الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique

Département d'Automatique

Mémoire de projet de fin d'études pour l'obtention du diplôme d'ingénieur d'état en automatique

Navigation autonome d'un drone quadrirotor en milieu intérieur

Realisé par :
Mr. Berghiche Mohamed Amine
Mr. Bousteila Islem

Encadré par :
Mr. Stihi Omar

Membres du jury :

Président	Mr. Hakim Achour
Promoteur	Mr. Omar Stihi
Examinateur	Pr. Samir Ladaci

ENP 2022

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique

Département d'Automatique

Mémoire de projet de fin d'études pour l'obtention du diplôme d'ingénieur d'état en automatique

Navigation autonome d'un drone quadrirotor en milieu intérieur

Realisé par :
Mr. Berghiche Mohamed Amine
Mr. Bousteila Islem

Encadré par :
Mr. Stihi Omar

Membres du jury :

Président	Mr. Hakim Achour
Promoteur	Mr. Omar Stihi
Examinateur	Pr. Samir Ladaci

ENP 2022

ملخص

يناقش هذا العمل مفهوما معتمدا في مجال الروبوتات : الاستقلالية. في الواقع، هدفنا هو إنشاء نموذج لملاحة طائرة بدون طيار رباعية المحركات تعتمد على وحدة التحكم Pixhawk4. أولاً ، قمنا بتصميم نظام ملاحة باستخدام خوارزمية SLAM بالاضافة إلى مجس لقياس إرتفاع الطائرة ولوحة NVIDIA التي بدورها تنسق بين مختلف الأنظمة، زد على ذلك توفير موقع ثنائي الأبعاد للملاحة الداخلية. ثم بعد ذلك، نقوم بتنفيذ مخطط للمسار من أجل تجنب العقبات على الطريق بالاعتماد على نظرية الحقول المحتملة.

كلمات مفتاحية البيئة الداخلية ، الاستقلالية ، SLAM ، تجنب العقبات ، نظرية الحقول المحتملة

Abstract

This work focuses on a modern aspect in mobile robotics : "**Autonomy**". Basically, the ultimate goal of this work is to come out with a navigation system for a quad-rotor UAV based on the flight controller Pixhawk4. First, we designed an indoor navigation system using a 2D lidar, an altimeter and NVIDIA board that coordinates between the different components of the systems as well as running a SLAM algorithm to provide a 2D localisation for indoor navigation. Moreover, we implement a path planning algorithm based on potential field theory to avoid obstacles on the way.

Keywords : Quadrotor, Localisation, Indoor Navigation, SLAM, Autonomy, Path planning, Potential Field.

Resumé

Ce travail traite un aspect moderne en robotique mobile : "**l'Autonomie**". En réalité, l'objectif de ce travail est l'implémentation d'un prototype de navigation pour un drone quadri-rotor basé sur le microcontrolleur Pixhawk4. Dans un premier temps, on conçoit un système de navigation en milieu intèrieur à l'aide d'un lidar 2D, un altimètre et une carte NVIDIA qui exécute un algorithme SLAM pour se localiser en milieu intérieur selon x-y. La carte NVIDIA joue le rôle de maître, étant le coordinateur principal entre les différents composants du système. Dans un second temps, on met en pratique un programme d'évitement d'obstacle qui se base sur la théorie des champs potentiels artificels.

Mots clés : Quadrirotor, Localisation, Milieu Intérieur, Autonomie, SLAM, Evitement d'Obstacle, Champs Potentiels

Dedicace

“

À nos chers parents, nos familles et nos amis

”

Mohamed & Islem

Remerciements

Avant tout, nous remercions Dieu le Tout-Puissant de nous avoir donné le courage, la volonté et la patience pour mener à bien ce travail.

Nous remercions nos chers parents qui nous ont soutenu tout au long de notre parcours d'étude.

Nous tenons à exprimer notre gratitude à notre encadrant, M. Omar STIHI, pour son soutien et ses conseils au long du déroulement de cette thèse.

Nous remercions ainsi tous nos enseignants d'avoir rendu toutes nos leçons faciles et amusantes. Nous avions toujours hâte d'assister à leurs cours et de nous inspirer par leur encadrement modèle.

Nous tenons également à remercier par avance Monsieur Mourad ADNANE et Monsieur Iskander ZOUAGHI, pour leurs soutient concrét et moral.

Chacun d'entre nous tient également à remercier l'autre, d'avoir eu le courage et la patience de se lancer dans ce défi jusqu'au bout malgré tous les difficultés qui se sont présenté.

Table des matières

Liste des Figures

Liste des Tablaux

Liste d'Abréviations

1 Introduction **13**

2 Commande et Observation **16**

2.1 Modélisation du quadri-rotor 17

2.1.1 Repérage du quadri-rotor dans l'espace 17

2.1.2 Matrice de rotation 18

2.2 Modèl dynamique 19

2.2.1 Hypothèses simplificatrices 19

2.2.2 La dynamique de translation 19

2.2.3 La dynamique de rotation 20

2.3 Synthèse de la commande PD 21

2.3.1 La structure de contrôle 22

2.3.2 Contrôle des angles d'orientation 23

2.3.3 Contrôle de position 24

2.3.3.1 Contrôle à l'équilibre 25

2.3.3.2 Contrôle d'une trajectoire 3-D 25

2.3.4 Resultats de simulation d'un quadri-rotor sur Matlab 25

2.3.5 Résultats de simulation d'un quadri-rotor basé sur Pixhawk4 . . . 32

2.3.6 Conclusion 34

2.4 Estimation d'état par un filtre de Kalman étendu 35

2.4.1 Modèle d'observation du quadri-rotor par un filtre de Kalman étendu 35

2.4.2 L'algorithme du filtre 38

2.4.2.1 la phase de prédiction 38

2.4.2.2 La phase de correction 39

3 Localisation et Cartographie Simultanées 41

3.1 Définition 42
3.1.1 Les origines 42
3.1.2 Formulation du problème SLAM 43
3.1.3 La localisation 44
3.1.4 La cartographie 45
3.1.5 Le SLAM 45
3.2 Résolution du SLAM 46
3.2.1 Représentation de la carte 46
3.2.2 L'approche directe 46
3.2.3 L'approche basée sur les caractéristique géométrique 47
3.2.4 L'approche basée sur une grille d'occupation 47
3.2.5 La mise à jour de la carte 48
3.3 Hector SLAM 49
3.3.1 L'estimation de la position par la mise en correspondance 50
3.3.1.1 Amer-amer 50
3.3.1.2 Scan-amer 50
3.3.1.3 Scan-scan 50
3.3.2 La localisation par Hector SLAM 50
3.3.3 Le calcul de la probabilité 51
3.3.4 La mise en correspondance 52
3.4 Résultats de simulation 54
3.5 Conclusion 54

4 Évitement d'obstacle 55

4.1 Problématique 56
4.2 Pourquoi champ de potentiel 57
4.3 Le principe de la méthode 57
4.3.1 Potentiel attractif 59
4.3.2 Potentiel répulsif 60
4.3.3 Potentiel total 61
4.4 Implémentation de l'algorithme des champs potentiels 61
4.5 Probléme du minimum local 62
4.6 Résultats de simulation 63
4.6.1 Fixer un but à l'origine en présence d'un obstacle 63

4.6.1.1 Interprétation 65
4.6.2 Fixer un but à (4,-6) en présence d'un obstacle 65
4.6.2.1 Interprétation 67
4.6.3 Fixer un but à (4,-6) en présence de deux obstacles 67
4.6.3.1 Interprétation 68

5 Description de matériel et logiciel 69
5.1 Présentation du quadri-rotor 70
5.2 Description des matériels et logiciels utilisés 70
5.2.1 Le matériel 70
5.2.1.1 Contrôleur de vol PIXHAWK4 71
5.2.1.2 NVIDIA JETSON NANO 71
5.2.1.3 Le Châssis 72
5.2.1.4 Moteur sans balais (BLDC) 73
5.2.1.5 Variateur de vitesse (ESC) 74
5.2.1.6 Les hélices 75
5.2.1.7 Carte répartition d'alimentation PM07 76
5.2.1.8 Une batterie au lithium polymère 77
5.2.1.9 YDLIDAR X4 78
5.2.1.10 Un altimetre ultrason MAXIBOTIX MB 1043 79
5.2.1.11 Radio télécommande FLYSKY FS-i10 80
5.2.2 Les logiciels 81
5.2.2.1 Mission Planner 81
5.2.2.2 ROS 82
5.2.2.3 Gazeboo 82
5.2.2.4 MAVROS 82
5.2.2.5 PX4-Autopilot 82
5.2.2.6 HECTOR SLAM 83
5.2.2.7 Le package tf 83
5.3 Description de l'environnement de simulation : 83
5.4 Description de l'implémentation pratique 87
5.4.1 Les étapes d'implémentation 88
5.4.2 Problème téchnique 89

6 Conclusion 90

7 Annexe 95

7.1 Paramètres utilisés en simulation 96
7.2 Codes développés 97

Table des figures

1.1 Plateforme du quadri-rotor equipée par un PIXHAWK4, une carte NVIDIA, un capteur sonore et un lidar 2D 15

2.1 Repérage du système et les forces/moments qui s'appliquent sur le quadri-rotor 17

2.2 la structure de contrôle 22

2.3 Structure de contrôle PD plus détaillée 24

2.4 Evolution en 3D du quadri-rotor dans l'espace 26

2.5 Evolution de la position du quadri-rotor en fonction du temps 26

2.6 Evolution de la vitesse du quadri-rotor en fonction du temps 27

2.7 Evolution de l'erreur de position du quadri-rotor en fonction du temps . . 27

2.8 Evolution des signaux de commande en fonction du temps 28

2.9 Etapes de translation d'un quadri-rotor 29

2.10 Evolution en 3D du quadri-rotor en fonction du temps 30

2.11 Evolution de la position du quadri-rotor en fonction du temps 30

2.12 Evolution de la vitesse du quadri-rotor en fonction du temps 31

2.13 Evolution de l'erreur de position du quadri-rotor en fonction du temps . . 31

2.14 Evolution des signaux de commande en fonction du temps 32

2.15 A gauche : la réponse du drone de simulation à la consigne de position (1,0,2.5) A droite : la réponse du drone de simulation à la consigne de position (0,1,2.5) 32

2.16 En haut : la réponse du drone à la consigne de position (2,0,2.5) En bas : la réponse du drone à la consigne de position (0,2,2.5) 33

2.17 La réponse du drone à la consigne d'orientation $(0,0,-\frac{\pi}{2})$ 33

2.18 La réponse du drone à la consigne de position (2,2,2.5) 34

2.19 Schéma représentatif du filtre EKF 37

2.20 Schéma representatif détaillé du filtre EKF 40

3.1 L'idée de base du SLAM 43

3.2 la localisation 44

3.3 la cartographie 45
3.4 Représentation graphique du problématique SLAM 46
3.5 Grille d'occupation 47
3.6 mise-a-jour de la carte de grille d'occupation 48
3.7 schémas globale de fonctionnement du Hector SLAM 51
3.8 Filtrage bilinéaire de la grille d'occupation 51
3.9 A gauche : la réponse du drone de simulation à la consigne de position (1,0,2.5) A droite : la réponse du drone de simulation à la consigne de position (0,1,2.5) 54

4.1 Le champ de gradient attractif utilisé, ploté par Python 58
4.2 Le champ de gradient répulsif utilisé, ploté par Python 58
4.3 Le champ de gradient combiné, ploté par Python 59
4.4 Visualisation de la loi attractive 60
4.5 Visualisation de la loi répulsive 61
4.6 Visualisation de la loi des champs potentiels résultants 61
4.7 Schéma representatif de l'algorithme d'évitement 62
4.8 Exemple d'une loi potentielle présentant un minimum global et un minimum local 63
4.9 Drone de simulation maintient son altitude à 2.5m aprés avoir lui envoyé la commande takeoff(2.5) 64
4.10 Drone de simulation maintient sa position à l'origine avant de lui rapprocher un obstacle 64
4.11 Drone de simulation s'éloinge de sa position aprés avoir détecté un obstacle, sous l'effet des forces répulsives artificielles 64
4.12 Drone de simulation retourne à l'origine aprés l'avoir éloigné de l'obstacle, sous l'effet de la force attractive artificielle 65

5.1 Image démonstrative de la navigation en milieu intérieur du drone. 70
5.2 flight contrôleur PIXHAWK4 71
5.3 NVIDIA JETSON NANO 72
5.4 Le châssis de notre quadri-rotor. 73
5.5 Moteur holybro 2216 KV 880 74
5.6 ESC 20A BLHeli S 75
5.7 les hélices 76
5.8 La carte répartition d'alimentation 77
5.9 Battrie LIPO 78
5.10 YDLIDAR X4 79

5.11 Altimetre 80
5.12 Radio télécommande. 81
5.13 L'arbre tf 83
5.14 Plateforme du drone de simulation sur Gazebo 84
5.15 Drone de simulation sur Gazebo en milieu interieur 84
5.16 Graph des topics obtenu par la commande rqt_graph décrivant l'intéraction du système avec Hector SLAM, pour estimer la position x-y 85
5.17 Graph des noeuds obtenu par la commande rqt_graph décrivant l'intéraction du système avec le noeud $'gnc_node'$, pour effectuer l'évitement d'obstacle 86
5.18 Graph des noeuds obtenu par la commande rqt_graph décrivant l'intéraction du système avec le noeud $'offb'$, pour lui donner des consignes à suivre 87
5.19 Schéma descriptif du système 88
5.20 Schéma général du système 88

Liste des tableaux

7.1 Les paramètres physique utilisés pour le système de simulation 96
7.2 Les paramètres des lois de commande utilisées en simulation 96

Liste d'Abréviations

UAV	*Unmanned Aerial Vehicle*
APF	*Artificial Potential Field*
PD	*Proportionnel-Dérivé*
EKF	*Extended Kalman Filter*
SLAM	*Simultaneous Localization And Mapping*
ROS	*Robotic Operating System*
LIDAR	*Light Detection And Ranging*
IMU	*Inertial Measurement Unit*
ESC	*Electronic Speed Controller*
BLDC	*Brushless Motor*
LiPo	*Lithium polymer*

Chapitre 1

Introduction

Récemment, le déploiement des véhicules aériens (UAV) a attiré de plus en plus l'attention de plusieurs domaines, en particulier l'industrie minière ainsi que les applications civiles et militaires. Cela met en avant la nécessité d'une précision élevée des systèmes de navigation et de positionnement ainsi qu'une bonne adaptation aux différents environnements. De plus, des applications très innovantes ont été implémentées [1] dans le sens à utiliser des drones autonomes équipés par des capteurs afin d'effectuer des tâches de cartographie ou d'inspection dans des milieux fermés où il n y a pas accès au GPS (Global Positioning System). En effet, l'accès au système de positionnement GPS est limité dans les milieux extérieurs ouverts, à cause de sa capacité limitée à transmettre ses signaux. En effet, il a besoin de quatre satellites au minimum pour déterminer les coordonnés exactes d'un point dans l'espace. Il est, par conséquent, inaccessible dans certains milieux tels que les bâtiments, les mines souterraines et même dans des milieux extérieurs protégés. En outre, la centrale inertielle IMU (Inertial Measurement Unit) est aussi un instrument de navigation très courant. Or, il n'est pas assez fiable non plus à cause de son accumulation d'erreurs.

En revanche, La recherche sur les véhicules aériens et l'avancement technologique des systèmes embarqués tels que les micro-ordinateurs et les capteurs embarqués, ont notamment amélioré les performances requises de tels systèmes. En effet, les véhicules aériens qui sont destinés aux environnements où le GPS n'est pas accessible peuvent compter sur des systèmes de localisation autre que le GPS en utilisant, par exemple, des capteurs embarqués afin de faire la localisation et la cartographie simultanées (SLAM). Pour adresser ce problème, la fusion d'un capteur visuel et un capteur laser (Optical Flow + Lidar) ont été proposées par [2] et [3] comme une alternative très utile des méthodes de positionnement traditionnelles grâce à leur indépendance d'une assistance externe et le fait qu'elles sont moins sensibles aux interférences extérieures.

Dans l'article [4], une recherche approfondie a été menée par J. Qi, N. Yu et X. Lu sur la technologie de localisation des micros véhicules aériens sans pilote basée sur la fusion d'un capteur à flux optique et d'un IMU.

En outre, L'estimation de la position est essentielle pour plusieurs taches de navigation, y compris la localisation, la cartographie et le contrôle. Les techniques utilisées dépendent principalement des capteurs embarqués, qui doivent être sélectionné attentivement en fonction des limitations de l'environnement et des exigences de l'application. L'une des techniques les plus récentes et les plus utilisées est le SLAM visuelle (Visual SLAM).

Néanmoins, la plupart des travaux dans ce sens sont limités à des petits espaces de travail ayant déjà des exigences sur les conditions de luminosité. De plus, le temps de calcul est très élevé pour une dynamique rapide d'un véhicule aérien. Par contre, les lasers et les capteurs à large gamme ont été utilisés dans des milieux de travail assez large grâce à leurs larges gammes ainsi que leur détection rapide des distances.
Dans l'article [5], D. Kominiak , S. S. Mansouri , C. Kanellakis et G. Nikolakopoulos ont proposé une plateforme d'un micro véhicule aérien équipé d'un lidar 2D et un capteur à flux optique pour une navigation autonome au sein des tunnels sombres.

Ce travail se focalise, dans un premier temps, sur la fusion des données d'un capteur lidar 2D et un IMU dans le but de suivre, avec robustesse, l'état de l'UAV en utilisant un filtre de Kalman étendu (EKF).

L'avantage du filtre de Kalman étendu par rapport aux autres algorithmes de filtrage c'est qu'en fusionnant toutes les mesures disponibles, il rejette les mesures avec des erreurs importantes. Cela rend le véhicule moins sensible aux perturbations qui affectent l'un des capteurs. Un EKF permet également d'utiliser des mesures à partir des capteurs optionnels tels que le capteur à flux optique ou le capteur à laser afin d'assister la navigation.

Dans un second temps, on implémente un algorithme d'évitement d'obstacles basé sur la théorie des champs de potentiels artificiels [6], [7], en exploitant les distances des obstacles issues du lidar.

La méthode du champ de potentiel artificiel (APF) est une méthode de génération de trajectoire en ligne [8]. Elle permet aux robots de naviguer dans un environnement inconnu en évitant les obstacles mobiles et immobiles. Elle est très répandue dans la robotique mobile en raison de sa simplicité mathématique et algorithmique [9], [10]. Cependant, cette approche est connue aussi par une contrainte majeure ; qui concerne la possibilité de rester figé dans un minimum local. Il existe plusieurs approches qui essaient de résoudre ce problème [1], [11].
Le projet de P. H. Moreno , A. Paranjape et T. Mylvaganam [1] vise à construire un quadrirotor autonome capable de naviguer dans des milieux inconnus. Ils font aussi une étude comparative entre quelques approches de planification de trajectoire. Ainsi, ils implémentent une planification par les champs potentiels qui s'exécute au long d'un algorithme qui résout le problème du minimum local.

L'objectif principal de ce travail consiste à proposer une plateforme d'un drone UAV, comme montré dans la figure 1.1, pour une navigation autonome dans les milieux intérieurs. La plateforme est équipée par un 2D Lidar, un altimètre, un contrôleur de vol Pixhawk4 et un micro-ordinateur NVIDIA Jetson Nano, tandis que, l'architecture de software a été développé en correspondance avec les capteurs utilisés dans le but de réaliser une navigation autonome.

Finalement, ce projet explique le fonctionnement du matériels utilisés et les fonctionnalités software requises pour permettre une poursuite de développement éventuelle par rapport au prototype proposé.


FIG. 1.1 : Plateforme du quadri-rotor equipée par un PIXHAWK4, une carte NVIDIA, un capteur sonore et un lidar 2D

Chapitre 2

Commande et Observation

2.1 Modélisation du quadri-rotor

Dans cette section, on présente la dynamique d'un drone quadri-rotor par un modèle d'état. Pour ce faire, on aura besoin d'une représentation par repères ; un repère terrestre (World frame) noté R_W qui s'agit d'un repère fixe ainsi qu'un repère mobile (body frame) noté R_B qui soit attaché au centre de masse du quadri-rotor.

Comme son nom l'indique, il comporte quatre rotors (moteur + hélice), chacun développe une force verticale de portance. Ainsi, en raison de l'emplacement de ces rotors par rapport au centre du drone (voir 2.1), un moment perpendiculaire au plan de rotation des hélices sera produit.

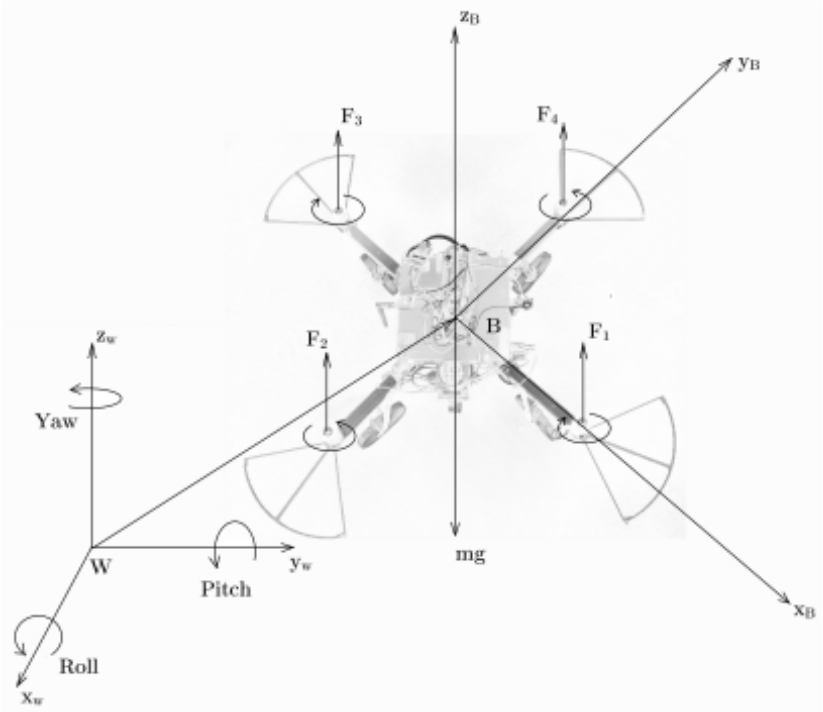


FIG. 2.1 : Repérage du système et les forces/moments qui s'appliquent sur le quadri-rotor

Remarque : La convention Z-X-Y d'angles d'Euler est utilisé pour modéliser la rotation du quadri-rotor dans le repère terrestre W.

2.1.1 Repérage du quadri-rotor dans l'espace

Repère terrestre (World frame)

Le repère terrestre (ou le repère inertiel) est noté $W(0_W, x_W, y_W, z_W)$, d'origine O_W est d'axes x_W ,y_W,z_W lié à la terre, supposée immobile, ou l'axe Z_W est dirigé vers le haut.

Repère lié au corps du quadri-rotor (Body frame)

Il est noté $B(o_B, x_B, y_B, z_B)$, d'origine o_B, qui coïncide avec le centre de gravité du quadri-rotor, et d'axes x_B,y_B et z_B figure 2.1, sachant que la translation suivant l'axe x_B représente un déplacement du drone vers l'avant, la translation suivant l'axe y_B représente un déplacement vers la droite ainsi que l'axe z_B est perpendiculaire au plan des moteurs.

2.1.2 Matrice de rotation

On définit une matrice de rotation à partir des angles d'Euler qui peuvent être obtenu en effectuant trois rotations successives suivant les axes Z-X-Y afin de représenter la rotation du quadri-rotor dans le repère Body. On considère que les centres o_W et o_B des deux repères sont confondus, cela signifie que le repère B ne fait que des rotations par rapport au repère W. Pour aller du repère World vers le repère Body lié au quadri-rotor, on fait, premièrement, une rotation autour de l'axe z_W par l'angle de lacet ψ, puis une rotation autour de l'axe x du premier repère intermédiaire par l'angle ϕ, et finalement une rotation autour de l'axe y_b par l'angle θ.

Première rotation

La première rotation est une rotation de ψ autour de l'axe z_w, sa matrice de rotation est :

$$R_\psi = \begin{pmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

Deuxième rotation :

La deuxième rotation est une rotation autour de l'axe intermédiaire $x^{'}$ par l'angle ϕ, sa matrice de rotation est :

$$R_\phi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{pmatrix} \tag{2.2}$$

Troisième rotation :

La troisième rotation est une rotation de θ autour de l'axe y_b, sa matrice de rotation est :

$$R_\theta = \begin{pmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{pmatrix} \tag{2.3}$$

Matrice de rotation globale

Nous obtenons enfin la matrice de rotation globale qui nous permet de passer de R_B à R_W en multipliant de manière successive les trois matrices de rotation précédentes :

$$Rot = R_{\psi}.R_{\phi}.R_{\theta} = \begin{pmatrix} c\theta c\psi - s\phi s\psi s\theta & -c\phi s\psi & c\psi s\theta + c\theta s\phi s\psi \\ c\theta s\psi + c\psi s\phi s\theta & c\phi c\psi & s\psi s\theta - c\psi c\theta s\phi \\ -c\phi s\theta & s\phi & c\phi c\theta \end{pmatrix} \tag{2.4}$$

Où : $c\theta$ et $s\theta$ sont respectivement $cos(\theta)$ et $sin(\theta)$.

Les vitesses angulaires

Les vitesse angulaires du quadri-rotor dans le repére Body sont p, q et r. Elles sont exprimées en fonction des angles de roulis, tangage et lacet qui sont données par :

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} c\theta & 0 & -c\phi s\theta \\ 0 & 1 & s\phi \\ s\theta & 0 & c\phi c\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2.5}$$

2.2 Modèl dynamique

2.2.1 Hypothèses simplificatrices

Dans cette étude nous adoptons quelques hypothèses simplificatrices :

- La structure du quadri-rotor est supposée rigide et symétrique.
- La matrice d'inertie I est supposée constante.
- Les forces de portance et de trainée sont supposées proportionnelles au carré de la vitesse angulaire des rotors.
- Le repère lié au corps du quadri-rotor est supposé confondu avec son centre de gravité.

2.2.2 La dynamique de translation

D'après la seconde loi de la dynamique du Newton dans le repère World :

$$m\ddot{r} = \sum F_{ext} \tag{2.6}$$

Où :

- $m \in R^{+}$, est la masse totale du quadri-rotor.
- r = $\begin{bmatrix} x \\ y \\ z \end{bmatrix} \in R^{3.1}$, est le vecteur de position du centre de masse du quadri-rotor dans le repère inertiel ou World.
- $\sum \mathrm{F}_{ext} \in R^{3.1}$, est le vecteur des forces externes totales.

Le poids

C'est la force de gravité qui s'applique sur le drone, elle est donnée par :

$$P = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \tag{2.7}$$

Où : g $\in R^+$, est l'accélération de la pesanteur de la terre.

La force de portance

C'est la force totale générée par la rotation des hélices des quatre moteurs, elle est dirigée vers le haut. C'est-à-dire qu'elle a tendance à faire monter le quadri-rotor. Elle est donnée par :

$$F_p = Rot. \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \tag{2.8}$$

Où : $T = \sum_{i=1}^{4} F_i$ est la force de portance totale des quatre hélices en remplaçant l'expression des forces externes dans l'équation de la dynamique de translation 2.6 nous obtenons après simplification le système d'équations différentielles suivant :

$$m\ddot{r} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R. \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} \tag{2.9}$$

On définit la première entrée du système :

$$u_1 = F_1 + F_2 + F_3 + F_4 \tag{2.10}$$

2.2.3 La dynamique de rotation

En plus des forces, chaque rotor produit un moment perpendiculaire au plan de rotation des hélices, M_i. Le rotor 1 et 3 tourne autour de l'axe $-z_b$ tandis que le rotor 2 et 4 tourne autour $+z_b$, pour que le moment selon z qui serait généré par la rotation des hélices soit nul, M_1 et M_2 agissent dans la direction $+z_b$ tandis que M_2 et M_4 agissent dans la direction $-z_b$. l est la distance entre le centre de gravité du quadri-rotor et l'axe de rotation de l'un des rotors.

D'après la loi de la dynamique d'Euler :

$$\frac{d(I\Omega)}{dt} = \sum \Gamma_{ext} \tag{2.11}$$

Et comme la vitesse angulaire est exprimée dans le repère lié au quadri-rotor, alors :

$$\frac{d(I\Omega)}{dt} = I\dot{\Omega} + \Omega \times I\omega \tag{2.12}$$

Ce qui nous donne :

$$I\dot{\Omega} = -\Omega \times I\Omega + \sum \Gamma_{ext} \tag{2.13}$$

Où : $I \in R^{(3\times3)}$ est la matrice d'inertie du quadri-rotor.

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{2.14}$$

- $\sum \Gamma_{ext} \in R^{3.1}$ est le vecteur des moments produits par les rotors.
- $\Omega = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \in R^{3.1}$ est le vecteur des vitesses de rotations instantanées dans le repère du quadri-rotor.

L'accélération angulaire déterminée par les équations d'Euler est donnée par :

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} l(F_2 - F_4) \\ l(F_3 - F_1) \\ M_1 - M_2 + M_3 + M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.15}$$

Où, $l \in R^+$, est la distance entre le centre de gravité du quadri-rotor et l'axe de rotation.

On définit la deuxième entrée du système $u_2 \in R^{3.1}$

$$u_2 = \begin{bmatrix} l(F_2 - F_4) \\ l(F_3 - F_1) \\ M_1 - M_2 + M_3 + M_4 \end{bmatrix} \tag{2.16}$$

Modèle du moteur

Chaque rotor a une vitesse angulaire ω_i et génère une force verticale F_i due aux interférences aérodynamiques des hélices, elle est donnée par :

$$F_i = K_F.\omega_i^2 \tag{2.17}$$

Les moments produits par la vitesse angulaire du rotor sont donnés par :

$$M_i = K_M\omega_i^2 \tag{2.18}$$

Où :

- $K_F \in R^+$ est le coefficient de portance.
- $K_M \in R^+$ est la constante de trainée.
- $\omega_i \in R^+$ est la vitesse de rotation du moteur i.

2.3 Synthèse de la commande PD

Le quadri-rotor fait l'objet de beaucoup de recherche dans le domaine de la commande. Alors, plusieurs lois de commande ont été proposées. Parmi ces dernières : la commande PID [12] , la commande LQR [13] , le backstepping [14] , la commande par mode glissant [15] .

Nous avons opté pour la commande PD car elle montre des performances très acceptables et robustes en simulations, où nous testons notre contrôleur dans deux scenarios. Le premier scenario est la poursuite d'une ligne et le deuxième est la poursuite d'une trajectoire hélicoïdale.

2.3.1 La structure de contrôle

Le système de contrôle (appelé aussi contrôle bas-niveau) possède deux boucles de régulation, une boucle interne et une autre externe, comme montré dans la figure 2.2. La boucle interne assure la régulation d'attitude (angles d'orientation) tandis que la boucle externe assure la régulation de position. Dans la boucle interne, nous aurons besoin de spécifier l'orientation du quadri-rotor par une matrice de rotation qui contient les trois angles de rotation (angle d'Euler). Nous ferons, ensuite, le retour d'état des angles et des vitesses angulaires actuels. A partir de là, nous voulons calculer la commande u_2. Dans la boucle externe, ou la régulation de la position s'effectue, nous spécifions la position et l'angle de lacet désirées $\phi^d es$ Et nous retournons la position et la vitesse actuelles et à partir de cela, nous calculons la commande u_1.

Dans notre système, la boucle de régulation d'attitude utilise, d'une part, l'IMU pour connaitre les angles de roulis, tangage, et lacet actuels, ce qui se fait à une fréquence autour de 100 Hz. D'autre part, la régulation de la boucle externe utilise l'estimation de position par un algorithme SLAM afin de contrôler la trajectoire en trois dimensions. La fréquence de l'estimation de position est environ 10 Hz.

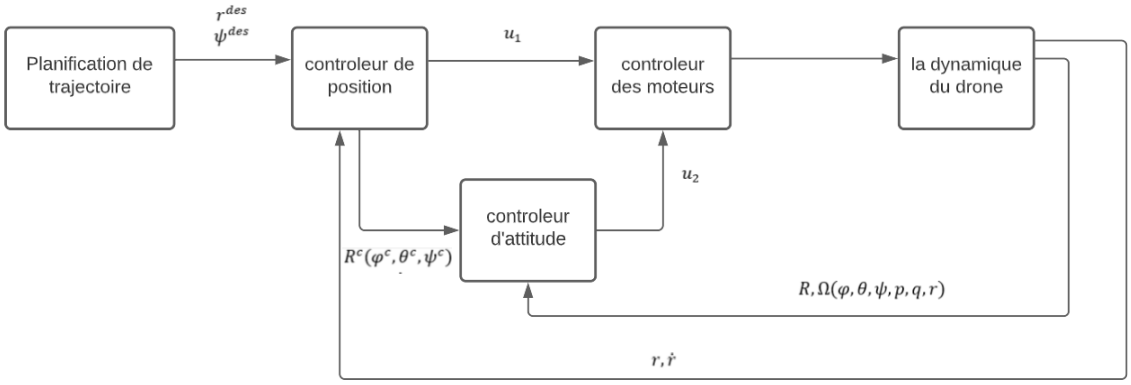


FIG. 2.2 : la structure de contrôle

Notre contrôleur a été obtenue par une linéarisation des équations de mouvement 2.1 et 2.9 et autour de l'état d'équilibre, où : $r = r_0$, $\theta = \phi = 0, \phi = \phi_0, \dot{r} = 0, et \dot{\phi} = \dot{\theta} = \dot{\psi} = 0$, et les angles tangage, roulis, et lacet sont très faibles ($c\phi \approx 1, c\theta \approx 1, s\phi \approx \phi, ets\theta \approx \theta$). A l'équilibre, la force nominale de poussée par les hélices doit satisfaire :

$$F_{i,0} = \frac{mg}{4} \tag{2.19}$$

La vitesse de chaque moteur est donnée par :

$$\omega_{i,0} = \omega_h = \sqrt{\frac{mg}{4K_F}} \tag{2.20}$$

2.3.2 Contrôle des angles d'orientation

Dans cette section nous présentons un contrôleur des angles d'orientation (attitude), en vol stationnaire la position et l'orientation sont fixes donc toutes les vitesses sont nulles, de plus, les angles de roulis et de tangage sont égales à zéro, par conséquent, nous allons linéariser la dynamique autour de cette configuration d'équilibre. D'après l'équation (2.15) nous pouvons écrire :

$$\begin{cases} I_{xx}\dot{p} = lK_F(\omega_2^2 - \omega_4^2) - qr(I_{zz} - I_{yy}) \\ I_{yy}\dot{q} = lK_F(\omega_3^2 - \omega_1^2) - pr(I_{xx} - I_{zz}) \\ I_{zz}\dot{r} = (\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \\ \end{cases} \tag{2.21}$$

A noter que le produit du moment d'inertie est très faible (car les axes autour desquels il est calculé sont proches aux axes principaux du quadri-rotor), et $I_{xx} \approx I_{yy}$ due à la symetrie.

On peut aussi supposer que la vitesse angulaire r suivant z_B est faible donc le terme dans l'équation (2.21) qui possède un produit avec r est très faible par rapport à l'autre terme. A l'équilibre $\dot{\phi} \approx p, \dot{\theta} \approx q$, et $\dot{\psi} \approx r$. Pour cette raison on peut utiliser un simple proportionnel dérivé.
Le vecteur des vitesses de rotation des rotors peut être écrit sous forme d'une combinaison linéaire à quatre termes :

$$\begin{bmatrix} \omega_1^d es \\ \omega_2^d es \\ \omega_3^d es \\ \omega_4^d es \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 & 1 \\ 1 & 1 & 0 & -1 \\ 1 & 0 & 1 & 1 \\ 1 & -1 & 0 & -1 \end{bmatrix} \begin{bmatrix} \omega_h + \Delta w_F \\ \Delta\omega_\phi \\ \Delta\omega_\theta \\ \Delta\omega_\psi \end{bmatrix} \tag{2.22}$$

Où la vitesse nominale du rotor nécessaire pour l'équilibre est ω_h et les variations autour du point d'équilibre sont $\Delta\omega_F$, $\Delta\omega_\phi$, $\Delta\omega_\theta$ et $\Delta\omega_\psi$. $\Delta\omega_F$ est la vitesse de rotation de chaque moteur à l'équilibre selon la direction z_B, tandis que $\Delta\omega_\phi$,$\Delta\omega_\theta$ et $\Delta\omega_\psi$ produise les moments qui affectent les angles de roulis, tangage, et lacet. Cette approche a été utilisé par [16].

Maintenant, on linéarise l'équation (2.21) autour du point d'équilibre et nous écrivons les accélerations angulaires désirées en fonction des entrées du système :

$$\begin{cases} \dot{p}^{des} = \frac{4K_F l w_h}{I_{xx}} \Delta\omega_\phi \\ \dot{q}^{des} = \frac{4K_F l w_h}{I_{yy}} \Delta\omega_\theta \\ \dot{r}^{des} = \frac{8K_M l w_h}{I_{zz}} \Delta\omega_\phi \end{cases} \tag{2.23}$$

Au plus près de l'état d'équilibre, $\dot{\phi} \approx p, \dot{\theta} \approx q, et \dot{\psi} \approx r$, nous utilisons la loi de commande proportionnel-dérivé qui prend la forme suivante

$$\begin{bmatrix} \Delta\omega_\phi \\ \Delta\omega_\theta \\ \Delta\omega_\psi \end{bmatrix} = \begin{bmatrix} K_{p,\phi}(\phi^{des} - \phi) + K_{d,\phi}(p^{des} - p) \\ K_{p,\theta}(\theta^{des} - \theta) + K_{d,\phi}(q^{des} - q) \\ K_{p,\psi}(\psi^{des} - \psi) + K_{d,\psi}(r^{des} - r) \end{bmatrix} \tag{2.24}$$

Afin de trouver les vitesses de rotation souhaitées de chaque rotor on remplace 2.24 dans (2.22).

2.3.3 Contrôle de position

Dans cette partie on va présenter une méthode de contrôle de la position x-y qui utilise les angles de roulis et tangage comme une entrée au niveau de la boucle de régulation d'attitude, donc il s'agit bien d'une régulation en cascade. Premièrement on développe un contrôleur autour du point d'équilibre, ce dernier est utilisé pour suivre une trajectoire ou bien pour maintenir une position désirée x-y-z. Deuxièment on synthétise une loi de commande qui assure la poursuite d'une trajectoire en trois dimensions.

On définit le vecteur $r^{des}(t)$ et $\psi^{des}(t)$ qui représente la trajectoire et le lacet de référence qu'on veut suivre. Où $r^{des(t)} = \begin{bmatrix} x^{des}(t) & y^{des}(t) & z^{des}(t) & \psi^{des}(t) \end{bmatrix}$, pour que ça soit une trajectoire réaliste, nous voulons que ce vecteur soit différenciable deux fois. De plus, nous voulons que le vecteur d'erreur converge de façons exponentielles vers zéro.
On définit l'erreur de position par :

$$e_p = r^{des}(t) - r(t) \tag{2.25}$$

Où :

- $r \in R^{4.1}$ est la position actuelle.
- $r^des \in R^{4.1}$ est la position désirée.

Et on définit aussi l'erreur de vitesse par :

$$e_v = \dot{r}^{des} - \dot{r} \tag{2.26}$$

Où :

- $\dot{r} \in R^{4.1}$, est la vitesse actuelle.
- $\dot{r}^{des} \in R^{4.1}$, est la vitesse désirée.

Nous voulons que la dynamique de l'erreur soit une solution de l'équation différentielle du controleur proportionnel-dérivé suivant :

$$(\ddot{r}^{des} - \ddot{r}_c) + K_d e_v + K_p e_p = 0 \tag{2.27}$$

Cette équation nous permet alors de calculer les accélérations de commandes $\ddot{r}_c$
Voici un schéma représentatif plus détaillé de la structure de contrôle PD :

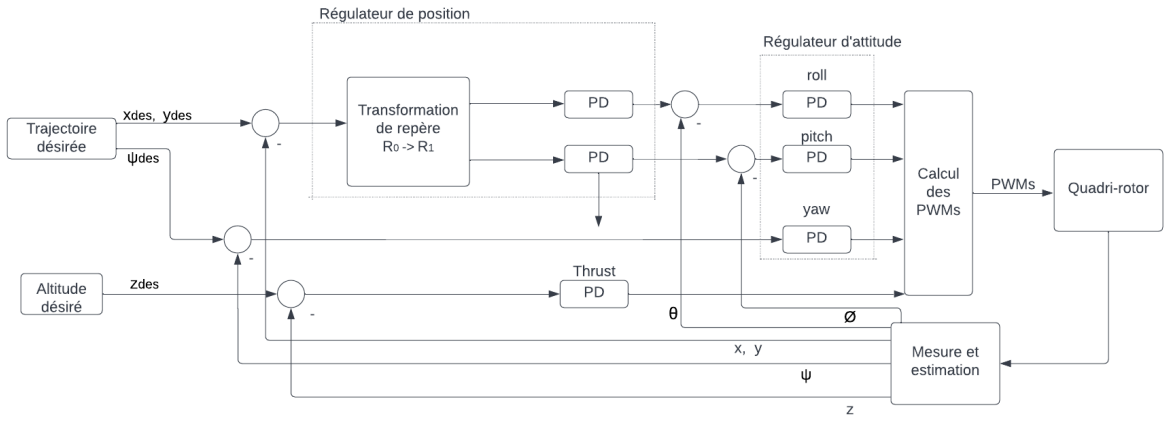


Fig. 2.3 : Structure de contrôle PD plus détaillée

2.3.3.1 Contrôle à l'équilibre

A l'équilibre $\psi_T(t) = \psi_0$, l'accélération de commande est calculé a partir de l'équation différentielle du régulateur proportionnel-dérivé (2.27), où $\dot{r}^{des}$ et $\ddot{r}^{des}$ sont nulles.

Alors, on linéarise l'équation (2.9) pour trouver la relation entre l'accélération desireé et les angles de roulis et tangage (roll and pitch).

$$\begin{cases} \ddot{r_1}^{des} = g(\theta^{des} cos\psi^{des} + \phi^{des} sin\psi^{des}) \\ \ddot{r_2}^{des} = g(\theta^{des} sin\psi^{des} - \phi^{des} cos\psi^{des}) \\ \ddot{r_3}^{des} = \frac{8K_F w_h}{m} \Delta w_F \end{cases} \tag{2.28}$$

Cette relation (2.28) sera inversée pour calculer les angles de roulis et tangage desirée.

$$\begin{cases} \ddot{\phi}^{des} = \frac{1}{g}(\ddot{r_1}^{des} sin\psi^{des} + \ddot{r_2}^{des} cos\psi^{des}) \\ \ddot{\theta}^{des} = \frac{1}{g}(\ddot{r_1}^{des} cos\psi^{des} + \ddot{r_2}^{des} sin\psi^{des}) \\ \Delta w_F = \frac{m}{8K_F w_h} \ddot{r_3}^{des} \end{cases} \tag{2.29}$$

2.3.3.2 Contrôle d'une trajectoire 3-D

Un contrôleur de trajectoire tridimensionnel est la poursuite d'une trajectoire en 3-D avec des accélérations faibles, de sorte que les hypothèses soient validées et que le drone est autour du point de stationnarité. On s'inspire de l'approche utilisée dans [17] sauf que nous utilisons cette approche en 3-D au lieu de 2-D. Cette méthode consiste à calculer le point le plus proche de la trajectoire r^{des} à la position actuelle r. On définit le vecteur tangent unitaire de la trajectoire associée à ce point $\hat{t}$, le vecteur normal $\hat{n}$, ainsi que le vecteur binormal $\hat{b}$.. On définie aussi l'erreur de position et de vitesse par :

$$e_p = ((r^{des} - r).\hat{n})\hat{n} + ((r^{des} - r).\hat{b})\hat{b} \tag{2.30}$$

Et

$$e_v = \dot{r}^{des} - \dot{r} \tag{2.31}$$

Noter que nous ignorons l'erreur de position selon la direction tangente en considérant que l'erreur de position selon la normale $\hat{n}$, et la binormale $\hat{b}$.

On applique la même stratégie suivie par le contrôle à l'équilibre pour trouver la commande d'accélération $\ddot{r}_c$. A partir de l'équation du proportionnel-dérivé on aura :

$$\ddot{r}_c = \ddot{r}^{des} + K_d e_v + K_p e_p \tag{2.32}$$

2.3.4 Resultats de simulation d'un quadri-rotor sur Matlab

Dans cette partie, nous allons tester notre contrôleur avec deux trajectoires, la première est une trajectoire linéaire et la seconde est une trajectoire hélicoïdale.
L'objectif consiste à faire en sorte que le contrôleur suive les deux trajectoires et que l'erreur entre la position désirée et la position actuelle converge d'une façon exponentielle

vers zéro. Les paramètres physiques du quadri-rotor et les coefficients du régulateur sont disponibles en annexe, tableau 7.1.

Dans le premier scenario, nous avons donnée comme référence un point de coordonnées $\{1, 1, 1\}$. On voit les résultats de la simulation sur les figures suivantes.

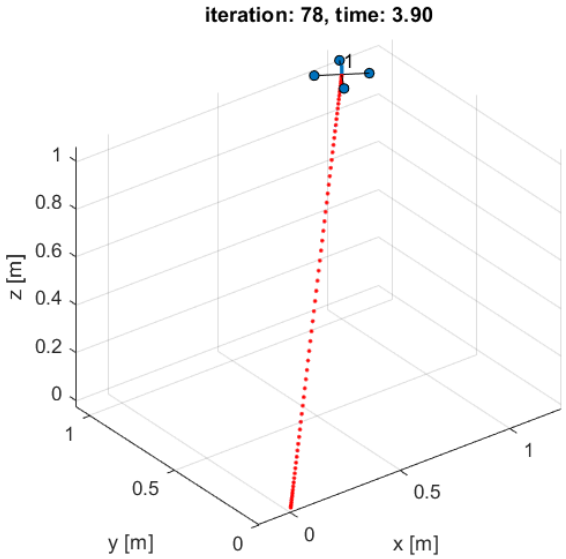


FIG. 2.4 : Evolution en 3D du quadri-rotor dans l'espace

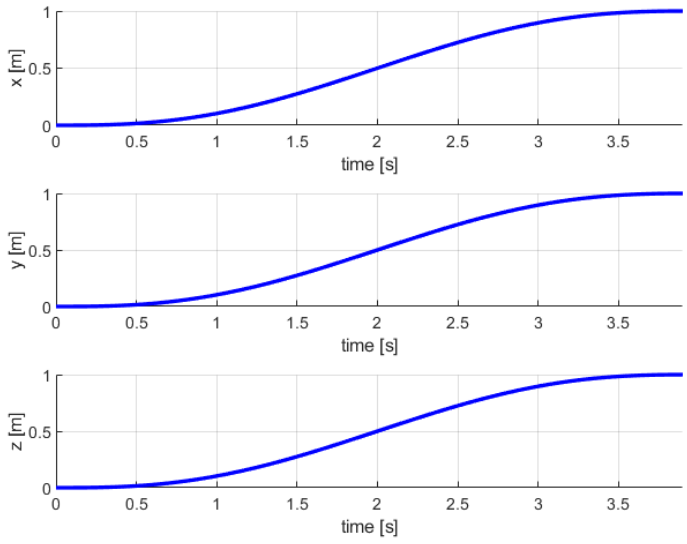


FIG. 2.5 : Evolution de la position du quadri-rotor en fonction du temps

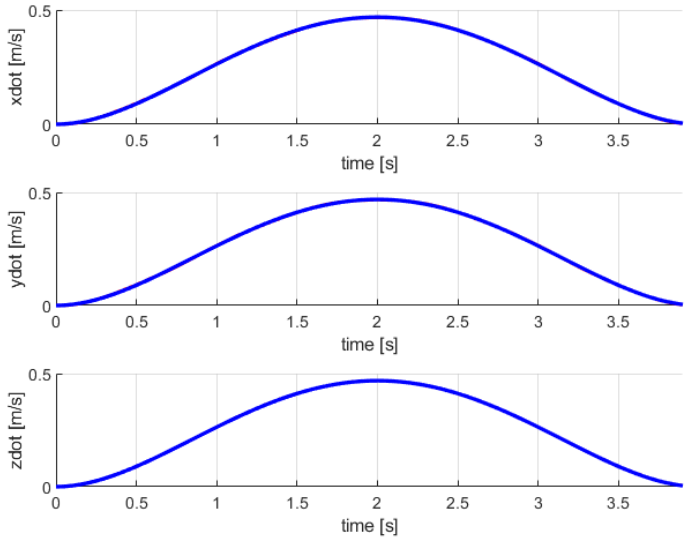


FIG. 2.6 : Evolution de la vitesse du quadri-rotor en fonction du temps

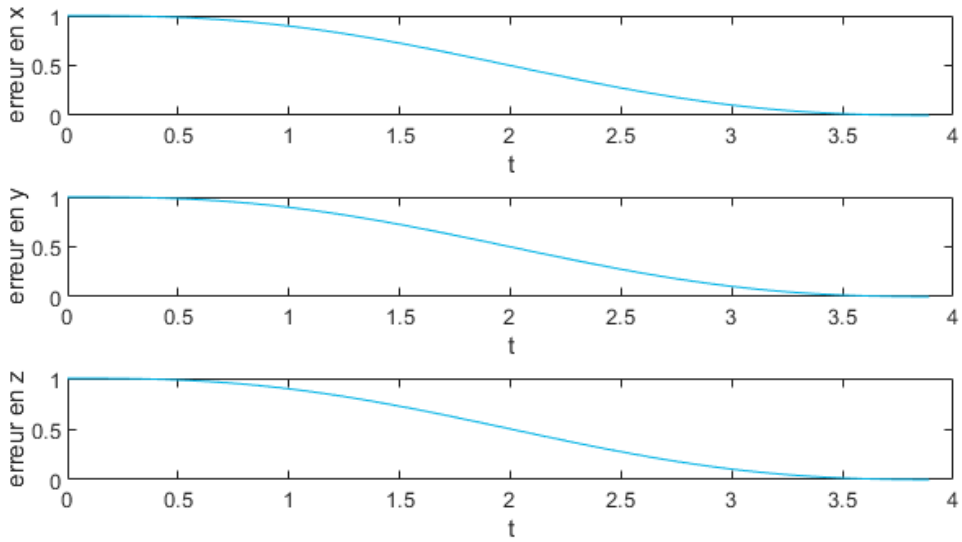


FIG. 2.7 : Evolution de l'erreur de position du quadri-rotor en fonction du temps

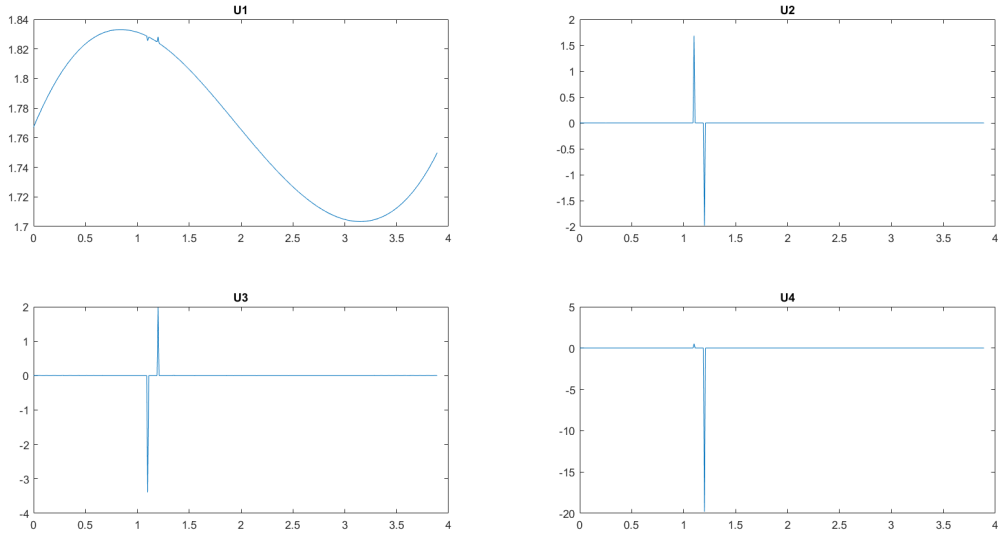


FIG. 2.8 : Evolution des signaux de commande en fonction du temps

Interprétation :

A l'état initial, le quadri-rotor était sur terre. Afin qu'il surmonte son poids, le signal u_1 de la force de poussée s'initialise à 1.76, et puis, il augmente au fur et à mesure. A l'instant 1s le quadri-rotor commence à décoller à une vitesse d'environ 0.25 m/s, pour ce faire, la force de poussée s'affaiblit jusqu'à ce que le quadri-rotor atteint son altitude désiré à l'instant 3.5s où la force de poussée augmente encore une fois pour assurer la force nécessaire permettant au quadri-rotor de maintenir son altitude à 1m.

Les commandes u_2, u_3 représentent les moments selon les axes X, Y respectivement. Comme montré dans la figure 2.9, pour que le quadri-rotor fasse une translation suivant Y, une impulsion positive du moment selon X a été menée à l'instant 1.2s, ce qui provoque une petite rotation selon l'axe X en faisant de sorte que la force de poussée ait une composante selon l'axe Y, ce qui implique une translation selon Y.
Ensuite, cette impulsion positive du moment est suivie par une impulsion négative qui s'explique par le fait que ; lorsque le quadri-rotor s'approche à la position Y désirée, il doit annuler la rotation due à la première impulsion afin de se fixer à la position Y désirée.

La translation suivant X se fait de la même manière.

La commande u_4 représente le moment exercé sur le quadri-rotor selon l'axe Z. Elle présente une impulsion à l'instant 1.2s afin de suivre la consigne de l'angle de lacet selon l'axe Z.

On présente ci-dessous les étapes de translation d'un quadri-rotor vers un point désiré :

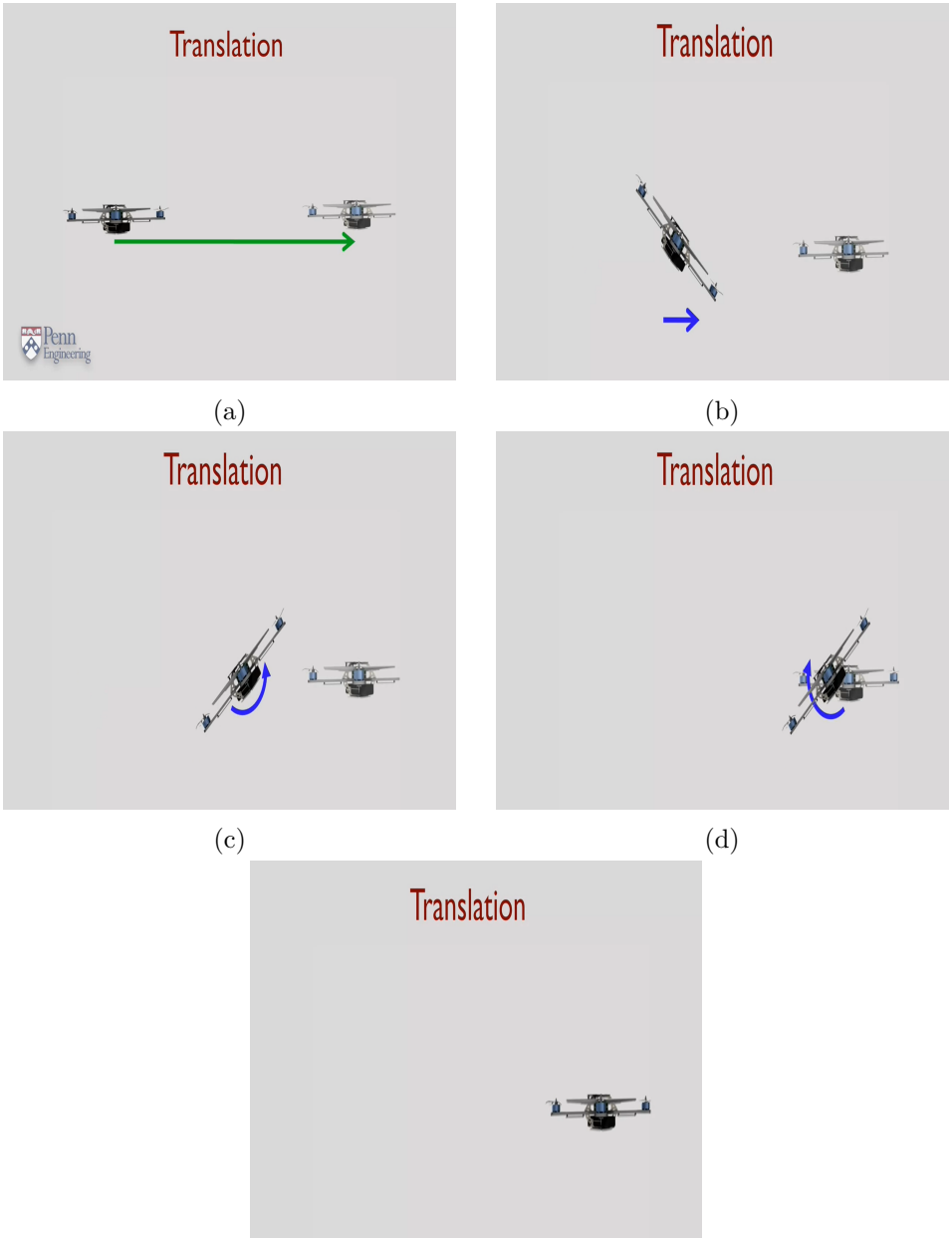


FIG. 2.9 : Etapes de translation d'un quadri-rotor

Pour le deuxième scenario, nous avons imposé comme référence une trajectoire hélicoïdale de rayon $r = 5m$ et d'altitude $z = 1m$. Les résultats de simulation sont montrès dans les figures suivantes.
On constate, ainsi, que le quadri-rotor suit bien la trajectoire désirée.

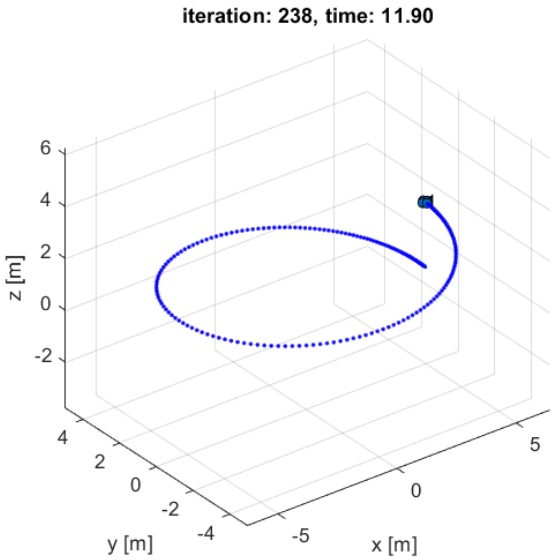


Fig. 2.10 : Evolution en 3D du quadri-rotor en fonction du temps

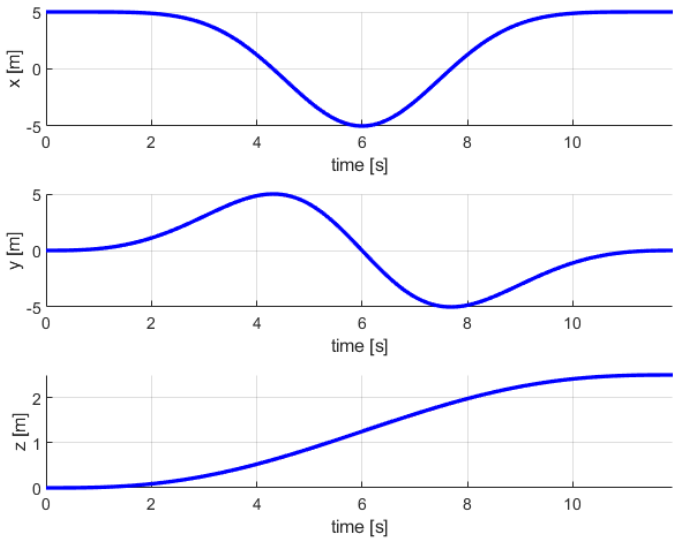


Fig. 2.11 : Evolution de la position du quadri-rotor en fonction du temps

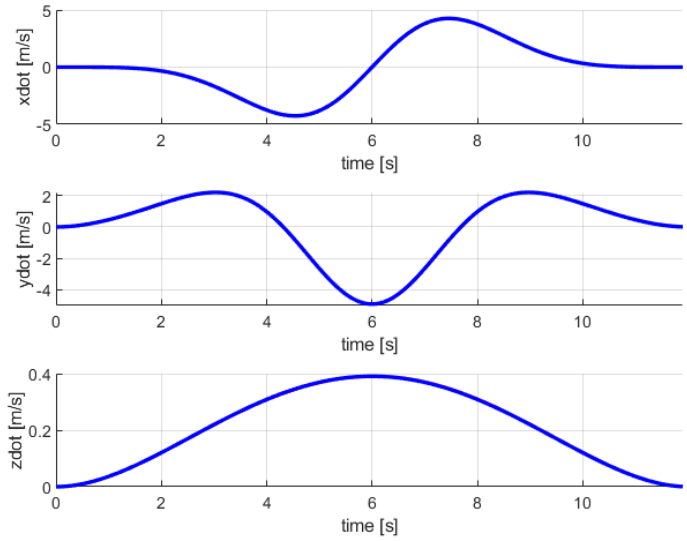


FIG. 2.12 : Evolution de la vitesse du quadri-rotor en fonction du temps

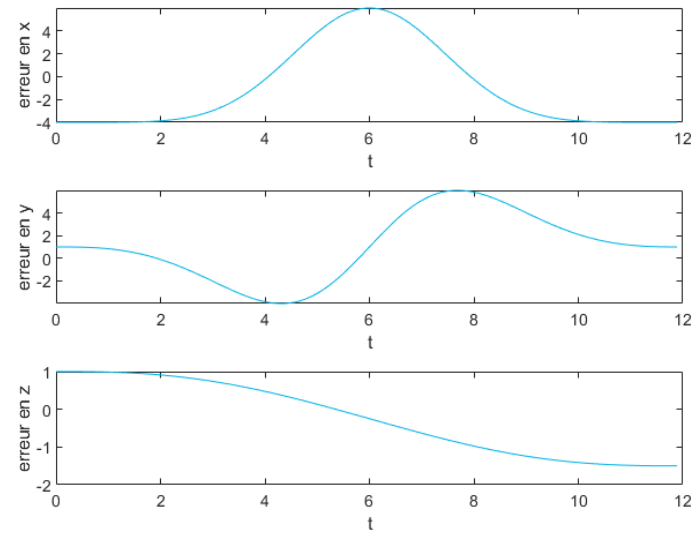


FIG. 2.13 : Evolution de l'erreur de position du quadri-rotor en fonction du temps

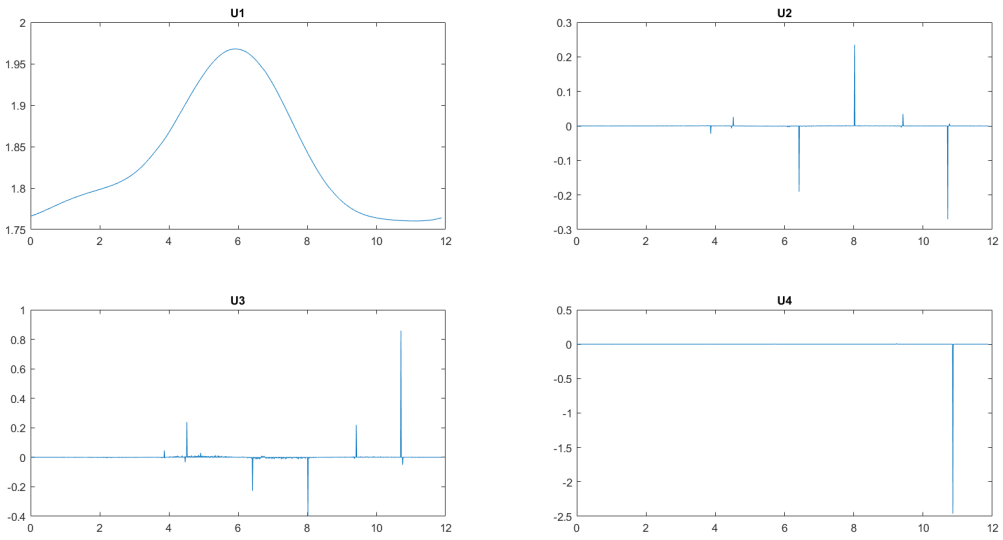


FIG. 2.14 : Evolution des signaux de commande en fonction du temps

2.3.5 Résultats de simulation d'un quadri-rotor basé sur Pixhawk4

Dans cette partie, nous allons présenter les réponses d'un quadri-rotor basé sur un Pixhawk4 qui fait appel à un régulateur PD pour assurer la poursuite des consignes qu'il reçoit. L'environnement de simulation est bien détaillé dans la section 5.3

La figure 3.9 représente la position x-y du drone suite aux consignes (1,0,2.5) et (0,1,2.5) :

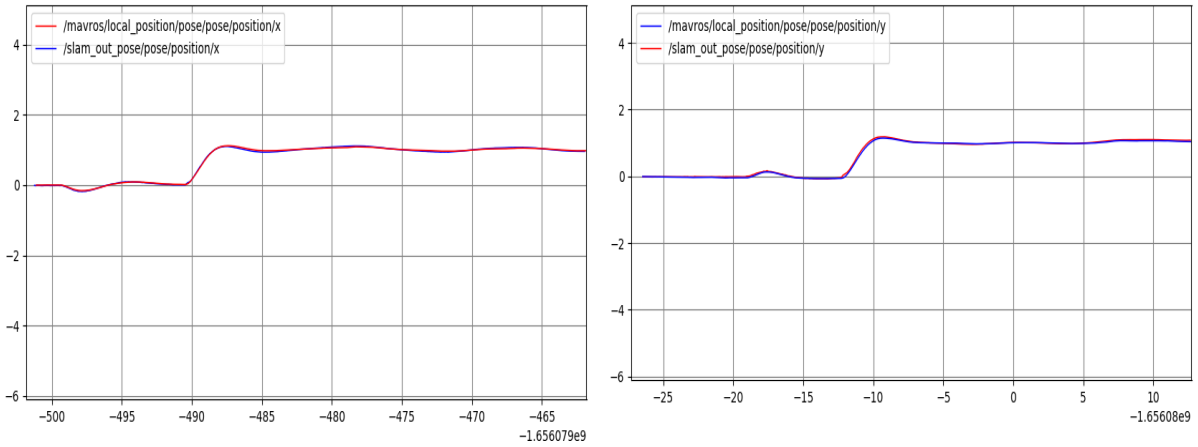


FIG. 2.15 : A gauche : la réponse du drone de simulation à la consigne de position (1,0,2.5) A droite : la réponse du drone de simulation à la consigne de position (0,1,2.5)

La figure 2.16 représente les orientations selon x et y (roll et pitch) mesurées par l'IMU, comme une réponse de la consigne de position (0,2,2.5) :

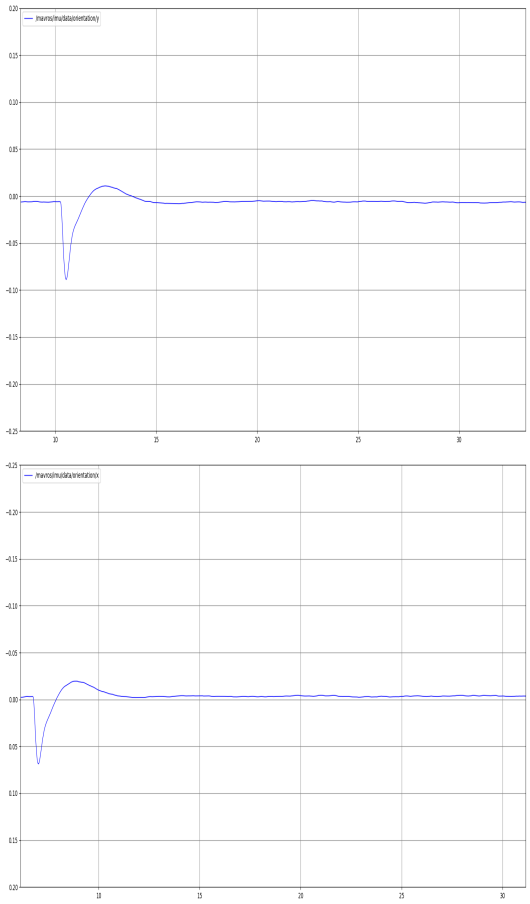


FIG. 2.16 : En haut : la réponse du drone à la consigne de position (2,0,2.5) En bas : la réponse du drone à la consigne de position (0,2,2.5)

La figure 2.17 représente l'orientation selon z (yaw) mesurées par l'IMU, comme une réponse d'une consigne d'orientation $(0,0,-\frac{\pi}{2})$:

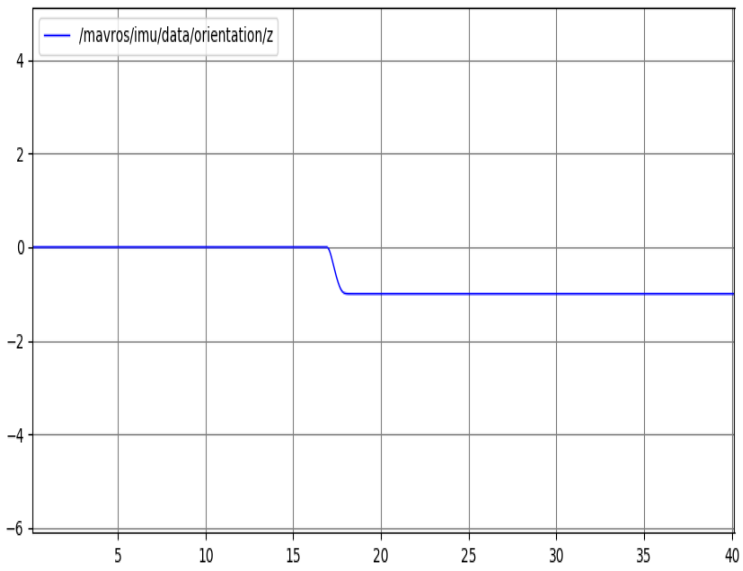


FIG. 2.17 : La réponse du drone à la consigne d'orientation $(0,0,-\frac{\pi}{2})$

La figure 2.18 représente les vitesse angulaire selon x et y mesurées par l'IMU, comme une réponse de la consigne position (2,2,2.5) :

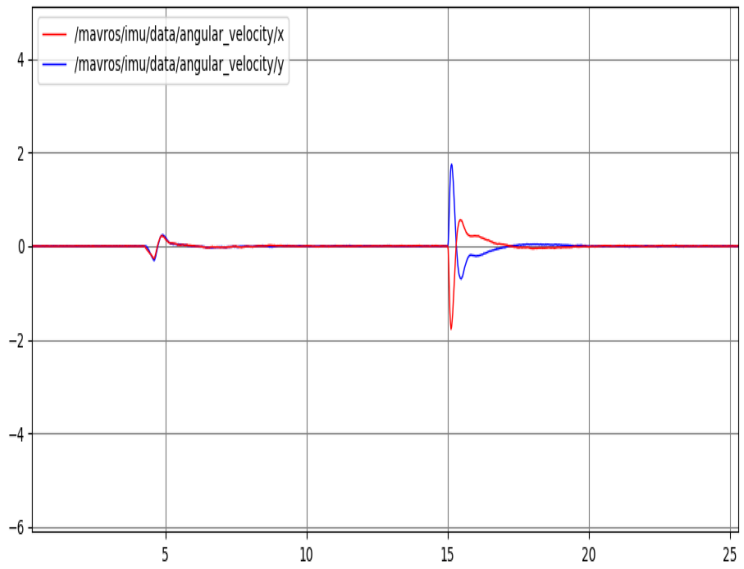


FIG. 2.18 : La réponse du drone à la consigne de position (2,2,2.5)

2.3.6 Conclusion

Pour les différents scénarios, le régulateur PD assure bien la poursuite de la trajctoire désirée avec de bonnes pérformances en termes de stabilité et du temps de réponse.

2.4 Estimation d'état par un filtre de Kalman étendu

Cette section a pour but d'expliquer un observateur d'état non linéaire d'un quadri-rotor. Il s'agit d'un filtre de Kalman étendu qui fait la fusion des données des capteurs (IMU, 2D LIDAR, ALTIMETRE). Quoique, le caractère asynchrone des capteurs (i.e. Chaque capteur a sa propre fréquence de fusion et de mesure), provoque parfois une perte des données utiles et des retards de communication. De plus, les mesures des capteurs sont bruitées, ce qui donne une estimation moins fiable. Afin d'avoir une meilleure estimation d'état, une telle technique de filtrage est nécessaire.

2.4.1 Modèle d'observation du quadri-rotor par un filtre de Kalman étendu

Les équations 2.9 et 2.15 sont utilisées pour exprimer le système sous forme d'une représentation d'état de la forme $\dot{X} = f(X, U)$. avec X est le vecteur d'état, et U est le vecteur d'entrée à injecter au modèle du quadri-rotor :

$$X = \begin{bmatrix} x & y & z & \phi & \theta & \psi & \dot{x} & \dot{y} & \dot{z} & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T \tag{2.33}$$

$$U = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^T \tag{2.34}$$

Où les entrées sont définis par :

$$\begin{cases} u_1 = F_1 + F_2 + F_3 + F_4 \\ u_2 = l(F_2 - F_4) \\ u_2 = l(F_3 - F_1) \\ u_4 = M_1 - M_2 + M_3 - M_4 \end{cases} \tag{2.35}$$

D'après 2.9, 2.21, 2.33, et 2.34 nous obtient après simplification :

$$\dot{X} = f(X, U) = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ u_x \frac{u_1}{m} \\ u_y \frac{u_1}{m} \\ cos\theta cos\phi \frac{u_1}{m} - g \\ a_1 \dot{\theta}\dot{\psi} + b_1 u_2 \\ a_2 \dot{\phi}\dot{\psi} + b_2 u_3 \\ b_3 u_4 \end{pmatrix} \tag{2.36}$$

Où :

$$\begin{cases} a1 = \frac{I_{yy}-I_{zz}}{I_{xx}} \\ a2 = \frac{I_{zz}-I_{xx}}{I_{yy}} \\ b_1 = \frac{l}{I_{xx}} \\ b_2 = \frac{l}{I_{yy}} \\ b_3 = \frac{l}{I_{zz}} \\ u_x = cos\theta sin\phi sin\psi + sin\theta cos\psi \\ u_y = -cos\theta sin\phi cos\psi + sin\theta sin\psi \end{cases}$$

(2.37)

On simplifie l'équation (2.36) tout en négligeant les termes de $\ddot{\phi}$, $\ddot{\theta}$ et $(a_1\dot{\theta}\dot{\psi}, a_2\dot{\phi}\dot{\psi})$ car ils sont supposés très faibles. La représentation d'état utilisée pour l'estimation est formulée comme suit :

$$\begin{cases} x_k = f(x_{k-1}, u_{k-1}) + w_{k-1} \\ z_k = h(x_k) + v_k \end{cases} \tag{2.38}$$

Où :

- x, est le vecteur d'état a l'instatnt k.
- z, est le vecteur des signaux mesurés a l'instatnt.
- u, est le vecteur d'entré du système
- w et v, sont les bruit de processus et de mesure respectivement. Ces dernier sont considérés des bruits blancs gaussiens de moyenne nulle et de covariance Q et R.
- f et h sont des fonctions non linéaires du système.

Soit A et B les matrices jacobiennes définies comme suit : $A = \frac{\partial f}{\partial X}$ Et $B = \frac{\partial f}{\partial U}$. U est le vecteur d'entrée du système, il est lié aux vitesses des moteurs. A chaque pas d'échantillonnage, la linéarisation est effectuée localement et cela n'est valable qu'avec une erreur de linéarisation assez faible. En outre, admettant que la fréquence des capteurs est très élevée, les matrices jacobiennes de A et B seront calculées et utilisées dans les algorithmes de prédiction et de correction 2.4.2. La matrice C des mesures est reliée aux mesures issues des capteurs disponibles. Alors le système pourra s'écrire sous la forme :

$$\begin{cases} \dot{x_k} = Ax_{k-1} + Bu_{k-1} + w_{k-1} \\ z_k = Cx_k + v_k \end{cases} \tag{2.39}$$

Où :

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.40}$$

Dans un premier temps on va examiner l'observabilité du système. Cette dernière est calculé à partir de la matrice de mesure C et la matrice A, en considérant qu'il y a douze états donnés par :

$$\xi = \begin{bmatrix} CA \\ CA^2 \\ . \\ . \\ . \\ . \\ CA^{11} \end{bmatrix} \tag{2.41}$$

L'observabilité du système est calculée dans la position d'équilibre : $A = \frac{\partial f(X,U)}{\partial X}$ et $B = \frac{\partial f(X,U)}{\partial U}$. La position d'équilibre est la position de stationnarité. Au niveau de ce point, toutes les vitesses linéaires et angulaires sont nulles. De plus, les angles de tangage, de roulis et de lacet (roll, pitch and yaw) sont aussi nuls.
Soit le vecteur d'entrée $U_e = \begin{bmatrix} U_{1e} & U_{2e} & U_{3e} & U_{4e} \end{bmatrix}^T$, Ou , $U_{1e} = mg$ et $U_{1e} = U_{2e} = U_{3e} = 0$. En calculant le rang de la matrice d'observabilité et on le trouve 12, d'autre part, le nombre total du vecteur d'états est aussi 12. Donc, le système est observable.

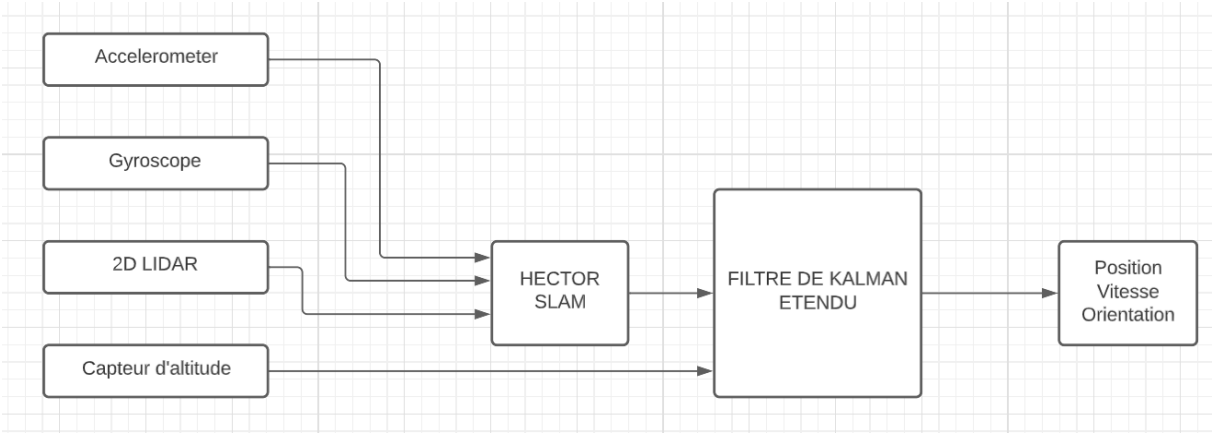


Fig. 2.19 : Schéma représentatif du filtre EKF

2.4.2 L'algorithme du filtre

L'algorithme d'EKF se déroule en deux étapes, la première étape sert à la prédiction et la deuxième à la correction.

2.4.2.1 la phase de prédiction

Cette étape sert à calculer l'estimation d'état a priori et la covariance d'erreur P. Elle est considéré comme une mesure de l'incertitude au niveau l'état estimé, P_K^- provient du bruit de processus et de la propagation de l'incertain.

$$\begin{cases} \hat{x}_{k|k-1}^- = f(\hat{x}_{k-1|k-1}, u_{k-1}) & \text{prédiction du vecteur d'état a priori} \\ \hat{P}_{k|k-1}^- = A_{K-1} P_{k-1|k-1} A_{K-1}^T + Q_{k-1} & \text{prédiction de la covariance} \end{cases} \tag{2.42}$$

Cependant, L'utilisation d'un modèle d'état théorique qui modélise le quadri-rotor afin de prédire l'état du système à ce niveau-là n'est pas pratique, en regard de l'incertitude gênante pouvant se produire lorsqu'on bascule de la téorie vers la pratique. De plus, certains modèles théoriques nécessitent des calculateurs puissants pour les implémenter en pratique.

Pour cette raison, l'algorithme EKF de PX4 utilise le modèle de l'IMU comme prédicteur d'état ; ce modèle représente un simple intégrateur des mesures acquis de l'IMU. La prédiction d'état se fait comme suit :

1. Les vitesses angulaires mesurées par le gyroscope sont intégrées pour calculer la position angulaire.
2. Les accélérations venues de l'accéléromètre sont converties à l'aide de la position angulaire du corps X, Y, Z par rapport aux axes terrestres (Nord, Est et Bas) qui sont connus par la mesure de magnetomètre.
3. Les accélérations sont intégrées pour calculer la vitesse.
4. La vitesse est intégrée, ainsi, pour calculer la position.
5. Afin de prédire les bruits engendrés par chaque capteur, des paramètres liés au contrôleur de vol, qui sont modifiables, servent à déterminer une estimation de ce bruit.
   A titre d'exemple, le bruit estimé du gyroscope et de l'accéléromètre est utilisé pour estimer la croissance de l'erreur dans les angles, les vitesses et la position calculée à l'aide des mesures de l'IMU. L'augmentation de ces paramètres entraîne une augmentation plus rapide de l'estimation de l'erreur des filtres. Si aucune correction n'est apportée à l'aide d'autres mesures (par exemple SLAM ou bien GPS), cette estimation d'erreur continuera de croître. Ces erreurs estimées sont capturées dans une grande matrice appelée « matrice de covariance ».

Remarque importante :
Les étapes 1 à 5 sont répétées à chaque fois que nous obtenons de nouvelles données de l'IMU jusqu'à ce qu'une nouvelle mesure d'un autre capteur soit disponible. Si nous avions une estimation initiale parfaite, des mesures IMU parfaites et des calculs parfaits, nous

pourrions continuer à répéter 1 à 4 tout au long du vol sans aucun autre calcul requis. Cependant, les erreurs dans les valeurs initiales, les erreurs dans les mesures IMU et les erreurs d'arrondi dans nos calculs signifient que nous ne pouvons attendre que quelques secondes avant que les erreurs de vitesse et de position deviennent trop importantes.

2.4.2.2 La phase de correction

La deuxième étape de l'algorithme utilise les estimations a priori issues de la phase de prédiction pour les mettre à jour en calculant les estimations a posteriori de l'état et de la covariance d'erreur, le gain de Kalman filtre est calculé de manière à minimiser la covariance d'erreur a posteriori.
Ci-dessous les équations théoriques du modèle de correction :

$$\begin{cases} \tilde{y}_k = z_k - h(\hat{x}_{k-1|k-1}) & \text{résidu de mesure} \\ S_k = C_k P_{k|k-1} C_k^T + R_k & \text{innovation} \\ k_K = P_{K|K-1} C_k^T S_k^{-1} & \text{gain de Kalman} \\ \hat{x}_{k|k} = \hat{x}^-_{k|k-1} + k_K \tilde{y}_k & \text{mise à jour d'état} \\ P_k = (I - K_k C_k) P_{k|k-1} & \text{mise à jour de la covariance d'estimation} \end{cases} \tag{2.43}$$

Voici une description non mathématique très simplifiée du fonctionnement du filtre :

1. Lorsqu'une estimation SLAM arrive, le filtre calcule la différence entre la position prédite de l'étape 4 et la position estimé. Cette différence s'appelle une 'Innovation'.

2. L'Innovation' de l'étape 1, la 'Matrice de covariance' de l'étape 5 et l'erreur d'estimation du SLAM de la position spécifiée sont combinées pour calculer une correction de chacun des états du filtre. C'est ce qu'on appelle une « correction d'état ». C'est la partie intelligente du filtre de Kalman, car il est capable d'utiliser la connaissance de la corrélation entre différentes erreurs et différents états pour corriger des états autres que celui mesuré. Par exemple, les mesures estimées par le SLAM sont aptes à corriger les erreurs de position, de vitesse, des angles et de biais du gyroscope. La quantité de correction est contrôlée par le rapport de l'erreur d'état par l'erreur de mesure. Cela signifie que si le filtre pense que sa propre position calculée est plus précise que la mesure fournie par le SLAM, la correction de l'estimation SLAM sera plus petite. Similairement, s'il pense que sa propre position calculée est moins précise que l'estimation SLAM, la correction de l'estimation SLAM sera plus importante. La précision supposée de l'estimation SLAM est contrôlée par le un paramètre, Agrandir ce paramètre fait penser au filtre que la position SLAM est moins précise.

3. Parce que nous avons maintenant pris une mesure, la quantité d'incertitude dans chacun des états qui ont été mis à jour est réduite. Le filtre calcule la réduction de l'incertitude due à la 'correction d'état' et il met à jour la 'matrice de covariance d'état' pour revenir ensuite à l'étape 1

L'algorithme du filtre de Kalman étendu nous permet de fusionner les données de l'IMU, du SLAM, du baromètre, du capteur sonore, et d'autres capteurs pour calculer une estimation plus précise et plus fiable de l'état du système, (i.e. position, vitesse et orientation angulaire).
Un schéma plus détaillé est présenté dans la figure 2.20 :

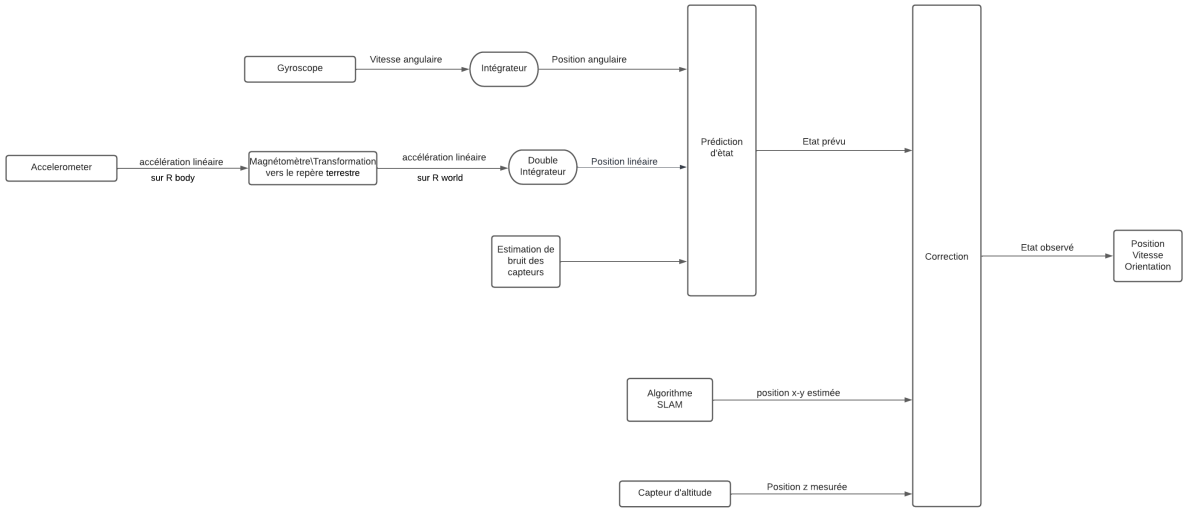


Fig. 2.20 : Schéma representatif détaillé du filtre EKF

Chapitre 3

Localisation et Cartographie Simultanées

Introduction

Dans ce chapitre, nous allons présenter le problème du SLAM, afin de mieux situer son implication dans notre projet. Nous commençons d'abord par les origines et la formulation initiale du problème, ce qui nous ramène à la description probabiliste du SLAM. Ensuite nous expliquons l'algorithme Hector Slam qu'on a implémenté dans notre projet comme source de localisation dans le plan horizontale, avec des formules clés qui lui sont liées.

3.1 Définition

3.1.1 Les origines

Le problème de localisation et cartographie simultanées (SLAM) traite deux questions importantes en robotique mobile. La première question est : 'Ou suis-je' ? Dont la réponse est donnée par la localisation du robot. La deuxième, concerne l'environnement du robot : ' A quoi ressemble l'environnement ou je me trouve ?'.

Dans un système SLAM, un véhicule robotisé placé dans un environnement inconnu et une position inconnue, doit construire la carte de l'environnement tout en essayant de se localiser par rapport à cette carte. Un tel robot compte sur plusieurs capteurs qui lui permettent de récupérer les informations dont il a besoin.

La formulation du problème SLAM a permis de définir une cadre de résolution probabiliste. L'énoncé du SLAM, pour la première fois, été durant la conférence IEEE Robotics and automation conference en 1986 à San Francisco [18].

SLAM est devenu un sujet de recherche très courant, spécialement durant les dernières décennies grâce à son utilité énorme pour les robots autonomes. Il est largement utilisé lorsqu'un robot mobile devra être complétement autonome. Il y a aussi des cas ou les robots ne peuvent totalement pas compter sur les systèmes de positionnement extérieurs comme le GPS, c'est le cas pour les milieux intérieurs, due à l'interruption de la ligne de vision. Les formules de calcul de GPS nécessitent un récepteur et au minimum quatre satellites pouvant se connecter entre eux sans aucune perturbation des signaux. Pour ce faire, le recepteur GPS doit obligatoirement être accéssible, ce qui ne pourra jamais être le cas dans un milieu intérieur, d'où l'importance du SLAM.
Les recherches sont focalisées sur l'amélioration des algorithmes de SLAM en termes d'efficacité et du temps de calcul. Le SLAM est destiné aux problèmes de positionnement, de cartographie ou bien des deux à la fois. Il permet alors à un système de naviguer dans un environnement inconnu sans avoir besoin de prédéfinir une cartographie de celui-ci ou bien de dépendre sur d'autres systèmes de localisation comme le GPS.

Le problème de positionnement, appelé aussi l'odométrie, est considéré le problème le plus basique en robotique mobile. Par ailleurs, l'odométrie est, par définition, l'estimation de position en fonction du temps, en utilisant des capteurs appropriés tel que IMU, encodeurs, cameras, lidars ou autres. Cependant, certains capteurs sont limités par des contraintes relatives à la précision, l'environnement et l'application [19]. Par exemple, les encodeurs soumissent souvent au glissement des roues qui pourra induire des erreurs non

tolérables. De plus, ils sont restreints aux robots terrestres uniquement et évidemment inutiles pour les véhicules aériens ou marins. Pour cette raison, l'odométrie est souvent effectuée par des techniques d'estimation issues des capteurs extéroceptifs (cameras, lidars, GPS...).
Par contre, tout capteur est susceptible aux bruits, et en raison que l'odométrie soit construite à partir des données imparfaites, elle va générer des erreurs sur l'estimation de position. Une solution à ce problème fait appel aux systèmes SLAM.

Les systèmes SLAM sont variés en fonction de leur complexité, les capteurs qu'ils utilisent ainsi que leur efficacité. Les deux familles majeures des systèmes SLAM sont les systèmes SLAM basé sur les Lidars (ou télémétrie laser) ainsi que les systèmes SLAM basés sur des camera (SLAM visuelle) [20], [21]. Dans notre projet on a utilisé le SLAM lidar en raison de sa simplicité et son efficacité par rapport à notre application.

3.1.2 Formulation du problème SLAM

Le SLAM est constitué d'un ensemble de techniques permettant un robot de construire une carte de l'environnement permettant de se localiser dans un milieu présentant des obstacles.

La trajectoire du robot et la position des amers dans la carte sont estimées au fur et au mesure, sans avoir besoin d'aucune connaissance a priori.

Soit un robot qui se déplace dans un environnement inconnu, et à partir des capteurs que dispose le robot, on peut faire la perception de son environnement tout en observant un certain nombre d'amers. La figure 3.1 montre une description du problème.

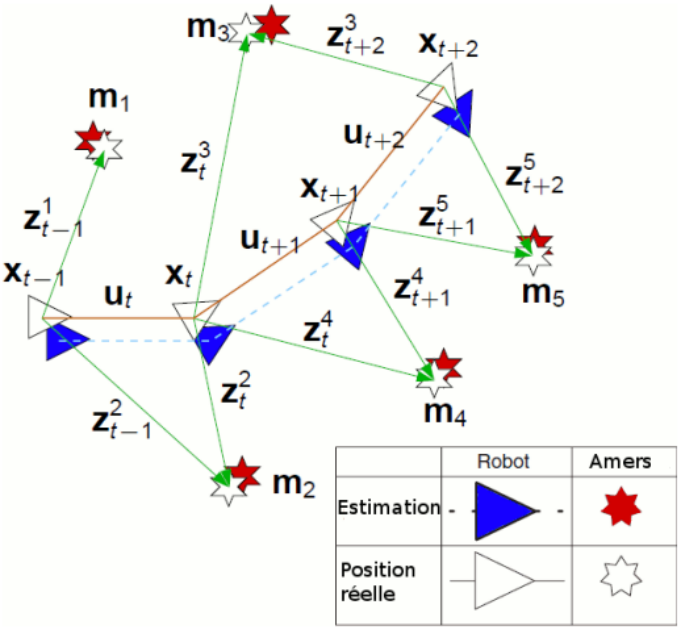


Fig. 3.1 : L'idée de base du SLAM

On définit les quantités suivantes :

- x_k : le vecteur d'état. Il contient la position et la rotation du robot à l'instant k.
- u_k : le vecteur de commande. L'application de la commande u_k a l'instant k-1 mène le robot de l'état $x_{(k-1)}$ a l'état x_k .
- m_i : le vecteur position de l'amer i.
- z_k : l'observation a l'instant k .

On définit aussi les ensembles suivants :

- $X_{0:k} = \{x_0, x_1, ..., x_k\} = \{X_{0:k-1}, x_k\}$: l'ensemble des vecteurs d'états jusqu'à l'instant k.
- $U_{0:k} = \{u_0, u_1, ..., u_k\} = \{U_{0:k-1}, u_k\}$: l'ensemble des vecteurs de commande jusqu'à l'instant k.
- $Z_{0:k} = \{z_0, z_1, ..., z_k\} = \{Z_{0:k-1}, z_k\}$: l'ensemble des observations jusqu'à l'instant k.
- $m = \{m_0, m_1, ..., m_n\}$: la carte d'environnement contenant une liste d'objet statique.

3.1.3 La localisation

La problématique de localisation du robot consiste à estimer sa position dans un environnement donnée, en utilisant toutes ses observations, l'historique des commandes et la connaissance de l'environnement. La figure 3.2 illustre ce principe.

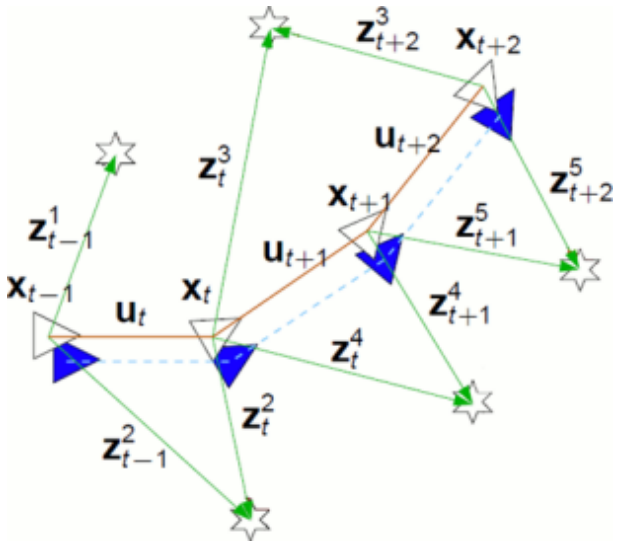


FIG. 3.2 : la localisation

Ce phénomène peut être représenté par l'estimation de la probabilité de distribution :

$$P(x_k|Z_{0:k}, Z_{0:k}, m) \tag{3.1}$$

Cette quantité de probabilité sert à estimer la position du robot en utilisant toutes les observations, les commandes précédentes et à quoi se ressemble l'environnement. Au

fait, tous les algorithmes SLAM utilisent cette distribution de probabilité pour faire la localisation, tandis que, chacun utilise différentes techniques et hypothèses pour obtenir une bonne estimation.

3.1.4 La cartographie

La cartographie consiste à déterminer une carte de l'environnement, en utilisant les données des capteurs et l'historique des positions réelles du robots (voir la figure 3.3). On peut exprimer cela par la distribution de probabilité 3.1.

La position du robot peut s'obtenir en utilisant des capteurs d'odométrie dans un environnement interne ou un récepteur GPS en externe. Ces positions doivent être précis, fiable et correct afin d'obtenir une bonne estimation de la carte.

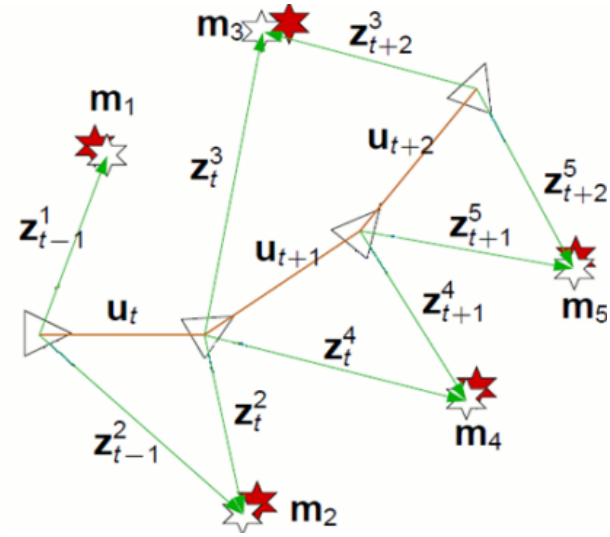


FIG. 3.3 : la cartographie

3.1.5 Le SLAM

On pourra réunir les problèmes de localisation et cartographie en un seul problème qui est bien le SLAM, par conséquent, la problématique du SLAM revient au calcul, à chaque instant k, de la quantité de probabilité 3.1

Le calcul de cette probabilité est généralement effectué récursivement, le SLAM en ligne intéresse à l'estimation de la position à l'instant k et la carte d'environnement m, pour cette raison, si on intégre tous les emplacements ainsi que toutes les rotations possibles allant de x_0 jusqu'à x_{k-1}, comme montre l'équation 3.2, on trouve la distribution de probabilité à l'instant k.

$$P(x_k, m|Z_{0:k}, Z_{0:k}, x_0) = \int_{x_0} ... \int_{x_{k-1}} P(x0 : k, m|z_{0:k}, u_{0:k}) dx_{k-1}.....dx_0 \qquad (3.2)$$

La structure globale du SLAM est présentée dans le schémas de la figure 3.4, dans ce dernier les cercles gris représentent les donnée connues, tandis que, les cercles blancs signifient les quantités à estimer.

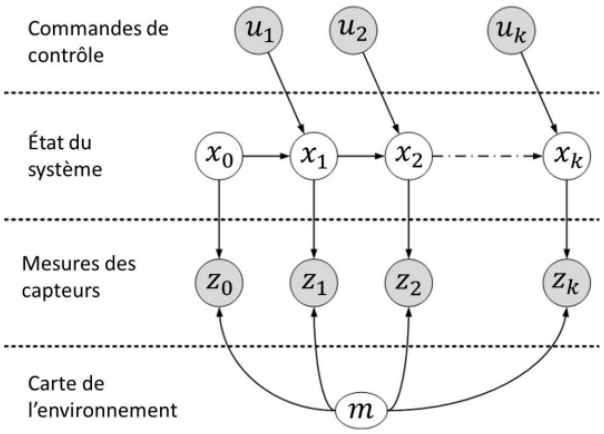


FIG. 3.4 : Représentation graphique du problématique SLAM

3.2 Résolution du SLAM

Plusieurs recherches sont effectuées pour la résolution du problème SLAM, Les techniques principales de SLAM sont basées sur des méthodes d'estimation statistiques, qui s'agissent des techniques de filtrage statistique permettant d'estimer l'état d'un système dynamique à partir des données provenantes d'un ou plusieurs capteurs. Les algorithmes développés dans ce sens peuvent être classés selon les types de capteurs utilisés, les techniques de calcul adopté ainsi que les types de cartes représentant l'environnement. En générale, les algorithmes du SLAM sont soit basés sur la vision par ordinateur en utilisant une caméra [22], ou bien sur des capteurs à laser [23]. Il existe deux grandes approches calcul, d'une part, les algorithmes basés sur l'utilisation du filtrage de Kalman [24], et d'autre part, les algorithmes qui utilisent des filtres à particule [25].

3.2.1 Représentation de la carte

Le choix de la présentation de la carte de l'environnement est une étape importante dans le SLAM. Dans [26], l'auteur explique trois approches fondamentales de représentation de l'environnement :

- L'approche directe [27].
- L'approche basée sur les caractéristique géométrique (feature-based) [28].
- L'approche basée sur une grille d'occupation (grid-based) [29].

3.2.2 L'approche directe

L'approche directe de la présentation de la carte de l'environnement consiste à utilisé les données brutes des mesures du capteur laser sans aucune extraction d'amer ou de caractéristique prédéfinies (lignes, coins . . .).

3.2.3 L'approche basée sur les caractéristique géométrique

En englais elle s'appelle feature-based. Cette méthode utilise des caractéristiques prédéfinies généralement des primitives géométriques, comme des lignes, des cercles …etc. afin de crée la carte de l'environnement. Pour détecter les caractéristique géométriques plusieurs méthodes existants [30] et [31].

3.2.4 L'approche basée sur une grille d'occupation

Les grilles d'occupation (grid-based) ont été introduit par [29], tel que, la carte de l'environnement est divisée en cellule rectangulaire.

Une carte en grille peut se présenter comme montré dans la figure 3.5. Il s'agit d'une matrice ou une séquence de cellule. La résolution de la représentation de la carte dépend directement de la taille de la grille d'occupation.

En plus de cette discrétisation de l'espace, une mesure de probabilité d'occupation est estimée pour chaque cellule indiquant si celle-ci est occupée ou libre, tel que chaque cellule a son occupation donée par une variable aléatoire.

L'idée de la carte de grille d'occupation est assez simple, l'occupation elle-même peut être considérée comme une variable aléatoire binaire, nous pouvons dire que la cellule m donnée par une cordonnée dans le plan x-y est noté $m_{x,y}$, qui peut prendre deux valeurs, soit zéro soit un, et nous faisons correspondre ces valeurs dans un espace réel , donc on peut définir une cellule comme suit :

$$m_{x,y} = \{libre, occupe\} = \{0, 1\} \tag{3.3}$$

La présentation de la carte par les grilles d'occupations nous ramène à attribuer à chaque grille une probabilité $p(m_{x,y})$, ce que pense le robot, y a t-il un obstacle dans cette cellule ou pas, et c'est la raison pour laquelle nous devons mettre à jour l'état de chacune des cellules au fur et à mesure le déplacement du robot dans l'environnement. Et pour cela nous utilisons le filtrage bayésien, ou nous mettrons à jour de manière récursive l'état de la probabilité d'occupation pour chaque cellule.

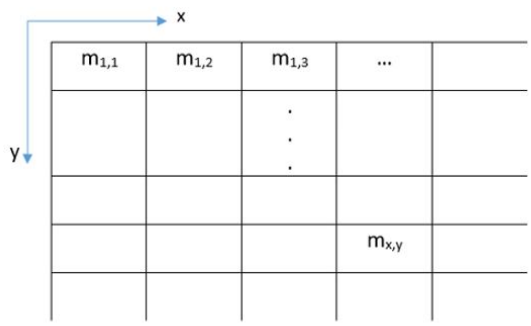


Fig. 3.5 : Grille d'occupation

3.2.5 La mise à jour de la carte

L'algorithme de mise-a-jour de la carte des cellules d'occupation estime la probabilité d'occupation de la cellule $m_{x,y}$ a posteriori noté $p(m_{x,y}|z)$, en utilisant les données provenant du lidar à chaque échantillon de la mise à jour de la carte, tout en utilisant l'état de chaque cellule de la carte a priori $p(m_{x,y})$ ainsi que le modèle de mesure qu'on a défini nous même $p(z|m_{x,y})$ (voir la figure 3.6).

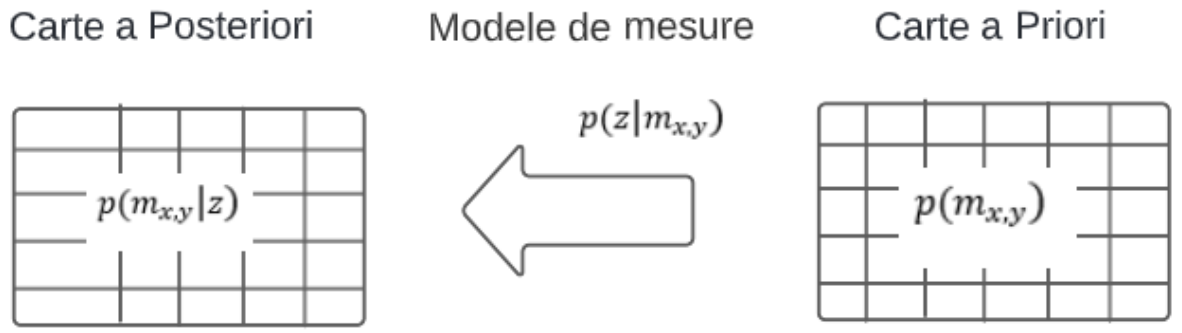


FIG. 3.6 : mise-a-jour de la carte de grille d'occupation

Le modèle de mesure $p(z|m_{x,y})$ est la probabilité d'observer un certain z qui peut prendre soit 1 soit 0 comme valeur, conditionnée par ce que nous savons sur la valeur de $m_{x,y}$, donc il y a que quatre cas pouvant se présenter par cette probabilité conditionnelle, ils sont les suivants :

- $p(z = 1|m_{x,y} = 1)$: vraie mesure que la cellule est occupée.
- $p(z = 0|m_{x,y} = 1)$: faux mesure que la cellule est libre.
- $p(z = 1|m_{x,y} = 0)$: faux mesure que la cellule est occupée.
- $p(z = 0|m_{x,y} = 0)$: vrai mesure que la cellule est libre.

Le filtrage bayésien récursive [32], [33] est une méthode utilisée pour mettre à jour la carte d'occupation. On définit la formule de la probabilité bayésien comme suit :

$$p(m_{x,y}|z) = \frac{p(z|m_{x,y})p(m_{x,y})}{p(z)} \tag{3.4}$$

Donc, ce que nous voulons faire est de calculer la probabilité des cellules qu'elles soient occupées ou libres à posteriori en fonction de la mesure provenant du scan lidar et de la probabilité d'occupation a priori.

Nous pouvons écrire la probabilité à posteriori associée à chaque cellule en fonction de la probabilité préalable. Pour cela, nous introduisons un concept pour simplifier ce truc récursive, donc au lieu de traiter des probabilité on traite la fonction *odd* qui se définit dans l'équation 3.5 comme le rapport de probabilité d'évènement devisée par la probabilité complémentaire de cet évènement :

$$odd = \frac{p(X)}{p(X^c)} \tag{3.5}$$

On définit la fonction *odd* que la cellule $m_{x,y}$ soit occupée en prenant compte la mesure z dans l'équation suivante :

$$odd(p(m_{x,y} = 1|z)) = \frac{p(m_{x,y} = 1|z)}{p(m_{x,y} = 0|z)} \tag{3.6}$$

Tel que :

$$p(m_{x,y} = 1|z) = \frac{p(z|m_{x,y} = 1)p(m_{x,y} = 1)}{p(z)} \tag{3.7}$$

$$p(m_{x,y} = 0|z) = \frac{p(z|m_{x,y} = 0)p(m_{x,y} = 0)}{p(z)} \tag{3.8}$$

En remplaçant 3.7 et 3.8 dans 3.6 on obtient :

$$\frac{p(m_{x,y} = 1|z)}{p(m_{x,y} = 0|z)} = \frac{p(z|m_{x,y} = 1)p(m_{x,y} = 1)}{p(z|m_{x,y} = 0)p(m_{x,y} = 0)} \tag{3.9}$$

On fait rentrer la fonction log sur les deux coté de l'équation 3.9 on obtient :

$$log(\frac{p(m_{x,y} = 1|z)}{p(m_{x,y} = 0|z)}) = log(\frac{p(z|m_{x,y} = 1)}{p(z|m_{x,y} = 0)}) + log(\frac{p(m_{x,y} = 1)}{p(m_{x,y} = 0)}) \tag{3.10}$$

On aura alors la relation de la mise-a-jour de la carte finale suivante :

$$log\ odd^{+} = log\ odd\ mesure + log\ odd^{-} \tag{3.11}$$

Cela veut dire que la fonction *logodd* de chaque cellule est égale à la fonction *logodd* précedente plus la fonction *logodd* de la probabilité de mesure.
A l'issue de cet algorithme on aurait à chaque instant une carte discrète de l'environnement, sous forme de cellules, chaque cellule serait associée par une probabilité que ce soit libre ou occupé. Cette distribution de probabilités le long de la carte sera mise à jour à chaque échantillon par la relation 3.11.

3.3 Hector SLAM

Ce système SLAM a été décrit par [33] étant une méthode flexible et évolutive, qui nécessite une faible ressource de calcul. Il offre une solution au problème du SLAM sur des processeurs à faible consommation d'énergie et à faible côut de calcul. Hector SLAM est un 2D SLAM basé sur les télémètres laser de haute résolution et haute fréquence avec un taux de balayage élèvé, comme d'autres systèmes SLAM tel que Gmapping [34] et Carthographer [35]. Dans [33] il est expliqué comment une unité centrale inertielle IMU se combine avec un lidar pour fair une estimation d'ètat selon six degrés de liberté : l'algorithme SLAM effectue une carte en 2D ainsi que Hector SLAM présente sa carte par l'approche de grille d'occupation. Dans le but d'établir le SLAM, on utilise la processus de scan-matching afin d'aligner les scans laser avec la carte précédante, à l'issue de cette étape il estime la position. Le premier balayage laser est inscrit sur la carte et les scans suivants sont mises en correspondance avec celle-ci. De plus, il utilise l'approche de Gauss-Newton expliqué dans [36] pour trouver la transformation qui donne la meilleure correspondance de balayage laser avec la carte. Les scan LIDAR sont écrits sur la carte en fonction des critères de déplacement minimum de translation ou de rotation par rapport à l'emplacement de l'enregistrement de la cartographique précédente.

3.3.1 L'estimation de la position par la mise en correspondance

Afin d'estimer la position du quadri-rotor dans le plan x-y, l'algorithme Hector SLAM passe pour le scan-matching (la mise en correspondance). Le scan-matching a largement été utilisé dans des nombreux travaux. On va commencer par une brève description de certaines approches connues afin de mettre les explications données dans cette partie dans le bon contexte. Les algorithmes de scan-matching peuvent être divisés en trois parties.

3.3.1.1 Amer-amer

une amer est une marque dans la carte du SLAM pouvant être un point, une droite, un plan.. . Lorsqu'on cherche des correspondances entre des amers, on parle d'une approche 'amer-amer'. On extrait des lignes [37], ou des coins [38] à partir des données du laser afin de les comparer. Dans de tell méthode, l'algorithme a besoin d'un environnement structuré afin d'en extraire facilement les caractéristiques et les amers les plus connus.

3.3.1.2 Scan-amer

On cherche une mise en correspondance entre des points d'un scan laser et des amers, tel des lignes **cox1991blanche** par exemple. Cet amer peut être une partie d'une carte que l'algorithme de SLAM vient de construire. Il n'est pas toujours nécessaire d'avoir des environnements structurés.

3.3.1.3 Scan-scan

Dans les approches qui se basent sur une comparaison 'scan-scan', l'algorithme ne dépend pas de l'existence d'amers dans l'environnement. La mise en correspondance est faite directement sur les données brutes du laser, en utilisant une variation que l'algorithme cherche a minimiser [28].

Plusieurs travaux de recherche se sont intéressés a l'amélioration de l'étape de scan-matching dans les algorithmes du SLAM. Certain visent a accélérer la mise en correspondance en changeant la façon d'interpréter les données du laser [29], tandis que autres méthodes choisissent de changer l'espace de travail pour améliorer la qualité de l'appariment [39].

Le scan-matching implémenté dans Hector SLAM se base sur une comparaison entre les données du scan laser et la carte établie.

3.3.2 La localisation par Hector SLAM

Pour mieux expliquer le processus de localisation, on représente les étapes clés du processus de localisation.Voici une explication des étapes représenter dans la figure 3.7.

- Le robot commence sa cartographie. Il enregistre les données du laser dans la carte.
- Le robot se deplace. Il recupere de nouvelle données du capteur laser. La carte reste inchangée.

- L'algorithme de SLAM calcule la valeur de la probabilité totale du nuage des points avec la carte locale en utilisant la méthode expliquée dans la section 3.4.2 . Il utilise pour cela les nouvelles données du laser et la carte établie.
- Lorsque l'algorithme trouve la meilleure estimation de la position du quadrirotor, il met à jour sa carte en y ajoutant les nouvelles données du laser.

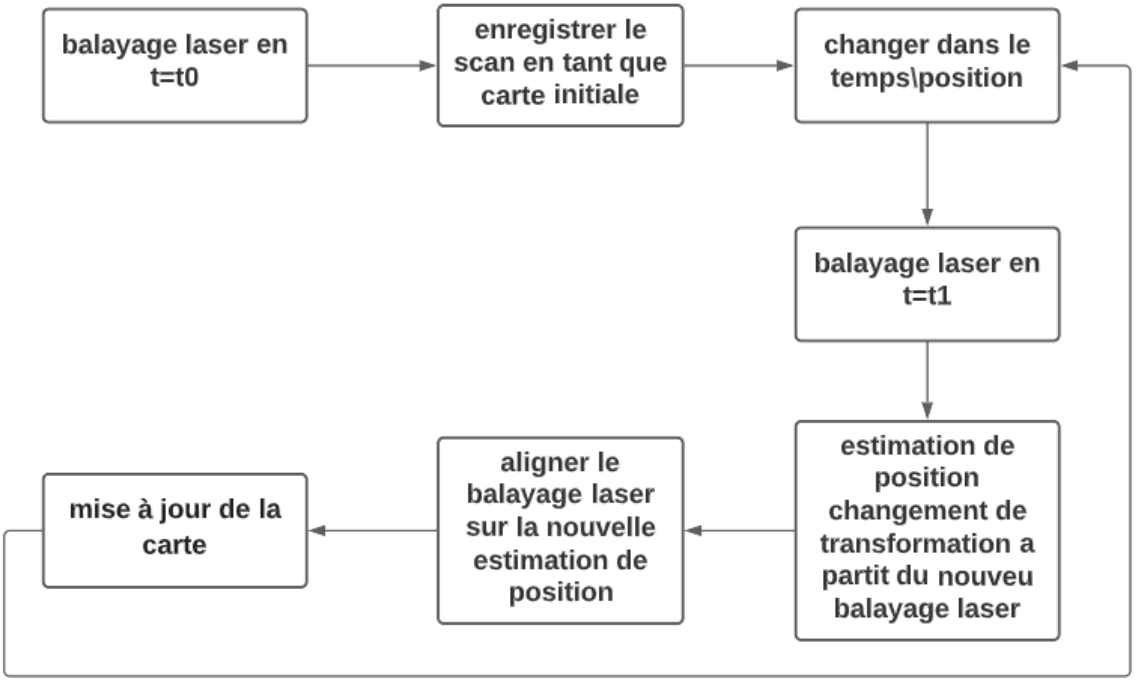


Fig. 3.7 : schémas globale de fonctionnement du Hector SLAM

3.3.3 Le calcul de la probabilité

La nature discrète des grilles d'occupation limite la précision qu'on peut atteindre, et elle ne permet non plus d'effectuer un calcul direct des valeurs interpolées ou dérivées. Pour cela un schéma d'interpolation figure 3.8 permettant une cellule de sous-grille la précision par filtrage bilinéaire est utilisé afin d'estimer à la fois les probabilités d'occupation et les dérivées. Intuitivement les valeurs des cellules de la carte grille peuvent être considérées comme une distribution de probabilité continue.

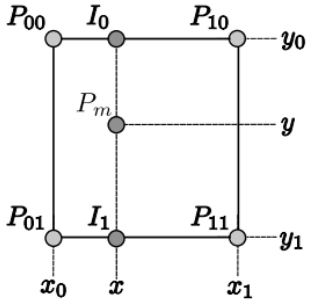


Fig. 3.8 : Filtrage bilinéaire de la grille d'occupation

Etant donnée une coordonnée dans la carte continue Pm, sa valeur de probabilité est $M(P_m)$, ainsi que le gradient $\nabla M(P_m) = (\frac{\partial M}{\partial x}(P_m), \frac{\partial M}{\partial y}(P_m))$ peut-être approximée en utilisant les quatre coordonnées entières les plus proches P_{00} ,P_{01},P_{10} et P_{11}, comme représente la figure 3.8. L'interpolation bilinéaire le long des axes x et y nous conduit a écrire :

$$M(P_m) = \frac{y - y_0}{y - y_0}(\frac{x - x_0}{x_1 - x_0}M(P_{11}) + \frac{x_1 - x}{x_1 - x_0}M(P_{01})) + \frac{y_1 - y}{y1 - y_0}(\frac{x - x_0}{x_1 - x_0}M(P_{10}) + \frac{x_1 - x}{x_1 - x_0}M(P_{00})) \tag{3.12}$$

Le gradient peut être approchée par :

$$\frac{\partial M}{\partial x}(P_m) = \frac{y - y_0}{y_1 - y_0}(M(P_{11}) - M(P_{01})) + \frac{y_1 - y}{y_1 - y_0}(M(P_{10}) - M(P_{00})) \tag{3.13}$$

$$\frac{\partial M}{\partial y}(P_m) = \frac{x - x_0}{x_1 - x_0}(M(P_{11}) - M(P_{10})) + \frac{x_1 - x}{x_1 - x_0}(M(P_{01}) - M(P_{00})) \tag{3.14}$$

3.3.4 La mise en correspondance

La phase de la mise en correspondance (scan-matching) doit être la plus précise possible, tout en restant rapide afin de garantir un fonctionnement en temps réel. La mise en correspondance s'appuie sur un algorithme d'optimisation cherchant la meilleure position du robot, de sorte que l'observation (le dernier scan laser) corresponde au mieux à la carte. Pour qu'il puisse être rapide et précis, cet algorithme de recherche s'appuie sur une représentation de la carte de SLAM facilitant la recherche d'un minimum de distance entre les données du scan laser et la carte. L'algorithme Hector SLAM est basé sur l'approche d'optimisation non linéaire Gauss-Newton qui était inspiré par [21], afin de trouver la meilleure estimation de position. La méthode consiste à trouver la position du LIDAR pour assurer que la probabilité totale du nuage des points dans la carte locale est suffisamment grande. Dans le but de trouver la transformation rigide $\xi = (p_x, p_y, \psi)$, l'idée est de maximiser la probablité de tous les scans étant donné que le robot se trouve dans la position ξ. Cela revient à miximiser la fonction objective suivante :

$$\xi^* = \arg\max_{\xi} \sum_{i=1}^{n} [M(S_i(\xi))]^2 \tag{3.15}$$

Notamment, on peut transformer notre probleme de maximisation à un problème de minimisation :

$$\xi^* = \arg\min_{\xi} \sum_{i=1}^{n} [1 - M(S_i(\xi))]^2 \tag{3.16}$$

Où
$S_i(\xi)$ est la position dans le repère global du point final du scan
$S_i = (s_{i,x}, s_{i,y})^T$ est une fonction en terme de la position du robot dans le repére global ξ

$$S_i(\xi) = \begin{pmatrix} cos\psi & -sin\psi \\ sin\psi & cos\psi \end{pmatrix} \begin{pmatrix} s_{i,x} \\ s_{i,y} \end{pmatrix} + \begin{pmatrix} p_x \\ p_y \end{pmatrix} \tag{3.17}$$

La fonction $M(S_i(\xi))$ retourne la probabilité de la carte tel que la coordonnée est donnée par $S_i(\xi)$. On commence par la position ξ et on veut estimer la variation $\Delta\xi$ qui

minimise l'erreur de mesure suivante :

$$\sum_{i=1}^{n}[1 - M(S_i(\xi + \Delta\xi))]^2 \to 0 \tag{3.18}$$

Le développement de Taylor du premier ordre de $M(S_i(\xi + \Delta\xi))$ nous donne :

$$\sum_{i=1}^{n}[1 - M(S_i(\xi)) - \nabla M(S_i(\xi))\frac{\partial S_i(\xi)}{\partial \xi}\Delta\xi]^2 \to 0 \tag{3.19}$$

Cette équation est à minimiser, en appliquant la dérivée partielle par rapport à la variation de position $\Delta\xi$ qui soit égale à zéro :

$$2\sum_{i=1}^{n}[\nabla M(S_i(\xi))\frac{\partial S_i(\xi)}{\partial \xi}]^T[1 - M(S_i(\xi)) - \nabla M(S_i(\xi))\frac{\partial S_i(\xi)}{\partial \xi}\Delta\xi] = 0 \tag{3.20}$$

La solution pour $\Delta\xi$ est donnée par l'équation de Gauss-Newton, ce qui revient à un problème d'optimisation :

$$\Delta\xi = H^{-1}\sum_{i=1}^{n}[\nabla M(S_i(\xi))\frac{\partial S_i(\xi)}{\partial \xi}]^T[1 - M(S_i(\xi))] \tag{3.21}$$

Où :

$$H = [\nabla M(S_i(\xi))\frac{\partial S_i(\xi)}{\partial \xi}]^T[\nabla M(S_i(\xi))\frac{\partial S_i(\xi)}{\partial \xi}] \tag{3.22}$$

L'approximation du gradient de la carte $\nabla M(S_i(\xi))$ est donnée dans la section 3.3.3. à partir de l'équation $S_i(\xi)$ on trouve :

$$\frac{\partial S_i(\xi)}{\partial \xi} = \begin{pmatrix} 1 & 0 & -sin(\psi)s_{i,x} - cos(\psi)s_{i,y} \\ 0 & 1 & cos(\psi)s_{i,x} - sin(\psi)s_{i,y} \end{pmatrix} \tag{3.23}$$

En utilisant $\nabla M(S_i(\xi))$ et $(\partial S_i(\xi))/\partial\xi$, l'équation de Gauss-Newton 3.21 serait calculé.
Il est important de noter que l'algorithme se base sur des approximations linéaires non lisses du gradient de la carte $\nabla M(S_i(\xi))$, ce qui signifie que la convergence quadratique locale vers un minimum ne peut pas être toujours garantie. Néanmoins, l'algorithme fonctionne avec une précision suffisante dans la pratique, vu qu'il y en a pas des variations brusques pouvant rendre la fonction objective non lisse. Pour de nombreuses applications, une approximation gaussienne de l'incertitude de la correspondance est souhaitable. Une approche consiste à utiliser la matrice hessienne pour l'estimation de la matrice de covariance est approchées par :

$$R = var\{\xi\} = \sigma^2 H^{-1} \tag{3.24}$$

3.4 Résultats de simulation

On présente ici les résultas de simulation des signaux obtenus par l'environnement de simulation developpé dans la section 5.3
Dans la figure 3.9, on compare la position x-y estimée par Hector SLAM avec la position mesurée par le GPS (considerée absolue) :

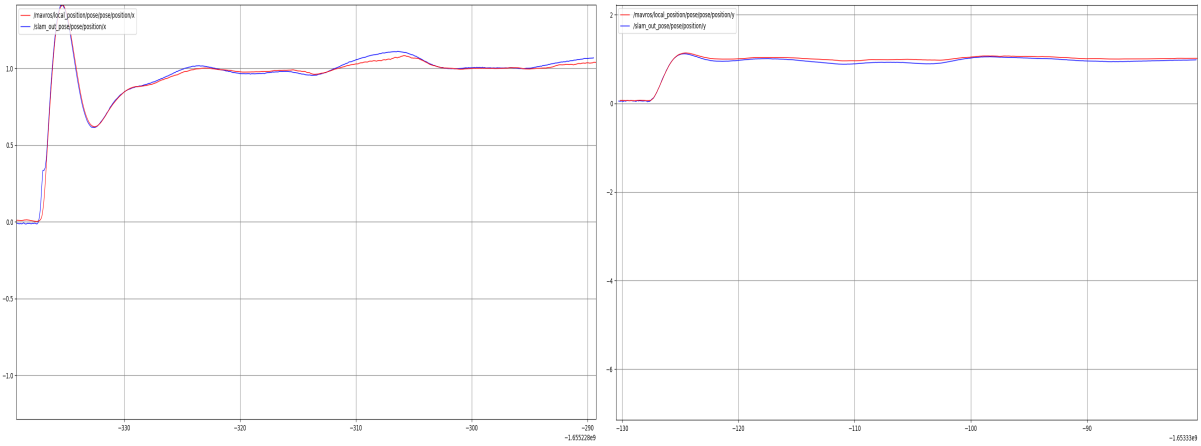


Fig. 3.9 : A gauche : la réponse du drone de simulation à la consigne de position (1,0,2.5) A droite : la réponse du drone de simulation à la consigne de position (0,1,2.5)

3.5 Conclusion

La position estimée par Hector SLAM est presque confondue avec la postion réelle, par conséquent, l'estimation de Hector SLAM est suffisamment précise qu'on peut la considérer comme une source de localisation en intérieur.

Chapitre 4

Évitement d'obstacle

4.1 Problématique

La planification de trajectoire se définit par le processus qui calcule une séquence de configurations appelées collision-free, i.e. une structure spatiale du robot qui soit libre de toute collision avec les obstacles lui entourant. En 3D, et dans le cadre de naviguer un robot aérien, sa configuration peut être définit par le vecteur contenant la position et l'attitude de ce dernier : (x, y, z, Ψ, θ, ϕ). Généralement, la planification de trajectoire en robotique s'intéresse aux translations et rotations requises pour faire bouger le robot proprement. Néanmoins, dans le but de sélectionner un algorithme de planification approprié à notre application, certains critères ont été définit [10] :

- Complétude : l'aptitude de l'algorithme à toujours trouver une solution au problème de planification si une existe ou bien indiquer une faillite en temps fini dans le cas échéant.
- Optimalité : dépend du temps d'exécution et/ou la longueur du chemin planifié, i.e. plus l'algorithme s'exécute rapidement et/ou le chemin est court, plus ce critère est vérifié.
- Complexité de calcul : pour dire à quel point l'algorithme soit valable lorsqu'on augmente la complexité de l'environnement.
- Effet de bruit du capteur : La robustesse de l'algorithme vis-à-vis l'incertitude que donne le capteur utilisé.

Après avoir considéré ces critères ainsi que les contraintes matériels qu'on face, on finit par choisir l'algorithme du « Champ de Potentiel ». En effet, c'est une méthode de planification séduisante par son élégance et sa simplicité, proposé pour la première fois par O. Khatib dans [8].

4.2 Pourquoi champ de potentiel

On a choisi cette méthode pour plusieurs raisons tel que :

- La planification est en temps réel ; ce qui s'aligne avec notre objective étant un drone autonome naviguant dans un environnement inconnu ;
- Cet algorithme est robuste par rapport à la complexité de l'environnement ainsi que le bruit du capteur, ce qui est bien attendu de notre implémentation ;
- La simplicité de l'algorithme ;
- La contrainte majeure de cette méthode c'est qu'il ne répond pas au critère d'optimalité ; Or, dans un premier temps, on priorise pas un chemin optimal ni un temps d'exécution court. Plutôt, on s'intéresse beaucoup plus sur la robustesse du système.

4.3 Le principe de la méthode

Le principe de champ potentiel est inspiré de la physique. Par exemple, une particule chargée naviguant dans un champ magnétique, l'idée est qu'en fonction de la force induite par ce champ, la particule peut arriver à la source du champ ou s'en éloigner.
En robotique, on peut stimuler le même effet en créant un champ de potentiel artificiel qui va attirer le robot vers le but. Dans un premier temps, supposons qu'il n'y ait pas d'obstacle dans l'environnement et que le robot doive converger vers un but. Il faut, donc, calculer la position relative du robot par rapport au but, puis appliquer les forces appropriées qui conduiront le robot vers le but.
Dans l'approche par champ potentiel, nous créons un champ attractif dirigé vers l'objectif. Le champ de potentiel est défini sur tout l'espace libre, et à chaque échantillon, on calcule le champ de potentiel pour chaque position du robot, puis on calcule la force induite par ce champ. Le robot doit alors se déplacer en fonction de cette force.
La figure ci-dessous 4.1 représente les champs potentiels d'attraction vers le but dans le segment du plan XY : [-10,10] $\times$ [-10,10], tel que le but est situé au centre (0, 0)

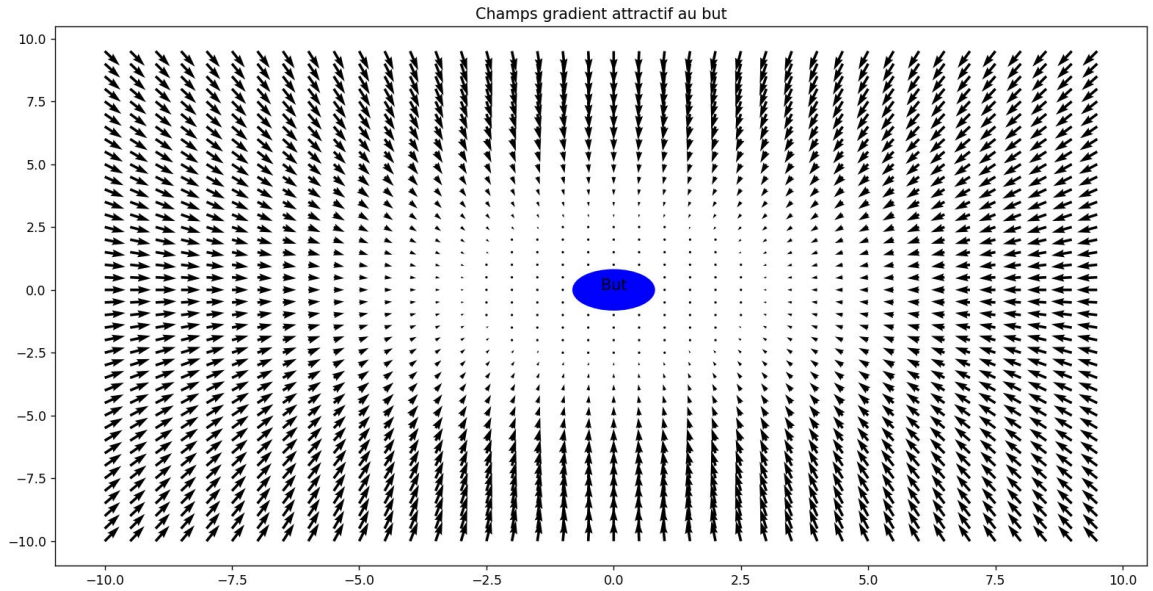


FIG. 4.1 : Le champ de gradient attractif utilisé, ploté par Python

On peut aussi définir un autre comportement qui permet au robot d'éviter les obstacles. Nous faisons en sorte que chaque obstacle génère un champ répulsif autour de lui. Si le robot s'approche de l'obstacle, une force répulsive lui repousse loin de l'obstacle, comme montré ci-dessous 4.2 :

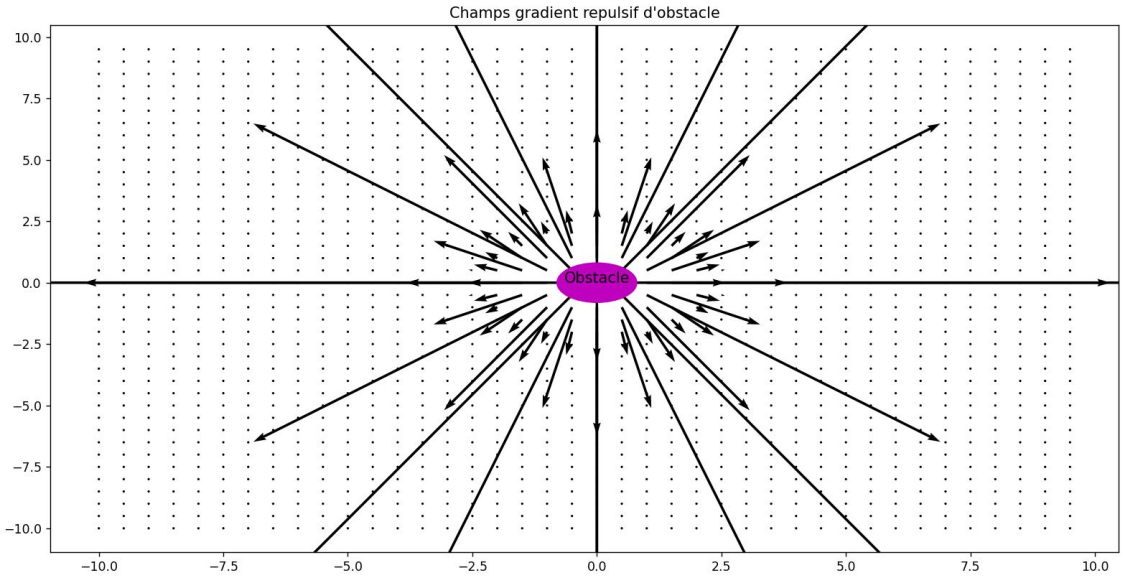


FIG. 4.2 : Le champ de gradient répulsif utilisé, ploté par Python

Les deux comportements, peuvent être combinés en superposant les deux champs potentiels. En effet, le mouvement du robot peut être interprété comme le mouvement d'une particule dans un champ vectoriel de gradient généré par des particules électriques. Dans cette analogie, le robot est une charge positive, le but est une charge négative et les obstacles sont des ensembles de charges positives. Les gradients dans ce contexte peuvent être interprétés comme des forces qui attirent la particule de robot chargée positivement vers une particule négative qui représente le but. Les obstacles agissent comme des charges

positives qui génèrent des forces répulsives faisant éloigner le robot des obstacles.
La combinaison des forces attractives et répulsives va conduire le robot dans un chemin sûr vers le but sans commettre des collisions, comme montre la figure ci-dessous 4.3 :

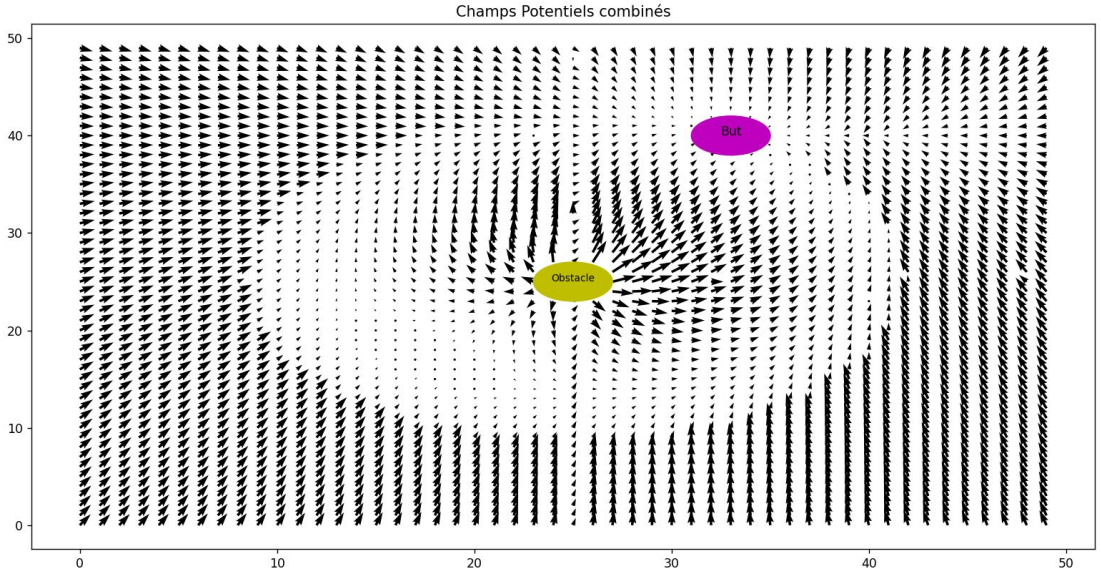


FIG. 4.3 : Le champ de gradient combiné, ploté par Python

4.3.1 Potentiel attractif

Le q_0 = (x, y) est la position 2D actuelle du drone dans le plan xy. Le choix le plus courant du potentiel attractif est la forme parabolique standard qui croît en quadrature avec la distance du but :

$$U_{att}(q_0) = \frac{1}{2} K_a d^2_{goal}(q_0) \tag{4.1}$$

tel que
d_{goal} : la distance eucleudienne bidimentionnelle entre le but et le drone,
K_a : le facteur d'attraction.
La force attractive dans cette approche est, par définition, le gradient négatif du potentiel attractif :

$$\hat{F} = -\nabla U_{att}(q_0) = -K_a(q_0 - q_{goal}) \tag{4.2}$$

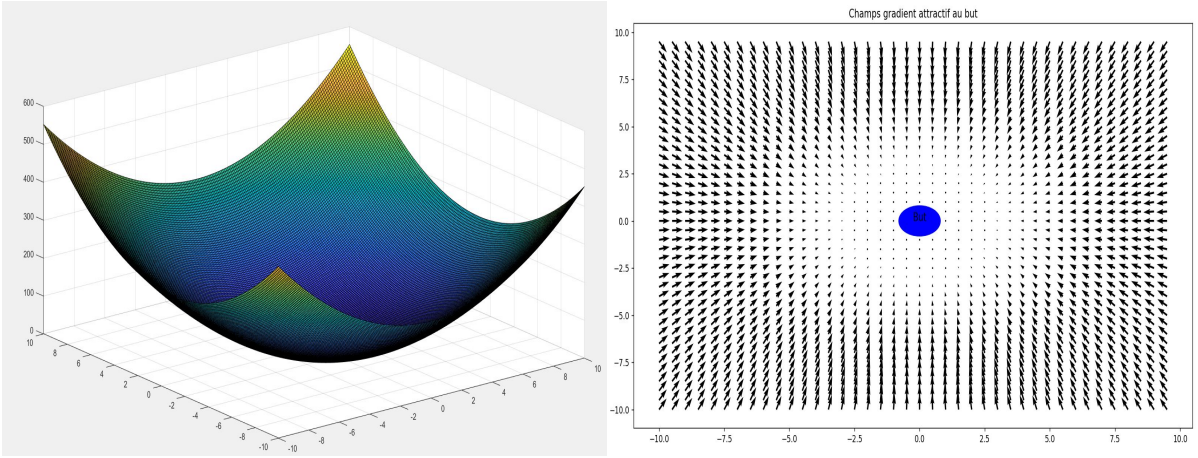


FIG. 4.4 : Visualisation de la loi attractive

4.3.2 Potentiel répulsif

Le potentiel répulsif fait éloigner le robot des obstacles qui soient détectés par le capteur embarqué dans robot (lidar 2D). Plus le robot s'approche de l'obstacle plus le potentiel répulsif est fort. Similairement, Plus le robot s'eloingne de l'obstacle plus le potentiel répulsif est faible. Par ailleurs, étant donné la nature linéaire du problème, le potentiel répulsif résultant sera la somme des effets répulsifs de chaque obstacle, d'où : $U_{rep(q)} = \sum_i U_(rep, i)$

C'est bien raisonnable de considérer que l'influence d'un obstacle est limitée seulement à son voisinage jusqu'à une distance donnée pour qu'un obstacle qui soit loin du robot ne risques pas de repousser ce dernier. De plus, l'amplitude du potentiel répulsif devra augmenter lorsque le robot s'approche de l'obstacle. Pour rendre compte de cela, un éventuel potentiel répulsif généré par un obstacle i est donné comme suit **??** :

$$U_{rep,i}(q) = \begin{cases} \frac{1}{2} K_{obs,i} (\frac{1}{d_{obs,i}(q)} - \frac{1}{d_0})^2 & \text{si } \mathrm{d}_{obs,i} < \mathrm{d}_0 \\ 0 & \text{sinon} \end{cases} \tag{4.3}$$

$d_{obs,i}$: distance minimale du robot aux obstacle i,
$k_{obst,i}$: facteur de répulsion,
d_0 : distance seuil d'influence

La force répulsive est donnée par le gradient négatif du potentiel $U_{rep,i}$:

$$F_{rep,i}(q) = \begin{cases} \frac{1}{2} K_{obs,i} (\frac{1}{d_{obs,i}(q)} - \frac{1}{d_0}) \frac{1}{d^2_{obs,i}} \frac{q - q_{obs}}{d_{obs,i}} & \text{si } \mathrm{d}_{obs,i} < \mathrm{d}_0 \\ 0 & \text{sinon} \end{cases} \tag{4.4}$$

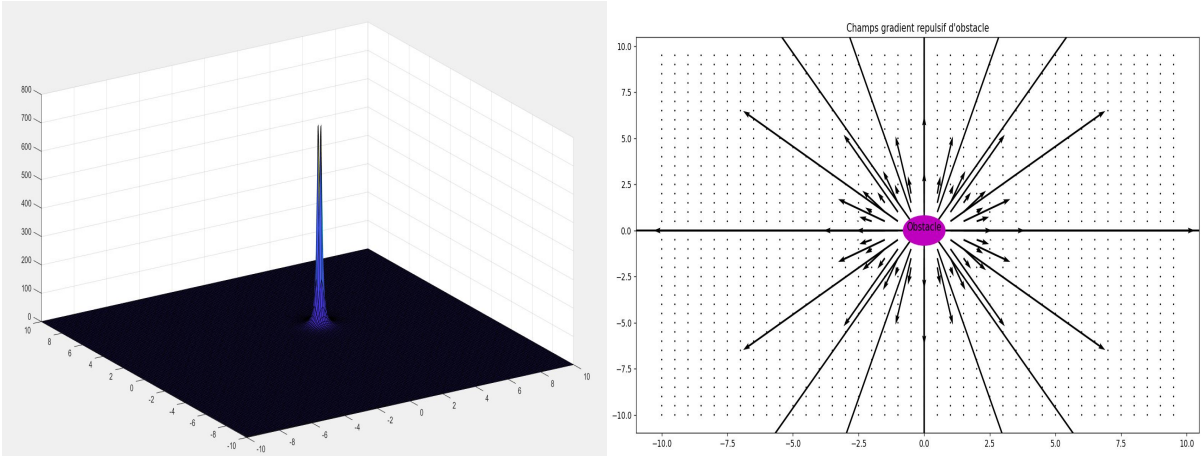


FIG. 4.5 : Visualisation de la loi répulsive

4.3.3 Potentiel total

Le champ potentiel résultant est donné par la superposition du potentiel répulsif résultant (U_{rep} avec le potentiel attractif :

$$U(q) = U_{att}(q) + \sum_i U_{rep}(q) \tag{4.5}$$

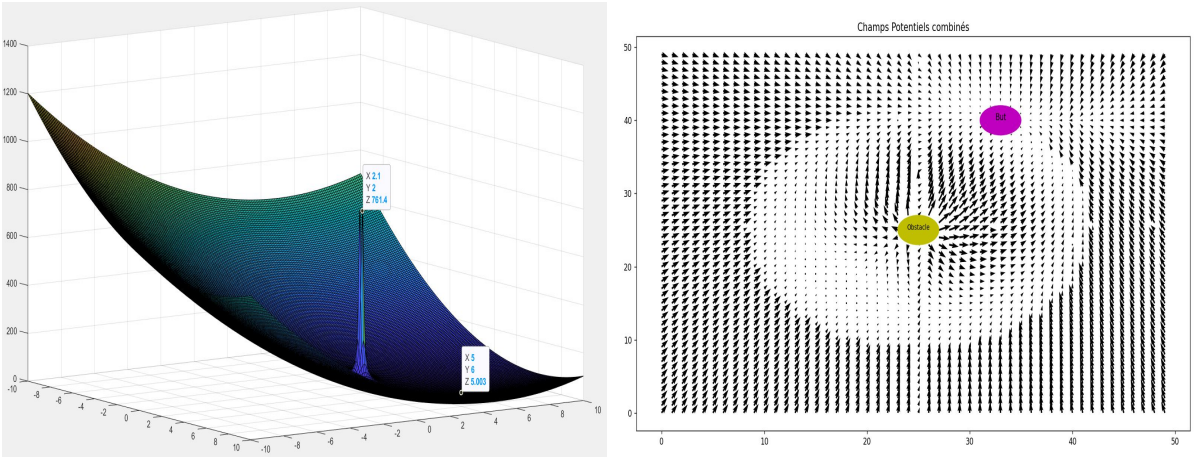


FIG. 4.6 : Visualisation de la loi des champs potentiels résultants

4.4 Implémentation de l'algorithme des champs potentiels

Dans notre approche, la génération de la trajectoire est basée sur l'idée des champs potentiels. À partir des formules ci-dessus de la loi attractive 4.1, 4.2 et répulsive 4.3, 4.4 ainsi que la loi de potentiel résultant 4.5, on calcule des commandes qui seront injectées au modèle du drone, comme representé dans la figure 5.17

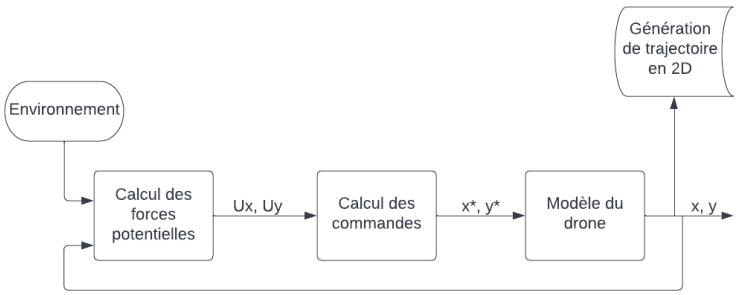


FIG. 4.7 : Schéma representatif de l'algorithme d'évitement

D'apres [40], la loi de commande le plus convenable comme entrée du système se définit par la relation : $\dot{q} = F(q)$
Tel que :
$\dot{q}$: la vitesse désiré à injecter dans le système
F(q) : la force calculé par le champ potentiel

En effet, on a développé un algorithme C++, sous l'environnement ROS, qui recoit le scan de lidar comme entrée sous forme d'un message ROS (/scan topic), ainsi, il extrait le nuage des points (par le modèle 5.1) représentant la position 2D des obstacles en coordonnées cartisiennes. Ensuite, en appliquant les formules ci-dessus 4.3, 4.4, on calcule les forces répulsives des points scannés et les forces attractives vers la position de but, qui soit pré-définie au début de l'algorithme. Finalement, à partir de la relation 4.5, on affecte aux deplacements Δx, Δy les forces resultantes selon x et y réspectivement, puis, on les envoie comme des déplacements désirés en faisant appel à la fonction setposition ayant les arguments $(x_{actuel} + \Delta x, y_{actuel} + \Delta y)$. Cette fonction envoie une position désirée que le drone devra suivre en mettant en execution les algorithmes de contrôle bas-niveau de Pixhawk4.
Le code C++ est disponible en annexe 7.2.

4.5 Probléme du minimum local

Cependant, en utilisant cette méthode, le robot peut facilement tomber dans un minimum local (comme montré dans la figure 4.8). Par conséquent, des efforts supplémentaires sont nécessaires pour éviter cette situation. Le problème du minimum local est parfois inévitable lorsqu'un objet se déplace dans des environnements inconnus, car l'objet ne peut pas prédire les minima locaux avant qu'il détecte les obstacles formant les minima locaux. L'échappement des minima locaux présente un sujet de recherche actif dans la planification de trajectoire basée sur le champ potentiel.
L'article [11] propose un nouveau concept qui utilise un obstacle virtuel pour échapper aux minima locaux qui se produisent dans la planification de chemin local, l'obstacle virtuel est situé autour des minima locaux pour repousser un objet des minima locaux.
Le projet [1] implémente aussi une planification par les champs potentiels d'un UAV et propose un algorithme stochastique qui résout le problème du minimum local ; "Simulated Annealing".

Remarque : Dans notre projet, on n'avait pas suffisamment de temps pour faire un algorithme d'échappement des minima locaux.

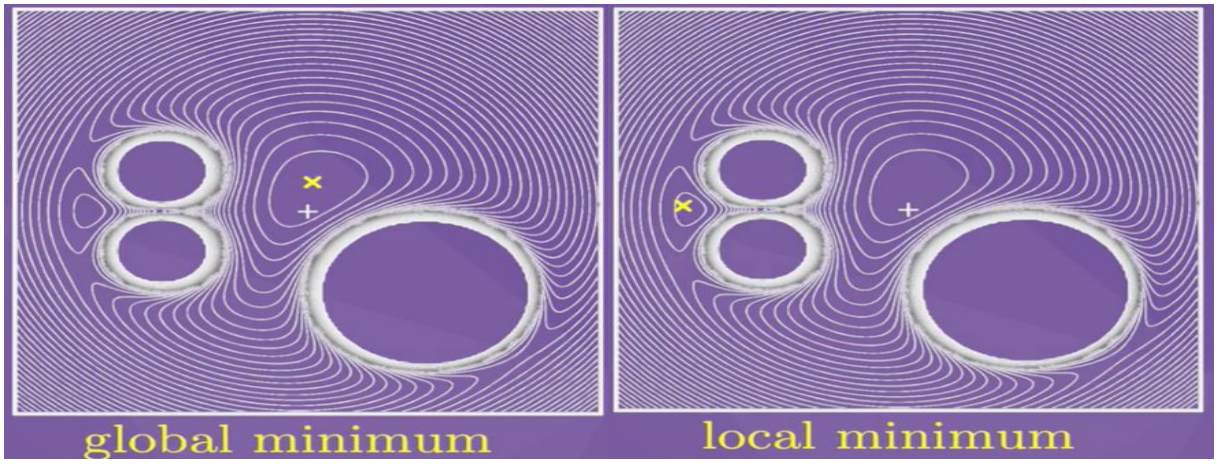


Fig. 4.8 : Exemple d'une loi potentielle présentant un minimum global et un minimum local

4.6 Résultats de simulation

On execute l'algorithme 7.2 parallèlement lors de l'execution du système de simulation décrit en 5.3, dans un milieu ouvert. Afin de tester les lois potentielles attractives et répulsives sur le drone de simulation, on procéde des testes de simulation qui montrent l'effet de l'algorithme sur le drone :

4.6.1 Fixer un but à l'origine en présence d'un obstacle

Dans un premier temps, on fait décoller le drone à 2.5m par la commande takeoff() (voir la figure 4.9), ensuite, on fixe un but à l'origine et puis on perturbe le système en lui rapprochant un obstacle :

Remarque importante :
Il faut noter que la plannification de trajectoire se fait seulement en 2D selon x y, avec une altitude fixée à 2.5m, car le lidar 2D ne donne que les distances des obstacles selon son plan horizontal.

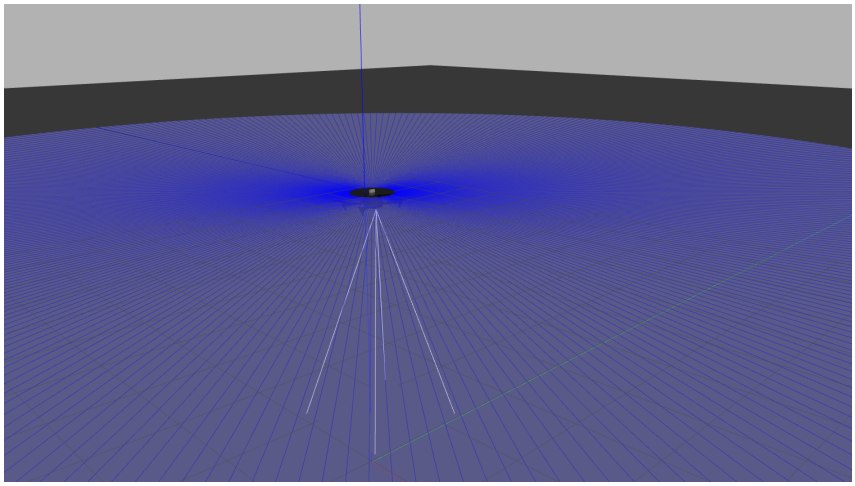


FIG. 4.9 : Drone de simulation maintient son altitude à 2.5m aprés avoir lui envoyé la commande takeoff(2.5)

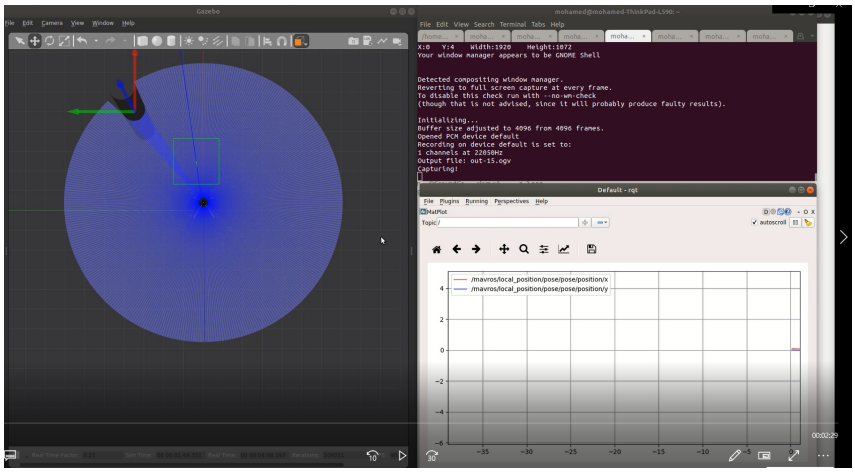


FIG. 4.10 : Drone de simulation maintient sa position à l'origine avant de lui rapprocher un obstacle

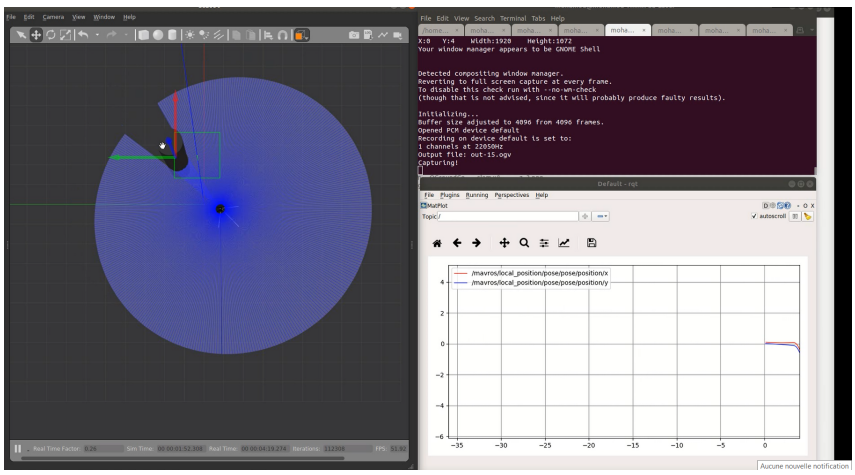


FIG. 4.11 : Drone de simulation s'éloinge de sa position aprés avoir détecté un obstacle, sous l'effet des forces répulsives artificielles

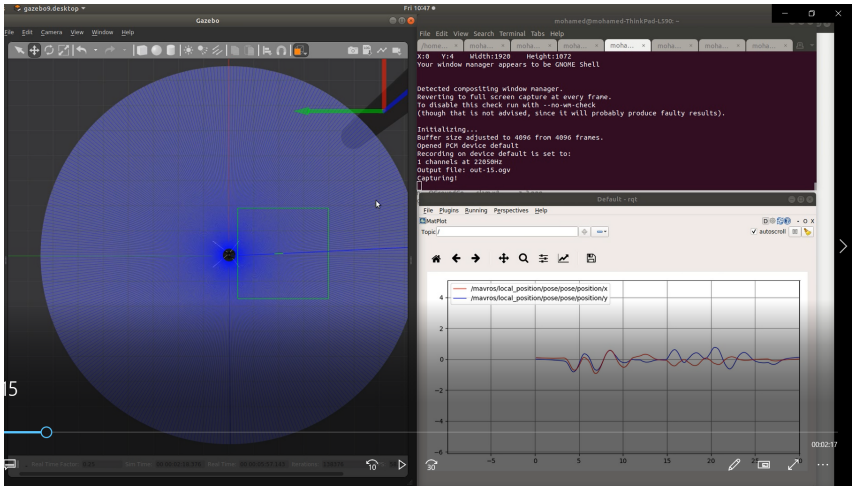


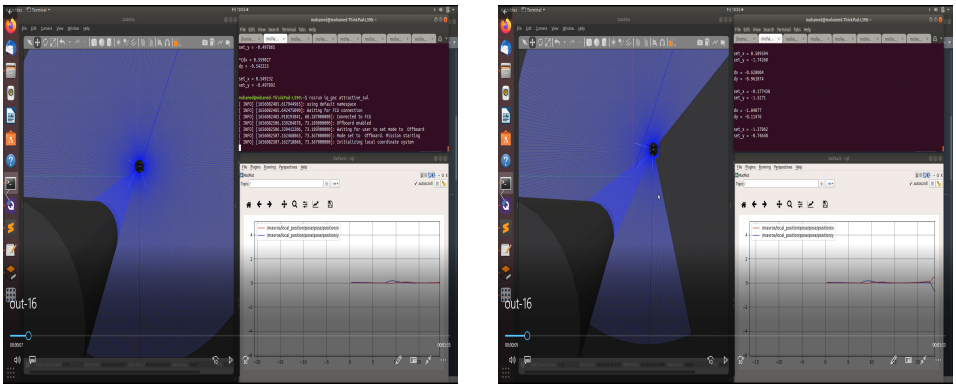
FIG. 4.12 : Drone de simulation retourne à l'origine aprés l'avoir éloigné de l'obstacle, sous l'effet de la force attractive artificielle

4.6.1.1 Interprétation

Lorsque l'obstacle est assez loin du drone, les forces répulsives artificielles sont, à ce stage là, négligables par rapport à la force attractive vers l'origine. Par contre, dés que la distance vers l'obstacle dépasse un certain seuil (à savoir d_0), elles deviennent de plus en plus fortes jusqu'au point de faire éloinger le drone de sa position désirée afin d'éviter la collision avec l'obstacle.

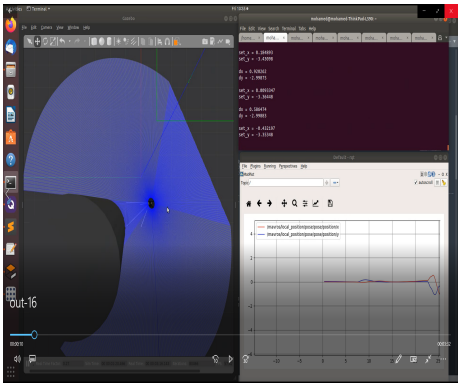
4.6.2 Fixer un but à (4,-6) en présence d'un obstacle

Dans un second temps, on fait décoller le drone à 2.5m par la commande takeoff(), ensuite, on fixe un but au point (4,-6) toute en placant un obstacle au long du chemin à la position (2,-2) :
Voici une séquence de captures d'écran montrant une visualisation de cette simulation :

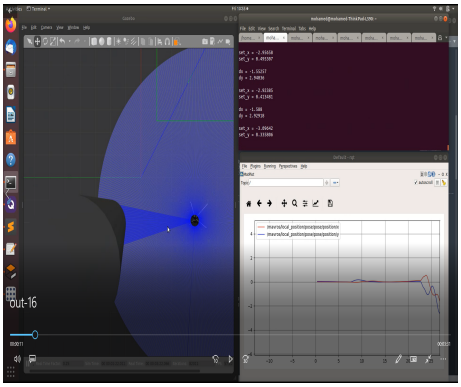


(a) Avant de lancer l'algorithme d'évitement à t=0

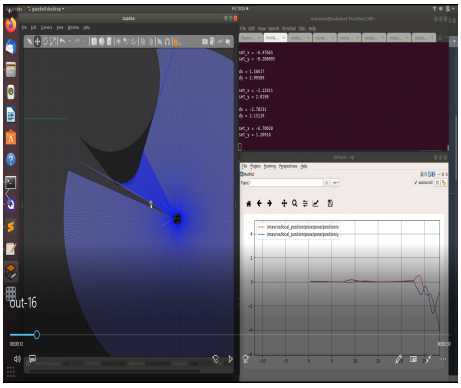
(b) Réaction du drone au moment de lancement de l'algorithme à t=t1



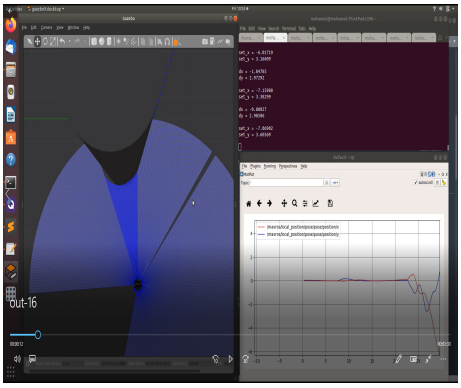
(a) drone à t=t2



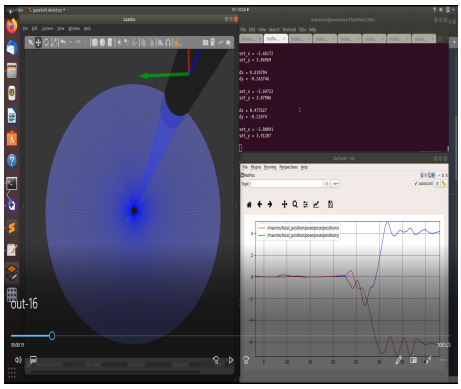
(b) drone à t=t3



(a) drone à t=t4

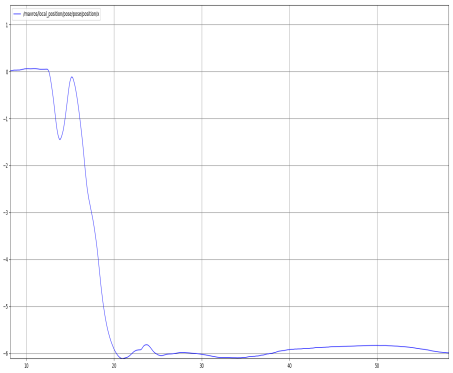


(b) drone à t=t5

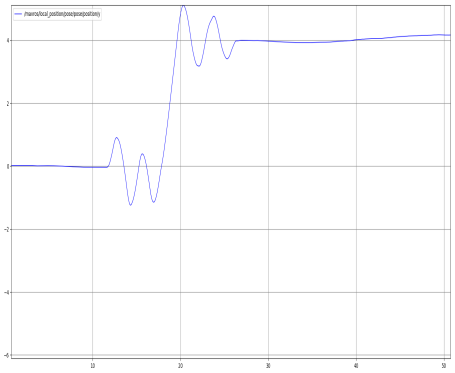


(a) drone, à $t = t_{final}$, arrive au but (4,-6)

Voici les signaux de x et y au long de la simulation



(a) La position x lors de l'évitement



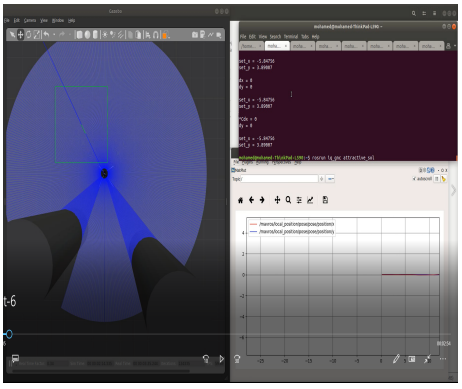
(b) la position y lors de l'évitement

4.6.2.1 Interprétation

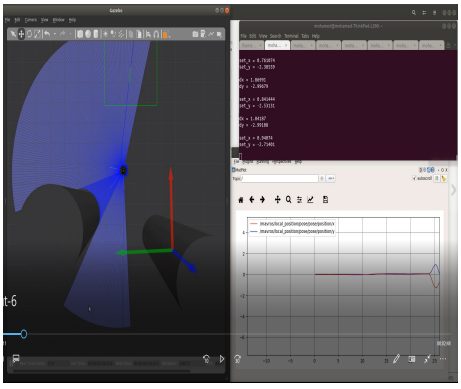
En présence d'un seul obstacle, l'algorithme de l'évitement des champs potentiels assure bien son travail qui consiste à faire une planification de trajectoire en linge. Cependant, il ne suit pas une trajectoire optimale en fonction de la longueur du chemin suivi, due aux forces répulsives exagérantes pouvant affecter le drone lorsqu'il se trouve dans un voisinage indésirable (obstacle).
Une amélioration éventuelle des paramétres de l'algorithme (K_obst, K_att, d_0, ...etc) pourrait minimiser, à un certain degrés, le chemin planifié par celui-ci.

4.6.3 Fixer un but à (4,-6) en présence de deux obstacles

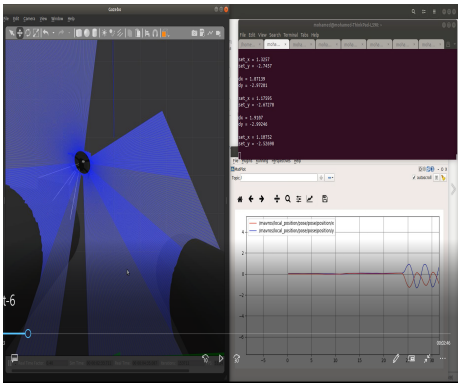
Finalement, on fixe un but au point (4,-6) toute en placant deux obstacle au long du chemin prévu, comme montré dans les figures ci-dessous :



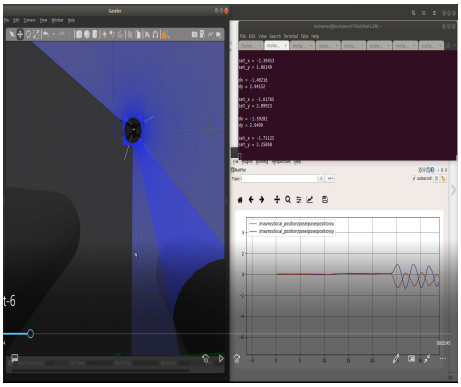
(a) le drone avant de lancer l'algorithme d'évitement à t=0



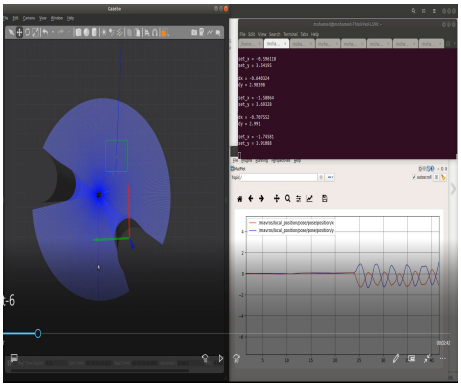
(b) Réaction du drone au moment de lancement de l'algorithme à t=t1



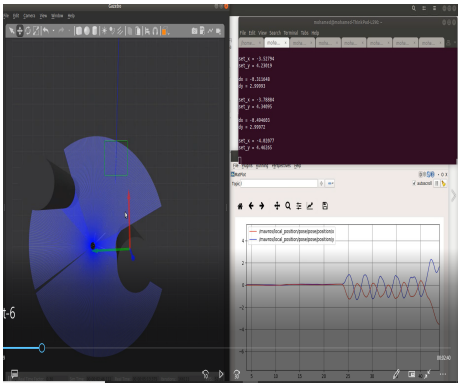
(a) drone à t=t2, étant figé dans un minimum local



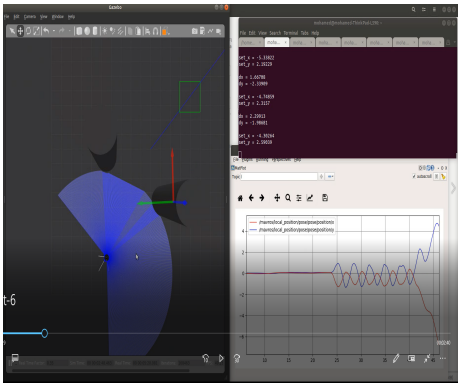
(b) drone à t=t3, restant figé dans un minimum local



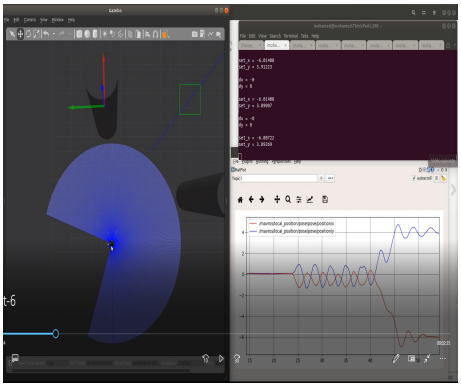
(a) drone à t=t4, aprés avoir déplacé l'un des obstacles afin d'éliminer le minimum local



(b) drone à t=t5, traversant entre deux obstacles



(a) drone à t=t6, allant au but (4,-6)



(b) drone à t=$t_{final}, arrivantaubut(4, -6)$

4.6.3.1 Interprétation

En présence de deux obstacles, La planification par cet algorithme devient moins fiable, en raison du problème persistant de minimum local qui pourrait se produire (cas de la figure 4.19a, 4.19b)
A ce moment là, un algorithme d'échapement aux minima locaux est indisponsable pour assurer un bon fonctionnement au sein des environnements plus complexes (ayant plusieurs obstacles).

Chapitre 5

Description de matériel et logiciel

Dans ce chapitre, nous allons rentrer dans les détails techniques de la réalisation d'un quadri-rotor et la mise en œuvre de l'algorithme SLAM pour la navigation en milieu intérieur et l'évitement d'obstacle. Nous commençons par une présentation de notre drone, ensuite, nous allons présenter le matériel utilisé.

5.1 Présentation du quadri-rotor

Comme montré dans la figure 5.1 le quadri-rotor prend la configuration croix ou X .

Fig. 5.1 : Image démonstrative de la navigation en milieu intérieur du drone.

5.2 Description des matériels et logiciels utilisés

5.2.1 Le matériel

Pour la réalisation de notre quadri-rotor, nous avons utilisé les composants suivants :

1. Un flight contrôleur PIXHAWK4.
2. Un ordinateur compact NVIDIA JETSON NANO.
3. Un châssis en fibre de carbone X500.
4. Quatre moteurs holybro 2216 KV880 et BLHeli S ESC 20A.
5. Carte de répartition de l'alimentation PWM07.
6. Une battrie LiPo
7. 2D LIDAR YDLIDAR X4.
8. Un altimetre ultrason MAXIBOTIX MB 1043.

9. Radio télémétrie 433 MHZ.

10. Radio télécommande FLYSKY FS-i10.

5.2.1.1 Contrôleur de vol PIXHAWK4

Le PIXHAWK4 est un contrôleur de vol, qui combine l'ensemble des composants en unité fonctionnelle complète, il collecte des informations sur l'état du drone à partir des capteurs embarqués(accéléromètre, gyroscope...etc) pour qu'il les traites en fusionnant leurs données. De plus, il fait le control bas niveau en commandant les quatre moteurs via ESCs par des signaux PWM. Le contrôleur de vol est, particulièrement, un microcontrôleur dédiés aux véhicules aériens. Pour notre drone, nous avons utilisé un microcontrôleur adéquat, conçu spécialement pour les systèmes autonomes, appelé PIXHAWK4. Il posséde des fonctionnalités plus avancées que celles d'un microcontrôleur basique.
Les caractéristiques techniques de PIXHAWK4 sont disponible sur le site officielle.

Fig. 5.2 : flight contrôleur PIXHAWK4

5.2.1.2 NVIDIA JETSON NANO

La carte NVIDIA JETSON NANO est un micro ordinateur puissant placé sur le drone. Elle supporte différents systèmes d'exploitation libre GNU-Linux permettant l'utilisation du middleware ROS (Robotic Operating System) qui sera détaillé plus tard en 5.2.2.2. Il en existe plusieurs modèles : TX1, TX2, Jetson Nano... , nous utilisons le modèle JETSON NANO, figure 5.3.

La carte embarque un SoC Quad-Core ARM Cortex-A57 cadencé à 1,43 GHz, et une puce graphique Maxwell avec 128 coeurs CUDA cadencé à 921 MHz. Le tout est associé à 4 Go de mémoire vive en LPDDR4 et 16 Go de stockage eMMC 5.1. Quoique, cette carte ne dispose pas d'un module Wi-Fi, elle est, néanmoins, équipé par :

1. 4 ports USB (USB 3.0 et USB 2.0 Micro-B)

2. 1 emplacement MicroSD
3. 1 port réseau Gigabit
4. 1 sortie HDMI 2.0 (compatible 4K @ 60 images/secondes)
5. 1 sortie vidéo eDP 1.4 (embedded DisplayPort)
6. 1 connecteur pour caméra MIPI CSI-2
7. Plusieurs connecteurs génériques GPIO, I2C, SPI, I2S et UART

Fig. 5.3 : NVIDIA JETSON NANO

Pour l'éventuelle raison, Nous utilisons le module WiFi/Bluetooth EW-7611ULB afin d'accéder à une connexion WIFI dont nous allons avoir besoin pour lui envoyer des commandes et recevoir des données en temps réel à base d'un ordinateur portable interfacé par le système d'exploitation Linux, en utilisant, à savoir, le protocole de communication SSH.

5.2.1.3 Le Châssis

C'est la plateforme sur laquelle se disposent tous les composants du quadri-rotor. Le choix de ce dernier influe grandement les performances, la robustesse et la stabilité du système.

Pour avoir de bonnes caractéristiques de vol, il faut bien choisir le châssis. Ce dernier doit être symétrique par rapport aux axes portants des moteurs, et aussi par rapport à son centre de gravité pour éviter le déséquilibre à cause des moments inertiels. Il doit y avoir un contre promis entre le poids total et les forces de poussé exercées par les moteurs pour qu'il soit optimale. Dans notre cas, on a choisi un châssis en fibre de carbone qui prend la configuration 'X' avec une basse au milieu carré est présenté dans la figure 5.4.

Fig. 5.4 : Le châssis de notre quadri-rotor.

5.2.1.4 Moteur sans balais (BLDC)

Le moteur sans balais (BLDC) se comporte comme un moteur à courant continu traditionnel. Il présente des caractéristiques semblables à celles des moteurs à courant continu et alternatif, sauf qu'il est avantageux par certaines caractéristiques en le rendant pratique pour l'utilisation dans les véhicules aériens :

1. Une forte dynamique de vitesse et d'accélération sans l'usure mécanique des moteurs courant continus La commutation électronique se substituant à la commutation mécanique.

2. Il est contrôlable par rapport à sa vitesse de rotation, par l'intermédiaire d'un signal PWM .

3. Ils entrent dans la catégorie des moteurs synchrones, ce qui signifie que le champ magnétique créé par le stator et celui généré par le rotor tournent à la même fréquence.

Le choix d'un moteur BLDC doit être fait en prenant en compte l'usage que l'on en fera et les différentes caractéristiques qu'il possède, tel que : Kv : la constante de vitesse par volt, elle est indiquée en RPM (rotations par minute) par volt, à vide. Cela signifie donc qu'un moteur de 1000 Kv tourne à 1000 tours/min à 1 Volt, à vide. Dans notre cas, nous ne faisons pas tourner le moteur à vide mais avec une hélice. Pour maintenir le nombre de tours par minute il faut compenser la résistance aérodynamique de l'hélice, cela nécessite plus de couple ou alors plus de puissance. Généralement, plus le moteur tourne vite plus son couple est faible, et inversement. Pour cette raison, un moteur avec un Kv faible pourra donc supporter de plus grandes hélices qu'un moteur à Kv élevé.

Dans notre projet, on a utilisé un moteur BLDC HOLYBRO 2216 KV880, figure 4.5, ayant entre 2280 Kv avec une masse de 39g et pour une hauteur de 22 mm et de diamatre 16mm, comme il montre la figure 5.5.

Fig. 5.5 : Moteur holybro 2216 KV 880

5.2.1.5 Variateur de vitesse (ESC)

Le contrôle d'un moteur BLDC s'effectue nécessairement grâce à un circuit électronique auxiliaire. En effet, c'est le circuit de commande qui va exciter de façon successive les différentes bobines du stator. Pour créer un champ magnétique tournant, le circuit de commande devra exciter les bobines dans un ordre approprié (séquence de commutations) et cela au moment opportun. C'est l'ESC (appelé aussi variateur électronique) qui délivre le voltage et l'ampérage adéquats au moteur. Son rôle est de réguler la tension pour faire varier la vitesse de rotation, mais aussi de laisser passer l'ampérage dont il aura besoin.

Dans le cas d'un quadri-rotor, nous avons besoin d'utiliser quatre ESC, un pour chacun des quatre moteurs BLDC. Comme chaque ESC est alimenté par l'alimentation principale, le connecteur unique de cette dernière doit en quelque sorte être réparti entre les quatre ESC. Pour ce faire, une carte de répartition de l'alimentation est utilisée.Dans notre travaille, nous utilisons quatre ESC 20A BLHeli S, figure 4.7.

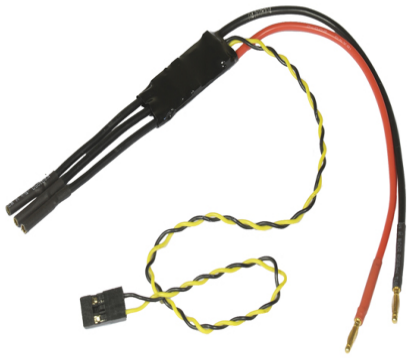


FIG. 5.6 : ESC 20A BLHeli S

5.2.1.6 Les hélices

Les hélices sont les extensions de la voile, qui réalise une propulsion vélique : en tournant, l'hélice mobilise la force de portance de l'air mis en mouvement et la transfère au corps sur lequel elle est attachée. Cette force de portance est une composante de la réaction du fluide, qui agit perpendiculairement au plan tangent des pales de l'hélice. Elles doivent être adaptées à la taille du quadri-rotor et aussi aux moteurs qu'on a choisis. Un quadri-rotor utilise deux hélices qui tournent dans le sens horaire (CW) et deux autres hélices qui tournent dans le sens anti-horaire (CCW), afin d'annuler les moments pouvant être générés selon l'axe z. En effet, lorsque les quatre hélices tournent en un seul sens, cela génère des moments selon z à cause des interférences aérodynamiques de chaque hélice, et par conséquent, une rotation du drone autour de z va notamment s'engendrer dans le même sens que la rotation des hélices, ce qui déstabilisera le drone éventuellement.

Pour notre projet, nous avons choisi des hélices de diamètre égale à 12.7 cm, figure 4.8.

FIG. 5.7 : les hélices

5.2.1.7 Carte répartition d'alimentation PM07

La carte de gestion de l'alimentation (PM07) sert à la fois de module d'alimentation et de carte de distribution d'alimentation. En plus de fournir une alimentation régulée au Pixhawk 4 et aux ESC, il envoie des informations au pilote automatique sur la tension et le courant de la batterie fournis au contrôleur de vol et aux moteurs.

Caractéristiques :

- Courant PCB : sorties totales 120A (MAX)
- Courant de sortie UBEC 5v : 3A
- Tension d'entrée UBEC : 7 51v (LiPo 2 12s)
- Dimensions : 68.50.8 mm
- Trous de montage : 45 x 45 mm
- Poids : 36g

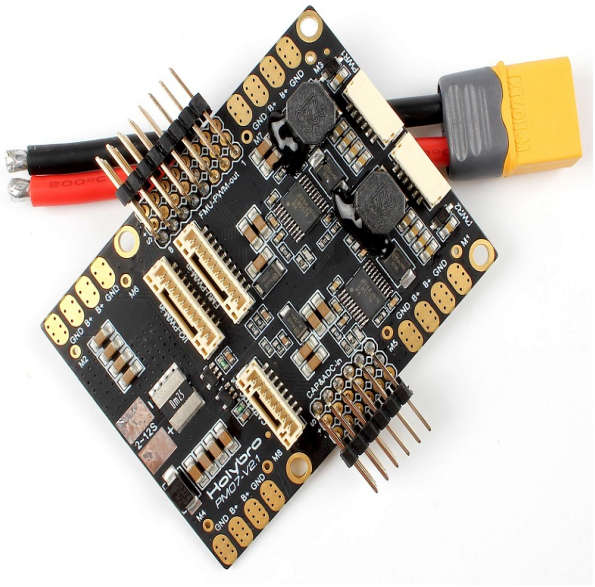


FIG. 5.8 : La carte répartition d'alimentation

5.2.1.8 Une batterie au lithium polymère

Évidemment, les drones ont besoin de batteries embarquées pour alimenter leur fonctionnement. Les batteries au lithium polymère (LiPo) sont parmi les types de batteries les plus couramment utilisés pour les drones car elles offrent l'avantage d'une densité d'énergie élevée par rapport à leur taille et leur poids, avec une tension par cellule plus élevée, afin qu'elles puissent alimenter les systèmes attachés au drone, avec moins de cellules que les autres piles rechargeables. Ils se déchargent également plus lentement que les autres types, de sorte qu'ils tiendront une charge plus longtemps lorsqu'ils ne sont pas utilisés. Cependant, s'ils ne sont pas chargés ou utilisés correctement, ils ne peuvent pas fournir des performances optimales pendant longtemps et peuvent même commencer à fumer et à prendre feu. Nous utilisons la batterie Turnigy 5000mah 3s Lipo 11.1v 30-40c. C'est une batterie Lipo ayant 3 cellules et fournissant jusqu'à 11.1V et un amperage de 5A. Le poids total du drone et estimé à 1.8KG. En utilisant cette batterie, le drone pourra voler jusqu'à 10 min avant que la batterie se décharge moins que le seuil de fonctionnement qu'on a défini parmi les conditions de pré-vol ; ce seuil est fixé à 10.2V car, après certaines expériences, on a appris qu'au-delà de ce volage, le drone ne fonctionnerait plus normalement.

FIG. 5.9 : Battrie LIPO

5.2.1.9 YDLIDAR X4

Le terme LiDAR est un acronyme anglais pour « Light Detection And Ranging » signifiant en français « détection et estimation de la distance par la lumière ». YDILIDAR X4 figure 4.9 est un 360-degree deux-démentielle laser. Basée sur le principe de triangulation, il est équipé par une partie optique, électrique et un algorithme désigné pour obtenir avec précision une mesure à haute fréquence. La structure mécanique tourne 360 dégrées en permanant pour donner la sortie de l'information sous forme de nuage de points pendant le scan. Ce Lidar 2D scanne l'environnement en un plan 2D afin d'obtenir les coordonnées XY de chaque point scanné. Le modèle utilisé pour extraire le nuage des points d'un scan à un instant t est le suivant :

$$\begin{cases} X = dcos(\phi) \\ Y = dsin(\phi) \end{cases} \tag{5.1}$$

d : la distance vers l'obstacle mesurée par le lidar. ϕ : l'angle que fait le scanneur avec son propre axe OX. X, Y : les coordonnées du point scanné par rapport à la base locale du lidar.
Note : Ces coordonnées sont liées à la base locale du lidar. Pour passer à la base globale, une transformation doit être effectuée entre la référence globale et celle associée au lidar.
Caractéristiques :

- Balayage à 360 degrés
- Haute précision, performances stables
- Large plage de mesure
- Forte résistance aux interférences lumineuses ambiantes
- Faible consommation d'énergie, petite taille, performances stables et longue durée de vie
- Sécurité oculaire de classe I
- La vitesse du moteur est réglable, la vitesse proposée est de 6 jusqu'au 12Hz.
- Gamme à grande vitesse, fréquence allant jusqu'à 5KHz.

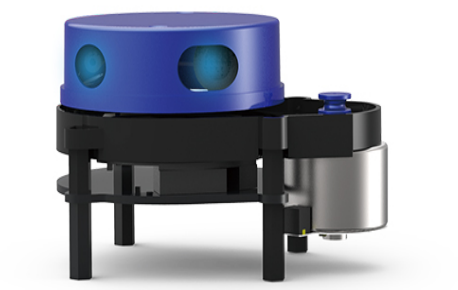


Fig. 5.10 : YDLIDAR X4

5.2.1.10 Un altimetre ultrason MAXIBOTIX MB 1043

Le sonar Maxbotix MB 1043 est un capteur pour la mesure d'altitude. En effet, il utilise un transducteur pour envoyer et recevoir des impulsions ultrasonores qui relaient des informations sur la proximité du sol (étant orienté vers le sol). Il est conçu pour une utilisation en milieu intérieur à une courte portée (jusqu'à 7 m).

FIG. 5.11 : Altimetre

5.2.1.11 Radio télécommande FLYSKY FS-i10

Un système de radiocommande (RC) est nécéssaire, si vous souhaitez controlez manuellement votre véhicule a partir d'un émetteur, Un système RC possède une télécommande au sol qui est utilisée par l'opérateur pour commander le véhicule. La télécommande a des commandes physiques qui peuvent être utilisées pour spécifier le mouvement du véhicule (par exemple, vitesse, direction, accélérateur, lacet, tangage, roulis, etc.), et pour activer les modes de vol du pilote automatique (par exemple, décollage, atterrissage, retour à terre, mission, etc.). Sur les systèmes RC compatibles avec la télémétrie, la télécommande peut également recevoir et afficher des informations du véhicule (par exemple, niveau de batterie, mode de vol), La télécommande contient un module radio qui est lié et communique avec un module radio (compatible) sur le drone. L'unité embarquée sur véhicule est connectée au contrôleur de vol. Le contrôleur de vol détermine comment interpréter les commandes en fonction du mode de vol actuel du pilote automatique et de l'état du véhicule, et entraîne les moteurs et les actionneurs du véhicule de manière appropriée. Le système complet ce compose en deux partie :

1. Transmetteur (TX) : c'est la télécommande ou on commande le drone.

2. Receveur (RX) : ca consiste d'une partie de réception des commandes du transmetteur via les ondes radio, ce dernier est attaché au drone et connecté avec le contrôleur de vol.

FIG. 5.12 : Radio télécommande.

5.2.2 Les logiciels

Il existe une grande variété de logiciels pour le contrôle des robots autonomes, chacun d'eux a ses propres avantages et inconvénients. Notre travail est principalement effectué sous l'environnement ROS, l'interface Mission Planner pour la gestion du notre contrôleur de vol et Gazeboo pour les simulations dynamiques en 3D.
Tout d'abord l'environnement ROS inclut des packages qui executent de nombreuses fonctions, Dèjà disponibles et prêtes à l'utilisation. Pour cette raison, on en a profité en exploitant des algorithmes dèjà existants tel que Hector_Slam pour faire du slam et Mavros pour accéder à Pixhawk depuis ROS... etc.
De plus, un grand avantage qui nous a vraiment aidé ; le firmeware PX4 5.2.2.5, qui a été installé dans notre contrôleur de vol (PIXHAWK4). il est disponible en open-source et accessible pou toute modofication éventuelle.

5.2.2.1 Mission Planner

Le contrôleur de vol PIXHAWK4 de notre drone est compatible avec deux firmware PX4 FMU et ArduPilot. Chaque firmware nécessite un software spécifique (QGroundcontrol et Mission Planner respectivement). Dans notre projet on a utilisé le logiciel Qground-Control pour travailler avec le firmware PX4 version 12.1. Chaque firmware possède des avantages et des inconvénients et les deux sont disponibles en open-source.

Moyennant un radio télémétrie, ce logiciel, qui soit installé dans un laptop, communique avec le microcontrôleur Pixhawk4 à travers le protocole Mavlink **mavlink**. On pourra alors envoyer ou recevoir des messages Mavlink à partir d'une station de contrôle au sol (laptop) ; en utilisant ce logiciel, on pourra accéder à l'état du drone à tout moment, calibrer les composants munis au flight-contrôleur ou même ajuster les paramètres lié au firmware utilisé tel que les coefficients de régulateur PID, l'observateur utilisé, la source de localisation… etc.

On pourra aussi dessiner les différents signaux de mesures en temps réels comme l'altitude, l'orientation, la position, la vitesse...etc, ce qui nous facilite notamment l'interprétation et la manipulation des données afin de faire des modifications là où il faut.

5.2.2.2 ROS

ROS (Robotic operating system) est un open-source Framework pour la programmation des robots construit dans l'environnement Linux. Il comprit plusieurs outils, packages et bibliothèques pouvant simplifier les taches du contrôle d'un robot (ou plusieurs) ainsi que l'interconnexion entre ses différentes parties. Notre travail est principalement développé sous l'environnement ROS. La distribution ROS Melodic a été utilisée pour établir un network avec Linux Ubuntu 18.04 LTS sur la carte NVIDIA JETSON NANO.

5.2.2.3 Gazeboo

Gazebo est un simulateur 3D, cinématique, dynamique et multi-robot permettant la simulation des robots mobiles, des robots industriels ou même des robots humanoïdes, dans des environnements complexes. Le simulateur gazebo est un outil de simulation très pratique en robotique ; étant capable de visualiser la dynamique des robots qui soient commandés par des algorithmes qui s'exécutent parallèlement. En outre, vu sa compatibilité avec ROS, on pourra alors tester nos codes et les valider en simulation avant de les implémenter en pratique.

5.2.2.4 MAVROS

MAVROS est un package ROS, disponible en open-source sur Github, qui a pour but de lier le nœud de Pixhawk4 avec l'environnement ROS. En effet, il représente une interface entre le système Mavlink et le ROS master, i.e. son rôle consiste principalement transformer des messages Mavlink (génerés par le protocole Mavlink) en messages ROS (tous les topics qui commence par /mavros) et vice versa. Par conséquent, l'état du drone, qui a été fournie originalement par le protocole Mavlink, sera disponible sur ROS, ainsi, il pourra être utilisé par d'autres algorithmes (d'autres nœuds sur ROS) tel que le SLAM, l'évitement d'obstacle... etc.

5.2.2.5 PX4-Autopilot

PX4 est un software de contrôle de vol open source destiné aux drones et autres véhicules sans pilote. Le projet fournit un ensemble flexible d'outils permettant aux développeurs de drones de partager des technologies afin de créer des solutions sur mesure pour les applications des drones. PX4 propose une norme pour fournir un support matériel de drone et une pile logicielle, permettant à un écosystème de créer et de maintenir du matériel et des logiciels de manière évolutive.
PX4 fait partie de Dronecode ; une organisation sans but lucratif administrée par la Foundation Linux pour favoriser l'utilisation des logiciels open source sur des véhicules aériens. Dronecode supporte également QGroundControl, MAVLink et le SDK.

5.2.2.6 HECTOR SLAM

Hector SLAM est un open source package, disponible sur github. Etant un package de ROS. Il faut le confugirer selon nos souhait. *hector_mapping* est un noeud du Hector_SLAM, ce dernie va souscrire à un topic dans lequel sont publiées nos données laser afin de publier la position estimé.

Deux topic sont nécessaires à la construction de la carte de l'environnement map_metadata et map.hectormapping publie donc dans ces deux topics afin de construire la carte.

slam_out_pose est un topic dans laquel hector_mapping publie des estimations de la pose du robot.

5.2.2.7 Le package tf

le package tf est un open-source de ROS. Il est nécessaire pour l'adaptation des différents repères dans notre robot (repère LIDAR, repère odometrie...etc). A travers ce dernier on effectue des transformations statiques ou dynamiques pour aller d'une base à une autre. La figure 5.13 illustre un exemple d'un arbre tf :

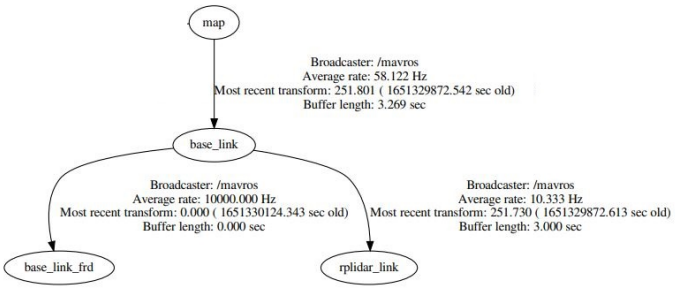


Fig. 5.13 : L'arbre tf

Tel que :

- map : la base terrestre.
- base_link : la base liée au drone.
- rplidar_link : la base liée au lidar 2D.

5.3 Description de l'environnement de simulation :

En simulation, on a installé le firmware PX4-Autopilot, disponible sur Github, dans un système d'exploitation Linux Ubuntu 18.04 LTS muni de la distribution ROS Melodic. Ce firmware a été utilisé pour simuler le flight-contrôleur PX4 sur l'environnement de simulation ROS/Gazebo. On a préparé un environnement de simulation ayant les mêmes fonctionnalités que notre drone réel ; i.e. un flight-contrôleur Pixhawk4 contrôlant quatre moteurs BLDC et disposé sur une plateforme d'un quadri-rotor avec un lidar 2D (RPlidar) rattaché juste en dessus et un altimètre orienté en bas, comme montré dans la figure 5.14 :

D'autre part, pour que l'algorithme hector_slam puisse faire du SLAM en exploitant le scan de lidar, on a établi un environnement fermé qui se constitue de 4 murs entourans le drone des quatre côtés, comme montré dans la figure 5.15 :

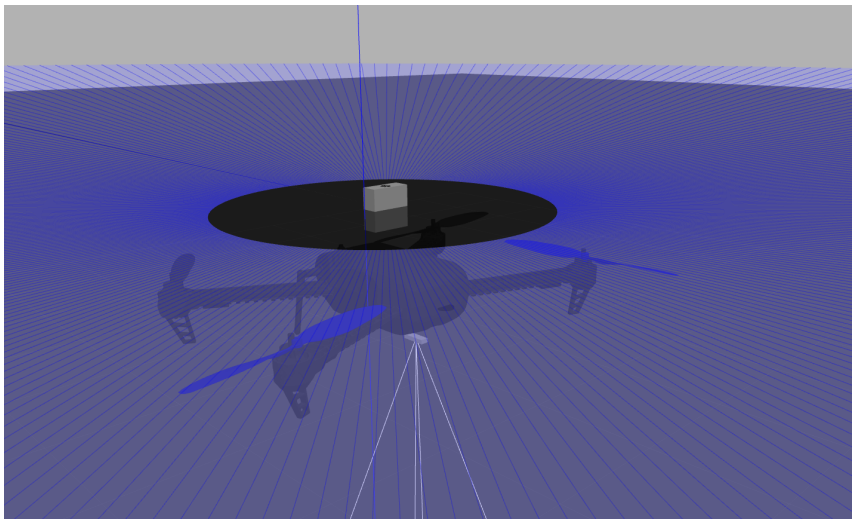


FIG. 5.14 : Plateforme du drone de simulation sur Gazebo

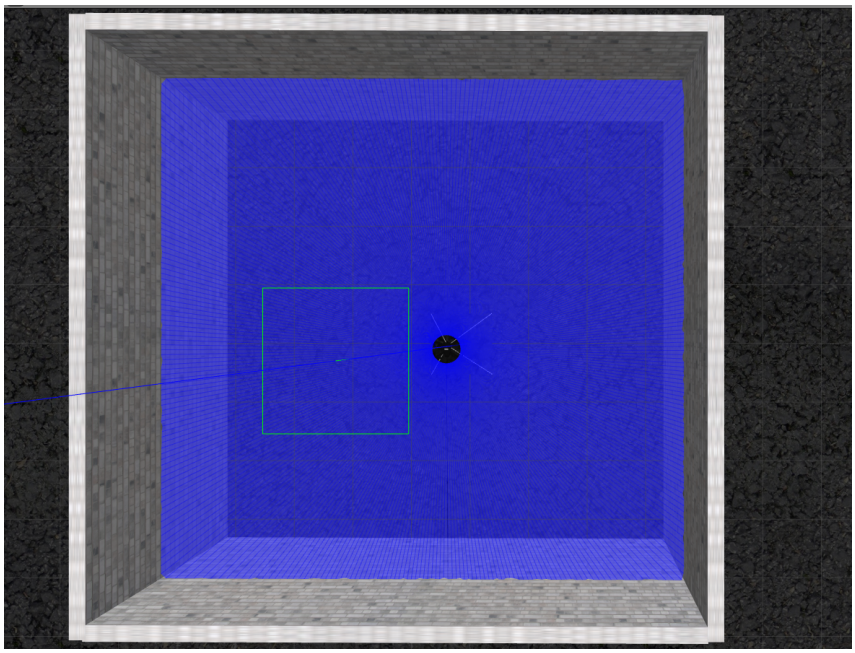


FIG. 5.15 : Drone de simulation sur Gazebo en milieu interieur

Le firmware PX4-Autopilot est considéré comme un nœud sur ROS grâce au interfaçage du package MAVROS. Ces deux nœuds sont chargés de simuler, sur Gazebo, toutes les parties hardware du système, le contrôle bas-niveau du drone ainsi que l'acquisition des données sortantes des capteurs munis au PX4 (IMU, Altimètre, PX4flow, GPS). De plus, Le simulateur Gazebo publie le scan du lidar, après tout balayage de l'environnement, à travers le topic /scan, par conséquent, ce message sera disponible sur ROS et pourra être exploité par d'autres nœuds.

L'algorithme Hector_slam s'exécute en parallèle sous ROS. Il reçoit le message /scan à partir du Gazebo ainsi que les données de l'IMU à partir du firmware PX4-Autopilot.

Ensuite, il construit une carte de l'environnement de simulation et estime une localisation 2D dans les milieux intérieurs ou le scan de lidar soit réflexible en présence des objets qui doive être inclus dans sa plage de mesure (10 mètre), dans ce cas, le nœud Hector_slam peut calculer la position xy du lidar. D'autre part, vu que le lidar est fixe par rapport à la plateforme du drone, la localisation de celui-ci en 2D peut être considérée comme une localisation du drone. Hector_slam publie alors cette localisation, à travers le topic /slam_out_pose, sous forme d'une position 2D vers le nœud MAVROS pour qu'il puisse se localiser en 2D dans les milieux intérieurs. Ainsi, l'altitude du drone (coordonnée selon z) sera mesurée par l'altimètre, celui-ci fait partie du nœud PX4-Autopilot. Par conséquent, le drone de simulation pourra se localiser en 3D sans avoir besoin des données issues de GPS.
Finalement, le nœud Publisher a pour but de communiquer le topic /slam_out_pose issu du SLAM vers le nœud de MAVROS.
La figure 5.16 montre le graph des noeuds et des topics du système ROS expliqué au dessus :

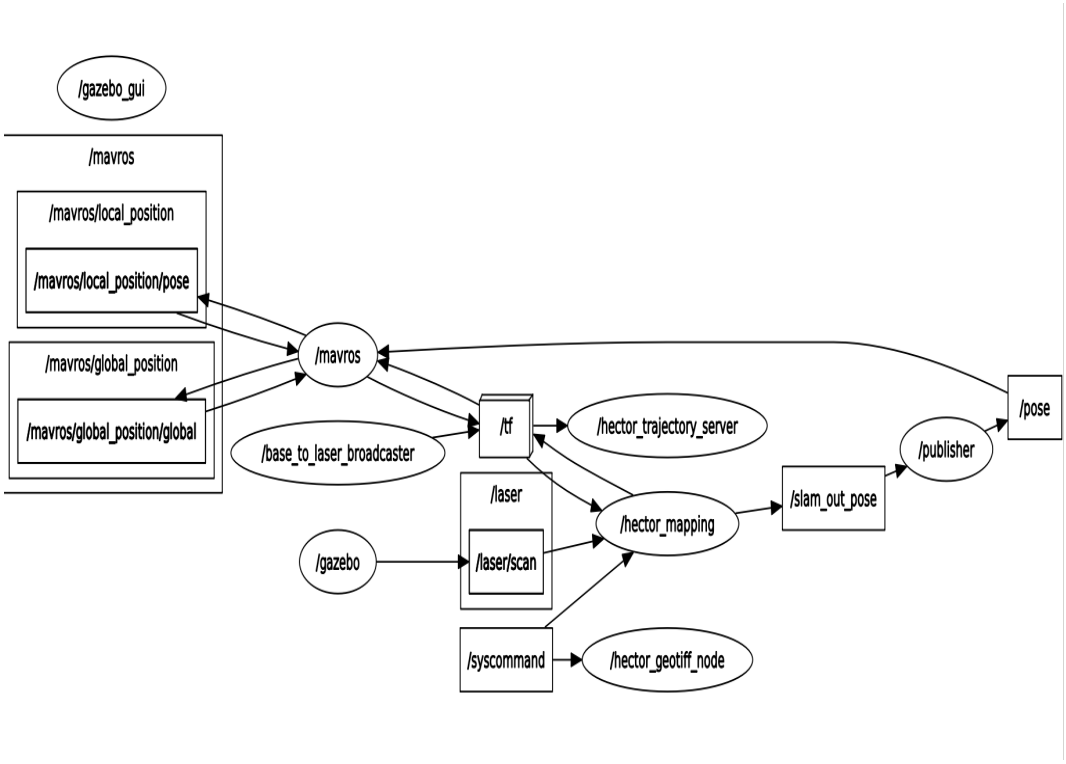


Fig. 5.16 : Graph des topics obtenu par la commande rqt_graph décrivant l'intéraction du système avec Hector SLAM, pour estimer la position x-y

Le code d'évitement d'obstacle qu'on a développé (5.17) représente aussi un autre nœud, il reçoit le message /scan, calcule des commandes haut-niveau et les envoie vers le firmware PX4-Autopilot sous forme d'un topic /mavros/setpoint/position. Ces commandes représentent des positions désirées à injecter dans le modèle de PX4 pour qu'il les suive en mettant en exécution les algorithmes de contrôle au niveau de Pixhawk4.

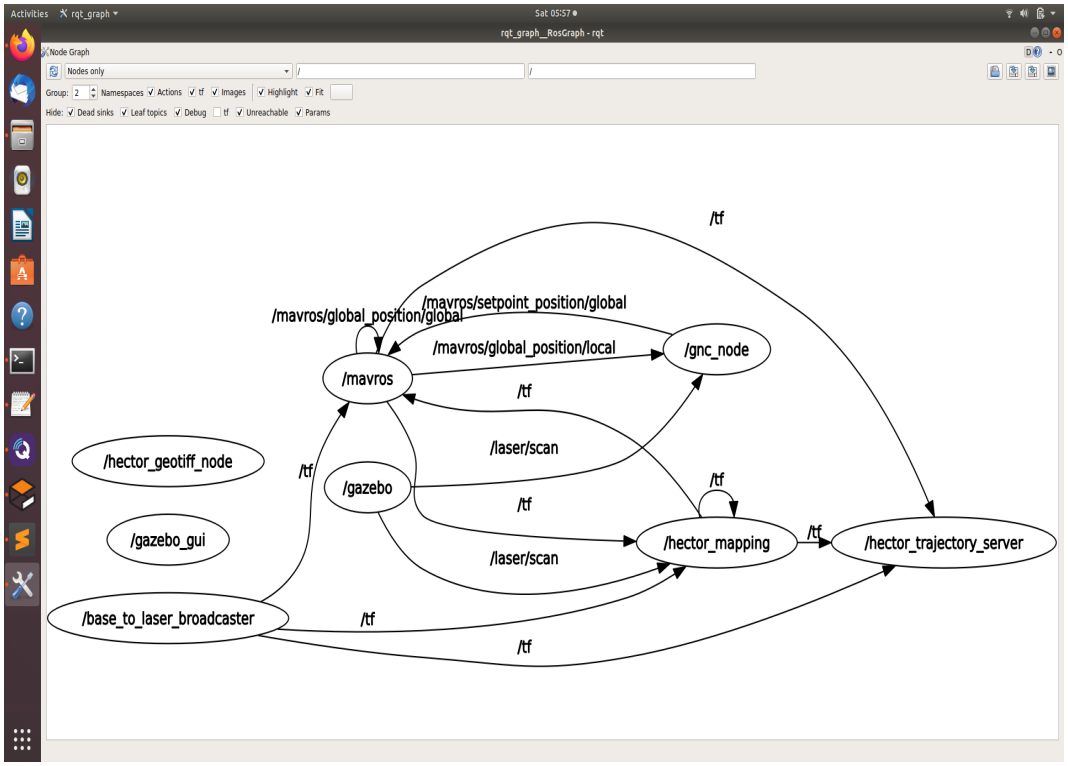


FIG. 5.17 : Graph des noeuds obtenu par la commande rqt_graph décrivant l'intéraction du système avec le noeud $'gnc_node'$, pour effectuer l'évitement d'obstacle

Remarque :

L'algorithme d'évitement présente une sorte de contrôle haut-niveau, car il calcule des commandes abstraites et les envoie comme des positions de consignes vers le système du contrôle bas-niveau.

On execute l'algorithme (7.2) qui représente un autre noeud dans le reseau des noeuds ROS dèjà existants (voir la figure 5.18). Ce noeud communique avec MAVROS en lui envoyant des setpoints de position et/ou d'orientation.

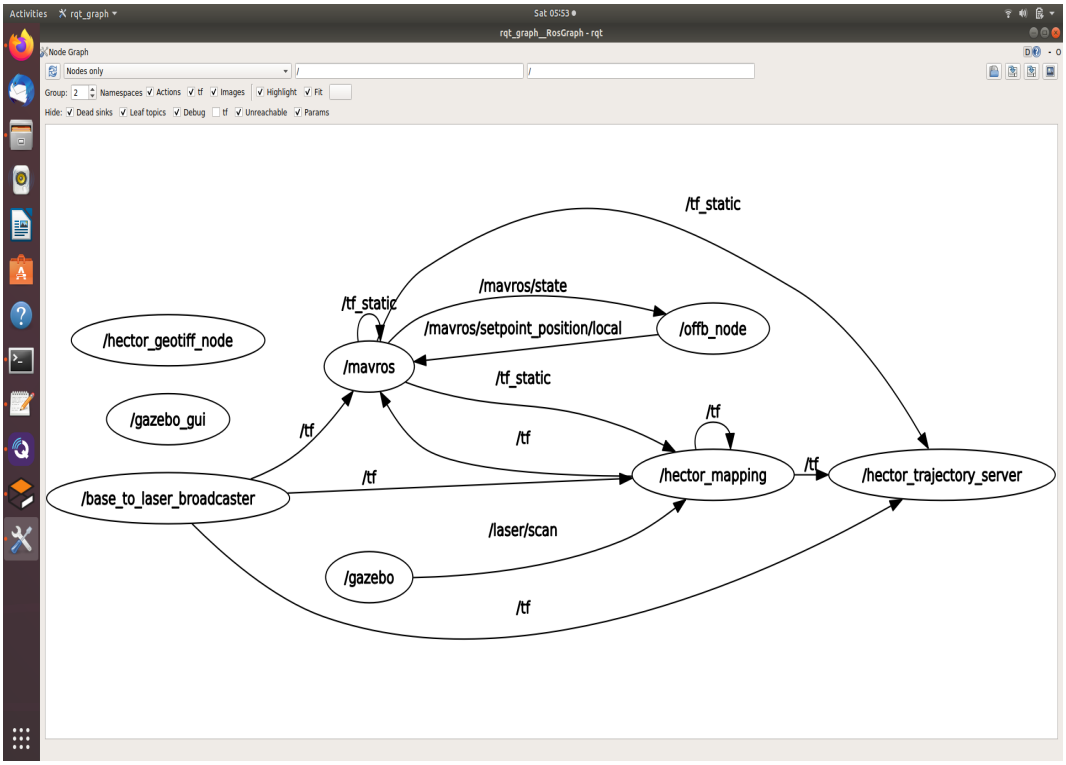


FIG. 5.18 : Graph des noeuds obtenu par la commande rqt_graph décrivant l'intéraction du système avec le noeud $'offb'$, pour lui donner des consignes à suivre

5.4 Description de l'implémentation pratique

Ce projet consiste à implémenter une navigation autonome d'un drone quadri-rotor en milieu intérieur où il n y a pas accès aux signaux de GPS. Le drone est équipé par un flight contrôleur Pixhawk4 qui assure le contrôle bas niveau des moteurs ainsi que l'acquisition des données des capteurs embarqués ou connectés à celui-ci. De plus, ce microcontrôleur dispose de plusieurs interfaces et protocoles de communication [interfaces] pour relier les différentes parties du système, comme montré dans la figure 5.19 : Le drone est attaché aussi par un micro-ordinateur appelé Nvidia Jetson Nano, qui s'interface par le système d'exploitation Linux/Ubuntu18. On y a installé le framework ROS Melodic afin de faciliter la coordination entre les différents composants du système. Cette carte communique avec Pixhawk par le protocole UART en lui donnant des ordres (tel que la position de consigne, vitesse de consigne…) et recevoir des données (tel que la position actuelle, la vitesse actuelle…), au fait, c'est une liaison Maitre/esclave. Au-dessus de la plateforme on rattache un LIDAR 2D (RPLidar X4) et on le communique avec NVIDIA via le protocole USB. Le lidar fait le balayage de l'environnement à 360° et envoie son scan à la carte NVIDIA.

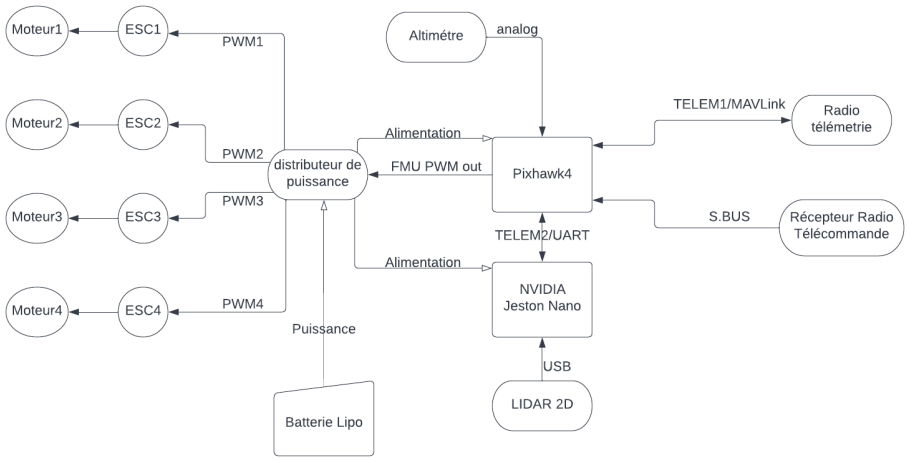


Fig. 5.19 : Schéma descriptif du système

D'autre part, afin d'assurer une indépendance complète de toute connexion par fil, on a connecté la carte NVIDIA avec un laptop au sol via une communication sans fils ; à savoir SSH/WiFi pour qu'on puisse envoyer des commandes vers NVIDIA à partir d'un laptop. Par conséquent, notre système est complétement indépendant de toute communication directe et il attend pour qu'on lui envoie des commandes, via SSH, à partir d'une station au sol. En effet, une bréve illustration du système de communication est donnée dans la figure 5.20

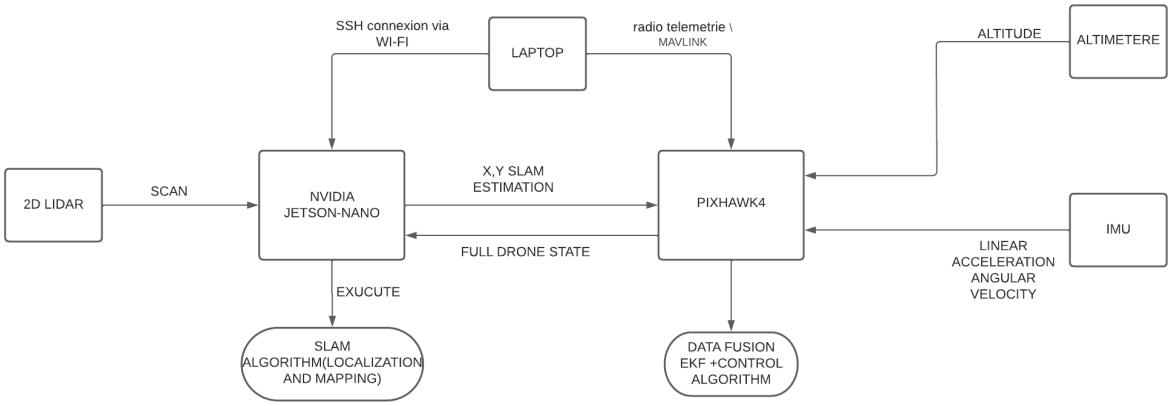


Fig. 5.20 : Schéma général du système

5.4.1 Les étapes d'implémentation

Après avoir fait toutes les interconnexions décrites dans la figure 5.19, On est prêt à démarrer le système en effectuant les étapes suivantes :

1. On lance, sur NVIDIA, le nœud RPLidar X4 qui fait partie du package RPlidar_ros_driver, disponible sur Github. Ce nœud est responsable à faire tourner le lidar et acquérir les données qu'il mesure sous forme d'un topic ROS (*/scan*).

2. On lance, ainsi, le nœud de *Hector_slam* qui reçoit le topic /scan, publié par le nœud RPlidar X4 et qui exécute, ensuite, l'algorithme SLAM pour donner une estimation de la localisation 2D qu'on va publier sous forme d'un topic ROS (*/slam_out_pose*).

3. Ensuite, on lance le nœud de mavros, qui communique avec Pixhawk4 en transformant les messages MAVLink en messages ROS. Ce nœud reçoit, par ailleurs, le topic */slam_out_pose* et le communique, ainsi, à pixhawk4 comme étant une source de localisation x-y.

4. Finalement, on lance le nœud d'évitement d'obstacle, qui utilise le topic /scan, exécute l'algorithme d'évitement et envoie des commandes vers le nœud mavros sous forme d'un topic ROS (mavros/setpoint/position). Le Pixhawk4, par conséquent, suives ces commandes en allant vers les positions désirées toutes en mettant en exécution les algorithmes de contrôle bas-niveau qui sont lié à la firmware de PX4.

5.4.2 Problème téchnique

Le système décrit ci-dessus était cencé naviguer d'une manière autonome en milieu interieur à l'aide d'un altimètre, un lidar 2D et en implémantant un algorithme SLAM pour bien se localiser en intérieur sans avoir besoin de GPS. Néanmoins, aprés plusieur essaies pour faire fonctionner un système de localisation basé seulement sur le SLAM ainsi que l'altimètre MAXIBOTIX MB 1043 (5.11), on s'est rendu compte qu'il y a un problème de syncronisation qui persiste entre les capteurs utilisés. En effet, l'altimètre communique sa mesure d'altitude à une fréquence maximale de 2Hz, alors que le système SLAM estime la position x y à 10Hz. Dans ces conditions, la régulation simultanée de la position x y z serait impossible. D'ailleur, si on fixe l'altitude à un hauteur donné, le drone peut maintenir sa position x y en interieur (comme montré dans la figure 5.1). Par contre, il peut pas faire une mission entièrement autonome car ça nécessite une régulation simultanée de x, y et z. Pour cette raison, on a pas eu le temps ni le matériel adéquat pour implémenter l'algorithme d'évitement d'obstacle (5.17) en pratique, or, on l'a implémenté quand même **en simulation** pour voir à quel point il est robuste (voir la section 4.4).

Chapitre 6

Conclusion

Les drones quadri-rotors sont trés populaires dans le domaine des véhicules aeriens grâce à leur simplicité de fabrication. Cependant, ce type des drones présente une dynamique non linéaire, ayant 6 degrés de liberté, ce qui pose un grand défi par rapport à leur commande et leur observation d'ètat.

Nous avons commencé notre travail par une brève introduction où on présente la problématique qui nous a motivé à travailler dessus. Il s'agit d'une navigation en milieux intérieurs d'un robot aerien autonome. Ainsi, on a proposé des techniques permettant de naviguer notre drone en milieu intérieur. Nous avons également modélisé la dynamique du quadri-rotor, ensuite, nous avons synthétisé un régulateur PD sur Matlab pour simuler le contrôle du quadri-rotor afin d'avoir une idée sur le contrôle bas niveau assuré par notre contrôleur de vol (Pixhawk4).

De plus, nous avons présenté la problématique générale du SLAM ainsi que le système Hector SLAM et les techniques qu'il utilise pour faire la cartographie et la localisation. Ce système SLAM fournit une localisation 2D en milieu intérieur.

Apres avoir assuré la localisation du drone en milieu intérieur, on a développé un algorithme d'évitement d'obstacle basé sur la théorie des champs potentiels artificiels. Il a pour but de planifier la trajectoire du drone en allant vers une position desirées prédéfinit dans le plan x-y tout en evitatnt les obstacles au long du chemin suivi.

Apres cela, nous avons fait une description générale du matériels et software utilisés dans notre projet. Puis, on a discuté les résultats des simulations qu'on a effectué sur ROS/Gazebo.

D'autre part, Avant d'entamer la partie pratique on faisait des testes de simulation, Sachant que les configurations software et hardware de la simulation sont identiques à celles qu'on a utilisé en pratique.

Maintenant, on présente les points essentiels que nous avons conclus à l'issue de notre travail :

- L'utilisation de ROS est prouvée d'être trés efficace et utile en robotique, en vu de son adaptabilité avec de nombreux systèmes robotiques ainsi que les ressources software qui y sont disponibles. En outre, il représente une plateforme qui facilite énormement la liaison de plusieurs entités hardware et software au sein du même système, tel que les capteurs, les actionneurs, les micro controleurs, les algorithmes de contrôle haut niveau, les systèmes de localisation...etc.
  De plus, il offre un environnement de simulation assez réaliste étant compatible avec des simulateurs puissants tel que Gazebo, Rviz et beaucoup d'autres. Cet avantage offre un outil pratique pour tester des algorithmes et des configurations de la structure du systèmes avant de les implémenter en pratique, car parfois on risque d'endommager notre système réel dans le cas où les algorithmes et les configurations à implementer ne sont pas valides.
  A titre d'exemple, on avait la possibilité de travailler avec le software dronekit ; il représente un autre outil de développement permettant de contrôler en haut niveau le PIXHAWK4 par des codes python, or, ce dernier n'offre pas un background pouvant relier plusieurs sous systèmes d'une manière efficace, donc, il est limité en terme d'applications, d'où l'importance critique de ROS et son avantage brillant par rapport aux autres softwares.

- Les robots mobiles sont souvent des systèmes trés complexes qui se décomposent en plusieurs sous systèmes ayant differentes caracteristiques. Pour que ces derniers puissent fonctionner en coordination, ils necessitent contuniement un réseau de communication efficace reliant ses composants ainsi qu'une compatibilité de materiels utilisés. Cela mets en avant les provocations du problème téchnique qui nous a empêché d'implémenter une mission autonome en milieu intérieur basé sur le système de localisation proposé ; (un système SLAM qui estime la position x-y en 2D et un altimètre qui mesure l'altitude z). En effet, afin de combiner deux systèmes différents pour faire la localisation en plusieurs dimensions, les données qu'ils fournissent doivent être en syncronisation ou bien à deux fréquences assez proches ; i.e. la disponibilité de chaque ètat du système requiert une syncronisation avec les autres ètats. Sinon le système de localisation se déstabilise, ainsi, le système de contrôle ne serait plus robuste, et par conséquent, tout le système se perd dans l'espace.

- Dans le chapitre 5.17, on a proposé un algorithme basé sur la théorie des champs potentiels artificiels. Cette méthode a montré une bonne planification en présence d'un seul obstacle. Cependant, lorsqu'on augmente le nombre des obstacles, on risque de se trouver figé dans un minimum local, d'où la limitation de cet algorithme au sein des environnements plus complexes.

Bibliographie

[1] P. H. Moreno, A. Paranjape et T. Mylvaganam, "Deep Earth Drones", 2018.

[2] L. Bauersfeld et G. Ducard, "RTOB SLAM : Real-Time Onboard Laser-Based Localization and Mapping", *Vehicles*, t. 3, n° 4, p. 778-789, 2021.

[3] E. López, R. Barea, A. Gómez et al., "Indoor SLAM for micro aerial vehicles using visual and laser sensor fusion", in *Robot 2015: Second Iberian Robotics Conference*, Springer, 2016, p. 531-542.

[4] J. Qi, N. Yu et X. Lu, "A uav positioning strategy based on optical flow sensor and inertial navigation", in *2017 IEEE International Conference on Unmanned Systems (ICUS)*, IEEE, 2017, p. 81-87.

[5] D. Kominiak, S. S. Mansouri, C. Kanellakis et G. Nikolakopoulos, "MAV development towards navigation in unknown and dark mining tunnels", in *2020 28th Mediterranean Conference on Control and Automation (MED)*, IEEE, 2020, p. 1015-1020.

[6] A. Benghezal, R. Louali, A. Bazoula et T. Chettibi, "Trajectory generation for a fixed-wing UAV by the potential field method", in *2015 3rd International Conference on Control, Engineering & Information Technology (CEIT)*, IEEE, 2015, p. 1-6.

[7] M. I. Ribeiro, "Obstacle avoidance", *Instituto de Sistemas e Robótica, Instituto Superio Técnico*, t. 1, 2005.

[8] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots", in *Autonomous robot vehicles*, Springer, 1986, p. 396-404.

[9] S. Scherer, S. Singh, L. Chamberlain et M. Elgersma, "Flying fast and low among obstacles : Methodology and experiments", *The International Journal of Robotics Research*, t. 27, n° 5, p. 549-574, 2008.

[10] A. J. Berry, J. Howitt, D.-W. Gu et I. Postlethwaite, "A continuous local motion planning framework for unmanned vehicles in complex environments", *Journal of Intelligent & Robotic Systems*, t. 66, n° 4, p. 477-494, 2012.

[11] M. G. Park et M. C. Lee, "A new technique to escape local minimum in artificial potential field based path planning", *KSME international journal*, t. 17, n° 12, p. 1876-1885, 2003.

[12] S. Bouabdallah, A. Noth et R. Siegwart, "PID vs LQ control techniques applied to an indoor micro quadrotor", in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, IEEE, t. 3, 2004, p. 2451-2456.

[13] I. D. Cowling, J. F. Whidborne et A. K. Cooke, “Optimal trajectory planning and LQR control for a quadrotor UAV”, in *International Conference on Control*, 2006.

[14] T. Madani et A. Benallegue, “Backstepping control for a quadrotor helicopter”, in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2006, p. 3255-3260.

[15] M. O. Efe, “Robust low altitude behavior control of a quadrotor rotorcraft through sliding modes”, in *2007 Mediterranean Conference on Control & Automation*, IEEE, 2007, p. 1-6.

[16] D. Gurdan, J. Stumpf, M. Achtelik, K.-M. Doth, G. Hirzinger et D. Rus, “Energy-efficient autonomous four-rotor flying robot controlled at 1 kHz”, in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, IEEE, 2007, p. 361-366.

[17] G. Hoffmann, S. Waslander et C. Tomlin, “Quadrotor helicopter trajectory tracking control”, in *AIAA guidance, navigation and control conference and exhibit*, 2008, p. 7410.

[18] R. C. Smith et P. Cheeseman, “On the representation and estimation of spatial uncertainty”, *The international journal of Robotics Research*, t. 5, n° 4, p. 56-68, 1986.

[19] E. Skjellaug, “Feature-Based Lidar SLAM for Autonomous Surface Vehicles Operating in Urban Environments”, mém. de mast., NTNU, 2020.

[20] ——, “Feature-Based Lidar SLAM for Autonomous Surface Vehicles Operating in Urban Environments”, mém. de mast., NTNU, 2020.

[21] M. B. Alatise et G. P. Hancke, “Pose estimation of a mobile robot based on fusion of IMU data and vision data using an extended Kalman filter”, *Sensors*, t. 17, n° 10, p. 2164, 2017.

[22] A. J. Davison et D. W. Murray, “Simultaneous localization and map-building using active vision”, *IEEE transactions on pattern analysis and machine intelligence*, t. 24, n° 7, p. 865-880, 2002.

[23] C. Estrada, J. Neira et J. D. Tardós, “Hierarchical SLAM : Real-time accurate mapping of large environments”, *IEEE transactions on Robotics*, t. 21, n° 4, p. 588-596, 2005.

[24] J. Leonard et P. Newman, “Consistent, convergent, and constant-time SLAM”, in *IJCAI*, 2003, p. 1143-1150.

[25] J.-H. Kim et S. Sukkarieh, “Airborne simultaneous localisation and map building”, in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, IEEE, t. 1, 2003, p. 406-411.

[26] T. Vu, “Localisation, mapping avec detection, classification et suivi des objets mobiles”, thèse de doct., Ph. D. dissertation, Institut National Polytechnique de Grenoble, 2009.

[27] F. Lu et E. Milios, “Globally consistent range scan alignment for environment mapping”, *Autonomous robots*, t. 4, n° 4, p. 333-349, 1997.

[28] J. J. Leonard et H. F. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot.", in *IROS*, t. 3, 1991, p. 1442-1447.

[29] A. Elfes, "Using occupancy grids for mobile robot perception and navigation", *Computer*, t. 22, nº 6, p. 46-57, 1989.

[30] S. T. Pfister, S. I. Roumeliotis et J. W. Burdick, "Weighted line fitting algorithms for mobile robot map building and efficient data representation", in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, IEEE, t. 1, 2003, p. 1304-1311.

[31] D. Forsyth et J. Ponce, *Computer vision : A modern approach.* Prentice hall, 2011.

[32] S. Thrun, "Probabilistic robotics", *Communications of the ACM*, t. 45, nº 3, p. 52-57, 2002.

[33] S. Kohlbrecher, O. Von Stryk, J. Meyer et U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation", in *2011 IEEE international symposium on safety, security, and rescue robotics*, IEEE, 2011, p. 155-160.

[34] B. Balasuriya, B. Chathuranga, B. Jayasundara et al., "Outdoor robot navigation using Gmapping based SLAM algorithm", in *2016 Moratuwa Engineering Research Conference (MERCon)*, IEEE, 2016, p. 403-408.

[35] M. Filipenko et I. Afanasyev, "Comparison of various slam systems for mobile robot in an indoor environment", in *2018 International Conference on Intelligent Systems (IS)*, IEEE, 2018, p. 400-407.

[36] B. D. Lucas, T. Kanade et al., *An iterative image registration technique with an application to stereo vision*. Vancouver, 1981, t. 81.

[37] J. Gutmann, "Robuste navigation autonomer mobiler systeme (Ph. D. thesis)", *Albe rt-Ludwigs-Universitat Freiburg, Institut fur Informatik*, 2000.

[38] K. Lingemann, H. Surmann, A. Nuchter et J. Hertzberg, "Indoor and outdoor localization for fast mobile robots", in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, IEEE, t. 3, 2004, p. 2185-2190.

[39] T. Einsele, "Real-time self-localization in unknown indoor environment using a panorama laser range finder", in *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robot and Systems. Innovative Robotics for Real-World Applications. IROS'97*, IEEE, t. 2, 1997, p. 697-702.

[40] K. Lynch, *"Modern Robotics", Chapter10.6 Virtual Potential Fields*. Coursera.

Chapitre 7

Annexe

7.1 Paramètres utilisés en simulation

Les paramètres physiques pour les simulations du quadri-rotor sont résumés dans le tableau suivant :

Paramètre	Valeur	Unité
Masse du quadri-rotor	2	kg
Distance entre le centre d'un moteur et le centre de gravité	0.086	m
Moment d'inertie du quadri-rotor par rapport à son axe 'X'	0.00025	kg.m^2
Moment d'inertie du quadri-rotor par rapport à son axe 'Y'	0.000232	kg.m^2
Moment d'inertie du quadri-rotor par rapport à son axe 'Z'	0.000373	kg.m^2
Constante de gravité	9.8	m/s^2

TAB. 7.1 : Les paramètres physique utilisés pour le système de simulation

Les gains du contrôleur P-D utilisés dans les lois de la commande dans le deuxième chapitre sont ressemés dans le tableau suivant :

Symbole	Valeur
$k_{p,x}$	100
$k_{p,y}$	100
$k_{p,z}$	800
$k_{d,x}$	1
$k_{d,y}$	1
$k_{d,z}$	1
$k_{p,\phi}$	160
$k_{p,\theta}$	160
$k_{p,}$	160
$k_{d,\phi}$	1
$k_{d,\theta}$	1
$k_{d,}$	1

TAB. 7.2 : Les paramètres des lois de commande utilisées en simulation

7.2 Codes développés

Code d'évitement d'obstacle

```
//Algorithme d'évitement d'obstacle
#include <ros/ros.h>
#include <darknet_ros_msgs/BoundingBoxes.h>
#include <sensor_msgs/LaserScan.h>
#include <gnc_functions.hpp>
#include <mavros_msgs/State.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include<geometry_msgs/TwistStamped.h>
#include <iostream>

using namespace std;

geometry_msgs::PoseStamped pose;
geometry_msgs::TwistStamped vel_cmd;
mavros_msgs::State current_state;
geometry_msgs::Point goal;

geometry_msgs::Point current_pos;
float dVx = 0;
float dVy = 0;
float dx=0, dy=0;
float current_heading;
float deg2rad = (M_PI/180);
float rad2deg = (180/M_PI);
float r=0.2, s=1.8;
float avoidance_vector_x = 0;
float avoidance_vector_y = 0;
float norm(float x, float y)
{
  return sqrt(pow(x,2) + pow(y,2));
}
float distance(float x1, float y1, float x2, float y2)
{
  return sqrt(pow((x1-x2),2) + pow((y1-y2),2));
}

void scan_cb(const sensor_msgs::LaserScan::ConstPtr& msg)
{

  sensor_msgs::LaserScan current_2D_scan;
    current_2D_scan = *msg;
  avoidance_vector_x = 0;
  avoidance_vector_y = 0;

  for(int i=1; i<current_2D_scan.ranges.size(); i++)
  {
    float d0 = 5;
    float k = 10;

    if(current_2D_scan.ranges[i] < d0 && current_2D_scan.ranges[i] > .35)
    {
```

```
      float x = cos(current_2D_scan.angle_increment*i);
      float y = sin(current_2D_scan.angle_increment*i);
      float U = -.5*k*pow(((1/current_2D_scan.ranges[i]) - (1/d0)), 2);

      avoidance_vector_x = avoidance_vector_x + x*U;
      avoidance_vector_y = avoidance_vector_y + y*U;

    }
  }

}



void attraction_set(float k=5)
{
    current_pos.x = current_pose_g.pose.pose.position.x;
    current_pos.y = current_pose_g.pose.pose.position.y;
  float d = distance(current_pos.x,current_pos.y,goal.x,goal.y);
  float theta = atan2((goal.y-current_pos.y), (goal.x-current_pos.x));

  if(d > r+s)
  {

    //float U = 0.5*k*pow(distance(current_pos.x,current_pos.y,goal.x,goal.
    y), 2);
    //dVx =  k*(goal.x-current_pos.x) - b*current_pose_g.twist.twist.linear
    .x;
    //dVy =  k*(goal.y-current_pos.y) - b*current_pose_g.twist.twist.linear
    .y;
    dx = k*s*cos(theta) - avoidance_vector_x;
    dy = k*s*sin(theta) - avoidance_vector_y;

    //vel_cmd.twist.linear.x = current_pose_g.twist.twist.linear.x + dVx;
    //vel_cmd.twist.linear.y = current_pose_g.twist.twist.linear.y + dVy;


  }
  else if(d>r && d<s+r)
  {
    dx = k*(d-r)*cos(theta) - avoidance_vector_x;
    dy = k*(d-r)*sin(theta) - avoidance_vector_y;

  }
  else
  {
    dx = -avoidance_vector_x;
    dy = -avoidance_vector_y;
  }
  current_heading = get_current_heading();
  dx = dx*cos((current_heading)*deg2rad) - dy*sin((current_heading)*deg2rad
   );
  dy = dx*sin((current_heading)*deg2rad) + dy*cos((current_heading)*deg2rad
   );


  if( norm(dx, dy) > 3)
```

```
105   {
106     dx = 3 * (dx/norm(dx,dy));
107     dy = 3 * (dy/norm(dx,dy));
108   }
109   cout << "dx = " << dx << endl;
110   cout << "dy = " << dy << endl;
111   cout << "  " << endl;
112   cout << "set_x = " << current_pos.x + dx << endl;
113   cout << "set_y = " << current_pos.y + dy << endl;
114   cout << "  " << endl;
115   //set_destination(current_pos.x + dx, current_pos.y + dy, goal.z, 0);
116   pose.pose.position.x = current_pos.x + dx;
117     pose.pose.position.y = current_pos.y + dy;
118     pose.pose.position.z = goal.z;
119     local_pos_pub.publish(pose);
120
121
122
123 }
124
125 int main(int argc, char **argv)
126 {
127   goal.x =  -6;
128   goal.y =  4;
129
130   goal.z = 2.5;
131   //initialize ros gnc node
132   ros::init(argc, argv, "gnc_node");
133   ros::NodeHandle n;
134   ros::Subscriber sub = n.subscribe("/darknet_ros/bounding_boxes", 1,
       scan_cb);
135   ros::Subscriber collision_sub = n.subscribe<sensor_msgs::LaserScan>("/
       laser/scan", 1, scan_cb);
136
137     //initialize ros offb node
138   ros::init(argc, argv, "offb_node");
139   ros::NodeHandle nh;
140
141   ros::Subscriber state_sub = nh.subscribe<mavros_msgs::State>("mavros/
       state", 10, state_cb);
142   ros::Subscriber odom_sub = nh.subscribe<nav_msgs::Odometry>("/mavros/
       local_position/odom", 10, pose_cb);
143
144   ros::Publisher local_pos_pub = nh.advertise<geometry_msgs::PoseStamped>("
       mavros/setpoint_position/local", 10);
145   ros::Publisher vel_pub = nh.advertise<geometry_msgs::TwistStamped>("/
       mavros/setpoint_velocity/cmd_vel", 10);
146
147   ros::ServiceClient arming_client = nh.serviceClient<mavros_msgs::
       CommandBool>("mavros/cmd/arming");
148   ros::ServiceClient set_mode_client = nh.serviceClient<mavros_msgs::
       SetMode>("mavros/set_mode");
149
150
151   //ros::Rate rate(20);
152   int counter = 0;
153
```

```
154
155   //initialize control publisher/subscribers
156   init_publisher_subscriber(n);
157     // wait for FCU connection
158   wait4connect();
159         //set offboard mode
160     mavros_msgs::SetMode offb_set_mode;
161     offb_set_mode.request.custom_mode = "OFFBOARD";
162
163     mavros_msgs::CommandBool arm_cmd;
164     arm_cmd.request.value = true;
165
166     ros::Time last_request = ros::Time::now();
167
168
169     while(ros::ok() && counter==0){
170         if( current_state.mode != "OFFBOARD" &&
171             (ros::Time::now() - last_request > ros::Duration(5.0))){
172             if( set_mode_client.call(offb_set_mode) &&
173                 offb_set_mode.response.mode_sent){
174                 ROS_INFO("Offboard enabled");
175               counter++;
176             }
177             last_request = ros::Time::now();
178         }
179         else {
180             if( !current_state.armed &&
181                 (ros::Time::now() - last_request > ros::Duration(5.0))){
182                 if( arming_client.call(arm_cmd) &&
183                     arm_cmd.response.success){
184                     ROS_INFO("Vehicle armed");
185                 }
186                 last_request = ros::Time::now();
187             }
188         }
189
190
191     pose.pose.position.x = 0;
192     pose.pose.position.y = 0;
193     pose.pose.position.z = goal.z;
194     local_pos_pub.publish(pose);
195
196
197
198 }
199
200   //wait for used to switch to mode Offboard
201   wait4start();
202
203   //create local reference frame
204   initialize_local_frame();
205
206   //request takeoff
207   takeoff(goal.z);
208   local_pos_pub.publish(pose);
209   //specify control loop rate. We recommend a low frequency to not over
      load the FCU with messages. Too many messages will cause the drone to be
```

```
    sluggish
  ros::Rate rate(10.0);
  while(ros::ok())
  {
    ros::spinOnce();
    rate.sleep();
    attraction_set();
    //vel_pub.publish(attraction_set());
  }




  return 0;

}
```

Code du mode offboard

```
//Algorithme du mode offboard
/**
 * @file offb_node.cpp
 * @brief Offboard control example node, written with MAVROS version 0.19.x
   , PX4 Pro Flight
 * Stack and tested in Gazebo SITL
 */

#include <ros/ros.h>
#include <geometry_msgs/PoseStamped.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>

mavros_msgs::State current_state;
void state_cb(const mavros_msgs::State::ConstPtr& msg){
    current_state = *msg;
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "offb_node");
    ros::NodeHandle nh;

    ros::Subscriber state_sub = nh.subscribe<mavros_msgs::State>
            ("mavros/state", 10, state_cb);
    ros::Publisher local_pos_pub = nh.advertise<geometry_msgs::PoseStamped>
            ("mavros/setpoint_position/local", 10);
    ros::ServiceClient arming_client = nh.serviceClient<mavros_msgs::
    CommandBool>
            ("mavros/cmd/arming");
    ros::ServiceClient set_mode_client = nh.serviceClient<mavros_msgs::
    SetMode>
            ("mavros/set_mode");

    //the setpoint publishing rate MUST be faster than 2Hz
```

```
    ros::Rate rate(20.0);

    // wait for FCU connection
    while(ros::ok() && !
current_state.connected){
        ros::spinOnce();
        rate.sleep();
    }

    geometry_msgs::PoseStamped pose;
    pose.pose.position.x = 1;
    pose.pose.position.y = 0;
    pose.pose.position.z = 2.5;
    //pose.pose.orientation.z = 3.14;

    //send a few setpoints before starting
  //  for(int i = 100; ros::ok() && i > 0; --i){
 //       local_pos_pub.publish(pose);
  //      ros::spinOnce();
  //      rate.sleep();
 //   }

    mavros_msgs::SetMode offb_set_mode;
    offb_set_mode.request.custom_mode = "OFFBOARD";

    mavros_msgs::CommandBool arm_cmd;
    arm_cmd.request.value = true;

    ros::Time last_request = ros::Time::now();

    while(ros::ok()){
        if( current_state.mode != "OFFBOARD" &&
            (ros::Time::now() - last_request > ros::Duration(5.0))){
            if( set_mode_client.call(offb_set_mode) &&
                offb_set_mode.response.mode_sent){
                ROS_INFO("Offboard enabled");
            }
            last_request = ros::Time::now();
        } else {
            if( !current_state.armed &&
                (ros::Time::now() - last_request > ros::Duration(5.0))){
                if( arming_client.call(arm_cmd) &&
                    arm_cmd.response.success){
                    ROS_INFO("Vehicle armed");
                }
                last_request = ros::Time::now();
            }
        }

       local_pos_pub.publish(pose);

        ros::spinOnce();
        rate.sleep();
    }

    return 0;
}
```

Code du noeud publisher

```
//Algorithme du noeud publisher

#include <ros/ros.h>
#include <geometry_msgs/PoseStamped.h>
#include <iostream>

geometry_msgs::PoseStamped pose_in;
void pub_pose(const geometry_msgs::PoseStamped::ConstPtr& msg){
    pose_in.pose = msg->pose;
    std::cout << "GOT POSITION\n";
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "publisher");
    ros::NodeHandle nh;

    ros::Subscriber state_sub = nh.subscribe<geometry_msgs::PoseStamped>
            ("/slam_out_pose", 10, pub_pose);
    ros::Publisher local_pos_pub = nh.advertise<geometry_msgs::PoseStamped>
            ("/pose", 10);


    //the setpoint publishing rate MUST be faster than 2Hz
    ros::Rate rate(20.0);

    // wait for FCU connection
    while(ros::ok()){
        ros::spinOnce();
        rate.sleep();
    }


    geometry_msgs::PoseStamped pose_out;
    pose_out.pose.position.x = pose_in.pose.position.x;
    pose_out.pose.position.y = pose_in.pose.position.x;
    pose_out.pose.position.z = pose_in.pose.position.x;


    local_pos_pub.publish(pose_out);
    ros::spinOnce();
    rate.sleep();


    return 0;
}
```