Learning to Communicate: A Machine Learning Framework for Heterogeneous Multi-Agent Robotic Systems

Hyung-Jin Yoon*, Huaiyu Chen†, Kehan Long‡, Heling Zhang§,
Aditya Gahlawat¶, Donghwan Lee‖, and Naira Hovakimyan**

We present a machine learning framework for multi-agent systems to learn both the optimal policy for maximizing the rewards and the encoding of the high dimensional visual observation. The encoding is useful for sharing local visual observations with other agents under communication resource constraints. The actor-encoder encodes the raw images and chooses an action based on local observations and messages sent by the other agents. The machine learning agent generates not only an actuator command to the physical device, but also a communication message to the other agents. We formulate a reinforcement learning problem, which extends the action space to consider the communication action as well. The feasibility of the reinforcement learning framework is demonstrated using a 3D simulation environment with two collaborating agents. The environment provides realistic visual observations to be used and shared between the two agents.

I. Introduction

Communication is crucial for the satisfactory performance of multi-agent systems. Different sensors and actuators of the agents can be better used when their individually collected information is shared and collaboratively processed. However, designing communication protocols suitable for multi-agent systems is not a trivial task. The low-cost, high-resolution image sensors can provide a large amount of information, which might be hard to process in real time. Moreover, the transmission of information is constrained by the limited bandwidth of the communication network. It is therefore desirable for the communication protocol to compress the visual data to allow its transmission over the resource-constrained network, provided that the vital for the collaborative mission execution is not lost. In light of these considerations, we propose a machine learning framework for multi-agent systems, where visual information is shared between the agents to accomplish collaborative tasks. The proposed framework employs a reinforcement learning problem formulation. The control policy of each agent dictates not only the local actuator commands, but also the communication messages for transmission to the other agents. The approach is tested in a 3D simulation environment, developed using a game/virtual reality development tool. As an experimental validation, we implement the proposed algorithm for collaborative search in a game environment using a team of a high-altitude and low-altitude aerial vehicles.

It is important to mention that application of the proposed approach to real-world environment for cooperative flight missions, as the one in [1], should be considered only after the *safety* issues of the end-to-end deep reinforcement learning (DRL) are addressed. Training the deep neural network with a large number of parameters requires an even larger number of training samples. Typically, training of DRL takes a few millions of time-steps [2, 3], which is not affordable in real flight missions. An incidence of catastrophic failure is very likely during the transient of this long training period. This fundamental drawback has prevented safety-critical applications of deep learning methods and the variants of those. However, the capability to process high-dimensional sensor signals, such as camera images, have attracted a broad audience to develop vision-based control applications. Some of the evident examples in safety-critical applications are collision avoidance maneuvers that cannot afford trials and errors by DRL. A possible solution to this issue is to restrict the role of DRL in the optimization, while ensuring the safety during the transient by model-based controllers [4]. We note that *safe reinforcement learning* is not in the scope of this paper. Instead, we focus on the

---

*Graduate Student, Mechanical Engineering, University of Illinois at Urbana-Champaign
†Undergraduate Student, Mechanical Engineering, University of Illinois at Urbana-Champaign
‡Undergraduate Student, Department of Mathematics, University of Illinois at Urbana-Champaign
§Undergraduate Student, Electrical & Computer Engineering, University of Illinois at Urbana-Champaign
¶Postdoctoral Research Associate, Mechanical Engineering, University of Illinois at Urbana-Champaign
‖Postdoctoral Research Associate, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign
**Professor, Mechanical Engineering, University of Illinois at Urbana-Champaign AIAA Fellow.

communication between the agents and the centralized policy improvement module (central critic) within a cooperative decision making framework. Our work of exploring the potential of DRL for multi-agent systems is a small step to bring DRL from experimental science to practical engineering.

A. Related Work

Reinforcement learning has been applied to various multi-agent systems [5–8]. Since the agents are co-evolving together, traditional reinforcement learning approaches such as Q-learning [9] do not perform well as they do with single-agent applications. For example, independent reinforcement learning formulations for multi-agent systems, where agents do not exchange any information and treat the other agents as a part of the environment, have displayed poor performance [6]. A possible reason for this could be that the co-evolving of the other agents breaks the Markov assumption of the Markov decision process, which is the typical model for most reinforcement learning techniques.

The framework of *centralized training with decentralized execution* has been successfully applied to multi-agent reinforcement learning problems [7, 8]. Since this framework allows a central critic to access global information generated by the agents, the critic can still rely on the Markov assumption to estimate policy evaluation using the Bellman equation [10]. However, this framework relies on the communication of all observations and actions from all local agents for the centralized training despite formulating distributed control policies. In the case of visual observations, sending high-resolution images from many locally distributed agents over a wireless network can be costly or infeasible.

Information sharing between agents in the context of reinforcement learning has been explored in [11, 12]. The authors of the cited papers demonstrate that the reinforcement learning problem formulation can be useful to find communication policies (protocols). This is accomplished by including performance metrics (to be optimized) in the rewards of the underlying Markov decision process. In a similar fashion, we use the reward function to ensure that the messages capture enough information for the centralized critic to evaluate local policies.

In this paper, we extend the framework of *centralized training with decentralized execution* to include additional optimization of the inter-agent communication. Specifically, we employ the multi-agent deep deterministic policy gradient (MADDPG) algorithm, introduced in [7], and extend the action space to consider communication between the agents and the central critic. Also, we consider autoencoders [13] to compress high dimensional visual observations into low dimensional features. This allows each agent to send local visual observations to the central critic, while not violating the constraints of communication resource usage.

The remainder of the paper is organized as follows. In Section II, the proposed reinforcement learning method that considers communication actions for a multi-agent system is presented. In Section III, we introduce a 3D simulation environment and implement the proposed reinforcement learning method in the simulation environment. In Section IV, the simulation results are analyzed. Section V summarizes the paper.

II. Method

We begin by formulating the Markov decision process (MDP) for multi-agent systems. The MDP for N agents is defined by a set of all possible states $\mathcal{S}$ of the global environment, a set of actions $\mathcal{A}_1, \ldots, \mathcal{A}_N$, and a set of local observations of the environment $O_1, \ldots, O_N$. The ith agent*, $i \in \{1, \ldots, N\}$, chooses its actions from $\mathcal{A}_i$ and receives observations from O_i. Furthermore, given a local observation, the ith agent chooses its action via a deterministic policy $\mu_{\theta^{(i)}}^{(i)} : O_i \to \mathcal{A}_i$, which is parameterized by the real-valued vector $\theta^{(i)}$. For the complete system of N-agents, we define the joint-policy $\mu_\theta : O \to \mathcal{A}$ as $\mu_\theta := (\mu_{\theta^{(1)}}^{(1)}, \mu_{\theta^{(2)}}^{(2)}, \ldots, \mu_{\theta^{(N)}}^{(N)})$, where $\theta := (\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(N)})$, $O := O_1 \times O_2 \times \cdots \times O_N$, and $\mathcal{A} := \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_N$. At any given time $t \in \mathbb{N}$, we denote the action taken by Agent i by $\mathbf{a}_t^{(i)}$. The time-dependent action $\mathbf{a}_t^{(i)}$ is defined as

$$\mathbf{a}_t^{(i)} := (a_{m,t}^{(i)}, a_{c,t}^{(i)}) \in \mathcal{A}_i, \tag{1}$$

where $a_{m,t}^{(i)}$ is the physical action taken by Agent i through its actuators, and $a_{c,t}^{(i)}$ is a virtual action to be communicated to the other agents. As previously described, Agent i uses its local policy $\mu_{\theta^{(i)}}^{(i)}$ to generate actions $\mathbf{a}_t^{(i)}$ at each t, thus, we may write

$$\mathbf{a}_t^{(i)} := (a_{m,t}^{(i)}, a_{c,t}^{(i)}) = \mu_{\theta^{(i)}}^{(i)}(o_t^{(i)}), \quad o_t^{(i)} \in O_i.$$

---

*We use superscript (i) to denote the ith agent.

2

The state-transition model $\mathcal{T}$ determines the probability of the next global state given the current state and the actions taken by all the agents, i.e., $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_N \times \mathcal{S}' \to [0, 1]$. Since we consider the operation of a team of agents, and not individual agents, we consider collaborative rewards. To be precise, we consider the collaborative reward r, which depends on the global states, the actions, and the next state, i.e. $r : \mathcal{S} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_N \times \mathcal{S}' \to \mathbb{R}$. Therefore, the goal is to maximize the expectation of the sum of all future rewards $\sum_{t=0}^{\infty} \gamma^t r_t$, where $\gamma \in (0, 1)$ is a discount factor, and r_t is the reward earned at time step t.
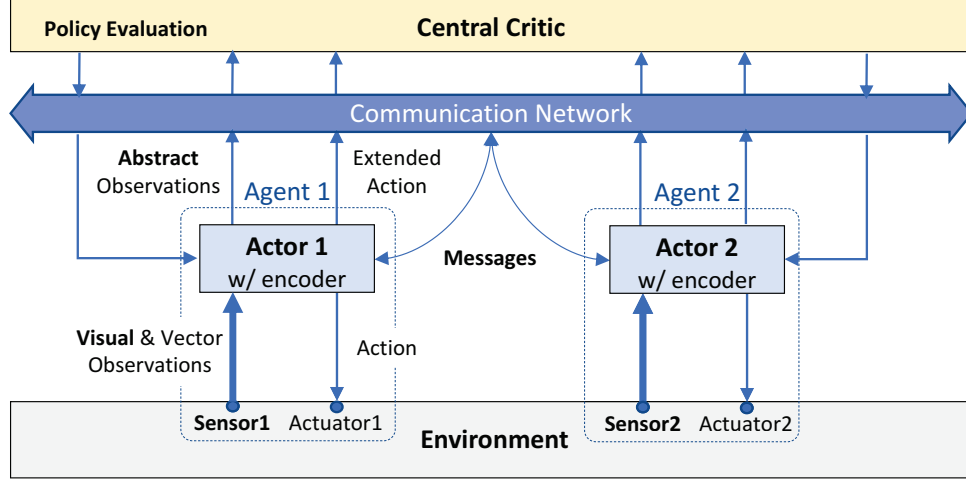

Fig. 1   Overview of decentralized policy (actor) and central critic.

We consider a two-agent system, presented in Figure 1, as a concrete example of the proposed framework. Both Agents 1 and 2 have onboard actors that compute their local control policies $\mu_{\theta^{(i)}}^{(i)}$, $i \in \{1, 2\}$. Additionally, using the communication network, each agent interacts with a centralized critic, which is an algorithm that evaluates the performance (value functions) of the agents. Further details on the actor-critic framework can be found in [14]. Each actor determines the action taken by its agent using the signals from onboard sensors like cameras and IMUs, and the virtual action $a_{c,t}^{(i)}$ of the other agent (see Eqn. (1)). We refer to the virtual actions shared between the agents as messages in Figure 1. In addition to the actions taken by the agents, the central critic requires *all* the observations available to each agent in order to evaluate the performance of the multi-agent system. The sensor measurements, actions, and messages (virtual actions) can be communicated easily. However, the communication of high-resolution raw images recorded by the onboard cameras of each agent creates a considerable network burden and thus presents a significant challenge. In order to mitigate this issue, each agent is also equipped with an autoencoder which transforms the raw camera images into low dimensional features (encoded images). Then, the sensor measurements, messages, actions, and low dimensional features, which we collectively refer to as abstract observations, can be transmitted with ease between the agents and the central critic. This is due to the fact that the encoder compresses the raw images significantly. To summarize, the agents communicate their respective virtual actions with each other while transmitting their abstract observations to the central critic. The critic then communicates the performance evaluation back to each agent.

For the considered multi-agent setup, we need to address the two major components: i) the central critic, and ii) the autoencoders and actors onboard each agent. We refer to the actor and the autoencoder jointly as the actor-encoder.
Central Critic: Reinforcement learning algorithms developed for single agents can be applied to multi-agent systems by using the *centralized training with decentralized execution (CTDE)* framework, see, for e.g. [7, 8]. Similar to the work in [7], we apply the CTDE framework to the deep deterministic policy gradient (DDPG) algorithm [3], which is a variant of the deterministic policy gradient (DPG) [15]. In the DDPG algorithm, the critic estimates the true state-action value function $q^{\mu_\theta}(s, a)$ for the joint policy μ_θ using the following recursive equation:

$$q^{\mu_\theta}(s_t, \mathbf{a}_t) = \mathbb{E}[r(s_t, \mathbf{a}_t) + \gamma q^{\mu_\theta}(s_{t+1}, \mathbf{a}_{t+1})], \tag{2}$$

where the expectation $\mathbb{E}$ is taken with respect to the state-action distribution determined by the policy μ and the underlying MDP, and $\gamma \in (0, 1)$ is the discount factor. The critic cannot directly use (2) to estimate $q^{\mu_\theta}(s, a)$, since it

does not have access to the true global state s_t. Instead, the proposed critic uses the abstract observations transmitted from each agent to estimate the state-action value function. The critic approximates the state-action value function $q^{\mu_\theta}(s, a)$ by $Q(\mathbf{o}, \mathbf{a}; w)$, where w is a parameter, and $\mathbf{a}$ is the global action (actions taken by all the agents), which, using (1), can be expressed as

$$\mathbf{a} = (\mathbf{a}^{(1)}, \mathbf{a}^{(2)}), \quad \mathbf{a}^{(i)} = (a_m^{(i)}, a_c^{(i)}), \quad i \in \{1, 2\}.$$

Note that we have dropped the time-dependence of $\mathbf{a}$ on t for brevity. Additionally, $\mathbf{o}$ is the concatenation of all the observations communicated by all the agents to the critic. Therefore, $\mathbf{o}$ contains the sensor measurements collected by each agent, the messages containing the virtual actions transmitted between the agents, and the abstract visual observations (encoded images) compressed by the autoencoders onboard each agent. The parameter w in $Q(\mathbf{o}, \mathbf{a}; w)$ is estimated by minimizing the temporal difference (TD) error [16]. The loss function $l(w)$ to be minimized for the estimation of $Q(\mathbf{o}, \mathbf{a}; w)$ is

$$l(w) = \frac{1}{M} \sum_{m=1}^{M} (Q(\mathbf{o}_m, \mathbf{a}_m; w) - y_m)^2, \quad y_m = r_m + \gamma Q(\mathbf{o}'_m, \mu_\theta(\mathbf{o}'_m); w), \tag{3}$$

where M transitions are uniform random samples from a replay buffer $\mathcal{D}_{\text{replay}}$, which stores the recently observed L transitions, i.e.:

$$\mathcal{D}_{\text{replay}} := ((\mathbf{o}_{t-L}, \mathbf{a}_{t-L}, r_{t-L}, \mathbf{o}'_{t-L}), \ldots, (\mathbf{o}_{t-1}, \mathbf{a}_{t-1}, r_{t-1}, \mathbf{o}'_{t-1})).$$

Here, each transition $(\mathbf{o}, \mathbf{a}, r, \mathbf{o}')$ consists of the concatenated abstract observation $\mathbf{o}$, the global action by all agents $\mathbf{a}$, collaborative reward r, and the next observation $\mathbf{o}'$. Our use of the replay buffer $\mathcal{D}_{\text{replay}}$ is motivated by its successful application to *Deep Reinforcement Learning* [2, 3]. However, the size of replay buffers used in [2, 3] is million samples. For the type of multi-agent systems that we consider, which use high-resolution images, the storage of such replay buffers would be infeasible[†]. This is where our novel approach of encoding raw images is advantageous, since the compressed images require significantly less memory resource to store. Thus, our approach of encoding high-resolution images not only enables low-burden communication, but also the storage of replay buffers, which is crucial for the operation of the central critic.

Autoencoders and Actors: We now explain the autoencoders and actors onboard each agent. As aforementioned, the central critic requires the concatenated observation $\mathbf{o}$, a major component of which is the set of compressed images recorded by each agent's onboard camera. This compression is performed by autoencoders [13] onboard each agent. The efficacy of using autoencoders in reinforcement learning was empirically demonstrated in [17]. In the proposed approach, the autoencoders are learned on-line. The autoencoder consists of an encoder and a decoder. To ensure that the encoder does not remove the principle components from the raw image, an image is reconstructed by the decoder from the encoded image and compared to the original raw image. This comparison then guides the learning of the encoder. We denote the encoder onboard Agent i by $f(\cdot; \xi_e^{(i)})$, where $\xi_e^{(i)}$ are the parameters to be learned. Given a set of images $\mathcal{I}$, the encoder compresses the images to a pre-specified dimension D, i.e. $f(\cdot; \xi_e^{(i)}) : \mathcal{I} \to \mathbb{R}^D$. The decoder, denoted by $g(\cdot; \xi_d^{(i)}) : \mathbb{R}^D \to \mathcal{I}$, performs the reconstruction. Here, $\xi_d^{(i)}$ are the parameters of the decoder. The encoder and decoder parameters, $\xi_e^{(i)}$ and $\xi_d^{(i)}$, respectively, are learned by recursively minimizing the loss:

$$l(\xi_e^{(i)}, \xi_d^{(i)}) = \frac{1}{P} \sum_{p=1}^{P} d(v_p^{(i)}, g(f(v_p^{(i)}; \xi_e^{(i)}); \xi_d^{(i)})), \tag{4}$$

where $d(\cdot, \cdot)$ is a difference metric[‡] between two images. Additionally, $\{v_p^{(i)}\}_{p=1}^{P}$ is the mini-batch sample from image data buffer $\mathcal{D}_{\text{image}}^{(i)}$ stored locally on the ith agent and contains the recently observed camera images taken by the agent. Upon completion of the compression, each agent communicates its encoded image $f(v_p^{(i)}; \xi_e^{(i)})$ to the central critic, which is then used to construct the concatenated observation $\mathbf{o}$.

Once the central critic computes the approximate state-action value function $Q(\mathbf{o}, \mathbf{a}; w)$ using (3), it communicates this value and the associated mini-batch back to each agent. The onboard actors use these to improve their local policies $\mu_{\theta^{(i)}}^{(i)}$, $i \in \{1, 2\}$. As previously mentioned, the actors use deep deterministic policy gradient (DDPG) algorithm [3] to

---

[†]A million color images with 200×200 resolution would take 480 Gigabyte of memory to store.

[‡]Mean squared error is an example of metric between images.

improve the policies of their agents by estimating the policy gradient based on the communicated mini-batch as

$$\nabla_{\theta^{(i)}} J \approx \frac{1}{P} \sum_{p=1}^{P} \nabla_{\mathbf{a^{(i)}}} Q(\mathbf{o}, \mathbf{a^{(i)}}; w)\big|_{\mathbf{o}=\mathbf{o}_P, \mathbf{a^{(i)}}=\mu_{\theta^{(i)}}^{(i)}(\mathbf{o}_P)} \nabla_{\theta^{(i)}} \mu_{\theta^{(i)}}^{(i)}(\mathbf{o})\big|_{\mathbf{o}_P}. \tag{5}$$

Here, $\mathbf{o}_P$ is the concatenated observation contained in the communicated mini-batch. Then, each agent's local policy is improved by recursively adding the policy gradient $\nabla_{\theta^{(i)}} J$ to the current policy $\mu_{\theta^{(i)}}^{(i)}$ with a decaying gain (step-size) in order to improve the expected sum of the future rewards. Finally, we would like to mention the process by which each agent optimizes the messages (virtual actions $a_{c,t}^{(i)}$ in (1)) to communicate with other agents. This optimization is automatically handled by the reinforcement learning framework via the inclusion of the global action (action taken by all the agents) in the collaborative reward function r_t, which we previously defined to be

$$r_t = r(s_t, \mathbf{a}_t),$$

and where s_t is the global state. Since, by (1),

$$\mathbf{a}_t = (\mathbf{a}_t^{(1)}, \mathbf{a}_t^{(2)}), \quad \text{and} \quad \mathbf{a}_t^{(i)} := (a_{m,t}^{(i)}, a_{c,t}^{(i)}),$$

we see that the message/virtual actions are encoded in the collaborative reward r_t.

In conclusion, the proposed reinforcement learning algorithm has 3 parameter updates: (1) autoencoder, (2) critic, and (3) policy update. Since the algorithm has multiple parameters being updated simultaneously, we require that each parameter update uses different time-scales for the associated step-sizes. This is because, as explained in [14], the policy needs to be updated slowly so that the critic can track the changes of the Markov chain (Controlled MDP). Similarly, we employ the approach of the multiple-time-scale algorithm [18]. The autoencoder is updated at the slowest time-scale, and the critic is updated at the fastest time-scale. This allows the policy evaluation performed by the critic to track the slowly varying changes of the Markov chain.

III. Experiments

We develop a 3D simulation environment to test and validate the proposed algorithms for the two-agent system illustrated in Figure 1. The 3D environment simulates an urban scene, in which two unmanned aerial vehicles (UAVs) operate collaboratively to identify and approach a person of interest. The two UAVs represent the agents in our framework. The first agent flies at a low-altitude and is equipped with a front-view camera and sensors measuring its position and velocity. The second agent operates at a higher altitude and is equipped with a down-facing camera. The first agent's front-view camera images are not sufficient to allow it to search and move towards the target person. Therefore, it must also rely on the down-facing camera images taken by the second agent. Figure 2 illustrates the positions of the agents in the simulation environment.

We notice that similar problems have been considered in cooperative path-following tasks in [1, 19], where the two heterogeneous UAVs had to execute a cooperative road search mission. Here we focus on decision making for strategy development through sharing of information, while in [1, 19] the focus was on cooperative path following, where the paths were generated apriori. The framework of this paper can lead to (near) real-time optimal navigation algorithms that a cooperative path following framework can use efficiently.

A. The 3D Simulation Environment

We now explain the development of the simulator and the setup of the experiments. We use *Unity 3D* [20], a game development editor, to construct the environment. Using the physics engine of the editor, we can conveniently model the rigid-body dynamics of the agents. The physics engine updates the state of the rigid body dynamics at 60 Hz in the game time[§]. A proportional–integral–derivative (PID) controller generates the moment and the force exerted on the agent to track the velocity commands given by the onboard actors. The first agent can execute forward, backward, and yaw movements, while the high-altitude second agent hovers at a fixed position. The proposed algorithm interacts with the game environment at 10 Hz in game time. Furthermore, the game engine checks for collisions of the agents with all objects in the scene such as buildings, cars, and traffic signs. Such a collision results in a negative reward for the respective agent, thus encouraging a collision-free safe behavior. Additionally, an important feature of our simulator is the realistic visualization of the 3D environment. This allows us to simulate the capture of high-resolution images by cameras onboard each agent.

[§]The game engine keeps track of the states of the environment with its own timestamps that we refer to as game time.
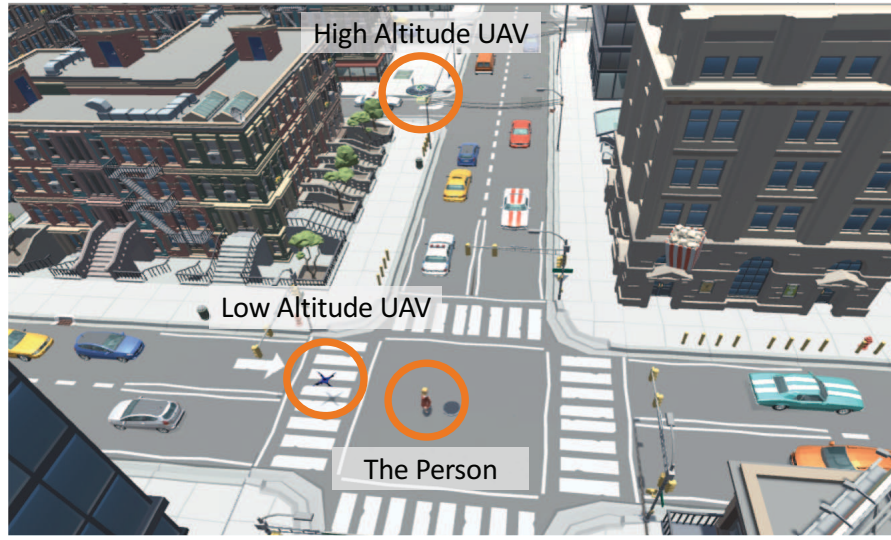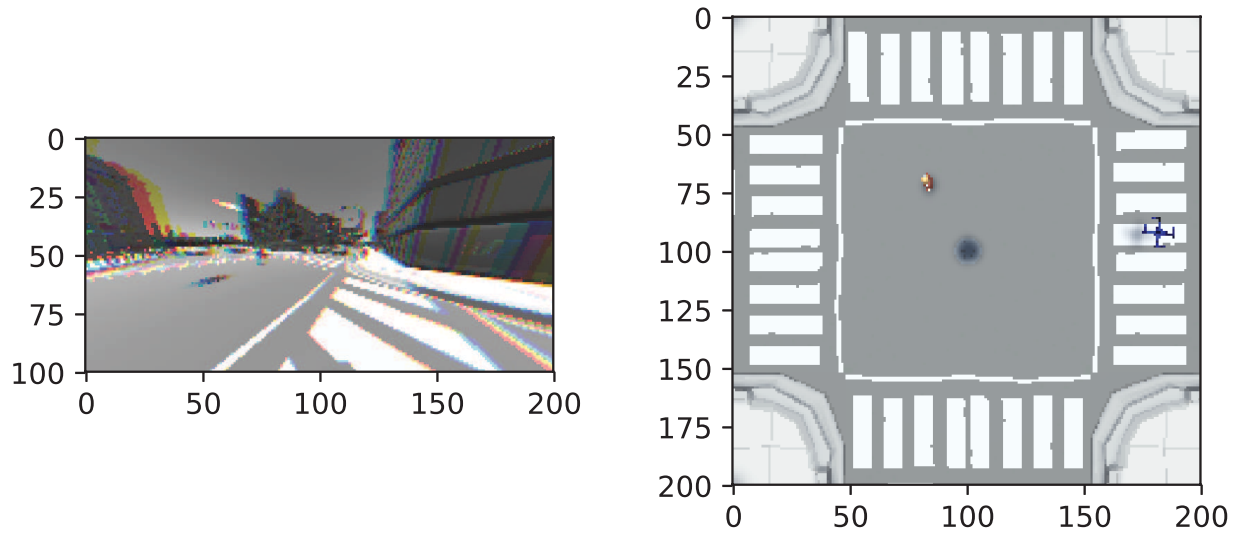
Fig. 2 **The 3D Environment with two agents: the first agent (low-altitude UAV)'s task is to reach the person, and the second agent (high-altitude UAV) sends messages to the first agent to share the top-view. An illustrative video of the environment can be found [here].**

(a) Stacked 3 frames of front view by the low-altitude UAV.   **(b) Color image of top view by the high-altitude UAV.**

Fig. 3 **The high dimensional visual observations: (1) The stacked frames image (size: $3 \times 100 \times 200$) contains the images taken in the recent three frames; (2) The color image (size: $3 \times 200 \times 200$) has information of the person's location.**

6

To train the various components of the proposed algorithm (central critic, onboard actors, and autoencoders), we run episodes of the collaborative search task in the simulator. As mentioned before, the two-agent team is tasked with finding and approaching the target person. The first agent (low-altitude UAV) and the second agent (high-altitude UAV) must communicate with each other and the central critic so that they can efficiently use each other's different observations to accomplish the task. Each episode is designed as follows:

1) **Episode initialization:** Both the target person and the first agent start at uniformly random positions and orientations. These initial positions always lie on the intersection in Figure 2. The second agent is initialized at a fixed position overlooking the intersection.

2) **Episode termination:** An episode is terminated, when the two-agent team either succeeds or fails. The team succeeds, if the first agent gets within 3 meters of the target person. Conversely, the team fails, if the first agent either gets more than 30 meters away from the target person or collides with an object.

3) **Reward design:** In order to train the central critic and the agents' actors and autoencoders, we design the rewards as follows: the successful completion of the task leads to a reward of 100, while a failure leads to a reward of -100. Furthermore, the central critic receives a reward of 1 for every time step, over which the first agent moves closer to the target person. Finally, we impose a reward of -0.05 per time step as a time penalty.

During each episode, the two-agent team operates using the method described in Section II. Both agents communicate messages (virtual actions) between each other. Additionally, each agent uses its autoencoder to compress its camera images and communicates the abstract observations (sensor measurements, compressed images, and actions) to the central critic. The central critic uses these inputs from the agents to approximate the state-action values and communicates it back to each agent along with the associated mini-batches. Each agent's actor then uses this information to improve its policy and take actions. This process repeats at each time step. The design of rewards ensures that over multiple episodes the two-agent team learns to collaborate and ensure that the task is completed, i.e., the first agent successfully approaches the target person.

B. Implementation of Multi-Agent Deep Deterministic Policy Gradient (MADDPG)

Algorithm 1 describes the proposed multi-agent reinforcement learning framework. Compared to the MADDPG algorithm [7], the proposed framework has an additional component, an autoencoder that encodes high dimensional images into low dimensional features to be used by the central critic. As mentioned previously, with this extension, we are able to keep the memory resource usage for replay buffer within a reasonable range.

We apply the (MADDPG) [7] algorithm to the two agent system explained in Section III.A. Each agent has a different set of sensors and actuators. The first agent senses its 12-dimensional rigid body state, the stacked frames of the front-view camera mounted on it as shown in Figure 3a, and receives messages from the second agent. The first agent is actuated by a PID controller, which receives forward, backward, and yaw commands from its onboard actor. The second agent observes the color images of the top view as shown in Figure 3b and the messages sent by the other agent. The second agent does not have any actuators. The second agent's action contains only communication messages to inform the first agent of the location of the person inferred from the top view images. The communication messages sent at the current time step become available to the other agents in the next time step. The communication messages contain the virtual actions generated by the onboard actors in Figure 4.

As mentioned earlier, the performance of the agents is evaluated by the central critic. The central critic uses the concatenated observation **o** formulated using the messages communicated by each agent. The agents communicate their encoded images, sensor measurements, and the messages received from other agents to the central critic. Therefore, for our two-agent system, the concatenated observation available to the central critic is

$$\mathbf{o} := (o^{(1)}, o^{(2)}, f(v^{(1)}; \xi_e^{(1)}), f(v^{(2)}; \xi_e^{(2)}), m^{(1)}, m^{(2)}),$$

where $o^{(i)}$, $i \in \{1, 2\}$, denotes the ith agent's sensor measurements, $f(v^{(i)}; \xi_e^{(i)})$ denotes the encoded image from the ith agent, $m^{(i)}$ denotes the messages received by the ith agent, $v^{(1)}$ denotes the first agent's stacked frame image in Figure 3a, and $v^{(2)}$ denotes the second agent's color image, as in Figure 3b.

The first agent's actor selects an action $a_m^{(1)}$ to *actuate* the agent and an action $a_c^{(1)}$ to *communicate* based on the local observations and received messages. At the next time-step, the communicated messages update the received messages, i.e. $m^{(2)} \leftarrow a_c^{(1)}$ and $m^{(1)} \leftarrow a_c^{(2)}$. As defined in (1), the extended action by the first agent is
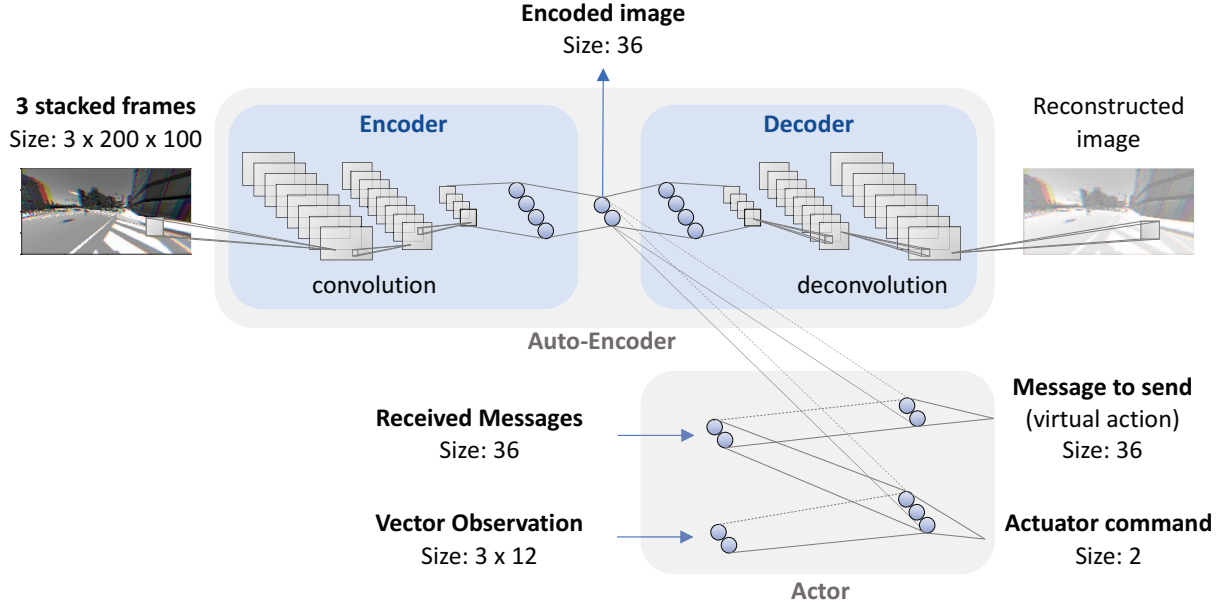
$$\mathbf{a}^{(1)} := (a_m^{(1)}, a_c^{(1)}),$$

Fig. 4 The computation graph of the actor-encoder network for the first agent. The convolution layers are shared by the autoencoder and the actor. The vector observation is the stacked 3 frames of 12-dimensional state of the rigid body. Batch normalization [21] is used between layers to keep the signals within an appropriate scale. The second agent uses the same neural network without generating actuator command.

where we have dropped the temporal dependence. We also define the extended abstract observation for the first agent as

$$\mathbf{o}^{(1)} := (o^{(1)},\ f(v^{(1)};\xi_e^{(1)}),\ m^{(1)}).$$

Similarly, we define the extended action $\mathbf{a}^{(2)} := (a_m^{(2)}, a_c^{(2)})$ and observation $\mathbf{o}^{(2)} := (o^{(2)}, f(v^{(2)};\xi_e^{(2)}), m^{(2)})$ for the second agent. The extended abstract observation $\mathbf{o} = (\mathbf{o}^{(1)}, \mathbf{o}^{(2)})$ and the extended action $(\mathbf{a}^{(1)}, \mathbf{a}^{(2)})$ are the inputs to the central critic network, as illustrated in Figure 5, and are used to calculate the state-action value.
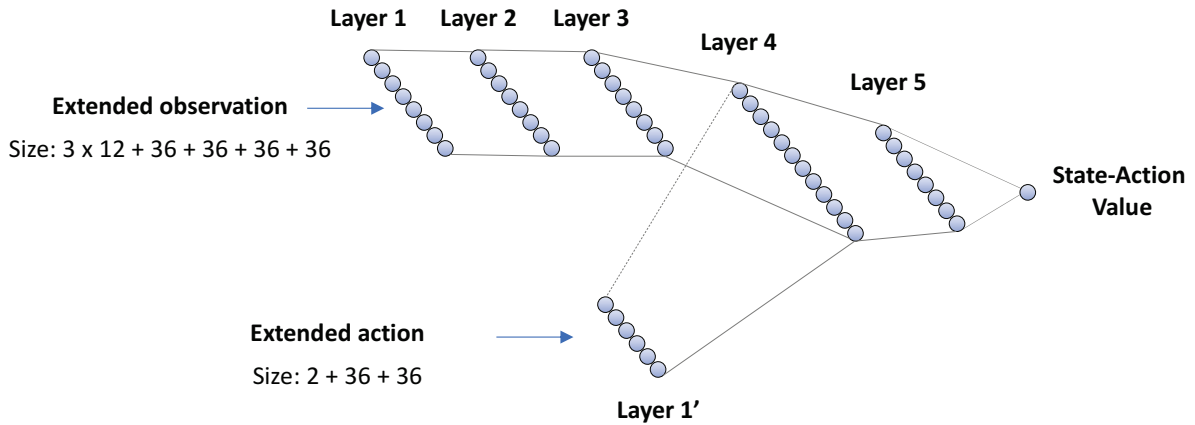

Fig. 5 The computation graph of the critic-network that uses the extended observation, which is the concatenation of encoded images, the received messages and the vector observations.

Algorithm 1 The Extended MADDPG for the Multi-Agent RL illustrated in Figure 2.

1: **for** episode=1 to M **do**
2:     Choose the step size function: $\epsilon_1(t)$, $\epsilon_2(t)$, $\epsilon_3(t)$.
3:     Initialize parameters: $\theta^{(1)}, \theta^{(2)}, \theta^{(1)}_{\text{target}}, \theta^{(2)}_{\text{target}}, w, w_{\text{target}}, \xi_e^{(1)}, \xi_d^{(1)}, \xi_e^{(2)}, \xi_d^{(2)}$.
4:     Initialize a random process $\mathcal{N}_t$, for actuator action exploration.
5:     Central critic: Receive initial abstract observation $\mathbf{o} \leftarrow (o^{(1)}, o^{(2)}, f(v^{(1)}; \xi_e^{(1)}), f(v^{(2)}; \xi_e^{(2)}), m^{(1)}, m^{(2)})$ using encoders $f(\cdot; \xi_e^{(i)})$.
6:     **for** t=1 to max-episode-length **do**
7:         **for** agent i= 1,2 **do**
8:             Select the extended action $\mathbf{a}^{(i)} = \mu^{(i)}_{\theta^{(i)}}(\mathbf{o}^{(i)})$.
9:             Execute exploratory action i.e. $a_m^{(i)} + \mathcal{N}_t$ on the environment and receive reward r.
10:            Collect the new sensor observation $o^{(i)}, v^{(i)}$ on the state of the environment.
11:            Locally store the raw images ($v^{(i)}$) in the image buffer $\mathcal{D}^{(i)}_{\text{image}}$.
12:            **Local encoder update**:
13:            Sample a random mini-batch $\mathcal{M}^{(i)}_{\text{image}}$ from $\mathcal{D}^{(i)}_{\text{image}}$.
14:            Update the actor-encoders using gradient descent with the step-size $\epsilon_3(t)$ to minimize the loss,

$$l(\xi_e^{(i)}, \xi_d^{(i)}) = \frac{1}{M} \sum_{m=1}^{M} d(v_m, g(f(v_m; \xi_e^{(i)}); \xi_d^{(i)})), \quad v_m \in \mathcal{M}^{(i)}_{\text{image}}.$$

15:            **Local actor update**:
16:            Receive the sampled mini-batch $\mathcal{M}$ from the central critic.
17:            Update the i^{st} actor with the step-size $\epsilon_2(t)$ using the sampled policy gradient:

$$\nabla_{\theta^{(i)}} J \approx \frac{1}{M} \sum_{m=1}^{M} \nabla_{\mathbf{a}^{(i)}} Q(\mathbf{o}_m, \mathbf{a}^{(i)}, \mathbf{a}_m^{(i-)}; w)|_{\mathbf{a}^{(i)} = \mu^{(i)}(\mathbf{o}_m)} \nabla_{\theta^{(i)}} \mu^{(i)}_{\theta^{(i)}}(\mathbf{o}_m),$$

18:            where $i- = 2$ if $i = 1$ and vice versa.
19:            Update the parameters of the target networks $\theta^{(i)}_{\text{target}} \leftarrow \tau\theta^{(i)} + (1 - \tau)\theta^{(i)}_{\text{target}}$.
20:            **Communication**:
21:            $m^{(i-)} \leftarrow a_c^{(i)}$ and send it to $(i-)$th agent. Receive message $m^{(i)} \leftarrow a_c^{(i-)}$ from $(i-)$th agent.
22:         **end for**
23:         **for** Central Critic **do**
24:            Receive $(o^{(1)}, o^{(2)}, f(v^{(1)}; \xi_e^{(1)}), f(v^{(2)}; \xi_e^{(2)}), m^{(1)}, m^{(2)})$ from the local agents.
25:            $\mathbf{o}' \rightarrow (o^{(1)}, o^{(2)}, f(v^{(1)}; \xi_e^{(1)}), f(v^{(2)}; \xi_e^{(2)}), m^{(1)}, m^{(2)})$.
26:            Store the transition $(\mathbf{o}, \mathbf{a}^{(1)}, \mathbf{a}^{(2)}, r, \mathbf{o}')$ in replay buffer $\mathcal{D}_{\text{replay}}$.
27:            Sample a random mini-batch $\mathcal{M}$ from $\mathcal{D}_{\text{replay}}$

$$\mathcal{M} = ((\mathbf{o}_1, \mathbf{a}_1^{(1)}, \mathbf{a}_1^{(2)}, r_1, \mathbf{o}'_1), \dots, (\mathbf{o}_M, \mathbf{a}_M^{(1)}, \mathbf{a}_M^{(2)}, r_M, \mathbf{o}'_M)).$$

28:            Send $\mathcal{M}$ and w to all local agents.
29:            Set $y_m = r_m + \gamma Q(\mathbf{o}'_m, \mathbf{a}^{(1)}, \mathbf{a}^{(2)}; w_{\text{target}})|_{\mathbf{a}^{(1)} = \mu^{(1)}(\mathbf{o}'_m; \theta^{(1)}_{\text{target}}), \mathbf{a}^{(2)} = \mu^{(2)}(\mathbf{o}'_m; \theta^{(2)}_{\text{target}})}$.
30:            Update the central critic parameter w with the step-size $\epsilon_1(t)$ by minimizing the loss,

$$l(w) = \frac{1}{M} \sum_{m=1}^{M} (Q(\mathbf{o}_m, \mathbf{a}_m^{(1)}, \mathbf{a}_m^{(2)}; w) - y_m)^2.$$

31:            Update the parameters of the target networks $w_{\text{target}} \leftarrow \tau w + (1 - \tau)w_{\text{target}}$.
32:         **end for**
33:         $\mathbf{o} \leftarrow \mathbf{o}'$.
34:     **end for**
35: **end for**

IV. Simulation Results

We ran one million simulation steps to verify the feasibility of the proposed algorithm. The reinforcement learning algorithm improves the mean rewards per episode, as shown in Figure 6a. Also, the rate of the successful episodes has been improved, as shown in Figure 7a. However, the success indicator values in Figure 7a show that only 1 out of 4 trials is successful. There are a few possible reasons why the performance of the trained policy is not satisfactory. First, the incomplete state observation is a fundamental challenge for the Markov assumption based reinforcement algorithms. Our proposed algorithm assumes that the autoencoder's image compression preserves useful information for the task. However, the loss function for the autoencoder in (4) only considers the mean squared error between the raw image and the reconstructed image. Without incorporating the reward of the task in the training of the autoencoder might result in the loss of the state information, which is necessary for the successful task completion.
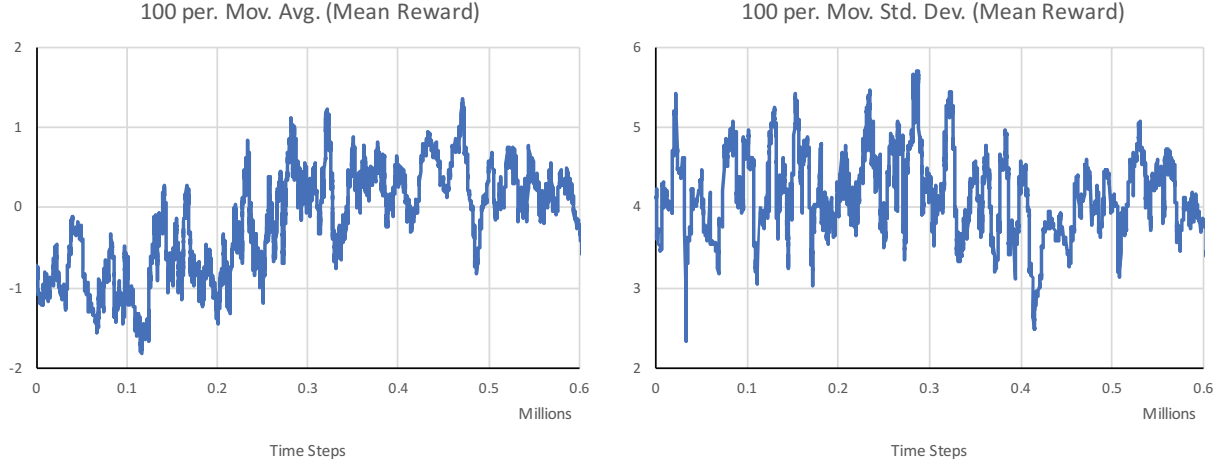
(a) Moving average of the mean rewards.

(b) Moving standard deviation of the mean rewards.

Fig. 6   Moving average and standard deviation of the mean rewards per episode.
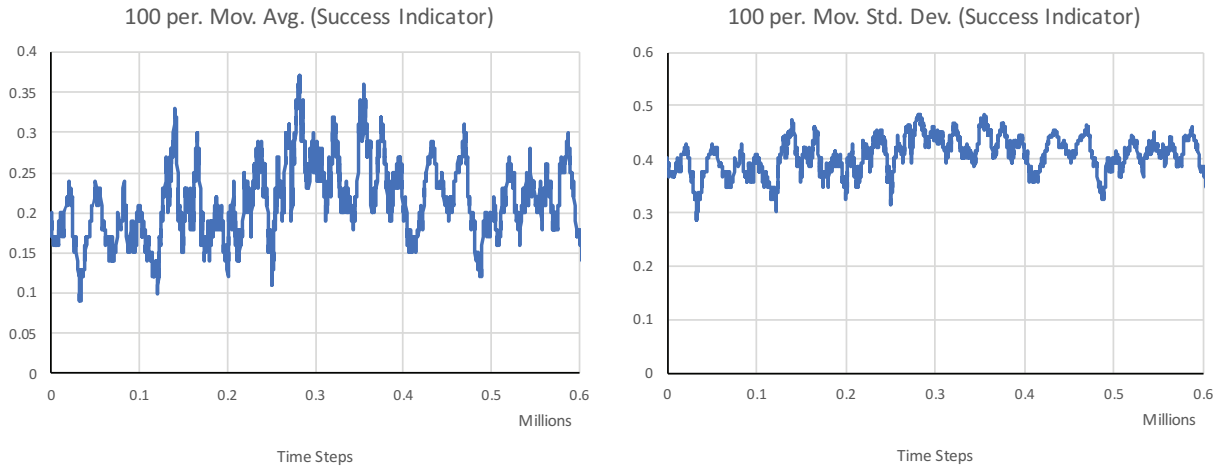
(a) Moving average of the success indicator.

(b) Moving standard deviation of the success indicator.

Fig. 7   Success indicator (1: successful episode, 0: otherwise).

In addition to the incomplete state observation, another challenge would be the uniform random initialization of an episode. Figure 6b shows large variations of mean reward per episode despite decaying learning rate. As the training progresses, the step-size (or learning rate) is decreased as shown in Figure 9 so that the actors (control policy) eventually
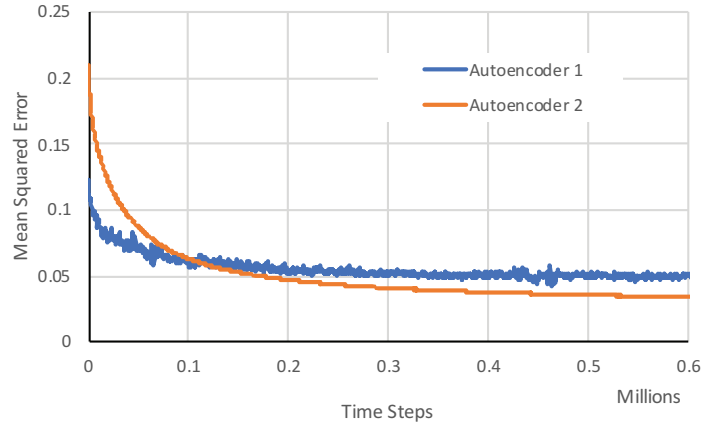
10

Fig. 8 Mean squared error (MSE) between the raw images and the images reconstructed by the autoencoders. Autoencoder 1 processes the stacked frames in Figure 3a and Autoencoder 2 processes the color images in Figure 3b.

stop evolving. However, there are still large variations in the mean reward. This suggests that the variation is mostly due to the uniformly random initialization, since the policy itself is being very slowly updated in the later training iterations. Obviously, robustness to the random initialization is desired for the reinforcement learning algorithm. However, the uniform randomness here might not be appropriate, since it makes difficult for the reinforcement learning algorithm to recognize patterns. Also, the environments (Atari games, robot dynamics, etc.) used in the successful deep reinforcement learning papers [2, 3] are more deterministic regarding the initialization compared to our simulation environment.
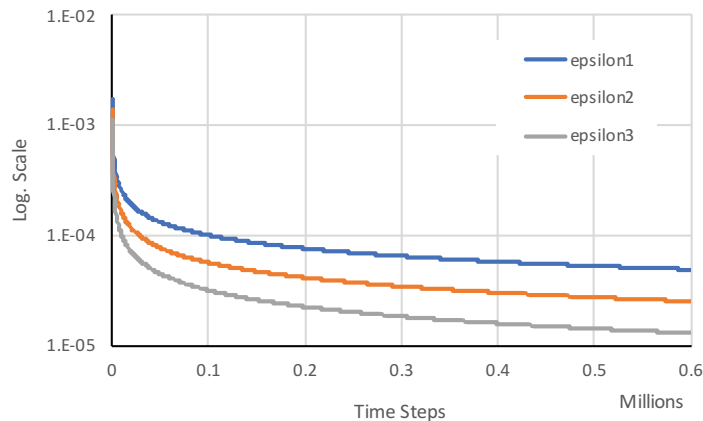

Fig. 9 The optimization scheduler decays the step-sizes to update the parameters. The fastest step-size $\epsilon_1(t)$ is for the central critic update, slower step-size $\epsilon_2(t)$ is for the actors , and the slowest $\epsilon_3(t)$ is for the encoder.

V. Conclusion

We present a reinforcement learning framework that takes into account decision making regarding sharing local information between agents for collaborative tasks. We use autoencoders to reduce the dimensions of the camera images taken by each agent. The encoded visual observations are fed to an actor-network, which decides communication messages to the other agents. The data compression by local agents makes it possible to implement deep reinforcement learning algorithms [2, 3], which use a large replay buffer (10^6 samples). To validate the feasibility of the proposed

11

framework, we develop a 3D simulation environment that provides camera images from different perspectives of the two heterogeneous agents. Simulation results show that the algorithm improves the performance of a given collaborative task.

A. Future Work

The proposed idea to use autoencoders to compress local visual observations in order to send it to the central critic, while keeping the usage of the communication resource economically feasible, is convincing. In the algorithm, each agent updates the autoencoder independently of the other agent. Minimizing the mean squared error of the autoencoder without considering the collaborative task might not lead to useful image encoding. In order to learn the dependence between local visual observations and the collaborative task reward, the agents need to share the raw high dimensional observations. However, sharing the high-dimensional data over the network cannot be as frequent as sharing the low-dimensional compressed image, due to communication resource constraints. So the data sharing will be asynchronous and in multiple time-scales, depending on the types of data being shared. Multiple time-scales and the asynchronous stochastic algorithm from [18] have the potential to address this issue. Intuitively, communication protocols should not change drastically for the agents to perform collaborative tasks. Hence, the use of communication networks to send the high dimensional local visual observation to the central critic, in order to update the communication protocol (autoencoder), should be less frequent than the update of the local policy for each agent.

Furthermore, navigation tasks in the 3D environment require the reinforcement learning algorithm to memorize the previously taken path. Humans navigate by constructing a map while memorizing the previous path. In other words, the human estimates the global state (location of the human in the map) based on the previous trajectory (taken path). So the 3D navigation is inherently a partially observable Markov decision process (POMDP), since the global state (the location) is hidden. To address the challenge of the POMDP, the use of the internal memory to predict or infer hidden states in the context of the reinforcement learning task was proposed in [22, 23]. For the 3D game environment, a recurrent neural network was employed to overcome the inherent partial observability of the navigation task [24]. However, the training of the recurrent neural network in [22–24] only uses the Bellman optimality equation, which does not consider how well the recurrent neural network can predict or infer hidden states. Maximum likelihood estimate (MLE) framework is a natural approach to infer hidden states. In [25], the authors proposed an on-line HMM estimation based reinforcement learning algorithm, which converges to the maximum likelihood estimate.

Advantages of using autoencoders or recurrent neural networks in reinforcement learning tasks can be from their capability to mimic the true environment, in the sense that the learning algorithm can predict or infer hidden states. The future work will focus on devising algorithms, which converge to the maximum likelihood estimate of the environment model that uses the autoencoder and the recurrent neural network. The estimated environment model will be used to construct state estimator. Reinforcement learning algorithms with the state estimator will be applied to various collaborative tasks in the 3D simulation environment.

Deploying the vision-based reinforcement learning to real cooperative missions as the ones in [1] has the potential to improve the resilience of multiagent systems to communication failures. For example, camera images contain the information on the location of other agents, which can be extrapolated to achieve coordination in the presence of communication failures. However, training of such intelligent agents requires significant random exploration to determine the optimal policies. Therefore, a method to *safely explore the policy space* is required to deploy the reinforcement learning algorithms on real missions. The future work will investigate methods to autonomously constrain the policy space, while learning the optimal policy.

Acknowledgment

References

[1] Kaminer, I., Pascoal, A. M., Xargay, E., Hovakimyan, N., Cichella, V., and Dobrokhodov, V., *Time-Critical Cooperative Control of Autonomous Air Vehicles*, Butterworth-Heinemann, 2017.

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al., "Human-level control through deep reinforcement learning," *Nature*, Vol. 518, No. 7540, 2015, p. 529.

[3] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[4] Shalev-Shwartz, S., Shammah, S., and Shashua, A., "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016.

[5] Busoniu, L., Babuska, R., and De Schutter, B., "A comprehensive survey of multiagent reinforcement learning," *IEEE Trans. Systems, Man, and Cybernetics, Part C*, Vol. 38, No. 2, 2008, pp. 156–172.

[6] Matignon, L., Laurent, G. J., and Le Fort-Piat, N., "Independent reinforcement learners in cooperative Markov games: A survey regarding coordination problems," *The Knowledge Engineering Review*, Vol. 27, No. 1, 2012, pp. 1–31.

[7] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., and Mordatch, I., "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in Neural Information Processing Systems*, 2017, pp. 6382–6393.

[8] Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S., "Counterfactual multi-agent policy gradients," *arXiv preprint arXiv:1705.08926*, 2017.

[9] Watkins, C. J., and Dayan, P., "Q-learning," *Machine learning*, Vol. 8, No. 3-4, 1992, pp. 279–292.

[10] Sutton, R. S., and Barto, A. G., *Reinforcement learning: An introduction*, Vol. 1, MIT press Cambridge, 1998.

[11] Foerster, J., Assael, I. A., de Freitas, N., and Whiteson, S., "Learning to communicate with deep multi-agent reinforcement learning," *Advances in Neural Information Processing Systems*, 2016, pp. 2137–2145.

[12] Mao, H., Ni, Y., Gong, Z., Ke, W., Ma, C., Xiao, Y., Wang, Y., Wang, J., Wang, Q., Liu, X., et al., "ACCNet: Actor-Coordinator-Critic Net for 'Learning-to-Communicate' with deep multi-agent reinforcement learning," *arXiv preprint arXiv:1706.03235*, 2017.

[13] Hinton, G. E., and Salakhutdinov, R. R., "Reducing the dimensionality of data with neural networks," *science*, Vol. 313, No. 5786, 2006, pp. 504–507.

[14] Konda, V. R., and Tsitsiklis, J. N., "Actor-critic algorithms," *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[15] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M., "Deterministic policy gradient algorithms," *ICML*, 2014.

[16] Tsitsiklis, J. N., and Van Roy, B., "Analysis of temporal-diffference learning with function approximation," *Advances in neural information processing systems*, 1997, pp. 1075–1081.

[17] Lange, S., Riedmiller, M., and Voigtlander, A., "Autonomous reinforcement learning on raw visual input data in a real world application," *Neural Networks (IJCNN), The 2012 International Joint Conference on*, IEEE, 2012, pp. 1–8.

[18] Kushner, H., and Yin, G. G., *Stochastic approximation and recursive algorithms and applications*, Vol. 35, Springer Science & Business Media, 2003.

[19] Xargay, E., Dobrokhodov, V., Kaminer, I., Pascoal, A. M., Hovakimyan, N., and Cao, C., "Time-critical cooperative control of multiple autonomous vehicles: Robust distributed strategies for path-following control and time-coordination over dynamic communications networks," *IEEE Control Systems*, Vol. 32, No. 5, 2012, pp. 49–73.

[20] Juliani, A., Berges, V.-P., Vckay, E., Gao, Y., Henry, H., Mattar, M., and Lange, D., "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2018.

[21] Ioffe, S., and Szegedy, C., "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[22] Heess, N., Hunt, J. J., Lillicrap, T. P., and Silver, D., "Memory-based control with recurrent neural networks," *arXiv preprint arXiv:1512.04455*, 2015.

[23] Hausknecht, M., and Stone, P., "Deep recurrent Q-learning for partially observable MDPs," *CoRR, abs/1507.06527*, 2015.

[24] Lample, G., and Chaplot, D. S., "Playing FPS Games with Deep Reinforcement Learning." *AAAI*, 2017, pp. 2140–2146.

[25] Yoon, H.-J., Lee, D., and Hovakimyan, N., "Hidden Markov Model Estimation-Based Q-learning for Partially Observable Markov Decision Process," *arXiv preprint arXiv:1809.06401*, 2018.

13