of using mavros commands to control a
145     UAV.
146     """
147     c = MavController()
148     rospy.sleep(1)
149
150     alt = float(raw_input('Enter takeoff altitude.\n'))
151
152     tol_pos = 0.2
153
154     c.takeoff(alt)
155     while c.pose.position.z < alt - tol_pos or c.pose.position.z >
156         alt + tol_pos:
157             print(c.pose.position.z)
158     rospy.sleep(1)
159     print("position reached:",c.pose.position.x,c.pose.position.y,c
160         .pose.position.z)
161     while True:
```

```

157     d_x,d_y,d_z = raw_input('Enter x,y,z position DELTA.\n').
158     split(',')
159     print("move ")
160     d_x = float(d_x)
161     d_y = float(d_y)
162     d_z = float(d_z)
163
164     x_n = c.pose.position.x + d_x
165     y_n = c.pose.position.y + d_y
166     z_n = c.pose.position.z + d_z
167
168     c.goto_xyz_rpy(x_n,y_n,z_n,0,0,0)
169     error_x = abs(c.pose.position.x - x_n)
170     error_y = abs(c.pose.position.y - y_n)
171     error_z= abs(c.pose.position.z - z_n)
172
173     while error_x > tol_pos or error_y > tol_pos or error_z >
174     tol_pos :
175         error_x = abs(c.pose.position.x - x_n)
176         error_y = abs(c.pose.position.y - y_n)
177         error_z= abs(c.pose.position.z - z_n)
178         rospy.sleep(1)
179         print(error_x,error_y,error_z)
180         print("position reached:",c.pose.position.x,c.pose.position.
181             .y,c.pose.position.z)
182
183         rospy.sleep(0.1)
184         land_bool = raw_input('Land? Y/n\n')
185         if land_bool == 'Y' or land_bool == 'y' :
186             break
187
188 #print("Waypoint 1: position control")
189 #c.goto_xyz_rpy(0.0,0.0,1.2,0,0,-1*pi_2)
190 #rospy.sleep(2)
191 #c.goto_xyz_rpy(0.4,0.0,1.2,0,0,-1*pi_2)
192 #rospy.sleep(3)
193 #print("Waypoint 2: position control")
194 #c.goto_xyz_rpy(0.4,0.0,1.2,0,0,0)
195 #rospy.sleep(2)
196 #c.goto_xyz_rpy(0.4,0.4,1.2,0,0,0)
197 #rospy.sleep(3)
198 #print("Waypoint 3: position control")
199 #c.goto_xyz_rpy(0.4,0.4,1.2,0,0,pi_2)
200 #rospy.sleep(2)
201 #c.goto_xyz_rpy(0.0,0.4,1.2,0,0,pi_2)
202 #rospy.sleep(3)
203 #print("Waypoint 4: position control")
204 #c.goto_xyz_rpy(0.0,0.4,1.2,0,0,2*pi_2)
205 #rospy.sleep(2)
206 #c.goto_xyz_rpy(0.0,0.0,1.2,0,0,2*pi_2)
207 #rospy.sleep(3)
208
209 #print("Velocity Setpoint 1")
210 #c.set_vel(0,0.1,0)
211 #rospy.sleep(5)
212 #print("Velocity Setpoint 2")

```

```

212     #c.set_vel(0, -0.1, 0)
213     #rospy.sleep(5)
214     #print("Velocity Setpoint 3")
215     #c.set_vel(0,0,0)
216     #rospy.sleep(5)
217
218     print("Landing")
219     c.land()
220
221 if __name__=="__main__":
222     simple_demo()

```

## C.4.2 Yaw control

```

1 #!/usr/bin/env python
2
3 ##
4 #
5 # Control a MAV via mavros
6 #
7 ##
8 import numpy as np
9 import rospy
10 import tf
11 from geometry_msgs.msg import Pose, PoseStamped, Twist, Quaternion
12 from mavros_msgs.msg import OverrideRCIn
13 from mavros_msgs.msg import RCIn
14 from mavros_msgs.srv import CommandBool
15 from mavros_msgs.srv import SetMode
16 from mavros_msgs.srv import CommandTOL
17
18 pi_2 = 3.141592654 / 2.0
19 pi = 2*pi_2
20
21 class MavController:
22     """
23     A simple object to help interface with mavros
24     """
25     def __init__(self):
26
27         rospy.init_node("mav_control_node")
28         #rospy.Subscriber("/mavros/vision_pose/pose", PoseStamped,
29         self.pose_callback)
30         rospy.Subscriber("/mavros/local_position/pose", PoseStamped,
31             self.pose_callback)
32         rospy.Subscriber("/mavros/rc/in", RCIn, self.rc_callback)
33
34         self.cmd_pos_pub = rospy.Publisher("/mavros/
35             setpoint_position/local", PoseStamped, queue_size=1)
36         self.cmd_vel_pub = rospy.Publisher("/mavros/
37             setpoint_velocity/cmd_vel_unstamped", Twist, queue_size=1)
38         self.rc_override = rospy.Publisher("/mavros/rc/override",
39             OverrideRCIn, queue_size=1)
40
41         # mode 0 = STABILIZE
42         # mode 4 = GUIDED
43         # mode 9 = LAND
44         self.mode_service = rospy.ServiceProxy('/mavros/set_mode',

```

```

    SetMode)
40      self.arm_service = rospy.ServiceProxy('/mavros/cmd/arming',
41                                         CommandBool)
42      self.takeoff_service = rospy.ServiceProxy('/mavros/cmd/
43                                         takeoff', CommandTOL)
44
45      self.rc = RCIn()
46      self.pose = Pose()
47      self.timestamp = rospy.Time()
48      self.euler = np.zeros(3)
49
50  def rc_callback(self, data):
51      """
52      Keep track of the current manual RC values
53      """
54      self.rc = data
55
56  def pose_callback(self, data):
57      """
58      Handle local position information
59      """
60      self.timestamp = data.header.stamp
61      self.pose = data.pose
62      self.euler[0],self.euler[1],self.euler[2] = tf.
63      transformations.euler_from_quaternion([self.pose.orientation.x,
64      self.pose.orientation.y,self.pose.orientation.z,self.pose.
65      orientation.w])
66
66      if self.euler[2] < 0:
67          self.euler[2]+= 2*pi
68
69  def goto(self, pose):
70      """
71      Set the given pose as a the next setpoint by sending
72      a SET_POSITION_TARGET_LOCAL_NED message. The copter must
73      be in GUIDED mode for this to work.
74      """
75
76      pose_stamped = PoseStamped()
77      pose_stamped.header.stamp = self.timestamp
78      pose_stamped.pose = pose
79
80      self.cmd_pos_pub.publish(pose_stamped)
81
82  def goto_xyz_rpy(self, x, y, z, ro, pi, ya):
83      pose = Pose()
84      pose.position.x = x
85      pose.position.y = y
86      pose.position.z = z
87
88      quat = tf.transformations.quaternion_from_euler(ro, pi, ya
89      )# ya + pi/2
90
91      pose.orientation.x = quat[0]
92      pose.orientation.y = quat[1]
93      pose.orientation.z = quat[2]
94      pose.orientation.w = quat[3]
95      self.goto(pose)
96      #print(quat)
97
98

```

```
91     def set_vel(self, vx, vy, vz, avx=0, avy=0, avz=0):
92         """
93             Send comand velocities. Must be in GUIDED mode. Assumes
94             angular
95             velocities are zero by default.
96         """
97         cmd_vel = Twist()
98
99         cmd_vel.linear.x = vx
100        cmd_vel.linear.y = vy
101        cmd_vel.linear.z = vz
102
103        cmd_vel.angular.x = avx
104        cmd_vel.angular.y = avy
105        cmd_vel.angular.z = avz
106
107        self.cmd_vel_pub.publish(cmd_vel)
108
109    def arm(self):
110        """
111            Arm the throttle
112        """
113        return self.arm_service(True)
114
115    def disarm(self):
116        """
117            Disarm the throttle
118        """
119        return self.arm_service(False)
120
121    def takeoff(self, height=1.0):
122        """
123            Arm the throttle, takeoff to a few feet, and set to guided
124            mode
125        """
126        # Set to stabilize mode for arming
127        #mode_resp = self.mode_service(custom_mode="0")
128        mode_resp = self.mode_service(custom_mode="4")
129        self.arm()
130
131
132        # Set to guided mode
133        #mode_resp = self.mode_service(custom_mode="4")
134
135        # Takeoff
136        takeoff_resp = self.takeoff_service(altitude=height)
137
138        #return takeoff_resp
139        return mode_resp
140
141    def land(self):
142        """
143            Set in LAND mode, which should cause the UAV to descend
144            directly,
145            land, and disarm.
146        """
147        resp = self.mode_service(custom_mode="9")
148        self.disarm()
149
150
```

```

146 def simple_demo():
147     """
148         A simple demonstration of using mavros commands to control a
149         UAV.
150     """
151     c = MavController()
152     rospy.sleep(1)
153
154     alt = float(raw_input('Enter takeoff altitude.\n'))
155     tol_pos = 0.2
156     tol_yaw = pi_2 / 5.0 # 18 deg of error tolerance
157
158     c.takeoff(alt)
159     while c.pose.position.z < alt - tol_pos or c.pose.position.z >
160         alt + tol_pos:
161             print(c.pose.position.z)
162             rospy.sleep(1)
163             print("position reached:",c.pose.position.x,c.pose.position.y,c.
164             .pose.position.z,c.euler[2]*pi/180)
165
166             while True:
167                 print(c.euler[2])
168                 d_yaw = raw_input('Enter YAW position.\n')
169                 print("move")
170                 d_yaw = float(d_yaw)*pi/180
171                 d_x = 0
172                 d_y = 0
173                 d_z = 0
174
175                 x_n = c.pose.position.x + d_x
176                 y_n = c.pose.position.y + d_y
177                 z_n = c.pose.position.z + d_z
178                 #yaw_n = c.euler[2] + d_yaw
179                 yaw_n = d_yaw
180                 print("new position:",x_n,y_n,z_n,yaw_n*180/pi)
181                 c.goto_xyz_rpy(x_n,y_n,z_n,0,0,yaw_n)
182                 error_x = abs(c.pose.position.x - x_n)
183                 error_y = abs(c.pose.position.y - y_n)
184                 error_z= abs(c.pose.position.z - z_n)
185                 error_yaw= abs(c.euler[2] - yaw_n)
186
187                 while error_x > tol_pos or error_y > tol_pos or error_z >
188                     tol_pos or error_yaw > tol_yaw:
189                     error_x = abs(c.pose.position.x - x_n)
190                     error_y = abs(c.pose.position.y - y_n)
191                     error_z= abs(c.pose.position.z - z_n)
192                     error_yaw= abs(c.euler[2] - yaw_n)
193                     rospy.sleep(0.5)
194                     print(c.euler[2]*180/pi,yaw_n*180/pi,error_yaw*180/pi)
195                     print("position reached:",c.pose.position.x,c.pose.position.
196                     .y,c.pose.position.z,c.euler[2]*180/pi)
197
198                     rospy.sleep(0.1)
199                     land_bool = raw_input('Land?  Y/n\n')
200                     if land_bool == 'Y' or land_bool == 'y' :
201                         break

```

```

199
200
201
202     print("Landing")
203     c.land()
204
205 if __name__=="__main__":
206     simple_demo()

```

### C.4.3 Keyboard Control

```

1 #!/usr/bin/env python
2
3 ##
4 #
5 # Control a MAV via mavros
6 #
7 ##
8 from getch import getch, pause
9
10 import rospy
11 import tf
12 from geometry_msgs.msg import Pose, PoseStamped, Twist, Quaternion
13 from mavros_msgs.msg import OverrideRCIn
14 from mavros_msgs.msg import RCIn
15 from mavros_msgs.srv import CommandBool
16 from mavros_msgs.srv import SetMode
17 from mavros_msgs.srv import CommandTOL
18
19 pi_2 = 3.141592654 / 2.0
20
21 class MavController:
22     """
23         A simple object to help interface with mavros
24     """
25     def __init__(self):
26
27         rospy.init_node("mav_control_node")
28         rospy.Subscriber("/mavros/local_position/pose", PoseStamped,
29                         self.pose_callback)
29         rospy.Subscriber("/mavros/vision_pose/pose", PoseStamped,
30                         self.orientation_callback)
30         rospy.Subscriber("/mavros/rc/in", RCIn, self.rc_callback)
31
32         self.cmd_pos_pub = rospy.Publisher("/mavros/
33 setpoint_position/local", PoseStamped, queue_size=1)
33         self.cmd_vel_pub = rospy.Publisher("/mavros/
34 setpoint_velocity/cmd_vel_unstamped", Twist, queue_size=1)
34         self.rc_override = rospy.Publisher("/mavros/rc/override",
35             OverrideRCIn, queue_size=1)
35
36         # mode 0 = STABILIZE
37         # mode 4 = GUIDED
38         # mode 9 = LAND
39         self.mode_service = rospy.ServiceProxy('/mavros/set_mode',
40             SetMode)
40         self.arm_service = rospy.ServiceProxy('/mavros/cmd/arming',
41             CommandBool)

```

```

41         self.takeoff_service = rospy.ServiceProxy('/mavros/cmd/takeoff', CommandTOL)
42
43         self.rc = RCIn()
44         self.pose = Pose()
45         self.timestamp = rospy.Time()
46         self.euler = []
47
48     def rc_callback(self, data):
49         """
50             Keep track of the current manual RC values
51         """
52         self.rc = data
53
54     def pose_callback(self, data):
55         """
56             Handle local position information
57         """
58         self.timestamp = data.header.stamp
59         self.pose = data.pose
60
61         #self.euler = tf.transformations.euler_from_quaternion([
62         #    data.pose.orientation.x,data.pose.orientation.y,data.pose.
63         #    orientation.z,data.pose.orientation.w])
64
65     def orientation_callback(self, data):
66         """
67             Handle local position information
68         """
69
70     def goto(self, pose):
71         """
72             Set the given pose as a the next setpoint by sending
73             a SET_POSITION_TARGET_LOCAL_NED message. The copter must
74             be in GUIDED mode for this to work.
75         """
76
77         pose_stamped = PoseStamped()
78         pose_stamped.header.stamp = self.timestamp
79         pose_stamped.pose = pose
80
81         self.cmd_pos_pub.publish(pose_stamped)
82
83     def goto_xyz_rpy(self, x, y, z, ro, pi, ya):
84         pose = Pose()
85         pose.position.x = x
86         pose.position.y = y
87         pose.position.z = z
88
89         quat = tf.transformations.quaternion_from_euler(ro, pi, ya
90 + pi_2)#it had ya + pi_2
91
92         pose.orientation.x = quat[0]
93         pose.orientation.y = quat[1]
94         pose.orientation.z = quat[2]

```

```
93         pose.orientation.w = quat[3]
94         self.goto(pose)
95         #print(quat)
96
97     def set_vel(self, vx, vy, vz, avx=0, avy=0, avz=0):
98         """
99             Send comand velocities. Must be in GUIDED mode. Assumes
100            angular
101            velocities are zero by default.
102            """
103
104        cmd_vel = Twist()
105
106        cmd_vel.linear.x = vx
107        cmd_vel.linear.y = vy
108        cmd_vel.linear.z = vz
109
110        cmd_vel.angular.x = avx
111        cmd_vel.angular.y = avy
112        cmd_vel.angular.z = avz
113
114        self.cmd_vel_pub.publish(cmd_vel)
115
116    def arm(self):
117        """
118            Arm the throttle
119            """
120
121        return self.arm_service(True)
122
123    def disarm(self):
124        """
125            Disarm the throttle
126            """
127
128        return self.arm_service(False)
129
130    def takeoff(self, height=1.0):
131        """
132            Arm the throttle, takeoff to a few feet, and set to guided
133            mode
134            """
135
136        # Set to stabilize mode for arming
137        #mode_resp = self.mode_service(custom_mode="0")
138        mode_resp = self.mode_service(custom_mode="4")
139        self.arm()
140
141
142        # Set to guided mode
143        #mode_resp = self.mode_service(custom_mode="4")
144
145        # Takeoff
146        takeoff_resp = self.takeoff_service(altitude=height)
147
148        #return takeoff_resp
149        return mode_resp
150
151    def land(self):
152        """
153            Set in LAND mode, which should cause the UAV to descend
154            directly,
155            land, and disarm.
```

```

148     """
149         resp = self.mode_service(custom_mode="9")
150         self.disarm()
151
152 def simple_demo():
153     """
154         A simple demonstration of using mavros commands to control a
155         UAV.
156     """
157
158     c = MavController()
159     rospy.sleep(1)
160
161     while True:
162         print(c.euler)
163
164     tol_pos = 0.1
165
166     alt = float(raw_input('Enter altitude\n'))
167     print("actual position",c.pose.position)
168     print("actual orientation",c.euler)
169
170     #take off and wait until it reaches the desired position
171     c.takeoff(alt)
172     error_alt = abs(c.pose.position.z - alt)
173     while error_alt > tol_pos :
174         error_alt = abs(c.pose.position.z - alt)
175         print(c.pose.position.z)
176
177
178     rospy.sleep(1)
179     while True :
180         ch = getch()
181
182         if ch == 'z':
183             c.goto_xyz_rpy(c.pose.position.x + 0.25,c.pose.position
184 .y,c.pose.position.z,0,0,c.euler[2])
185             print('command:',c.pose.position.x + 0.25,c.pose.
186 position.y,c.pose.position.z,0,0,c.euler[2])
187             if ch == 's':
188                 c.goto_xyz_rpy(c.pose.position.x - 0.25,c.pose.position
189 .y,c.pose.position.z,0,0,c.euler[2])
190                 print('command:',c.pose.position.x - 0.25,c.pose.
191 position.y,c.pose.position.z,0,0,c.euler[2])
192
193             if ch == 'd':
194                 c.goto_xyz_rpy(c.pose.position.x, c.pose.position.y +
195 0.25, c.pose.position.z,0,0,c.euler[2])
196                 print('command:',c.pose.position.x, c.pose.position.y +
197 0.25, c.pose.position.z,0,0,c.euler[2])
198             if ch == 'q':
199                 c.goto_xyz_rpy(c.pose.position.x, c.pose.position.y -
200 0.25, c.pose.position.z,0,0,c.euler[2])
201                 print('command:',c.pose.position.x, c.pose.position.y -
202 0.25, c.pose.position.z,0,0,c.euler[2])
203             if ch == 'u':
204                 c.goto_xyz_rpy(c.pose.position.x, c.pose.position.y, c.
205

```

```

    pose.position.z + 0.25, 0, 0, c.euler[2])
        print('command:',c.pose.position.x, c.pose.position.y,
c.pose.position.z + 0.25, 0, 0, c.euler[2])
    if ch == 'h':
        c.goto_xyz_rpy(c.pose.position.x, c.pose.position.y, c.
pose.position.z - 0.25, 0, 0, c.euler[2])
        print('command:',c.pose.position.x, c.pose.position.y,
c.pose.position.z - 0.25, 0, 0, c.euler[2])
    if ch == 'a':
        c.goto_xyz_rpy(c.pose.position.x, c.pose.position.y, c.
pose.position.z, 0, 0, c.euler[2] + pi_2/2)
        print('command:',c.pose.position.x, c.pose.position.y,
c.pose.position.z, 0, 0, c.euler[2] + pi_2/2)
    if ch == 'e':
        c.goto_xyz_rpy(c.pose.position.x, c.pose.position.y, c.
pose.position.z, 0, 0, c.euler[2] - pi_2/2)
        print('command:',c.pose.position.x, c.pose.position.y,
c.pose.position.z, 0, 0, c.euler[2] - pi_2/2)
207     if ch =='l':
208         break
210
211     print("actual position",c.pose.position)
212     print("actual orientation",c.euler)
213
214
215     print("Landing")
216     c.land()
217
218 if __name__=="__main__":
219     simple_demo()

```

## C.5 Machine Vision

### C.5.1 Gradient

```

1 #gradient calculation module
2
3 import numpy as np                                # fundamental package for
4     scientific computing
4 import matplotlib.pyplot as plt                   # 2D plotting library
5     producing publication quality figures
5 import pyrealsense2 as rs                         # Intel RealSense cross-
6     platform open-source API
6 import cv2 as cv, cv2
7 from cv_bridge import CvBridge, CvBridgeError
8 import time
9
10 import rospy
11 from sensor_msgs.msg import Image
12
13 #initialize global buffers for moving average
14 global grad_bufferXL
15 global grad_bufferXR
16 buffer_size=10
17 grad_bufferXL =  np.arange(0,buffer_size,1)
18 grad_bufferXR =  np.arange(0,buffer_size,1)

```

```

19
20 # Instantiate CvBridge
21 bridge = CvBridge()
22
23 def image_callback(msg):
24
25     try:
26         # Convert your ROS Image message to OpenCV2
27         cv2_img = bridge.imgmsg_to_cv2(msg)
28         depth_array = np.array(cv2_img, dtype=np.float32)
29         depth_array = depth_array/np.max(depth_array)
30     except CvBridgeError, e:
31         print(e)
32     else:
33         process(depth_array)
34 def N_sobel(d_image): #calculates sobel in x and y directions
35
36     img_sx = cv2.Sobel(d_image, cv2.CV_32F, 1, 0, ksize=3)
37     img_sy = cv2.Sobel(d_image, cv2.CV_32F, 0, 1, ksize=3)
38
39     img_sx = img_sx/np.max(img_sx)
40     img_sy = img_sy/np.max(img_sy)
41
42     return(img_sx,img_sy)
43
44 def grad_avg_LR(img_sx,img_sy):# calculates gradient average by
45     #sector and direction #xL,xR; yL,yR
46
47     global grad_bufferXL
48     global grad_bufferXR
49
50     img_sx_n=np.sign(img_sx)
51     img_sy_n=np.sign(img_sy)
52     avg = { 'Lx' : 0 , 'Ly' : 0 ,
53             'Rx' : 0 , 'Ry' : 0 ,
54             'Tx' : 0 , 'Ty' : 0 }
55
56     #Lx
57     avg[ 'Lx' ] = np.mean(img_sx_n[:, :img_sx_n.shape[1]/2])
58     #Ly
59     avg[ 'Ly' ] = np.mean(img_sy_n[:, :img_sy_n.shape[1]/2])
60     #Rx
61     avg[ 'Rx' ] = np.mean(img_sx_n[:, img_sx_n.shape[1]/2 :])
62     #Ry
63     avg[ 'Ry' ] = np.mean(img_sy_n[:, img_sy_n.shape[1]/2 :])
64
65
66
67
68     avg[ 'Lx' ],grad_bufferXL= MovAvg(avg[ 'Lx' ],grad_bufferXL) #
69     #filter
70     avg[ 'Rx' ],grad_bufferXR= MovAvg(avg[ 'Rx' ],grad_bufferXR) #
71     #filter
72
73     #Tx
74     avg[ 'Tx' ] = (avg[ 'Lx' ]+avg[ 'Rx' ])/2
75     #Ty

```

```

74     avg[ 'Ty' ] = (avg[ 'Ly' ]+avg[ 'Ry' ])/2
75
76
77
78     return(avg)
79
80 def draw_grid(image_bgr,density=10):#draws grid
81     h,w,c =image_bgr.shape
82     dy=(int)(h/density)
83     dx=(int)(w/density)
84     for y in range(0,h,dy):
85         cv2.line(image_bgr, (0, y), (w, y), (255, 0, 0), 1, 1)
86     for x in range(0,w,dx):
87         cv2.line(image_bgr, (x, 0), (x, h), (255, 0, 0), 1, 1)
88     return(image_bgr)
89
90 def dir_draw_axe(image_bgr,avg_x,avg_y,factor=50,axe = 'x',
91                   divergence = False):#draws total gradient and allows to choose
92     axe
93     h,w,c=image_bgr.shape
94     av_X = avg_x*factor
95     av_Y = avg_y*factor/2
96     start = ((int)(w/2),(int)(h/2))
97     endx = ( (int)(w/2) + (int)(av_X) , (int)(h/2) )
98     endy = ( (int)(w/2) , (int)(h/2) + (int)(av_Y) )
99
100    if axe == 'x' :
101        if not divergence:
102            if np.abs(av_X) > 0.02*w:
103                image_bgr_dir = cv2.arrowedLine(image_bgr,start,endx,(0,
104                                                255, 0),thickness=2,tipLength = 0.2 )
105            else:
106                image_bgr_dir = cv2.circle(image_bgr, start, 30,
107                                           (255,0,0),thickness=2)
108        else:
109            image_bgr_dir = cv.putText(image_bgr, "X", ((int)(w/2),(int)
110                                         (h/2)), cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),3)
111
112    if axe == 'y' :
113        image_bgr_dir = cv2.arrowedLine(image_bgr,start,endy,(0, 255,
114                                                0),thickness=2,tipLength = 0.2 )
115    if axe == 'xy' :
116        end = ( (int)(w/2) + (int)(av_X), (int)(h/2) + (int)(av_Y) )
117        image_bgr_dir = cv2.arrowedLine(image_bgr,start,endx,(255, 0,
118                                                0),thickness=2,tipLength = 0.2 )
119        image_bgr_dir = cv2.arrowedLine(image_bgr_dir,start,endy,(0,
120                                                255, 0),thickness=2,tipLength = 0.2 )
121
122    # if (np.sqrt(av_X**2+av_Y**2)>0.001*factor):
123    #     image_bgr_dir = cv2.arrowedLine(image_bgr,start,end,(255,
124                                                0, 0),thickness=2,tipLength = 0.2 )
125    # else:
126    #     image_bgr_dir = cv2.circle(image_bgr, start, 30,
127                                           (255,0,0),thickness=2)
128    return(image_bgr_dir)
129
130 def dir_draw_LR(image_bgr,avg,factor=50):#draws left and right

```

```

gradient in x direction ,searchs for divergence and calls
dir_draw_axe
122 h,w,c=image_bgr.shape
123 # global grad_bufferXL
124 # global grad_bufferXR
125 #Obtaining the average gradient by zone
126 av_Lx = avg['Lx']*factor
127 av_Rx = avg['Rx']*factor
128 av_Ly = avg['Ly']*factor
129 av_Ry = avg['Ry']*factor
130
131 av_X = avg['Tx']*factor
132 av_Y = avg['Ty']*factor
133
134 # ma_XL,grad_bufferXL= MovAvg(av_XL,grad_bufferXL)
135 # ma_XR,grad_bufferXR= MovAvg(av_XR,grad_bufferXR)
136
137 # av_XL = ma_XL
138 # av_XR = ma_XR
139
140 #total gradient
141 # av_X = (av_XL+av_XR)/2#np.mean(avg[0,:])
142 # av_Y = np.mean(avg[1,:])
143
144 #computing centers and ends of arroxs
145 startxL = ((int)(w/4),(int)(h/2))
146 startxR = ((int)(w*3/4),(int)(h/2))
147 endxL = ( (int)(w/4) + (int)(av_Lx) , (int)(h/2) )
148 endxR = ( (int)(w*3/4) + (int)(av_Rx) , (int)(h/2) )
149
150 if True:#np.abs(av_XL) > 0.02*w: #draw arrow if module bigger
than tolerance LEFT
151     image_bgr_dir = cv2.arrowedLine(image_bgr,startxL,endxL,(255,
0, 0),thickness=2,tipLength = 0.2 )
152 else: #else draw circle, meaning that we are in the center of
the zone LEFT
153     image_bgr_dir = cv2.circle(image_bgr, startxL, 30, (255,0,0),
thickness=2)
154 if True:#np.abs(av_XR) > 0.02*w: #draw arrow if module bigger
than tolerance RIGHT
155     image_bgr_dir = cv2.arrowedLine(image_bgr,startxR,endxR,(255,
0, 0),thickness=2,tipLength = 0.2 )
156 else: #else draw circle, meaning that we are in the center of
the zone RIGTH
157     image_bgr_dir = cv2.circle(image_bgr, startxR, 30, (255,0,0),
thickness=2)
158
159
160 #check for divergence between left and right zone
161 if np.sign(av_Lx) > 0 and np.sign(av_Rx) < 0 :
162     divergence = False
163 elif np.sign(av_Lx) < 0 and np.sign(av_Rx) > 0 :
164     divergence = True
165 else:
166     divergence = False
167
168 #draw total gradient
169 image_bgr_dir = dir_draw_axe(image_bgr_dir,av_X,av_Y,divergence)

```

```

        = divergence , factor=10)

170
171
172
173     return(image_bgr_dir)
174
175
176 def MovAvg(value,grad_buffer): #calculates moving average of given
177     global array
178
179     amount = np.arange(0,len(grad_buffer),1)**2
180     grad_buffer = np.delete( grad_buffer,0)
181     grad_buffer = np.append(grad_buffer, value)
182     ma = np.average(grad_buffer, weights=amount)
183     return ma,grad_buffer
184
185
186 def gradient(image):
187     #density = 16 #1; 2; 3; 4; 5; 6; 8; 10; 12; 15; 20; 24; 30; 40;
188     # 60 COMMON FACTORSS
189
190     img_sx,img_sy = N_sobel(image)
191
192
193     avg = grad_avg_LR(img_sx,img_sy)
194
195     #avgs = grad_avg(img_sx,img_sy,density)
196
197     #avg_x,avg_y = grad_avg_b(img_sx,img_sy)
198
199     #image_bgr = draw_grid(image_bgr,density)
200
201     #image_bgr = avg_draw(image_bgr,avgs,100) #compute with grid
202
203     #image_bgr_dir = dir_draw(image_bgr,avgs)
204
205     #image_bgr_dir = dir_draw_axe(image_bgr,avg_x,avg_y) #compute
206     #with direct avg over img_sX
207     image_bgr = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
208     image_bgr_dir = dir_draw_LR(image_bgr,avg) #compute with direct
209     # avg over img_sX
210     return image_bgr_dir,avg

211
212
213 def process(image):
214     colorized_depth = image
215
216     # colorized_depth = cv.cvtColor(colorized_depth, cv.COLOR_BGR2GRAY
217     # )
218
219
220     colorized_depth = cv.GaussianBlur(colorized_depth,(9,9),0)
221     img_bgr_f,avg = gradient(colorized_depth)
222
223     #img_bgr_f = cv.resize(img_bgr_f,(3*img_bgr_f.shape[1], 3*
224     #     img_bgr_f.shape[0]), interpolation = cv.INTER_CUBIC)
225     cv.imshow('grad',img_bgr_f)

```

```

221 # print("--- %s seconds - Hz---" % (time.time() - start_time), 1/(
222     time.time() - start_time))
223 #cv.imshow('grad_o',img_bgr_o)
224 # start_time = time.time()
225 #print(np.max(colorized_depth))
226 cv.waitKey(1)
227 return avg
228
229
230 def main():
231
232     # Setup:
233     # pipe = rs.pipeline()
234     # cfg = rs.config()
235
236     # Start streaming
237     # cfg.enable_stream(rs.stream.depth, 640, 480, rs.format.z16,
238     # 30)
239     #cfg.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8,
240     # 30)
241
242     # Skip 5 first frames to give the Auto-Exposure time to adjust
243     # for x in range(5):
244     #     pipe.wait_for_frames()
245
246     # colorizer = rs.colorizer(3)#config colorizer type 3=black to
247     # white
248
249     # Start streaming
250     # pipe.start(cfg)
251
252     #initialize global buffers for moving average
253     global grad_bufferXL
254     global grad_bufferXR
255     buffer_size=10
256     grad_bufferXL = np.arange(0,buffer_size,1)
257     grad_bufferXR = np.arange(0,buffer_size,1)
258
259     global image
260
261     rospy.init_node("Gradient")
262     rospy.Subscriber("/d400/depth/image_rect_raw", Image,
263     image_callback)
264
265     start_time = time.time()
266     # Store next frameset for later processing:
267
268     rospy.spin()
269
270     #while True:
271
272         # frame_f,original = get_filtered(pipe,filters_init())
273
274         # colorized_depth = np.asarray(colorizer.colorize(
275         # frame_f).get_data())
276
277         # colorized_depth = image
278         # cv2.imshow('test',image)

```

```
273         # #colorized_depth = cv.cvtColor(colorized_depth, cv.
274 COLOR_BGR2GRAY)
275
276         # colorized_depth = cv.GaussianBlur(colorized_depth,(9,9)
277 ,0)
278         # img_bgr_f,avg = gradient(colorized_depth)
279
280         # img_bgr_f = cv.resize(img_bgr_f,(3*img_bgr_f.shape[1], 3*
281 img_bgr_f.shape[0]), interpolation = cv.INTER_CUBIC)
282         # cv.imshow('grad',img_bgr_f)
283         # print("--- %s seconds - Hz- "% (time.time() - start_time)
284 , 1/(time.time() - start_time))
285         # #cv.imshow('grad_o',img_bgr_o)
286         # start_time = time.time()
287         # #print(np.max(colorized_depth))
288         # cv.waitKey(1)
289
290
291
292 if __name__ == '__main__':
293
294     # def filters_init():#initializes filters
295     #     #creates a dictionary with the filters and configures its
296     #     #parameters
297     #     filters = {}
298     #     filters["decimation"] = rs.decimation_filter()
299
300     #     filters["depth_to_disparity"] = rs.disparity_transform(True
301 )
302
303     #     filters["spatial"] = rs.spatial_filter()
304     #     filters["spatial"].set_option(rs.option.filter_magnitude,
305 5)
306     #     filters["spatial"].set_option(rs.option.
307 filter_smooth_alpha, 1)
308     #     filters["spatial"].set_option(rs.option.
309 filter_smooth_delta, 50)
310     #     filters["spatial"].set_option(rs.option.holes_fill, 3)
311
312     #     filters["disparity_to_depth"] = rs.disparity_transform(
313 False)
314
315     #     filters["hole_filling"] = rs.hole_filling_filter()
316
317     #     return filters
318
319
320     # def get_filtered(pipe,filters): #gets filtered and original
321 frames from camera
322     #     #recieves the pipe to get the transmission and a
323     #     #dictionary with the corresponding filters
324     #     #and implements them
325     #     frameset = pipe.wait_for_frames()
326     #     frame = frameset.get_depth_frame()
327     #     frame_original=frame
```

```

319     # I can't execute the following with a for because
320     # dictionaries aren't ordered
321     # frame = filters["decimation"].process(frame)
322     # frame = filters["depth_to_disparity"].process(frame)
323     # frame = filters["spatial"].process(frame)
324     # #frame = temporal.process(frame)
325     # frame = filters["disparity_to_depth"].process(frame)
326     # frame = filters["hole_filling"].process(frame)
327
328     # return frame,frame_original
329
329 main()

```

## C.5.2 Interpreter

```

1 import numpy as np                                # fundamental package for
2   scientific computing
3 import matplotlib.pyplot as plt                  # 2D plotting library
4   producing publication quality figures
5 import pyrealsense2 as rs                         # Intel RealSense cross-
6   platform open-source API
7 import cv2 as cv, cv2
8 import time
9 import openspace as op
10 import depthgradient_ROS as gd
11 import rospy
12 from std_msgs.msg import String
13 from sensor_msgs.msg import Image
14 from cv_bridge import CvBridge, CvBridgeError
15
16 print("Environment Ready")
17
18 def Publisher(avg):
19
20     pub = rospy.Publisher('avg', String, queue_size=10)
21     #rospy.init_node('Interpreter', anonymous=True)
22     rate = rospy.Rate(5) # 5hz
23     if not rospy.is_shutdown():
24         message = str(avg["Tx"])
25         rospy.loginfo(message)
26         pub.publish(message)
27
28
29 def image_callback(msg):
30
31     try:
32         # Convert your ROS Image message to OpenCV2
33         cv2_img = bridge.imgmsg_to_cv2(msg)
34         depth_array = np.array(cv2_img, dtype=np.float32)
35         depth_array = depth_array/np.max(depth_array)
36     except CvBridgeError, e:
37         print(e)
38     else:
39         avg = gd.process(depth_array)
40         try:

```

```

41         Publisher(avg)
42     except rospy.ROSInterruptException:
43         pass
44
45
46 def main():
47
48
49
50     start_time = time.time() #to check preformance
51     rospy.init_node("Gradient")
52     rate = rospy.Rate(5)
53     rospy.Subscriber("/d435/depth/image_rect_raw", Image,
54     image_callback)
55
56
57     rospy.spin()
58
59
60
61
62
63 if __name__ == '__main__':
64     main()

```

### C.5.3 Évasion d'obstacles

```

1 #calcul de trajectoire
2 import numpy as np                                     # fundamental package for
3                                         scientific computing
4 import matplotlib.pyplot as plt                        # 2D plotting library
5                                         producing publication quality figures
6 import pyrealsense2 as rs                             # Intel RealSense cross-
7                                         platform open-source API
8 import cv2 as cv, cv2
9 import time
10
11
12 def os_filters_init():#initializes filters
13     #creates a dictionary with the filters and configures its
14     parameters
15     filters = {}
16     filters["decimation"] = rs.decimation_filter()
17
18     filters["depth_to_disparity"] = rs.disparity_transform(True)
19
20     filters["spatial"] = rs.spatial_filter()
21     filters["spatial"].set_option(rs.option.filter_magnitude, 5)
22     filters["spatial"].set_option(rs.option.filter_smooth_alpha, 1)
23     filters["spatial"].set_option(rs.option.filter_smooth_delta,
24     50)
25     filters["spatial"].set_option(rs.option.holes_fill, 3)
26
27     filters["disparity_to_depth"] = rs.disparity_transform(False)

```

```

26     filters["hole_filling"] = rs.hole_filling_filter()
27
28     return filters
29
30
31 def os_get_filtered(pipe,filters): #gets filtered and original
32     frames from camera
33     #recieves the pipe to get the transmission and a dictionary with
34     #the corresponding filters
35     #and implements them
36     frameset = pipe.wait_for_frames()
37     frame = frameset.get_depth_frame()
38     frame_original=frame
39     #I can't execute the following with a for because dictionaries
40     #aren't ordered
41     frame = filters["decimation"].process(frame)
42     frame = filters["depth_to_disparity"].process(frame)
43     frame = filters["spatial"].process(frame)
44     #frame = temporal.process(frame)
45     frame = filters["disparity_to_depth"].process(frame)
46     frame = filters["hole_filling"].process(frame)
47
48     return frame,frame_original,frameset
49
50
51 def get_factor(gray_depth,frame_f,value=50): #returns conversion
52     factor between real depth and grayscale and the value used to
53     calculate it (you can specify it), by default is 200 because of
54     the noise
55     #search index of a pixel with value 200
56     h,w = gray_depth.shape
57     value_indexes = np.where(gray_depth == value)
58     listOfCoordinates= list(zip(value_indexes[0], value_indexes[1]))
59
60     while not listOfCoordinates:
61         value=value-1
62         value_indexes = np.where(gray_depth == value)
63         listOfCoordinates= list(zip(value_indexes[0], value_indexes[1]))
64
65     point= listOfCoordinates[0]
66     y_p = point[0]
67     x_p = point[1]
68     point_depth = frame_f.as_depth_frame().get_distance((int(x_p)),
69     (int(y_p)))
70     factor = point_depth/value
71
72     return(factor)
73
74 def get_factor_b(gray_depth,frame_f):
75     h,w = gray_depth.shape
76     step = h/16
77     i=0
78     factor = np.zeros([400])
79     for y in range(0,h,step):
80         for x in range(0,w,step):
81             depth = frame_f.as_depth_frame().get_distance(x,y)
82             value =gray_depth[y,x]

```

```

75         factor[i]=value/depth+0.001
76         i+=1
77
78     return np.mean(factor)
79
80
81 def factor_error_m(gray_depth,frame_f,factor):
82     h,w = gray_depth.shape
83     R_depth = frame_f.as_depth_frame().get_distance((int(w/2)),(int(h/2)))
84     E_depth = gray_depth[(int(h/2)), (int(w/2))]*factor
85     error = R_depth-E_depth
86     return error
87
88 def m_to_px(depth_intrin,size,depth,axe=0):
89
90     if axe == 0:
91         start = [-size/2,0,depth]
92         end = [size/2,0,depth]
93         start_px = rs.rs2_project_point_to_pixel(depth_intrin,start)
94         end_px = rs.rs2_project_point_to_pixel(depth_intrin,end)
95         return (end_px[0]-start_px[0])
96     else:
97         start = [0,-size/2,depth]
98         end = [0,size/2,depth]
99         start_px = rs.rs2_project_point_to_pixel(depth_intrin,start)
100        end_px = rs.rs2_project_point_to_pixel(depth_intrin,end)
101        return (end_px[1]-start_px[1])
102
103 def find_free_path(image,depth_intrin,drone_w,drone_h,layer_depth,
104 step=25):
105     layer = image.copy()
106     h,w = layer.shape
107     rw= int(m_to_px(depth_intrin,drone_w,layer_depth,0))
108     rh= int(m_to_px(depth_intrin,drone_h,layer_depth,1))
109     paths = np.zeros([100,2])
110     i=0
111     for y in range(0,h-rh,step):
112         for x in range(0,w-rw,step):
113             sector = layer[y:y+rh,x:x+rw]
114             if np.all(sector>10):
115                 layer[y:y+rh,x:x+rw] = 175
116                 paths[i,...]=[int(x+rw/2),int(y+rh/2)]
117     layer[layer<10] = 255
118     layer[layer<200] = 0
119     contours, hier = cv2.findContours(np.abs(layer-255), cv2.
120     RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
121     return contours,paths,layer
122
123 def get_bin_layer(gray_depth,factor,depth):
124     ret,thresh = cv.threshold(gray_depth,depth/factor,255,cv.
125     THRESH_BINARY)
126     kernel=np.ones([3,3])
127     thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel,
128     iterations=10)
129     return thresh

```

```

126
127 def get_rectangle(drone_w = 0.50, drone_h = 0.20, drone_depth =
128     0.25):
129     global depth_intrin
130     portal_center = [0,0,drone_depth]
131     portal_topL = [-drone_w/2,-drone_h/2,drone_depth]
132     portal_bottomR = [drone_w/2,drone_h/2,drone_depth]
133
134     #find rectangle in pixel dimentions
135     uvtl = rs.rs2_project_point_to_pixel(depth_intrin,portal_topL)
136     uvbr = rs.rs2_project_point_to_pixel(depth_intrin,
137     portal_bottomR)
138     #center = rs.rs2_project_point_to_pixel(depth_intrin,
139     portal_center)
140
141     #round values
142     uvtl_r = [int(round(x)) for x in uvtl]
143     uvbr_r = [int(round(x)) for x in uvbr]
144     #center = [int(round(x)) for x in center]
145     rectangle = np.zeros([2,2],dtype=int)
146     rectangle[0,:] = uvtl_r
147     rectangle[1,:] = uvbr_r
148     return rectangle
149
150
151 def main():
152
153     # Setup:
154     pipe = rs.pipeline()
155     cfg = rs.config()
156
157     global depth_intrin
158
159     # Start streaming
160     cfg.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
161     #cfg.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8,
162     30)
163
164
165     colorizer = rs.colorizer(3)#config colorizer type 3=black to
166     white
167
168     # Start streaming
169     pipe.start(cfg)
170
171
172     # Store next frameset for later processing:
173     while True:
174
175         frame_f,original,frameset = os_get_filtered(pipe,
176         os_filters_init())
177
178         colorized_depth = np.asarray(colorizer.colorize(frame_f)
179         .get_data())
180         colorized_depth = cv.GaussianBlur(colorized_depth,(9,9),0)
181
182         gray_depth = cv.cvtColor(colorized_depth,cv.COLOR_BGR2GRAY)
183         colorized_depth_o = np.asarray(colorizer.colorize(
184         original).get_data())

```

```
176
177     h,w = gray_depth.shape #240x320
178
179     factor = get_factor(gray_depth,frame_f,50)
180
181
182     #if factor equals zero, we ignore the frame
183     if factor != 0 :
184
185         # test_point=(300,100)
186         depth_intrin = frame_f.profile.as_video_stream_profile()
187         .intrinsics
188
189         # R_depth = frame_f.as_depth_frame().get_distance((int
190         # (300)), (int(100)))
191
192         # depth_point = rs.rs2_deproject_pixel_to_point(
193         # depth_intrin,[0,0], R_depth)#intrinsics, pixel coord (y,x) and
194         # depth coord
195
196         drone_w = 0.50
197         drone_h = 0.20
198         drone_depth = 0.25
199
200         rectangle = get_rectangle(drone_depth = 0.5)
201
202
203         thresh1 = get_bin_layer(gray_depth,factor,0.5)
204         thresh2 = get_bin_layer(gray_depth,factor,0.5)
205
206         colorized_depth[thresh1<10,2] += 50
207
208         contours1,paths1,image_path = find_free_path(thresh2,
209         depth_intrin,drone_w,drone_h,0.5)
210
211
212         cv2.drawContours(colorized_depth, contours1, -1, (0,
213         255, 0), 1)
214
215         if np.any(thresh1[rectangle[0,1]:rectangle[1,1],
216         rectangle[0,0]:rectangle[1,0]] <10):
217             colorized_depth = cv.rectangle(colorized_depth ,
218             (rectangle[0,0],rectangle[0,1]), (rectangle[1,0],rectangle[1,1]),
219             (0,0,255),1)
220         else:
221             colorized_depth = cv.rectangle(colorized_depth ,
222             (rectangle[0,0],rectangle[0,1]), (rectangle[1,0],rectangle[1,1]),
223             (0,255,0),1)
224
225         colorized_depth = cv.resize(colorized_depth,(2*
226         colorized_depth.shape[1], 2*colorized_depth.shape[0]),
227         interpolation = cv.INTER_CUBIC)
```

```
221         cv.imshow("colorized",colorized_depth)
222
223
224
225
226
227     # cv.imshow("binary",binary_1)
228
229
230     cv.waitKey(1)
231
232
233     pipe.stop()
234
235
236 if __name__ == '__main__':
237     main()
```