



ÉCOLE DE L'AIR

DIPLÔME D'INGÉNIEUR
- MÉMOIRE DE STAGE DE FIN D'ÉTUDES -

**Développement d'algorithme de
planification de trajectoire de drones
utilisant des réseaux de neurones et
de l'apprentissage par renforcement**

THE UNIVERSITY OF ARIZONA

Maxime LEROUX

sous la direction de :
Dr. Eniko ENIKOV



Table des matières

1	Introduction	1
2	Motivation du projet	3
2.1	Utilisation opérationnelle	3
2.2	Cadre de l'étude	5
2.3	Matériel de travail	6
I	Modèle mathématique	7
3	Hypothèses, normes et notations utilisées	9
3.1	Hypothèses de travail	9
3.2	Normes utilisées	11
4	Objectifs	15
4.1	Maintien de l'attitude	15
4.2	Navigation simple	15
4.3	<i>Manœuvres</i>	16
5	Mise en équation initiale	17
5.1	Modélisation du drone et détermination des constantes	17
5.2	Calcul des efforts et moments du trirotor	20
5.3	Modèle de la plateforme à 6 degrés de liberté	22
5.4	Étude de l'équilibre	23
II	Établissement de lois de commande	25
6	Modélisation sous simulink	27
6.1	De la nécessité d'un modèle	27
6.2	Structure générale les lois de commandes	27
7	Une loi de commande classique	29
7.1	Première simulation : boucle ouverte	29
7.2	Linéarisation des Efforts	30
7.3	Contrôleur Proportionnel-Integral-Dérivée (PID)	31
7.4	Une nouvelle répartition des commandes du drone	33

8 Contrôle via représentation d'état	35
8.1 Principe d'une commande par retour d'état	35
8.2 Le problème du régulateur quadratique linéaire	37
8.3 Résultat du LQR	39
9 Utilisation d'un Model Predictive Controller (MPC)	43
9.1 Préliminaires	43
9.2 Détermination des paramètres du contrôleur	46
9.3 Résultats du MPC	48
10 L'intelligence Artificielle : une solution miracle ?	51
10.1 Un domaine à la pointe	51
10.2 Comment faire apprendre à apprendre ?	53
10.2.1 Prolégomènes	53
10.2.2 Quelle politique adopter ?	54
10.2.3 Vers l'utilisation de réseaux de neurones	56
10.3 Tentative d'implémentation	59
11 Perspectives et conclusion	63
Bibliographie	65
Annexes	69

VERSION à faire signer POUR CHAQUE TRAVAIL

Déclaration d'intégrité relative au plagiat

Je certifie que

1. je suis l'auteure ou l'auteur
 - du présent travail
 - ou
 - de ma contribution au présent travail d'équipe;
2. j'ai attribué et cité tout contenu emprunté selon les pratiques méthodologiques attendues.

Nom: Leroux

Prénom: Maxime

Date: 01/06/2021

Signature:



L'envoi par messagerie électronique à partir d'une adresse @ecole-air.fr et/ou le dépôt sur le site moodle.ecole-air.fr attestera de l'identité du signataire.

Réserve de responsabilité

Le ministère de la Défense et l'université n'entendent donner, ni approbation, ni improbation aux idées émises dans les mémoires et autres documents soutenus, en vue de l'obtention de grades universitaires et diplômes. Ces opinions doivent être considérées comme propres à leurs auteurs et n'expriment en rien la position des institutions auxquelles ils appartiennent.

Abstract

At the root of many debates and phantasms, Unmanned Aerial Vehicles (UAVs) are nowadays a major asset in the control of the third dimension. Multi-rotors UAVs are praised for a rising number of applications, such as cinema, racing or monitoring at-risk zones such as aeronautical platform. If the civilian industry has already used them for some years now, the military first tried to prevent any incident involving UAVs before considering uses of this quite recent technology.

Such a use of a UAV in the military is studied in the memoire. We will present the bases of a project allowing the setup of a remotely controlled surveillance point using a Vertical Take-Off and Landing (VTOL) multi-rotor UAV able to operate during conflicts in urban areas while ignoring electromagnetics counter-measures. To do that, we will implement different types of controllers, ranging from usual PID to artificial-intelligence powered ones with reinforcement learning.

Résumé

Source de nombreux débats et fantasmes, les drones sont de nos jours un élément essentiel du contrôle de la 3^{ème} dimension. En particulier, les drones multi-rotors sont de plus en plus prisés pour nombre d'activités telles que le cinéma, la course ou la surveillance de zones à risque, tel que des plateformes aéronautiques. Si l'industrie civile les utilise depuis déjà quelques années, le domaine militaire a d'abord majoritairement tenté de s'en prémunir avant d'étudier des possibilités d'utilisation de cette technologie récente.

Une telle possibilité est étudiée dans ce mémoire. Nous nous attachons à présenter une ébauche de projet permettant la mise en place d'un point de surveillance contrôlable à distance à l'aide d'un drone multi rotor pouvant opérer lors de conflits en environnement urbain en faisant fi de toute mesure de brouillage. Pour ce faire, nous allons implémenter différents contrôleurs, allant de PID classiques à des intelligences artificielles utilisant l'apprentissage par renforcement.

Remerciements

Je souhaiterais tout d'abord remercier très chaleureusement le Dr. Enikov de m'avoir permis de faire ce stage dans une grande université américaine malgré la condition sanitaire dégradée dans laquelle ce stage a été réalisé.

Je voudrais aussi remercier le Dr. Faure ainsi que les cadres de la DGER, sans qui ce stage n'aurait pas pu exister.

Je souhaite aussi remercier le Commandant Lefebvre et la Commandant Doussineau pour m'avoir accordé assez de temps pour trouver le stage malgré des contraintes temporelles serrées.

Je remercie aussi particulièrement le Dr. Bateman, M. Roman et M. Barache qui m'ont fourni de l'aide au début de mon stage.

Enfin, je tiens à remercier l'ensemble des professeurs qui nous ont accompagnés jusqu'ici, et ont, à ce titre, contribué à ce mémoire.

Chapitre 1

Introduction

De nombreuses innovations majeures ont jalonné l'histoire de l'aviation. De l'avion des frères Wright aux Rafales et A400M en passant par les P-47, on ne peut que constater les avancées toujours plus audacieuses des ingénieurs aéronautiques.

Aussi, nous pouvons nous poser une question, simple en apparence : quelle sera la prochaine innovation majeure dans le domaine de l'aviation ?

Au vu des avancées spectaculaires de l'intelligence artificielle (IA) dans tous les domaines, il est certain que l'aviation sera impactée. Et impactée, elle l'est déjà. En effet, de plus en plus d'appareils utilisent l'IA comme aide au pilotage. Mais qu'en est-il du moment où elle sera assez performante pour remplacer le pilote ?

Ce jalon semble bien proche. En effet, les États-Unis sont en passe de mettre en œuvre ce qu'ils nomment le 5th Generation Aerial Target (5GAT) [1]. Ayant l'allure d'un F-22 Raptor, ce drone sera capable d'imiter des chasseurs comme des Sukhoi Su-57 russe ou des Chengdu J-20 et Shenyang J-31 chinois. À terme, il sera même possible de l'utiliser en tant qu'aéronef de reconnaissance. Le programme SCAF souhaite lui aussi mettre en œuvre des drones, que ce soit en tant que simple transporteur de bombes ou d'équipier de l'avion maître, habité. Plus inquiétant, des drone autonomes tueurs ont potentiellement été utilisé en Libye en Mars 2020 [2, 3].

Du drone militaire au drone photographe en passant par le drone FPV, les différents organismes sont en pleine réflexion quant à la législation de cette nouvelle technologie qui représente un enjeu sécuritaire et technologique pour notre monde. De plus en plus présents, ils effraient - à raison - les politiques. Mme Florence Parly, actuelle ministre des Armées, lors de son discours de clôture de la 15e Université d'été de la Défense à Toulon [4], évoquait ce nouvel outil et les menaces qu'il représentait en matière de Défense. Plus récemment, on a vu pendant le confinement l'interdiction de l'utilisation des drones par les forces de l'ordre afin de faire respecter ce dernier [5].

Les drones semblent être cette révolution de l'aviation, et du contrôle de la 3ème dimension en général.

Utilisés massivement en opération par l'armée française, les MQ-9 Reaper sont devenus un atout incontournable dans la lutte contre le terrorisme. Depuis octobre 2020, de

nouvelles versions permettant l'emport d'armement ont été reçues et très vite employés, notamment dans des missions SCAR (Strike Coordination And Reconnaissance). D'une autonomie de 24H, dotés de capteurs très performants, ce drone piloté à distance permet un renseignement de très bonne facture.

Néanmoins, l'emploi des REAPERS n'est pas sans inconvénients : malgré une autonomie exceptionnelle comparée aux chasseurs qui, jusque-là, effectuaient le renseignement, elle reste limitée à moins de 24H [6]. Il est aussi obligé de voler très haut sous peine d'être repéré au bruit par les troupes adverses, le coût de l'heure de vol est évaluée à près de 6000€, et enfin l'armée française n'en déploie pas plus de deux ou trois par opération. De ce fait, son utilisation pour du renseignement en zone urbaine pose des problèmes certains.

Une alternative utilisée par les forces spéciales Française depuis 2016 est le microdrone Prox Dynamics PD-100 dit Black Hornet, de la société FLIR [7]. Ce drone permet du renseignement de proximité, mais est limité par son temps d'utilisation (25 minutes) et sa portée : l'opérateur doit se placer à proximité du drone pour pouvoir le commander, et donc se mettre en danger en zone hostile.

Ainsi, ce mémoire vise à proposer une alternative à l'utilisation des MQ-9 Reaper et des Black Hornets en opération pour de la surveillance de bâtiment en zone urbaine.

Chapitre 2

Motivation du projet

2.1 Utilisation opérationnelle

Nous l'avons évoqué en introduction, si les Reaper et les Black Hornet sont des atouts certains en opération, ils souffrent de limites d'utilisations que nous allons tacher de préciser ici.

MQ-9 Reaper :



FIGURE 2.1 – Un des MQ-9 Reaper Français

Commandés aux États-Unis en 2013, les premiers Reapers sont employés par les forces françaises dès janvier 2014. En Octobre 2020, 6 nouveau drones sont livrés à l'escadron 1/33 Belfort, et son capable d'emporter de l'armement.

En opération, les Reapers sont devenus un atout certain et capital. Victime de leur succès, leur nombre limité entraîne nécessairement un choix d'objectifs à surveiller.

Par ailleurs, afin d'éviter d'être repérés, les opérateurs de cet aéronef se doivent de voler très haut afin de ne pas être repéré. Si ce n'est pas un problème majeur en terrain découvert, cela devient tout de suite plus gênant en milieu urbain.

En effet, si l'on veut surveiller un bâtiment particulier sur une longue durée, les informations fournies par les capteurs du drone pourraient ne pas suffire. Par ailleurs, son autonomie de 24H devient vite limitante pour de telles missions de renseignement.

Pour résumer, l'utilisation des Reaper pour de la surveillance en zone urbaine est une des nombreuses possibilités offerte par ce système d'arme, mais est loin d'être sa

spécialité. Au vu du nombre limité disponibles en opération, il semble judicieux de l'utiliser dans les domaines où il excelle, comme le SCAR.

Prox Dynamics PD-100 - Black Hornet : MQ-9 Reaper :



FIGURE 2.2 – Un Black Hornet en vol

Utilisés par les forces spéciales Françaises depuis 2016, notamment par le Commando Parachutiste de l’Air n°10, le Black Hornet mesure 10 cm de long et 2,5 de large et pèse 18 grammes, batterie comprise. Il a une portée théorique de 1500 mètres et 25 minutes d’autonomie. Utilisé le plus souvent par deux, son temps de recharge est d’environ 25 minutes pour 90% de batterie.

Il dispose de trois caméras, une devant, une sous le ventre et une orientée vers le bas à 45°. Il dispose aussi d’un capteur infrarouge dans ses versions les plus récentes. Armé de ce dispositif, les forces spéciales peuvent effectuer du renseignement de proximité de dernière minute, crucial dans leurs opérations.

Néanmoins, ce nanodrone présente de clairs désavantages. Avec sa courte portée, l’opérateur doit se rapprocher très proche du danger potentiel. Par ailleurs, il est difficile à utiliser en ville, le bruit - bien que très faible pour un drone - reste suffisant pour qu’il soit aisément repéré par des passants ou des gardes.

Ainsi, il semblerait que cette solution ne soit pas non plus la meilleure pour effectuer de renseignement en zone urbaine. Bien que présentant des atouts certains, cette solution ne permet pas de rester en surveillance autour d’un point d’intérêt assez longtemps et sans mettre en danger son opérateur.

Il semblerait que là où le Reaper excelle, le Black Hornet pêche ; et vice-versa. On souhaiterais ainsi trouver un compromis offrant la sécurité du Reaper avec la proximité du Black Hornet. **De façon cruciale**, cette solution doit pouvoir offrir un temps sur zone très conséquent et ne doit pas mettre en danger son opérateur, tout en permettant une surveillance continue d’une zone.

2.2 Cadre de l'étude

Afin de résoudre notre problème, on se propose de mettre en œuvre un drone de dimensions moyennes, pourvu d'un algorithme lui permettant d'aller d'un point donné à un autre, et d'effectuer un manœuvre pour s'arrimer sur plus ou moins n'importe quelle surface à l'aide d'un module situé sous le drone.

L'intérêt étant que, une fois arrimé à sa surface, on dispose d'une caméra contrôlable à distance placée dans un endroit stratégique - et ce sans avoir pris le moindre risque humain. En envoyant le drone de nuit, ou de concert avec un chasseur pour détourner l'attention pendant les quelques secondes nécessaires à la manœuvre, on peu sans risque placer une caméra dans une zone difficile d'accès pour l'espionner de façon durable et sécurisée.

Par ailleurs, une fois sa mission effectuée, pour peu que le drone soit placé en hauteur, il est tout à fait envisageable de le faire repartir pour ne laisser aucune trace.

On peut envisager pléthores de conceptions différentes pour le rendre le plus camouflé possible une fois en place et autant de modules interchangeables pour qu'il puisse s'adapter au plus de surfaces possibles.

Néanmoins, dans ce mémoire, nous nous concentrerons sur l'établissement de lois de contrôles visant à réaliser le pilote automatique du drone, ainsi que le programme permettant la manœuvre décrite ci-dessus. Nous formaliserons par la suite ce que nous entendons par **manœuvre**.

2.3 Matériel de travail

Pratiquement, pour réaliser ce projet, nous avons à notre disposition un drone monté par des élèves de *The University of Arizona* (UofA) en 2013.

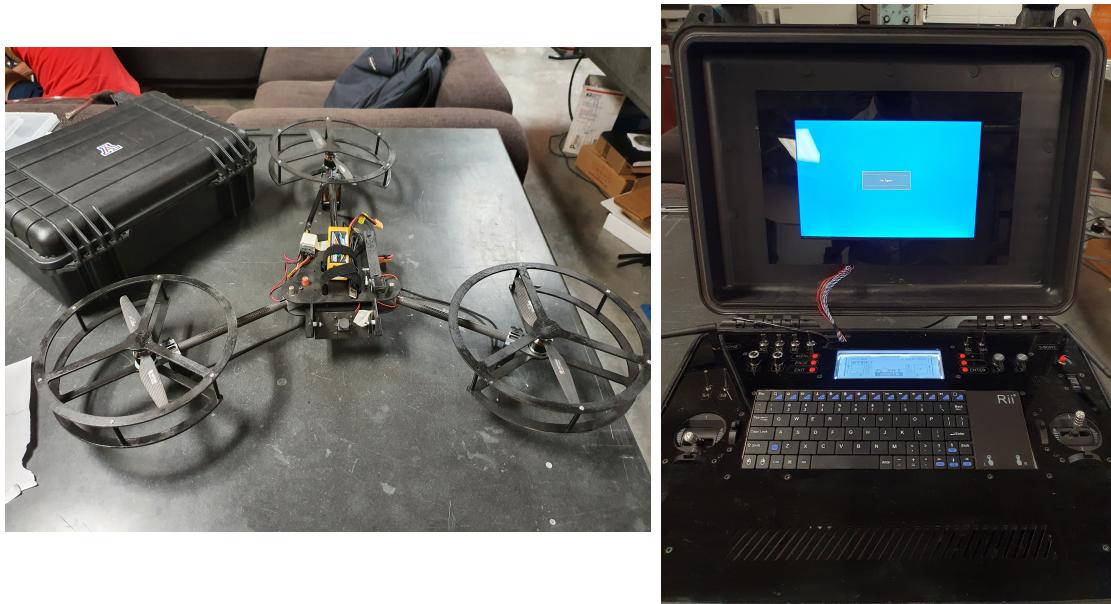


FIGURE 2.3 – Drone mis à disposition avec la mallette de commande

Ce drone est un modèle possédant 3 rotors tournant dans le sens trigonométrique, avec le rotor arrière muni d'un servomoteur lui permettant de pivoter sur son axe. Les trois bras ont la même longueur et forment avec leurs voisins respectifs un angle de 120° . Une mallette refermant un ordinateur simple permettant de piloter ce drone et de recevoir l'image que fournit la caméra frontale a été développée en même temps que le drone.

Nous avons aussi à notre disposition les logiciels MATLAB et Simulink, qui nous serviront à effectuer nos calculs et simulations.

Enfin, l'UofA a mis à notre disposition un laboratoire entier possédant divers outils et instruments de mesure.

Première partie

Modèle mathématique

Chapitre 3

Hypothèses, normes et notations utilisées

3.1 Hypothèses de travail

Dans la suite de ce mémoire, on considérera plusieurs hypothèses qui seront valables pour le reste des chapitres.

- Terre plate

On se place dans des laps de temps et un espace relativement limité (de l'ordre de la dizaine de kilomètres), aussi on se permet de négliger l'influence de la rotundité de la terre sur notre drone.

- Force de Coriolis négligée

Comme annoncé ci-dessus, on considère que l'évolution du drone sera assez limitée dans le temps, son influence sur notre drone sera donc négligeable.

En effet, la norme de l'accélération de Coriolis, notée a_c vaut :

$$a_c = 2\omega \cdot v \cdot \sin \phi$$

avec :

$\omega = \frac{2\pi}{86164} \text{ rad.s}^{-1}$ (86164 étant le nombre de secondes d'un jour sidéral),

$v = 2m.s^{-2}$ la vitesse de l'objet étudié dans son référentiel,

$\phi = 90 \text{ deg}$ pour prendre le cas le plus défavorable.

On évalue ainsi $a_c \approx 3.10^{-4}m.s^{-2}$, ce qui représente moins d'un millième de la pesanteur. On peut donc négliger ce terme.

- Structure rigide

Au vu de la taille et du poids de notre drone et des forces mises en œuvre, on peut considérer qu'étant donné un matériau convenable pour le drone il n'y aura que très peu de flexion et de torsion sur le corps du drone et sur les pales des hélices.

On néglige donc cet aspect dans notre étude.

- Masse constante

On considère que, une fois décollé, le drone ne prends ni ne se sépare de modules. Par ailleurs, fonctionnant à l'aide de batteries dite LiPo (Lithium-ion Polymère), le carburant ne fait pas changer la masse au cours du vol.

Ainsi, on travaillera à masse constante pour notre drone.

- ρ constant

Le modèle de l'atmosphère valable entre 0 et 11 000 mètres employé dans ce mémoire donne :

$$\rho = \frac{P_0 \left(1 + \frac{ah}{T_0}\right)^{5,2561}}{RT} \quad \text{avec :} \quad T_0 = 288,15 \text{ K}$$

$$T = T_0 + \frac{ah}{T_0} \quad a = -6,5 \cdot 10^{-3} \text{ K.m}^{-1}$$

$$P_0 = 1013,25 \text{ hPa}$$

$$R = 267,3 \text{ J.K}^{-1}.mol^{-1}$$

Notre drone n'évoluera pas plus haut de 300 mètres, aussi en évaluant l'expression précédente pour $h = 0$ et $h = 300$, on trouve ainsi : $\frac{\rho_{300m}}{\rho_{sol}} = 0,967$

On considère que cette différence est très faible et ainsi on néglige la variation de ρ selon l'altitude. Ainsi, la portance d'une hélice n'est pas fonction de l'altitude.

3.2 Normes utilisées

Représentation des rotations

On utilisera par la suite les angles d'Euler pour représenter les rotations. Ils sont construits autour de trois rotations successives :

- une rotation d'angle ψ autour de l'axe \vec{z}_E
- une rotation d'angle θ autour de l'axe \vec{y}_1
- une rotation d'angle ϕ autour de l'axe \vec{x}_2

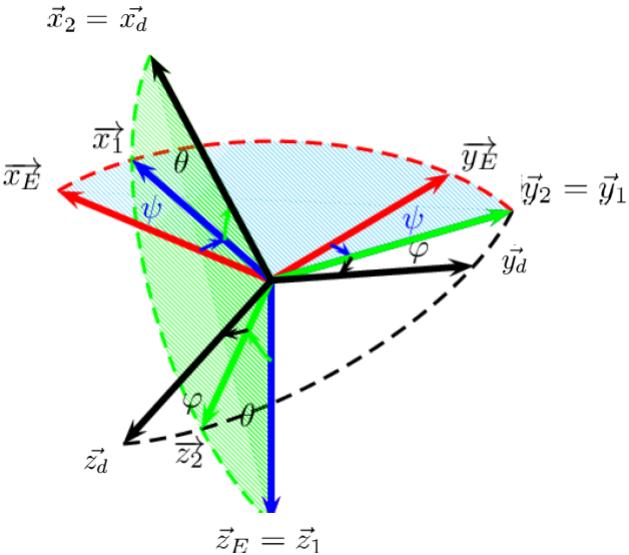


FIGURE 3.1 – Transformation des angles d'Euler et définition des axes

On a, pour un vecteur $v^{\mathcal{R}_E}$ quelconque exprimé dans le repère terrestre \mathcal{R}_E et ce même vecteur exprimé dans le repère du drone \mathcal{R}_d :

$$\vec{v}_{\mathcal{R}_E} = T_{dE} \cdot \vec{v}_{\mathcal{R}_d}$$

avec :

$$T_{dE} = \begin{pmatrix} \cos \theta \cos \psi & \sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi & \cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi \\ \cos \theta \sin \psi & \sin \phi \sin \theta \sin \psi + \cos \phi \cos \psi & \cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi \\ -\sin \theta & \sin \phi \cos \theta & \cos \phi \cos \theta \end{pmatrix} \quad (3.1)$$

Par ailleurs, $T_{dE} \in \mathcal{O}_3$, ainsi $T_{dE}^{-1} = T_{dE}^T = T_{Ed}$.

On notera toutefois que cette matrice présente une singularité pour $\theta = \pm \frac{\pi}{2}$, correspondant au blocage de Cardan [8].

Repères utilisés

On utilise deux repères pour notre drone, l'un fixe par rapport au repère terrestre et l'autre lié au drone, et fixe dans le référentiel lié au drone. Ces deux repères sont liés par la matrice de passage T_{dE} , décrite dans l'équation (3.1). Les repères décrits ci-après peuvent être visualisés sur la figure 3.1.

Repère terrestre

$\mathcal{R}_E (O, \vec{x}_E; \vec{y}_E; \vec{z}_E)$, orthonormé direct galiléen.

L'axe $(O\vec{x}_E)$ pointe vers le nord géographique, et l'axe $(O\vec{y}_E)$ pointe vers l'est. Le point O, lui, est défini tel que au début du vol, il coïncide avec le point G.

Repère lié au drone

$\mathcal{R}_d (G, \vec{x}_d; \vec{y}_d; \vec{z}_d)$, orthonormé direct.

L'axe $(G\vec{x}_d)$ pointe vers le nez du drone, et l'axe $(G\vec{y}_d)$ pointe à tribord. Le point G est défini comme étant le centre de gravité du drone, supposé au centre du drone. À noter que l'axe $(G\vec{z}_d)$ pointe vers le ventre du drone. Ainsi, quand le drone est à plat, cet axe pointe vers le sol.

Table des symboles

Dans la suite de ce mémoire, on utilisera les notations suivantes :

G	Centre du gravité du drone
\mathcal{R}_E	Repère terrestre
\mathcal{R}_d	Repère drone
T_{dE}	Matrice de passage de \mathcal{R}_d vers \mathcal{R}_E
m	Masse du drone (kg)
b	Coefficient de portance d'une hélice ($N.s^2.rad^{-2}$)
d	Coefficient de trainée d'une hélice ($N.m.s^2.rad^{-2}$)
P_i	Poussée générée par le rotor i (N)
T_i	Trainée générée par le rotor i ($N.m$)
l	Longueur des bras du drone (m)
$(k_u; k_v; k_w)$	Coefficients de trainée du drone
$(k_p; k_i; k_d)$	Paramètres relatifs au contrôleurs de type PID
I	Matrice d'inertie du drone
$(I_{xx}; I_{yy}; I_{zz})$	Coefficients de la matrice d'inertie du drone ($kg.m^2$)
V	Norme de la vitesse du drone dans \mathcal{R}_E
ω_i	Vitesse de rotation de l'hélice i ($rad.s^{-1}$)
δ	Angle du rotor de queue (rad)

Efforts (\mathcal{R}_d)	Vitesses (\mathcal{R}_d)	Positions (\mathcal{R}_E)
F_x	u	x_E
F_y	v	y_E
F_z	w	z_E
Moments (\mathcal{R}_d)	Vitesses de rotations (\mathcal{R}_d)	Attitude ($\mathcal{R}_d/\mathcal{R}_E$)
Moment de Roulis L	Roulis p	Gîte ϕ
Moment de Tangage M	Tangage q	Assiette θ
Moment de Lacet N	Lacet r	Cap ψ

Chapitre 4

Objectifs

Maintenant que les normes sont posées et le matériel est présenté, il nous est possible de présenter plus en détail les objectifs de ce projet.

4.1 Maintien de l'attitude

Le premier objectif que nous nous fixons est de parvenir à maintenir l'attitude du drone. Nous verrons plus tard que, sans implémentation de loi de commande (en boucle ouverte donc), le drone n'est pas stable : il diverge (lentement mais sûrement) et tombe à la moindre perturbation. Dans une certaine mesure, il est tel un crayon posé en équilibre instable sur sa mine.

On souhaite donc implémenter dans un premier temps un contrôle de l'attitude du drone. On veut que le drone puisse rester en place indéfiniment, même s'il est perturbé. Cela reviens à contrôler les angles de gîte, d'assiette et de cap. On maintiendra aussi l'altitude, afin que notre drone puisse rester en vol quasi-stationnaire.

4.2 Navigation simple

Le second objectif est de pouvoir diriger le drone. Le contrôle de l'attitude ne garantit pas que le drone reste au même point, et nous une perturbation constante (du vent par exemple), il sera déporté. On cherchera ainsi, étant donné un point dans \mathcal{R}_E , que le drone s'y dirige et y reste.

4.3 *Manœuvres*

Enfin, l'objectif clef que nous nous sommes fixés lors de ce projet est, étant donné une position et une attitude, de générer une trajectoire qui emmène le drone dans cet état. Mathématiquement, on souhaite que étant donné un état quelconque de la forme $\varepsilon = (x_E; y_E; z_E; \phi; \theta; \psi; V)^T$, le drone puisse se retrouver dans cet état ; l'intérêt étant que l'on veut s'autoriser n'importe quelle attitude et position, et des vitesses raisonnables pour un drone (de l'ordre du $m.s^{-1}$).

En particulier, au vu de la problématique générale et en vue de la possible utilisation opérationnelle de notre système, on s'attachera à emmener le drone de l'état $\varepsilon_0 = (0; 0; 0; 0; 0; 0)^T$ à l'état $\varepsilon_1 = (1; 1; -1; 0; 120 \cdot \frac{\pi}{180}; 0; 4)^T$

Chapitre 5

Mise en équation initiale

5.1 Modélisation du drone et détermination des constantes

Afin de modéliser le drone, on considère que le centre de gravité G et les points d'application des forces sont dans le même plan ($G, \vec{x}_d; \vec{y}_d$).

De plus, on suppose que le rotor 3 pivote sur l'axe $G\vec{x}_d$.

Afin de modéliser la poussée et la traînée de chaque hélice, on utilise le modèle classique stipulant :

$$P_i = b\omega_i^2 \quad \text{et} \quad T_i = d\omega_i^2$$

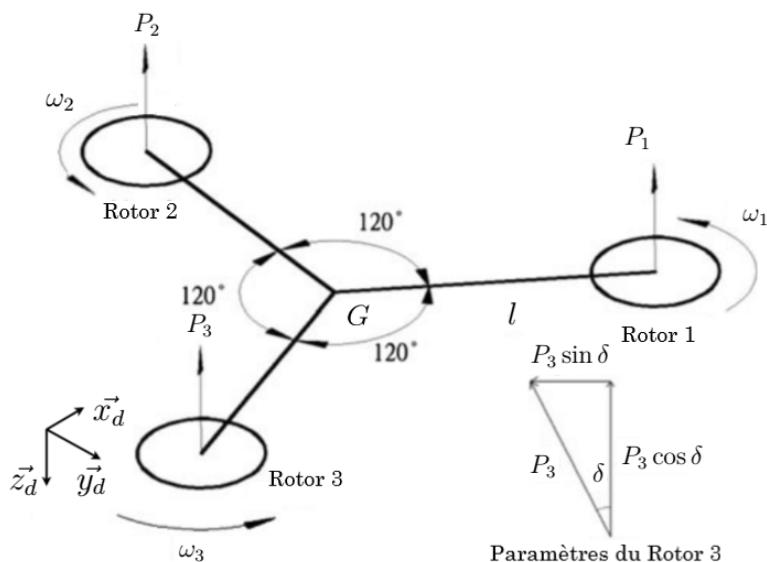


FIGURE 5.1 – Modélisation du drone

Pour calculer les moments d'inertie, on va décomposer notre drone en éléments simples : trois cylindres pour les moteurs, et un parallélépipède pour le centre du drone contenant tous les éléments. On néglige aussi le poids des tiges connectant les moteurs. C'est une modélisation simpliste, mais qui a l'avantage de garder la matrice d'inertie diagonale.

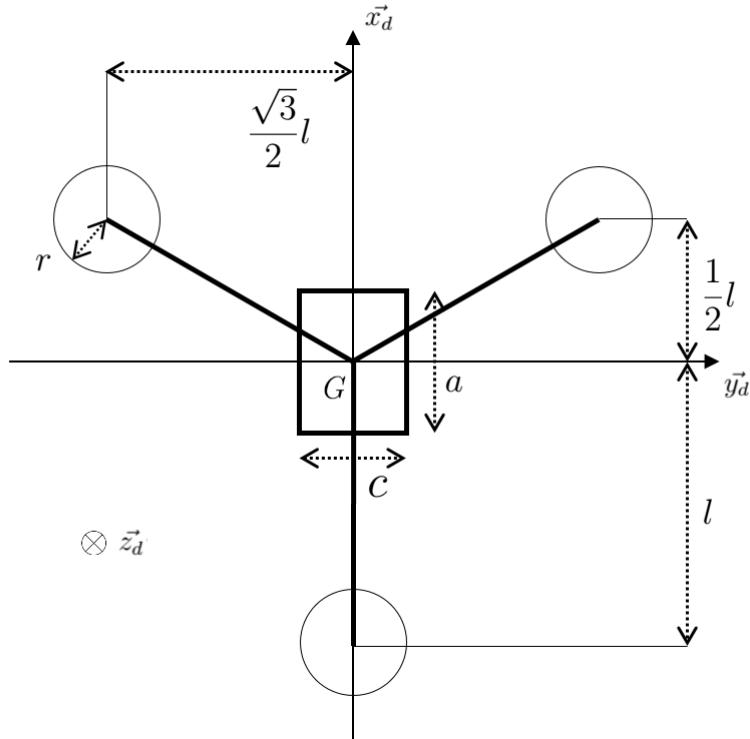


FIGURE 5.2 – Modélisation du drone, vue de dessus

On pose aussi h la hauteur du parallélépipède central et des cylindres qui modélisent les hélices, m_{mot} la masse de l'ensemble hélice - moteur - cadre autour de l'hélice et m_0 la masse du parallélépipède central que l'on suppose homogène. En utilisant le théorème de Huygens [9], on détermine les moments d'inertie sur les trois axes principaux.

$$\begin{aligned} I_{xx} &= \frac{m_0}{12}(c^2 + h^2) + m_{mot}(r^2 + \frac{h^2}{3} + \frac{7l^2}{4}) \\ I_{yy} &= \frac{m_0}{12}(a^2 + h^2) + \frac{m_{mot}}{2}(3r^2 + l^2) \\ I_{zz} &= \frac{m_0}{12}(a^2 + c^2) + \frac{m_{mot}}{12}(3r^2 + h^2 + 36l^2) \end{aligned}$$

On pèse et mesure le drone, ce qui donne les paramètres suivants :

Paramètres	Valeur numérique
m	1.713 kg
b	0.0344 N.s ² .rad ⁻²
d	0.0017 N.m.s ² .rad ⁻²
l	0.4 m
a	0.2 m
c	0.08 m
h	0.03 m
r	0.07 m
m_0	1.113 kg
m_{mot}	0.2 kg
I_{xx}	0.058 kg.m ²
I_{yy}	0.021 kg.m ²
I_{zz}	0.97 kg.m ²

b et d ont été déduit des caractéristiques connues d'hélices semblables, nous n'avions pas de quoi construire un dispositif de mesure de poussée [10].

5.2 Calcul des efforts et moments du trirotor

On notera que tous ces efforts sont exprimés, sauf indication contraire, dans le repère du drone \mathcal{R}_d .

- Poids :

$$\vec{g}_{\mathcal{R}_E} = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \Rightarrow \vec{g} = T_{Ed} \vec{g} = \begin{pmatrix} -mg \sin \theta \\ mg \sin \phi \cos \theta \\ mg \cos \phi \cos \theta \end{pmatrix} \quad (5.1)$$

- Efforts dû aux hélices :

$$P = b(\omega_1^2 + \omega_2^2 + \omega_3^2 \cos \delta) \quad (5.2)$$

avec :

$$\overrightarrow{P} = -P \vec{z}_d$$

On a aussi une force latérale parasite dû au rotor 3 :

$$\overrightarrow{F_{parasite}} = -b\omega_3^2 \sin \delta \vec{y}_d \quad (5.3)$$

- Trainée :

$$\vec{D} = \begin{pmatrix} -k_u u |u| \\ -k_v v |v| \\ -k_w w |w| \end{pmatrix} \quad (5.4)$$

Néanmoins, on va négliger la trainée dans les équations afin de simplifier le problème. On l'implémentera néanmoins dans le modèle afin de pouvoir la rajouter au besoin.

En additionnant les efforts projeté dans le repère attaché au drone, on obtient :

$$\overrightarrow{F_{x_d}} = -mg \sin \theta \vec{x}_d \quad (5.5)$$

$$\overrightarrow{F_{y_d}} = (mg \sin \phi \cos \theta - b\omega_3^2 \sin \delta) \vec{y}_d \quad (5.6)$$

$$\overrightarrow{F_{z_d}} = (mg \cos \phi \cos \theta - b(\omega_1^2 + \omega_2^2 + \omega_3^2 \cos \delta)) \vec{z}_d \quad (5.7)$$

Ensuite, on détermine les moments de roulis, tangage et lacet :

$$L = \frac{\sqrt{3}}{2} bl(\omega_2^2 - \omega_1^2) \quad (5.8)$$

$$M = bl\left(\frac{1}{2}(\omega_2^2 + \omega_1^2) - \omega_3^2 \cos \delta\right) \quad (5.9)$$

$$N = d(\omega_2^2 + \omega_1^2 + \omega_3^2 \cos \delta) - bl\omega_3^2 \sin \delta \quad (5.10)$$

Ces équations représentent le groupe moto-propulsif du drone. À ce stade, on peut déjà constater que ces équations sont bien plus complexes et intriquées que les équations usuelles d'un quadrirotor. En effet, il est bien plus ardu de trouver la combinaison de commandes permettant de n'agir que sur un seul paramètre du drone (roulis, tangage, lacet ou poussée).

Pour rappel, voici les expressions usuelles des efforts et moments pour un quadrirotor X usuel.

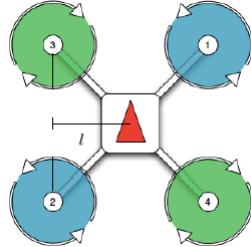


FIGURE 5.3 – Modèle de quadrirotor en X - configuration *props-in*

$$\vec{F}_{x_d} = -mg \sin \theta \vec{x}_d$$

$$\vec{F}_{y_d} = mg \sin \phi \cos \theta \vec{y}_d$$

$$\vec{F}_{z_d} = (mg \cos \phi \cos \theta - b \sum_{i=1}^4 \omega_i^2) \vec{z}_d$$

$$L = bl(\omega_3^2 + \omega_2^2 - \omega_1^2 - \omega_4^2)$$

$$M = bl(\omega_3^2 + \omega_1^2 - \omega_2^2 - \omega_4^2)$$

$$N = d(\omega_1^2 + \omega_2^2 - \omega_3^2 - \omega_4^2)$$

On voit qu'il est bien plus simple de ne contrôler qu'un seul moment sans modifier les autres (Par exemple, on peut augmenter M en imposant $\omega_4^2 = \omega_1^2$ et $\omega_3^2 = \omega_2^2$; ainsi L et N resteront constants).

5.3 Modèle de la plateforme à 6 degrés de liberté

Ces équations sont assez génériques pour ce type de problème, aussi on ne détaillera pas leur calcul.

- Dynamiques de rotation

$$\dot{p} = \frac{1}{I_{xx}}(L + (I_{yy} - I_{zz})qr) \quad (5.11)$$

$$\dot{q} = \frac{1}{I_{yy}}(M + (I_{zz} - I_{xx})rp) \quad (5.12)$$

$$\dot{r} = \frac{1}{I_{zz}}(N + (I_{xx} - I_{yy})pq) \quad (5.13)$$

- Cinématiques de rotation

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\ 0 & \cos \theta & -\sin \phi \\ 0 & \frac{\sin \phi}{\cos \theta} & \frac{\cos \phi}{\cos \theta} \end{pmatrix}}_{T_{euler}} \begin{pmatrix} p \\ q \\ r \end{pmatrix} \quad (5.14)$$

- Dynamiques de translation

$$\dot{u} = \frac{F_{x_d}}{m} - qw + rv \quad (5.15)$$

$$\dot{v} = \frac{F_{y_d}}{m} + pw - ru \quad (5.16)$$

$$\dot{w} = \frac{F_{z_d}}{m} - pv + qu \quad (5.17)$$

- Cinématiques de translation

$$\begin{pmatrix} \dot{x}_E \\ \dot{y}_E \\ \dot{z}_E \end{pmatrix} = \underbrace{\begin{pmatrix} \cos \theta \cos \psi & \sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi & \cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi \\ \cos \theta \sin \psi & \sin \phi \sin \theta \sin \psi + \cos \phi \cos \psi & \cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi \\ -\sin \theta & \sin \phi \cos \theta & \cos \phi \cos \theta \end{pmatrix}}_{T_{bE}} \begin{pmatrix} u \\ v \\ w \end{pmatrix} \quad (5.18)$$

5.4 Étude de l'équilibre

Afin d'étudier le vol stationnaire du drone, on pose les hypothèses suivantes :

- Les moments et forces qui s'exercent sur le drone sont nuls :
 $F_{x_d} = F_{y_d} = F_{z_d} = L = M = N = 0$
- Vitesses angulaires nulles : $p_e = q_e = r_e = 0$
- Trainée négligée
- Effets gyroscopiques et forces centrifuges négligées
- Les vitesses linéaires sont nulles : $u_e = v_e = w_e = 0$
- On a à l'équilibre $\omega_{1e}^2 = \omega_{2e}^2 = \omega_{3e}^2 \cos \delta_e = \omega_e$, d'où :

$$\boxed{\omega_{3e} = \frac{\omega_e}{\sqrt{\cos \delta_e}}} \quad (5.19)$$

En se plaçant à l'équilibre dans (5.10), il vient :

$$\begin{aligned} (5.10)_e &\Leftrightarrow 0 = d(\omega_{2e}^2 + \omega_{1e}^2 + \omega_{3e}^2 \cos \delta_e) - bl\omega_{3e}^2 \sin \delta_e \\ &\Leftrightarrow d(2\omega_e^2 + \frac{\omega_e^2}{\cos \delta_e} \cos \delta_e) = bl \frac{\omega_e^2}{\cos \delta_e} \sin \delta_e \\ &\Leftrightarrow 3d\omega_e^2 = bl\omega_e^2 \tan \delta_e \end{aligned}$$

Ainsi :

$$\boxed{\delta_e = \arctan \frac{3d}{bl}} \quad (5.20)$$

En utilisant (5.5) à l'équilibre, il vient immédiatement $\theta_e = 0$.

De même, en utilisant les deux autres équations résultant du principe fondamental de la dynamique (5.6) et (5.7), on a :

$$\begin{aligned} (5.7)_e &\Leftrightarrow 0 = mg \cos \phi_e \cos \theta_e - b(\omega_{1e}^2 + \omega_{2e}^2 + \omega_{3e}^2 \cos \delta_e) \\ &\Leftrightarrow mg \cos \phi_e = 3b\omega_e^2 \\ &\Leftrightarrow \cos \phi_e = \frac{3b\omega_e^2}{mg} \end{aligned} \quad (5.21)$$

$$\begin{aligned} (5.6)_e &\Leftrightarrow 0 = mg \sin \phi_e \cos \theta_e - b\omega_{3e}^2 \sin \delta_e \\ &\Leftrightarrow mg \sin \phi_e = b\omega_{3e}^2 \sin \delta_e \\ &\Leftrightarrow mg \sin \phi_e = b\omega_e^2 \tan \delta_e \\ &\Leftrightarrow \sin \phi_e = \frac{b\omega_e^2}{mg} \tan \delta_e \\ &\Leftrightarrow \sin \phi_e = \frac{\cos \phi_e}{3} \tan \delta_e \\ &\Leftrightarrow \tan \phi_e = \frac{1}{3} \tan \delta_e \end{aligned}$$

Ainsi :

$$\boxed{\phi_e = \arctan \frac{d}{bl}} \quad (5.22)$$

Enfin, en utilisant (5.21), on obtient :

$$\boxed{\omega_e = \sqrt{\frac{mg}{3b} \cos \phi_e}} \quad (5.23)$$

Voici donc les paramètres obtenus après application numérique :

Paramètres d'équilibre	Application numérique	Unité représentative
ω_{1e}	$9,7 \text{ rad.s}^{-1}$	583 RPM
ω_{2e}	$9,7 \text{ rad.s}^{-1}$	583 RPM
ω_{3e}	10 rad.s^{-1}	602 RPM
δ_e	$0,3588 \text{ rad}$	20.55 degrés
ϕ_e	$0,1244 \text{ rad}$	7.12 degrés

Ces résultats sont cohérents avec l'interprétation intuitive que l'on peut avoir.

En effet, à l'équilibre, si les 3 rotors tournent à la même vitesse pour compenser le poids du drone, on a un moment de lacet non nul. Cela mène à un angle δ positif, et donc à une augmentation de la vitesse du rotor 3.

On a donc une force latérale parasite selon $-\vec{z}_d$ que l'on doit compenser avec un angle ϕ positif.

On retrouve les conclusions apportées par les calculs.

En revanche, on voit bien une première limite de ce type de configuration de multi-rotors : on a une situation d'équilibre bien plus complexe que celui d'un drone plus usuel comme présenté figure 5.3. Pour ce type de drone, le vol stationnaire nécessite juste que les quatre rotors tournent à la même vitesse.

Deuxième partie

Établissement de lois de commande

Chapitre 6

Modélisation sous simulink

6.1 De la nécessité d'un modèle

La création d'un modèle simulink reprenant les équations décrite dans la partie précédente a été la première étape nécessaire à la réalisation du projet. Nous souhaitions un modèle sur lequel nous avions un contrôle total, permettant de linéariser des parties précises dans les équations au besoin. Par ailleurs, un modèle est dans notre cas absolument nécessaire. Tester chaque tentative de loi de commande sur le drone aurait été - d'abord - bien plus long et laborieux ; mais surtout très dangereux. Malgré les protections présentes autour des hélices il est très probable que, muni d'une loi de commande dysfonctionnelle, on aie perdu le contrôle du drone et qu'il soit allé s'écraser contre un équipement du laboratoire. Aussi, nous allons utiliser le schéma suivant [10] :

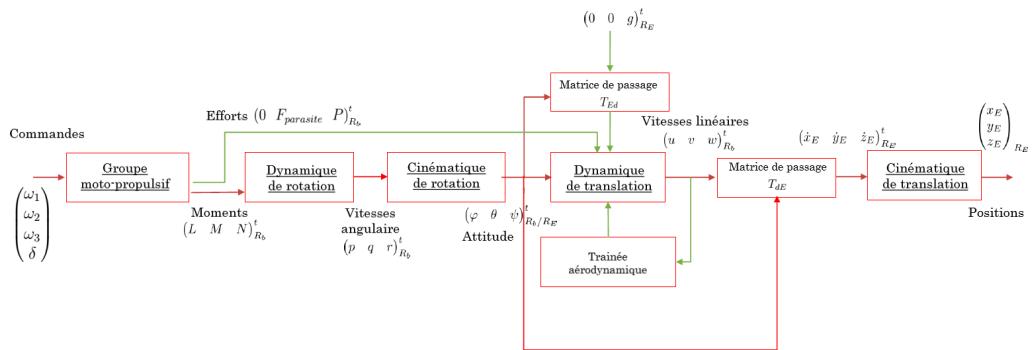


FIGURE 6.1 – Schéma fonctionnel du modèle de drone

Chacun des blocs reprends les équations détaillées précédemment, et sont présentés en Annexe 1 page 69.

6.2 Structure générale les lois de commandes

Maintenant, il s'agit de définir la structure des différentes lois de commandes que nous allons implémenter. Si elles sont toutes par essences différentes, on peut néanmoins créer un schéma global de fonctionnement. Ci-après une structure commune d'une chaîne de transmission de l'information permettant le pilotage.

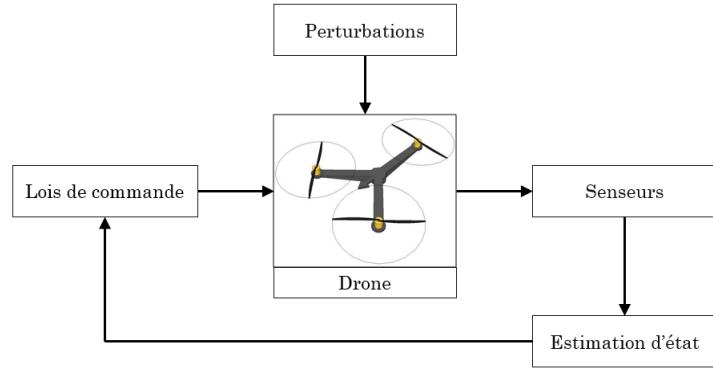


FIGURE 6.2 – Schéma global de transmission de l’information

Dans le reste de ce mémoire, on supposera que les senseurs et l’estimation d’état permet de fournir des mesures que l’on considérera parfaites par la suite.

Pour les lois de commandes en elles-même, on va séparer le bloc en deux : une boucle dite *intérieure* se chargera du maintien d’attitude du drone, et une seconde boucle dite *extérieure* se chargera de la navigation. On obtiens alors le schéma résumant la structure des lois de commandes :

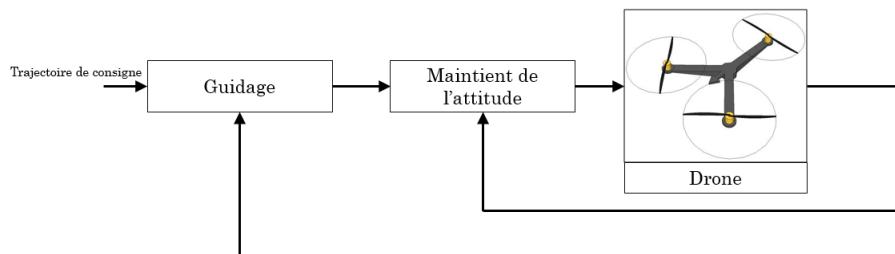


FIGURE 6.3 – Structure des lois de commandes

Aussi, dans un premier temps, nous allons nous intéresser à un modèle linéarisé de notre drone, pour des petites variations autour de son équilibre déterminé dans la partie 5.4. En effet, on peut supposer que le maintien d’attitude et la navigation ne nécessitent pas de grosses variations d’angles pour être efficaces.

De plus, la linéarisation des équations du drone nous permet d’étudier plus facilement son comportement. Les équations non linéaires sont très utiles pour la simulation, mais sont difficilement exploitables pour l’établissement de commandes.

Ce sera évidemment un problème lorsque nous implémenterons les lois pour les manœuvres plus techniques, mais nous souhaitons utiliser des lois de commandes conventionnelles avant d’en essayer d’autres, plus novatrices et qui se rapprochent de ce qui est utilisé dans les systèmes de pointe aujourd’hui.

Chapitre 7

Une loi de commande classique

7.1 Première simulation : boucle ouverte

Avant toute chose, on vérifie que notre modèle se comporte conformément à notre intuition et nos connaissances en physique. On réalise donc des tests de vol stationnaire en utilisant les paramètres déterminés dans la partie 5.4. Ensuite, on fait varier les commandes légèrement autour de cette valeur et on regarde le comportement du drone à l'aide du bloc *UAV animation* de la toolbox *UAV* de Simulink.

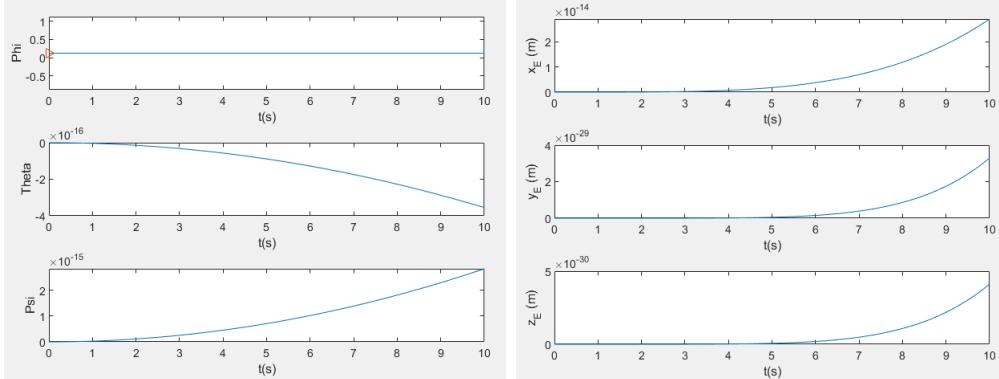


FIGURE 7.1 – Évolution des paramètres du drone en fonction du temps en vol stationnaire

Si les commandes à l'équilibre semble bien maintenir le drone, la boucle ouverte n'est pas satisfaisante puisque c'est un point d'équilibre instable (on voit très clairement une divergence). Il est donc nécessaire de créer un contrôleur d'attitude.

7.2 Linéarisation des Efforts

Afin d'étudier la mise en œuvre de lois de commandes classiques, on va linéariser les équations (5.2), (5.8), (5.9) et (5.10) au point d'équilibre. On notera par la suite \tilde{G} la grandeur associée à G autour du point d'équilibre telle que, au voisinage de ce point, $G = G_e + \tilde{G}$. Il vient, en utilisant la formule de Taylor pour une fonction de plusieurs variables - à savoir :

Pour toute fonction $f: \mathbb{R}^2 \rightarrow \mathbb{R}$ différentiable en $(a, b) \in \mathbb{R}^2$,

$$f(a + h, b + k) = f(a, b) + \frac{\partial f}{\partial x}(a, b)h + \frac{\partial f}{\partial y}(a, b)k + o(h + k)$$

Avec la convention $\frac{\partial f}{\partial x}$ la dérivée partielle selon la première variable et $\frac{\partial f}{\partial y}$ la dérivée partielle selon la seconde variable.

On utilise donc la formule de Taylor pour déterminer les moments linéarisés :

$$\tilde{L} = \sqrt{3}bl\omega_e(\tilde{\omega}_2 - \tilde{\omega}_1) \quad (7.1)$$

$$\tilde{M} = bl\omega_e(\tilde{\omega}_1 + \tilde{\omega}_2 - 2\sqrt{\cos \delta_e} \tilde{\omega}_3 + \omega_e \tan \delta_e \tilde{\delta}) \quad (7.2)$$

$$\begin{aligned} \tilde{N} = 2dl\omega_e(\tilde{\omega}_1 + \tilde{\omega}_2 + \sqrt{\cos \delta_e} \tilde{\omega}_3 - \frac{1}{2}\omega_e \tan \delta_e \tilde{\delta}) \\ - bl\omega_e(2\frac{\sin \delta_e}{\sqrt{\cos \delta_e}} \tilde{\omega}_3 + \omega_e \tilde{\delta}) \end{aligned} \quad (7.3)$$

Les autres équations linéarisées sont plus simples à déterminer, et donnent : les équations (5.11), (5.12), (5.13) et (5.14) ; tout en se plaçant à l'équilibre. On obtient :

$$\tilde{p} = \frac{\tilde{L}}{I_{xx}} \quad \tilde{\phi} = \tilde{p} \quad (7.4)$$

$$\tilde{q} = \frac{\tilde{M}}{I_{yy}} \quad \tilde{\theta} = \cos \phi_e \tilde{q} - \sin \phi_e \tilde{r} \quad (7.5)$$

$$\tilde{r} = \frac{\tilde{N}}{I_{zz}} \quad \tilde{\psi} = \sin \phi_e \tilde{q} + \cos \phi_e \tilde{r} \quad (7.6)$$

On va supposer dans un premier temps que ϕ_e est assez petit pour que l'on puisse négliger $\sin \phi_e$ devant $\cos \phi_e$, afin de découpler les équations (7.5) et (7.6).

On se retrouve donc avec des fonctions de transfert classiques pour ce type de système : un double intégrateur.

$$\frac{\tilde{\phi}}{\tilde{L}} = \frac{1}{s^2 I_{xx}} \quad \frac{\tilde{\theta}}{\tilde{M}} = \frac{\cos \phi_e}{s^2 I_{yy}} \quad \frac{\tilde{\psi}}{\tilde{N}} = \frac{\cos \phi_e}{s^2 I_{zz}}$$

Avec s la variable de Laplace.

On va pouvoir maintenant mettre en œuvre différentes lois de commandes et, au besoin, revenir sur des simplifications effectuées.

7.3 Contrôleur Proportionnel-Integral-Dérivée (PID)

Pour notre première loi de commande, on se propose d'utiliser sur chaque axe du drone un contrôleur de type PID. ce contrôleur conjugue trois actions de correction élémentaires :

- **Proportionnel**

Ce réglage introduit un couple de rappel vers la position de référence. Cependant, un mauvais réglage de k_p peut entraîner des conséquences néfastes sur le comportement du système. Trop faible, il sera sensible aux perturbations et lent pour revenir à l'équilibre. Trop élevé, il sera susceptible d'osciller autour de sa position de référence et même de diverger. Il faut donc introduire des "frottements".

- **Dérivé**

Ce réglage introduit un moment de frottement pour le système. Cet effet se manifeste lorsque le système est en mouvement et permet de réduire le risque d'instabilité introduit par le correcteur proportionnel. Le paramètre correspondant est k_d . Si il est trop faible, le retour à la position de référence sera sous le joug de l'effet proportionnel, donc susceptible d'être instable. S'il est trop élevé, le retour à la référence sera trop amorti et lent.

- **Integral**

Le paramètre k_i permet que, sous l'action d'une perturbation constante, le système puisse revenir à la référence. L'effet dérivé étant nul sans mouvement, si la perturbation est constante et écarte lentement le système de son équilibre, il est possible que l'action de la commande proportionnelle ne soit pas suffisante pour le ramener à la référence.

Pour chaque contrôleur, on doit déterminer 3 paramètres : k_p , k_i et k_d . Comme on a découplé chacun des axes, on utilise la toolbox de MATLAB sisotool (Single Input Single Output). On rentre les fonctions de transfert correspondantes et on détermine les paramètres nécessaires pour que le système soit rapide et précis en boucle fermée. Sans perte de généralité, on ne montre que le réglage du PID de l'axe de tangage.

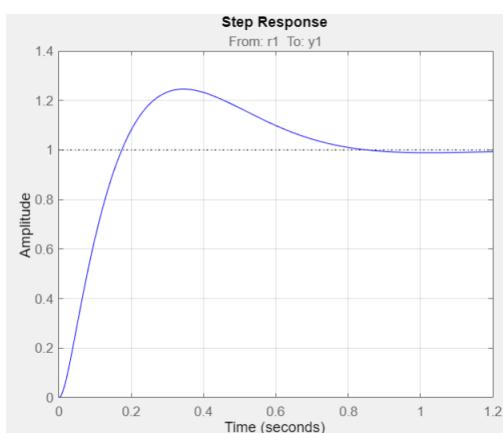


FIGURE 7.2 – Réponse du PID de l'axe de tangage à un échelon de commande

On observe un dépassement de 23%, mais on accepte ce point faible pour garder un temps de réponse faible. Une fois les trois contrôleurs mis en place, il nous reste à transformer les moments calculés en commandes en utilisant les équations (7.1), (7.2) et (7.3). Il s'agit d'exprimer les variations des commandes autour de l'équilibre $(\tilde{\omega}_1, \tilde{\omega}_2, \tilde{\omega}_3, \tilde{\delta})$ en fonction des moments $(\tilde{L}, \tilde{M}, \tilde{N})$. Comme il est illusoire de trouver une expression élégante, on effectue une résolution numérique sous *MATLAB* en imposant des contraintes pour obtenir une équation supplémentaire et ainsi pouvoir résoudre le système de manière unique.

Les valeurs trouvées pour le réglage des PID sont :

	Roulis	Tangage	Lacet
k_p	2.1213	2.1213	4.2426
k_i	2.2500	2.2500	9.0000
k_d	0.8700	0.3150	29.100

Néanmoins, dès que l'on passe au modèle réel non linéarisé, on obtient une erreur qui empêche le programme de compiler. L'erreur concerne un intégrateur qui sature, la simulation diverge.

Après quelques tentatives de rendre les contrôleurs moins agressifs pour éviter toute saturation, nous ne sommes pas parvenus à faire fonctionner une simulation. Nous allons donc devoir exclure ce type de contrôleurs.

7.4 Une nouvelle répartition des commandes du drone

Nous avons identifié, à la suite de ces résultats peu concluants, deux simplifications que nous pensons mauvaises :

- Supposer que $\sin \phi_e$ est négligeable devant $\cos \phi_e$
- Linéariser les expressions des moments

En effet, le découplage induit par la première hypothèse simplificatrice conduit à un schéma de lois de commandes beaucoup plus simple mais qui ne semble pas pertinent au vu de l'attitude du drone à l'équilibre. De plus, les expressions des moments sont très difficilement exploitables et conduisent à trop d'imprécisions. En effet, ces expressions doivent être "retournées" pour pouvoir exprimer les $\tilde{\omega}_i$ et $\tilde{\delta}$ en fonction des moments linéarisés, et ce en ajoutant un condition assez arbitraire reliant les différentes commandes.

On va donc procéder à une re-répartition des commandes. L'objectif est de faciliter la linéarisation, et de rendre l'action sur les contrôles similaire à ceux d'un hélicoptère. Ainsi, on pose :

$$u_m = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \delta \end{pmatrix}; \quad u_\beta = \begin{pmatrix} \beta_{coll} \\ \beta_{lat} \\ \beta_{longi} \\ \beta_{pied} \end{pmatrix}; \quad u = \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \cos \delta \\ \omega_3^2 \sin \delta \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix}$$

Avec :

$$\begin{aligned} \beta_{coll} &= -b(\omega_1^2 + \omega_2^2 + \omega_3^2 \cos \delta) \\ \beta_{lat} &= \frac{\sqrt{3}}{2} bl(\omega_2^2 - \omega_1^2) \\ \beta_{longi} &= bl\left(\frac{1}{2}(\omega_2^2 + \omega_1^2) - \omega_3^2 \cos \delta\right) \\ \beta_{pied} &= d(\omega_2^2 + \omega_1^2 + \omega_3^2 \cos \delta) - bl\omega_3^2 \sin \delta \end{aligned}$$

Et u_m le vecteur des commandes exploitables par le drone, u_β le vecteur des commandes, et u le vecteur des commandes intermédiaires qui servira pour simplifier les calculs.

On pose maintenant $M \in \mathcal{M}_4(\mathbb{R})$ tel que $u_\beta = Mu$, avec :

$$M = \begin{pmatrix} -b & -b & -b & 0 \\ -\frac{\sqrt{3}}{2}bl & \frac{-\sqrt{3}}{2}bl & 0 & 0 \\ \frac{bl}{2} & \frac{bl}{2} & -bl & 0 \\ d & d & d & -bl \end{pmatrix} \quad (7.7)$$

$M \in \mathbf{GL}_4(\mathbb{R})$, on a donc immédiatement $\tilde{u} = M^{-1}u_\beta$ avec $u = u_e + \tilde{u}$

Enfin, on a :

$$\omega_1 = \sqrt{u_1}; \quad \omega_2 = \sqrt{u_2}; \quad \omega_3 = \sqrt[4]{u_3^2 + u_4^2}; \quad \delta = \arctan \frac{u_4}{u_3}$$

On reprends les linéarisations réalisés en (7.4), (7.5) et (7.6) en remplaçant u_β dans les équations. On linéarise aussi (5.17) en suivant le même protocole. On travaillera donc avec ces équations :

$$\tilde{w} = -g \sin \phi_e \tilde{\phi} + \frac{\tilde{\beta}_{coll}}{m} \quad (7.8)$$

$$\tilde{p} = \frac{\tilde{\beta}_{lat}}{I_{xx}} \quad \tilde{\phi} = \tilde{p} \quad (7.9)$$

$$\tilde{q} = \frac{\tilde{\beta}_{longi}}{I_{yy}} \quad \tilde{\theta} = \cos \phi_e \tilde{q} - \sin \phi_e \tilde{r} \quad (7.10)$$

$$\tilde{r} = \frac{\tilde{\beta}_{pied}}{I_{zz}} \quad \tilde{\psi} = \sin \phi_e \tilde{q} + \cos \phi_e \tilde{r} \quad (7.11)$$

On peut ensuite réécrire notre système linéarisé pour le contrôle de l'attitude sous la forme d'une représentation d'état. X est le vecteur d'état, et Y est le vecteur de sortie. En l'occurrence, on suppose que $X = Y$.

$$\dot{X} = AX + Bu_\beta$$

$$\dot{Y} = CX + Du_\beta$$

Avec :

$$X = \begin{bmatrix} \tilde{w} \\ \tilde{\phi} \\ \tilde{\theta} \\ \tilde{\psi} \\ \tilde{p} \\ \tilde{q} \\ \tilde{r} \end{bmatrix} ; \quad A = \begin{bmatrix} 0 & -g \sin \phi_e & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cos \phi_e & -\sin \phi_e \\ 0 & 0 & 0 & 0 & 0 & \sin \phi_e & \cos \phi_e \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} ; \quad B = \begin{bmatrix} \frac{1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{I_{xx}} & 0 & 0 \\ 0 & 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix}$$

On pose aussi $C = \mathbf{I}_7$, $D = \mathbb{O}_{7 \times 4}$ puisque l'on a supposé que les variables d'état étaient parfaitement mesurées et sans action directe (*ie* : $X = Y$).

On note :

$$n = \dim(X)$$

$$p = \dim(u_\beta)$$

On va maintenant, en utilisant cette représentation d'état, implémenter des lois de commande plus sophistiquées.

Chapitre 8

Contrôle via représentation d'état

8.1 Principe d'une commande par retour d'état

Une commande par retour d'état se construit par un retour du vecteur de sortie multiplié par un gain K que l'on soustrait à la commande.

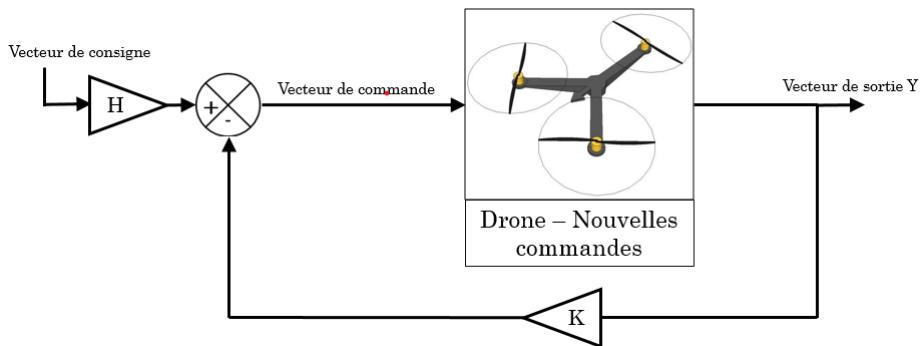


FIGURE 8.1 – Schéma fonctionnel d'une commande par retour d'état

Après l'ajout de ce retour, le système est régit par :

$$\dot{X} = (A - BK)X + BHe$$

avec :

$H \in \mathcal{M}_{n \times p}(\mathbb{R})$, la matrice de pré-commande

$K \in \mathcal{M}_n(\mathbb{R})$, la matrice de contre-réaction

$e \in \mathbb{R}^n$, le vecteur des consignes

$u_\beta = He - KX$ le vecteur des commandes, avec $u_\beta \in \mathbb{R}^p$

Le calcul du gain K est la pierre angulaire de ce type de correcteur. En effet, le comportement du système est lié aux racines de l'équation $|sI_n - A + BK| = 0$ et, par théorème, les pôles du système en boucle fermée sont précisément les racines décrites. Or, les pôles du système définissent son comportement. Ainsi, avec un K bien choisi, on pourra décider du comportement qu'adoptera le système [11].

Grâce à la fonction `place` de *MATLAB*, on peut obtenir immédiatement K étant donné A , B et la liste des pôles souhaités pour notre système. Quand à H , on le calcule aussi via *MATLAB* connaissant A , B , et K afin de ne pas avoir d'erreur statique (*i.e.* : pour un échelon unitaire en commande, la réponse sera rapide mais pas égale à 1). Néanmoins, si on a réduit le problème de contrôle à un choix de pôles pour notre système, ce choix est loin d'être évident.

On possède des règles et des moyens de calculer de bonnes approximations pour ces pôles sachant le comportement voulu, mais ce pour des systèmes de petite dimension. Ici, avec 7 états, on ne pourra vraisemblablement pas déterminer des pôles optimaux.

Ainsi, il nous faut trouver un autre moyen de régler un correcteur de ce type.

8.2 Le problème du régulateur quadratique linéaire

Nommé aussi LQR (Linear Quadratic Regulator), ce type de contrôleur propose un moyen de résoudre ce problème de placement de pôle en le transposant dans un problème d'optimisation. Le principe d'un LQR est, en reprenant le schéma 8.1, de déterminer K à partir de conditions sur les performances que l'on attend pour le système.

On cherche à calculer l'entrée qui minimise la fonction de coût suivante :

$$J = \int_0^{\infty} X(t)^T Q X(t) + u_{\beta}(t)^T R u_{\beta}(t) dt \quad (8.1)$$

Avec :

$$u_{\beta} = -KX$$

$$Q \in \mathcal{M}_n(\mathbb{R})$$

$$R \in \mathcal{M}_p(\mathbb{R})$$

Le calcul de K se réalise en résolvant l'équation algébrique de Riccati [12], qui ne sera pas détaillée dans ce mémoire. Nous utiliserons pour résoudre ce problème la fonction `lqr(A, B, Q, R)` de MATLAB.

Comme on peut le voir dans l'équation ci-dessus, Q pénalise les écarts entre les états du système et la consigne ; R pénalise quand à lui les commandes trop intenses et brutales. Q et R sont des matrices diagonales, dont les coefficients permettent de régler notre correcteur. Un coefficient élevé pour un état signifie que l'on souhaite une erreur faible pour ce dernier. Pour une commande, cela signifie que l'on ne veut pas de grosses amplitudes (on utilisera cela si une commande particulière consomme beaucoup d'énergie et n'est pas à privilégier) [13].

Q et R seront de la forme :

$$Q = \begin{bmatrix} d_1 & & 0 \\ & \ddots & \\ 0 & & d_r \end{bmatrix}$$

avec les coefficients $d_{[1; r]}$ les coefficients de pondération.

Afin de déterminer Q et R , nous nous sommes demandés quels états étaient capitaux pour la tenue d'attitude de notre drone, et quelles actionneurs étaient les plus énergivores.

Les angles de gîte ϕ et d'assiette θ sont bien plus importants que le cap ψ pour la tenue d'assiette, ils auront donc un poids équivalent. De plus, les vitesses angulaires ne sont pas capitales et découlent des angles d'attitude, elles auront donc un poids moins important. Pour w , on lui donne un poids intermédiaire.

Pour les actionneurs, les trois moteurs auront le même poids, et le servomoteur un poids inférieur puisque sa consommation est quasiment négligeable face à ces derniers. Comme on ne contrôle pas directement les moteurs mais une combinaison de ces derniers, on choisit de privilégier l'action de β_{pied} par rapport aux autres, puisqu'elle utilise majoritairement le servomoteur comparée aux autres actions.

Enfin, on souhaite favoriser la tenue d'attitude face à la consommation des actionneurs,

les coefficients de Q seront donc plus importants que ceux de R.

On choisit :

$$Q = \text{diag}(2.5; 5; 5; 2.5; 1; 1; 0.5)$$

$$R = \text{diag}(0.1; 0.1; 0.1; 0.01)$$

avec, pour rappel :

$$X^T = [\tilde{w} \quad \tilde{\phi} \quad \tilde{\theta} \quad \tilde{\psi} \quad \tilde{p} \quad \tilde{q} \quad \tilde{r}]$$

$$u_\beta^T = [\beta_{coll} \quad \beta_{lat} \quad \beta_{longi} \quad \beta_{pied}]$$

8.3 Résultat du LQR

Après calcul via *MATLAB*, on trouve :

$$K = \begin{pmatrix} 4.9941 & -0.8687 & 0 & 0 & -0.0141 & 0 & 0 \\ -0.2426 & 7.1665 & 0 & 0 & 3.2911 & 0 & 0 \\ 0 & 0 & 7.0180 & 0.6114 & 0 & 3.2087 & -0.0235 \\ 0 & 0 & -2.7342 & 15.6927 & 0 & -0.0051 & 8.9923 \end{pmatrix}$$

$$H = \begin{pmatrix} 4.9941 & 0.3481 & 0 & 0 \\ -0.2426 & 7.1665 & 0 & 0 \\ 0 & 0 & 7.0180 & 0.6114 \\ 0 & 0 & -2.7342 & 15.6927 \end{pmatrix}$$

Une fois le correcteur mis en place, on effectue un test de vol stationnaire sans perturbation : Pour la suite, le triangle rouge sur l'axe des abscisses indique la valeur de ϕ_e .

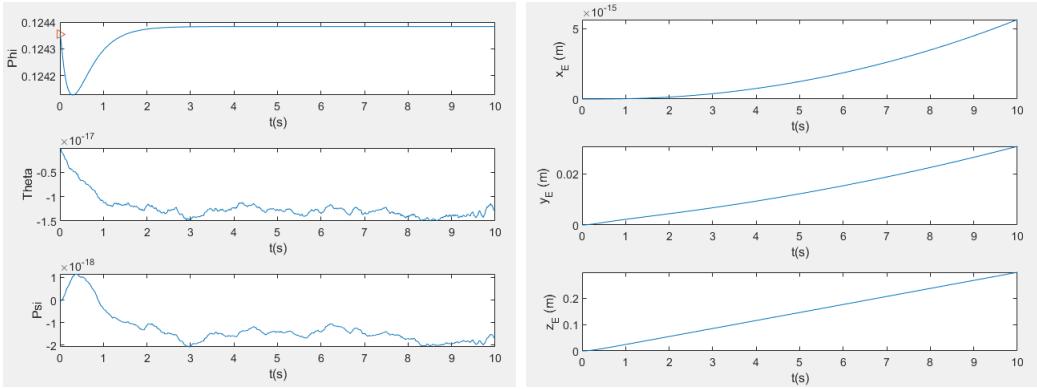


FIGURE 8.2 – Attitude et position du drone en boucle fermée

Le vol stationnaire est maintenu, et les légères déviations de la position sont dues à l'inertie du drone, puisque nous travaillons sans frottements. La position sera de toute manière amenée à être gérée par un autre contrôleur.

On passe maintenant au test de réjection de perturbation : sans perte de généralité, on applique un moment d'amplitude $A = 4 \text{ N.m}$ sur l'axe de roulis. La réponse sur les autres paramètres est similaire.

On observe un temps de réponse à 5% de 1.38 s et sans dépassement. Pour l'instant, cela nous convient ; on pourra au besoin modifier R et Q si le contrôleur n'est pas assez rapide pour nos applications de navigation. Comme le test de rejet de perturbation est concluant, on passe au suivi de consigne. Les paramètres sont :

$$w_{ref} = -0.6 \text{ m.s}^{-1} \text{ à } t = 3 \text{ s}$$

$$\phi_{ref} = \phi_e - 0.2 \text{ rad à } t = 6 \text{ s}$$

$$\theta_{ref} = 0.3 \text{ rad à } t = 1 \text{ s}$$

$$\psi_{ref} = \pi/2 \text{ rad à } t = 4 \text{ s}$$

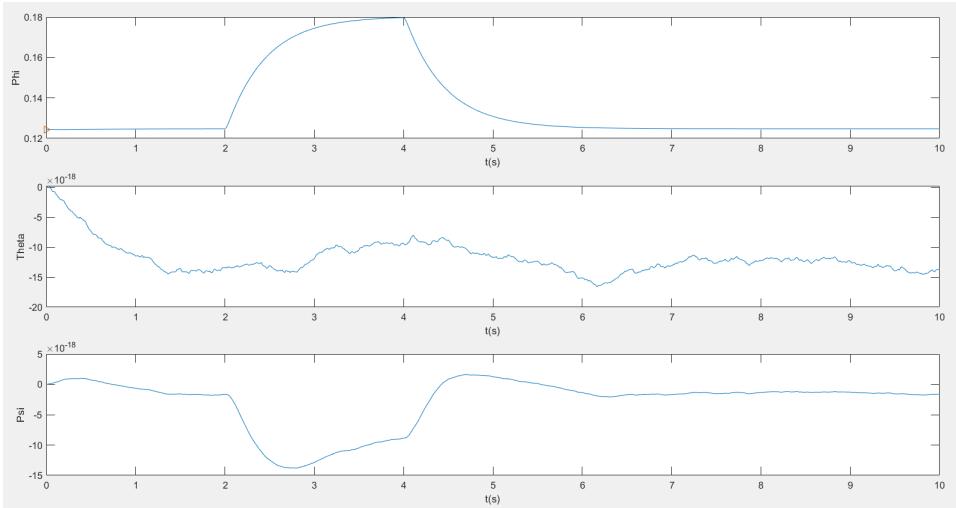


FIGURE 8.3 – Attitude du drone en réponse à une perturbation sur l’axe de roulis

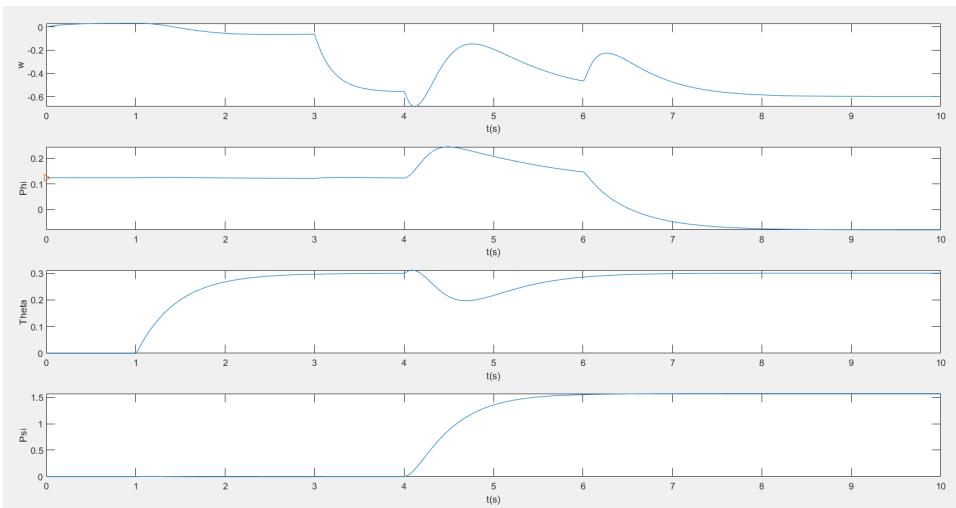


FIGURE 8.4 – Attitude et vitesse verticale du drone en réponse à la consigne

On observe deux choses sur ce graphe : les références fixées sont bien tenues, et on a une conjugaison importante entre w et ϕ d’un côté; et entre θ et ψ de l’autre. Cela se voit à 4 s sur la graphe de θ , la commande sur ψ a eu un impact sur θ (de même pour w et ϕ , l’effet est d’ailleurs plus prononcé). Une autre façon de le voir est en analysant K et H . Les colonnes de ces deux matrices correspondent aux composantes de u_β , et on voit que l’on pourrait les décomposer en matrice par bloc avec soit des matrices liant w et ϕ soit des matrices liant θ et ψ .

Néanmoins, si ce contrôleur serait tout à fait adapté pour un drone cinéma où l’on a besoin d’une très bonne stabilité et où les angles de travail restent faible : ce n’est pas notre cas. Dès que l’on dépasse une commande de plus de 1.12 rad pour θ ou ϕ , la linéarisation ne tient plus et le programme s’arrête. Pour déterminer l’angle ϕ limite, on lance une simulation avec pour commande une rampe de pente 0.1 rad.s^{-1} .

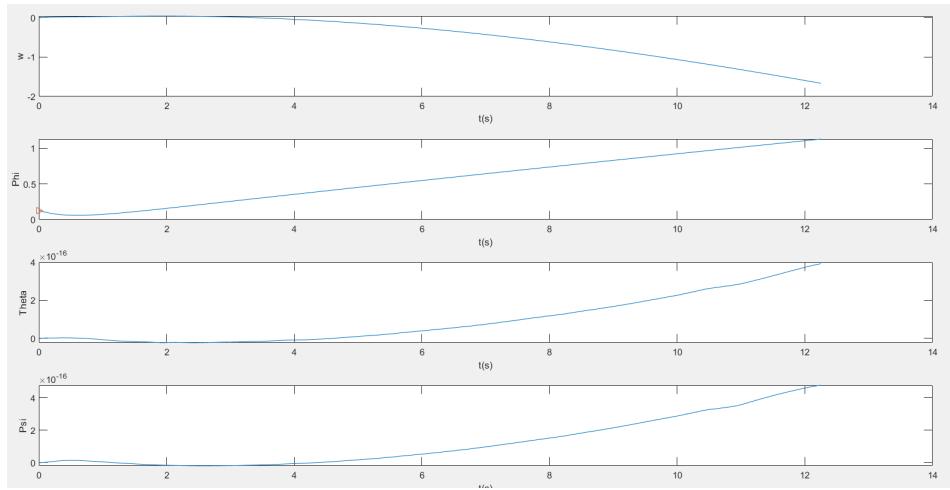


FIGURE 8.5 – Détermination de ϕ_{lim}

De plus, on observe que les autres paramètres se dégradent au fur et à mesure que ϕ augmente, faisant sortir le drone de la zone où la linéarisation est valable.

Au vu des types d'évolution que l'on veut faire effectuer à notre drone, nous allons tenter un autre type de contrôleur qui ne nécessite pas nécessairement de linéariser le modèle.

Chapitre 9

Utilisation d'un Model Predictive Controller (MPC)

9.1 Préliminaires

Il n'est pas rare que les ingénieurs fassent du bio-mimétisme pour créer de nouvelles machines ou de nouvelles procédures. Certaines ont eu très peu de succès, comme les ornithoptère [14], créées avec pour ambition d'être les premières machines volantes. On retrouve maintenant du bio-mimétisme dans des procédures : les algorithmes d'optimisation en colonie de fourmi en sont un bon exemple [15].

Le MPC, quand à lui, s'inspire du raisonnement humain. En effet, avec l'expérience, on acquiert une très bonne idée des conséquences de nos actions sur un système connu. Prenons l'exemple d'une voiture : un sujet qui conduit depuis longtemps saura avec une bonne précision la conséquence d'une pression sur l'accélérateur ou d'un coup de volant à gauche. On peut considérer qu'il a une sorte de *modèle* dans son esprit qu'il peut manipuler à loisir pour anticiper les conséquences de ses actions.

C'est - dans les grandes lignes - le fonctionnement d'un MPC. Pourvu qu'on lui fournisse un bon modèle, ce contrôleur va résoudre un problème d'optimisation à temps fini en utilisant ce dernier afin de réaliser la tâche imposée, ici le suivi d'un signal de référence. Le principal avantage de cette approche, c'est que ce contrôleur présente une multitude de paramètres permettant au concepteur de s'adapter à tous types de problèmes - notamment, l'utilisation de contraintes. En effet, on peut imposer au contrôleur des limitations dans sa recherche de solution, correspondant aux limitations physiques de notre matériel. On peut aussi lui imposer des contraintes dites "douces", qu'il tentera de ne pas franchir mais le fera pour un besoin impératif (par exemple, une limitation de vitesse pourra être dépassée pour éviter un accident).

Un MPC prends les paramètres suivants en entrée :

T_s : Le temps d'échantillonnage

C'est le rythme auquel le contrôleur va modifier les commandes et, surtout, résoudre le problème d'optimisation. Pour le déterminer, une bonne règle est de prendre pour référence le temps de rejet t_r d'une perturbation de notre système et de poser $\frac{t_r}{20} < T_s <$

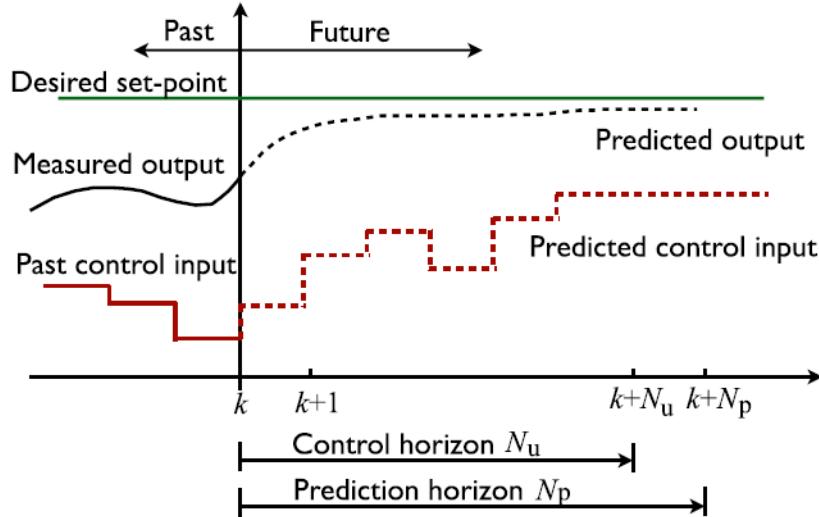


FIGURE 9.1 – Schéma basique de fonctionnement d'un MPC

$\frac{t_r}{10}$.

N_p : Horizon de prédiction

Cet entier naturel donne le nombre d'intervalles de temps sur lesquels on va utiliser le modèle pour prédire le comportement du système. Ce paramètre est utile pour anticiper des perturbations mesurées (dans notre exemple de voiture, un feu rouge au loin). Pour fixer ce paramètre, un minimum convenable est $N_p > \frac{T_{2\%}}{T_s}$ avec $T_{2\%}$ le temps de réponse à 2% (l'erreur entre le signal de référence et le signal est et reste inférieure à 2%). Ceci dit, un N_p trop élevé est inutile puisque, si l'on prédit trop loin, l'environnement a tout le temps de changer d'ici à ce que le système soit confronté à la perturbation.

N_u : Horizon de contrôle

Encore un entier naturel, inférieur à N_p , qui donne le nombre d'intervalle de temps sur lequel le contrôleur va calculer des commandes pour atteindre le plus efficacement son objectif. Si il est très grand, on va avoir une manœuvre plus fine, mais un temps de calcul beaucoup plus élevé. Par ailleurs, les premiers mouvements de correction sont en général les plus francs et les plus impactants, ceux qui suivent ont des effets plus restreints que l'on peut se permettre d'ignorer dans un premier temps (on viendra à créer des entrées de commandes pour ces corrections lorsqu'elles seront dans l'horizon de contrôle). On conseille généralement $N_u > 3$ et $0,1 \cdot N_p < N_u < 0,3 \cdot N_p$.

Contraintes

[Contraintes] : On en trouve deux types, des contraintes dites "dures" et d'autres dites "douces", qui peuvent porter aussi bien sur les commandes générées par le contrôleur que sur les variables de sortie. Les premières, comme évoqué un peu plus haut, sont des limites imposées au système que l'on ne peut dépasser (ce sont en général des limitations liées au matériel où une règle, comme s'arrêter devant un piéton). Les secondes sont des

limites que le système va tenter de respecter mais saura outrepasser en cas de besoin. Pour avoir une bonne convergence de la solution, il est préférable d'avoir des contraintes dures là où c'est absolument nécessaire.

Poids

Comme pour le LQR, on peut attribuer à chaque variable un poids qui indiquera à quel point on la favorise par rapport aux autres. Ainsi, on pourra favoriser le suivi de référence (avoir une réponse la plus rapide possible) face à des commandes progressives et sans à-coup. Cela allongera le temps de réponse, mais pourra éviter des comportements comme des oscillations.

Modèle

C'est le cœur du fonctionnement d'un MPC. On utilisera en général un modèle linéarisé autour d'un point d'utilisation bien choisi, mais il est possible de mettre en place d'autres méthodes de fonctionnement. En effet, si on s'éloigne trop du domaine où la linéarisation reste valable, le modèle ne tient plus et le programme se stoppe. On peut pallier à ce problème avec une variante : le MPC adaptatif (Adaptive MPC). En temps réel, il linéarise le système au voisinage de l'état où il se trouve, et ainsi possède toujours un modèle qui représente assez bien la réalité (bien que linéaire). Cependant, cela rajoute beaucoup de calculs si le système est hautement non-linéaire. On peut aussi utiliser un *gain-scheduled* MPC : on calcule des modèles linéarisés autour de plusieurs points, et on met en place un algorithme de sélection qui permet de choisir le modèle linéarisé qui est le plus adapté à l'état dans lequel se trouve le système.

Dans notre cas, cela va poser problème d'une part à cause du temps de calcul pour le MPC adaptatif et du nombre de modèles à réaliser pour le *gain-scheduled* MPC.

Enfin, il existe aussi le MPC non linéaire, utilisant le modèle non linéaire. Cela permet d'avoir les meilleures prédictions, et donc de générer les meilleures commandes. C'est néanmoins le plus difficile à utiliser en temps réel : pour un MPC linéaire, on a affaire à un problème d'optimisation dit convexe (il existe une seule solution optimale) ; mais pour un MPC non linéaire, on perd la convexité. Ainsi, il apparaît des extrema locaux qui donnent des solutions sous-optimales. Néanmoins, les MPC sont utilisés dans de nombreux systèmes très complexes [16], on peut donc légitimement essayer d'en implémenter un pour notre drone en dépit des limitations soulignées ci-dessus.

Ainsi, au vu de ces différentes caractéristiques, il semble que le MPC est adapté pour notre contrôle d'attitude [17]. On pourra établir des contraintes sur nos actionneurs et déterminer des taux des angles pour rester dans notre domaine linéarisé.

9.2 Détermination des paramètres du contrôleur

On pose les contraintes suivantes :

$$-\frac{\pi}{4} < \theta < \frac{\pi}{4} \quad \text{et} \quad -\frac{\pi}{4} + \phi_e < \phi < \frac{\pi}{4} + \phi_e$$

Ainsi, on restera à priori dans le domaine des petits angles et donc des nos limites de linéarisation. Comme à l'équilibre, le drone possède un angle ϕ non nul, on le retranscrit ici pour que $\tilde{\phi}$ soit bien centré dans son intervalle.

$$-2\pi < \psi < 2\pi$$

Le cap n'a pas d'influence sur la tenue d'attitude du drone dans notre modèle, on se laisse donc une plus grande marge de manœuvre.

$$0 < \omega_i < 2\omega_e$$

On suppose que la vitesse de rotation maximale est deux fois celle d'équilibre. Nous n'avons pas pu tester cette hypothèse sur le drone réel par soucis de batteries, mais c'est un paramètre facile à modifier.

$$-\frac{\pi}{2} < \delta < \frac{\pi}{2}$$

Cette équation retranscrit juste les limites structurelles du servomoteur.

Néanmoins, les entrées du MPC sont les composantes de u_β , qui sont une combinaison linéaire des ω_i et δ (cf. partie 7.4). On va donc déterminer les maxima et minima de ces composantes via un problème d'optimisation sous contrainte.

Pour ce faire, on utilise la fonction `fmincon` de *MATLAB*, qui résout le problème d'optimisation suivant :

$$\forall x \in \mathbb{R}, \min f(x) \text{ avec } \begin{cases} c(x) \leq 0 \\ c_{eq}(x) = 0 \\ A \cdot x \leq b \\ A_{eq} \cdot x \leq b_{eq} \\ x_{inf} \leq x \leq x_{sup} \end{cases}$$

Dans cette partie seulement, b et b_{eq} sont des vecteurs de \mathbb{R}^k , A et A_{eq} des matrices de $\mathfrak{M}_k(\mathbb{R})$, c et c_{eq} sont des fonctions à valeurs dans \mathbb{R}^k , f est un fonction à valeurs dans \mathbb{R} . Ces trois fonctions peuvent être non linéaires. k est le nombre de contraintes linéaires. Enfin, x_{inf} et x_{sup} sont les bornes inférieures et supérieures fixées pour x .

On utilise le code présenté en annexe page 72. Comme le processus d'optimisation dépend de l'un x_0 initial, on va itérer plusieurs fois le calcul afin d'obtenir presque sûrement le maximum global et ne pas rester dans un minimum local. Pour trouver les maxima, on utilise la même fonction `fmincon` mais sur $-f$.

Les résultats obtenus sont les suivants :

	β_{coll}	β_{lat}	β_{longi}	β_{pied}
minimum	-38.9370	-5.1916	-4.4961	-5.1916
maximum	0	5.1916	4.4961	6.5299
valeur à l'équilibre	-9.7342	0	0	5.5511e-17

On trouve une valeur négative pour β_{coll} , ce qui est logique puisque cette commande contrôle la poussée, orientée vers les z négatifs dans notre repère. Comme le MPC opère avec le vecteur d'état X défini dans la section 7.4, on travaille avec des écarts par rapport à l'équilibre, d'où l'utilité de calculer la valeur des composantes de u_β à l'équilibre.

Au vu du système étudié, on pose $T_s = 0.05$, $N_p = 20$, $N_u = 10$. On prends volontairement un horizon de contrôle grand pour éviter des phénomènes oscillatoires, d'autant plus que notre système est en équilibre instable.

On choisit maintenant les paramètres restants du MPC via le **MPC designer** disponible dans la *Model Predictive Control toolbox*, et attribuant les poids suivants :

Paramètre	β_{coll}	β_{lat}	β_{longi}	β_{pied}	\tilde{w}	$\tilde{\phi}$	$\tilde{\theta}$	$\tilde{\psi}$	\tilde{p}	\tilde{q}	\tilde{r}
Poids	$\frac{1}{\gamma}$	$\frac{1}{\gamma}$	$\frac{1}{\gamma}$	$\frac{1}{\gamma}$	γ	2γ	2γ	$\frac{\gamma}{2}$	$\frac{\gamma}{10}$	$\frac{\gamma}{10}$	$\frac{\gamma}{10}$

Où γ est un paramètre déterminé via le **MPC designer** pour avoir de bonnes performances. le curseur *State Estimation* ne nous est pas utile puisque nous avons supposé que nos

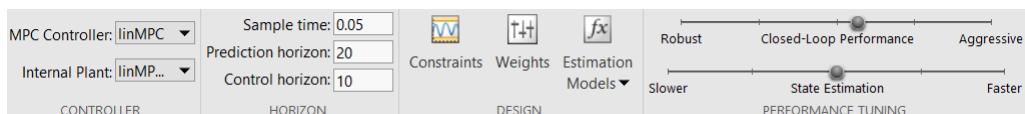


FIGURE 9.2 – Console de conception du MPC

mesures étaient parfaites, son réglage est donc superflu. Avec cet arrangement précis, on a $\gamma = 0.54881$. On pourra trouver en Annexe les résultats de simulations du MPC à la page 75.

9.3 Résultats du MPC

Maintenant que notre MPC est réglé et fonctionne en boucle fermée avec la représentation d'état, on va l'utiliser sur le système non linéarisé.

Tout d'abord, on vérifie qu'il permet bien de maintenir l'attitude du drone en l'absence de perturbations :

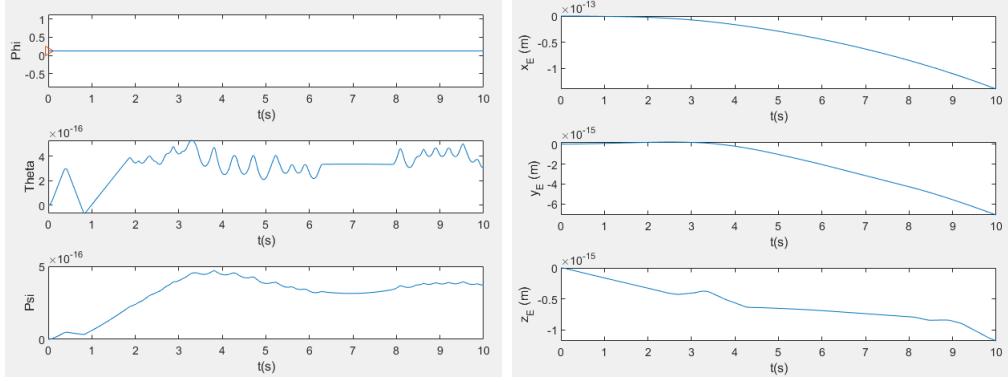


FIGURE 9.3 – Attitude et position du drone en boucle fermée

Les paramètres sont bien maintenus et ne divergent pas. On remarque une légère divergence de la position, mais qui est trop faible pour être problématique pour un contrôleur d'attitude. Elle sera en effet affectée par le contrôleur de position qui se trouvera sur la boucle extérieure, comme décrit sur la figure 6.3. On se concentre donc pour la suite uniquement sur l'attitude du drone.

On teste ensuite la capacité du MPC à rejeter une perturbation. On prends un moment de roulis appliqué pendant deux secondes d'amplitude $A_{perturbation} = 0.1 \text{ N.m}$, simulant une bourrasque :

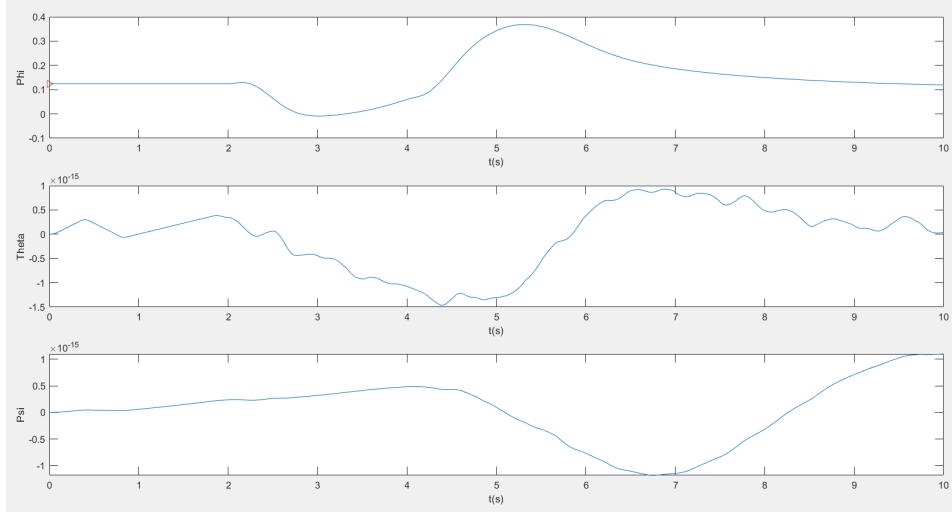


FIGURE 9.4 – Rejet d'une perturbation sur l'axe de roulis, pour $t \in [2, 4]s$

La perturbation est bien rejetée, puisque le gîte du drone ne diverge pas et retourne à sa valeur de référence une fois la perturbation passée. On observe que le gîte varie beaucoup, ce qui n'est pas un problème en soi au vu de la perturbation. Pour comparaison, voici le même test réalisé en boucle ouverte (donc sans contrôleur) :

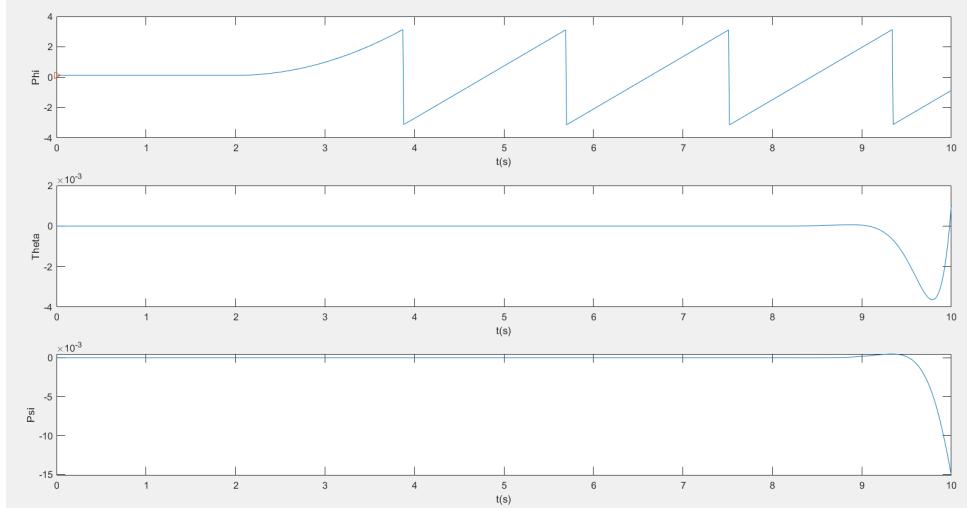


FIGURE 9.5 – Rejet d'une perturbation sur l'axe de roulis en BO, pour $t \in [2, 4]s$

Les discontinuités de la courbe de gîte proviennent du fait que les angles sont en radians, et donc définis dans l'intervalle $[-2\pi, 2\pi]$.

Enfin, on teste le suivi de consigne. On applique un échelon de consigne tel que $\phi_{ref} = \phi_e - 0.1 \text{ rad}$ à $t = 1 \text{ s}$ et $\theta_{ref} = -0.1 \text{ rad}$.

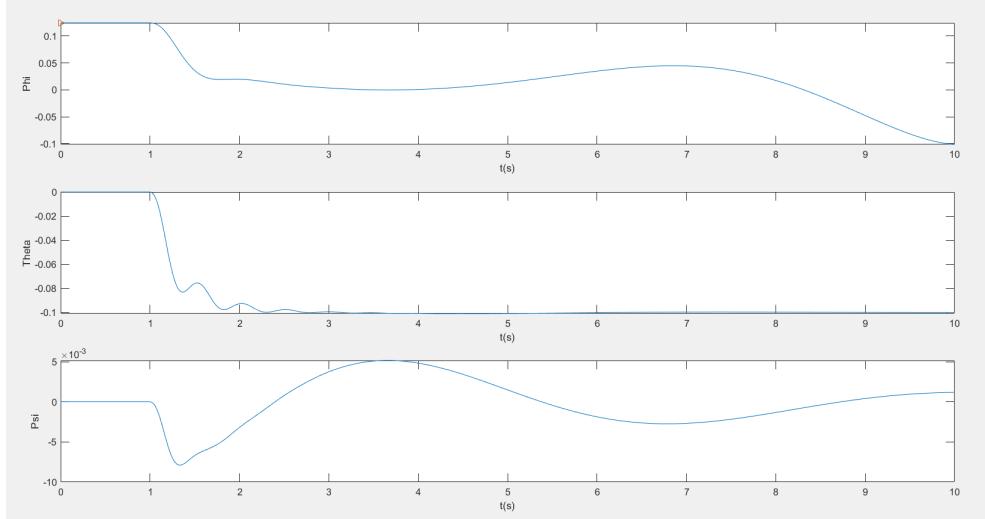


FIGURE 9.6 – Réponse à un échelon de consigne - $\phi_{ref} = \phi_e - 0.1 \text{ rad}$ et $\theta_{ref} = -0.1 \text{ rad}$

Pour θ , la référence est tenue mais est atteinte avec des oscillations indésirables. En revanche, la référence n'est pas tenue et on assiste même à des oscillations qui s'amplifient avec le temps pour ϕ . Cela se confirme si l'on augmente la durée de la simulation :

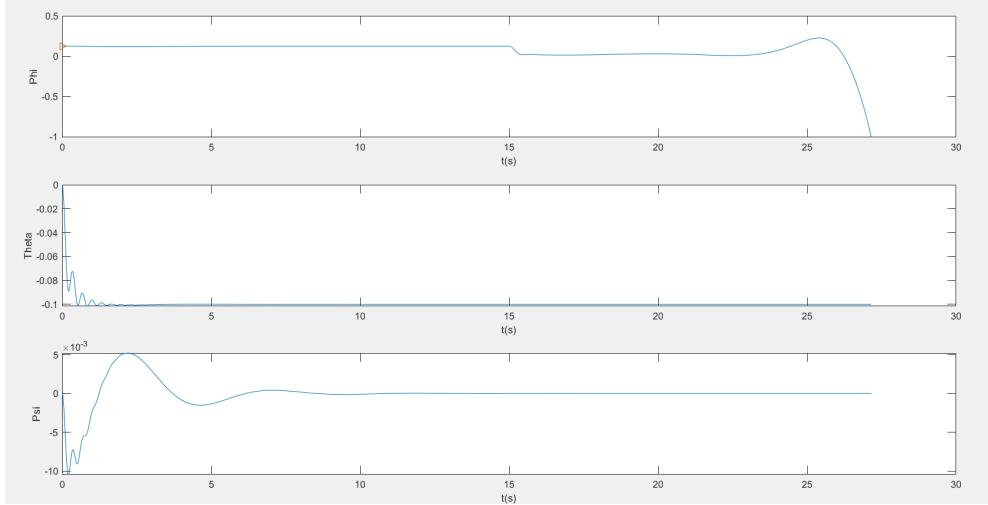


FIGURE 9.7 – Réponse à un échelon de consigne - $\phi_{ref} = \phi_e - 0.1 \text{ rad}$ et $\theta_{ref} = -0.1 \text{ rad}$

À $t = 27.5 \text{ s}$, le programme stoppe car un des intégrateurs sature à cause du gîte qui diverge. Ce n'est clairement pas satisfaisant. Nous avons tenté de changer les paramètres du MPC : le rendre moins agressif, changer les horizons de prédiction et de contrôle pour qu'il puisse anticiper plus loin. Rien n'y fait, on retrouve le comportement divergent de la simulation ci-dessus. Notons que nous avons espacé les deux échelons de consigne pour plus de visibilité des phénomènes. Ici, on a l'échelon de commande pour θ à $t = 0 \text{ s}$, et celui de ϕ à $t = 15 \text{ s}$.

Après beaucoup d'essais infructueux, nous en avons déduit que le problème devait venir de la représentation d'état. Malgré de la documentation très similaire sur le sujet [18], nous ne sommes parvenus à avoir des résultats acceptables avec le MPC.

Ceci dit, nous ne pouvons quand même pas retenir le régulateur LQR pour réaliser la *manœuvre* décrite page 16. En effet, utilisant une représentation d'état, nous ne pouvons l'exploiter que dans un domaine de vol ne permettant pas de grands angles, notamment. Il faut donc se tourner vers une autre manière de faire.

Chapitre 10

L'intelligence Artificielle : une solution miracle ?

10.1 Un domaine à la pointe

Le 11 Mai 1997, Garry Kasparov, Champion du monde d'échecs depuis 12 ans, s'incline face à Deep Blue. Ce cerveau de silice, développé par IBM au début des années 90, est le premier ordinateur parvenant à rivaliser avec le niveau humain dans un jeu jusqu'alors utilisé comme témoin de la vivacité d'esprit, voir de l'intelligence de ceux qui le pratiquent.

Progressivement, les algorithmes se sont fait de plus en plus perfectionnés, mais étaient chacun *un peu plus* performants que leurs prédecesseurs. Seulement, en 2016, une équipe de chercheurs en Intelligence Artificielle de DeepMind (une société rachetée par Google en 2015) parvient à mettre au point AlphaGo, battant successivement un très bon joueur puis le champion du monde de Go, un jeu ancestral japonais. Fin 2017, AlphaZero, une variante de ce programme adapté aux échecs, joue contre le programme le plus performant à l'époque : Stockfish 8. Avec seulement 4 heures d'entraînement et les règles du jeu d'échecs, AlphaZero a gagné 28 parties, égalisé 72, et n'en a perdu aucune. Un an plus tard, une revanche a été organisée sur 1000 parties. Les résultats sont tout aussi impressionnantes : 155 victoires, 839 nulles, 6 défaites [19].

DeepMind ne semble pas s'arrêter là : après des jeu de réflexion et à information totale (les deux joueurs ont accès aux mêmes informations qui sont - par ailleurs - les seules disponibles), ils sont parvenus fin 2019 à concourir contre les meilleurs joueurs du monde à Starcraft II avec le programme AlphaStar. Ce jeu est réputé comme étant extrêmement difficile à maîtriser, étant en temps réel et nécessitant de la prise de décision en fonction d'information seulement partielles de l'activité de son adversaire. En décembre 2020, ils révèlent finalement MuZero, un programme qui ne nécessite même pas de connaître les règles du jeu auquel il joue pour arriver aux niveaux de AlphaZero.
Enfin, si ces applications semblent bien superflues, un procédé similaire a permis des avancées phénoménales dans le problème du repliement des protéine grâce à AlphaFold [20].

La raison sous-jacente qui rend ces algorithmes si performants est l'utilisation de **l'apprentissage par renforcement**. Cette branche du machine learning s'oppose à deux autres types d'apprentissage, dits non supervisés et supervisés.

L'apprentissage non supervisé est utilisé pour reconnaître des motifs et structures, comme des corrélations, dans des bases de données très grandes et diverses (par exemple, étant donné une liste d'animaux et des caractéristiques précises, un programme d'apprentissage non supervisé pourrait les classer par espèce).

L'apprentissage supervisé est légèrement différent, et est principalement utilisé dans le reconnaissance d'image. Pour reprendre l'exemple précédent, un algorithme bien entraîné via de l'apprentissage supervisé à qui on présente une liste de caractéristiques animalière pourra fournir le nom de cet animal.

Le point commun entre ces deux approches est la nécessité d'une base de donnée gigantesque pour la phase d'entraînement. C'est un facteur déterminant et assez limitant : en effet, un algorithme entraîné de cette manière ne peut pas innover, puisqu'il cherche à coller le mieux possible aux données qu'on lui présente.

Ainsi, le type d'algorithme utilisé par DeepMind est bien différent puisqu'il est capable d'innover [21, 22].

C'est toute la force de l'apprentissage par renforcement, qui n'a pas fondamentalement besoin d'une base de donnée pour réaliser l'apprentissage. On peut s'en servir pour l'accélérer ou l'orienter, mais ce n'est pas capital.

C'est pour cela que ce type de machine learning est utilisé à tout-va par les chercheurs en IA depuis quelques années.

On va donc tenter de contrôler notre drone avec un algorithme utilisant l'apprentissage par renforcement [23].

10.2 Comment faire apprendre à apprendre ?

10.2.1 Prolégomènes

L'apprentissage par renforcement (Reinforcement Learning, abrégé RL pour le reste de ce mémoire) est un principe très général qui regroupe différentes manières d'appréhender le problème auquel on fait face.

Le principe fondateur d'un algorithme de RL est de trouver la séquence d'actions qui va générer la situation optimale. Ici, par situation optimale, on entend précisément "qui génère la récompense la plus importante".

On sépare ce qu'on appelle l'*agent* de l'*environnement*. Le premier correspond uniquement au programme en lui-même, et le second désigne tout le reste (ici : le drone et tout le monde physique autour de ce dernier). On nomme *état* les informations que l'agent reçoit : les pixels d'une image, des distances, des paramètres moteurs, ...etc.

Les *récompenses* sont ce qui va inciter l'agent à adopter le comportement que l'on souhaite, et sont distribuées par le concepteur. On peut donner des récompenses positives pour encourager des comportements que l'on souhaite, et des récompenses négatives pour "punir" l'agent d'une action indésirable.

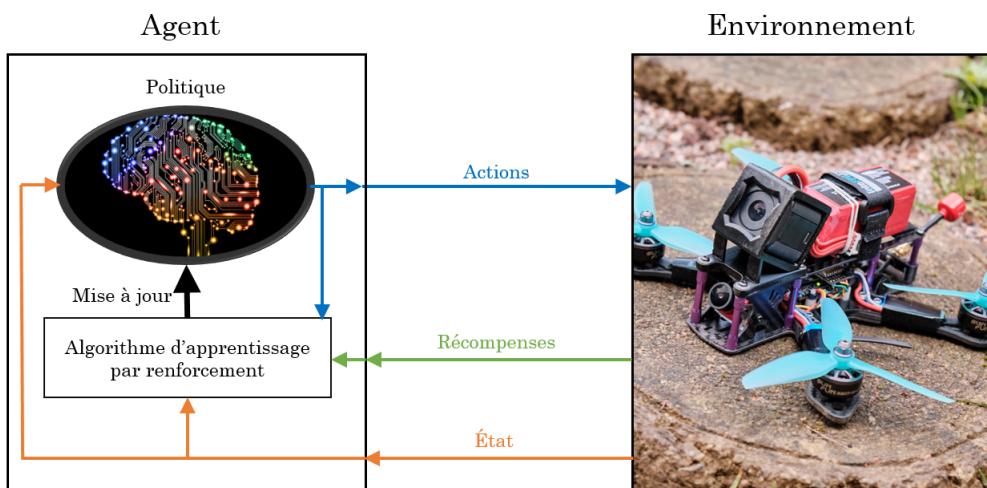


FIGURE 10.1 – Principe de fonctionnement d'un algorithme de RL

On distingue deux phases différentes dans l'entraînement d'un agent : l'*exploration*, où l'agent prend des décisions aléatoires ; et l'*exploitation*, où il prend des décisions conformément à sa politique.

Dans un état donné, l'agent va prendre une décision (en accord avec sa politique ou non), et selon la récompense qu'il va recevoir, il va mettre à jour sa politique pour favoriser - ou non - cette action s'il se retrouve dans ce même état. Déterminer quand arrêter l'exploration pour se concentrer sur l'exploitation est un des défis à relever pour entraîner un agent.

L'étape d'exploration est par ailleurs capitale, puisqu'elle va permettre à l'agent de po-

tentiellement découvrir des manières d’agir auxquelles le concepteur n’avait pas songé [24].

Si cela présente beaucoup d'avantages théoriques, cela constraint surtout le concepteur à créer une manière de distribuer les récompenses qui ne peut pas être "hackée" par l'agent. En effet, prenons l'exemple d'un robot aspirateur à qui on veut faire nettoyer une pièce. Pour ce faire, on définit la récompense comme la masse totale de "saletés" ramassées. L'agent (le programme régissant le robot) cherchera par tous les moyens à maximiser sa récompense. Au fur et à mesure de l'exploration, il pourra donc se rendre compte que renverser une poubelle lui donne une très grande masse de déchets à ramasser, et donc de récompense. Ce n'est bien évidemment pas ce que l'on attends d'un tel robot.

Définir la fonction de récompense est - on le voit - l'un des points les plus critiques lorsque l'on entraîne un agent. Si on ne récompense l'agent que lorsqu'il atteint un but final très précis, il est possible qu'il ne trouve jamais cet état escompté. On peut alors mettre des récompenses intermédiaires pour le guider, mais cela réduit alors sa capacité à innover et peut mener à un piratage de son circuit de récompense, comme évoqué plus haut.

10.2.2 Quelle politique adopter ?

La politique de l'agent peut prendre différentes formes : une fonction, une Q-table, ou encore un (ou plusieurs) réseaux de neurones.

Fonction

Si cette approche peut être la plus simple pour des exemples basiques, elle devient très vite impraticable dès que l'environnement est trop complexe. En effet, le concepteur devrait trouver une structure de fonction - par exemple : $\text{action} = c_1 \log(\sqrt{\text{états}}) + \frac{c_2}{\cosh(c_3 \cdot \text{états})}$ avec c_1, c_2, c_3 des paramètres libres ajustables par l'agent. Cela lui permettrait de trouver les bonnes actions étant donné les états. C'est évidemment quasiment impossible dès que l'on travaille avec des systèmes un tant soit peu complexes.

Q-table

Ici, on crée une grille $i \times j$ avec i le nombre d'états et j le nombre d'actions différentes. Chaque coefficient de cette table est interprété comme l'espérance du gain que l'on peut espérer, depuis un certain état, en réalisant une certaine action. Sans rentrer dans les détails, une Q-table est parfaitement adaptée aux environnements finis. En effet, pour des algorithmes de path planning sur des grilles de jeu, le nombre d'état est le nombre de case, et le nombre d'action est 4 pour les quatre directions cardinales. Discréteriser un problème continu est possible, mais deviens vite complexe, tout particulièrement quand on considère un drone évoluant dans l'espace avec 6 degrés de libertés.

Cela mène nécessairement à de trop compromis à faire entre précision de la discréétisation et temps de calcul, qui deviens très (trop) vite limitant dans ce cas précis. On souhaiterais éviter ce travers autant que faire se peut.

Réseau de neurone

Un réseau de neurone peut être vu comme un approximateur universel de fonction. Il existe des types très différents de réseaux de neurones, et ils sont utilisés dans tous types d'applications. Les mathématiques derrière les réseaux de neurones relèvent majoritairement de l'algèbre linéaire, et ne sont pas capitales ici. Néanmoins, il pourra être intéressant d'avoir une idée des objets que l'on manipule en visionnant [25]. Encore une fois, il faudra faire un compromis entre temps et impossibilité d'approximer notre fonction si le réseau de neurone n'est pas assez complexe, mais on garde la continuité des actionneurs. C'est donc cette dernière forme que nous allons sélectionner.

Ceci dit, il existe plusieurs manières de conditionner un réseau de neurone pour résoudre notre problème. Nous allons ici voir trois manières différentes d'en employer un.

10.2.3 Vers l'utilisation de réseaux de neurones

Policy Function

Policy étant le terme anglais pour désigner la politique de l'agent, cette méthode consiste à utiliser un réseau de neurone que l'on nomme *Acteur*, et de ne baser sa réponse que sur les observations qu'il fait de l'état de l'environnement. Un gros point fort d'utiliser un réseau de neurone en tant qu'acteur, c'est que les sorties sont des probabilités de faire telle ou telle action ; ainsi le dilemme exploration/exploitation est résolu par la structure même de l'acteur.

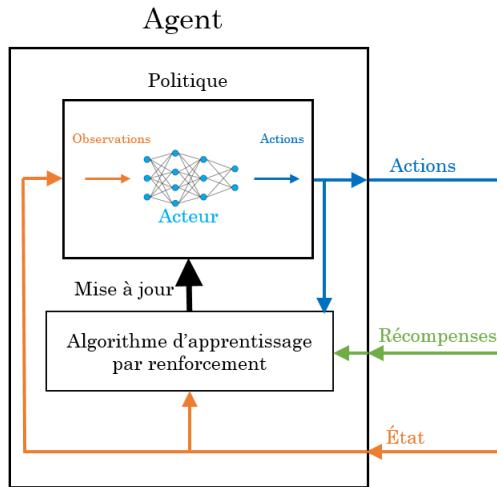


FIGURE 10.2 – Principe d'un algorithme basé sur une *Policy Function*

Le processus d'entraînement est le suivant : on fait appliquer la politique à l'agent, on collecte (ou pas) les récompenses, et selon les résultats on met à jour le réseau de neurone via une descente de gradient stochastique et l'algorithme de rétro-propagation [25]. Le principal problème à cette méthode, c'est que le temps d'entraînement peut être extrêmement long, surtout si les récompenses sont données occasionnellement ou uniquement lorsque l'agent atteint l'objectif voulu. De plus, il est très sensible aux extrema locaux, et on se retrouve trop souvent avec une solution sous-optimale. On pourrait régler ce problème par un procédé similaire à celui employé dans la partie 9.2. Néanmoins, un entraîner un réseau de neurone est long, et on n'a pas le loisir de le faire plusieurs dizaines de fois pour éviter un extremum local.

Value Function

Un point très positif du principe de la Q-table évoqué plus haut est que, en plus de prendre en compte l'état dans lequel est l'environnement, on regarde aussi l'action à effectuer. Cela permet d'éviter les problèmes des extrema locaux, et est assez rapide à entraîner. On peut considérer cette Q-Table comme un *critique*, qui jugerait les actions avant de les prendre pour tenter de prendre la meilleure (contrairement à l'acteur, qui n'a fondamentalement aucune idée de si son action est bonne ou pas en soi). Ainsi, on pourrait, en plus de l'état, fournir au critique les actions que l'on peut prendre et lui laisser évaluer la pertinence de chacune d'entre elles.

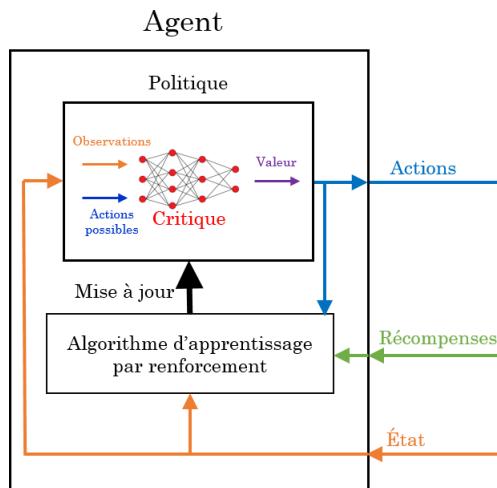


FIGURE 10.3 – Principe d'un algorithme basé sur une *Value Function*

Les valeurs en sortie du critique permettent de définir la politique : choisir - lors de la phase d'exploitation - l'action ayant obtenue la meilleure valeur (c'est à dire l'action qui maximise l'espérance de gain de l'agent).

Cependant, cette méthode présente un problème majeur : on ne peut pas examiner une infinité d'actions possibles. On retombe sur notre problème de discréétisation.

Actor - Critic

La base des classes d'algorithme d'apprentissage par renforcement dit Actor - Critic est de fusionner les deux méthodes précédentes.

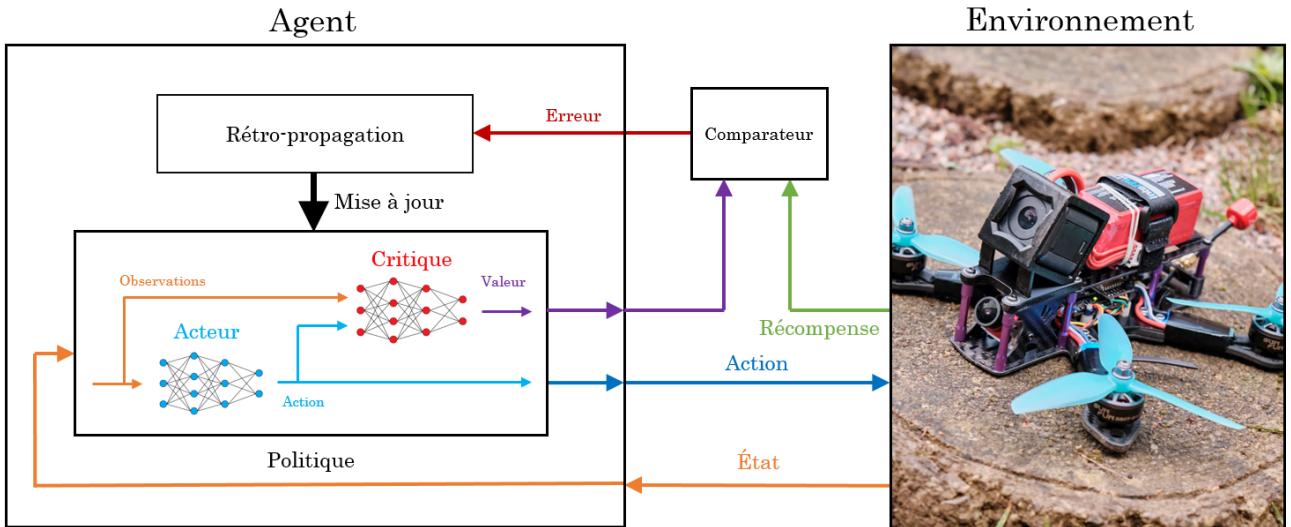


FIGURE 10.4 – Principe d'un algorithme de type Actor - Critic

On note une différence majeure : ce n'est plus la récompense en elle-même qui est analysée et utilisée pour modifier la politique, mais la comparaison entre la valeur qu'attribue le critique à l'action prise par l'acteur et la récompense réelle. Ainsi, la décision de l'acteur est éclairée par la valeur que le critique attribue à cette action.

Ce qui rend cette configuration aussi efficace c'est que le critique n'a qu'une seule action à analyser, ce qui permet de garder la continuité. On a (presque !) les avantages de chacune des méthodes sans les inconvénients. Presque, parce qu'il faut entraîner deux réseaux de neurones au lieu d'un. Il faudra donc bien choisir les réseaux de neurones étudiés, au risque d'avoir un temps d'entraînement trop important.

C'est donc cette méthode, cette classe d'algorithme, que l'on va tenter d'utiliser pour réaliser la manœuvre à notre drone.

10.3 Tentative d'implémentation

Pour implémenter cet algorithme, nous allons utiliser l'application *Deep Network Designer* de la Deep Learning Toolbox v14.2, ainsi que l'application *Reinforcement Learning Designer* de la Reinforcement Learning Toolbox v2.0.

La première servira à construire les différents réseaux de neurones nécessaires à notre algorithme ; la seconde servira à entraîner l'agent. Nous réaliserons cela dans l'environnement de *SIMULINK*.

Nous avons donc besoin de déterminer l'environnement. Ici, ce sera le drone. On souhaite amener le drone dans une certaine position et attitude de référence, on crée donc un vecteur \vec{O} de \mathbb{R}^6 contenant les valeurs de référence :

$$\vec{O} = [x_{Eref} \ y_{Eref} \ z_{Eref} \ \phi_{ref} \ \theta_{ref} \ \psi_{ref}]$$

Nous allons utiliser le même modèle que pour le reste de nos contrôleurs, mais utiliser comme observation le plus de signaux possibles. Le vecteur d'observation utilisé possède 47 composantes, à savoir :

- Les positions dans \mathbb{R}_E
- Les angles d'attitude
- Les vitesses de rotation
- Les vitesses dans \mathbb{R}_E
- Les vitesses dans \mathbb{R}_d
- Les moments appliqués sur le drone
- Les forces appliqués sur le drone
- Les vitesses de rotation des rotors et δ
- Le vecteur de commande u_β
- L'erreur entre \vec{O} et l'état du drone
- Les dérivées première et seconde de ces erreurs.

Il faut aussi créer une condition d'arrêt pour la simulation. On va considérer que, au-delà de 5 secondes de simulation, on a un échec. Au vu de la manœuvre que l'on s'est fixée, 5 secondes sont largement suffisantes et permettent d'éviter de bloquer la simulation. Pour rendre plus efficace l'apprentissage, on va aussi la stopper lorsque le drone dévie trop de la zone de travail. On va donc lui donner une sphère d'évolution au delà de laquelle il ne peut aller sans stopper la simulation et recevoir une récompense négative.

La zone de travail sera donc une sphère de rayon 1 m centré sur le point milieu entre la position initiale du drone - à savoir $[0 \ 0 \ 0]$ - et la position finale souhaitée : $[1 \ 1 \ -1]$. On aura donc comme autre condition d'arrêt :

$$(x_E - 1/2)^2 + (y_E - 1/2)^2 + (z_E + 1/2)^2 > 1 \quad (10.1)$$

Ensuite, on arrête la simulation si le drone parvient à se mettre dans un état suffisamment proche de \vec{O} . Ainsi, une autre condition d'arrêt sera :

$$(x_E - x_{Eref})^2 + (y_E - y_{Eref})^2 + (z_E - z_{Eref})^2 + (\phi - \phi_{ref})^2 + (\theta - \theta_{ref})^2 + (\psi - \psi_{ref})^2 \leq 0.1 \quad (10.2)$$

Enfin, on ne souhaite pas que le contrôleur envoie des commandes qui ne soient pas réalisables par nos moteurs et servo-moteur réels. La dernière condition d'arrêt est donc atteinte si une des commandes amène un rotor à tourner au-delà d'une certaine limite, ou impose un angle trop important au servo-moteur. La dernière condition d'arrêt sera donc :

$$\forall i \in [1; 3], \omega_i < 0 \text{ ou } \omega_i > 2\omega_e \quad \text{et} \quad |\delta| > \frac{\pi}{2} \quad (10.3)$$

On va maintenant créer le signal de récompense. Pour ce faire, on s'inspire de la littérature sur le sujet [26] pour éviter des comportements indésirables du drone. Notre fonction de récompense aura une partie continue, et une partie discontinue. Les parties composant cette récompense sont détaillées ci-dessous.

- Plus le drone s'approche de la position souhaitée, plus il est récompensé. On place donc le terme $\frac{1}{|x_E - x_{Eref}| + |y_E - y_{Eref}| + |z_E - z_{Eref}|}$ dans la récompense.
- On souhaite que le drone commence à prendre son attitude de référence au fur et à mesure qu'il se rapproche de la position de référence. Ainsi, nous allons ajouter le terme suivant dans la récompense :

$$\min \left(1, \frac{0.1}{|x_E - x_{Eref}| + |y_E - y_{Eref}| + |z_E - z_{Eref}|} \right) \cdot \frac{1}{|\phi - \phi_{ref}| + |\theta - \theta_{ref}| + |\psi - \psi_{ref}|}$$

Cette partie de la récompense vise à inciter le drone à commencer à modifier son assiette une fois qu'il se rapproche de la position de référence, et non pas de rester sur place et d'obtenir des récompenses en modifiant juste son assiette.

- Ensuite, on souhaite pénaliser des commandes trop agressives. On ajoute donc le terme suivant :

$$-0.01 * (\omega_1 + \omega_2 + \omega_3 + |\delta|)$$

Le fait que δ varie très peu nous arrange ici, puisque l'on veut (jusque dans un certain extrême) favoriser au maximum son utilisation.

- Enfin, on récompense de 20 si la simulation termine grâce à la condition (10.2); et on pénalise de 20 si elle se stoppe par un autre moyen.

On code donc ces fonctions sous *SIMULINK* (*cf.* Annexe 4 page 77).

Néanmoins, nous ne sommes pas parvenus à faire fonctionner le module de création de réseau de neurone. En effet, après avoir créé les objets dans la console comme suit, et avoir déterminé les différents paramètres, nous ne sommes pas parvenu à lier le bloc Agent du schéma *SIMULINK* à ces deux réseaux de neurone.

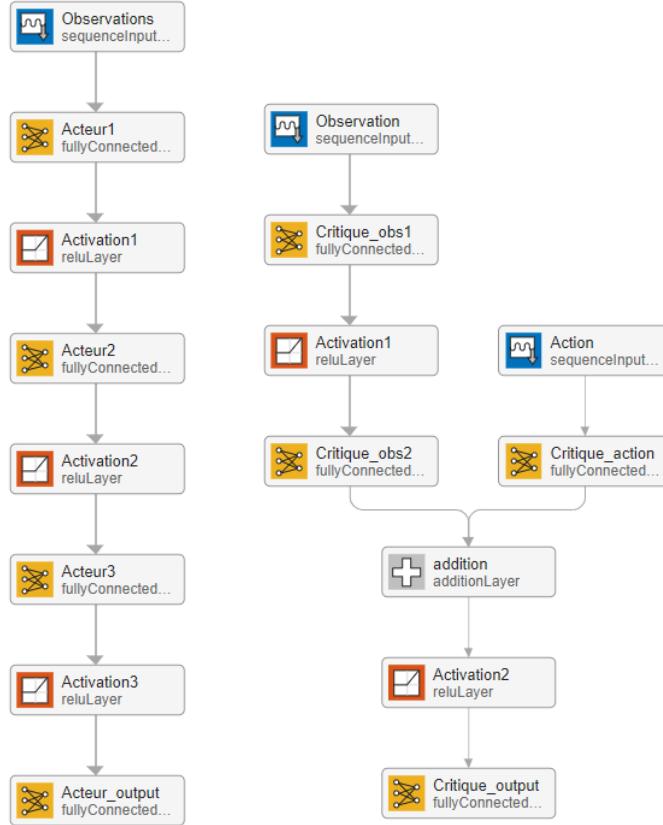


FIGURE 10.5 – Réseaux de neurones (resp. Acteur et Critique) créés pour l’algorithme de RN

Le temps nous manquant, nous n'avons pas pu continuer à essayer de faire fonctionner l'algorithme en se basant sur la documentation existante [27, 28, 29], comme nous l'avons fait pour créer les différentes fonction précédemment exposées.

Chapitre 11

Perspectives et conclusion

Lorsque nous avons commencé ce projet, nous avons décidé de tout reprendre depuis le début. Nous avons donc spécifié beaucoup d'élément théoriques et justifié nombres d'hypothèses simplificatrices.

Nous avons ensuite développé un modèle du drone sous *SIMULINK* le plus représentatif possible, afin de ne pas travailler avec une boite noire pour ce qui est des équations du drone.

Nous avons ensuite tenté d'appliquer des résultats assez classiques sur les quadrirotors à notre tricopter dans le chapitre 7. Nous nous sommes alors rendus compte que le drone avec lequel nous travaillions était moins simple à commander qu'un quadrirotor, et qu'il fallait développer des lois différentes que celles utilisées usuellement par les logiciels (comme Betaflight par exemple).

En s'inspirant de la littérature, nous avons donc ré-attribué les commandes du drone dans l'espoir de simplifier les lois de commande. Nous avons ensuite élaboré un contrôleur d'attitude en utilisant un régulateur Linéaire - Quadratique (LQR), qui est parfaitement utilisable pour piloter le drone.

À ce stade du stage, nous avons eu un choix : continuer à élaborer des contrôleurs toujours plus performants pour pouvoir s'approcher de notre objectif final de planification de trajectoire très complexe ; ou réaliser un algorithme de planification de trajectoire plus basique utilisant le LQR - tout en sachant que nous ne pourrions pas, à terme, valider l'objectif de manœuvrabilité très poussée du drone à cause de la linéarisation.

Nous avons donc fait le pari - risqué - d'ignorer la planification de trajectoire simple pour nous lancer dans des contrôleurs utilisés dans l'industrie de pointe.

Pour aborder le MPC, nous avons d'abord voulu utiliser un paradigme de fonctionnement plus simple pour appréhender les tenants et aboutissants de ce type de contrôleur avec un MPC Linéaire, dans l'espoir de mettre *in fine* au point un MPC non linéaire.

Cependant, les résultats du MPC ont été très décevants. Nous avons donc décidé de nous en tenir à notre pari, et de tenter de mettre en œuvre un algorithme de renforcement permettant de faire nos manœuvres. Nous avons explicité les points critiques dans la mise en œuvre d'un tel algorithme, à savoir l'environnement et le calcul de la récompense. Nous avons cependant manqué de temps pour y arriver à bout, le lecture de la documentation nous a pris beaucoup de temps malgré des connaissances préalables et les toolbox sont assez complexes à prendre en main. Il n'est aussi pas impossible que les ordinateurs à

notre disposition n'aient pas été assez puissant pour réaliser l'entraînement de l'agent en un temps raisonnable.

Si le stage avait duré plus longtemps, nous aurions continué à travailler sur l'algorithme d'apprentissage par renforcement. Nous aurions aussi tenté de mettre en place des MPC utilisant d'autres représentations d'état, puisque nous n'avons pas considéré la possibilité que la représentation d'état soit mauvaise.

Nos résultats permettent néanmoins à quelqu'un reprenant le projet de repartir sur une base que nous espérons la plus solide et la plus compréhensible possible. De plus, le LQR permet, pour peu qu'on arrive à la réparer, de faire voler le drone à partir de la mallette-radio.

Enfin, au delà de certaines déconvenues quand aux résultats des contrôleurs plus avancés, nous espérons que ce mémoire permet néanmoins un bon tour d'horizon, clair et progressif, des différentes manières de contrôler des drones ; les prochains maîtres de la troisième dimension.

Bibliographie

- [1] Mylène Julot. La chronique du cerpa - le drone-cible de cinquième génération de l'us air force en passe de voler., Avril 2020.
- [2] Organisation des Nations Unies. Rapport de l'onu - mars 2021, Mars 2021. URL <https://undocs.org/S/2021/229>. [En ligne ; Page 17, paragraphe 63].
- [3] Sebastian SEIBT. Des drones tueurs autonomes ont-ils été déployés en libye ?, Juin 2021. URL <https://www.france24.com/fr/%C3%A9co-tech/20210601-des-drones-tueurs-autonomes-ont-ils-%C3%A9t%C3%A9-d%C3%A9ploy%C3%A9s-en-libye>.
- [4] Florence Parly. Discours de clôture de florence parly - université d'été de la défense., 2017. URL <https://www.defense.gouv.fr/salle-de-presse/discours/discours-de-florence-parly/discours-de-cloture-de-florence-parly-universite-d-eete-de-la-defense-2017>.
- [5] Rédaction Vie Publique. Surveillance du respect des règles sanitaires par des drones : une décision du conseil d'État, 20 Mai 2020. URL <https://www.vie-publique.fr/en-bref/274348-surveillance-du-confinement-par-des-drones-decision-du-conseil-detat>.
- [6] Nathalie Guibert. Paris achète des drones américains pour rattraper son retard, 18 Mai 2013. URL https://www.lemonde.fr/international/article/2013/05/18/paris-achete-des-drones-americains-pour-rattraper-son-retard_3316410_3210.html.
- [7] Jean-Michel Normand. De nouveaux nanodrones black hornet pour les soldats français, 29 janvier 2019. URL https://www.lemonde.fr/la-foire-du-drone/article/2019/01/29/de-nouveaux-nanodrones-black-hornet-pour-les-soldats-francais_5416216_5037916.html.
- [8] Wikipédia. Blocage de cardan — wikipédia, l'encyclopédie libre, 2017. URL http://fr.wikipedia.org/w/index.php?title=Blocage_de_cardan&oldid=141688253. [En ligne ; Page disponible le 20-octobre-2017].
- [9] Elric THOMAS Rémy ROMAN Julien LAGET. Mécanique générale - notes de cours, 1er décembre 2020.

- [10] François Bateman. Cours supervision des drones, 2020.
- [11] Brian Douglas. State space, part 2 : Pole placement, 2019. URL <https://youtu.be/FXSpHy8LvmY?t=70>.
- [12] MIT. Lecture notes - mechanical engineering, maneuvering and control of surface and underwater vehicles, 2004. URL <https://ocw.mit.edu/courses/mechanical-engineering/2-154-maneuvering-and-control-of-surface-and-underwater-vehicles-13-49-fall-2004/lecture-notes/lec19.pdf>.
- [13] Brian Douglas. State space, part 4 : What is lqr control?, 2019. URL https://youtu.be/E_RDCF01Jx4?t=33.
- [14] Wikipédia. Ornithoptère — wikipédia, l'encyclopédie libre, 2021. URL <http://fr.wikipedia.org/w/index.php?title=Ornithopt%C3%A8re&oldid=183048632>. [En ligne ; Page disponible le 19-mai-2021].
- [15] Anirudh Shekhawat Pratik Poddar Dinesh Boswal. Ant colony optimization algorithms : Introduction and beyond. 2009. URL https://mat.uab.cat/~alseda/MasterOpt/ACO_Intro.pdf.
- [16] Verschueren Robin, Zanon Mario, Quirynen Rien, and Diehl Moritz. Time-optimal race car driving using an online exact hessian based nonlinear mpc algorithm, 2016.
- [17] M. Abdolhosseini Y. M. Zhang C. A. Rabbath. Journal of intelligent and robotic systems, volume 70, issue 1-4. April 2013. URL <https://doi.org/10.1007/s10846-012-9724-3>.
- [18] Kayacan E. Prach A. An mpc-based position controller for a tilt-rotor tricopter vtol uav. optim control appl meth., 2017. URL <https://doi.org/10.1002/oca.2350>.
- [19] Demis Hassabis David Silver Thomas Hubert Julian Schrittwieser. A general reinforcement learning algorithm that masters chess, shogi and go through self-play. 2017.
- [20] David Louapre. Le repliement des protéines : Résolu par l'intelligence artificielle alphafold ?, 2020. URL <https://www.youtube.com/watch?v=0GewxRMME8o>.
- [21] David Louapre. Une intelligence artificielle peut-elle être créative ?, 2019. URL <https://youtu.be/xuBzQ38DNhE?t=617>.
- [22] Cade Metz. In two moves, alphago and lee sedol redefined the future, 2016. URL <https://www.wired.com/2016/03/two-moves-alphago-lee-sedol-redefined-future/>.
- [23] Azer Bestavros William Koch Renato Mancuso Richard West. Reinforcement learning for uav attitude control, February 2019. URL <https://doi.org/10.1145/3301273>. [En ligne ; Page disponible le 19-mai-2021].

- [24] Two Minutes Papers. 4 experiments where the ai outsmarted its creators, 2019. URL <https://www.youtube.com/watch?v=GdTbqBnqhaQ>.
- [25] Grant Sanderson 3blue1brown. Neural networks, 2018. URL https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi.
- [26] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments, 2017.
- [27] MathWorks. Water tank reinforcement learning environment model, . URL <https://www.mathworks.com/help/reinforcement-learning/ug/water-tank-reinforcement-learning-environment-model.html>.
- [28] MathWorks. Water tank reinforcement learning environment model, . URL <https://www.mathworks.com/help/deeplearning/ref/deepnetworkdesigner-app.html>.
- [29] Sebastian Castro and MathWorks. Deep reinforcement learning for walking robots. URL https://www.mathworks.com/videos/deep-reinforcement-learning-for-walking-robots--1551449152203.html?s_eid=PSM_15028.

Annexes

Annexe 1 : Détail des blocs du schéma *SIMULINK*

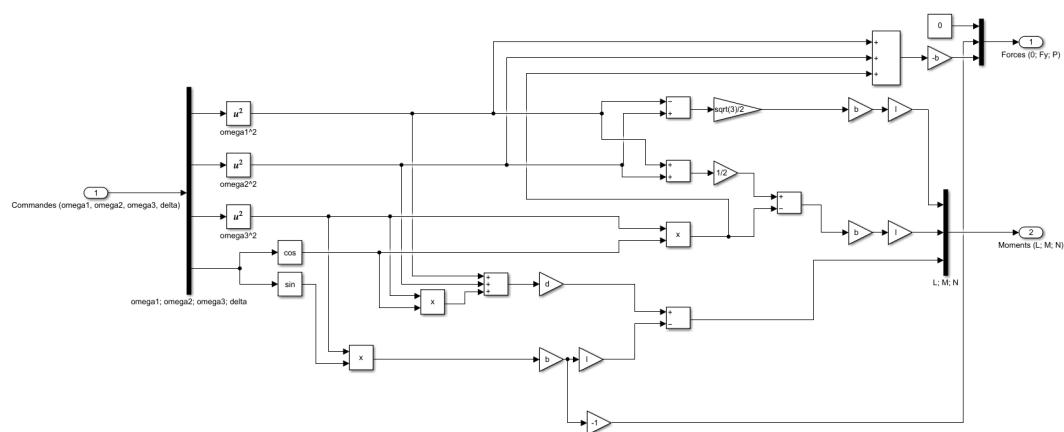


FIGURE 11.1 – Bloc - Groupe moto-propulsif

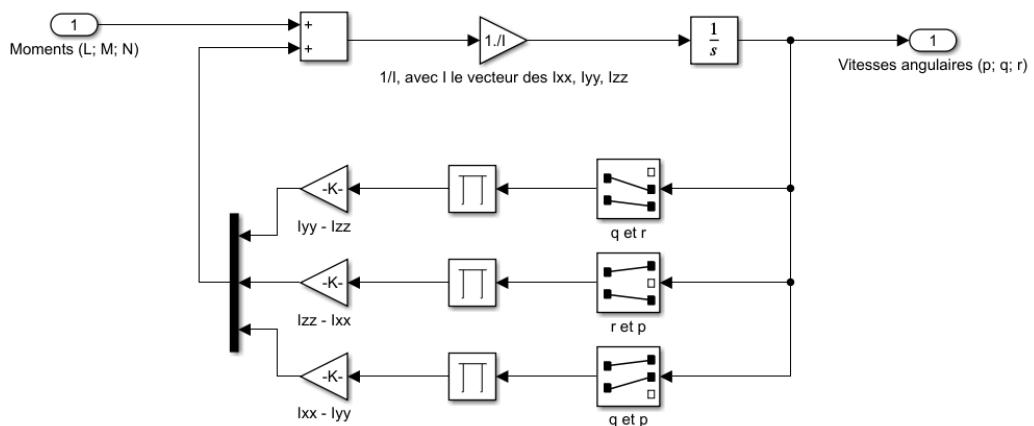


FIGURE 11.2 – Bloc - Dynamique de rotation

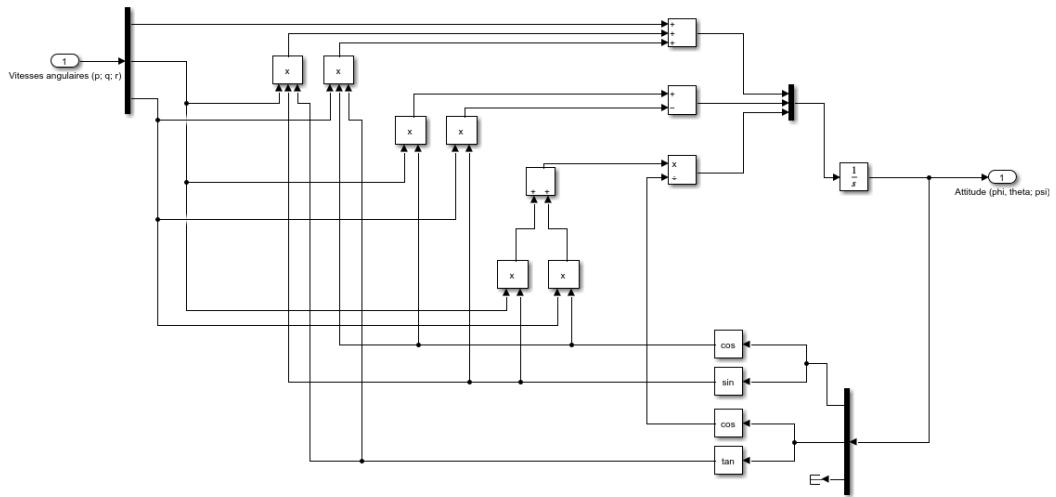


FIGURE 11.3 – Bloc - Cinématique de rotation

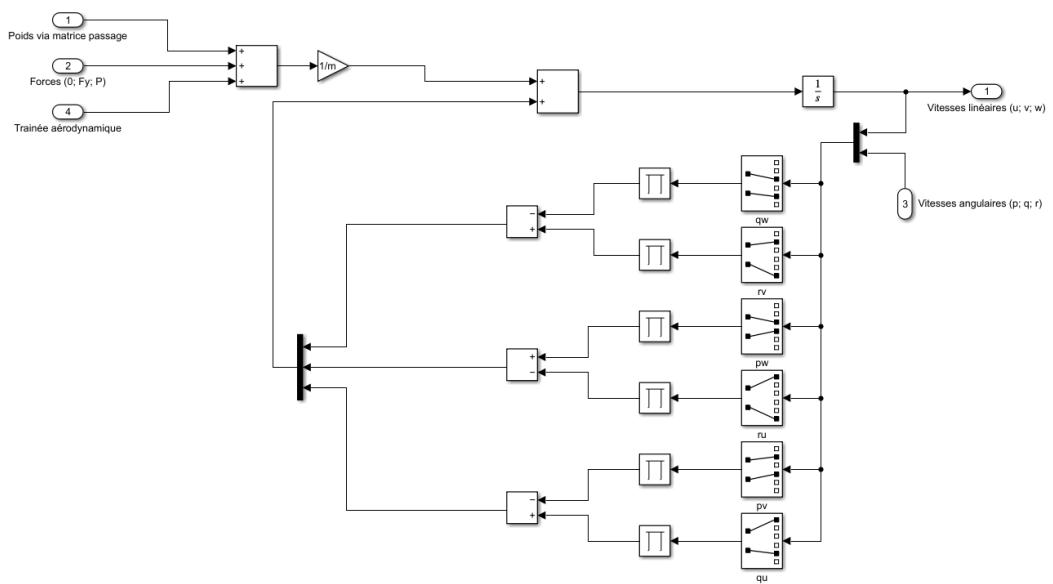


FIGURE 11.4 – Bloc - Dynamique de translation

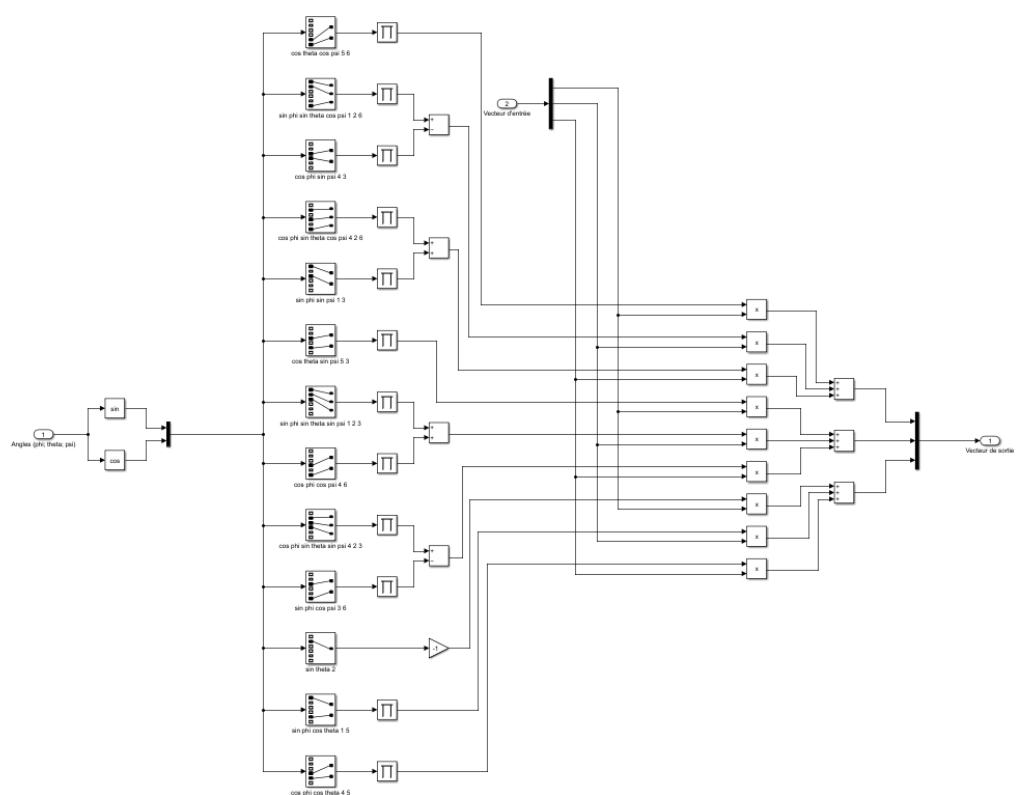


FIGURE 11.5 – Bloc - Matrice de passage TdE

Annexe 2 : Code optimisation

```
%% Initialisation

omegaMin = 0;
omegaMax = 2*omegaEq;
deltaMax = pi/2;
deltaMin = -pi/2;

ptDepart = [0; 0; 0; 0];

%limites des actionneurs
upperBound = [omegaMax; omegaMax; omegaMax; deltaMax];
lowerBound = [omegaMin; omegaMin; omegaMin; deltaMin];



---


%% Fonctions de calcul des compostantes de u_beta

betaColl = @(omegal, omega2, omega3, delta)...
    b*(omegal^2+omega2^2+omega3^2*cos(delta));

betaLongi = @(omegal, omega2, omega3, delta)...
    b*1*(0.5*(omegal^2+omega2^2)-omega3^2*cos(delta));

betaLat = @(omegal, omega2, omega3, delta)...
    sqrt(3)/2*b*1*(omega2^2-omegal^2);

betaPied = @(omegal, omega2, omega3, delta)...
    d*(omegal^2+omega2^2+omega3^2*cos(delta))-b*1*omega3^2*sin(delta);
```

```

%% Initialisation de la boucle de calcul

resMin = [Inf, Inf, Inf, Inf];
resMax = [-Inf, -Inf, -Inf, -Inf];

%% Boucle de calcul pour ne pas tomber dans un extremum local

for i = 1:20      % Avec quelques essais, on s'est rendu compte que 20 tours
    % de boucle suffisaient
    % On calcule une valeur aléatoire dans l'intervalle de fonctionnement
    % des moteurs et du servomoteur
    ptDepart = [rand*omegaMax; rand*omegaMax; rand*omegaMax; rand*pi-pi/2];

    [x1min, collMin] = fmincon(@(x) betaColl(x(1),x(2),x(3), x(4)), ...
        ptDepart, [], [], [], lowerBound, upperBound);

    [x2min, longiMin] = fmincon(@(x) betaLongi(x(1),x(2),x(3), x(4)), ...
        ptDepart, [], [], [], lowerBound, upperBound);

    [x3min, latMin] = fmincon(@(x) betaLat(x(1),x(2),x(3), x(4)), ...
        ptDepart, [], [], [], lowerBound, upperBound);

    [x4min, piedMin] = fmincon(@(x) betaPied(x(1),x(2),x(3), x(4)), ...
        ptDepart, [], [], [], lowerBound, upperBound);

    [x1max, collMax] = fmincon(@(x) -betaColl(x(1),x(2),x(3), x(4)), ...
        ptDepart, [], [], [], lowerBound, upperBound);

```

```

[x2max, longiMax] = fmincon(@(x) -betaLongi(x(1),x(2),x(3), x(4)),...
    ptDepart, [], [], [], lowerBound, upperBound);

[x3max, latMax] = fmincon(@(x) -betaLat(x(1),x(2),x(3), x(4)),...
    ptDepart, [], [], [], lowerBound, upperBound);

[x4max, piedMax] = fmincon(@(x) -betaPied(x(1),x(2),x(3), x(4)),...
    ptDepart, [], [], [], lowerBound, upperBound);

resTempMin = [collMin, longiMin, latMin, piedMin];
resTempMax = [collMax, longiMax, latMax, piedMax];

% On remplace les valeurs trouvées dans l'itération actuelle dans le
% résultat final si elles sont plus élevées (ou faible selon le cas) que
% les précédentes.

for j = 1:4
    if resTempMin(j)<resMin(j)
        resMin(j) = resTempMin(j);
    end

    if resTempMax(j)>resMax(j)
        resMax(j) = resTempMax(j);
    end
end
end

```

Annexe 3 : Simulations du MPC en réponse à un échelon de commande en utilisant la représentation d'état

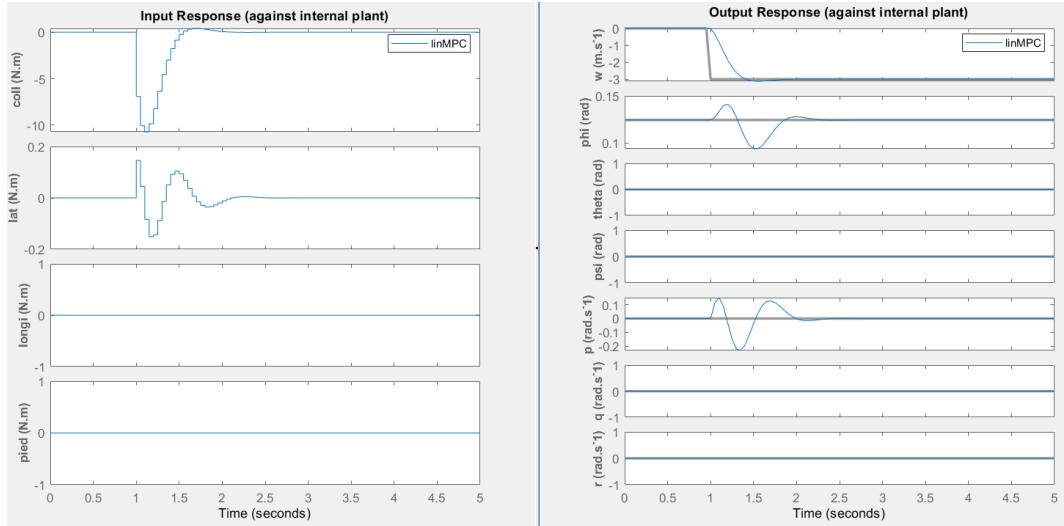


FIGURE 11.6 – Réponse du MPC pour un échelon de β_{coll}

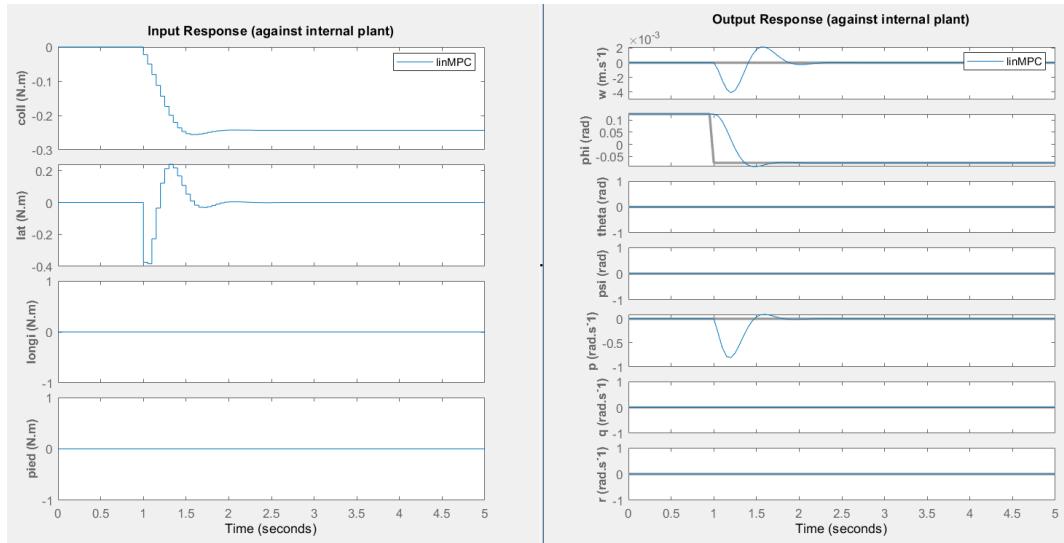


FIGURE 11.7 – Réponse du MPC pour un échelon de β_{lat}

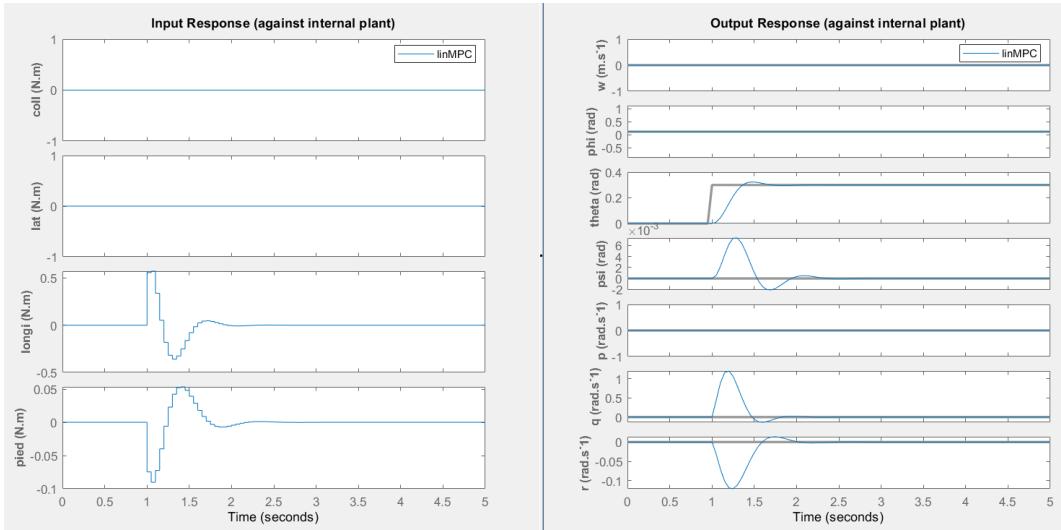


FIGURE 11.8 – Réponse du MPC pour un échelon de β_{longi}

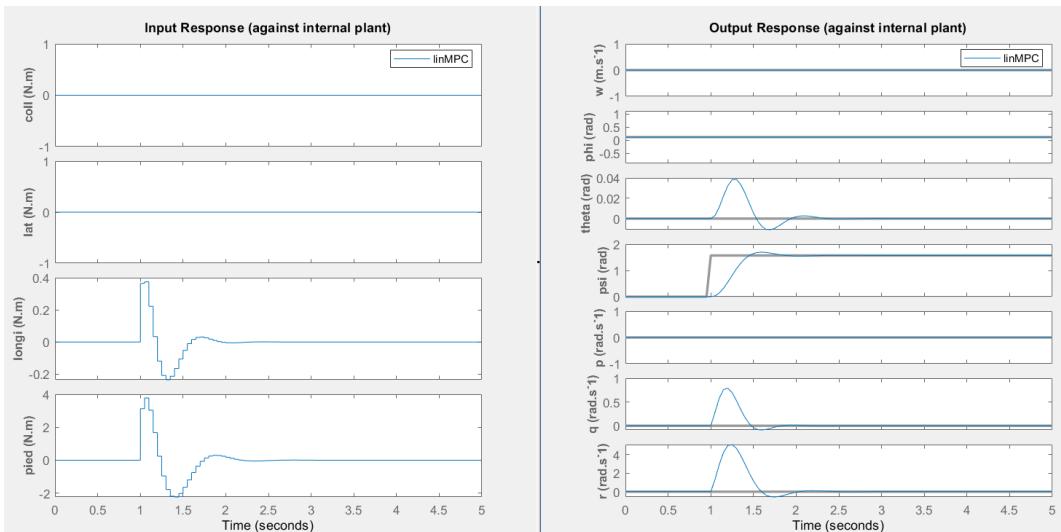


FIGURE 11.9 – Réponse du MPC pour un échelon de β_{pied}

Annexe 4 : Schémas SIMULINK relatifs à l'implémentation de l'algorithme de RL

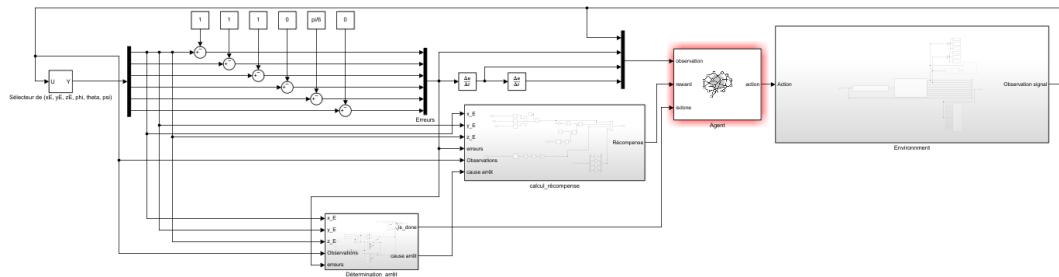


FIGURE 11.10 – Modèle *SIMULINK* pour l'algorithme de RL

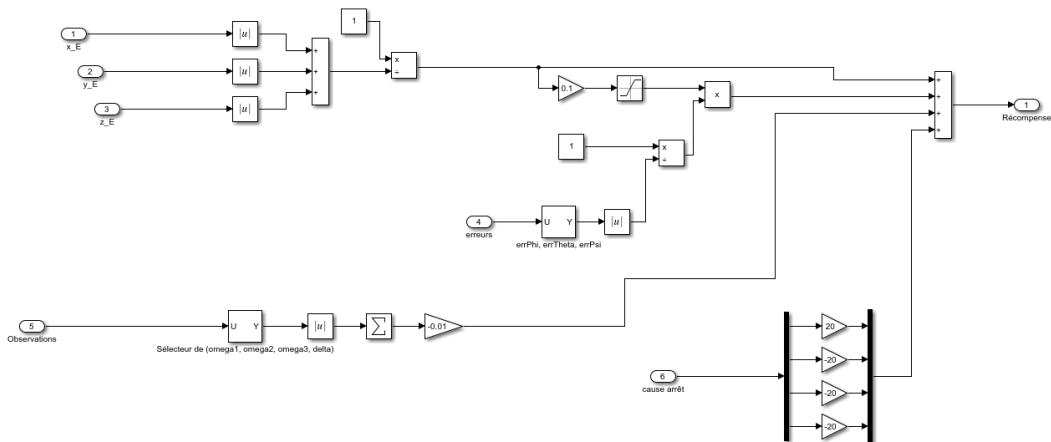


FIGURE 11.11 – Calcul de la récompense

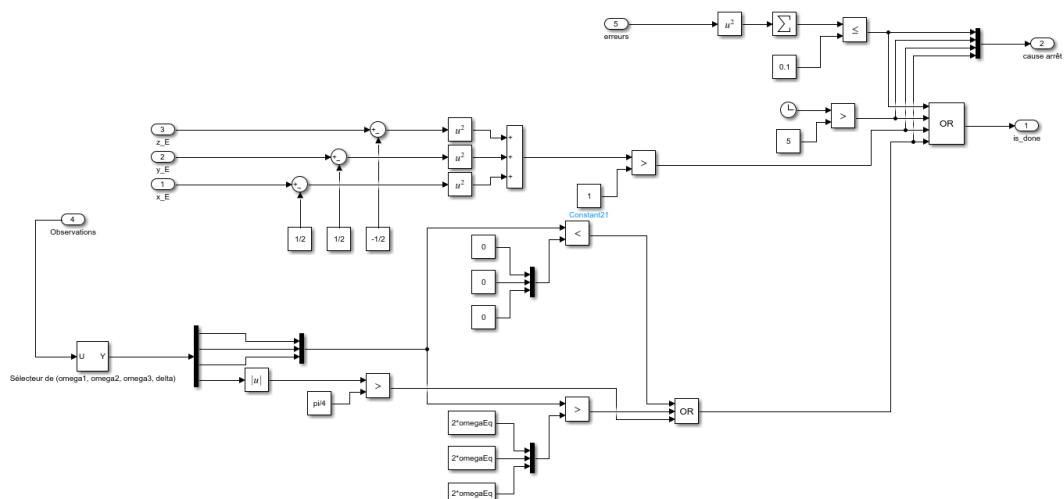


FIGURE 11.12 – Détermination de l'arrêt de la simulation