Guidage de drone pour la triangulation à N-vues basé sur l'apprentissage par renforcement multi-agents

Timothée Gavin*†‡, Murat Bronz† and Simon Lacroix‡
*IAS, Thales LAS, Rungis, France
†Dynamic Systems, OPTIM, Fédération ENAC ISAE-SUPAERO ONERA, Université de Toulouse, Toulouse, France
‡LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
timothee.gavin@thalesgroup.fr , murat.bronz@enac.fr , simon.lacroix@laas.fr

***Abstract*—Cet article présente une nouvelle approche pour le contrôle d'une flotte de drones qui peuvent suivre la position d'une cible volante à l'aide de caméras omnidirectionnelles embarquées. Les drones utilisent l'apprentissage par renforcement multi-agents (MARL) pour apprendre des stratégies décentralisées qui optimisent leur formation et leur mouvement autour de la cible, en minimisant l'incertitude sur la position triangulée. Nous définissons une fonction de récompense qui encourage les suiveurs à minimiser la trace de la matrice de covariance de la position triangulée, dérivée d'un modèle analytique de propagation de l'incertitude. Nous utilisons Multi-Agent PPO (MAPPO), une extension de la méthode Proximal Policy Optimization (PPO) au domaine multi-agents, pour entraîner les modèles à l'aide de cette fonction de récompense commune qui favorise une bonne configuration et permet d'éviter les collisions. Nous validons notre approche en simulation et en vol réel, démontrant son efficacité et son potentiel dans l'amélioration de la coordination multi-drones autonomes pour un suivi de cible précis.**

***Index Terms*—Apprentissage par renforcement, multi-agents, PPO, triangulation**

I. Introduction

Dans cet article, nous cherchons à faire les premiers pas vers une nouvelle approche pour le contrôle d'une flotte de drones qui peuvent suivre l'emplacement d'une ou de plusieurs cibles volantes à l'aide de caméras embarquées. L'algorithme de guidage doit optimiser le placement des drones afin de minimiser l'incertitude de la triangulation des cibles suivies, même lorsqu'elles se déplacent avec des trajectoires imprévisibles.

L'approche proposée utilise l'apprentissage par renforcement multi-agent (ou Multi-Agent Reinforcement Learning (MARL) en anglais) pour positionner de manière optimale plusieurs drones afin de trianguler la position d'une cible unique à l'aide de caméras omnidirectionnelles embarquées. Nous utilisons Multi-Agent PPO (MAPPO) [1], une extension de l'algorithme Proximal Policy Optimization (PPO) [2] au domaine multi-agent, pour entraîner des agents à suivre des stratégies décentralisées en utilisant une fonction de récompense commune spécialement définie qui encourage une bonne formation autour de la cible afin de minimiser l'incertitude dans la position triangulée. Les modèles entraînés sont validés en simulation et le comportement résultant est démontré dans des expériences en vol réel.

Nous étudions le MARL par rapport aux méthodes d'optimisation traditionnelles car nous espérons de meilleures performances dans la résolution des problèmes de triangulation impliquant un grand nombre de drones. Les méthodes d'optimisation traditionnelles, même en utilisant des heuristiques, sont lentes et coûteuses en temps de calcul pour les problèmes à grande échelle, ce qui n'est pas adapté à un algorithme de guidage embarqué. Bien que l'entrainement d'un agent RL soit une opération coûteuse en temps de calcul, il peut rapidement générer des solutions en ligne.

L'article est organisé comme suit. Après avoir passé en revue les travaux connexes pertinents sur le suivi des drones utilisant des drones dans la Section II, et présenté le contexte existant en matière de MARL et de triangulation dans la Section III, nous présentons notre approche de guidage de drone basé sur le MARL appliqué à la triangulation à N-vues dans la Section IV. Nous détaillons la configuration des simulations et leurs résultats dans la Section V. Enfin, nous décrivons les expériences en vol réel dans la Section VI.

II. Travaux Associés

La triangulation est le processus d'estimation de la position 3-D d'un point à partir de ses projections sur deux ou plusieurs images prises à partir de points de vue différents. Cette opération peut être réalisée en calculant l'intersection des rayons de projection associés à chaque image, et en minimisant l'erreur de reprojection. Cependant, la triangulation est sensible aux erreurs d'étalonnage de la caméra, ainsi qu'au bruit et aux valeurs aberrantes dans les points de l'image. Une technique courante pour optimiser le placement des caméras et l'estimation de la position de la cible dans la triangulation à N-vues est *l'ajustement de faisceau* [3]. Il s'agit d'une méthode d'optimisation des moindres carrés non linéaire qui affine simultanément les paramètres intrinsèques et extrinsèques de la caméra, en minimisant la somme des carrés des erreurs de reprojection sur tous les points d'image et toutes les vues. Cette méthode est largement utilisée pour la recontruction 3D de scènes à partir d'images calibrées prises

par différentes caméras. Cependant, le coût de calcul et les besoins en mémoire de l'ajustement des faisceaux augmentent de façon superlinéaire avec le nombre de caméras [4].

Le suivi est le processus d'estimation de la position et de l'orientation d'un objet en mouvement dans le temps à partir d'une séquence d'images. Des techniques telles que les filtres de Kalman et la régression polynomiale non linéaire sont couramment utilisées pour le suivi des drones [4]. La majorité des recherches sur les robots mobiles coopératifs pour l'observation de cibles mobiles se concentrent sur des cibles terrestres se déplaçant dans un plan 2-D [5]–[7]. Certaines études ont proposé l'utilisation de caméras embarquées pour le suivi en temps réel et la localisation en 3-D de plusieurs drones [8]. Cependant, le suivi de drones à l'aide d'une caméra embarquée sur un autre drone reste un domaine relativement peu exploré dans la littérature, car il pose plus de difficultés que le suivi à l'aide de systèmes de caméras fixes.

L'apprentissage par renforcement (Reinforcement Learning (RL) en anglais) est un paradigme d'apprentissage automatique qui permet à un agent d'apprendre de ses propres actions en recevant des récompenses et des punitions. Le RL a été appliqué à divers problèmes impliquant le placement optimal de capteurs et à des algorithmes de guidage pour les drones et les flottes de drones. Le RL a été utilisé pour mesurer la profondeur des scènes intérieures à l'aide de plusieurs caméras, l'agent apprenant à sélectionner les meilleures positions et orientations des caméras afin de minimiser l'erreur d'observation de la profondeur [9]. Alternativement, une approche basée sur le RL pour la poursuite et l'évasion de drones, où l'agent apprend à suivre un drone cible en utilisant des données de capteurs et détecteur d'objet, a été présentée dans [10]. Ces travaux se concentrent principalement sur le RL pour un seul agent, où un agent interagit avec l'environnement de manière indépendante. MARL est une branche de RL qui traite de plusieurs agents qui coopèrent ou sont en concurrence les uns avec les autres dans un environnement partagé. Le MARL a été utilisé pour apprendre la configuration optimale et le déplacement d'une flotte de drones pour la triangulation, en maximisant la couverture et la diversification des vues et en optimisant les communications [7], [11], mais encore une fois, ces études se concentrent sur l'observation de cibles terrestres se déplaçant dans un plan à deux dimensions. À notre connaissance, aucune étude récente n'a abordé le placement optimal d'une flotte de drones utilisant le MARL pour suivre des objets volants à l'aide de caméras embarquées.

III. Méthodes et outils

A. Processus de décision markoviens décentralisés partiellement observables (Dec-POMDP)

Nous considérons une tâche multi-agents entièrement coopérative qui peut être décrite comme un processus de décision markovien décentralisé et partiellement observable [12] défini par $(n, S, A, \Omega, T, O, R, \gamma)$, où n le nombre d'agents, S est l'espace d'état, $A = A_1 \times \ldots \times A_N$ et $\Omega = \Omega_1 \times \ldots \times \Omega_N$ sont l'ensemble des actions conjointes et des observations conjointes avec chaque A_i et Ω_i étant les ensembles d'actions locales et d'observations locales de l'agent i, $T : S \times A \times S \to [0, 1]$ est la fonction de probabilité de transition entre états, $O : S \times A \times \Omega \to [0, 1]$ est la fonction de probabilité d'observation sur les états, $R : S \times A \to \mathbb{R}$ est une fonction de récompense sur les transitions d'états, et enfin $\gamma \in [0, 1]$ est le facteur d'actualisation.

À chaque pas de temps t, chaque agent i choisit une action $a_{i,t} \in A_i$ sur la base de son historique d'observations locales $o_{i,1:t}$, et reçoit une observation locale $o_{i,t+1} \in \Omega_i$ basée sur l'état résultant s_{t+1}. L'action conjointe $a_t = (a_{1,t}, \ldots, a_{N,t})$ détermine la récompense immédiate $r_t = R(s_t, a_t)$ et l'état suivant s_{t+1} selon la probabilité de transition $T(s_t, a_t, s_{t+1}) = \Pr(s_{t+1}|s_t, a_t)$. L'objectif est de trouver une politique $\pi_\theta(a_{i,t}|o_{i,1:t})$ paramétrée par θ qui produise une action $a_{i,t}$ à partir de l'historique d'observations locales $o_{i,1:t}$, qui maximise le gain attendu $J(\theta) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t]$. La fonction de valeur d'état $V^{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s]$ est le gain attendu en partant de l'état s et en suivant la politique π_θ par la suite. La fonction de valeur action-état $Q^{\pi_\theta}(s, a) = \mathbb{E}_{\pi_\theta}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s, A_t = a]$ est le gain attendu en partant de l'état s, en prenant l'action a, et en suivant la politique π par la suite.

B. Algorithmes d'apprentissage par renforcement à un seul agent

La plupart des méthodes d'apprentissage par renforcement peuvent être divisées en deux groupes : les méthodes basées sur la valeur et les méthodes basées sur la politique. Les méthodes basées sur la valeur, telles que Deep Q-Learning (DQN) [13], utilisent des réseaux neuronaux profonds pour estimer les fonctions de valeur et dérivent la politique optimale π^* à partir des fonctions de valeur optimisées en choisissant pour chaque état s l'action qui maximise la valeur de l'action : $\pi^*(s) = \arg\max_a Q^*(s, a)$. Les méthodes basées sur la politique, quant à elles, paramètrent directement la politique $\pi_\theta(a|s)$ en fonction de l'état et de l'action. Les méthodes basées sur le gradient de la politique sont une sous-classe des méthodes basées sur la politique qui mettent à jour les paramètres de la politique θ en suivant le gradient du gain attendu : $\nabla_\theta J(\theta) = E_\pi[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)]$. L'utilisation du gain total de chaque épisode pour estimer le gradient entraîne une forte variance dans les estimations du gradient. Pour réduire cette variance, les méthodes Actor-Critic ont été introduites [14]. Ces méthodes maintiennent un approximateur explicite séparé (la Critique) pour estimer la fonction de valeur, qui est utilisée comme base de référence pour calculer la fonction Advantage. La fonction Advantage, $A^\pi(s, a) = Q^\pi(s, a) - V\pi(s)$, mesure l'amélioration d'une action a par rapport à l'action moyenne à l'état s dans le cadre de la politique π. Cela permet de réduire la variance des estimations du gradient, ce qui conduit à un apprentissage plus stable. PPO [2] est une évolution des méthodes de gradient de politique qui cherche à mettre à jour la politique d'une manière qui évite les changements importants et abrupts qui pourraient déstabiliser le processus d'apprentissage, garantissant ainsi un apprentissage stable et efficace.

C. Apprentissage par renforcement multi-agents

Les méthodes RL à un seul agent échouent souvent dans des contextes multi-agents en raison du fléau de la dimension et de la non-stationnarité. Récemment, les approches MARL ont abordé ces problèmes avec l'approche CTDE (Centralized Training, Decentralized Execution). En entraînant les agents de manière centralisée afin d'exploiter les informations globales, puis en exécutant les politiques apprises de manière décentralisée, CTDE garantit robustesse et capacité de montée à l'échelle. Plusieurs algorithmes MARL ont été développés dans le cadre du CTDE, y compris des méthodes basées sur la valeur comme QMIX [15] et VDAC [16], et des méthodes de gradient de politique comme COMA [17] et MADDPG [18]. PPO s'est révélée très prometteuse dans le domaine multi-agent: Multi-Agent PPO (MAPPO) [1], une extension de PPO dans le cadre CTDE, utilise un critique centralisé et des acteurs décentralisés et a démontré des performances supérieures dans diverses tâches complexes multi-agents par rapport à d'autres algorithmes MARL de l'état de l'art [1].

D. Triangulation linéaire à N-vues

La triangulation linéaire est une méthode permettant d'estimer les coordonnées 3-D d'un point à partir de ses projections 2-D sur deux ou plusieurs images prises par des caméras différentes. Le principe de la triangulation linéaire est basé sur le modèle du sténopé, qui relie le point 3-D $\mathbf{X}$ et sa projection 2-D $\mathbf{x}$ en coordonnées homogènes par la matrice 3×4 de projection de la caméra $\mathbf{P}$ sous la forme $\mathbf{x} = \mathbf{PX}$. La matrice de projection de la caméra $\mathbf{P}$ englobe à la fois les paramètres intrinsèques (tels que la distance focale et le centre optique) et les paramètres extrinsèques (rotation et la position) d'une caméra.

La triangulation linéaire résout un système linéaire surdéterminé de la forme $\mathbf{AX} = 0$, qui dans le cas d'une triangulation à N-vues, avec n de ces points d'image et matrices de projection donne :

$$\begin{bmatrix} u_1\mathbf{P}_1^{3T} - \mathbf{P}_1^{1T} \\ v_1\mathbf{P}_1^{3T} - \mathbf{P}_1^{2T} \\ u_2\mathbf{P}_2^{3T} - \mathbf{P}_2^{2T} \\ v_2\mathbf{P}_2^{3T} - \mathbf{P}_2^{2T} \\ \vdots \\ u_n\mathbf{P}_n^{3T} - \mathbf{P}_n^{1T} \\ v_n\mathbf{P}_n^{3T} - \mathbf{P}_n^{2T} \end{bmatrix} \begin{bmatrix} \lambda x \\ \lambda y \\ \lambda z \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{1}$$

où $\mathbf{X} = (\lambda x, \lambda y, \lambda z, \lambda)^T$, λ étant un facteur d'échelle inconnu, et u_i, v_i et P_i^{jT} sont les coordonnées de l'image et la $j$ème ligne de la matrice de projection P_i de la $i$ème caméra. Pour trouver $\mathbf{X}$, on trouve une solution X non nulle qui satisfait $\mathbf{AX} = 0$. Dans le cas de mesures bruitées, comme les n rayons définis par les points de projection 2-D ne se croisent pas en un seul point, le système n'a pas de solutions non nulles, et le problème est transformé en un problème de minimisation, généralement résolue avec des techniques des moindres carrés [19].

IV. Détails de l'approche

A. Présentation du scénario

Notre problème consiste en plusieurs drones, appelés ci-après "suiveurs", qui poursuivent un seul drone, la "cible", pour suivre sa position, en utilisant des capteurs embarqués tels que des caméras. Les suiveurs s'efforcent d'organiser la formation de leur flotte de sorte que l'incertitude de la position triangulée soit minimale.

Les suiveurs et les mouvements de la cible sont omnidirectionnels, nous n'avons pas mis en œuvre de modèle de vol de drone dans cette étude. Nous supposons que la cible à une vitesse de vol constante. Les drones volent dans une arène carrée sans aucun obstacle. Ni les suiveurs ni la cible ne peuvent sortir de l'arène : leurs actions sont limitées pour rester à l'intérieur des frontières de l'arène. Les collisions entre suiveurs, ou entre un suiveur et la cible, entraînent un échec. Le fait de s'écraser au sol entraîne également un échec. L'orientation des drones n'est pas prise en compte dans cette étude, on suppose que tous les drones sont alignés avec leur cap parallèle à la direction positive de l'axe X dans le système de coordonnées global.

Nous supposons que les suiveurs peuvent différencier les autres suiveurs de la cible. Nous n'avons pas pris en compte l'observabilité partielle dans cette étude. Nous faisons l'hypothèse forte que chaque suiveur connaît les positions 3D exactes dans le monde, sans bruit, des autres suiveurs de la flotte et de la cible. Cependant, les actions entreprises par les suiveurs sont décentralisées, ce qui signifie qu'ils ne connaissent pas les actions entreprises par les autres membres de la flotte.

La caméra embarquée sur les suiveurs est supposée être omnidirectionnelle, sans champ de vision restreint et non affectée par les occultations. Le centre de la caméra coïncide avec l'emplacement 3D des suiveurs. Nous supposons également que les paramètres intrinsèques de la caméra n'introduisent aucune distorsion dans les mesures. Plutôt que de mesurer l'emplacement de la projection d'un point du monde sur un plan de caméra 2D, la caméra omnidirectionnelle mesure simplement les angles polaires et azimutaux dans un repère sphérique centré sur la position du suiveur. Au vu de ces simplifications, l'incertitude de la triangulation n'est que le résultat de l'incertitude de la mesure le long des angles polaire

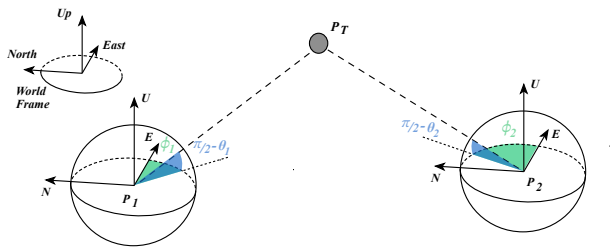

Fig. 1. Représentation des caméras omnidirectionnelles dans un repère sphérique. Les angles polaires θ sont représentés par rapport au plan horizontal pour faciliter la visualisation.

et azimutal dans chaque repère de suiveur. C'est ce qui a conduit à l'élaboration de la fonction de récompense décrite à la Section IV-D.

B. Algorithme d'apprentissage par renforcement multi-agents

Dans cette étude, nous utilisons l'algorithme Multi-Agent Proximal Policy Gradient (MAPPO) en raison de sa simplicité d'implémentation, de sa capacité à opérer dans des espaces d'état et d'action continus et de son efficacité démontrée dans diverses tâches MARL [1]. Nous supposons que tous les agents sont homogènes, ce qui nous permets de partager les modèles entrainés, accélérant ainsi le processus d'apprentissage et maximisant les informations extraites de chaque interaction avec l'environnement. MAPPO fonctionne dans un cadre CTDE, il utilise un critique centralisé et des acteurs décentralisés : alors que tous les agents sont régis par une politique commune et entraînés de manière centralisée, ils agissent de manière indépendante sur la base de leurs observations locales à chaque pas de temps, créant ainsi un système décentralisé. Cette approche est nécessaire car nous n'avons pas abordé le problème de la longueur variable de la représentation de l'état en fonction du nombre de suiveurs dans l'environnement.

C. Représentation de l'état

L'espace d'état, l'espace d'observation et l'espace d'action sont continus. Chaque position 3-D du suiveur en coordonnées cartésiennes globales est encodée dans un vecteur 3-D $\mathbf{p_i}$. La position 3-D de la cible est $\mathbf{p_T}$. Les n suiveurs sont ordonnés de telle sorte que $\mathbf{p_i}$ est toujours la position du $i$ème suiveur. L'état de l'environnement est le nuplet ordonné $\mathbf{s} = (\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}, \mathbf{p_T})$ et constitue l'entrée du réseau de neurone critique centralisé utilisé dans l'algorithme MAPPO. Pour chaque suiveur i, son observation de l'environnement est codée dans le nuplet ordonné $\mathbf{o_i} = (\mathbf{p_i}, \mathbf{p_1}, \ldots, \mathbf{p_{i-1}}, \mathbf{p_{i+1}}, \ldots, \mathbf{p_n}, \mathbf{p_T})$ et constitue l'entrée du réseau de neurone acteur décentralisé.

En ce qui concerne l'espace d'action, comme expliqué dans la section IV-A, le mouvement des suiveurs est holonome. À chaque étape, les agents naviguent dans une sphère d'un rayon de 20 cm centrée sur leur emplacement précédent (cette valeur est directement héritée de l'environnement de simulation CrazyRL [20] que nous avons adapté pour mettre en œuvre notre environnement de simulation, voir la section V). Ce rayon est déterminé par la vitesse maximale de l'agent (environ 2m/s) divisée par la fréquence de contrôle (10Hz dans l'environnement de simulation). Pour calculer la position du prochain pas, le réseau acteur produit un point 3-D dans $[-1, 1]^3$, qui est ensuite mis à l'échelle de 20 cm.

D. Structure de la fonction de récompense

À chaque étape, chaque agent reçoit une récompense commune destinée à encourager les suiveurs à atteindre des positions autour de la cible où l'incertitude sur le résultat de l'algorithme de triangulation est minimale.

L'évaluation de l'incertitude du résultat d'un algorithme d'estimation peut se faire selon deux approches. Une approche *a posteriori* implique la mise en œuvre d'une méthode Monte Carlo : l'incertitude de la sortie est statistiquement obtenue à partir d'un échantillonnage aléatoire répété compte tenu d'une distribution d'entrée bruitée. Bien qu'elle soit facile à mettre en œuvre, elle nécessite un nombre important d'exécutions, ce qui peut prendre beaucoup de temps. La seconde méthode est une approche *a priori*, qui implique le calcul de relations analytiques à l'aide d'approximations linéaires pour décrire la façon dont l'incertitude se propage des entrées aux sorties. Cette deuxième approche ne nécessite qu'une seule exécution pour calculer l'incertitude de la sortie. C'est donc l'approche privilégiée pour notre fonction de récompense, puisqu'elle sera calculée à chaque étape d'interaction avec l'environnement.

1) Triangulation avec des caméras omnidirectionnelles: Dans notre cas simplifié avec des caméras omnidirectionnelles, au lieu de mesurer la position de la projection du point sur le plan de la caméra 2-D, nous mesurons l'angle polaire et l'angle azimutal dans le repère sphérique centré sur la position du suiveur. Pour chaque suiveur à l'emplacement $\mathbf{p} = (p_x, p_x, p_z)$ en coordonnées cartésiennes globales, les coordonnées cartésiennes globales d'un point $\mathbf{X} = (x, y, z)$ sont liées à ses coordonnées sphériques locales de la manière suivante :

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = r \times \begin{bmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{bmatrix} + \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{2}$$

avec $\theta \in [0, \pi]$ l'angle polaire, et $\phi \in [0, 2\pi[$ l'angle azimutal. La distance radiale $r \in \mathbb{R}^+$, est inconnue dans le problème de triangulation. Définissons le vecteur suivant :

$$\mathbf{d_i} = \begin{bmatrix} d_{x,i} \\ d_{y,i} \\ d_{z,i} \end{bmatrix} = \begin{bmatrix} \sin\theta_i\cos\phi_i \\ \sin\theta_i\sin\phi_i \\ \cos\theta_i \end{bmatrix} \tag{3}$$

Ce vecteur, souvent appelé *cosinus directeurs*, définit la direction du point $\mathbf{X}$ dans le système de coordonnées cartésiennes locales du $i$ème suiveur à l'aide des coordonnées sphériques. Avec l'emplacement du suiveur $\mathbf{p_i}$, elles définissent une droite passant par $\mathbf{X}$. Dans le cas d'une triangulation avec deux caméras omnidirectionnelles, nous avons :

$$\begin{cases} \mathbf{X} = r_1 \times \mathbf{d_1} + \mathbf{p_1} \\ \mathbf{X} = r_2 \times \mathbf{d_2} + \mathbf{p_2} \end{cases} \tag{4}$$

Nous pouvons éliminer les distances radiales inconnues à l'aide d'un produit en croix afin d'obtenir trois équations pour chaque suiveur, très semblables à celles de la triangulation linéaire dans le cas des caméras à sténopé dans (1). Notez que la troisième équation est une composition linéaire des deux autres et qu'elle pourrait être supprimée.

$$\begin{cases} d_{y,i}(z - p_{z,i}) - d_{z,i}(y - p_{y,i}) = 0 \\ d_{z,i}(x - p_{x,i}) - d_{x,i}(z - p_{z,i}) = 0 \\ d_{x,i}(y - p_{y,i}) - d_{y,i}(x - p_{x,i}) = 0 \end{cases} \tag{5}$$

Nous obtenons, pour n suiveurs, un système linéaire surdéterminé de la forme $\mathbf{AX} = \mathbf{b}$, qui peut être résolu en

utilisant une méthode telle que la décomposition en valeurs singulières ou en utilisant des techniques de moindres carrés pour obtenir la valeur de $\mathbf{X}$.

$$\begin{bmatrix} \mathbf{A_1} & 0 & \dots & 0 \\ 0 & \mathbf{A_2} & \dots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \dots & \mathbf{A_n} \end{bmatrix} \begin{bmatrix} \mathbf{X}-\mathbf{p_1} \\ \mathbf{X}-\mathbf{p_2} \\ \vdots \\ \mathbf{X}-\mathbf{p_n} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{6}$$

with

$$\mathbf{A_i} = \begin{bmatrix} 0 & -d_{z,i} & d_{y,i} \\ d_{z,i} & 0 & -d_{x,i} \\ -d_{y,i} & d_{x,i} & 0 \end{bmatrix} \tag{7}$$

2) Propagation des incertitudes: Les équations de triangulation dans (6) ont la forme implicite :

$$f_\rho(\mathbf{X}, \mathbf{v}) = 0 \tag{8}$$

où f_ρ est paramétré par le vecteur de paramètres $\rho = (\mathbf{p_1}, \dots, \mathbf{p_n})$, $\mathbf{X}$ est le point 3-D triangulé en sortie de l'algorithme de triangulation et $\mathbf{v}$ est un vecteur d'entrées bruitées $\mathbf{v} = (\theta_1, \phi_1, \dots, \theta_n, \phi_n)$, régi par un bruit gaussien avec une matrice de covariance $\mathbf{\Sigma_v}$.

Dans ce cas, la matrice de covariance $\mathbf{\Sigma_X}$ de la sortie $\mathbf{X}$ est liée à la matrice de covariance $\mathbf{\Sigma_v}$ par [21], [22] :

$$\mathbf{J_X \Sigma_X J_X}^T = \mathbf{J_v \Sigma_v J_v}^T \tag{9}$$

où $\mathbf{J_X}$ et $\mathbf{J_v}$ sont les matrices jacobiennes des dérivées partielles de f_ρ par rapport, respectivement, à la sortie $\mathbf{X}$ et à l'entrée $\mathbf{v}$. La matrice de covariance de sortie peut alors être exprimée comme une fonction de la matrice de covariance d'entrée (qui est soit connue, soit présumée) par :

$$\mathbf{\Sigma_X} = \mathbf{J_X}^+ \mathbf{J_v} (\mathbf{\Sigma_v J_v}^T)(\mathbf{J_X}^+)^T \tag{10}$$

où $\mathbf{J_X}^+$ est la matrice pseudo-inverse de $\mathbf{J_X}$.

Dans notre cas, les jacobiens peuvent être exprimés analytiquement à l'aide de (6). Le vecteur d'entrée $\mathbf{v}$ peut être décomposé en $\mathbf{v} = (\mathbf{v_1}, \mathbf{v_2}, \dots, \mathbf{v_n})$ avec $\mathbf{v_i} = (\theta_i, \phi_i)$, alors $f_\rho(\mathbf{X}, \mathbf{v})$ peut être écrit :

$$f_\rho(\mathbf{X}, \mathbf{v}) = \begin{bmatrix} f_{\mathbf{p_1}}(\mathbf{X}, \mathbf{v_1}) & \dots & f_{\mathbf{p_n}}(\mathbf{X}, \mathbf{v_n}) \end{bmatrix}^T \tag{11}$$

avec chaque $f_{\mathbf{p_i}}(\mathbf{X}, \mathbf{v_i}) = [\mathbf{A_i} \cdot (\mathbf{X} - \mathbf{p_i})]^T$. D'où les matrices jacobiennes suivantes :

$$\mathbf{J_X} = \begin{bmatrix} \frac{\partial f_{\mathbf{p_1}}}{\partial \mathbf{X}} \\ \frac{\partial f_{\mathbf{p_2}}}{\partial \mathbf{X}} \\ \vdots \\ \frac{\partial f_{\mathbf{p_n}}}{\partial \mathbf{X}} \end{bmatrix} = \begin{bmatrix} \mathbf{A_1} \\ \mathbf{A_2} \\ \vdots \\ \mathbf{A_n} \end{bmatrix} \tag{12}$$

$$\mathbf{J_v} = \begin{bmatrix} \frac{\partial f_{\mathbf{p_1}}}{\partial \mathbf{v_1}} & 0 & \dots & 0 \\ 0 & \frac{\partial f_{\mathbf{p_2}}}{\partial \mathbf{v_2}} & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & \frac{\partial f_{\mathbf{p_n}}}{\partial \mathbf{v_n}} \end{bmatrix} \tag{13}$$

avec

$$\frac{\partial f_{\mathbf{p_i}}}{\partial \mathbf{v_i}} = \begin{bmatrix} \frac{\partial \mathbf{A_i}}{\partial \theta_i} \cdot (\mathbf{X} - \mathbf{p_i}) & \frac{\partial \mathbf{A_i}}{\partial \phi_i} \cdot (\mathbf{X} - \mathbf{p_i}) \end{bmatrix} \tag{14}$$

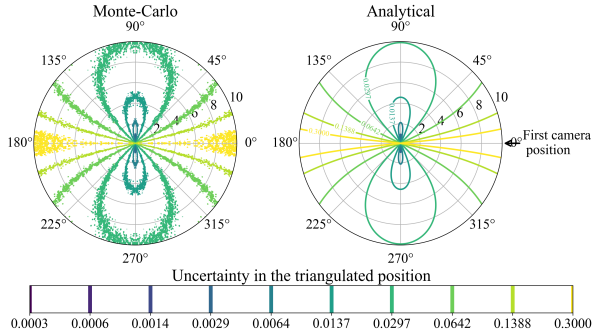

Fig. 2. Incertitude de la sortie triangulée sur l'axe X pour deux caméras dans le plan XY, l'une d'entre elle étant en $(x, y) = (10, 0)$.

3) Comparaison avec l'approche de Monte Carlo: Pour valider le modèle analytique précédent, nous avons comparé l'incertitude obtenue avec l'approche analytique aux résultats d'une approche Monte Carlo. Nous avons placé, en simulation, deux caméras à différents endroits d'une sphère autour d'un point cible. En supposant que les entrées sont soumises à un bruit gaussien, nous avons ajouté un bruit gaussien avec un écart type $(\sigma_{\theta_i}, \sigma_{\phi_i})$ aux mesures réelles et avons utilisé un algorithme de triangulation basé sur (6) et un solveur des moindres carrés pour calculer les positions de sortie estimées. Nous avons ensuite calculé l'incertitude statistique de la position de la cible pour chaque paire d'emplacements de caméra.

La Fig. 2 montre les résultats de la comparaison entre le modèle analytique proposé et les simulations de Monte Carlo pour différentes paires d'emplacements de caméra dans un plan. La première caméra est fixée à 10m de la cible le long de l'axe X, et la seconde caméra est positionnée uniformément dans le cercle autour de la cible sur le plan XY. Nous avons utilisé pour les deux caméras un écart-type d'entrée de $\sigma_{\theta_i} = \sigma_{\phi_i} = 0,003$ rad. Pour chaque paire d'emplacements de caméra, 100 exécutions de Monte Carlo ont été utilisées pour calculer l'incertitude statistique de la sortie. L'incertitude de la coordonnée x de $\mathbf{X}$ en fonction de la distance et de l'angle dans le plan XY de la deuxième caméra est représentée graphiquement. Les autres composantes ne sont pas présentées ici, mais la composante x est la plus illustrative, puisque la première caméra est alignée sur l'axe X. Il est facile de voir que les valeurs de Monte Carlo sont très proches des résultats analytiques.

4) Fonction de récompense: À chaque pas de temps, les suiveurs reçoivent la récompense suivante :

$$r_i = \begin{cases} \frac{1}{\sqrt{\mathrm{Tr}(\mathbf{\Sigma_X})}}, & \text{si } d_{i,target} > d_{threshold} \\ & \text{ou } d_{i,ground} > d_{crash} \\ & \text{ou } \forall j \neq i \ d_{i,j} > d_{crash} \\ -r_{too\_close} & \text{si } d_{crash} < d_{i,target} < d_{threshold} \\ -r_{penalty}, & \text{sinon} \end{cases} \tag{15}$$

Où $\mathrm{Tr}(\mathbf{\Sigma_X})$ désigne la trace de la matrice de covariance $\mathbf{\Sigma_X}$, c'est-à-dire la variance de la distance du vecteur $\mathbf{X}$

par rapport à sa moyenne. Cela encourage les suiveurs à atteindre des positions autour de la cible où l'incertitude sur le résultat de l'algorithme de triangulation est minimale. Il s'agit d'une récompense commune calculée à partir de la position de chaque suiveur, ce qui encourage la collaboration.

Si l'un des suiveurs entre en collision avec la cible, le sol ou un autre suiveur au cours d'un pas de temps donné (sur la base d'un seuil de distance d_{crash}), le suiveur reçoit une pénalité individuelle $(-r_{penalty})$. Cela encourage chaque suiveur à éviter les collisions.

Additionnellement, une faible pénalité individuelle $(-r_{too\_close} << -r_{penalty})$ est infligée lorsqu'un suiveur s'approche trop prêt de la cible (sur la base d'un seuil de distance $d_{threshold}$), pour encourager les suiveurs à se tenir à une distance de sécurité respectable de la cible. Sans cette pénalité, les suiveurs seraient encouragés à s'approcher au plus prêt de la cible pour réduire l'incertitude triangulée, jusqu'à frôler la collision. Cet ajout peut aussi être justifié par le fait que, dans la pratique, un drone qui suit une cible à l'aide d'une caméra ne pourrait pas le faire s'il était trop proche, en raison du champ de vision limité des caméras.

V. Expériences en simulation

Nous avons implémenté notre scénario en simulation en adaptant les environnements de simulation de CrazyRL [20] basés sur l'API standard de la Farama Foundation pour les environnements MARL, PettingZoo [23]. CrazyRL est une bibliothèque MARL Python qui fournit des environnements de simulation et des outils pour faire du MARL avec les drones Crazyflie 2.1, commercialisés par Bitcraze AB. Nous avons utilisé cette bibliothèque pour tester ultérieurement nos modèles RL entraînés lors de vols réels, comme présenté dans VI. Les environnements d'entrainement de CrazyRL sont très rapides, ce qui est nécessaire pour pouvoir entrainer nos agents dans un temps raisonnable, mais au détriment de la complexité, car le modèle dynamique du drone n'est pas pris en compte, comme spécifié dans IV-A. Les environnements de simulation existants de CrazyRL ont été fortement personnalisés pour s'adapter à la description de notre scénario, à notre nouvelle fonction de récompense et à l'ajout de fonctionnalités de "domain randomization" (les positions de départ de la cible et des suiveurs ont été rendus aléatoires à chaque exécution, une caractéristique non présente dans l'implémentation originale de CrazyRL mais nécessaire pour obtenir une meilleure généralisation).

Nos environnements de simulation et notre implémentation de MAPPO [1] ont été codés avec JAX [24], [25]. Contrairement aux processus d'apprentissage RL plus classiques, avec cette implémentation le simulateur est aussi porté sur le GPU, ce qui permet d'éliminer les délais introduits par les communications entre CPU et GPU et de tirer pleinement parti des fonctionnalités de parallélisme des GPUs pour accélérer le processus d'apprentissage par plusieurs ordres de grandeur.

Nos politiques sont paramétrées par un perceptron multicouche à deux couches avec 128 unités par couche. Le réseau acteur fait correspondre les observations de l'agent aux vecteurs de moyenne et d'écart-type d'une distribution gaussienne multivariable à laquelle on applique une transformation *tanh*, à partir de laquelle les actions sont échantillonnées dans l'espace continu. La transformation *tanh* est utilisée pour contraindre les actions en sortie dans un intervalle fini [26].

[1] fournit des recommandations de bonnes pratiques pour le choix des hyperparamètres pour l'entrainement des réseaux de neurones profonds à l'aide de MAPPO. Conformément à ces recommandations, les hyperparamètres pertinents utilisés pour entraîner nos modèles sont résumés dans le tableau I. Tirant parti du fait que nos agents sont homogènes, nous avons utilisé le partage des paramètres pour accélérer l'apprentissage. Nous normalisons les récompenses et les observations. Nous avons entraîné nos modèles sur un ordinateur doté de 128 Go de RAM DDR5, d'un processeur à 16 cœurs cadencés à 5,73 GHz et d'un GPU GeForce RTX 4090 pour 200 millions pas de temps dans l'environnement.

Les résultats présentés dans cet article ont été obtenus avec des agents entraînés contre une cible fixe : la position de la cible est toujours aléatoire à chaque épisode d'entraînement, mais elle ne bouge pas. La récompense cumulée moyenne des épisodes au cours de l'entraînement pour 2 à 4 suiveurs est présentée dans la Fig. 3. Il est facile de voir que dans les trois cas, l'entraînement a convergé. Nous pouvons observer que les gains cumulés finaux augmentent avec le nombre d'agent. Cette différence peut être attribuée à la fonction de récompense qui récompense la minimisation de l'incertitude dans la position triangulée, et cette incertitude est connue pour diminuer avec le nombre de points de vue. La vitesse de convergence diminue avec le nombre d'agent, ce qui est normal puisque le temps d'apprentissage augmente avec la taille du modèle à entraîner, bien que la taille du batch utilisé pour chaque épisode d'apprentissage avec MAPPO augmente également avec le nombre d'agents. Nous pouvons notamment observer que le cas à quatre suiveurs converge beaucoup plus lentement, il n'atteint pas complètement l'asymptote après 200 millions pas de temps.

Pour évaluer les performances des modèles entraînés, nous

TABLE I
Hyperparamètres d'entraînement

Hyperparamètres	Valeur
num training episodes	num training steps / buffer length
batch size	num envs × buffer length × num agents
mini batch size	batch size / num of mini-batches
num parallel envs	128
num training steps	60e6
buffer length	1024
num of mini-batches	1
num of epoch per training	15
actor learning rate	5e-4
critic learning rate	5e-4
clip parameter	0.2
entropy coefficient	0.01
value loss coefficient	0.5
optimizer	Adam
optimizer epsilon	1e-5
weight decay	None

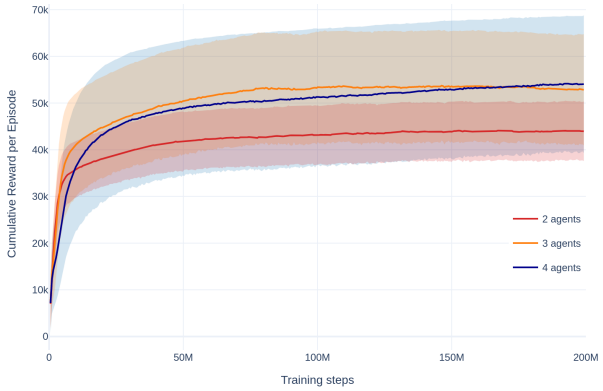

Fig. 3. Récompense moyenne cumulée par épisode au cours de l'entraînement pour différents nombres d'agents

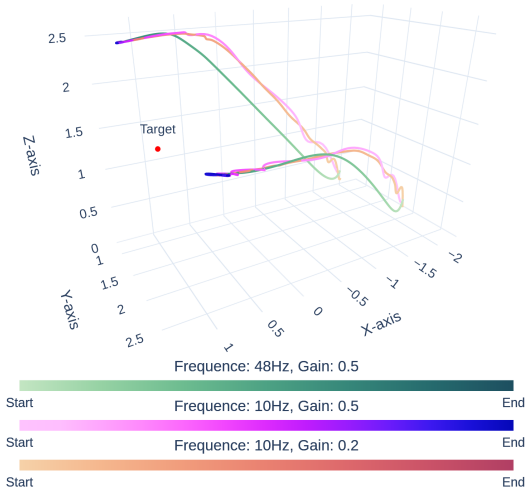

Fig. 4. Comparaison de l'effet de la fréquence de contrôle et du gain sur des trajectoires obtenues lors de vols simulés.

les avons évalués à la convergence dans un environnement réaliste, appelé *dronesim* [27], qui utilise le moteur Bullet Physics [28]. Cette fois, le modèle dynamique des drones est simulé et les drones sont dirigés à l'aide de lois de commande réalistes. Comme souvent dans les simulateurs de drones, le simulateur *dronesim* dispose de contrôleurs de guidage de haut niveau qui acceptent des positions de référence en entrée. Nos modèles calculent en sortie une commande en position autour de la position actuelle du drone dans le repère inertiel, que le contrôleur du drone convertit ensuite en commandes de bas niveau pour contrôler les rotors. Cependant, nos modèles ont été entraînés dans une simulation simpliste, où la dynamique du drone n'est pas prise en compte. En pratique, il est nécessaire d'ajuster la sortie du réseau en la multipliant par un gain, ajusté pour s'adapter à la dynamique du drone et à la fréquence de contrôle, afin de garantir que les commandes soient réalisables par le drone réel. Lorsqu'un contrôleur de drone reçoit une commande en position, il passe par des phases d'accélération, de vitesse maximale, et de décélération pour atteindre la position cible, lié à la dynamique et l'inertie du drone. Un gain élevé éloigne les commandes générées de la position du drone, ce qui encourage le drone à accélérer rapidement jusqu'à sa vitesse maximale. Au contraire, un gain faible rapproche les commandes en position du drone, trop proche et le drone n'accélérera pas jusqu'à sa vitesse maximale, le rendant plus lent. La fréquence de contrôle influe également sur la valeur du gain. Les commandes sont générées en supposant que le drone atteindra la position demandée au pas de temps suivant. Lorsque le gain est élevé et la fréquence faible, le drone peut dépasser la position désirée, produisant des oscillations, surtout en vol stationnaire. Si la fréquence de contrôle est très élevée, ce phénomène est largement amorti. Dans la Fig. 4 nous avons tracé les trajectoires de deux drones autour d'une cible fixe avec différentes valeurs de gain et de fréquence de contrôle. Il apparait que lorsque la fréquence de contrôle est haute, les trajectoires sont plus stables. Au contraire lorsqu'elle est faible, un gain faible stabilise le vol.

La Fig. 5 présente plusieurs trajectoires obtenues dans le simulateur *dronesim* avec un gain correctement ajusté et une fréquence de 10Hz. Dans la figure en haut à gauche, nous pouvons observer comment les suiveurs se placent autour d'une cible fixe et stationnent à des positions fixes à une distance de sécurité de la cible. De plus, même si nous n'avons pas été entraînés à le faire, nous avons évalué notre politique contre une cible en mouvement. Les trois autres figures présentent une cible se déplaçant en cercle à la même vitesse maximale que les suiveurs (2m/s). Nous pouvons observer que, bien qu'entraînée avec des cibles fixes, la politique résultante peut s'adapter à des cibles qui se déplacent lentement. Ce résultat est dû aux propriétés markoviennes inhérentes à l'apprentissage par renforcement, car les actions de contrôle sont calculées sur la base de l'état actuel uniquement et non des données antérieures. Cependant, comme les politiques

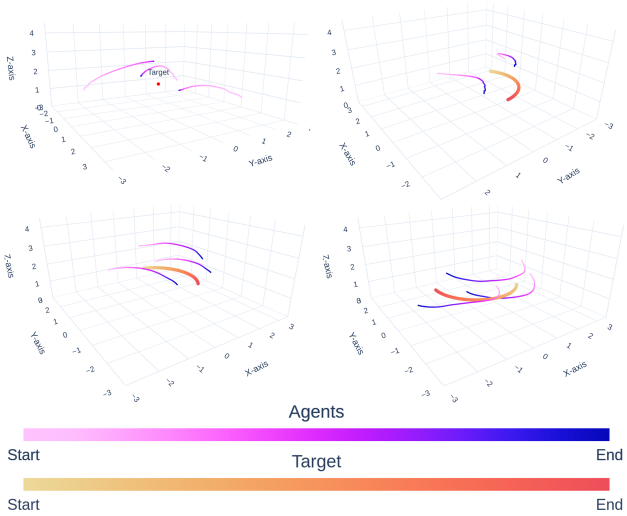

Fig. 5. Trajectoires obtenues lors de vols simulés. La cible est indiquée en gradients de rouge, Les trajectoires des drones sont tracées en gradients de bleu.

n'ont pas été formées avec une cible en mouvement, les suiveurs n'ont pas appris à anticiper les mouvements de la cible, ce qui entraîne un retard dans leur réponse.

VI. Test en vol réel

Pour observer le comportement dans des scénarios réels de la politique apprise, nous avons déployé les scénarios d'observation de cibles fixes utilisant deux suiveurs et trois suiveurs dans des démonstrations réelles. Les expériences ont eu lieu dans l'arène de vol de l'ENAC. Les dimensions de l'espace de vol sont $L \times W \times H = 8 \times 8 \times 4\,\text{m}$ et il est équipé d'un système de capture de mouvement, qui calculent la position et l'orientation des objets volant dans la volière.

Les engins utilisés lors de la démonstration sont des DJI Tello. Nous avons utilisé la bibliothèque Python DJITelloPy, qui dispose de contrôleurs de guidage de haut niveau acceptant des positions de référence en entrée. Par conséquent, comme pour V avec les drones simulés, nous transmettons directement la commande de position calculée, avec un gain de contrôle ajusté pour la dynamique de vol des Tellos et une fréquence de 10 Hz.

Des exemples de trajectoires de vol sont présentés dans les Fig. 6 et Fig. 7. Fig. 6, on remarque immédiatement que les trajectoires obtenues pendant les vols réels sont plus bruitées que les trajectoires simulés. Cela peut provenir du bruit dans les estimations de position des suiveurs, dans celle de la cible et de la dynamique de vol des suiveurs. Cependant, la politique de guidage réussit à atteindre et à se stabiliser au-dessus d'un point cible fixe.

Comme dans la section V, la politique est également testée sur une cible mobile, qui tourne en rond horizontalement au centre de la volière avec un rayon de 1,5 m et à 0,6 m de hauteur. Les trajectoires obtenues sont présentées Fig. 7, et nous pouvons comparer avec les mêmes vols effectués en simulation présentés Fig. 5. Les oscillations observées dans le vol réel sont dues aux perturbations du flux d'air de la

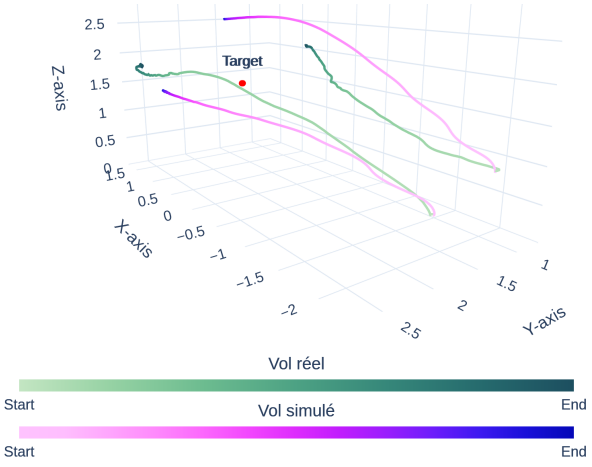

Fig. 6. Comparaison des trajectoires d'un vol réel et d'un vol simulé en direction d'une même cible.

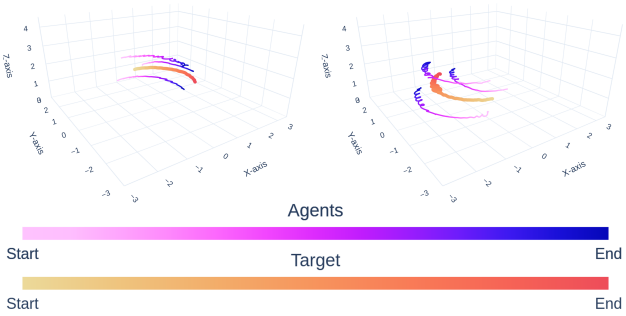

Fig. 7. Exemple de mouvements successifs de suiveurs lorsque la cible se déplace en cercle.

cible par le suiveur qui la survolent. Ces perturbations, qui ne sont pas simulés dans *dronesim*, font osciller la cible et par conséquent les suiveurs qui suivent ses mouvements.

VII. Conclusion

Nous avons introduit une approche de guidage de drones basée sur MARL pour la triangulation à N-vues de cibles volantes à l'aide de caméras omnidirectionnelles embarquées. Nous avons utilisé MAPPO pour apprendre avec succès des politiques décentralisées qui permettent aux drones d'ajuster dynamiquement leurs positions pour trianguler de manière optimale la cible, réduisant ainsi l'incertitude dans l'estimation de la localisation. Nous avons proposé une fonction de récompense basée sur la trace de la matrice de covariance de la position triangulée, calculée à l'aide d'un modèle analytique de propagation de l'incertitude. Une comparaison avec l'approche de Monte Carlo montre un bon accord avec le modèle proposé. En outre, nous avons évalué les politiques entraînées en simulation et les avons testées lors d'expériences en vol réel, montrant qu'elles peuvent gérer différents scénarios et être transférées dans des environnements réalistes. Notre approche est une étape prometteuse vers l'utilisation de MARL pour le suivi de plusieurs drones. Outre des évaluations et des analyses expérimentales plus approfondies, nous évaluerons à l'avenir le comportement du système avec un plus grand nombre d'agents. Plus important encore, nous ne supposerons plus que les positions des suiveurs sont connues de tous les suiveurs, mais seulement partiellement observées par la vision. Nous envisagerons également le cas de cibles multiples.

Remerciements

Les auteurs tiennent à remercier Arnaud SAMAMA, Frédéric BARBARESCO, Jean-Marc TRIN et Pierre-Louis CARTIER de Thales LAS, ainsi que Florence ALIGNE de Thales RT pour leur précieuse contribution à ce sujet.

References

[1] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, "The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 611–24 624, Dec. 2022.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Tech. Rep., Aug. 2017.

[3] S. Ramalingam, S. K. Lodha, and P. Sturm, "A generic structure-from-motion framework," *Computer Vision and Image Understanding*, vol. 103, no. 3, pp. 218–228, Sep. 2006.

[4] R. A. Zitar, A. Mohsen, A. E. Seghrouchni, F. Barbaresco, and N. A. Al-Dmour, "Intensive Review of Drones Detection and Tracking: Linear Kalman Filter Versus Nonlinear Regression, an Analysis Case," *Archives of Computational Methods in Engineering*, vol. 30, no. 5, pp. 2811–2830, Jun. 2023.

[5] A. Khan, B. Rinner, and A. Cavallaro, "Cooperative Robots to Observe Moving Targets: Review," *IEEE Transactions on Cybernetics*, vol. 48, no. 1, pp. 187–198, Jan. 2018.

[6] B. Bethke, M. Valenti, and J. How, "Cooperative Vision Based Estimation and Tracking Using Multiple UAVs," in *Advances in Cooperative Control and Optimization*, ser. Lecture Notes in Control and Information Sciences, P. M. Pardalos, R. Murphey, D. Grundel, and M. J. Hirsch, Eds. Berlin, Heidelberg: Springer, 2007, pp. 179–189.

[7] W. Zhou, J. Li, Z. Liu, and L. Shen, "Improving multi-target cooperative tracking guidance for UAV swarms using multi-agent reinforcement learning," *Chinese Journal of Aeronautics*, vol. 35, no. 7, pp. 100–112, Jul. 2022.

[8] S. Srigrarom, S. M. Lee, M. Lee, F. Shaohui, and P. Ratsamee, "An Integrated Vision-based Detection-tracking-estimation System for Dynamic Localization of Small Aerial Vehicles," in *2020 5th International Conference on Control and Robotics Engineering (ICCRE)*, Apr. 2020, pp. 152–158.

[9] Y. Chen, M. Tsukada, and H. Esaki, "Reinforcement Learning Based Optimal Camera Placement for Depth Observation of Indoor Scenes," Tech. Rep., Oct. 2021.

[10] M. A. Akhloufi, S. Arola, and A. Bonnet, "Drones Chasing Drones: Reinforcement Learning and Deep Search Area Proposal," *Drones*, vol. 3, no. 3, p. 58, Sep. 2019.

[11] Z. Xia, J. Du, J. Wang, C. Jiang, Y. Ren, G. Li, and Z. Han, "Multi-Agent Reinforcement Learning Aided Intelligent UAV Swarm for Target Tracking," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 1, pp. 931–945, Jan. 2022.

[12] F. A. Oliehoek, C. Amato *et al.*, *A concise introduction to decentralized POMDPs*. Springer, 2016, vol. 1.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[14] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[15] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning." PMLR, Jul. 2018, pp. 4295–4304.

[16] J. Su, S. Adams, and P. Beling, "Value-Decomposition Multi-Agent Actor-Critics," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, pp. 11 352–11 360, May 2021.

[17] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'18/IAAI'18/EAAI'18, vol. 32, no. 1. New Orleans, Louisiana, USA: AAAI Press, Apr. 2018, pp. 2974–2982.

[18] R. Lowe, Y. WU, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[19] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge: Cambridge University Press, 2004.

[20] F. Felten, C. Ledez, P.-Y. Houitte, E.-G. Talbi, and G. Danoy, "Crazyrl: A multi-agent reinforcement learning library for flying crazyflie drones," https://github.com/ffelten/CrazyRL, 2023.

[21] G. Di Leo, C. Liguori, and A. Paolillo, "Propagation of uncertainty through stereo triangulation," in *2010 IEEE Instrumentation & Measurement Technology Conference Proceedings*, May 2010, pp. 12–17.

[22] C. M. G and P. M. Harris, "Software Support for Metrology Best Practice Guide No 6 - uncertainty evaluation." Sep. 2006.

[23] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente *et al.*, "Pettingzoo: Gym for multi-agent reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 032–15 043, 2021.

[24] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018.

[25] DeepMind, I. Babuschkin, K. Baumli, A. Bell, S. Bhupatiraju, J. Bruce, P. Buchlovsky, D. Budden, T. Cai, A. Clark, I. Danihelka, A. Dedieu, C. Fantacci, J. Godwin, C. Jones, R. Hemsley, T. Hennigan, M. Hessel, S. Hou, S. Kapturowski, T. Keck, I. Kemaev, M. King, M. Kunesch, L. Martens, H. Merzic, V. Mikulik, T. Norman, G. Papamakarios, J. Quan, R. Ring, F. Ruiz, A. Sanchez, L. Sartran, R. Schneider, E. Sezener, S. Spencer, S. Srinivasan, M. Stanojević, W. Stokowiec, L. Wang, G. Zhou, and F. Viola, "The DeepMind JAX Ecosystem," 2020.

[26] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," Tech. Rep., Aug. 2018.

[27] L. T. Fernandez, M. Bronz, T. Lefebvre, and N. Bartoli, "Multi-Vehicle Simulation Framework for Heterogeneous Unconventional MAVs," in *IMAV 2023*, AACHEN, Germany, Sep. 2023.

[28] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.