2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

# Multi Agent Systems Learn to Safely Move Indoor Environment

Martina Suppa [1,2] Sina M.H. Hajkarim [1] Prathyush P. Menon [1] Antonella Ferrara [2]

*Abstract*— **This letter presents a path-planning algorithm for a fleet of autonomous agents operating in a bounded indoor environment with static and moving obstacles. The proposed algorithm uses a combination of Modified Artificial Potential Field (MAPF) and Reinforcement Learning (RL) to determine safe paths for the agents to their respective goal locations. The proposed approach ensures avoiding collision with the obstacles and among the agents. The better performance of our proposed method, suitable for a real-world operation, is illustrated by comparing it with multiple RL and MAPF concepts. In addition to simulations, we carry out practical experiments using multiple open-source flying development platforms in an indoor VICON lab environment to demonstrate the efficacy of the proposed approach.**

**Keywords:** Path-Planning, Multi-Agent, Artificial Potential Field, Reinforcement Learning

**Supplement Material:** https://youtu.be/GgolyrrGw5Y

## I. Introduction

A plethora of research exists addressing the multi-agent path planning problem using centralised, decentralised and distributed approaches. The prime goal in all these cases is to determine a set of paths for multiple robotic agents such that the paths avoid obstacles in a domain and conflict. Decentralised path planning requires that a sufficiently powerful processor is available onboard for each agent, which may increase the agent's weight and size. This is troublesome for Autonomous Aerial Vehicles (AAVs), for which lightness is often a relevant feature. Yet, the decentralised approach is often the preferred option in outdoor applications. Distributed methods are also becoming popular but need suitable means for exchanging information among the agents and additional onboard power usage.

In contrast, centralised path planning is the ideal solution in applications such as indoor exploration, where a powerful central control unit usually is available. Some indoor applications include automated industrial facility monitoring and management, searching enclosed spaces for survivors or suspects, and inspecting for possible contamination in indoor areas (e.g. Old nuclear store Sellafield, UK). Many of these applications demand regular, repeated robotic multi-agent system usage, yet there is a gap between theory and practice [1]. Indeed, it allows for adopting smaller and more agile agents to perform the exploration task. Moreover, centralised path planning may provide improved agent coordination and a more predictable decision-making process, making it scalable for managing a large number of agents at the same time [2].

In spite of the computational power offered by a centralized implementation, the design of a reliable and highly performing path-planning constitutes a problem that has not yet been completely solved, although numerous solutions with interesting performances are already available. As a matter of fact, many conventional single path-planning algorithms are suffering from trapping in local minima, being computationally expensive, having problems with dynamic obstacles, and lacking of scalability for the MAS. In recent years, novel methods [3]–[6] based on machine learning concepts, like deep Reinforcement Learning (RL), opened the door to new solutions suitable for trajectory/path planning in complex scenarios. Various proposals, specifically address centralised path planning problems in multi-agent systems [3], [4]. For example, in [7], authors propose an Integral RL algorithm based on Artificial Potential Field (APF) to minimize the time and energy of a MAS while reaching their goals, so that the agents avoid collision with themselves in an environment subject to constant or slowly varying unknown disturbances. In this method, after the policy improvement at each step, the full information of all agents' states is needed to update the control input of agents. Thus, it can be considered a centralised system.

In this study, a new path-planning method for a MAS autonomously flying in interior spaces with stationary and moving objects is presented. The method, which combines Soft Actor-Critic (SAC) and MAPF, guides the agents to their intended targets. Specifically, the proposed algorithm plans the path for the MAS so that no collision with static and dynamic obstacles present in the indoor environments occurs. The provided simulation results well illustrate the satisfactory performance of the suggested strategy. Additionally, the experimental results obtained in a multi-agent indoor flight test demonstrate how the proposed approach is actually usable in real-world applications.

## II. Problem Definition

The letter focuses on developing a path-planning algorithm for a fleet of homogeneous AAVs to navigate in a bounded indoor environment to reach user-defined goal locations. The environment contains static and dynamic obstacles. Since GPS is unreliable indoors, the agents are assumed to be equipped with depth cameras to measure local positions and create a map of the agents' surroundings. The agents know their initial positions relative to each other. The proposed algorithm uses this information and the surrounding map to update a collision-free path at each time step and avoid moving obstacles. At each agent level, the algorithm considers other agents as moving obstacles. The path-planning

[1] Martina Suppa, Sina M. H. Hajkarim and Prathyush P. Menon are with The Cooperative Robotics and Autonomous NEtworks (CRANE) lab, Centre for Future Clean Mobility, Faculty of Environment, Science, and Economy, University of Exeter, Exeter, United Kingdom. (email: (ms1317)(mh1004)(P.M.Prathyush)@exeter.ac.uk.)

[2] Martina Suppa and Antonella Ferrara are with Department of Electrical, Computer and Biomedical Engineering, University of Pavia, Pavia, Italy. (email: martina.suppa01@universitadipavia.it, antonella.ferrara@unipv.it.)

algorithm uses a combination of MAPF and RL algorithms to guide the agents to their respective goals, giving them a set of waypoints to follow.

***Notation:*** Given two integers $i$ and $j$ with $j \geq i$, $\mathbb{Z}_i^j$ denotes the set of integer numbers from $i$ to $j$, that is $\mathbb{Z}_i^j \doteq \{i, i+1, \dots, j-1, j\}$. Given two vectors $x, y \in \mathbb{R}^n$; and $\langle x, y \rangle$ denotes their standard inner product. For a vector $x \in \mathbb{R}^n$, $\|x\| \doteq \sqrt{\langle x, x \rangle}$ is its Euclidean norm.

## III. Methodology

This letter considers the path planning of $N$ autonomous agents to explore an unknown compact 2-dimensional (2D) indoor region $\mathcal{X} = [0,\ x_{\max}] \times [0,\ y_{\max}] \subset \mathbb{R}^2$. Let $\boldsymbol{x}_k^i := (x_k^i, y_k^i, \psi_k^i)^T \in \mathbb{R}^3$ and $\boldsymbol{p}_k^i := (x_k^i, y_k^i)^T$ respectively indicate the pose and position vector of the $i^{th}$ agent in the 2D region. The orientation of the $i^{th}$ agent is $\psi_k^i \in [0,\ 2\pi]$, as shown in Figure 1. Two reference frames are used, of which the inertial frame (the fixed Cartesian frame) defines the translation coordinates of the agent in $\mathcal{X}$, and the body frame defines the coordinates of the agent's orientation with respect to the inertial frame as presented in Figure 1.

The agents, equipped with relevant sensors (for e.g. depth cameras [8] and Inertial Measurement Unit (IMU)), are assumed to be homogeneous and satisfy the following discrete process model, with the interval time $T_s$:

$$\boldsymbol{x}_{k+1}^i = \boldsymbol{x}_k^i + \boldsymbol{a}_k^i \ , \ i \in \mathbb{Z}_i^N \ , \tag{1}$$

where $\boldsymbol{a}_k^i := (a_x^i, a_y^i, a_\psi^i)^T \in \mathbb{R}^3$ is the vector of inputs (action vector) which guide $i^{th}$ agent to its next waypoint $(x_{k+1}, y_{k+1})$ at an orientation $(\psi_{k+1})$. Following amplitude constraints are imposed on the inputs:

$$a_{x_{\min}} \leq a_x^i \leq a_{x_{\max}} \ , \ a_{y_{\min}} \leq a_y^i \leq a_{y_{\max}} \ , 0 \leq a_\psi^i \leq 2\pi \ , \tag{2}$$

The path for each agent is generated by (1). Moreover, it is assumed that the motion constraints imposed by the dynamics of the vehicle during the phase of path realisation are addressed by a low-level inner-loop control scheme [9].

The agents are assumed to start their mission from $N$ unique known positions, $\mathcal{P}_0 := \{\ \boldsymbol{p}_0^i \in \mathcal{X} \mid i \in \mathbb{Z}_1^N\}$, and to reach the distinct goal location set $\mathcal{P}_g := \{\ \boldsymbol{p}_g^i \in \mathcal{X} \backslash \mathcal{P}_0 \mid i \in \mathbb{Z}_1^N\}$. Assume that $\mathcal{L}^i$ denotes the set of waypoints that represent the path for the $i^{th}$ agent starting from $\boldsymbol{p}_{k=0}^i \in \mathcal{P}_0$ and leading to the goal point $\boldsymbol{p}_g^i \in \mathcal{P}_g$. From the perspective of computational effort, the region $\mathcal{X}$ is discretised to a grid of cells by a resolution factor $\delta$ yielding a new set $\hat{\mathcal{X}}$. Consider $\mathcal{O} \subset \hat{\mathcal{X}}$ as a subset area of the region which consists of all unknown static obstacles of any size or shape at $k^{th}$ step. Also, $\hat{\mathcal{O}}_k \subseteq \mathcal{O}$ is the subset of observed obstacles in the FoV of $\beta$ at $k^{th}$ step by all agents. For the dynamic obstacles, we assume that they do not have abrupt large changes in velocities or direction. Moreover, a path from the start to the goal location always exists. Given $\hat{\mathcal{O}}_k$, the Euclidean distances between each point of the set and any point $x \in \mathcal{X}$ is defined as $d(x; \hat{\mathcal{O}}_k)$.

**Definition 1.** *Let $A \subset \mathcal{X}$ and $x \in \mathcal{X}$. Suppose $d(x, a)$ is the set of possible distances between point $a \in A$ and $x$ which is constrained below by 0. The largest lower bound of the set of distances from $x$ to a point in $A$ is used to define the least distance of $x$ from set $A$ and can be written as:*

$$d(x; A) = \inf\{d(x, a) \mid a \in A\}.$$

**Assumption 1.** *Suppose $\boldsymbol{p}_{k=0}^i$ and $\boldsymbol{p}_g^i$ where $i \in \mathbb{Z}_1^N$ are placed out of the areas that are occupied by the obstacles ($\boldsymbol{p}_0^i$,$\boldsymbol{p}_g^i$ $\notin$ $\mathcal{O}$). A generated path known as $\mathcal{L}^i$ can be considered as an acceptable path from $\boldsymbol{p}_{k=0}^i$ to $\boldsymbol{p}_g^i$ if and only if the distance of all waypoints in $\mathcal{L}^i$ to obstacles $\hat{\mathcal{O}}_k$ is greater than zero:*

$$\mathcal{L}^i = \{x \in \mathcal{X} \backslash \hat{\mathcal{O}}_k \mid d(x; \hat{\mathcal{O}}_k) > 0\} \tag{3}$$

The design objective is to find the shortest path for the agents from their initial positions to reach their goals while avoiding any static and dynamic obstacles. Existence of healthy communication between the agents and a ground station is assumed. The ground station gets the observations of all the agents and generates a path for each agent, which is based on an optimal policy. As mentioned in the problem formulation, the agents consider each other as moving obstacles and have different observations:

$$\boldsymbol{s}_{k+1}^i = \boldsymbol{s}_k^i + \boldsymbol{w}_k \tag{4}$$

where $\mathbf{w}_k^i \sim \mathcal{N}(0, \sigma^2)$ is the observation noise and $\mathbf{s}_k$ is the observation vector of $i^{th}$ agent at the $k^{th}$ step of decision-making [10]. This vector can be constructed at each step as follows:

$$\boldsymbol{s}_k^i = (\boldsymbol{x}_k^i, \gamma_k^i, \|\boldsymbol{q}_{st,k}^i\|, \|\boldsymbol{q}_{dyn,k}^i\|, \|\boldsymbol{p}_k^i - \boldsymbol{p}_g^i\|)^T \tag{5}$$

where $\boldsymbol{x}_k^i$ is the pose, $\|\boldsymbol{q}_{st,k}^i\|$ and $\|\boldsymbol{q}_{dyn,k}^i\|$ are the norms of gradient of the MAPF vectors for static and moving obstacles. $\boldsymbol{q}_{st,k}^i$ and $\boldsymbol{q}_{dyn,k}^i$ are computed by summing the repulsive and attractive fields and their detail is presented in Section IV-A. $\|\boldsymbol{p}_k^i - \boldsymbol{p}_g^i\|$ is the distance between the $i^{th}$ agent at the $k^{th}$ step and its goal, and the $\gamma_k^i$ an augmented angle defined as follows:

$$\gamma_k^i = \cos^{-1} \frac{\|\boldsymbol{p}_k^i - \boldsymbol{p}_{k-1}^i\|^2 + \|\boldsymbol{p}_k^i - \boldsymbol{p}_g^i\|^2 - \|\boldsymbol{p}_{k-1}^i - \boldsymbol{p}_g^i\|^2}{2\|\boldsymbol{p}_k^i - \boldsymbol{p}_{k-1}^i\|\|\boldsymbol{p}_k^i - \boldsymbol{p}_g^i\|} \ . \tag{6}$$

where the geometrical interpretation of this angle is shown in Figure 1, and as can be seen the closer the angle $\gamma_k^i$ to $\pi$

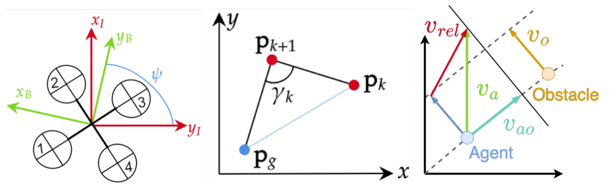

Fig. 1: *[Left] Body (green) and Inertial (red) frames and the orientation angle $\psi$. [Middle] An example of the $\gamma$ angle definition. The blue line is the optimal trajectory. [Right] The geometrical approach of $\boldsymbol{v}_{ao}$. $\boldsymbol{v}_o$ is the obstacle velocity, $\boldsymbol{v}_a$ the agent velocity, $\boldsymbol{v}_{rel}$ the relative velocity, and $\boldsymbol{v}_{ao}$ is the projection of the relative velocity.*

the more optimal the path for agent becomes.

Since the proposed algorithm is centralised, all agents can receive complete information about the other agents' position, velocity, and the subset $\hat{\mathcal{O}}_k$. In order to achieve their objectives, agents determine the shortest path from their starting positions while avoiding both static and dynamic obstacles. To do so, $R(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1})$ as the reward function is needed to evaluate how good or bad an action $\boldsymbol{a}_k^i$ is in the environment at $k^{th}$ step. To reach the best possible choice of action, we can train the agent to achieve the maximum reward value. In this case, the maximum reward value corresponds to the shortest path to the goal points without hitting the static or dynamic obstacles.

In particular, while training, a very high positive value is assigned every time the agent reaches its target, but it gets a negative reward for any other possible situation. During the planning, whenever the path of an agent hits an obstacle or crosses the boundaries, such a path receives large negative rewards. Likewise, whenever the distance between the agent and the goal increases, such a path also gets a negative reward but a relatively lower one. Based on standard RL, the following expected sum of the reward function should be maximized. The optimal policy $\pi_i^*$ that leads the $i^{th}$ agent to that maximum reward can be defined as follow:

$$\pi_i^* = \arg\max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[ \sum_{k=1}^{k_{max}} R(\boldsymbol{s}_k^i, \boldsymbol{a}_k^i, \boldsymbol{s}_{k+1}^i) \right] . \tag{7}$$

In (7), $\pi_i^*$ can be estimated by a neural network ($\phi$) at each step of decision-making. Like in [11], it is assumed that the agents can reach their respective goals in a finite number of steps of the decision-making ($k_{\max}$). The action ($\boldsymbol{a}_k^i$) for determining the next optimal waypoint of $\mathcal{L}^i$ and the orientation of the $i^{th}$ agent, $\boldsymbol{x}_{k+1}^i$ in (1), is determined as follows:

$$\boldsymbol{a}_k^i \sim \pi_{\phi_i}^*(\cdot|\boldsymbol{s}_k^i) , \tag{8}$$

where $\boldsymbol{a}_k^i$ represents a sample from the probability distribution of the policy function, defined by the parameters of neural network $\phi$.

## IV. Path Planning Formulation

The proposed method consists of two key elements: obstacle avoidance and path planning, as depicted in Figure 2. For the obstacle avoidance part, it is assumed that a modified version of APF is used to compute the APF vector described by (13) for static and dynamic obstacles. Then the norm of these vectors is used in the path planning phase as part of observations of the RL algorithm. Detailed explanations of these parts can be found in the following subsections.

### A. Modified Artificial Potential Field

The traditional APF has a repulsive field ($\boldsymbol{U}_{rep}$), which is often related to the static obstacles and an attractive field ($\boldsymbol{U}_{att}$), which is associated with the goal point [12]. These are defined as follows:

$$\boldsymbol{U}_{rep}(\boldsymbol{p}_k) = \begin{cases} \frac{k_{rep}}{2}\left(\frac{1}{\boldsymbol{p}_k - \boldsymbol{p}_o} - \frac{1}{r_o}\right)^2 , & \|\boldsymbol{p}_k - \boldsymbol{p}_o\| \le r_o , \\ 0 & \|\boldsymbol{p}_k - \boldsymbol{p}_o\| > r_o , \end{cases}$$
$$\boldsymbol{U}_{att}(\boldsymbol{p}_k) = k_{att}\,\frac{(\boldsymbol{p}_k - \boldsymbol{p}_g)^2}{2} , \tag{9}$$

where $k_{rep}$ is the repulsive coefficient for static obstacles, $k_{att}$ is the attractive coefficient for the goal point, $\boldsymbol{p}_k$ is the position of the agent at $k^{th}$ step ($(x_k, y_k) \in \mathcal{X}$), $\boldsymbol{p}_o \in \hat{\mathcal{O}}_k$ is the position of each obstacle, $\boldsymbol{p}_g$ is the goal point and $r_o$ is the safety range around the obstacle.

The classical APF can face the presence of local minima [12]. The Black-Hole APF (BHAPF), initially proposed in [13], can overcome this issue. In BHAPF, the traditional attractive field is amended as follows:

$$\boldsymbol{U}_{bh}(\boldsymbol{p}_k) = \begin{cases} -\frac{k_{bh}}{2}\left[r_{bh} - (\boldsymbol{p}_k - \boldsymbol{p}_g)\right]^2 & \|\boldsymbol{p}_k - \boldsymbol{p}_g\| \le r_{bh} , \\ 0 & \|\boldsymbol{p}_k - \boldsymbol{p}_g\| > r_{bh} , \end{cases} \tag{10}$$

where $r_{bh}$ is the radius of the black hole, and $k_{bh}$ is the attractive coefficient of this new black-hole field and it has to be be much more higher than the traditional attractive coefficient $k_{att}$.

Traditional APF considers only the obstacles' positions, not the velocity. Hence, the performance of the conventional APF in the presence of dynamic obstacles could be better. The relative velocity between the agent and the goal is considered in [14] for this purpose. An additional repulsive field term that accounts for the relative velocity is added with the attractive field $\boldsymbol{U}_{att}(\boldsymbol{p}_k)$ as follows:

$$\boldsymbol{U}_{mov}(\boldsymbol{p}_k) = k_{mov}\,\frac{\boldsymbol{v}_{\mathrm{ao}}}{\|\boldsymbol{p}_k - \boldsymbol{p}_{o_{mov}}\|} , \tag{11}$$

where $\boldsymbol{v}_{ao}$ is the projection of the relative velocity between the agent and moving obstacles, starting from the agent in the direction of the obstacle (see Figure 1), and $k_{mov}$ is the repulsive coefficient for moving obstacles. The total fields related to static and dynamic obstacles are computed separately as the summation of the attractive and repulsive

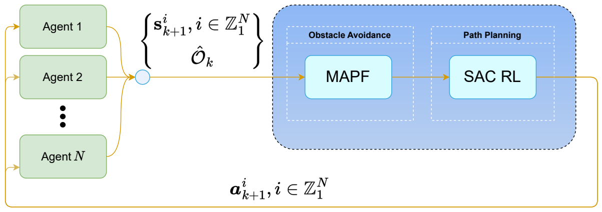
Agent 1
Agent 2
Agent $N$
$\left\{ \begin{matrix} \mathbf{s}_{k+1}^i, i \in \mathbb{Z}_1^N \\ \hat{\mathcal{O}}_k \end{matrix} \right\}$
Obstacle Avoidance
Path Planning
MAPF
SAC RL
$\boldsymbol{a}_{k+1}^i, i \in \mathbb{Z}_1^N$

Fig. 2: *A scheme of the proposed algorithm, that shows where and how the MAPF and the RL algorithms are used.*

fields discussed above:

$$\boldsymbol{U}_{st}(\boldsymbol{p}_k) = \boldsymbol{U}_{att}(\boldsymbol{p}_k) + \boldsymbol{U}_{bh}(\boldsymbol{p}_k) + \sum_{j=1}^{N} \boldsymbol{U}_{rep}(\boldsymbol{p}_k)$$
$$\boldsymbol{U}_{dyn}(\boldsymbol{p}_k) = \boldsymbol{U}_{att}(\boldsymbol{p}_k) + \boldsymbol{U}_{bh}(\boldsymbol{p}_k) + \sum_{l=1}^{M} \boldsymbol{U}_{mov}(\boldsymbol{p}_k) \tag{12}$$

where $N$ is the number of obstacles in $\hat{\mathcal{O}}_k$ and $M$ is the number of moving obstacles. Data from the depth cameras (such as the position and velocity of moving obstacles ($\boldsymbol{p}^k_{o_{mov}}$, $\boldsymbol{v}^k_{mov}$) and update of $\hat{\mathcal{O}}_k$ set), the agents positions $\boldsymbol{p}^i_k$ and orientations $\psi^i_k$ in the region $\mathcal{X}$, are received at each step ($k$) by the central computer. This information along with the Modified Artificial Potential Field (MAPF) concept [13], [14] are used to compute the gradient vectors of MAPF for static ($\boldsymbol{q}^i_{st_k}$) and dynamic ($\boldsymbol{q}^i_{dyn_k}$) obstacles. Then, based on the position of the $i^{th}$ agent, these vectors can be calculated at each step of decision-making ($k$):

$$\boldsymbol{q}^i_{st_k}(\boldsymbol{p}^i_k, \boldsymbol{p}_g, \hat{\mathcal{O}}_k) = -\frac{\nabla \boldsymbol{U}_{st}(\boldsymbol{p}^i_k)}{\|\nabla \boldsymbol{U}_{st}(\boldsymbol{p}^i_k)\|}\ , \tag{13a}$$

$$\boldsymbol{q}^i_{dyn_k}(\boldsymbol{p}^i_k, \boldsymbol{p}_g, \boldsymbol{p}^k_{o_{mov}}, \boldsymbol{v}^i_k, \boldsymbol{v}^k_{mov}) = -\frac{\nabla \boldsymbol{U}_{dyn}(\boldsymbol{p}^i_k)}{\|\nabla \boldsymbol{U}_{dyn}(\boldsymbol{p}^i_k)\|}\ , \tag{13b}$$

where $\boldsymbol{U}_{st}$ and $\boldsymbol{U}_{dyn}$ are the MAPF for static and dynamic obstacles, respectively.

### *B. Path Planning with Reinforcement Learning*

The Soft Actor-Critic (SAC) algorithm [15] is considered for this work since it applies to both continuous state and action space. The SAC method combines some advantages of deep Q-learning [16] and Optimal Policy algorithms [17]. The main difference with other combined methods (DDPG [18] and TD3 [19]) is that in SAC, the entropy, i.e. the randomness, of the choices taken is explicitly considered. The less entropy, the less the decision is taken randomly.

The policy function $\pi$, used by the actor-network, here defined as the optimal version $\pi^*$:

$$\pi^* = \arg\max_\pi \mathbb{E}_{\tau\sim\pi}\left[\sum_{k=1}^{k_{\max}} \xi^k \left(R\left(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1}\right) + \alpha H\left(\pi\left(\cdot \mid \boldsymbol{s}_k\right)\right)\right)\right] \tag{14}$$

where $\tau$ is the trajectory function , $\xi$ is the discount factor, $\alpha > 0$ is the entropy regularisation coefficient and $H$ is the entropy function [15].

A pseudo-code to show the main steps of the training algorithm using SAC is presented in Algorithm 1, where: $e_{\max}$ is the number of training episodes, $\delta$ the resolution factor, $\lambda_\pi$ and $\lambda_Q$ the learning rates of the actor and critic networks, $\theta_1$, $\theta_2$ and $\phi$ are the critic and actor networks parameters and $\tilde{\theta}_1$, $\tilde{\theta}_2$ and $\tilde{\phi}$ their targets. Moreover, $d$ is a binary variable to show the end of an episode, $D$ the memory buffer and $\beta$ the angle to create the FoV for vision simulation, so to update the subset $\hat{\mathcal{O}}_k$ at every step.

The algorithm performs the training loop on $e_{\max}$, which starts with the initialization of the process model in (1) to random values based on the region constraints. Each episode has a limited number of steps $k_{max}$. In each step, an action ($\boldsymbol{a}_k$) is chosen by the policy network $\pi_\phi$ and applied to the environment to compute the next state $\boldsymbol{s}_{k+1}$, the reward obtained $r_k$ and the new observed obstacles set $\hat{\mathcal{O}}_k$. The necessary information for updating network parameters is saved in the memory buffer ($D$). The learning of NN is done through gradient descent methods [15].

At the end of the training routine, i.e. the maximum number of episodes is reached, the output given is the model of the NN related to the parameters $\theta_1$, $\theta_2$, $\phi$, $\tilde{\theta}_1$, $\tilde{\theta}_2$, $\tilde{\phi}$.

**Algorithm 1** Single Agent Training Algorithm with SAC

```
input: $x_{\max}$, $y_{\max}$, $\delta$, $\boldsymbol{p}_g$, $\mathcal{O}$, $k_{\max}$, $e_{\max}$, $\lambda_\pi$, $\lambda_Q$, $\beta$
output: $\theta_1$, $\theta_2$, $\phi$, $\tilde{\theta}_1$, $\tilde{\theta}_2$, $\tilde{\phi}$
for $j \in \mathbb{Z}_1^{e_{max}}$ do
    Initialise: $k = 1$, $\hat{\mathcal{O}}_k = \varnothing$
    $x_k, y_k, \psi_k$ randomly chosen form their possible bounds
    $\boldsymbol{p}_k = [x_k, y_k]^T$
    $d_{g,k} = \|\boldsymbol{p}_k - \boldsymbol{p}_g\|$
    $\gamma_k = 0$, $\|\boldsymbol{q}_{st,k}\| = 0$, $\|\boldsymbol{q}_{dyn,k}\| = 0$
    Initialise: $(\boldsymbol{p}_k, \psi_k) \to \boldsymbol{x}_k$
    while $k < k_{\max}$ do
        $\boldsymbol{a}_k \sim \pi_\phi(\cdot|\boldsymbol{s}_k)$
        $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{a}_k$
        $\hat{\mathcal{O}}_k(\psi_{k+1}, \beta) \leftarrow \boldsymbol{x}_{k+1}(3)$ // updating the map
        $\boldsymbol{p}_{k+1} \leftarrow$ new position of the agent ($\boldsymbol{x}_{k+1}(1,2)$)
        $d_{g,k} = \|\boldsymbol{p}_k - \boldsymbol{p}_g\|$
        $\boldsymbol{s}_{k+1} = [\boldsymbol{p}_{k+1}, \psi_k, \gamma_k, \|\boldsymbol{q}_{st,k}\|, \|\boldsymbol{q}_{dyn,k}\|, d_{g,k}]^T$
        $r_k = R(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1})$ // calculating the reward
        $D \leftarrow D \bigcup (\boldsymbol{s}_k, \boldsymbol{a}_k, r_k, \boldsymbol{s}_{k+1}, d)$ // storing the experience
        if D has enough records then
            for a random batch $B \subset D$ do
                train actor/critic ($\phi$, $\theta_1$, $\theta_2$) and their target networks ($\tilde{\phi}$, $\tilde{\theta}_1$, $\tilde{\theta}_2$) based on [15]
        if $d = True$ then
            $k \leftarrow k + 1$ and break while-loop
```

A parallelisation technique is used to simulate the MAS on the accomplishment of training convergence. The idea is that each agent represents a core of the parallel computation, making the quick exchange of updated information possible. Each core (agent) utilises the same converged model obtained during the training phase (Algorithm 1). In order to have a more realistic simulation, any time that an agent reaches its goal, others can continue their steps towards their respective goals independently without any pause in the simulation.

The simulation routine for all $N$ agents can be synthesised by Algorithm 2, where $l_{\max}$ is the simulation length. Once the pool of the multi-parallel cores is started, the first loop depending on $l_{\max}$ begins, with an inner loop depending on $k_{\max}$. Inside the while-loop, the steps are the same as those seen in Algorithm 1 repeated for simulation of each $i^{th}$ agent. Also in this case, when the *done* signal for the $i^{th}$ agent is equal to $True$, the episode for that agent is finished, so all of its variables are initialised again for a new episode.

**Algorithm 2** Multiple Agent Simulation Algorithm

```
input: $\boldsymbol{p}_g^i,\ x_{\max},\ y_{\max},\ \delta,\ \mathcal{O},\ \beta,\ k_{\max}, l_{\max}$
output: $\theta_1,\ \theta_2,\ \phi,\ \tilde{\theta}_1,\ \tilde{\theta}_2,\ \tilde{\phi}$
Initialise: $\hat{\mathcal{O}}_k = \varnothing,\ \boldsymbol{s}_k^i,\ k = 1$
Start Pool Process
for $l \in \mathbb{Z}_1^{l_{max}}$ on each processor do
    while $k^i < k_{\max}$ do
        $\boldsymbol{a}_k^i \sim \pi_\phi^*(\cdot|\boldsymbol{s}_k^i)$
        $\boldsymbol{x}_{k+1}^i = \boldsymbol{x}_k^i + \boldsymbol{a}_k^i$
        $\hat{\mathcal{O}}_k(\psi_{k+1}^i, \beta) \leftarrow \boldsymbol{x}_{k+1}^i(3)$ // updating the map on the GS
        $\boldsymbol{p}_{k+1}^i \leftarrow$ new position of the $i^{th}$ agent $(\boldsymbol{x}_{k+1}^i(1,2))$
        $d_g^i = \|\boldsymbol{p}_k^i - \boldsymbol{p}_g^i\|$
        $\boldsymbol{s}_{k+1}^i = [\boldsymbol{p}_{k+1}^i, \psi_k^i, \gamma_k^i, \|\boldsymbol{q}_{st,k}^i\|, \|\boldsymbol{q}_{dyn,k}^i\|, d_{g,k}^i]^T$
        if $d^i = True$ then
            Initialise: $\boldsymbol{s}_k^i,\ k^i = 1$
            break while-loop
        $k^i \leftarrow k^i + 1$
```

## V. Results

We consider a $10m \times 5m$ region with four obstacles having a radius of $0.5m$ and height of 0.5 and 0.8 meters. A resolution factor $\delta = 0.1$, for the discretisation of the map, is used. The agents are supposed to start from three different points at $(7, 3)$, $(9, 1.8)$, and $(2.9, 0.6)$ as well as reach the goals at $(1.8, 2.5)$, $(6, 1)$, and $(6.8, 3)$ respectively. The agents flying at a constant speed of $0.3m/s$ and height of $0.4m$ should avoid them.

In the first simulation part, a comparison between the pairs of PPO [20], DDPG, and TD3, along with APF, BHAPF, and MAPF, is presented for the scenario above. This comparison shows that the proposed algorithm based on SAC+MAPF has better results. Then, the optimal policy ($\pi_\phi^*$) trained by SAC+MAPF in Algorithm 1 is used along with Algorithm 2 to plan the path of the agents in the assumed map. The second part uses the optimal policy for practical implementation in indoor experiments.

### A. Simulation Results

In addition to SAC and MAPF, several RL and PF algorithms are used for comparison. Other RL algorithms applied for comparison in our study are PPO, DDPG, and TD3. To do so, the Stable-Baselin3 library is used for the mentioned RL methods [21]. In these methods, the actor-network gets $\boldsymbol{s}_k^i$ introduced in (5) and generates the action ($\boldsymbol{a}_k^i$). Also, the critic network gets the pair of $(\boldsymbol{s}_k^i, \boldsymbol{a}_k^i)$ and generates a Q-value estimation. Furthermore, each RL method is coupled with APF, BHAPF, and MAPF and analysed. The average of the reward value over 50,000 training loops are presented in Table I.

From the results in the Table I, the supremacy of the performance of SAC algorithm with MAPF is evident. Algorithms DDPG and TD3 use deterministic policy and can be trapped in suboptimal solutions, whereas PPO is not well designed for continuous action space. On the contrary, SAC uses a stochastic policy and can manage continuous action spaces.

| | | PF Methods | | |
|---|---|---|---|---|
| RL Methods | | APF | BHAPF | MAPF |
| | PPO | -1103.98 | -985.58 | -1270.40 |
| | DDPG | -390.06 | -387.68 | -356.98 |
| | TD3 | -253.02 | -345.36 | -321.85 |
| | SAC | -298.95 | -216.68 | **-125.77** |

TABLE I: Average reward values for pairs of RL+PF algortihms

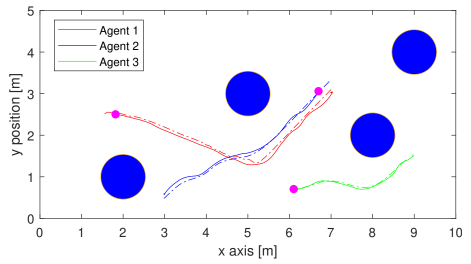

Fig. 4: *Solid lines indicate routes of agents with static blue obstacles, and dashed lines indicate simulation results. Starting points are shown with pink dots.*

The optimal policy based on Algorithm 1 is computed by assuming that the critic networks have 10 neurons as inputs, two hidden layers with the size of 64 neurons using the ReLU activation function, and 1 output neuron. Aside from that, the actor network has 7 neurons as the input layer, 2 hidden layers with the size of 64 neurons using the ReLU activation function, and 3 output neurons. Also, the hyper-parameters of Algorithm 1 are assumed as $\lambda_Q = 1$, $\lambda_\phi = 1$, and $\rho = 0.05$. The FoV angle of the agent is considered as ZED camera ($\beta = 90^o$) [8], and the maximum range of observation is considered as 4 meters. Figure 3 shows how the sensors observe their surrounding environment and how $\hat{\mathcal{O}}_k$ evolves. Moreover, 150 steps are considered the maximum number of steps in each training episode ($k_{\max}$). After 30,000 steps of training ($e_{\max}$), the convergence is reached, and Algorithm 2 is used to plan the path for three agents ($N = 3$) in the environment depicted in Figure 4.

As agents navigate the region, they avoid obstacles and perceive other agents as moving obstacles. The generated

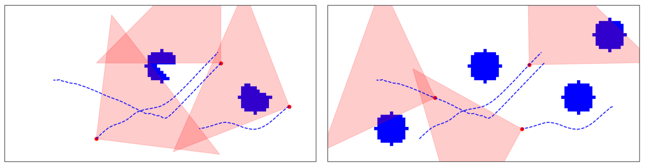

Fig. 3: *[Left] the observed map at the initial step of moving ($\hat{\mathcal{O}}_0$). [Right] The observed map at the $k^{th}$ step. The red areas show the FoV of the sensors at the respective steps.*

paths for three agents are shown in Figure 4, where the pink points are the goals, blue circles are the static obstacles, and the dashed lines are the trajectories followed by the agents at each simulation episode based on Algorithm 2.

## B. Real-World Experimental Results

The agents need complete pose information obtained using depth cameras and a communication system to implement the algorithm. Due to space constraints, small flying robots known as Crazyflies [22] are used. The position and attitude of the Crazyflies are measured using a VICON motion capture system that utilizes multiple synchronized cameras to capture 2D images and calculate 2D positions using triangulation. The lab has 16 Vicon Vero cameras in a $5 \times 15$ meter space, and three Crazyflies are used for the experiment. The Crazyflies communicate with a PC through a Crazyradio PA module, and the PC receives each agent's attitude and position data from the Vicon system.

In the lab, the same region with obstacles is replicated as in the simulation. It is assumed that the agents fly at a constant height (0.4 [m]) and speed. and the vision of the agents for detecting the obstacles are simulated in the ground station computer based on the real-time position of the agents measured by the Vicon system. Also, it is assumed that the sampling time for path planning is 0.5 seconds ($T_s = 0.5$ $[s]$). As can be seen, in Figure 5 and 4, the agents start their mission from initial position approximately similar to the ones considered in the simulation and they avoid static obsolesces as well as themselves and reach their goals. Similar to the simulation results depicted in Figure 4, the paths agent 1 and 2 have an intersection; however, they get close but do not collide due to the MAPF used in the proposed algorithm for dynamic obstacles. This can be seen in the video of the experiment, available to watch on here.

## VI. Conclusion and Future Works

This letter presents a new path-planning algorithm for a multi-agent system combining RL and MAPF methods. Even if the algorithm, by virtue of its centralised nature, was conceived for indoor exploration in the presence of static and moving obstacles, it can also be applied to outdoor environments without modifications. The results of simulations and experiments, obtained by relying on a fleet of small drones named Crazyflies, show the overall performance is excellent.

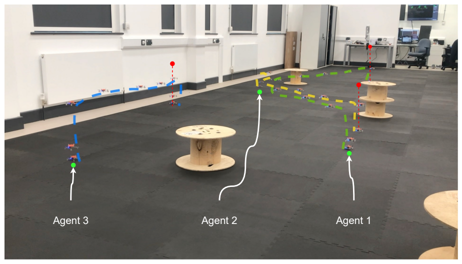

Fig. 5: *A long exposure photo of the experimental results for three agents. Green circles show the starting point, red ones show the goal points, and dashed curves show the path of each agent.*

Future work will be devoted to further improving the method to provide faster convergence and smoother trajectories. The extension of the validity to more complex environments using X500 v2 with ZED2 will also be investigated.

## References

[1] K. Groves, E. Hernandez, A. West, T. Wright, and B. Lennox, "Robotic exploration of an unknown nuclear environment using radiation informed autonomous navigation," *Robotics*, vol. 10, no. 2, p. 78, 2021.
[2] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.
[3] H. Qie, D. Shi, T. Shen, X. Xu, Y. Li, and L. Wang, "Joint optimization of multi-uav target assignment and path planning based on multi-agent reinforcement learning," *IEEE access*, vol. 7, pp. 146 264–146 272, 2019.
[4] D. Le and E. Plaku, "Multi-robot motion planning with dynamics via coordinated sampling-based expansion guided by multi-agent search," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1868–1875, 2019.
[5] M. Rubagotti, B. Sangiovanni, A. Nurbayeva, G. P. Incremona, A. Ferrara, and A. Shintemirov, "Shared control of robot manipulators with obstacle avoidance: A deep reinforcement learning approach," *IEEE Control Systems Magazine*, vol. 43, no. 1, pp. 44–63, 2023.
[6] B. Sangiovanni, G. P. Incremona, M. Piastra, and A. Ferrara, "Self-configuring robot path planning with obstacle avoidance via deep reinforcement learning," *IEEE Control Systems Letters*, vol. 5, no. 2, pp. 397–402, 2020.
[7] C. He, Y. Wan, Y. Gu, and F. L. Lewis, "Integral reinforcement learning-based multi-robot minimum time-energy path planning subject to collision avoidance and unknown environmental disturbances," *IEEE Control Systems Letters*, vol. 5, no. 3, pp. 983–988, 2020.
[8] "Zed 2 - ai stereo camera," https://www.stereolabs.com/zed-2/.
[9] D. E. Rivera, M. Morari, and S. Skogestad, "Internal model control: Pid controller design," *Industrial & engineering chemistry process design and development*, vol. 25, no. 1, pp. 252–265, 1986.
[10] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.
[11] N. Thumiger and M. Deghat, "A multi-agent deep reinforcement learning approach for practical decentralized uav collision avoidance," *IEEE Control Systems Letters*, vol. 6, pp. 2174–2179, 2021.
[12] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2. IEEE, 1985, pp. 500–505.
[13] Q. Yao, Z. Zheng, L. Qi, H. Yuan, X. Guo, M. Zhao, Z. Liu, and T. Yang, "Path planning method with improved artificial potential field—a reinforcement learning perspective," *IEEE Access*, vol. 8, pp. 135 513–135 523, 2020.
[14] X. Fan, Y. Guo, H. Liu, B. Wei, and W. Lyu, "Improved artificial potential field method applied for auv path planning," *Mathematical Problems in Engineering*, vol. 2020, p. 6523158, Apr 2020. [Online]. Available: https://doi.org/10.1155/2020/6523158
[15] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.
[16] 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.
[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015.
[19] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018.
[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
[21] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html
[22] "Crazyflie 2.1," https://www.bitcraze.io/products/crazyflie-2-1/, 2022, [Online; accessed 19-Oct-2022].