AER3900 : Projet intégrateur III

Stabilisation d'un drone à l'aide d'un contrôleur d'attitude basé sur un réseau neuronal profond.

Dionne, François

Matricule : 1949253

Travail présenté à

Prof. David Saussié

Département de génie mécanique

Polytechnique Montréal

À Montréal

Le 19 mars 2021

Contents

List of Figures

Résumé

La grande majorité des drones d'aujourd'hui sont actués par des systèmes de contrôle linéaires. Malgré l'efficacité de ceux-ci, les drones sont tout de même des systèmes non-linéaires, donc ils ne pourront jamais être parfaitement contrôlés par des systèmes linéaires.

C'est alors ce défi qui a été relevé dans ce projet intégrateur III. En effet, un contrôleur d'attitude pour drone, basé sur un réseau neuronal profond, a été entraîné et testé. Pour ce faire, un environnement d'entraînement basé sur le moteur de physique Pybullet a été développé. Le modèle de drone utilisé est le crazyfly. Un algorithme de Proxymal Policy Optimization a été employé pour l'entraînement. Finalement, deux méthodes d'entraînement ont été utilisées et comparées dans ce rapport.

Les contrôleurs entraînés ont été validés par simulation avec Pybullet-drones. Les performances ont été quantifiées pour permettre la comparaison des contrôleurs. Finalement, des recommandations ont été faites pour aider la continuation du projet.

Abstract

Drones are nowadays controlled by linear controllers. Despite their performance, drones are higly non-linear systems and never will be perfectly controller by such controllers.

This is the challenge that was completed during this project and discussed in this paper. An attitude controller, based on a deep neural network, was trained. To do so, a training environnement, also known as gym, was built, based on the Pybullet physics engine. The drone modeled used is based on the CrazyFly. To accomplish the reinforcement learning, Proximal Policy Optimization was used and two different training methods were compared.

The two trained controllers were validated using Pybullet-drones simulations. Performances were quantified to compare them and recommandations were made to guide future work that could be done for this project.

1 Introduction

Les quadrirotors sont des robots à priori instables. Ils sont également sous-actués, ayant seulement quatre moteurs pour contrôler leurs six degrés de liberté. Ces caractéristiques font en sorte qu'un système de contrôle relativement complexe est nécessaire pour effectuer des déplacements coordonnés dans l'espace. Les contrôleurs généralement utilisés pour le contrôle d'attitude de drones sont des contrôleurs PID. Linéarisant le comportement du drone, ils sont faciles à implémenter et efficaces.

Les quadrirotors sont cependant loin d'être des systèmes linéaires. On cherche alors d'autres méthodes que le PID pour contrôler l'attitude d'un drone, prenant en compte des phénomènes non-linéaires. C'est alors que les réseaux neuronaux entrent en jeu. Ceux-ci, après entraînement, permettent d'approximer n'importe quelle fonction. On peut alors penser qu'il est possible de créer un réseau neuronal prenant en entrée l'état du drone et l'état désiré et ayant en sortie les commandes pour les moteurs.

Dans ce projet, le but sera d'entraîner un tel réseau neuronal à l'aide de l'apprentissage par renforcement. Un modèle de drone sera simulé sur Pybullet-drones pour entraîner le contrôleur, puis celui-ci sera optimisé à l'aide de la PPO (proximal policy optimization). Une fois entraîné, le contrôleur sera validé dans un autre environnement Pybullet-drones pour quantifier sa performance.

2 Objectifs et échéancier

2.1 Objectifs et livrables

Les objectifs principaux de ce projet sont :

- Développer un contrôleur d'attitude de drone basé sur un réseau neuronal.
- Documenter le travail pour que les résultats soient répétables.

Les livrables du projet sont les suivants :

- Des simulations du contrôleur pour observer ses performances.
- Le contrôleur prêt à être implémenté dans un autre projet.
- Le rapport de projet.

2.2 Échéancier

Ce projet intégrateur se déroule pendant la session d'hiver 2021. Pour structurer le travail et mettre les échéances sur papier, un échéancier de Gantt a été élaboré. Il apparaît sur la figure 1. Les seules contraintes par rapport aux échéances sont le début du projet, le 3 février, et la remise du rapport, le 21 avril.

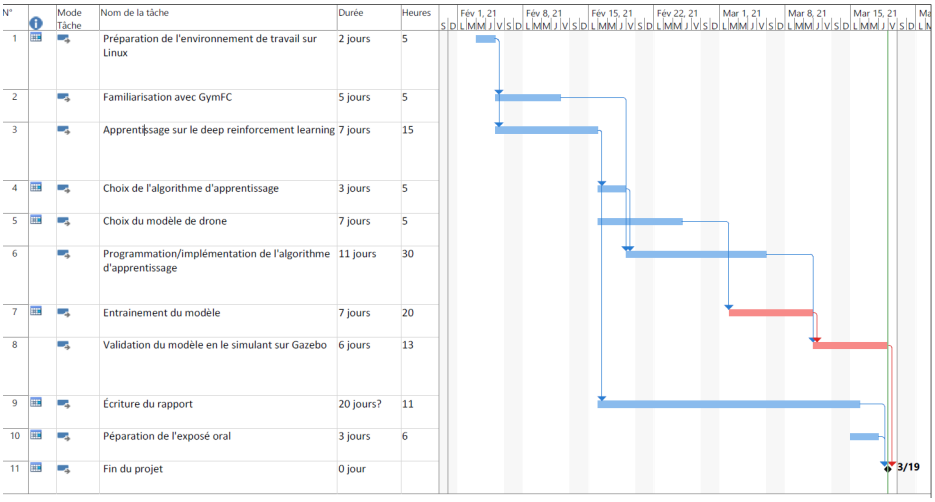

Figure 1: Échéancier du projet

3 Revue de la documentation

La revue de la documentation est une étape importante en début de projet. Elle permet de prendre conscience de ses lacunes et orienter le travail. Ce projet prend ses racines dans le contrôle d'attitude de quadricoptères, dans les réseaux neuronaux et dans l'apprentissage par renforcement. Pour faire la recherche de documentation pertinente, les mots clés suivants ont été employés : drone, contrôle d'attitude, neural networks, reinforcement learning et Proximal Policy Optimization (PPO).

3.1 Cadre théorique

Le contrôle d'attitude des drones est nécessaire, puisque ce sont des systèmes instables. Il désignera ici le contrôle des vitesses angulaires de tangage, lacet et de roulis. Dans l'industrie, ce sont des contrôleurs linéaires PID qui sont utilisés pour remplir cette fonction, mais ils ont des limites. En effet, les drones sont des systèmes dont la dynamique est non-linéaires, donc tous les effets ne sont pas pris en compte par les PID. Les réseaux neuronaux ont alors des propriétés intéressantes pour les systèmes non-linéaires. Il est possible de les entraîner avec des techniques d'apprentissage par renforcement pour qu'ils remplacent les PID dans le contrôle d'attitude de drones.

Les réseaux neuronaux sont des fonctions. Ils prennent une ou plusieurs variables en entrée et renvoient une ou plusieurs variables en sortie. Ils sont composés d'un nombre fini de neurones répartis dans plusieurs couches. La figure 2 présente un réseau neuronal avec trois couches de neurones cachés.

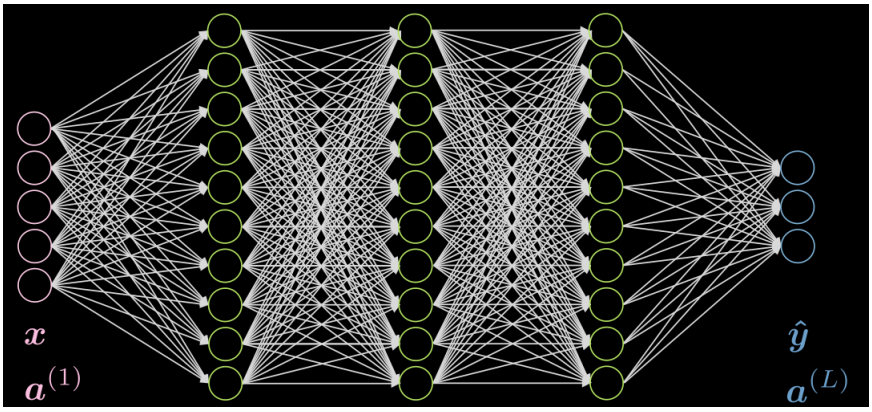

Figure 2: Réseau neuronal avec 3 couches cachées

Chaque neurone portera une valeur lors du calcul des sorties, qui dépend de la valeur des neurones précédents qui lui sont reliés. La figure 3 présente les calculs requis pour l'évaluation d'un neurone.

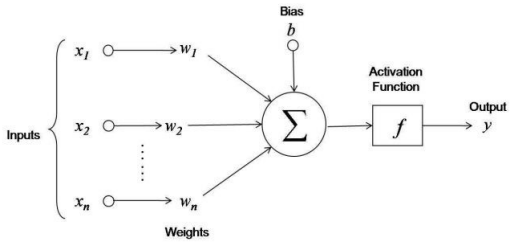

Figure 3: Opérations faites par un neurone

L'apprentissage par renforcement est une technique utilisée pour faire converger un RN vers la fonction désirée. En effet, au départ, les poids des RN sont initialisé aléatoirement et ont besoin d'être ajustés pour remplir leur fonction. Pour ce faire, des milliers de simulations du systèmes sont

faites en évaluant la réponse du contrôleur et en y attribuant une récompense, dépendant du taux de succès de celui-ci. Au bout de l'entraînement, la réponse du contrôleur devrait être optimale, il a alors convergé.

La Proximal Policy Optimization (PPO) est un algorithme d'apprentissage par renforcement utilisée pour les systèmes continus. On appelle la politique le RN qui calcul les actions devant être prises. Un pseudocode de la PPO est donné à la figure 4. En fait, il a été développé par OpenAI et plusieurs versions sont disponible en source ouvertes. Pour ce projet, la programmation sera faite à partir du pseudocode.

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
  Compute rewards-to-go $\hat{R}_t$.
  Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
  Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

  typically via stochastic gradient ascent with Adam.
  Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

  typically via some gradient descent algorithm.
end for

Figure 4: Pseudocode de la PPO

3.2 Travaux antérieurs

Ce projet se base sur des recherches récentes pour le contrôle d'attitude avec RN. Le document central est *Reinforcement Learning for UAV Attitude Control*, par des chercheurs de l'université de Boston. Le but de leur recherche était de « fournir une plateforme pour entraîner des contrôleurs d'attitude avec l'apprentissage par renforcement et de poser des bases pour la précision de tels contrôleurs

» (Koch, Mancuso, West, & Bestavros, 2019). La plateforme developpée est GymFC. Celle-ci se charge de faire l'interface entre le contrôleur et la simulation. Le rapport de cette recherche explique comment l'interface est faite et donne de bonnes pratiques pour l'utilisation de la plateforme. Les leçons apprises lors du développement de la plateforme seront utilisées pour gagner du temps dans l'entraînement du réseau, entre autres pour le choix des paramètres d'entraînement.

L'apprentissage par renforcement profond est une discipline en constante évolution. OpenAI, une compagnie développant des outils reliés à l'intelligence artificielle offre une panoplie de documents expliquant des concepts d'apprentissage profond. Ils ont récemment développé la PPO, qui est la technique la plus adapté aux contrôleurs continus en ce moment. Leur recherche sur le sujet est alors un document important pour le projet : Proximal Policy Optimization Algorithms (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017).

La *Brain Team*, de Google Research, ont fait un article sur les bonnes pratiques sur l'apprentissage de politiques : What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study (Andrychowicz et al., 2020). Des conseils y sont formulés, entre autres sur l'initialisation du RN et sur les valeurs à donner à certaines constantes de la PPO.

4 Méthodologie

4.1 Choix de l'environnement d'entraînement

Pour entraîner un RN, il faut d'abord avoir un environnement d'entraînement, couramment appelé *gym*. Cela peut être fait en créant un modèle de drone et en faisant des simulations directement sur python. Bien que cette méthode soit très simple, puisqu'aucune interface entre des logiciels n'est nécessaire, elle a le désavantage de limiter les possibilités d'intégrations sur des vrais drones. En effet, la transition entre la simulation et le monde réel peut montrer des lacunes fatales dans les

contrôleurs. Il est alors préférable d'utiliser des moteurs de simulation, comme Gazebo ou Pybullet, pour que la simulation soit la plus fidèle à la réalité que possible.

Cela étant dit, l'environnement Pybullet-drones sera utilisé pour l'entraînement du contrôleur dans ce projet. Celui-ci est développé à l'université de Toronto et offre une interface complète sur python pour la simulation de drone. De plus, il est déjà adapté pour être utilisé dans des applications d'apprentissage profond.

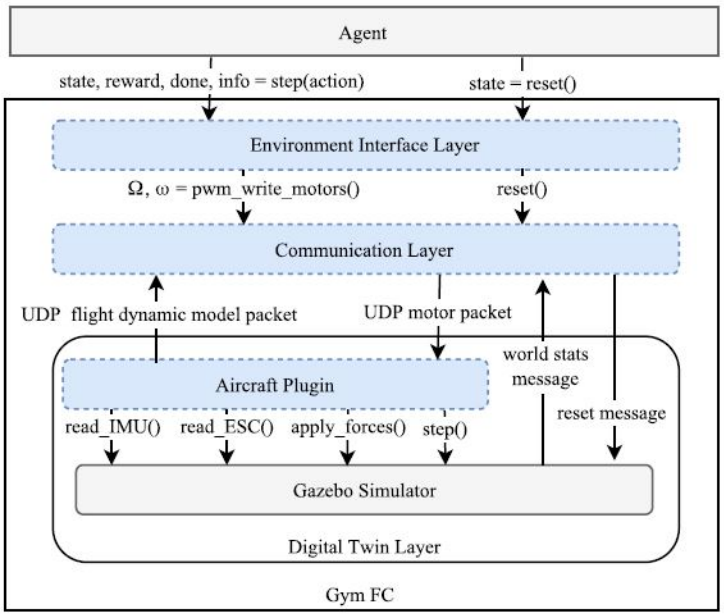

Figure 5: Schématisation de l'environnement de GymFC

La figure 5 présente la structure de GymFC. L'agent est le réseau neuronal en entraînement. Il réagit aux états qui lui sont envoyé et reçoit des récompenses qui lui permettent de s'améliorer, suivant la méthode d'optimisation qui est utilisée. Bien que la représentation soit celle de gymFC, Pybullet suit les mêmes principes.

L'entraînement sera alors basé sur l'environnement "Hover-Aviary" de Pybullet-drones, qui sera modifié pour faire du contrôle d'attitude, c'est à dire fixer le drone sur un pivot dans l'espace.

4.2 L'entraînement

L'entraînement réfère au processus utilisé pour améliorer la réponse du RN aux différents états. Cette amélioration dépend alors de quand et comment le RN est récompensé, c'est ce qu'on appelle l'ingénierie de récompense. La méthode présenté dans l'article sur GymFC est de récompenser le RN lorsque l'erreur entre l'état désiré et l'état présent diminue. La récompense est alors de la forme suivante (Koch et al., 2019):

$$r_t = -clip(\sum(\Omega_t^* - \Omega_t)/3\Omega_{max}) \tag{1}$$

où

r_t: La récompense

Ω_t^*: La vitesse angulaire désirée

Ω_t: La vitesse angulaire actuelle

Ω_{max}: La vitesse angulaire maximale permise par le contrôleur

La récompense est négative, puisque nous cherchons à la maximiser, c'est à dire minimiser l'erreur. On fait alors la somme des vitesses angulaires dans les trois axes de rotation, pour n'avoir qu'une seule récompense. L'opérateur clip permet de restreindre la récompense à une valeur entre 0 et 1. Cette forme de récompense ayant faite ses preuve, elle sera utilisée dans le projet.

Une fois la récompense calculée, il faut un algorithme pour la traiter et modifier les paramètres du RN pour que celui-ci s'améliore. La PPO est un des algorithmes répondant à ce besoin et c'est celui qui sera utilisé dans ce projet, étant donné qu'il est le plus adapté pour les systèmes continus (Andrychowicz et al., 2020).

La figure 6 présente les interactions entre le RN qui est entraîné, l'environnement GymFC et l'algorithme d'apprentissage par renforcement. L'environnement exporte l'état du drone, soit les

erreurs sur les vitesses angulaires du drone. L'agent, composé du contrôleur et de l'algorithme d'apprentissage (PPO), reçoit les états ainsi que la récompense. Le RN reçoit alors un feedback sur sa performance et exporte les prochaines actions à poser pour continuer de réduire les erreurs.

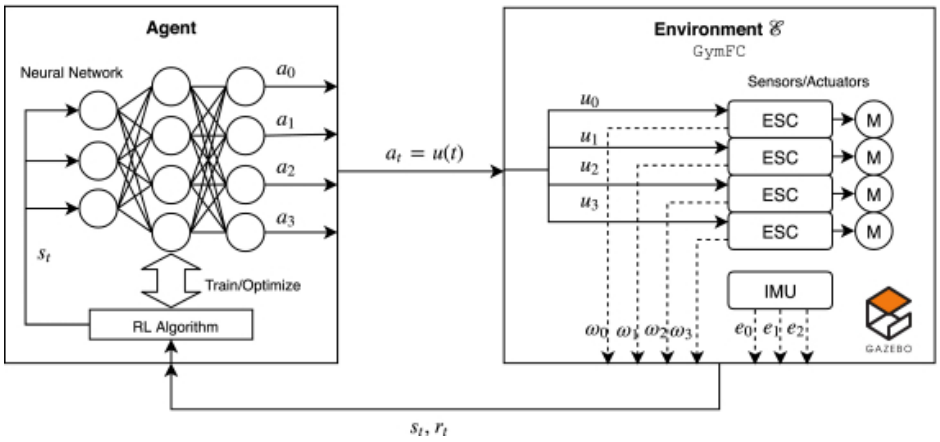

Figure 6: Schématisation de l'entraînement d'un agent

4.2.1 Recommandation de la Google Brain Team

Tel que mentionné dans le revue de la documentation, la Google Brain Team a écrit un article regroupant des recommandations dans l'entraînement par apprentissage par renforcement. Voici celles qui concernent la PPO (Andrychowicz et al., 2020):

- Commencer avec un seuil de la fonction *clip* de 0.25
- Initialiser la dernière couche cachée du RN avec des valeurs 100 fois plus petites que dans les autres couches.
- Utiliser la fonction Tanh comme fonction d'activation des neurones et pour transformer la distribution normale de sortie en action
- Le facteur de diminution γ doit être initialisé à 0.99 et réduit si nécessaire
- Initialiser le taux d'apprentissage à 0.0003

L'apprentissage par renforcement est très sensible à l'initialisation du RN et aux paramètres d'apprentissages. Plusieurs contrôleurs seront alors entraînés jusqu'à ce qu'un d'eux obtiennent les performances désirées, en faisant varier ces paramètres si nécessaire.

4.3 Validation

La validation du projet sera faite directement sur Pybullet-drones. Un environnement sera créé où l'on pourra quantifier et visualiser les performances du contrôleur.

5 Développement

Cette section portera sur le développement des différents outils utilisés pour entraîner et tester le réseau neuronal et les choix de design qui ont été fait pour y arriver. Le but ici est avant tout de pouvoir comprendre ce qui a été fait pour que le travail puisse être continué dans le futur. Les différents codes utilisés dans le projet sont disponibles sur le Github suivant : AJOUTER UN GITHUB

5.1 Installation de Pybullet-drones sur windows

Les instructions suivantes ont été trouvées sur le GitHub du projet Pybullet-drones, dans la section d'installation pour windows (https://github.com/utiasDSL/gym-pybullet-drones/tree/master/assignments-on-windows). Il est aussi disponible sur Mac et Linux. Les grandes lignes de l'installation sont les suivantes :

- Avoir Visual Studio and C++ 14.0 d'installé.
- Le IDE python recommandé pour les projets sur pybullet-drones est Pycharm.
- Télécharger les fichiers nécessaires sur le Github du projet.
- Créer un environnement virtuel python et installer tous les packages requis, tel qu'indiqué dans les instructions.
- Ajouter pytorch aux packages installés.
- Tester si les programmes de test fonctionnent.

Pour ce projet, les fichiers *BaseAvirary*, *BaseSingleAgentAviary* et *HoverAviary* ont été modifiés, donc ils doivent être remplacés par les fichiers du Github de ce projet intégrateur se trouvant dans le Github mentionné au début de la section. Le code d'entraînement se trouve dans le fichier *PPO_PI3.py* et le code pour le test des contrôleurs est *Test_control.py*

Les modifications dans la classe *BaseAviary* concernent le calcul d'un objectif de vitesse angulaire dans la fonction *reset*, qui redémarre la simulation. Cet objectif est alors aussi ajouté aux entrées de la fonction *step*, pour le calcul de la récompense à chaque pas de simulation. Pour simuler le drone comme s'il était fixé sur un pivot, la fonction *_dynamics* a aussi été modifiées pour empêcher le drone d'avoir une vitesse en x, y et z. Seul la rotation est alors possible.

5.2 Algorithme d'entraînement : PPO

Tel que présenté dans la revue de la documentation et illustré à la figure 4, la PPO est un algorithme d'entraînement bien documenté et plusieurs version sont disponible en source ouvertes. Celle de ce projet est inspirée de celle de Eric Yang Yu (Eric, 2020), qui a écrit un article détaillant l'implémentation de la PPO avec PyTorch. Celle-ci étant générale, des modifications ont été apportées pour l'intégration avec l'environnement d'entraînement. On retrouve le code d'entraînement dans la classe PPO de *PPO_PI3*.

En entraînement, pour forcer l'acteur à faire de l'exploration, les actions prises par celui-ci sont sélectionnées aléatoirement dans une distribution normale, centrée autour de l'action désirée par celui-ci.

5.2.1 Paramètres

L'entraînement par renforcement profond est très sensible aux paramètres d'entraînement. Pour obtenir un résultats optimal, il est alors nécessaire de faire plusieurs itérations en changeant, entre autres, ceux-ci. Ils peuvent alors être changés dans la fonction "hyperparameters" de la classe PPO. Ceux qui ont été utilisés pour faire converger les RN de ce projet sont :

- Nombre de steps par batch = 3200

- Nombre de steps par épisode = 50
- Discount factor (gamma) = 0.99 discount factor
- clip = 0.25
- Nombre de descente de gradient par batch = 2
- Learning rate = 0.0003

5.2.2 Réseaux neuronaux

L'initialisation des RN se fait avec la classe *FeedForwardNN*. L'initialisation de cette classe prend en entrée le nombre d'entrées du réseau, le nombre de sortie du réseau et la largeur des couches cachées du réseau. L'activation des neurones se fait avec une fonction tanh, tel que recommandé par la google brain team.

Dans la classe PPO, l'acteur et le critique sont initialisés. L'acteur est un RN avec 3 entrées, 4 sorties et est de largeur 32. La largeur a été déterminée avec les résultats de GymFC, qui recommande cette valeur. Le critique prend 3 entrées, les mêmes que l'acteur et n'a qu'une seule sortie. Il a une largeur de 5, choisie arbitrairement.

5.2.3 Fonction d'apprentissage

La fonction learn se trouvant dans la classe PPO est celle qui fait l'apprentissage. Elle prend en entrée le nombre de batch d'entraînement devant être faites. Voici les principales étapes de la fonction:

- Simulation d'une batch d'expériences.
- Calculer les paramètres de la PPO, avantage et probabilité logarithmique.
- Impression de l'avancement et enregistrement des réseaux.

- Calcul du *loss* des deux réseaux.
- Fait un pas de propagation arrière sur les deux réseaux.

5.2.4 Points de contrôle

L'entraînement pouvant être très long, il est important de faire des sauvegardes du réseau neuronal pendant l'entraînement, au cas où il divergerait. La fonction *Update* crée alors un fichier résumant le progrès du contrôleur et l'enregistre dans le dossier *Checkpoint* se trouvant dans le même dossier que le programme. Les fichiers sont nommés selon le numéro de la batch où la sauvegarde a été faite et de la récompense moyenne obtenue pendant cette batch. Il est possible de reprendre l'entraînement à partir d'une sauvegarde avec la fonction *load* de la classe PPO. Les points de sauvegarde sont sous la forme d'un dictionnaire python.

5.3 Programme de test du contrôleur

La fonction de test sert à observer le comportement du contrôleur face à des commandes prédéterminées. Le programme suppose que le fichier de sauvegarde du contrôleur à tester se trouve dans le dossier *Checkpoint*. Voici les principales tâches du programme :

- Initialisation du contrôleur à tester.
- Initialisation de l'environnement.
- Simulation de la ou les commandes désirées.
- Graphiques des actions et des réponses du contrôleur dans les trois axes.

La variable setpoint, sous forme d'un *numpy array* de forme 1 par 3, représente les objectifs de vitesses angulaires que le drone doit atteindre pendant la simulation.

6 Résultats

Les résultats de la validation des contrôleurs développés dans ce projet sont présentés dans la section suivante. Les résultats de deux contrôleurs seront présentés, ayant été entraînés avec des méthodes différentes. Le premier en étant fixé sur un pivot et le second en étant en chute libre.

6.1 Résultats du drone fixé

Dans les simulations de test suivantes, des commandes de vitesses angulaires de 1 rad/s sont initialement demandée, ensuite -1 rad/s et finalement 0 rad/s. Trois simulations différentes sont faites pour tester les trois axes. Une dernière simulation est faite pour tester les trois axes en même temps.

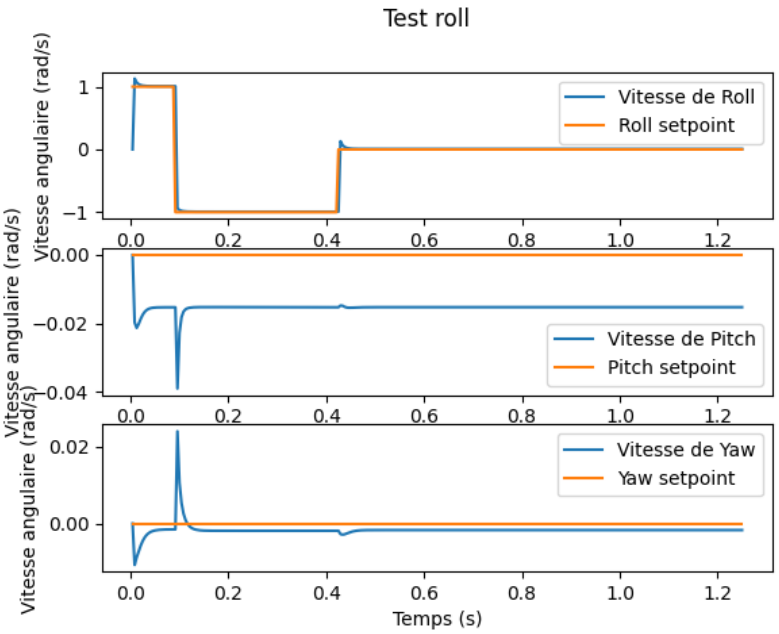

Figure 7: Réponse du contrôleur entraîné fixe à des commandes de roll

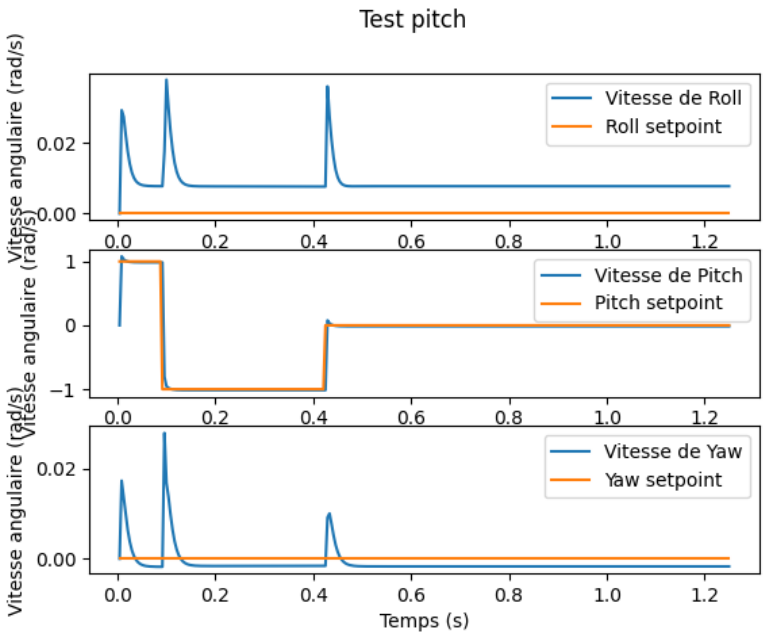

Figure 8: Réponse du contrôleur entraîné fixe à des commandes de pitch

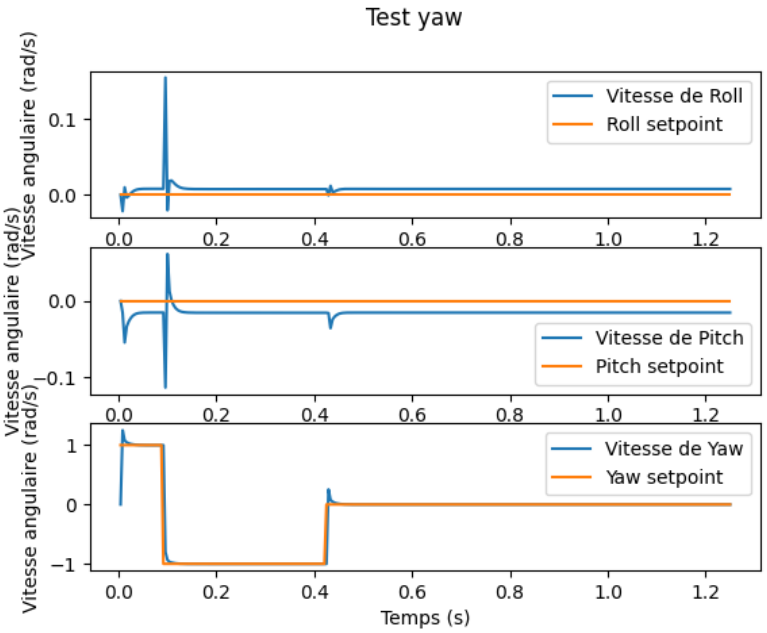

Figure 9: Réponse du contrôleur entraîné fixe à des commandes de yaw

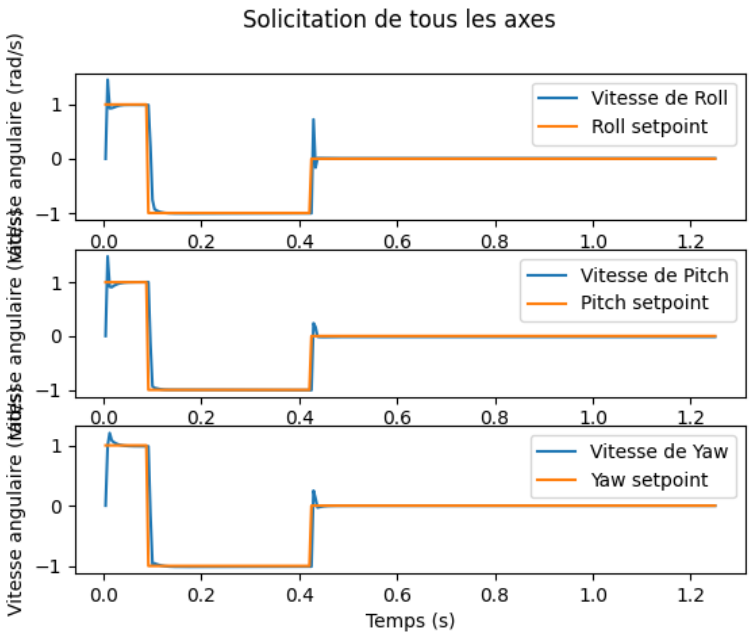

Figure 10: Réponse du contrôleur entraîné fixe à des commandes dans tous les axes

6.2 Résultats du drone entraîné libre

Les résultats aux tests de commande dans les différents axes sont présenté dans les figures suivantes.

Les commandes sont les mêmes que pour l'autre contrôleur.

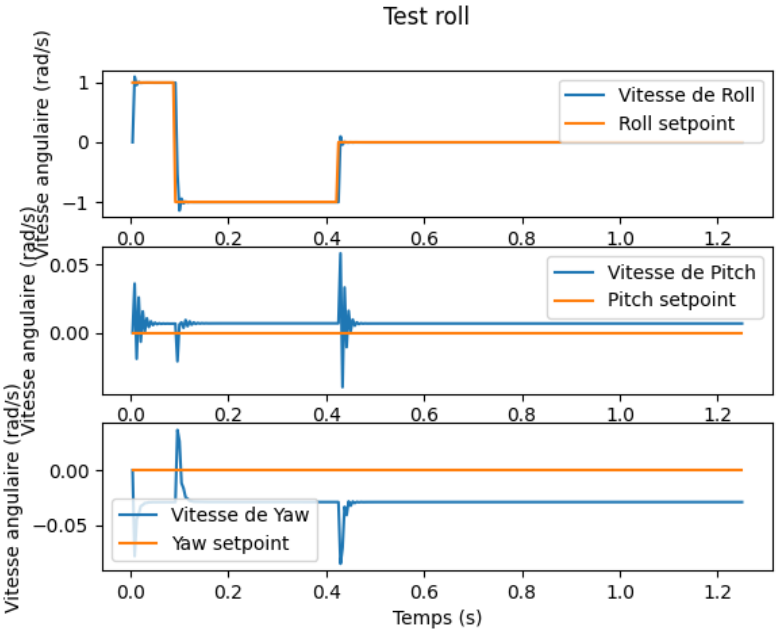

Figure 11: Réponse du contrôleur entraîné libre à des commandes de roll

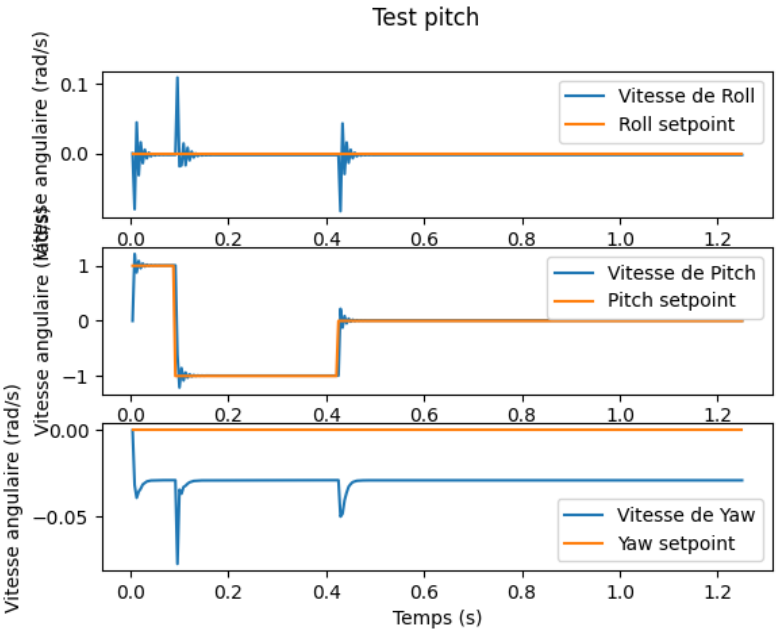

Figure 12: Réponse du contrôleur entraîné libre à des commandes de pitch

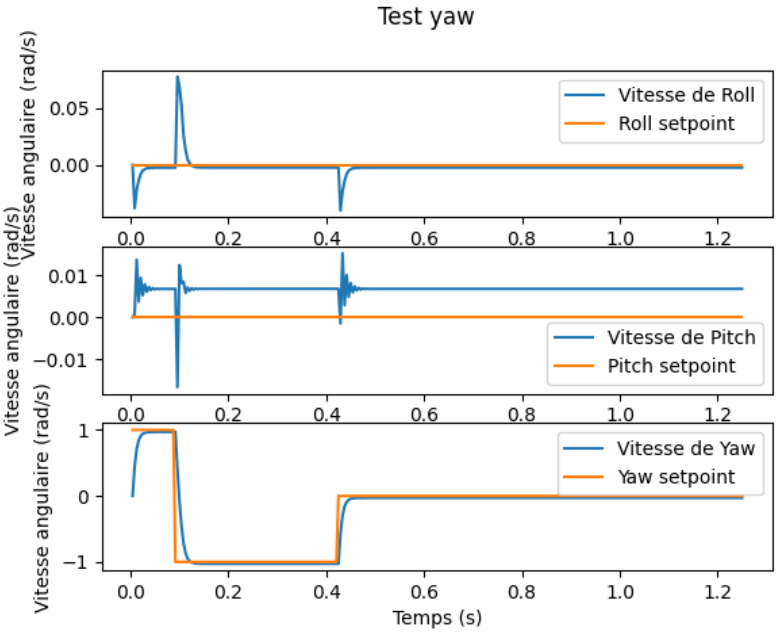

Figure 13: Réponse du contrôleur entraîné libre à des commandes de yaw

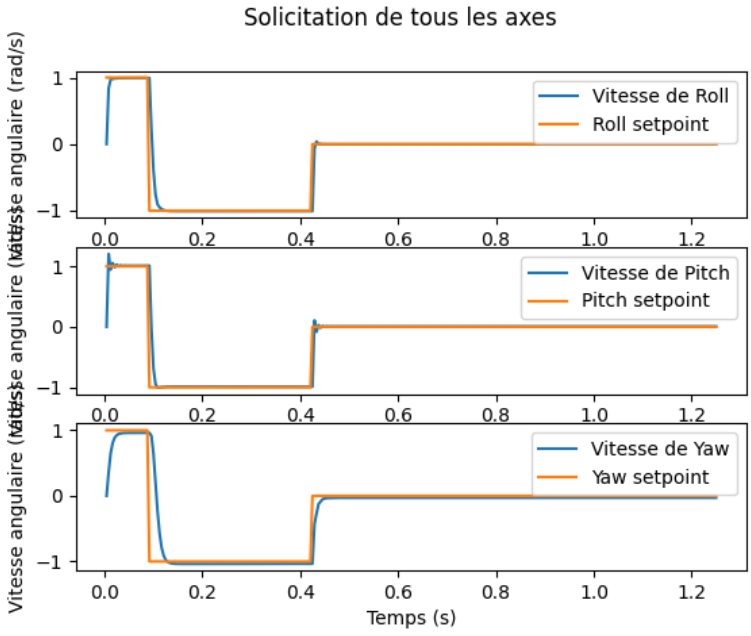

Figure 14: Réponse du contrôleur entraîné libre à des commandes dans tous les axes

7 Analyse des résultats

Les deux contrôleurs entraînés ont été présentés dans la section précédentes. Leur performance sera analysée dans cette section. Les tableaux suivant résument la performance observée dans la réponse des contrôleurs.

		Commande de 0 à 1 rad/s			Commande de 1 à -1 rad/s			Commande de 1 à 0 rad/s		
		Temps de réponse ms	Dépassement %	Erreur %	Temps de réponse ms	Dépassement %	Erreur %	Temps de réponse ms	Dépassement %	Erreur %
Drone entraîné fixe	Roulis	11	14	1,7	7	0	1,5	12	14	2
	Tangage	7	9	1,2	10	0	1,25	9	8	1,5
	Lacet	12	25	1,5	11	0	0,8	13	25	2
Drone entraîné libre	Roulis	9	10	0,2	13	10	0,2	9	10	0,25
	Tangage	14	22	0,6	22,5	21	0,7	17	23	0,7
	Lacet	21	0	3	28	0	3	22	0	3

Figure 15: Réponse des contrôleurs à des solicitations individuelles dans chaque axes

		Commande de 0 à 1 rad/s			Commande de 1 à -1 rad/s			Commande de 1 à 0 rad/s		
		Temps de réponse ms	Dépassement %	Erreur %	Temps de réponse ms	Dépassement %	Erreur %	Temps de réponse ms	Dépassement %	Erreur %
Drone entraîné fixe	Roulis	12	45	1,1	16,5	0	0,9	14	72	1
	Tangage	12	45	0,01	12,5	0	0,5	14	25	1,5
	Lacet	20	20	1,25	12,5	0	1,3	13	20	0,4
Drone entraîné libre	Roulis	12	0	0,6	26	0	0,6	10	0	0,2
	Tangage	12	20	1	17,5	0	1	10	20	0,5
	Lacet	25	0	4	34,7	0	4	19	0	3

Figure 16: Réponse des contrôleurs à des commandes dans tous les axes

7.1 Sollicitation individuelle des axes

Pour la sollicitation individuelle des axes, les deux contrôleurs présentent des caractéristiques constantes pour les trois commandes dans tous les axes. On remarque que pour les deux contrôleurs, une commande selon un seul axe a quand même une répercussion momentanée dans les autres axes, à cause du couplage de ceux-ci. La pente raide lors de l'augmentation de la vitesse angulaire et le temps de réponse rapide indique également que le contrôleur sature les moteurs.

Le contrôle du roulis est similaire pour les deux contrôleurs, avec un temps de réponse moyen de 10ms pour les deux. Le dépassement est également similaire. Le second contrôleur offre cependant une erreur en régime permanent plus faible, avec une moyenne de 0.22%, contre 1.75% pour le premier.

Pour le tangage, le temps de réponse est plus faible pour le premier contrôleur, avec une moyenne de 8.66ms, contre 18ms pour le deuxième. Le dépassement est également plus grand pour le second contrôleur, 22% en moyenne. L'erreur en régime permanent est plus faible pour le second contrôleur, avec une moyenne de 0.7% contre 1.3% pour le premier.

Pour le lacet, il est attendu que la réponse soit plus lente, puisque l'autorité du drone pour cet axe est inférieure aux autres. Le premier contrôleur offre une meilleure performance, avec un temps de réponse de 12ms en moyenne, contre 24ms pour le second. L'erreur en régime permanent est également inférieure sur le premier contrôleur.

7.2 Sollicitation couplée des axes

Lorsque les trois axes sont sollicités en même temps, les contrôleurs changent énormément leur comportement. Les réponses dans chaque axe ne sont plus alors constantes et les temps de réponse plus grands.

Pour le premier contrôleur, de forts dépassement sont atteints, pour la première et la dernière commande, dans tous les axes. Les temps de réponse restent cependant relativement constants pour tous les axes.

La performance du second contrôleur est restée quasi-constante, sauf pour le temps de réponse

de la seconde commande, qui a augmenté.

7.3 Difficultés rencontrées

Le projet était, au final, parsemé de plusieurs embûches imprévues. En effet, l'intention de départ était d'utiliser la plateforme GymFC pour l'entraînement du réseau. Cependant, celle-ci agit un peu en "boite fermée", voulant dire qu'il est difficile pour un premier utilisateur de bâtir son propre algorithme d'apprentissage et de visualiser ses résultats. En effet, GymFC agit plus comme une solution tout en un pour entraîner un contrôleur à partir d'un modèle qui lui est fourni. Puisque l'optique de ce projet était la partie apprentissage, cette plateforme a été délaissée. Cependant, elle reste un outil puissant pour l'implémentation d'un vrai contrôleur, étant donné qu'elle compresse le réseau neuronal final et le met sous la bonne forme pour être utilisé avec NeuroFlight, un logiciel embarqué pour les contrôleurs de vol de drones.

Une autre difficulté rencontrée a été la création de l'environnement d'entraînement sur Pybullet-Drones. En effet, puisque la plateforme n'offre pas encore d'environnement pour le contrôle d'attitude, celui-ci a dû être bâti, ce qui fût plus long que prévu. Finalement, c'est en modifiant un environnement existant que cette tâche a été accomplie.

7.4 Recommandations

Finalement, les deux contrôleurs obtiennent des réponses similaires, mais le second réagit mieux aux sollicitations simultanées dans les trois axes. Il est alors recommandé d'utiliser la méthode d'entraînement "libre" pour la continuation du projet. De plus, dans l'optique d'implémenter un contrôleur semblable dans un vrai drone, il serait intéressant de modifier l'équation de récompense pour limiter l'effort de commande et le dépassement. La réponse du contrôleur pourrait alors être moins aggressive, donc mieux adaptée à une implémentation réelle.

En vue d'une implémentation sur un drone, il faudrait aussi modifier la sortie du réseau neuronal pour être une valeur de signal PWM, entre 0 et 100. En effet, étant donné que le contrôleur actuel fait varier le RPM des moteurs, ce n'est pas une implémentation réaliste pour un drone réel.

8 Conclusion

Pour conclure, l'objectif principal du projet, qui étaient de développer un contrôleur d'attitude de drone basé sur un réseau neuronal, a été rempli. En effet, un environnement d'entraînement utilisant le moteur de physique Pybullet-drones et un modèle de drone crazyfly a été développé, puis un algorithme d'entraînement basé sur la PPO a été programmée pour entraîner le contrôleur. Finalement, deux méthodes principales d'entraînement ont été explorées et comparées. Les outils développés ont aussi été rassemblés dans un GitHub, se trouvant au lien suivant :

Pour continuer le projet, l'équation de récompense pourrait être modifiée pour réduire le dépassement observé et limiter l'effort de commande. Il faudrait également changer la sortie du contrôleur pour que celui-ci soit sous forme de signal PWM, plus adapté au contrôle de drone. Le contrôleur pourrait alors être implémenté et testé sur un drone réel.

References

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., ... others (2020). What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*.

Eric, Y. Y. (2020). Coding ppo from scratch with pytorch. *Medium.com*.

Koch, W., Mancuso, R., West, R., & Bestavros, A. (2019). Reinforcement learning for uav attitude control. *ACM Transactions on Cyber-Physical Systems*, *3*(2), 1–21.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.