# Multi-Agent Reinforcement Learning based Drone Guidance for N-View Triangulation

Timothée Gavin, Simon Lacroix, Murat Bronz

# Multi-Agent Reinforcement Learning based Drone Guidance for N-View Triangulation

Timothée Gavin[*x+], Murat Bronz[x], Simon Lacroix[+]
[*]*IAS, Thales LAS*, Rungis, France
[x]*Dynamic Systems, OPTIM ENAC, Université de Toulouse* Toulouse, France
[+]*RIS, LAAS-CNRS* Toulouse, France
*timothee.gavin@thalesgroup.fr , murat.bronz@enac.fr , simon.lacroix@laas.fr*

***Abstract*—This article presents a novel approach for controlling a fleet of drones that can track the location of a flying target using onboard omnidirectional cameras. The drones use Multi-Agent Reinforcement Learning (MARL) to learn decentralized policies that optimize their formation and motion around the target, minimizing the uncertainty in the triangulated position. We design a reward function that encourages the trackers to minimize the trace of the covariance matrix of the triangulated position, which is derived from an analytical model of uncertainty propagation. We use Multi-Agent PPO (MAPPO), an extension of Proximal Policy Optimization (PPO) to the multi-agent setting, to train the policies using this common reward function that encourages good formation and avoids collisions. We validate our approach in simulation and real-flight experiments, demonstrating its effectiveness and potential in enhancing autonomous multi-drone coordination for precise tracking.**

***Index Terms*—Reinforcement Learning, Multi-Agent, PPO, Triangulation**

## I. Introduction

In the context of pursuit-evasion scenarios involving multiple drones, there is a need to accurately track the location of small aerial vehicles in 3 dimensions. Whether to know the location of other members of the fleet or to track one or more targets before and during interception.

In this article, we aim to take the first steps toward a novel approach for controlling a fleet of drones that can track the location of one or more flying targets using onboard cameras. The guidance algorithm should optimize the placement of the drones to minimize the triangulation uncertainty on the tracked targets, even when they move with unpredictable trajectories.

The proposed approach uses Multi-Agent Reinforcement Learning (MARL) to optimally position multiple drones to jointly triangulate the position of a single target using on-board omnidirectional cameras. We use Multi-Agent PPO (MAPPO) [1], an extension of Proximal Policy Optimization (PPO) [2] to the multi-agent setting, to train decentralized policies using a purpose-built common reward function that encourages good formation around the target to minimize uncertainty in the triangulated position. The trained models are validated in simulation and the resulting behavior is demonstrated in real-flight experiments.

We investigate MARL over traditional optimization methods because we expect benefits in solving large-scale triangulation problems. Traditional optimization methods, even using heuristics, are slow and computationally expensive for large-scale problems, which is unsuitable with an onboard guidance algorithm. While training the RL agent is computationally expensive, it can compute fast solutions after learning.

The organization of this paper is as follows. After reviewing the relevant related work on drone tracking using drones in Section II, and presenting the existing background in MARL and triangulation in Section III, we present our MARL drone guidance approach for N-view triangulation in Section IV. We detail the simulation setup and results in Section V. Finally, we describe the real-flight experiments in Section VI.

## II. Related Work

Triangulation is the process of estimating the 3-D position of a point from its projections on two or more images taken from different viewpoints. This can be done by intersecting the projection rays associated with each image point, or by minimizing the reprojection error between the 3-D point and the image points. However, triangulation is sensitive to errors in the camera calibration and pose estimation, as well as to noise and outliers in the image points. A common technique to optimize the placement of the cameras and the triangulation estimation in N-view triangulation is *bundle adjustment* [3]. Bundle adjustment is a non-linear least squares optimization method that simultaneously refines the intrinsic and extrinsic camera parameters, by minimizing the sum of squared reprojection errors over all image points and views. This method is widely used in structure-from-motion problems, where it helps to reconstruct 3-D scenes from calibrated images taken by different cameras. However, the computational cost and memory requirements of bundle adjustment scale superlinearly with the number of cameras [4].

Tracking is the process of estimating the position and orientation of a moving object over time from a sequence of images. Techniques such as Kalman Filters and Nonlinear Polynomial Regression are commonly used for drone tracking [4]. Most research on cooperative mobile robots for observing moving targets focuses on ground targets moving in a 2-D plane [5]–[7]. Some studies have proposed the use of onboard cameras for real-time tracking and 3-D localization of multiple drones [8]. However, the tracking of drones using a camera aboard another drone remains a relatively less explored area in

the literature, as it poses more difficulties than tracking with fixed camera systems.

Reinforcement learning (RL) is a machine learning paradigm that enables an agent to learn from its own actions and rewards. RL has been applied to various problems involving the optimal placement of sensors and guidance algorithms for drones and fleets of drones. RL has been used for depth observation of indoor scenes using multiple cameras, where the agent learns to select the best camera positions and orientations to minimize the depth observation error [9]. Alternatively, an RL-based approach for drone pursuit-evasion, where the agent learns to follow a target drone using sensor data and a deep object detector, was presented in [10]. These works mainly focus on single-agent RL, where one agent interacts with the environment independently. MARL is a branch of RL that deals with multiple agents that cooperate or compete with each other in a shared environment. MARL has been used to learn the optimal formation and motion of a fleet of drones for triangulation, by maximizing the coverage and diversity of the views and to optimize the communications [7], [11], but again these studies focus on observing ground targets moving in a 2-D plane. To our knowledge, no recent studies have addressed the optimal placement of a fleet of drones using MARL to track flying objects using onboard cameras.

## III. Background

### A. Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs)

We consider a fully cooperative multi-agent task that can be described as a Decentralized Partially Observable Markov Decision Process [12] defined by a tuple $(n, S, A, \Omega, T, O, R, \gamma)$, where $n$ is the number of agents, $S$ is the state space, $A = A_1 \times \ldots \times A_N$ and $\Omega = \Omega_1 \times \ldots \times \Omega_N$ are the set of joint actions and joints observations with each $A_i$ and $\Omega_i$ being the local action and local observation sets of agent $i$, $T : S \times A \times S \rightarrow [0, 1]$ is the state transition probability function, $O : S \times A \times \Omega \rightarrow [0, 1]$ is the observation probability function, $R : S \times A \rightarrow \mathbb{R}$ is the instant reward function, and finally $\gamma \in [0, 1]$ is the discount factor.

At each time step $t$, each agent $i$ chooses an action $a_{i,t} \in A_i$ based on its local observation history $o_{i,1:t}$, and receives a local observation $o_{i,t+1} \in \Omega_i$ based on the resulting state $s_{t+1}$. The joint action $a_t = (a_{1,t}, \ldots, a_{N,t})$ determines the immediate reward $r_t = R(s_t, a_t)$ and the next state $s_{t+1}$ according to the transition probability $T(s_t, a_t, s_{t+1}) = \Pr(s_{t+1}|s_t, a_t)$. The objective is to find a policy $\pi_\theta(a_{i,t}|o_{i,1:t})$ parameterized by $\theta$ that produces an action $a_{i,t}$ from the local observation history $o_{i,1:t}$, which maximizes the expected discounted return $J(\theta) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t]$. The state value function $V^{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|S_t = s]$ is the expected return from starting in state $s$ and following policy $\pi_\theta$ thereafter. The state-action value function $Q^{\pi_\theta}(s, a) = \mathbb{E}_{\pi_\theta}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|S_t = s, A_t = a]$ is the expected return from starting in state $s$, taking action $a$, and following policy $\pi$ thereafter.

### B. Single-Agent Reinforcement Learning Algorithms

Most Reinforcement Learning methods can be divided into two groups: value-based and policy-based methods. Value-based methods, such as Deep Q-Learning (DQN) [13], use deep neural networks to estimate the value functions and derive the optimal policy $\pi^*$ from the optimized value functions by choosing for each state $s$ the action that maximizes the action-value: $\pi^*(s) = \arg\max_a Q^*(s, a)$. Policy-based methods on another hand directly parameterize the policy $\pi_\theta(a|s)$ as a function of the state and the action. Policy gradient methods are a subclass of policy-based methods that update the policy parameters $\theta$ by following the gradient of the expected return: $\nabla_\theta J(\theta) = E_\pi[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)]$. Using the full return from each episode to estimate the gradient leads to high variance in the gradient estimates. To reduce this variance, Actor-Critic methods were introduced [14]. These methods maintain an explicit separate function approximator (the Critic) to estimate the value function, which is used as a baseline to compute the Advantage function. The Advantage function, $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$, measures how much better an action $a$ is compared to the average action at state $s$ under policy $\pi$. This helps reduce the gradient estimates' variance, leading to more stable learning. PPO [2] is a further development in policy gradient methods that seeks to update the policy in a way that avoids large, abrupt changes that could destabilize the learning process, thereby ensuring stable and efficient learning.

### C. Multi-Agent Reinforcement Learning

Single-agent RL methods often fail in multi-agent settings due to the well-known curse of dimensionality and non-stationarity. Recently, MARL approaches addressed these issues with the Centralized Training, Decentralized Execution (CTDE) approach. By training the agents in a centralized manner to leverage global information and then executing the learned policies in a decentralized manner, CTDE ensures scalability and robustness. Several MARL algorithms have been developed under the CTDE framework, including value-based methods like QMIX [15] and VDAC [16], and policy gradient methods like COMA [17] and MADDPG [18]. PPO has shown great promise in the MARL setting: Multi-Agent PPO (MAPPO) [1], an extension of PPO in the CTDE framework, uses a centralized critic and decentralized actors and has demonstrated superior performance in various complex multi-agent tasks compared to other state-of-the-art MARL algorithms [1].

### D. N-View Linear Triangulation

Linear triangulation is a method to estimate the 3-D coordinates of a point from its 2-D projections in two or more images taken by different cameras. The principle of linear triangulation is based on the pinhole camera model, which relates the 3-D point $\mathbf{X}$ and its 2-D projection $\mathbf{x}$ in homogeneous coordinates by the $3 \times 4$ camera matrix $\mathbf{P}$ as $\mathbf{x} = \mathbf{PX}$. The camera matrix $\mathbf{P}$ encapsulates both the intrinsic parameters (such as focal length and optical center)

and extrinsic parameters (including rotation and position) of a camera.

Linear triangulation solves an overdetermined linear system of the form $\mathbf{AX} = 0$, which in the case of N-view triangulation, with $n$ of these image points and camera calibration matrices gives:

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

where $\mathbf{X} = (\lambda x, \lambda y, \lambda z, \lambda)^T$, $\lambda$ being an unknown scale factor, and $u_i$, $v_i$ and $P_i^{jT}$ are the image coordinates and the $j$th row of the camera matrix $P_i$ of the $i$th camera. To find $\mathbf{X}$, one finds a non-zero solution X that satisfies $\mathbf{AX} = 0$. In the case of noisy measurements, because the $n$ rays defined by the 2-D projection points and their respective camera centers don't intersect in a single point, the system has no non-zero solutions, and the problem is transformed into a minimization problem to find an approximate solution, usually solved with least-square techniques [19].

## IV. Multi-Agent Reinforcement Learning Drone Guidance for N-View Triangulation

### A. Presentation of the scenario

Our problem consists of multiple drones, named hereafter "trackers", chasing a single drone, the "target" to track its position, using on-board sensors such as cameras. The trackers aim to arrange their fleet formation so that the triangulated position uncertainty is minimal.

The trackers and the target motions are omnidirectional, we did not implement a drone flight model in this study. We assume that the target has a constant flight velocity. The drones fly in a squared arena without any obstacles in it. Neither the trackers nor the target can get out of the arena: their actions are clipped to stay inside the arena borders. Collisions between trackers, or between a tracker and the target, result in a failure. Crashing into the ground also results in a failure. The orientation of the drones is not considered in this study, it is assumed that all drones are aligned with their heading parallel to the positive direction of the X-axis in the global coordinate system.

We assume that the trackers can differentiate the other trackers from the target. We did not consider partial observability in this study. We make the strong assumption that every tracker knows the exact 3-D positions in the world, without noise, of the other trackers of the fleet and also of the target. However, the actions taken by the trackers are decentralized, meaning that they do not know the actions taken by the other members of the fleet.

The camera on board the trackers is assumed to be omnidirectional, without a restricted field of view, and unaffected by occlusions. The camera center coincides with the trackers' 3-D location. We also assume that the camera's intrinsic parameters do not introduce any distortion in the measurements. Rather than measuring the location of a world point's projection on a 2-D camera plane, the omnidirectional camera simply measures the polar and azimuth angles within a spherical frame that is centered around the tracker's position. In light of these simplifications, the triangulation uncertainty is only the result of the measurement uncertainty along the polar and azimuth angles within each tracker frame. This drove the shaping of the reward function described in Section IV-D.

### B. Multi-Agent Reinforcement Learning Algorithm

In this study, we employ the Multi-Agent Proximal Policy Gradient (MAPPO) algorithm as our Deep RL algorithm due to its simplicity, its ability to work in continuous state and action spaces, and its demonstrated effectiveness in various MARL tasks [1]. We assume all agents to be homogeneous, enabling us to use parameter sharing for training, thereby accelerating the learning process and maximizing the information extracted from each interaction with the environment. MAPPO operates on a CTDE framework, it uses a centralized critic and decentralized actors: while all agents are governed with a common policy, and trained centrally, they act independently based on their local observations at each time step, creating a decentralized system. We train a policy for a given number $n$ of trackers. This approach is needed as we did not address the varying length of the state representation depending on the number of trackers in the environment.

### C. State Representation

The state space, observation, and action spaces are continuous. Each tracker 3-D position in world Cartesian coordinates is encoded in a 3-D vector $\mathbf{p_i}$. The 3-D position of the target is $\mathbf{p_T}$. The $n$ trackers are ordered so that $\mathbf{p_i}$ is always the position of the $i$th tracker. The state of the environment is the ordered tuple $\mathbf{s} = (\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}, \mathbf{p_T})$ and is the input of the centralized critic network used in the MAPPO algorithm. For each tracker $i$, its observation of the environment is encoded in the ordered tuple $\mathbf{o_i} = (\mathbf{p_i}, \mathbf{p_1}, \ldots, \mathbf{p_{i-1}}, \mathbf{p_{i+1}}, \ldots, \mathbf{p_n}, \mathbf{p_T})$ and is the input of the decentralized actor-network.

As for the action space, as explained in Section IV-A, the trackers' motion is holonomic. At each step, the agents navigate within a sphere of a 20cm radius centered on their prior location (this value is directly inherited from the simulation environment CrazyRL [20] that we adapted to implement our simulation environment, see Section V). This radius is determined by the agent's maximum speed (approximately 2m/s) divided by the control frequency (10Hz in the training environment). To compute this next step position, the actor network outputs a 3-D point in $[-1, 1]^3$, which is subsequently scaled by 20cm.

### D. Reward Structure

At each time step, every agent receives a common reward designed to encourage the trackers to reach positions around

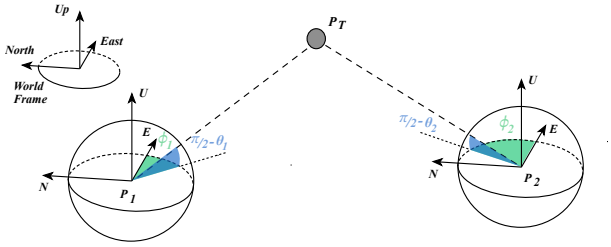


Fig. 1. Representation of omnidirectional cameras in spherical frame. Polar angles $\theta$ are shown with respect to horizontal plane for easy visualization.

the target where the uncertainty on the result of the triangulation algorithm is minimal.

The evaluation of the uncertainty of the result of an estimation algorithm can be done with two approaches. An *a posteriori* approach involves implementing a Monte Carlo method: the uncertainty of the output is statistically obtained from repeated random sampling given a noisy input distribution. While being easy to implement, it requires a significant number of runs, which can be time-consuming. The second method is an *a priori* approach, which involves developing analytical relationships using linear approximations to describe how uncertainty propagates from inputs to outputs. This second approach requires only one run to compute the uncertainty of the output. This makes it the preferred approach for our reward function, since it is computed at each interaction step with the environment.

*1) Triangulation with omnidirectional cameras:* In our simplified case with omnidirectional cameras, instead of measuring the position of the projection of the world point on the 2-D camera plane, we measure the polar and azimuth angle in the spherical frame centered on the tracker's position. For each tracker at location $\mathbf{p} = (p_x, p_x, p_z)$ in Cartesian global coordinates, the Cartesian global coordinates of a point $\mathbf{X} = (x, y, z)$ are related to its local spherical coordinates following:

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

with $\theta \in [0, \pi]$ the polar angle, and $\phi \in [0, 2\pi[$ the azimuth angle. The radial distance $r \in \mathbb{R}^+$, is unknown in the triangulation problem. Let's define the following vector :

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

This vector, often named *direction cosines*, defines the direction of the point $\mathbf{X}$ in the $i$th tracker local Cartesian coordinate system using the spherical coordinates. Together with the location of the tracker $\mathbf{p_i}$, they define a line passing by $\mathbf{X}$. In the case of triangulation with two omnidirectional cameras, we have:

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

We can eliminate the unknown radial distances with a cross-product to obtain three equations for each tracker, very similar to the one of linear triangulation in the case of pinhole cameras in (1). Note that, the third equation is a linear composition of the two others and could be removed.

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

We obtain, for $n$ trackers, an overdetermined linear system of the form $\mathbf{AX} = \mathbf{b}$, which is solvable using a method such as Singular Value Decomposition or using least-square techniques to obtain the value of $\mathbf{X}$.

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

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 of uncertainty:* The triangulation equations in (6) has the implicit form:

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

where $f_\rho$ is parameterized by the parameter vector $\rho = (\mathbf{p_1}, \dots, \mathbf{p_n})$, $\mathbf{X}$ is the triangulated 3-D point in output of the triangulation algorithm and $\mathbf{v}$ is a vector of noisy inputs $\mathbf{v} = (\theta_1, \phi_1, \dots, \theta_n, \phi_n)$, governed by Gaussian noise with covariance matrix $\mathbf{\Sigma_v}$.

In such case, the covariance matrix $\mathbf{\Sigma_X}$ of the output $\mathbf{X}$, is related to the covariance matrix $\mathbf{\Sigma_v}$ by [21], [22]:

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

where $\mathbf{J_X}$ and $\mathbf{J_v}$ are the Jacobians matrices of the partial derivatives of $f_\rho$ with respect to, respectively, the output $\mathbf{X}$ and the input $\mathbf{v}$. Then, the output covariance matrix can be expressed as a function of the input covariance matrix (which is either known or presumed) by:

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

where $\mathbf{J_X}^+$ is the pseudo-inverse of $\mathbf{J_X}$.

In our case, the Jacobians can be analytically expressed using (6). The input vector $\mathbf{v}$ can be decomposed in $\mathbf{v} = (\mathbf{v_1}, \mathbf{v_2}, \dots, \mathbf{v_n})$ with $\mathbf{v_i} = (\theta_i, \phi_i)$, then $f_\rho(\mathbf{X}, \mathbf{v})$ can be written:

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

with each $f_{\mathbf{p_i}}(\mathbf{X}, \mathbf{v_i}) = [\mathbf{A_i} \cdot (\mathbf{X} - \mathbf{p_i})]^T$. Hence, the following Jacobians matrices:

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

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

with

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

*3) Comparison with Monte Carlo approach:* To validate the previous analytical model of the output uncertainty, we compared the uncertainty obtained with the analytical approach with the results of a Monte Carlo approach. We set up, in simulation, two cameras at different locations in a sphere around a target point. Assuming that the inputs are subject to Gaussian noise, we added Gaussian noise with standard deviation $(\sigma_{\theta_i}, \sigma_{\phi_i})$ to the true measurements and used a triangulation algorithm based on (6) and a least-square solver to calculate estimated output positions. We then computed the statistical uncertainty of the target position for each pair of camera locations.

Fig. 2 shows the results of comparing the proposed analytical model and the Monte Carlo simulations for different pairs of camera locations in a plane. The first camera is fixed at 10m of the target along the X axis, and the second camera is positioned uniformly in the circle around the target on the XY plan. We used for both cameras an input standard deviation of $\sigma_{\theta_i} = \sigma_{\phi_i} = 0.003$ rad. For each pair of camera locations, 100 Monte Carlo runs were used to compute the statistical uncertainty of the output. The uncertainty of the $x$ coordinate of $\mathbf{X}$ as a function of the distance and the angle in the XY plane of the second camera is plotted. The other components are not presented here, but the $x$ component is the most illustrative, as the first camera is aligned on the X-axis.

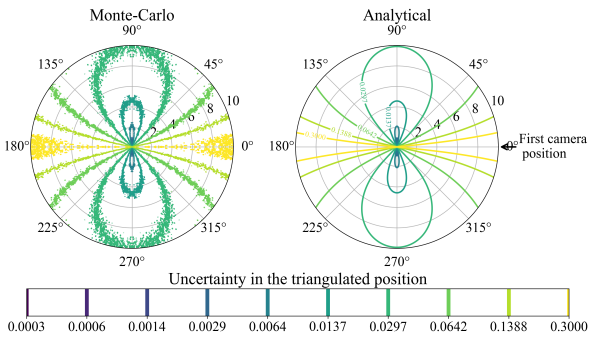


Fig. 2. Uncertainty of the triangulated output on the X axis for two cameras parallel to the XY plane.

It is easy to see that the Monte Carlo values closely align with the analytical results.

*4) Reward function:* At each time step, the trackers receive the following reward:

$$r_i = \begin{cases} \frac{1}{\sqrt{\mathrm{Tr}(\mathbf{\Sigma_X})}}, & \text{if } d_{i,target} > d_{threshold} \\ & \text{and } d_{i,ground} > d_{threshold} \\ & \text{and } \forall j \neq i \ d_{i,j} > d_{threshold} \\ -r_{penalty}, & \text{otherwise} \end{cases} \tag{15}$$

Where $\mathrm{Tr}(\mathbf{\Sigma_X})$ denotes the trace of the covariance matrix $\mathbf{\Sigma_X}$, i.e. the variance of the distance of the $\mathbf{X}$ vector to its means. This encourages the trackers to reach positions around the target where the uncertainty on the result of the triangulation algorithm is minimal. This is a common reward calculated using the position of every tracker, which encourages collaboration.

If any of the trackers collide with the target, the ground, or another tracker within a specified time step (based on a distance threshold $d_{threshold}$), then the tracker receives an individual penalty $(-r_{penalty})$. This encourages each tracker to avoid collisions.

## V. Simulation Experiments

We implemented our scenario in simulation by adapting the simulation environments of CrazyRL [20] based on Farama Foundation's standard API for MARL environments, PettingZoo [23]. CrazyRL is a MARL Python library that provides simulation environments and tools to do MARL with Crazyflie 2.1 drones, commercialized by Bitcraze AB. We used this library to test later our trained RL models in real flights as presented in VI. CrazyRL training environments are very fast, which is necessary to be able to train our agents in a reasonable amount of time, yet this comes at the expense of complexity, as the drone's dynamic model is not considered, as specified in IV-A. CrazyRL's existing simulation environments were heavily customized to fit our scenario description, our new reward function, and the addition of domain randomization (the starting positions of the target and the trackers were randomized on each run, a feature not present in the original CrazyRL implementation but necessary to achieve better generalization).

We trained our RL models using the original implementation of MAPPO [1] using Pytorch [24]. The training loop examples provided in the original implementation by the authors of [1] were adapted to fit PettingZoo's API.

Our policies are parameterized by a two-layer Multi-Layer Perceptron with 64 units per layer. The actor-network maps the agent observations to the mean and standard deviation vectors of a Multivariate Gaussian distribution followed by a Tanh transformation, from which the continuous actions are sampled. The Tanh transformation is used to constrain the output actions to a finite interval [25].

[1] provides best-practice suggestions for hyperparameter choice for training models using MAPPO. Following these recommendations, the relevant hyperparameters used to train our models are summarized in Table I. Leveraging the fact

TABLE I
TRAINING HYPERPARAMETERS

| Hyperparameters | Value |
|---|---|
| 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 | 30e6 |
| 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 | 0.1 |
| value loss coefficient | 0.5 |
| optimizer | Adam |
| optimizer epsilon | 1e-5 |
| weight decay | None |

that our agents are homogeneous, we used parameter sharing and parallelization to speed up training. We use reward and observation normalization. We trained our models on a desktop machine with 32 GB DDR2 RAM, an 8-core 2.00GHz CPU, and a GeForce 1080Ti GPU for 30 million environment steps and then evaluated them on several evaluation episodes to average the various metrics presented in Table II.

The results in this paper were obtained with agents trained against a fixed target. The target's position is still randomized on each training episode, but it doesn't move.

The average episode cumulated reward over the training for 2 and 3 trackers is shown in Fig. 3. It is easy to see that in both cases, the training converged. We can observe that compared to the 3-tracker case, the 2-tracker case converges to slightly lower final returns in the same amount of training steps. This difference can be attributed to the reward function that rewards the minimization of the uncertainty in the triangulated position, and this uncertainty is known to scale down with the number of points of view. Both cases have a similar speed of convergence, with the 2-tracker case slightly faster, which is to be expected as the training time scales up with the size of the model to train, even though the batch size used for each training episode with MAPPO also scales up with the number of agents.

To evaluate the performances of the trained models, we

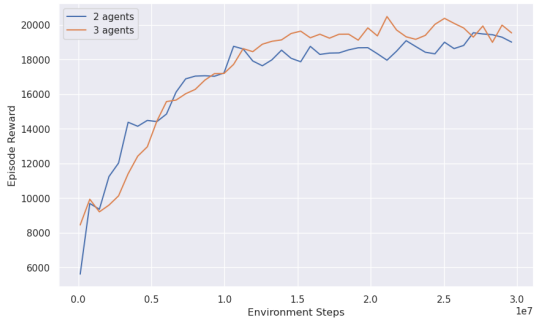


Fig. 3. Average episode cumulated reward during training for different numbers of agents

TABLE II
EVALUATION RESULTS IN SIMULATION

| Metrics | Values | | | |
|---|---|---|---|---|
| *Num agents* | 2 Trackers | | 3 Trackers | |
| *Env* | *Training* | *Bullet* | *Training* | *Bullet* |
| Mean angle (°) | 91.14 | 102.0 | 96.56 | 85.44 |
| Std dev (°) | 9.15 | 9.98 | 13.64 | 26.34 |
| Crash rate (%) | 12.26 | 22.0 | 6.90 | 14.34 |

first evaluated them at convergence against the same simplistic simulation environment used for the training. Afterward, the models were evaluated in a more realistic environment, named *dronesim* [26], that uses the Bullet Physics engine [27]. This time, the dynamic model of the drones is simulated, and the drones are guided using realistic command laws. As often in drone simulators, *dronesim* simulator has high-level guidance controllers that accept reference positions as input. Recall that the trained policy in the simplistic environment computes, as a control action, a position increment in a 20cm sphere around the current drone position in the inertial frame. The sum of the current drone position and the policy's position increment can then be used as a reference position and fed to the controller of the drone's autopilot, at 48Hz.

Each time, the final position of each tracker with respect to the target was measured to evaluate how they surrounded the target to optimally triangulate its position. We didn't concern ourselves with the distance to the target in this evaluation, as in real-life applications this optimal distance is a function of the field of view and camera intrinsic parameters, which are not modeled in this study. The remaining relevant metric is the angle determined by the lines connecting the agents' positions to the target position. We averaged the angles between agents and over 1000 iterations and also measured its standard deviation. We also kept track of the number of times at least one tracker crashed. The results are presented in Table II.

The uncertainty evaluations presented in Fig. 2 give a baseline of what the optimal formation around the target is. For two drones, the obvious best position to triangulate the target is with both drones at 90° from each other. We can see in Table II, that in the 2-tracker case, the obtained trained policy performs fairly well, reaching the optimal angular position in most of the cases. This validates experimentally our choice of reward function in the 2-tracker case. Results obtained with the 3-tracker case are presented in Table II.

Fig. 4 presents multiple trajectories obtained with the *dronesim* simulation. In the top two figures, we can observe how the trackers circle a fixed target and hover at fixed positions at a safe distance from the target. The position commands outputted by the RL policy network are plotted, and we can see a close match between the position commands and the final drones' trajectories. The inference time of the policy network was short enough to allow controlling the drones in a closed loop at a high frequency (48Hz). Finally, even

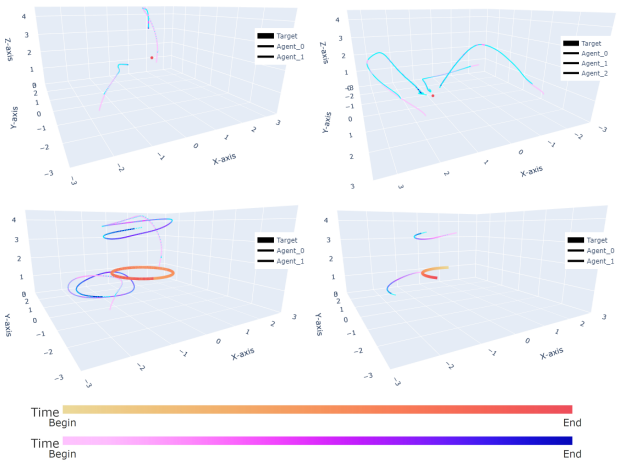


Fig. 4. Trajectories obtained during simulated flights using the Bullet Physics Engine. The trajectories of the drones are plotted in gradients of blue, and the position commands from the RL policy are in cyan.

though it was not trained for, we evaluated our policy against a moving target. The bottom two figures present a 2-tracker case with a target moving in a circle at the same maximum speed as the trackers (2m/s). We can observe that despite being trained with fixed targets, the resulting policy can adapt to slowly moving targets. This outcome is due to the Markovian properties inherent in reinforcement learning, as the control actions are computed based on the current state only and not on past data. However, since the policies were not trained with a moving target, the trackers did not learn to anticipate the movements of the target, leading to a delay in the trackers' response.

## VI. Test with Real-Flights

The control policy for the trackers is solely learned from the interactions inside the simple simulator. To illustrate the behavior of the learned policy in real-life scenarios, we have deployed the 2-tracker and 3-tracker fixed target observation scenarios into real-life demonstrations.

The experiments took place at ENAC's micro indoor flight arena named *crazyTown*, shown Fig. 5. The flight space dimenstions are $L \times W \times H = 3 \times 3 \times 2.5\,\text{m}$ and is equipped with *LightHouse* positioning system from Bitcraze AB. Unlike traditional motion capture systems, which calculate the position and orientation of the tracked objects in a centralized computer, the *LightHouse* system is completely decentralized, i.e. each vehicle calculates its position on-board with the specific hardware module.

The vehicles used during the demonstration are Crazyflie V2.1 with the corresponding Lighthouse modules for the localization. We used Crazyflie's Python library, which has high-level guidance controllers that accept reference positions as input. Therefore, similarly to V with the simulated drones, we feed directly the calculated position command to the Crazyflies, but at only 10Hz.

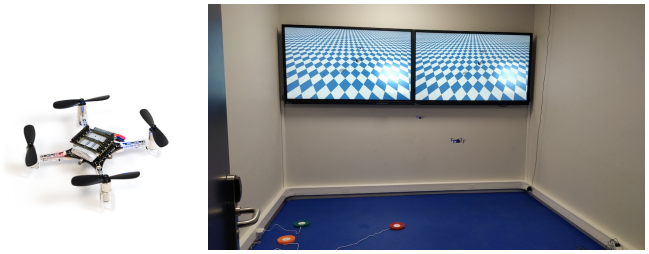

Fig. 5. The Left picture shows the 27g Crazyflie 2.1 quadrotor from Bitcraze AB that is used during this work, and the right picture captures an instance from the real flights with two trackers following a circling target inside crazyTown.

Examples of flight trajectories are shown in Fig. 6 and Fig. 7. One can immediately notice the noisy trajectories obtained during the real flights. This may come from noise in the trackers position estimates, and some effort may be put on smoothing the decisions during the learning phase.

However, the policy successfully reaches and stabilizes over a fixed target point for both 2-tracker and 3-tracker demonstrations, as shown in the bottom row of Fig. 6. As explained in Section V, the policy is also tested on a moving virtual target, which circles horizontally at the center of the crazyTown with a radius of 0.6m and at 0.6m height.

## VII. Conclusion

We have introduced a MARL-based drone guidance approach for N-view triangulation of flying targets using onboard omnidirectional cameras. We employed MAPPO to successfully learn decentralized policies that enable the drones to dynamically adjust their positions to optimally triangulate the target, reducing the uncertainty in the location estimation. We proposed a reward function based on the trace of the covariance matrix of the triangulated position, computed using

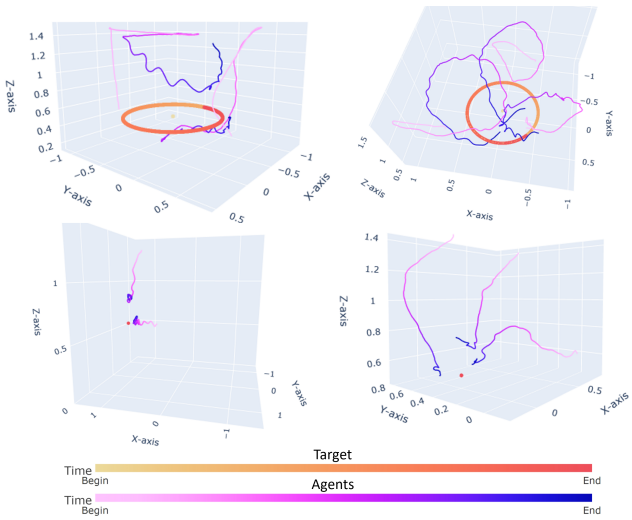


Fig. 6. The top row depicts real flights where the drones are tracking a target moving in circles. The bottom row presents real-flight with a fixed target.

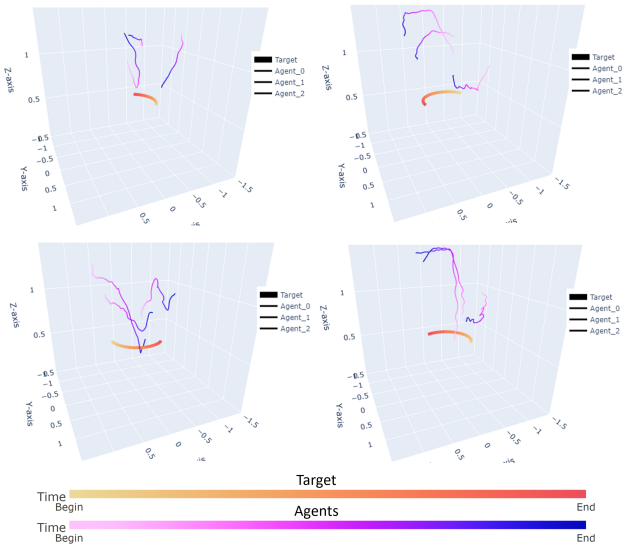


Fig. 7. Example of successive trackers movements when the target is moving in circles.

an analytical model of the uncertainty propagation. A comparison with Monte Carlo approach shows a good agreement with the proposed model. Furthermore, we have evaluated the trained policies in simulation and tested them in real-flight experiments, showing that they can handle different scenarios and transfer to realistic settings. Our approach is a promising step towards leveraging MARL for multi-drone tracking. Besides more thorough experimental evaluation and analyses, in future work, we will evaluate the system behavior with more agents. More importantly, we will no longer assume that the trackers' positions are known to every tracker, but only partially observed by vision. We will also consider the multiple targets case.

## Acknowledgment

The authors would like to thank Arnaud SAMAMA, Frédéric BARBARESCO, Jean-Marc TRIN, and Pierre-Louis CARTIER from Thales LAS, and Florence ALIGNE from Thales RT for their valuable input and contribution to the subject.

## References

[1] C. Yu, *et al.*, "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, *et al.*, "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, Sept. 2006.

[4] R. A. Zitar, *et al.*, "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, June 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, *et al.*, Eds. Berlin, Heidelberg: Springer, 2007, pp. 179–189.

[7] W. Zhou, *et al.*, "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, July 2022.

[8] S. Srigrarom, *et al.*, "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, Sept. 2019.

[11] Z. Xia, *et al.*, "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, *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[14] ——, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[15] T. Rashid, *et al.*, "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning." PMLR, July 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, *et al.*, "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, *et al.*, "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, *et al.*, "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." Sept. 2006.

[23] J. Terry, *et al.*, "Pettingzoo: Gym for multi-agent reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 032–15 043, 2021.

[24] A. Paszke, *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.

[25] T. Haarnoja, *et al.*, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," Tech. Rep., Aug. 2018.

[26] L. T. Fernandez, *et al.*, "Multi-Vehicle Simulation Framework for Heterogeneous Unconventional MAVs," in *IMAV 2023*, AACHEN, Germany, Sept. 2023.

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