
CONTRÔLES DE MAVS PAR UN SNN OPTIMISÉ AVEC PPO

Jérémie Gince

Département d'informatique et de génie logiciel

Université Laval, Québec, Canada

jeremie.gince.1@ulaval.ca

2 mai 2022

RÉSUMÉ

Dans ce projet, les dynamiques neuronales *Leaky-Integrate-and-Fire* (LIF) et *Adaptative-Leaky-Integrate-and-Fire* (ALIF) venant de la neurosciences sont comparés avec un perceptron multicouche (MLP) venant de l'apprentissage machine sur une tâche de contrôle. Ladite tâche est de faire atterrir un micro aéronef (MAV) dans un environnement simplifié conçu avec le moteur de jeu Unity. Les réseaux de neurones artificiels, tous entraînés par renforcement avec l'algorithme *Proximal Policy Optimization* (PPO), montrent que l'accomplissement d'une telle tâche est réalisable, mais ne parviennent pas à atteindre des résultats permettant de passer à une étape plus complexe. En effet, il est montré que le MLP réussit à faire atterrir le MAV avec un seul degré de liberté en utilisant une caméra à flux optique et une caméra à événements tandis que les dynamiques ALIF et LIF n'y parviennent pas dans la majorité des cas. Finalement, un pipeline de développement pour l'accomplissement de cet objectif est présenté avec des pistes de perfectionnement pour la continuité du projet.

1 Introduction

Depuis que l'apprentissage profond est en mesure d'atteindre des résultats phénoménaux en vision numérique, la robotique s'intéresse énormément à ce type de méthodes d'optimisations afin d'améliorer la prise de décision d'un agent dans un environnement complexe. Bien que l'apprentissage profond révolutionne le monde de la robotique, un problème persiste, l'efficacité énergétique des réseaux de neurones. En effet, les modèles utilisés possèdent généralement un grand nombre de paramètres et doivent être exécutés sur GPU, ce qui est extrêmement exigeant en terme énergétique. C'est évidemment un problème en robotique mobile, puisqu'un grand besoin d'énergie requiert de grosses piles. Il va donc sans dire que l'utilisation d'un gros modèle pour les MAVs (*Micro air vehicle*) est problématique.

Heureusement, avec l'arrivée des processeurs neuromorphiques comme le processeur Loihi [1] d'Intel, il est possible d'utiliser la puissance des réseaux à impulsions (*spiking neural networks*) afin de faire du traitement d'image complexe à faible coût énergétique. C'est pourquoi, en s'inspirant du travail de [2], il est intéressant d'entraîner un réseau à冲动 afin de contrôler l'atterrissement d'un MAV dans un environnement complexe utilisant une caméra à flux optique. N'ayant pas de ressource matérielle à disposition, l'expérience s'est fait en utilisant le moteur de jeu Unity [3] et le module ML-Agent [4] pour la simulation de l'environnement.

L'objectif du présent projet est donc de contrôler l'atterrissement d'un drone quadrimoteur dans un environnement statique. Le MAV utiliser dans le courant projet, l'agent, devra donc apprendre à naviguer dans son environnement en maximisant des récompenses reçues. Afin d'y arriver, l'agent sera optimisé par apprentissage par renforcement. Une méthode de plus en plus utilisée en robotique pour mettre en valeur la puissance des réseaux de neurones artificielle. Plus spécifiquement l'algorithme *Proximal policy optimization* de Schulman et al. [5] est utilisé avec quelques simplifications pour permettre l'optimisation de l'agent dans le présent projet.

Finalement, le code du projet implémenté avec Python [6], Pytorch [7], C# [8], Unity [3] et ML-Agent [4] est disponible dans le répertoire github [9].

2 État de l'art

Dans un premier temps, Dupeyroux et al. [2] ont été en mesure de contrôler l'atterrissement d'un *Parrot Behbop 2 Drone* utilisant le processeur neuromorphique Loihi [1] d'Intel. Pour y arriver, ils ont simulé l'environnement avec le langage python où ils ont entraîné en réseau de neurones à impulsions à contrôler le drone. De plus, la dynamique neuronale utilisée dans cet article était un simple LIF (*leaky-integrate-and-fire*) sans connexions récurrentes doté de 10 et 5 neurones cachés. Ce simple modèle prend en entrée une mesure de flux optique discrétisé en un vecteur de 20 compartiments et génère des impulsions qui sont intégrées afin d'obtenir la puissance qui doit être envoyée aux moteurs du MAV.

Pour ce qui est de la boucle d'entraînement utilisée par Dupeyroux et al. [2], ils ont opté pour de l'apprentissage par évolution. Cette dernière consiste à lancer une grande quantité de simulations avec des modèles possédant des paramètres aléatoires et à garder en mémoire les plus efficaces. Les modèles plus performants sont donc croisés ou mélangés de façon aléatoire afin de générer de nouveaux modèles à tester. Après un grand nombre d'itérations, ils obtiennent un réseau de neurones optimisé et capable de faire atterrir un MAV sans que celui-ci s'écrase au sol.

L'apprentissage par évolution fonctionne bien pour des tâches simples et sur des modèles ne possédant que peu de paramètres à optimiser. De plus, ce type de tâche est très classique à l'apprentissage par renforcement. En effet, le problème courant possède un environnement, un agent et une fonction de récompense. Il est donc logique d'utiliser de l'apprentissage par renforcement afin d'optimiser les paramètres du modèle. De plus, celui-ci est dérivable ce qui donne la possibilité d'utiliser la descente de gradient afin de minimiser une fonction d'erreur ou de façon équivalente, maximiser une fonction de récompense.

Afin d'effectuer une telle optimisation par renforcement, les travaux de Schulman et al. [5] seront utilisés. Dans ce dernier article, les auteurs proposent un algorithme d'optimisation, *Proximal policy optimization*, étant extrêmement efficace dans un environnement avec des récompenses éparses. Non seulement l'algorithme est aussi, voir plus, performant que [10], [11], mais est très simple mathématiquement. Avec une telle méthode d'apprentissage, ils sont en mesure de faire apprendre des tâches très complexes comme celles de [12](gym) et de [13](Roboschool) à un modèle d'apprentissage profond classique.

Afin d'être en mesure de faire prendre des décisions à un agent, il faut absolument lui donner des observations de son environnement. Pour des réseaux de neurones artificiels classiques, donner des observations est relativement simple puisqu'il n'y a pas beaucoup de contraintes. Toutefois, pour des réseaux de neurones à impulsions, il existe une contrainte complexifiant les choses, les observations se doivent d'être des impulsions. Dans [2], les auteurs discrétilisent l'entrée du réseau comme expliquer précédemment. Dans le projet courant, les observations de l'environnement pourront se faire à l'aide d'une caméra et les données de ce capteur seront envoyées au réseau. Il faut donc s'assurer qu'un réseau à impulsion soit en mesure de traiter ce type de données. C'est ce qui est fait dans le travail de Gince et Lamontagne-Caron [14]. Les auteurs de ce projet ont montré qu'il était possible de transformer des images en impulsions et de classifier celles-ci avec un SNN. Ils ont aussi à comparer plusieurs dynamiques neuronales afin d'être en mesure de juger qu'elle serait la plus appropriée pour une telle tâche. En addition, leur travail est inspiré de [15] montrant diverses techniques pour effectuer l'apprentissage d'un réseau de neurones à impulsions à l'aide de descente de gradient.

3 Méthodologie

La prédiction d'actions de l'agent placé dans son environnement s'exécute à l'aide de trois parties. Celles-ci sont le traitement de l'état de l'environnement, le traitement des données de l'environnement par les capteurs et finalement, la transformation des observations des capteurs par un modèle en un vecteur d'actions exécuter par un MAV. La relation entre ces trois compartiments est illustrée à la figure 3.1 et ils seront détaillés respectivement dans les sections 3.1, 3.2 et 3.3.

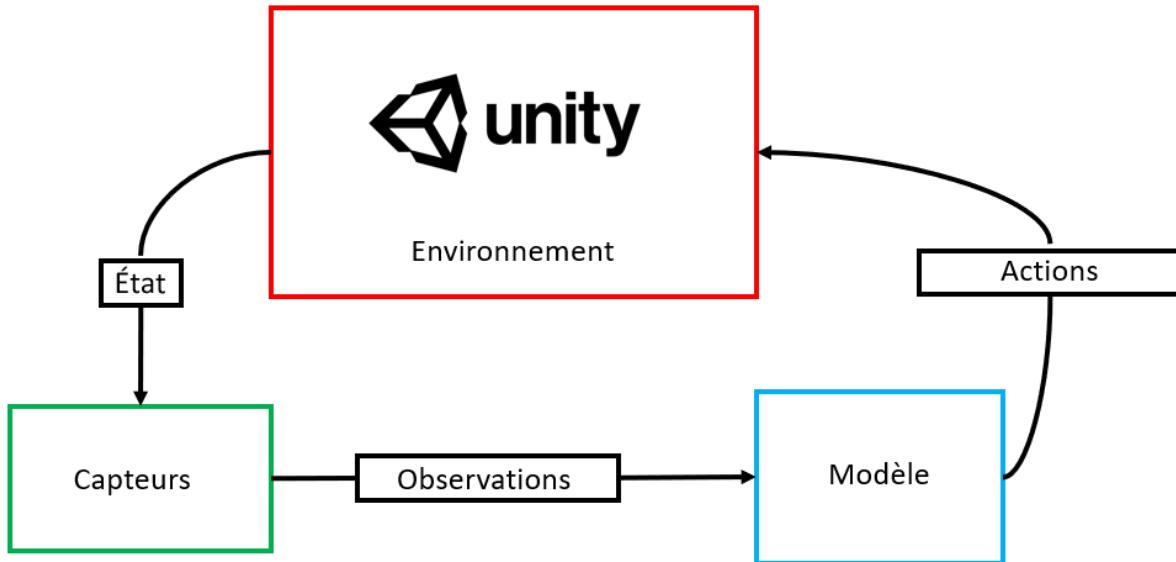


FIGURE 3.1 – Pipeline général du flux d'information sortant de l'environnement et revenant à celui-ci en passant par les capteurs et le modèle d'apprentissage.

3.1 Environnement

Le présent projet a pour buts de simplement démontrer qu'il est possible d'utiliser un réseau de neurones à impulsions pour contrôler un MAV ainsi que d'implémenter un pipeline d'apprentissage par renforcement pour ce même type de réseau. Le MAV utilisé dans la présente expérience est illustré à la figure 3.2.



FIGURE 3.2 – Image montrant le MAV quadrimoteur utilisé comme agent.

Pour simplifier le plus possible l'implémentation du tout, l'environnement se doit d'être le plus simple possible. Celui-ci si se résume donc à un plateau avec une texture riche ainsi que des marqueurs aléatoires aux extrémités. Le plateau est illustré à la figure 3.3 où l'on voit clairement une texture de bois sur un sol gris.

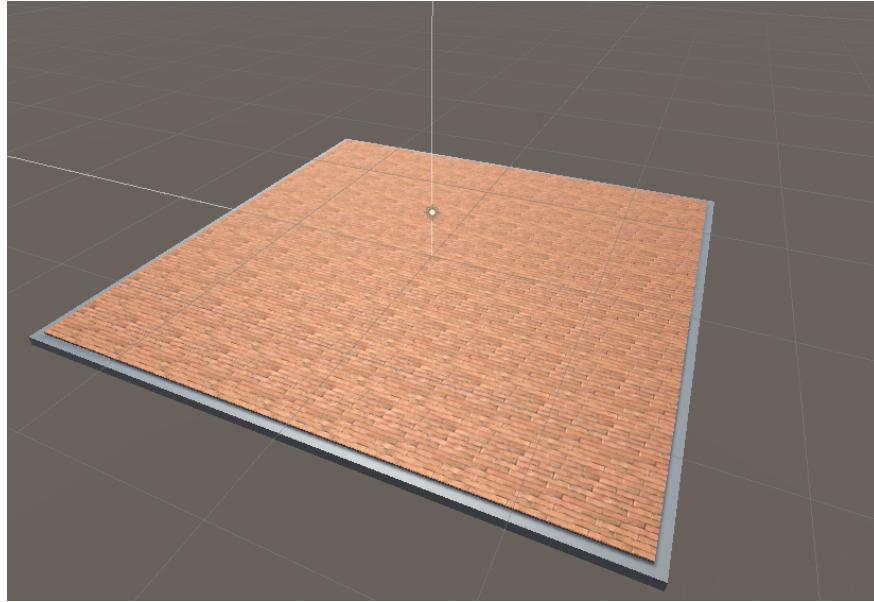


FIGURE 3.3 – Image illustrant le sol de l'environnement utilisé. Il est possible de voir que la texture sur le sol qui est très utile pour les caméras à trouver des points clés.

La texture appliquée sur le sol est extrêmement importante puisqu'elle permet aux capteurs de détecter des points clés comme des coins dans les images produites par les caméras. Dans le même ordre d'idée, l'environnement possède des marqueurs de types différents : capsule, cube, sphère et cylindre positionné de façon aléatoire afin d'ajouter de la texture.

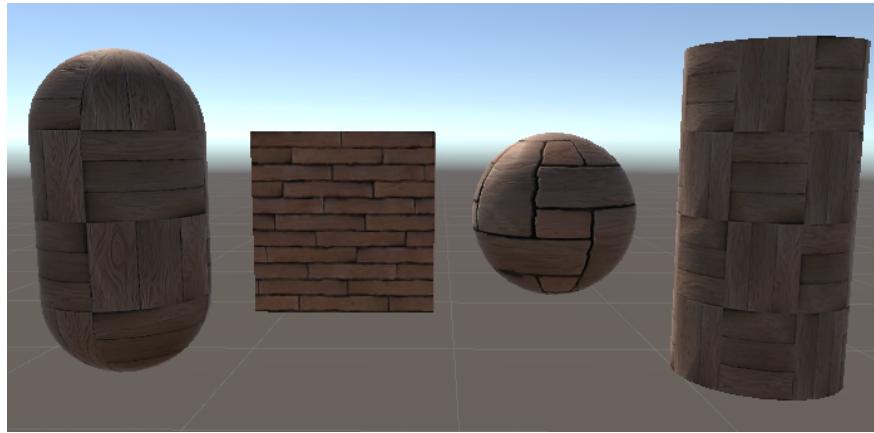


FIGURE 3.4 – Image illustrant les différents types de marqueurs utiliser pour mettre de la texture supplémentaire dans l'environnement. De plus, les marqueurs sont dotés de 3 types de textures différentes qui peuvent aussi être mises sur le sol.

Ces marqueurs sont illustrés à la figure 3.4. De plus, dans cette dernière, il est possible de voir trois différents types de matériel appliqués sur les marqueurs. Ces matériaux sont choisis aléatoirement sur chaque marqueur et sur le sol en début d'épisode. Ceci apporte encore plus de texture aléatoire à l'environnement.

Finalement, il est possible de voir le MAV dans l'environnement en début d'épisode à la figure 3.5. Il est ainsi possible de voir que l'environnement est riche en texture et pourra permettre aux capteurs d'obtenir de l'information sur ce qui entoure le MAV et donc sur son état.

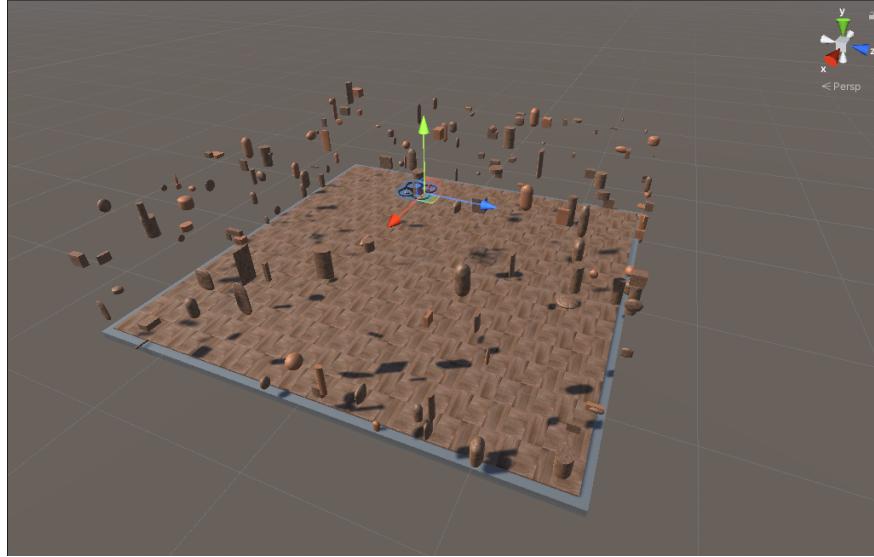


FIGURE 3.5 – Image illustrant le MAV avec des marqueurs distribués de façon aléatoire dans l'environnement.

3.1.1 Configurations

Le contrôle d'un MAV dans un environnement possédant une physique réaliste est extrêmement difficile à accomplir. Pour cette raison, deux types de configuration sont considérés. La première est la configuration à un seul degré de liberté. C'est la configuration simple où le MAV peut seulement se déplacer dans son axe y (l'altitude ou la hauteur). Puisqu'il y a un seul degré de liberté, le MAV peut maintenant être vu comme un agent à une dimension possédant un seul moteur. À ce moment, les modèles de prédictions d'actions se doivent de retourner un vecteur à une seule entrée représentant la puissance du moteur à appliquer.

La deuxième configuration est la plus réaliste. Le MAV possède maintenant 6 degrés de liberté. Trois degrés pour la position en x , y et z du robot et trois autres degrés de liberté pour les angles d'Euler en θ_x , θ_y et θ_z . Dans cette configuration, le modèle de prédiction se doit de retourner un vecteur à quatre entrées, soit une pour chaque puissance à appliquer à chacun des moteurs. Cette configuration est considérée comme très difficile à résoudre puisque le MAV peut rapidement se retrouver dans un angle défavorable qui le fera s'écraser tragiquement au sol.

3.1.2 Objectif de l'agent

L'agent étant l'acteur se développant dans l'environnement en observant des observables et en exécutant des actions doit atteindre un objectif. Celui-ci est défini simplement dans le but de faire atterrir le MAV convenablement sur la plateforme. Les critères afin qu'un agent soit considéré comme ayant accompli l'objectif sont les suivants :

- Module de vitesse linéaire plus petit que 1 m/s ;
- Module du vecteur d'angles d'Euler du robot plus petit que 15° ;
- Module de vitesse angulaire plus petit que $5^\circ/\text{s}$.

Si l'agent accomplit l'objectif en respectant ces derniers critères, il obtient une récompense de 1. Dans le cas contraire, s'il ne respecte pas un seul des critères ou qu'il sort de la zone attribuée (la plateforme), il obtient une récompense de -1.

3.2 Capteurs

Les capteurs embarqués sur le MAV prennent de l'information sur l'environnement l'entourant et fournissent ces observations au modèle de prédiction d'actions. Afin de choisir quels types de capteurs sont utiles pour la tâche courante, il faut se questionner sur le type de modèle utiliser ainsi que la dynamique physique du système. Dans le cas présent, le MAV est en chute avec la force gravitationnelle vers le sol et l'objectif est essentiellement de ne pas tomber trop rapidement. Il est alors logique d'utiliser des types de capteurs donnant de l'information sur la vitesse de chute en fonction de la hauteur du MAV. La caméra à flux optique (3.2.1) est très bien adaptée pour ce type de problème.

Dans le cas où le MAV possède plus qu'un degré de liberté et se met à tourner dans toutes les directions, la caméra à flux optique étant positionnée sur le ventre du MAV n'est plus suffisante. En effet, ce capteur est utile seulement s'il est

dirigé vers le sol. Dans ce cas, la caméra à événements (section 3.2.2) devient le capteur de choix, et ce surtout pour des réseaux de neurones à impulsions.

3.2.1 Caméra à flux optique

La méthodologie de ce capteur à flux optique provient de Dupeyroux et al. [2]. Cette caméra est installée sur le ventre du MAV et est dirigée vers le sol. Le but de ce capteur est de donner de l'information sur la vitesse et la hauteur de l'appareil utilisant une simple caméra. Pour ce faire, il faut se baser sur l'équation du flux optique décrite à l'équation 1 :

$$\hat{D}(t) = \frac{1}{N_D} \sum_{i=1}^{N_D} \frac{1}{\Delta t} \frac{\ell_i(t) - \ell_i(t - \Delta t)}{\ell_i(t - \Delta t)} \quad (1)$$

où N_D est le nombre de pairs de point clés, Δt est le temps entre deux images consécutives et $\ell_i(t)$ est la distance entre la paire i de point clé au temps t . Afin d'être capables de trouver ces points clés, le détecteur de coins FAST [16] est utilisé. Puisque les coins sont détectés sur des images subséquentes dans le temps, ils doivent être appariés entre ces dernières. Comme dans [2], l'algorithme d'appariement de points utilisé est le *pyramidal Lucas-Kanade tracker* [17].

Le flux optique étant une valeur réelle peut être tout simplement donné à un réseau de neurones comme le MLP sans problème. Par contre, les réseaux de neurones à impulsion requièrent des données binaires en entrée représentant une série temporelle d'impulsions. La valeur de flux optique se doit donc d'être transformée pour ce type de réseau. Pour ce faire, la valeur de flux optique sera transformée en un vecteur de taille n représentant n boîtes de valeurs. Ce vecteur aura la forme : $[-\infty, -b(n/2), \dots, 0, \dots, b(n/2), \infty]$ avec b étant la taille des boîtes. Ce vecteur contiendra donc des zéros et une seule valeur 1 représentant l'index de la valeur de flux optique pouvant être calculé avec l'équation 2. Ce type de vecteur aussi communément nommé un *one hot vector* en apprentissage machine.

$$I = \text{Floor}[\text{Clamp}(v/b, -n/2, n/2) + n/2] \quad (2)$$

Le flux optique est désormais le bon type de donnée à fournir à un réseau de neurones à impulsions. Le seul détail restant est que ce vecteur est empilé t_{int} fois dans le temps pour représenter une série temporelle de la valeur de flux optique.

3.2.2 Caméra à événements

La caméra à événements est utilisée pour fournir de l'information sur le changement d'intensité visuel à l'agent. Ce changement d'intensité peut être perçu comme une vitesse de déplacement des objets dans la scène relativement à la caméra. De plus, l'information est stockée comme des impulsions. Celle-ci est donc très adaptée pour les SNN qui requièrent des impulsions en entrée. En prenant des images subséquentes au fil du temps, il est possible d'obtenir l'image à événements S_x avec l'équation 3. La matrice S_x contient donc des zéros et des uns aux endroits où le changement d'intensité entre deux images subséquentes dépasse un certain seuil en échelle logarithmique.

$$S_x^t = H(\log(|x^t - x^{t-1}|) - s_{th}) \quad (3)$$

Dans l'équation 3, la fonction $H(\cdot)$ est la fonction Heaviside montrée à l'équation 9, x^t est l'image venant de la caméra au temps t et s_{th} est le seuil d'activation équivalent à 10 dans l'expérience courante.

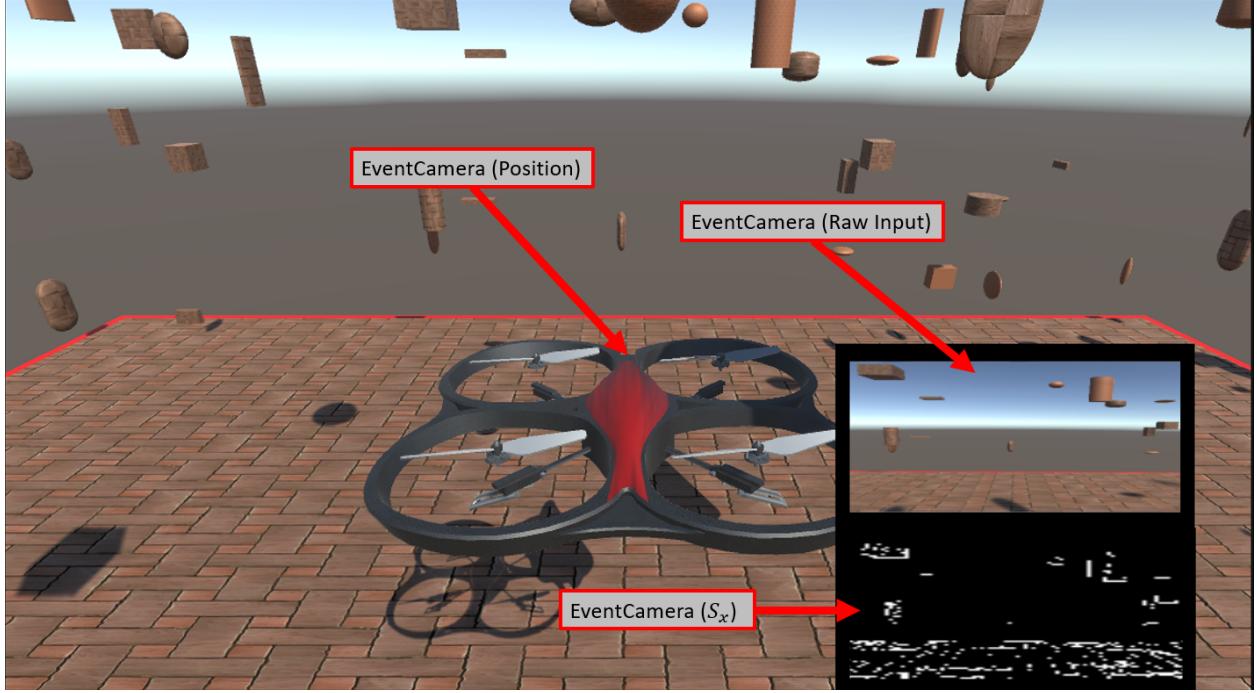


FIGURE 3.6 – Démonstration de la caméra à évènements (*EventCamera*) embarquée sur le MAV dans son environnement. Dans le coin inférieur droit, l'image *EventCamera (Raw Input)* est l'entrée brute de la caméra étant la variable x^t dans l'équation 3. En dessous de cette dernière, *EventCamera (S_x)* correspond à la sortie du capteur étant la variable S_x dans l'équation 3. Le drone étant en mouvement, il est possible de voir des impulsions dans les variations des textures de l'environnement.

Pour illustrer l'entrée et la sortie de ce capteur, il est possible de voir un exemple du MAV en mouvement dans l'environnement dans la figure 3.6. Dans cette dernière, les coins et les variations rapides dans les textures des objets de l'environnement se voit très bien dans la sortie S_x de la caméra à évènement.

3.3 Modèles

Les modèles comparés dans la présente étude viennent d'un côté de l'apprentissage profond classique et d'un autre côté de la neurosciences. Le premier modèle qui est donc testé est le perceptron multicouche (MLP) et les deux autres modèles provenant de la neurosciences sont en fait deux variantes de la même dynamique neuronale *leaky-integrate-and-fire*. Le pipeline général du flux d'information de l'environnement au modèle peut être observé à la figure 3.1.

Dans cette dernière figure, les observations (la sortie des capteurs) sont envoyées au modèle et ce dernier retourne des actions qui modifieront l'état de l'environnement. Maintenant, le pipeline associé au MLP est très direct puisqu'il consiste simplement à mettre les observations dans un vecteur qui passe dans des couches cachées et en ressort un vecteur d'actions (section 3.3.1). Pour ce qui est des SNN, le pipeline est moins direct et se fait en deux étapes distinctes. Ce pipeline est représenté par la figure 3.7.

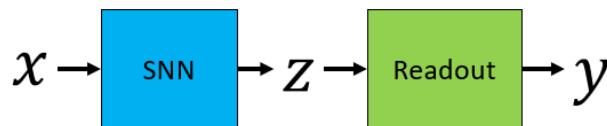


FIGURE 3.7 – Pipeline général du flux d'information sortant de l'environnement et revenant à celui-ci en passant par les capteurs et le modèle d'apprentissage.

Dans la figure 3.7, le bloc SNN (*Spiking Neural Network*) est la dynamique neuronale à impulsions. Les différentes dynamiques pouvant être utilisées comme SNN seront présentées dans les sections 3.3.2 pour LIF et 3.3.3 pour ALIF. Cette dynamique prend en entrée un ensemble d'impulsions x et retourne un autre ensemble d'impulsions z étant les impulsions des neurones de sortie (de la dernière) couche du bloc SNN.

Le bloc nommé *Readout* désigne la dynamique de lecture. En effet, le SNN retourne des impulsions qui ne peuvent pas être représentées comme un vecteur d'actions. Il faut donc effectuer une transformation sur ces impulsions de sorties pour les transformer en un vecteur de valeurs continues de même taille que le nombre d'actions de l'agent. Afin d'y arriver, la dynamique expliquée à la section 3.3.4 est utilisée.

3.3.1 MLP (multi-layer perceptron)

Le perceptron multicouche est un modèle d'apprentissage profond extrêmement utilisé en apprentissage automatique et est certainement le plus populaire. Sa popularité vient certainement de son arrivée historiquement précoce et pour sa simplicité. En effet, ce type de modèle est extrêmement simple et peut se résumer à l'équation 4.

$${}^\ell y = \text{ReLU} \left(\sum_i^{N_{\ell-1}} {}^\ell W_{ij}^{\text{in}} \cdot {}^\ell x_i + {}^\ell b \right) \quad (4)$$

$$\text{ReLU}(x) = \max(0, x) \quad (5)$$

Dans l'équation 4, ℓ réfère à l'indexe de la couche cachée. Dans la figure 3.8, il est possible de voir deux couches cachées utilisant la fonction ReLU. La dernière couche est considérée comme la couche de sortie et n'utilise pas la ReLU, car la sortie du réseau se doit d'être une valeur continue négative ou positive.

Dans la figure 3.8, la couche *Flatten* sert à mettre toutes les observations, peu importe leur type en un unique vecteur en une dimension. Ce dernier peut être considéré comme un vecteur de caractéristiques qui sera passé à la première couche du réseau.

3.3.2 Dynamique LIF (leaky-integrate-and-fire)

Cette section ainsi que les sections 3.3.3 et 3.3.4 sont tirés de Gince et Lamontagne-Caron [14], et font une revue des dynamiques utilisées pour le réseau de neurones à impulsions.

La dynamique LIF, inspiré de [15], [18], modélise le potentiel synaptique et les impulsions d'un neurone au fil du temps. La forme de ce potentiel n'est pas considérée comment réaliste [19], mais le temps auquel le potentiel dépasse un certain seuil l'est. Ce potentiel est trouvé par l'équation récurrente 6.

$$V_j^{t+\Delta t} = \left(\alpha V_j^t + \sum_i^N W_{ij}^{\text{rec}} z_i^t + \sum_i^N W_{ij}^{\text{in}} x_i^{t+\Delta t} \right) (1 - z_j^t) \quad (6)$$

Les variables de l'équation 6 sont décrites par les définitions suivantes :

- N désigne le nombre de neurones du réseau ;
- V_j^t désigne le potentiel du neurone j au pas de temps t ;
- Δt désigne le pas de temps de l'intégration ;
- α est la constante de décroissance du potentiel au fil du temps (équation 7) ;

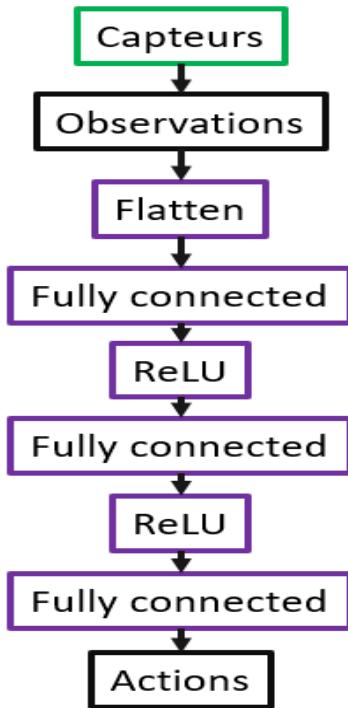


FIGURE 3.8 – Pipeline du modèle MLP (perceptron multicouche) pour la prédiction d'actions à partir d'observations fournies par les capteurs.

- x^t désigne l'entrée du réseau au temps t ;
- W_{ij}^{rec} désigne la matrice de connexions récurrentes avec la convention que i est connecté vers j ($i \rightarrow j$) ;
- W_{ij}^{in} désigne la matrice de connexions avec l'entrée du réseau ;
- z_j^t désigne la sortie du neurone j au temps t (équation 8).

La constante de décroissance du potentiel est décrite par l'équation suivante :

$$\alpha = e^{-\Delta t / \tau_m} \quad (7)$$

avec τ_m étant la constante de temps de décroissance du potentiel membranaire qui est généralement 20 ms.

La sortie du neurone j au temps t dénotée z_j^t est définie par l'équation 8 :

$$z_j^t = H(V_j^t - V_{\text{th}}) \quad (8)$$

où V_{th} désigne le seuil d'activation du neurone et la fonction $H(\cdot)$ est la fonction Heaviside définie comme 9 :

$$H(x) = \begin{cases} 1 & \text{si } x > 0; \\ 0 & \text{sinon.} \end{cases} \quad (9)$$

Il est à noté que le potentiel est intégré pendant T pas de temps avec l'intégration d'Euler utilisant un pas de temps Δt .

Finalement, cette dynamique peut facilement se généraliser à plusieurs couches de neurones. Il suffit d'ajouter un indice ℓ à toutes les variables d'état indiquant l'index de la couche. L'équation 6 peut donc être généralisée à plusieurs couches. Pour cette généralisation, l'équation 10, la couche avec $\ell = 0$ se voit être la couche d'entrée du réseau.

$${}^\ell V_j^{t+\Delta t} = \left(\alpha \cdot {}^\ell V_j^t + \sum_i^{N_\ell} {}^\ell W_{ij}^{\text{rec}} \cdot {}^\ell z_i^t + \sum_i^{N_{\ell-1}} {}^\ell W_{ij}^{\text{in}} \cdot {}^\ell x_i^{t+\Delta t} \right) (1 - {}^\ell z_j^t) \quad (10)$$

Les variables supplémentaires de l'équation 10 sont décrites par les définitions suivantes :

- ℓ est l'indice de la couche ;
- N_ℓ désigne le nombre de neurones de la couche ℓ ;
- l'entrée e de la couche ℓ , étant ${}^\ell x_i^t$, correspond à la sortie de la couche précédente, donc ${}^\ell x_i^t = {}^{\ell-1} z_j^t$ avec ${}^{-1} x_i^t = S_x^t$ étant l'entrée du réseau.

3.3.3 Dynamique ALIF (adaptive leaky-integrate-and-fire)

La dynamique ALIF, inspiré de Bellec et al. [18], est très semblable à la dynamique LIF (section 3.3.2). En fait, ALIF possède exactement la même équation de mise à jour de potentiel que LIF (équations 6 et 10). La différence vient du fait que le potentiel de seuil varie avec le temps et l'entrée du neurone. En effet, le seuil est augmenté à chaque impulsion en sortie et est par la suite diminuée avec un certain taux afin de revenir à son seuil de départ V_{th} . L'équation de seuil 8 se voit donc légèrement modifier en changeant $V_{\text{th}} \rightarrow A_j^t$. Donc, la sortie du neurone j au temps t dénoté z_j^t est redéfinie par l'équation 11 :

$$z_j^t = H(V_j^t - A_j^t). \quad (11)$$

La mise à jour du seuil d'activation est alors décrite par 12 :

$$A_j^t = V_{\text{th}} + \beta a_j^t \quad (12)$$

avec la variable d'adaptation a_j^t décrite par 13 et β un facteur d'amplification plus grand que 1 et typiquement équivalent à $\beta \approx 1.6$ [18].

$$a_j^{t+1} = \rho a_j^t + z_j^t \quad (13)$$

Avec le facteur de décroissance ρ comme :

$$\rho = e^{-\Delta t / \tau_a} \quad (14)$$

Un grand avantage de la dynamique ALIF comparativement à LIF est qu'il tend à avoir plus de mémoire dans le temps. En effet, comme Bellec et al. [18] l'ont montré, la dynamique ALIF est analogue à un *long short-term memory* (LSTM) en apprentissage automatique. Cette mémoire accrue serait due essentiellement au fait que ALIF possède plus de variables d'états gardant de l'information au fil du temps. La première variable d'état est le potentiel qui est intégré dans le temps et réinitialisé à chaque impulsion. La seconde est le seuil adaptatif qui n'est jamais réinitialisé tout au long de l'intégration. De son côté, LIF possède seulement le potentiel qui varie dans le temps, donc il garde de l'information sur les pas de temps précédent jusqu'à la prochaine impulsion où le neurone "oublie".

3.3.4 Dynamique de lecture (*Readout Layer*)

Les dynamiques à impulsions retournent évidemment une série d'impulsions pour chaque neurone de sortie, mais cela n'est pas compatible avec la prédiction d'actions. En effet, pour effectuer une telle prédiction, la sortie du réseau se doit d'être un vecteur de valeurs continues où chaque entrée correspond à la puissance envoyée dans les moteurs du MAV. Afin d'être en mesure d'avoir un tel type de sortie de réseau avec des SNN, il suffit d'intégrer au fil du temps les impulsions de sortie du SNN. C'est ce qui est fait par ce qui est couramment appelé la couche de lecture *Readout Layer*. Cette intégration dans le temps se fait à l'aide de l'équation 15 inspirée de Bellec et al. [18].

$$y_k^{t+1} = \kappa y_k^t + \sum_j W_{kj}^{\text{out}} z_j^t + b_k^{\text{out}} \quad (15)$$

Les paramètres de l'équation 15 sont :

- y_k^t la sortie de la couche au temps t pour la classe k ;
- κ la constante de décroissance exprimée par $\kappa = e^{-\Delta t/\tau_{\text{out}}}$;
- W_{kj}^{out} les poids synaptiques de la couche de lecture ;
- b_k^{out} le poids de biais de la couche de lecture associée à la classe k ;
- z_j^t la sortie du neurone j au temps t de la couche de sortie du SNN.

3.4 Entraînement

La boucle d'apprentissage pour l'entraînement des modèles est illustrée à la figure 3.9. Dans cette dernière, l'objet principal à comprendre en commençant est le *Trainer*. Celui-ci sert à optimiser les paramètres du modèle à l'aide d'expériences venant de l'environnement. Une expérience peut être décrite par la liste d'objets suivants :

- Observation : sortie des capteurs ;
- Actions : les actions exécuter après l'observation ;
- Récompense : récompense obtenue après l'application des actions ;
- Terminal : étiquette indiquant si l'agent est arrivé à la fin de l'épisode ;
- Prochaine observation : observation reçue après l'exécution des actions ;
- Récompense espérée : récompense espérée à la fin de l'épisode ;
- Avantage : avantage de l'application des actions sur l'état de l'agent.

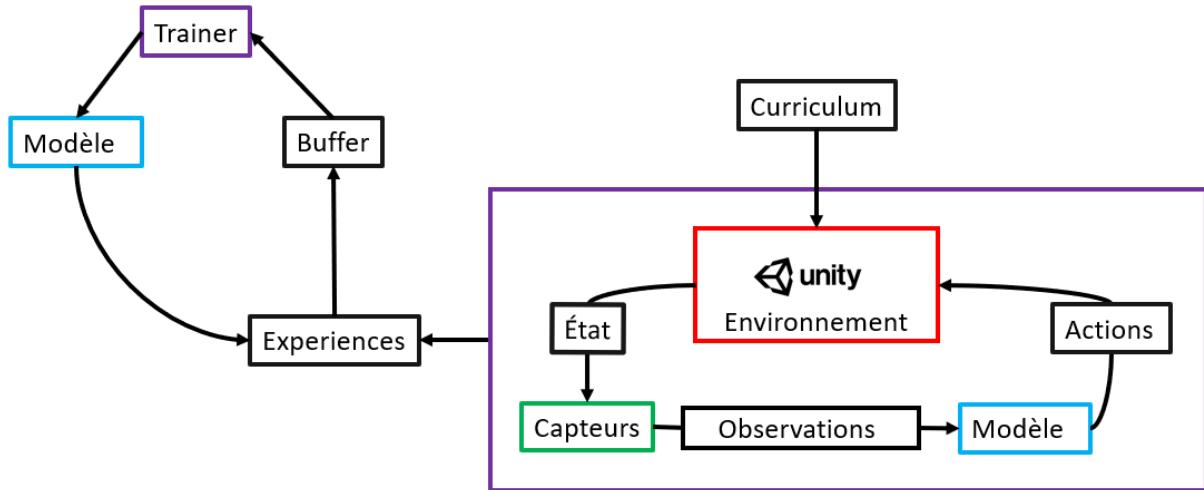


FIGURE 3.9 – Schéma de la boucle d'apprentissage utilisé pour l'entraînement des modèles.

En continuant dans la taxonomie, une trajectoire est l'ensemble des expériences d'un agent obtenues du début d'un épisode jusqu'à la fin de celui-ci. C'est-à-dire de la première expérience jusqu'à ce que l'étiquette *terminal* devienne vraie. Ces expériences sont stockées dans l'objet *Buffer* qui contient un nombre limité de données. Lorsque le conteneur est rempli, la méthode classique est de retirer les plus vieilles expériences. Dans cette étude, le *Buffer* pourra aussi supprimer les expériences contenant le plus petit avantage en valeur absolue. Cet avantage sera défini dans la section

3.4.1. Cette méthode de gestion de mémoire permettra de garder seulement les expériences qui aideront à l'apprentissage de l'agent.

Dans un premier temps, le *Buffer* sera partiellement remplie en générant des trajectoires à l'aide du modèle initialisé aléatoirement. Ceci permettra d'avoir suffisamment de données pour faire le premier entraînement. Par la suite, le modèle est entraîné par descente de gradient sur un sous-ensemble des données du *Buffer*. Plus précisément, l'algorithme PPO (*Proximal Policy Optimisation*) vu à la section 3.4.1 est utilisé pour faire l'optimisation du réseau sur les données d'entraînement. Une fois fait, de nouvelles expériences sont prises avec le modèle partiellement optimisé et sont stockées dans le *Buffer*. Le tout est répété un certain nombre de fois afin d'obtenir un modèle optimisé.

Le dernier objet qui n'a pas encore été abordé de la figure 3.9 est le *Curriculum*. Ce dernier est utile pour altérer l'environnement tout au long des itérations d'entraînement. Il faut bien comprendre que la tâche à accomplir est extrêmement difficile à faire pour un réseau initialisé aléatoirement. Pour donner une chance au modèle de réussir quelques expériences et de s'optimiser avant de tenter une tâche trop difficile, l'environnement est simplifié en début d'entraînement et se complexifie de plus en plus que l'agent progresse dans son apprentissage. Le *Curriculum* est donc un ensemble de leçons à accomplir pour l'agent. Les premières leçons sont simples et les dernières sont plus complexes et现实的.

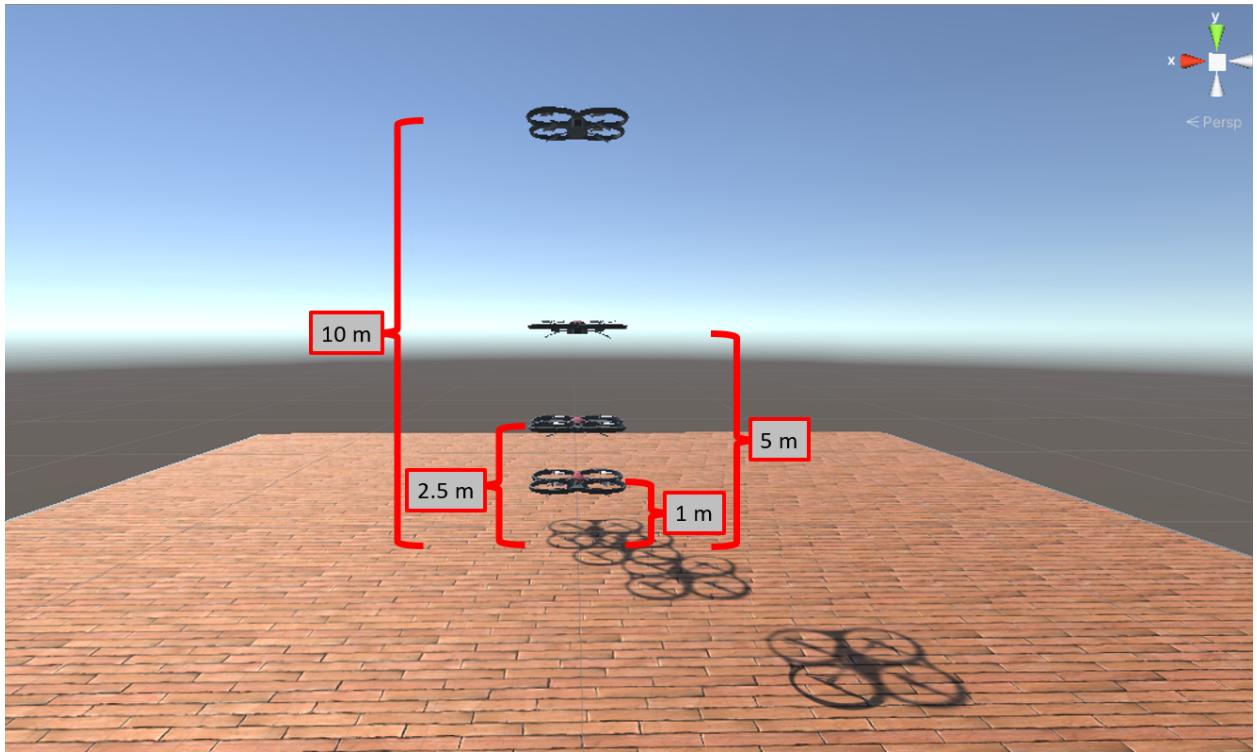


FIGURE 3.10 – Illustration des différentes hauteurs maximales que peut prendre le MAV en début d'épisode tout au long du curriculum.

Dans la figure 3.10, il est possible de voir différentes hauteurs maximales que peut prendre le MAV en début d'épisode. En effet, à chaque début d'épisode l'environnement place l'aéronef à une hauteur aléatoire entre $y_{min} = 1$ et y_{max} . Cette hauteur maximale est déterminée par le *Curriculum* dépendamment de la leçon où est rendu l'agent. Dans l'étude présente, le *Curriculum* possède 50 leçons où chacune d'entre elles est accomplie du moment que l'agent a réussi 5 épisodes à plus de 0.9 de récompense en moyenne. Les 37 premières leçons (3/4 de 50) vont de 1 mètre d'altitude à 5 mètres à intervalles réguliers et les autres leçons vont de 5 à 10 mètres de hauteur encore à intervalles réguliers.

3.4.1 PPO *Proximal Policy Optimization*

L'algorithme d'optimisation *Proximal Policy Optimization* proposé par Schulman et al. [5] est comme dit précédemment, considéré comme l'état de l'art pour l'apprentissage par renforcement. Comme tout problème d'apprentissage machine

ayant comme objectif de minimiser une fonction par descente de gradient, une fonction de perte est requise. C'est en fait cette fonction de perte qui donne toute la puissance à PPO. Celle-ci est représentée par l'équation 16.

$$L^{\text{CLIP}} = -\frac{1}{D} \sum_i^D \min(r_i(\theta) \hat{A}_i, \text{clip}(r_i(\theta), 1 - \varepsilon, 1 + \varepsilon) \hat{A}_i) \quad (16)$$

Dans l'équation 16, le terme $r_i(\theta)$ est représenté par l'équation 17, i est l'indexe de la donnée d'entraînement, θ est l'ensemble des paramètres du modèle à optimiser et \hat{A}_i est l'avantage calculer à partir de l'équation 18 pour l'étude actuelle.

$$r_i(\theta) = \frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{\text{old}}}(a_i|s_i)} \quad (17)$$

L'équation 17 représente le ratio entre les prédictions des deux estimateurs. En effet, l'estimateur π_θ est le réseau à son état courant tandis que $\pi_{\theta_{\text{old}}}$ est le même réseau, mais à l'itération d'entraînement précédent. De plus, les variables a_i et s_i correspondent respectivement à l'ensemble d'actions et à l'ensemble d'observations pour l'expérience i .

$$\hat{A}_t = \sum_{j=t}^{T-1} \gamma^{j-t} r_j \quad (18)$$

De son côté, l'équation 18 représente l'avantage de prendre une certaine action à l'observation s_i . Dans cette dernière, l'indice t représente le pas de temps de la trajectoire du MAV et la variable r_t est la récompense obtenue au temps t . Dans le cas présent, l'agent reçoit une récompense de 1 ou de -1 seulement à la toute fin de sa trajectoire. Dans ce cas, l'avantage peut être vu comme une décroissance de la récompense par un facteur γ dans le temps inverse (l'avantage à $t = T$ sera équivalent à r_T tandis que l'avantage à $t = 1$ sera équivalente à $\gamma^T r_T$). Le facteur de décroissance γ se doit être plus petit que 1 et est généralement équivalent à 0.99.

4 Résultats

4.1 Buffer

Dans un premier temps, le type de *Buffer* est testé afin de savoir si et comment ce paramètre affect la vitesse de convergence du modèle. Comme expliqué précédemment dans la section 3.4, les expériences peuvent être retirées du *Buffer* de deux façons : en retirant la plus vielle expérience ou en retirant l'expérience possédant le plus petit avantage en absolue. Ces deux types de *Buffer* différents sont essayés pour diverses raisons. Le premier type est utilisé dans Schulman et al. [5] et en général dans la communauté en apprentissage par renforcement puisque c'est simple et que cela semble bien fonctionner. D'un autre côté, l'équation 16 montre que la fonction de perte du réseau et donc la mise à jour des poids est directement proportionnelle à l'avantage associé aux expériences. Donc une expérience avec un avantage près de zéro n'apportera pas d'information supplémentaire pour la mise à jour des paramètres et n'y contribuera pas non plus. Pour cette raison, il est mieux, et ce sera montré dans les prochains résultats, de retirer les expériences ayant le moins d'impact sur la mise à jour des paramètres du réseau. En effet, c'est pour ne pas perdre ceux qui apportent beaucoup d'information même si ces derniers ont été expérimentés il y a longtemps.

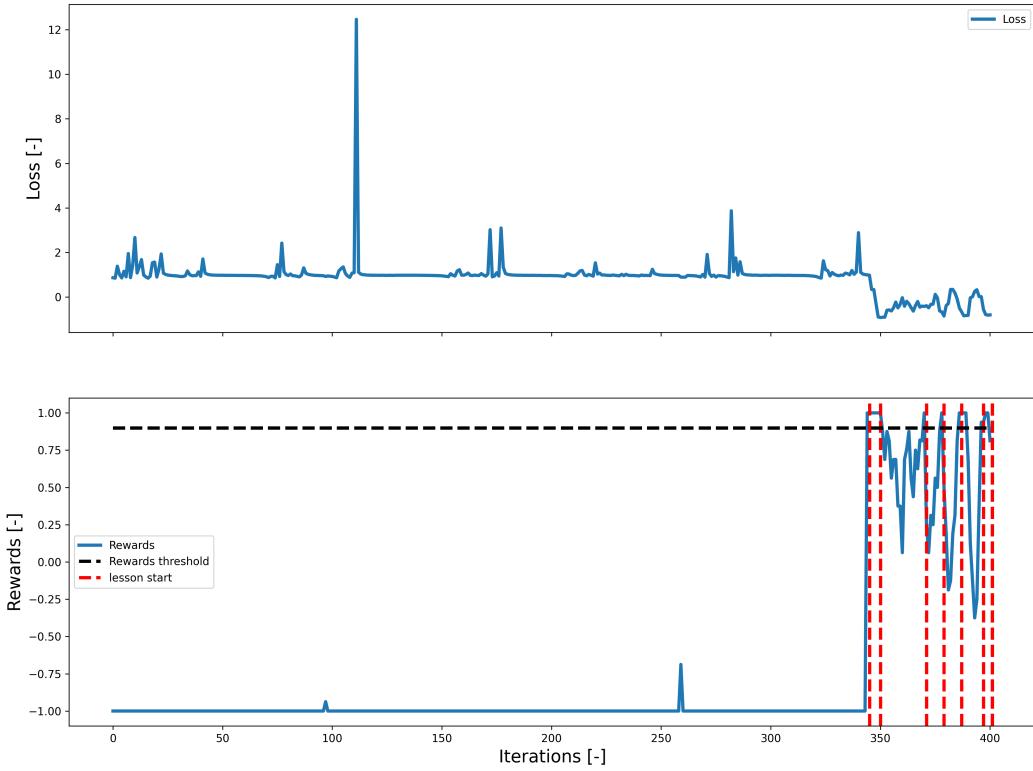


FIGURE 4.1 – Contrôle du MAV avec un MLP de neurones cachés 128x128 avec la position de l'aéronef en entrée. L'environnement à un seul degré de liberté est utilisé avec le *Buffer* retirant le plus vieux élément.

Pour comparer seulement le type de *Buffer*, un MAV contrôlé par un MLP ayant la position de l'aéronef en entrée seulement est placé dans un environnement à un seul degré de liberté. Ce robot est donc entraîné avec le *Curriculum* présenté à la section 3.4 avec les deux types de *Buffer*.

La figure 4.1 montre la perte et les récompenses moyennes du modèle en fonction des itérations d'entraînement pour le MLP avec le *Buffer* FIFO (première expérience entrée, première sortie). Il est possible de voir que l'algorithme commence à converger autour de 350 itérations pour atteindre le premier objectif du *Curriculum*. De son côté, la figure 4.2 montre l'entraînement du même MLP, mais cette fois avec le *Buffer* Priority (l'expérience avec le moins d'avantages en absolu est retirée). Ce dernier commence à converger à environ 130 itérations pour atteindre le premier objectif du *Curriculum*. Cette comparaison montre sans doute que le *Buffer* Priority aide à la vitesse de convergence du modèle lors de l'entraînement.

Après l'entraînement des deux modèles, ceux-ci sont testés sur 1000 trajectoires avec une altitude aléatoire entre 1.0 mètre et 2.5 mètres. L'objectif est de voir lequel performe le mieux en période de test. Les résultats de ce test montrant que le *Buffer* Priority performe encore mieux est au tableau 1. Ce type de *Buffer* sera donc utilisé par défaut pour les prochains entraînements.

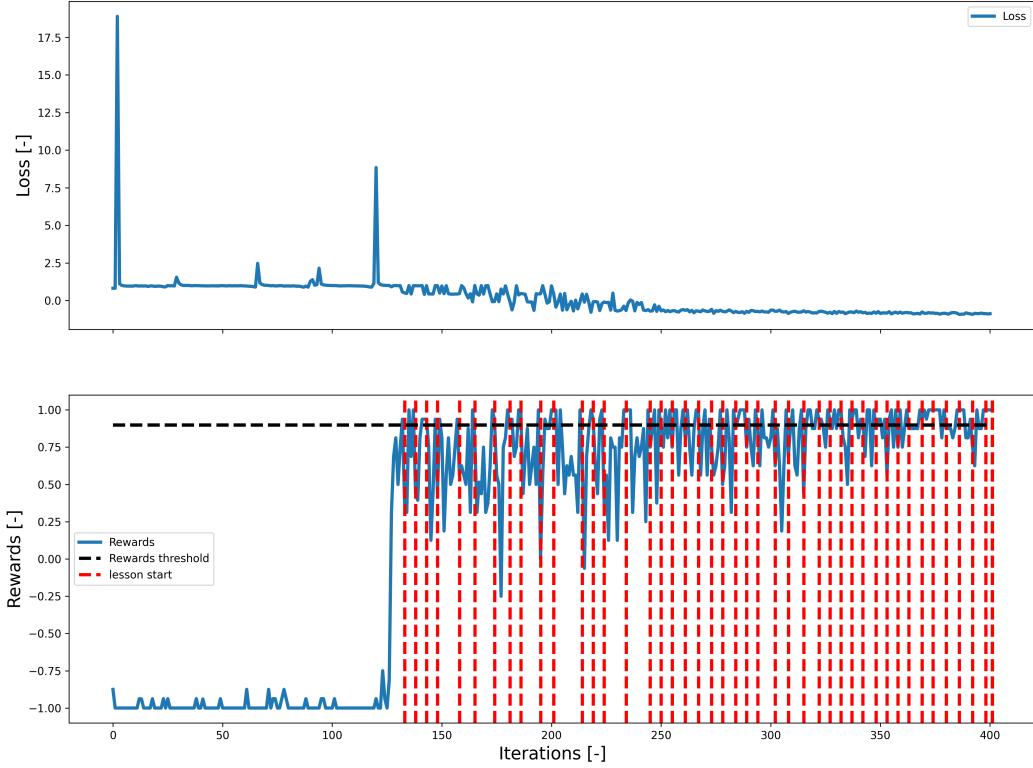


FIGURE 4.2 – Contrôle du MAV avec un MLP de neurones cachés 128x128 avec la position de l'aéronef en entrée. L'environnement à un seul degré de liberté est utilisé avec le *Buffer* retirant l'expérience avec le moins d'avantages en absolue.

<i>Buffer</i>	Mean Rewards	Std Rewards
FIFO	-0.530	0.848
Priority	1.000	0.000

TABLE 1 – Comparaison du type de *Buffer* sur le modèle entraîner sur 400 itérations et testé sur 1000 trajectoires à une hauteur aléatoire entre 1.0 mètre et 2.5 mètres. Le modèle utilisé est le MLP avec la taille de neurones cachés de 128x128 et la position du MAV comme observation. Le *Buffer* FIFO est celui qui retire le plus vieil élément et le *Buffer* Priority est celui qui retire l'expérience avec le moins d'avantages en absolue.

4.2 Un degrés de liberté

Dans cette section, les différents modèles et dynamiques sont comparés avec les différents capteurs pour l'environnement à un seul degré de liberté. Le *Curriculum* énoncé à la section 3.4 est encore utilisé. Les différents modèles sont entraînés pendant un maximum de 1000 itérations et un maximum de 1 heure. Dans un premier temps, le MLP est testé dans l'environnement à un degré de liberté avec l'état du robot en entrée. Le but de cette expérience est de voir si la tâche est faisable dans les meilleures circonstances possibles, c'est-à-dire utilisant toutes l'information requise pour accomplir la tâche. Les résultats au test de cette expérience sont affichés au tableau 2. Ce dernier montre que la tâche est effectivement possible à accomplir. En effet, le MLP réussit à parfaitement atterrir dans une telle configuration avec des moyennes de récompense de 1.000.

Modèle	Inputs	Mean Rewards	Std Rewards
MLP-128x128	P	1.000	0.000
MLP-128x128	P+V	1.000	0.000

TABLE 2 – Comparaison des résultats obtenus par le MLP avec l'état direct du MAV sur l'environnement à un degré de liberté. Les résultats de ce tableau révèlent les modèles testés sur 1000 trajectoires à une hauteur aléatoire entre 1.0 mètre et 2.5 mètres. L'objectif ici est d'utiliser ces résultats à titre de comparatif et pour montrer que la tâche est réalisable.

Les acronymes utilisés au tableau de résultats 2 sont les suivants :

- P : la position $[x \ y \ z]$ du MAV ;
- V : la vitesse linéaire du MAV dans les 3 directions cartésiennes.

Dans un deuxième temps, les différents modèles sont comparés avec les différents capteurs disponibles pour la tâche donnée. Les résultats de cette expérience sont fournis au tableau 3. Dans ce dernier, il est possible de voir que la tâche est maintenant bien plus difficile à accomplir. En effet, les moyennes de récompenses des différents modèles ne sont plus de 1.000, mais bien inférieur. Dans son cas, le MLP réussit tout de même super bien à remplir ses objectifs avec des couches cachées de 128x128 neurones et le vecteur de flux optique en entrée. De même avec la caméra à évènement le MLP réussit le test avec 1.000 de récompenses en moyenne. Toutefois, la combinaison des capteurs font complètement diverger le modèle et ne lui permet pas d'apprendre à accomplir la tâche dans le temps donné.

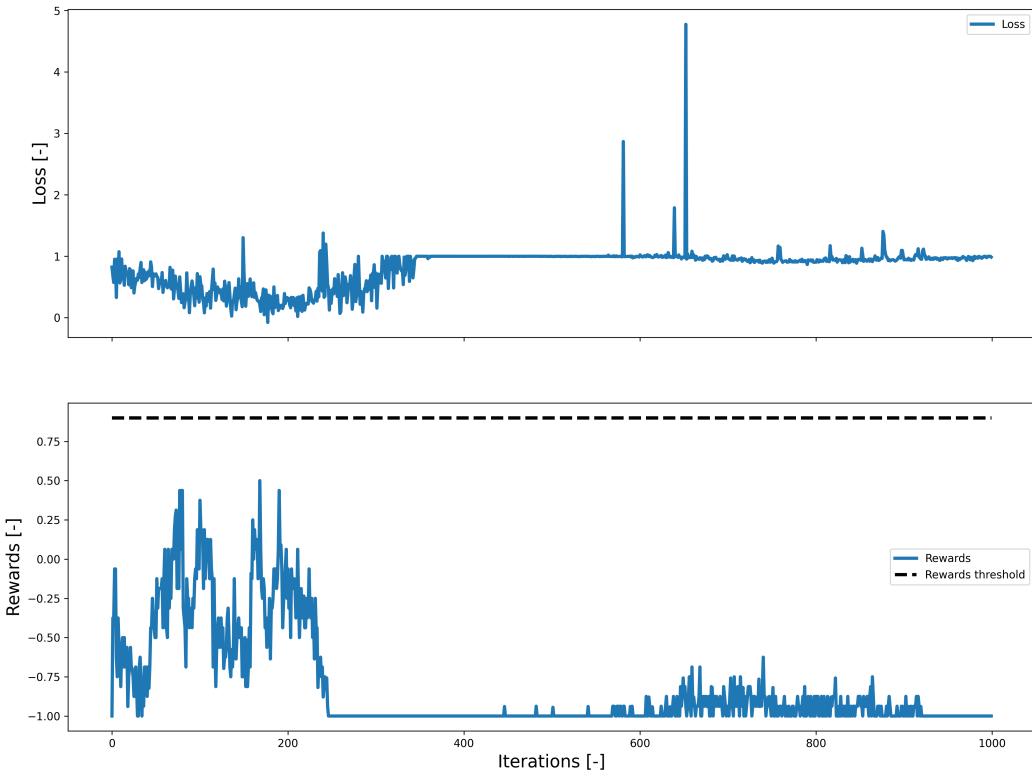


FIGURE 4.3 – Contrôle du MAV avec un ALIF de neurones cachés 64x8 avec la caméra à évènement comme capteur en entrée. L'environnement à un seul degré de liberté est utilisé sur 1000 itérations d'entraînement.

Pour ce qui est de LIF, c'est le modèle qui réussit le mieux des SNN avec la capteur de flux optique avec une récompense moyenne de -0.080. Par contre, LIF est moins performant de ALIF du moment que la caméra à évènement est utilisée.

Ceci est sûrement dû au fait que LIf a tendance à avoir un taux de décharge très élevé dès que trop d'entrées sont actives. Ceci arrive souvent avec la caméra à évènements. Ce défaut de LIF vient du fait que le seuil de potentiel d'activation est constant, ce qui fait augmenter drastiquement l'activité des neurones si trop d'impulsions entrent dans le réseau. La conséquence d'une suractivité du réseau est une perte d'information de la sortie de celui-ci, ce qui empêche de bien faire la prédiction d'action. La dynamique ALIF pallie ce problème avec son seuil adaptatif qui fait diminuer l'activité du réseau si ce dernier devient trop actif.

De son côté, le modèle ALIF réussit à faire atterrir le MAV à quelques reprises avec les différents capteurs, mais ne performe pas aussi bien que le MLP. En fait, il performe de façon assez médiocre avec des moyennes de récompenses de -0.300, -0.140 et de -0.960 en test. Par contre, ces résultats ne représentent pas complètement la performance de ALIF. En effet, en observant la courbe d'apprentissage de ce modèle à la figure 4.3, il est possible de voir que le modèle a commencé à apprendre en début d'entraînement, mais s'est mis à diverger à environ 250 itérations. En début d'entraînement, ALIF a presque réussi la première leçon du *Curriculum* avant de diverger. Ce genre de phénomène arrive généralement pour les faibles résultats présentés dans le prochain tableau. Le résultat affiché au tableau 3 pour cette configuration est fait avec le modèle ayant la meilleure performance durant l'entraînement. Donc, le modèle ALIF entraîné avec une altitude maximale de 1 mètre a tout de même réussi à obtenir une performance plus grande que -1.000 sur le test avec une hauteur maximale de 2.5 mètres. Il est logique de croire que le modèle ait réussi que les trajectoires ayant commencé à environ 1 mètre d'altitude. Toutefois, le modèle ALIF obtient un score moyen de 0.300 sur 100 trajectoires de test à 2.5 mètres d'altitude. Ce résultat sera discuté plus en détail dans la section 5, mais il s'agit d'un bon indice d'une erreur de *Curriculum* qui nuit à l'apprentissage des modèles.

Modèle	Inputs	Mean Rewards	Std Rewards
MLP-128x128	OF	0.000	0.000
MLP-128x128	OFH	1.000	0.000
MLP-128x128	EventCam8x8	1.000	0.000
MLP-128x128	OFH+EventCam8x8	-1.000	0.000
LIF-10x5	OFH	-0.080	0.997
LIF-64x8	EventCam8x8	-0.980	0.199
LIF-64x8	OFH+EventCam8x8	-0.940	0.341
ALIF-10x5	OFH	-0.300	0.954
ALIF-64x8	EventCam8x8	-0.140	0.990
ALIF-64x8	OFH+EventCam8x8	-0.960	0.280

TABLE 3 – Comparaison des différents modèles sur l'environnement à un degré de liberté. Les résultats de ce tableau révèlent les modèles testé sur 100 trajectoires à une hauteur aléatoire entre 1.0 mètre et 2.5 mètres .

Les acronymes utilisés au tableau de résultats 3 sont les suivants :

- OF : Caméra à flux optique représenté sous forme de valeur réelle ;
- OFH : Caméra à flux optique représenté sous forme de *one hot vector* ;
- EventCam8x8 : Caméra à évènement de taille 8 pixels par 8 pixels.

4.3 Six degrés de liberté

Considérant que les résultats à un seul degré de liberté sont très bas et ne démontrent pas une maîtrise de l'atterrissement pour cette configuration, il est très difficile de croire que le contrôle du MAV fonctionnera à six degrés de liberté. En effet, c'est qu'il est possible de constater avec le tableau 4. Dans ce dernier, aucun algorithme ne semble converger durant sa période d'entraînement limité à 1000 itérations et à 1 heure. En fait, le seul modèle qui a été en mesure d'apprendre un tout petit peu est le MLP avec la caméra à flux optique et la caméra à évènements obtenant -0.960 comme moyenne de récompenses en test. Plusieurs avenues seront discutées dans la section 5 afin d'améliorer suffisamment les modèles et l'entraînement pour être en mesure de refaire ces mêmes tests et obtenir des résultats satisfaisants. Pour l'instant, la seule conclusion à retenir de ces résultats à six degrés de liberté est que le pipeline d'apprentissage n'est pas encore prêt pour un tel niveau de difficulté.

Modèle	Inputs	Mean Rewards	Std Rewards
MLP-64x16	P+A+V+AV	-1.000	0.000
MLP-128x128	EventCam8x8	-1.000	0.000
MLP-128x128	OFH+EventCam8x8	-0.960	0.280
LIF-64x8	EventCam8x8	-1.000	0.000
LIF-64x8	OFH+EventCam8x8	-1.000	0.000
ALIF-64x8	EventCam8x8	-1.000	0.000
ALIF-64x8	OFH+EventCam8x8	-1.000	0.000

TABLE 4 – Comparaison des différents modèles sur l'environnement à six degrés de liberté. Les résultats de ce tableau révèlent les modèles testés sur 100 trajectoires à une hauteur aléatoire entre 1.0 mètre et 2.5 mètres.

Les acronymes utilisés au tableau de résultats 4 sont les suivants :

- A : les angle d'Euler $[\theta_x \quad \theta_y \quad \theta_z]$ du MAV ;
- AV : la vitesse angulaire autour des trois directions cartésiennes.

5 Discussion

Le tableau des résultats 3 présenté à la section 4.2 montre à quel point le présent problème est difficile, et cela même en le simplifiant à un seul degré de liberté. Il faut bien sûr comprendre que le présent projet a été fait en un temps assez restreint ce qui a grandement limité l'exploration des paramètres des modèles comparés. Toutefois, il est possible d'en tirer certaines conclusions.

Dans un premier temps, il est possible de voir que le MLP fonctionne très bien à un degré de liberté en ayant des moyennes de récompenses de 1.000 en test avec les différents types de capteurs. Ces résultats montrent qu'il est possible pour l'agent d'accomplir la tâche demandée avec les informations fournies. D'un autre côté, les dynamiques à impulsions ont beaucoup plus de difficulté. En effet, en général le modèle ALIF performe davantage que la dynamique LIF dans ce type de problème, mais termine tout de même avec de basses performances. La dynamique LIF fonctionne seulement à un degré de liberté avec le flux optique en entrée. Toutefois, le fait que les SNN ont réussi à apprendre partiellement une tâche montre qu'il est possible d'obtenir des résultats avec ce type de réseau.

De plus, les réseaux utilisés ne prennent pas en compte la corrélation entre les pixels de la caméra à événements. Il serait très bénéfique d'effectuer des convolutions en entrée de réseau pour utiliser ces corrélations. Les SNN deviendraient des ConvSNN et seraient certainement plus aptes à performer dans ce genre de tâche.

Pour ce qui est de l'entraînement lui-même, quelques problèmes se sont mis en lumière lors de l'analyse des résultats. Entre autre, le fait que l'entraînement des réseaux à une grande tendance à diverger. Quand une divergence survient, il est très rare que le réseau soit en mesure de réapprendre la tâche et de se remettre à performer. Le problème peut venir de la fonction de perte utiliser pour l'optimisation des paramètres. En effet, la fonction de perte dépend de l'avantage (équation 18) et celle-ci pourrait représenter un problème. Dans Schulman et al. [5] la fonction d'avantage utilisé est la suivante :

$$\hat{A}_t = \sum_{i=0}^{T-1} (\gamma \lambda)^i (r_{t+i} + \gamma V_\phi(s_{t+i+1}) - V_\phi(s_{t+i})) \quad (19)$$

avec V_ϕ étant un estimateur tentant d'estimer la récompense de fin de trajectoire optimiser par descente de gradient. Dans leur cas, T ne correspond pas au temps de trajectoire, mais bien au temps d'horizon qui est beaucoup plus petit. Dans le présent projet, la fonction d'avantage a été simplifiée au maximum afin de ne pas avoir recourt aux estimateurs V_ϕ qui auraient dû être comparés selon plusieurs types comme un MLP, LIF et ALIF. Toutefois, cette simplification a sûrement impacté négativement les performances des modèles en abaissant la généralité de ces derniers.

La divergence du modèle durant l'entraînement pourrait aussi être minimisée d'une autre façon. Durant l'entraînement du modèle, les récompenses moyennes ainsi que la perte moyenne de chaque itération sont gardées en mémoire et il serait certainement possible de les utiliser pour empêcher la divergence du réseau au long terme. L'idée est de regarder à quel moment la perte et la récompense moyenne se mettent à diverger pour revenir en arrière et reprendre les paramètres du modèle qui fonctionnait relativement bien avant la divergence. Le tout se ferait durant l'entraînement même. Il faudrait sans aucun doute laisser le modèle explorer suffisamment, alors une simple diminution de la performance ne

devrait pas générer un recouvrement du meilleur modèle. Par contre, au moment que le modèle se met à diverger pendant quelques itérations et perd énormément en performance, cela pourrait générer un recouvrement des meilleurs paramètres du modèle. Cette idée n'est pas lancée à l'aveugle. En fait, elle a été utiliser à la mitaine durant l'entraînement de certains réseaux ce qui a permis à ceux-ci d'obtenir de meilleures performances. Évidemment, ça ne règle pas tous les problèmes, donc certains modèles divergeaient tout de même. D'un autre côté, il est évident qu'une méthode plus quantitative sera plus performante que la méthode qualitative utilisée.

Une autre méthode permettant d'obtenir une convergence plus rapide en entraînement serait d'utiliser des heuristiques d'enseignement. En réalité l'objet *Curriculum* est plus complexe que ce qui a été montré dans cette étude. Ce dernier est en mesure de contenir un réseau enseignant permettant d'aider l'agent à apprendre durant l'entraînement. Si un enseignant est ajouté à une leçon, l'agent sera optimisé sur la prédiction des actions exécutée par l'enseignant sur des trajectoires faites par ce dernier. De cette façon, l'agent est optimisé à prédire les mêmes actions que l'heuristique utilisée pour une leçon donnée. Cette étape d'apprentissage se fait juste avant la maximisation de l'avantage à chaque itération d'entraînement. Cette fonctionnalité a été implémentée, mais due à des problèmes numériques des heuristiques, ils n'ont pas été ajoutés en tant qu'enseignants pour les leçons qui ont généré les résultats de ce rapport. Cette technique aussi appelée "clonage de comportement" en apprentissage par renforcement. Schulman et al. [5] introduisent comment incorporer une telle fonctionnalité avec PPO. Il est donc certain qu'en corrigeant les problèmes reliés aux heuristiques ou en utilisant les MLP comme heuristiques pour l'apprentissage des SNN, les réseaux risquent fort bien de converger plus rapidement.

Comme présenté rapidement dans la section des résultats sur les expériences à 1 degré de liberté, le *Curriculum* a certainement introduit une source de difficulté supplémentaire. En effet, ce dernier fait commencer l'entraînement à 1 mètre du sol et fait augmenter la hauteur maximale jusqu'à 10 mètres. L'objectif étant toujours d'augmenter la difficulté tout au long de l'entraînement. Par contre, il semblerait qu'il est plus difficile de faire atterrir le MAV à 1 mètre du sol qu'à 2 mètres. Cela est surtout dû au temps de réaction beaucoup plus petit à 1 mètre du sol. Le *Curriculum* devrait dans ce cas commencer l'entraînement à 2 mètres du sol, ce qui pourrait grandement aider à l'apprentissage des modèles.

6 Conclusion

Le présent projet est seulement à sa première itération et est loin d'être terminé, mais les bases y sont placées pour permettre de trouver un bon réseau à impulsion capable d'effectuer le contrôle d'un MAV. Ce projet a demandé une énorme tâche d'implémentation en passant par la mise au point de capteurs, de dynamiques neuronales, de pipeline d'apprentissage par renforcement, d'outils de gestion de mémoire à la communication entre les langages C# et Python. En plus de modéliser l'environnement avec le moteur de jeu Unity. En somme, le projet comporte beaucoup de compartiments différents demandant des champs d'expertises diverses.

Au niveau des résultats, les réseaux à impulsions ainsi que la méthode d'entraînement ne sont pas encore au point et provoquent plus d'écrasement que d'atterrissements en douceur pour le moment. Toutefois, il a été montré que le contrôle du MAV par des techniques provenant de la neuroscience et de l'apprentissage profond est tout à fait possible. En effet, les résultats non négligeables de récompenses pour le contrôle à un degré de liberté appuient ces dires. Du côté des perceptrons multicouches, ils ont été capables de très bien résoudre le problème à un degré de liberté démontrant que cette tâche est bel et bien réalisable dans l'environnement implémenté. Il manque maintenant à réussir à diminuer l'écart de performance entre les MLP et les réseaux à impulsions.

Pour terminer, beaucoup de fonctionnalités pour ce projet ont été implémentées et montrent qu'il sera possible d'atteindre les objectifs avec du travail supplémentaire. Un grand nombre de voies d'améliorations ont été apportées dans la discussion et celles-ci permettront certainement d'augmenter de façon significative les performances des divers modèles comparés dans cette étude.

Références

- [1] M. DAVIES, N. SRINIVASA, T.-H. LIN et al., “Loihi : A Neuromorphic Manycore Processor with On-Chip Learning,” *IEEE Micro*, t. 38, n° 1, p. 82-99, jan. 2018, Conference Name : IEEE Micro, ISSN : 1937-4143. DOI : 10.1109/MM.2018.112130359.
- [2] J. DUPEYROUX, J. J. HAGENAARS, F. PAREDES-VALLÉS et G. C. H. E. de CROON, “Neuromorphic control for optic-flow-based landing of MAVs using the Loihi processor,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, ISSN : 2577-087X, mai 2021, p. 96-102. DOI : 10.1109/ICRA48506.2021.9560937.
- [3] U. TECHNOLOGIES. “Unity real-time development platform | 3d, 2d VR & AR engine.” (), adresse : <https://unity.com/> (visité le 09/03/2022).
- [4] A. JULIANI, V.-P. BERGES, E. TENG et al., “Unity : A General Platform for Intelligent Agents,” *arXiv :1809.02627 [cs, stat]*, 6 mai 2020. arXiv : 1809.02627. adresse : <http://arxiv.org/abs/1809.02627> (visité le 09/03/2022).
- [5] J. SCHULMAN, F. WOLSKI, P. DHARIWAL, A. RADFORD et O. KLIMOV, “Proximal Policy Optimization Algorithms,” *CoRR*, t. abs/1707.06347, 2017. arXiv : 1707.06347. adresse : <http://arxiv.org/abs/1707.06347>.
- [6] Python.org. adresse : <https://www.python.org/>.
- [7] PyTorch. adresse : <https://pytorch.org/>.
- [8] BILLWAGNER. “C# docs - get started, tutorials, reference.” (), adresse : <https://docs.microsoft.com/en-us/dotnet/csharp/> (visité le 03/05/2022).
- [9] J. GINCE, “MAV Control With SNN,” 2022. adresse : <https://github.com/JeremieGince/MAVControlWithSNN>.
- [10] J. SCHULMAN, S. LEVINE, P. MORITZ, M. I. JORDAN et P. ABBEEL, “Trust Region Policy Optimization,” *CoRR*, t. abs/1502.05477, 2015. arXiv : 1502.05477. adresse : <http://arxiv.org/abs/1502.05477>.
- [11] J. SCHULMAN, P. MORITZ, S. LEVINE, M. JORDAN et P. ABBEEL, “High-Dimensional Continuous Control Using Generalized Advantage Estimation,” *arXiv :1506.02438 [cs]*, 20 oct. 2018. arXiv : 1506.02438. adresse : <http://arxiv.org/abs/1506.02438> (visité le 29/04/2022).
- [12] G. BROCKMAN, V. CHEUNG, L. PETTERSSON et al., *OpenAI Gym*, 2016. eprint : arXiv:1606.01540.
- [13] “Roboschool,” OpenAI. (15 mai 2017), adresse : <https://openai.com/blog/roboschool/> (visité le 29/04/2022).
- [14] J. GINCE et R. LAMONTAGNE-CARON, “SNN Image Classification,” 2022. adresse : <https://github.com/JeremieGince/SNNImageClassification>.
- [15] E. O. NEFTCI, H. MOSTAFA et F. ZENKE, “Surrogate Gradient Learning in Spiking Neural Networks : Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks,” *IEEE Signal Processing Magazine*, t. 36, n° 6, p. 51-63, nov. 2019, Conference Name : IEEE Signal Processing Magazine, ISSN : 1558-0792. DOI : 10.1109/MSP.2019.2931595.
- [16] M. TRAJKOVIĆ et M. HEDLEY, “Fast corner detection,” *Image and Vision Computing*, t. 16, n° 2, p. 75-87, 20 fév. 1998, ISSN : 0262-8856. DOI : 10.1016/S0262-8856(97)00056-5. adresse : <https://www.sciencedirect.com/science/article/pii/S0262885697000565> (visité le 01/05/2022).
- [17] J.-Y. BOUGUET et al., “Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm,” *Intel corporation*, t. 5, n° 1-10, p. 4, 2001.
- [18] G. BELLEC, F. SCHERR, A. SUBRAMONEY et al., “A solution to the learning dilemma for recurrent networks of spiking neurons,” *Nature Communications*, t. 11, n° 1, p. 3625, 17 juill. 2020, ISSN : 2041-1723. DOI : 10.1038/s41467-020-17236-y. adresse : <https://www.nature.com/articles/s41467-020-17236-y> (visité le 18/12/2021).
- [19] E. M. IZHKEVICH, *Dynamical Systems in Neuroscience*. MIT Press, 2007, 522 p., ISBN : 978-0-262-09043-8.