CS6007: Multi Agent Machine Learning

Prof. Avishek Ghosh

Course project report on

Exploring Delay-Aware Multi-Agent Reinforcement Learning

Preethi Chappidi
22B1074
Dept. of CSE

Jincy P Janardhanan
24M1075
Dept. of EE

November 8, 2025

Abstract

This project investigates multi-agent reinforcement learning (MARL) under realistic action delays. We first replicate the DAMA-DDPG algorithm, which explicitly incorporates delay awareness, in multi-agent environments to analyze how integer timestep delays impact policy stability and cumulative rewards.

We then extend this work by exploring non-integer delays (e.g., 1.7 timesteps), which are common in real-world systems. Using the concept of a virtual effective action, we approximate these fractional delays by interpolating between past actions. This allows us to evaluate the robustness of delay-aware MARL like DAMA-DDPG against more realistic, continuous delays.

Our goal is to quantify the effects of both integer and non-integer delays on model rewards, providing practical insights for designing robust MARL systems.

1 Introduction

Multi-Agent Reinforcement Learning (MARL) has emerged as a powerful paradigm for solving complex problems involving multiple autonomous agents, from robotic swarms to autonomous vehicle networks. Algorithms based on the Centralized Training with Decentralized Execution (CTDE) framework, such as the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [1], have demonstrated significant success in learning coordinated behaviors in cooperative, competitive, and mixed environments.

A core assumption in many standard MARL algorithms is that actions are executed immediately after they are computed. However, in real-world deployments, this is often not the case. Agents frequently experience action delays due to sensor processing times, computational latency, network communication delays, or physical actuation limits. These delays can severely disrupt the learning process, as an action based on an old observation may be invalid or counter-productive when it is finally executed, leading to unstable training and poor final performance.

1.1 Problem Statement

Standard MARL algorithms like MADDPG are not designed to account for action delays and often fail catastrophically when delays are introduced. While recent research has begun to address this issue—most notably with the Delay-Aware Multi-Agent DDPG (DAMA-DDPG) algorithm [2], which explicitly models integer timestep delays—a significant gap remains.

Existing work primarily focuses on delays that are integer multiples of the environment's simulation timestep. In contrast, real-world delays are continuous and may not align neatly with these discrete steps. A delay of 1.7 timesteps, for example, cannot be directly represented in a discrete-time simulation. The performance and robustness of delay-aware MARL algorithms like DAMA-DDPG under such realistic, *non-integer* delays remain an open question.

1.2 Research Objectives and Contributions

This project aims to investigate the impact of both integer and non-integer action delays on multi-agent learning and to evaluate the robustness of delay-aware algorithms. Our work is twofold:

1. **Replication and Validation:** We first replicate the DAMA-DDPG algorithm and validate its performance in standard multi-agent environments (e.g., cooperative navigation) under integer delays. This serves to establish a baseline and verify our implementation.

2. **Extension and Analysis:** We extend the delay-aware MARL framework to handle non-integer delays by leveraging the concept of a *virtual effective action* [3]. This method approximates fractional delays by interpolating between past actions.

2 Background and Literature Review

This section provides comprehensive coverage of the foundational concepts and prior work essential to understanding this project. We begin with core concepts in single-agent and multi-agent reinforcement learning, then provide detailed explanations of the major papers that form the basis of our work.

2.1 Single-Agent Reinforcement Learning and DDPG

Reinforcement Learning (RL) frames learning as an interaction between an agent and an environment through states, actions, and rewards. Formally, this is modeled as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ is the set of possible states
- $\mathcal{A}$ is the set of possible actions
- $\mathcal{P}(s'|s, a)$ is the state transition probability function
- $\mathcal{R}(s, a)$ is the reward function
- $\gamma \in [0, 1]$ is the discount factor that prioritizes immediate vs future rewards

The goal is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected cumulative discounted reward:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

where the expectation is taken over trajectories generated by following policy π.

The state-value function $V^\pi(s)$ and action-value function $Q^\pi(s, a)$ are defined as:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s \right]$$

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a \right]$$

The Q-function evaluates the expected return of taking action a in state s and subsequently following policy π, effectively answering the question, *"How good is it to take this action in this state?"* In contrast, the value function measures the expected return of being in state s under policy π, that is, *"How good is it to be in this state if we continue following policy π?"*

2.1.1 Deep Deterministic Policy Gradient (DDPG)

For continuous action spaces, the Deep Deterministic Policy Gradient (DDPG) algorithm [4] marked a key step forward in deep reinforcement learning. It is an actor-critic, model-free, off-policy method that merges ideas from Deep Q-Networks (DQN) with the Deterministic Policy Gradient theorem.

A major challenge in deep RL is the *moving target problem*, where simultaneous learning of the policy and value networks causes shifting value estimates and instability. DDPG mitigates this by employing *target networks* ($\mu'(s|\theta^{\mu'})$ and $Q'(s, a|\theta^{Q'})$), which are slowly updated through *soft updates*. These targets provide stable references for the main actor $\mu(s|\theta^\mu)$ and critic $Q(s, a|\theta^Q)$, improving convergence.

For deterministic policies $\mu : \mathcal{S} \to \mathcal{A}$, the policy gradient can be computed through the Q-function using the chain rule. DDPG realizes this by training:

- **Actor** $\mu(s|\theta^\mu)$: maps states to continuous actions,
- **Critic** $Q(s, a|\theta^Q)$: approximates the action-value function,
- **Target Networks** μ' and Q': slowly updated copies of actors and critics.

The actor agent proposes actions, while the critic agent evaluates them, guiding the actor toward actions yielding higher Q-values.

Critic Update: The critic minimizes the mean-squared Bellman error:

$$L(\theta^Q) = \mathbb{E}_{s_t,a_t,r_t,s_{t+1}}\big[(y_t - Q(s_t, a_t|\theta^Q))^2\big],$$

where

$$y_t = r_t + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'}).$$

Actor Update: The actor follows the deterministic policy gradient:

$$\nabla_{\theta^\mu} J \approx \mathbb{E}_{s_t}\big[\nabla_a Q(s, a|\theta^Q)|_{a=\mu(s_t)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_t}\big],$$

which adjusts policy parameters in the direction that increases the Q-value.

Target Network Soft Updates:

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}, \quad \theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'},$$

where $\tau \ll 1$ (typically 0.01), ensuring smooth target evolution and stable learning.

To further enhance stability, DDPG employs:

- **Experience Replay**: samples mini-batches from a replay buffer $\mathcal{D}$ to break temporal correlations.
- **Batch Normalization**: normalizes inputs to handle varying scales across environments.

2.2 Multi-Agent Reinforcement Learning and CTDE

Multi-Agent Reinforcement Learning (MARL) extends RL to settings with N agents interacting in a shared environment, typically modeled as a Markov Game defined by the tuple:

$$(\mathcal{S}, \mathcal{A}_1, \ldots, \mathcal{A}_N, \mathcal{P}, \mathcal{R}_1, \ldots, \mathcal{R}_N, \gamma, O_1, \ldots, O_N)$$

where:

- $\mathcal{S}$: Set of global states
- $\mathcal{A}_i$: Action space of agent i
- $\mathcal{P}$: Transition function $\mathcal{P}(s'|s, \mathbf{a})$ where $\mathbf{a} = (a_1, \ldots, a_N)$
- $\mathcal{R}_i$: Reward function for agent i, $\mathcal{R}_i(s, \mathbf{a})$
- O_i: Observation function for agent i, $o_i \sim O_i(s)$

Non-stationarity of the environemnt is a major challenge in MARL. From the perspective of any single agent, the environment appears non-Markovian because other agents are simultaneously learning and changing their policies. Formally, for agent i, the environment transition depends on:

$$P(s_{t+1}|s_t, a_i^t) = \sum_{\mathbf{a}_{-i}} P(s_{t+1}|s_t, a_i^t, \mathbf{a}_{-i})\pi_{-i}(\mathbf{a}_{-i}|s_t)$$

where $\mathbf{a}_{-i}$ represents actions of all other agents and π_{-i} represents their joint policy, which changes over time.

2.2.1 Centralized Training with Decentralized Execution (CTDE)

The CTDE framework [5] addresses the non-stationarity problem through the following decomposition of tasks:

- **Centralized Training**: During learning, agents have access to global information including:
  - Full state information $s \in \mathcal{S}$
  - Observations and actions of all other agents
  - Coordination signals or communication channels
- **Decentralized Execution**: During deployment, each agent i selects actions based only on its local observation o_i using its policy $\pi_i(a_i|o_i)$

Formally, for agent i, the centralized value function during training is:

$$Q_i^{\pi}(\mathbf{x}, a_1, \ldots, a_N) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{i,t+k} | \mathbf{x}_t = \mathbf{x}, a_{1,t} = a_1, \ldots, a_{N,t} = a_N\right]$$

where $\mathbf{x}$ represents the global state or the concatenation of all agent observations. During execution, each agent uses $\pi_i(a_i|o_i)$.

While the environment is non-stationary from a local perspective (because other agents are learning), it becomes stationary when considering the joint action-space and global state. This enables stable learning because the centralized critic sees the full picture.

2.3 Multi-Agent Deep Deterministic Policy Gradient (MADDPG)

The Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm [1] extends DDPG to the multi-agent setting under the CTDE framework. MADDPG maintains for each agent i:

- **Decentralized Actor** $\mu_i(o_i|\theta_i^{\mu})$: Takes local observation o_i and outputs action a_i
- **Centralized Critic** $Q_i(\mathbf{x}, a_1, \ldots, a_N|\theta_i^{Q})$: Takes global state $\mathbf{x}$ and all agents' actions $(a_1, \ldots, a_N)$

The policy gradient for agent i's actor is derived from the deterministic policy gradient theorem:

$$\nabla_{\theta_i} J(\mu_i) = \mathbb{E}_{\mathbf{x},a\sim\mathcal{D}}\left[\nabla_{\theta_i}\mu_i(a_i|o_i)\nabla_{a_i}Q_i^{\mu}(\mathbf{x}, a_1, \ldots, a_N)|_{a_i=\mu_i(o_i)}\right]$$

where $\mathcal{D}$ is the experience replay buffer containing tuples $(\mathbf{x}, a_1, \ldots, a_N, r_1, \ldots, r_N, \mathbf{x}')$.

This gradient has two parts:

1. $\nabla_{a_i}Q_i^{\mu}(\mathbf{x}, a_1, \ldots, a_N)$: How the Q-value changes as we change agent i's action
2. $\nabla_{\theta_i}\mu_i(a_i|o_i)$: How the actor's output changes as we change its parameters

The centralized critic for agent i is updated by minimizing the temporal difference error:

$$\mathcal{L}(\theta_i^{Q}) = \mathbb{E}_{(\mathbf{x},a,r,\mathbf{x}')\sim\mathcal{D}}[(y_i - Q_i(\mathbf{x}, a_1, \ldots, a_N|\theta_i^{Q}))^2]$$

where

$$y_i = r_i + \gamma Q_i^{\mu'}(\mathbf{x}', a_1', \ldots, a_N'|\theta_i^{Q'})|_{a_j'=\mu_j'(o_j')}$$

and μ' are target policies.

Advantages:

- **Stable Learning**: The centralized critics see the full environment, making their learning targets stationary despite other agents learning
- **Flexibility**: Can handle cooperative, competitive, and mixed environments by designing appropriate reward functions
- **Scalability**: Each agent learns independently while benefiting from centralized information during training

2.4 The Challenge of Action Delays

Action delays represent a fundamental challenge in real-world RL deployments. In delayed environments, when an agent takes an action a_t at time t based on observation o_t, the action actually affects the environment at time $t + \delta$, where δ is the delay. This creates a misalignment between the observation that triggered the action and the state in which it executes.

Formally, in a delayed MDP, the transition dynamics become:

$$s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_{t-\delta})$$

and the reward becomes:

$$r_t = \mathcal{R}(s_t, a_{t-\delta})$$

The delayed Q-function becomes:

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t, a_t \right]$$

but note that r_t depends on $a_{t-\delta}$, creating a complex temporal relationship.

This delay breaks the Markov property and can severely degrade performance because:

- **State-Action Misalignment**: Actions are executed in states different from those they were computed for
- **Temporal Credit Assignment**: Rewards received at time t are consequences of actions taken at time $t - \delta$, making credit assignment ambiguous
- **Inefficient Exploration**: Actions may have unpredictable effects due to the delay, making exploration less effective
- **Instability**: The policy improvement theorem no longer holds directly in delayed settings

The Bellman equation for delayed MDPs becomes:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}}[r_t + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^\pi(s_{t+1}, a_{t+1})]]$$

but $r_t = \mathcal{R}(s_t, a_{t-\delta})$, so the immediate reward depends on a past action rather than the current one.

2.5 Prior Work on Delay-Aware Reinforcement Learning

2.5.1 Continuous Time Reinforcement Learning

Doya (2000) [6] formulated RL in continuous time, providing a theoretical foundation for handling delays. The idea was to model the system using continuous-time dynamics:

$$\tau \frac{ds(t)}{dt} = f(s(t), a(t))$$

where τ is the time constant, and define the value function in continuous time:

$$V(s(t)) = \mathbb{E}\left[\int_t^\infty e^{-\frac{t'-t}{\tau}} r(s(t'), a(t'))dt'\right]$$

The continuous-time Hamilton-Jacobi-Bellman (HJB) equation becomes:

$$\max_a \left\{ r(s,a) + \frac{\partial V}{\partial s} f(s,a) \right\} = \frac{V(s)}{\tau}$$

Doya derived continuous-time versions of policy iteration and value iteration. For policy evaluation:

$$\tau \frac{dV^\pi}{dt} = -r(s, \pi(s)) + V^\pi(s) - \frac{\partial V^\pi}{\partial s} f(s, \pi(s))$$

By working directly in continuous time, we can handle arbitrary time delays naturally, as the formalism doesn't rely on discrete time steps aligning with delays.

2.5.2 Virtual Effective Action

Schuitema et al. (2010) [3] proposed the **virtual effective action** approach specifically for handling continuous delays in real-time dynamic systems. The core idea is to interpolate between past actions to approximate what the action *would have been* at the precise time it should have been executed.

For a delay δ (which may be non-integer), and given a sequence of past actions $a_t, a_{t-1}, \ldots, a_{t-K}$, the virtual effective action $\tilde{a}_{t-\delta}$ is computed as:

$$\tilde{a}_{t-\delta} = \sum_{k=0}^{K} w_k(\delta) a_{t-k}$$

where the weights $w_k(\delta)$ are determined by the interpolation method and satisfy $\sum_{k=0}^{K} w_k(\delta) = 1$.

For linear interpolation between two time steps:

$$\tilde{a}_{t-\delta} = (1-\alpha) a_{\lfloor t-\delta \rfloor} + \alpha a_{\lceil t-\delta \rceil}$$

where $\alpha = (t-\delta) - \lfloor t-\delta \rfloor$ represents the fractional part of the delay.

For higher-order interpolation (cubic), the weights are given by:

$$w_k(\delta) = \prod_{\substack{m=0 \\ m \neq k}}^{K} \frac{\delta - m}{k - m}$$

We can think of this as "filling in the gaps" between discrete time steps. If we only have actions at integer times but need an action at time $t-1.7$, we estimate it by blending the actions at $t-1$ and $t-2$ in the right proportion.

The modified transition function using virtual effective actions becomes:

$$s_{t+1} \sim \mathcal{P}(s_{t+1} | s_t, \tilde{a}_{t-\delta})$$

2.5.3 Smith Predictor and Control Theory Approaches

Altıntaş et al. (2002) [7] analyzed predictor-based controllers, including the Smith predictor, for time-delay systems. The Smith predictor is a classic control theory approach that uses a model of the plant to predict the current state in the absence of delays.

The structure involves three main components:

- **Process Model** $G(s)e^{-\theta s}$: Represents the actual plant with delay θ
- **Delay-Free Model** $G(s)$: Models the plant without delays
- **Model Error Correction**: Feedback of the difference between actual and predicted outputs

The control law is:

$$U(s) = C(s)[R(s) - (Y(s) - G(s)e^{-\theta s}U(s) + G(s)U(s))]$$

which simplifies to:

$$U(s) = \frac{C(s)}{1 + C(s)G(s)(1 - e^{-\theta s})}R(s) - \frac{C(s)}{1 + C(s)G(s)(1 - e^{-\theta s})}Y(s)$$

The Smith predictor effectively removes the delay from the feedback loop by using a model to predict what the output would be without delay, then correcting for model inaccuracies.

While originally designed for linear systems, the conceptual framework of predictive compensation has inspired RL approaches that learn predictive models to handle delays.

2.6 DAMA-DDPG: Delay-Aware Multi-Agent DDPG

The DAMA-DDPG algorithm [2] explicitly incorporates action delays into the MARL learning process, building upon MADDPG with modifications for delay awareness.

2.6.1 Delay Modeling and Architecture

Each agent i has a known, fixed action delay $\delta_i \in \mathbb{Z}^+$ (in integer timesteps). The algorithm maintains:

- **Delay Buffer** $\mathcal{B}_i^a = [a_i^0, a_i^1, \ldots, a_i^t]$: A FIFO queue storing the most recent $\delta_i + 1$ actions for each agent
- **Modified Actor** $\mu_i(o_i|\theta_i^\mu)$: Same as MADDPG, takes current observation and outputs action
- **Modified Critic** $Q_i(\mathbf{x}, \tilde{a}_1, \ldots, \tilde{a}_N|\theta_i^Q)$: Takes the global state and the *actions about to be executed* from the delay buffers

At each timestep t, the algorithm proceeds as:

1. Each agent i receives observation o_i^t and computes action $a_i^t = \mu_i(o_i^t|\theta_i^\mu)$
2. Store a_i^t in delay buffer $\mathcal{B}_i^a$
3. Retrieve action to execute: $\tilde{a}_i^t = a_i^{t-\delta_i}$ (from delay buffer)
4. Execute joint action $\tilde{\mathbf{a}}^t = (\tilde{a}_1^t, \ldots, \tilde{a}_N^t)$ in environment
5. Receive rewards r_i^t and next observations o_i^{t+1}
6. Store transition $(\mathbf{x}^t, \tilde{\mathbf{a}}^t, \mathbf{r}^t, \mathbf{x}^{t+1})$ in experience replay buffer $\mathcal{D}$

2.6.2 Modified Learning Updates

The critic update for agent i becomes:

$$\mathcal{L}(\theta_i^Q) = \mathbb{E}_{(\mathbf{x},\tilde{\mathbf{a}},\mathbf{r},\mathbf{x}')\sim\mathcal{D}}\left[(y_i - Q_i(\mathbf{x}, \tilde{a}_1, \ldots, \tilde{a}_N|\theta_i^Q))^2\right]$$

where the target is:

$$y_i = r_i + \gamma Q_i^{\mu'}(\mathbf{x}', \tilde{a}'_1, \ldots, \tilde{a}'_N|\theta_i^{Q'})$$

and $\tilde{a}'_j$ are the actions from the delay buffers that will be executed at the next state.

The actor gradient is modified to account for the delayed execution:

$$\nabla_{\theta_i} J(\mu_i) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D},\tilde{\mathbf{a}}\sim\mathcal{B}^a}\left[\nabla_{\theta_i}\mu_i(o_i|\theta_i^\mu)\nabla_{a_i}Q_i(\mathbf{x}, \tilde{a}_1, \ldots, \tilde{a}_N|\theta_i^Q)|_{a_i=\mu_i(o_i)}\right]$$

The major difference from standard MADDPG is that the critic evaluates the Q-value based on the actions that are *actually being executed* ($\tilde{a}_i$) rather than the actions just computed (a_i). This aligns the critic's evaluation with the actual effect on the environment.

2.6.3 Major Innovations and Limitations

DAMA-DDPG's major innovations include:

- **Explicit Delay Modeling**: Architectural changes that explicitly account for action delays
- **Temporal Alignment**: Critic evaluates based on executed actions rather than computed actions
- **Consistent Credit Assignment**: Rewards are properly attributed to the actions that caused them

However, DAMA-DDPG has several limitations:

- **Integer Delays Only**: Assumes $\delta_i \in \mathbb{Z}^+$, cannot handle fractional delays
- **Fixed Known Delays**: Requires delays to be known and constant throughout training
- **Synchronous Execution**: Assumes all agents operate on the same clock
- **Discrete Time**: Relies on discrete time steps aligning with the simulation

These limitations motivate our extension to non-integer delays using the virtual effective action approach, bridging the gap between DAMA-DDPG's architectural insights and Schuitema et al.'s interpolation methods for continuous delays.

3 Methodology

This section presents our Fractional-Delay Aware Multi-Agent Deep Deterministic Policy Gradient (FD-MADDPG) algorithm, which extends the MADDPG framework to handle both integer and non-integer action delays through virtual effective action computation and augmented temporal observation spaces.

3.1 Problem Formulation

Consider a multi-agent system with N agents operating in a partially observable Markov game. Each agent i has:

- Observation space: $\mathcal{O}_i \subseteq \mathbb{R}^{d_o}$
- Action space: $\mathcal{A}_i \subseteq \mathbb{R}^{d_a}$
- Policy: $\pi_i : \mathcal{O}_i \times \mathbb{R}^{K \cdot d_a} \to \mathcal{A}_i$
- Delay: $\delta_i \in \mathbb{R}_{\geq 0}$ (can be fractional)

The key challenge is that action a_i^t computed at time t based on observation o_i^t is executed at time $t + \delta_i$, creating temporal misalignment.

3.2 Fractional-Delay Aware MADDPG Algorithm

The FD-MADDPG algorithm extends the standard MADDPG framework to handle fractional delays. In this section, we provide a detaied overview of the algorithm.

1. Initialization:
MADDPG agents are initialized with *augmented observation spaces*, where each observation is extended by $K \cdot d_a$ dimensions to include recent action histories. Action history buffers $\mathcal{H}_i$ are initialized with zeros, and a replay buffer $\mathcal{D}$ of capacity C (typically 10^6) is created.

2. Episode Execution:
For M episodes of T timesteps, agents receive initial observations $\mathbf{o}$, augmented with their action histories to form $\tilde{o}_i^t$.

3. Interaction and Update:
At each timestep, agent i selects $a_i^t = \pi_i(\tilde{o}_i^t)$, and $\mathcal{H}_i$ is updated via FIFO to retain the latest K actions. To handle fractional delays $\delta_i = I + f$, a *virtual effective action* is computed as:

$$\tilde{a}_i^t = (1 - f)\, \mathcal{H}_i[I] + f\, \mathcal{H}_i[I + 1].$$

The virtual actions $\tilde{\mathbf{a}}^t$ are executed in the environment to obtain next observations and rewards $(\mathbf{o}^{t+1}, \mathbf{r}^t)$. Transitions $(\tilde{\mathbf{o}}^t, \mathbf{a}^t, \mathbf{r}^t, \tilde{\mathbf{o}}^{t+1})$ are stored in $\mathcal{D}$.

4. Learning:
Each critic minimizes the TD loss

$$\mathcal{L}(\theta_i^Q) = \mathbb{E}_{\mathcal{B}}\left[(Q_i(\tilde{\mathbf{o}}, \mathbf{a}) - y_i)^2\right], \quad y_i = r_i + \gamma Q_i'(\tilde{\mathbf{o}}', \pi_1'(\tilde{o}_1'), \ldots, \pi_N'(\tilde{o}_N')).$$

Actors are updated via deterministic policy gradient:

$$\nabla_{\theta_i^\pi} J = \mathbb{E}_{\mathcal{B}}\left[\nabla_{a_i} Q_i(\tilde{\mathbf{o}}, \mathbf{a}) \nabla_{\theta_i^\pi} \pi_i(\tilde{o}_i)\right],$$

and target networks are softly updated:

$$\theta_i^{Q'} \leftarrow \tau \theta_i^Q + (1 - \tau)\theta_i^{Q'}, \quad \theta_i^{\pi'} \leftarrow \tau \theta_i^\pi + (1 - \tau)\theta_i^{\pi'}.$$

Algorithm 1 Fractional-Delay Aware MADDPG (FD-MADDPG)

Require: Number of agents N, delay parameters $\delta_i \in \mathbb{R}_{\geq 0}$, buffer size $K = \lceil \max_i \delta_i \rceil + 1$
Ensure: Trained policies π_i robust to fractional delays

Initialize MADDPG agents with augmented observation dimensions $d_o + K \cdot d_a$
Initialize action history buffers $\mathcal{H}_i \leftarrow \{\mathbf{0}\}^K$ for $i = 1, \ldots, N$
Initialize replay buffer $\mathcal{D}$ with capacity C
for episode $= 1, 2, \ldots, M$ **do**
    Receive initial observations $\mathbf{o} = [o_1, \ldots, o_N]$
    Augment observations: $\tilde{\mathbf{o}} \leftarrow \text{Augment}(\mathbf{o}, \mathcal{H})$
    for $t = 1, 2, \ldots, T$ **do**
        Compute actions: $\mathbf{a}^t \leftarrow [\pi_1(\tilde{o}_1^t), \ldots, \pi_N(\tilde{o}_N^t)]$
        Update buffers: $\mathcal{H}_i \leftarrow \text{UpdateBuffer}(\mathcal{H}_i, a_i^t)$ for $i = 1, \ldots, N$
        Compute virtual effective actions:
        for $i = 1$ to N **do**
            $I \leftarrow \lfloor \delta_i \rfloor$, $f \leftarrow \delta_i - I$
            $\tilde{a}_i^t \leftarrow (1 - f) \cdot \mathcal{H}_i[I] + f \cdot \mathcal{H}_i[I + 1]$
        end for
        Execute $\tilde{\mathbf{a}}^t$ in environment, observe $\mathbf{o}^{t+1}, \mathbf{r}^t$
        Augment next observations: $\tilde{\mathbf{o}}^{t+1} \leftarrow \text{Augment}(\mathbf{o}^{t+1}, \mathcal{H})$
        Store $(\tilde{\mathbf{o}}^t, \mathbf{a}^t, \mathbf{r}^t, \tilde{\mathbf{o}}^{t+1})$ in $\mathcal{D}$
        if $|\mathcal{D}| \geq B$ and $t \mod U = 0$ **then**
            Sample batch $\mathcal{B} \sim \mathcal{D}$
            Update critics for $i = 1, \ldots, N$:
                $y_i \leftarrow r_i + \gamma Q_i'(\tilde{\mathbf{o}}', \pi_1'(\tilde{o}_1'), \ldots, \pi_N'(\tilde{o}_N'))$
                $\mathcal{L}(\theta_i^Q) \leftarrow \mathbb{E}_{\mathcal{B}}[(Q_i(\tilde{\mathbf{o}}, \mathbf{a}) - y_i)^2]$
                Update Q_i using $\nabla_{\theta_i^Q} \mathcal{L}(\theta_i^Q)$
            Update actors for $i = 1, \ldots, N$:
                $\nabla_{\theta_i^\pi} J \leftarrow \mathbb{E}_{\mathcal{B}}[\nabla_{a_i} Q_i(\tilde{\mathbf{o}}, \mathbf{a}) \nabla_{\theta_i^\pi} \pi_i(\tilde{o}_i)]$
                Update π_i using $\nabla_{\theta_i^\pi} J$
            Update target networks:
                $\theta_i^{Q'} \leftarrow \tau \theta_i^Q + (1 - \tau) \theta_i^{Q'}$
                $\theta_i^{\pi'} \leftarrow \tau \theta_i^\pi + (1 - \tau) \theta_i^{\pi'}$
        end if
        $\tilde{\mathbf{o}}^t \leftarrow \tilde{\mathbf{o}}^{t+1}$
    end for
end for

Table 1: Mathematical Notation in FD-MADDPG Algorithm

Notation	Description
N	Number of agents in the multi-agent system
δ_i	Action delay for agent i, can be any non-negative real number (e.g., 2.3 timesteps)
K	Action history buffer size, calculated as $K = \lceil \max_i \delta_i \rceil + 1$
$\mathcal{H}_i$	Action history buffer for agent i, storing the last K actions: $\mathcal{H}_i = [a_i^t, a_i^{t-1}, \ldots, a_i^{t-K+1}]$
$\tilde{o}_i^t$	Augmented observation for agent i at time t: $\tilde{o}_i^t = [o_i^t, \mathcal{H}_i[0], \ldots, \mathcal{H}_i[K-1]]$
$\mathbf{a}^t$	Vector of actions computed by all agents at time t: $\mathbf{a}^t = [a_1^t, \ldots, a_N^t]$
$\tilde{\mathbf{a}}^t$	Vector of virtual effective actions executed at time t: $\tilde{\mathbf{a}}^t = [\tilde{a}_1^t, \ldots, \tilde{a}_N^t]$
I, f	Integer and fractional parts of delay: $I = \lfloor \delta_i \rfloor$, $f = \delta_i - I$
$\mathcal{D}$	Experience replay buffer storing transitions $(\tilde{\mathbf{o}}^t, \mathbf{a}^t, \mathbf{r}^t, \tilde{\mathbf{o}}^{t+1})$
B	Batch size for training updates
U	Update frequency (number of timesteps between training updates)
γ	Discount factor for future rewards
τ	Target network update coefficient (typically $\tau \ll 1$)

Temporal Consistency: By augmenting observations with action history, each agent can reason about the temporal consequences of its actions under delays.

Fractional Delay Handling: The virtual effective action computation provides a continuous mapping from fractional delays to executable actions, ensuring smooth performance transitions as delay parameters change.

Credit Assignment: The algorithm maintains proper temporal credit assignment by ensuring that:

- Critics evaluate based on actions that were actually executed (virtual actions)
- Actors learn to compute actions that will be effective after the delay period
- The replay buffer stores the relationship between computed actions and their delayed consequences

Thus the FD-MADDPG algorithm provides a principled approach for handling realistic delay scenarios in multi-agent systems while maintaining the theoretical foundations and training stability of the original MADDPG framework.

3.3 Theoretical Foundations

3.3.1 Consistency with Integer Delays

When $\delta_i \in \mathbb{Z}_{\geq 0}$, the proposed formulation reduces to the standard delay-aware MADDPG:

$$\Phi(\mathcal{H}_i, \delta_i) = a_i^{t-\delta_i}, \quad \text{for } \delta_i \in \mathbb{Z}_{\geq 0}.$$

3.3.2 Effective Action under Non-Integral Delays

Consider each agent $i \in \{1, \ldots, N\}$ experiencing a non-negative real-valued action delay δ_i. Decompose it as:

$$\delta_i = I_i + f_i, \quad I_i = \lfloor \delta_i \rfloor, \quad f_i = \delta_i - I_i, \; f_i \in [0, 1).$$

At time step $t - I_i$, agent i executes action $a_i^{t-I_i}$, but due to the fractional delay f_i, the environment remains partly influenced by the preceding action $a_i^{t-I_i-1}$. During the interval $[t, t+1)$:

- $a_i^{t-I_i-1}$ persists for a fraction f_i of the step.
- $a_i^{t-I_i}$ influences the remaining $(1 - f_i)$ fraction.

Theorem. The resulting virtual (effective) action applied at time t is:

$$\tilde{a}_i^t = (1 - f_i)\, a_i^{t-I_i} + f_i\, a_i^{t-I_i-1}.$$

This linear interpolation ensures smooth transitions between consecutive actions and continuity in delay values:

$$\lim_{\epsilon \to 0} \Phi(\mathcal{H}_i, \delta_i + \epsilon) = \Phi(\mathcal{H}_i, \delta_i).$$

When $f_i = 0$, it recovers the standard discrete-time case $\tilde{a}_i^t = a_i^{t-I_i}$.

3.3.3 Augmented Observation Space

To enable temporal reasoning, each agent receives an augmented observation:

$$\tilde{o}_i^t = [o_i^t, \mathcal{H}_i[0], \mathcal{H}_i[1], \ldots, \mathcal{H}_i[K-1]] \in \mathbb{R}^{d_o + K \cdot d_a},$$

where $\mathcal{H}_i$ stores the most recent K actions. This augmentation provides temporal context for improved credit assignment.

3.3.4 Modified Bellman Equation

The centralized critic Q_i estimates the expected return over joint virtual actions:

$$Q_i(\tilde{\mathbf{o}}, \mathbf{a}) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_i^{t+k} \,\middle|\, \tilde{\mathbf{o}}^t = \tilde{\mathbf{o}}, \mathbf{a}^t = \mathbf{a}, \tilde{a}_j^{t+k} = \Phi(\mathcal{H}_j, \delta_j) \; \forall j \right].$$

3.3.5 Computational Complexity

The virtual action computation introduces $\mathcal{O}(N)$ additional operations per timestep, and the augmented observation increases the critic input dimension by $N \cdot K \cdot d_a$. Both overheads scale linearly with system parameters.

3.3.6 Convergence Guarantees

Assuming differentiable policies, bounded rewards, and sufficient exploration, and that the augmented observation serves as a sufficient statistic for the delayed system, FD-MADDPG preserves the convergence properties of standard MADDPG.

4 Experiments and Results

To evaluate the proposed delay-aware multi-agent reinforcement learning (MARL) framework, experiments were conducted on the `Simple Spread` environment from the PettingZoo MPE suite. This environment involves three agents cooperating to cover multiple landmarks while avoiding collisions, with continuous action spaces bounded within $[0, 1]$ using a sigmoid activation function.

Three models were compared:

- **Delay-Unaware MARL:** Standard MADDPG without delay modeling.
- **Integral-Delay MARL:** Incorporates fixed integer-step action delays (DAMA-DDPG).
- **Non-Integral-Delay MARL:** Handles fractional delays via linear interpolation for smoother temporal behavior (FD-MADDPG).

Evaluation Metric:

The performance of each model was assessed using the *mean episodic reward*, averaged across all agents, as a function of the episode number. This metric captures both the rate of convergence and the overall coordination performance of the agents under different delay conditions.

Simulation Settings:

The experiments were conducted using the following configuration:

- **Environment:** `simple_spread_v3 (PettingZoo)`
- **Number of Agents:** 3
- **Delay Values:** 0, 1, 1.7, 2, and 2.7
- **Training Duration:** 10,000 episodes for plotted results, extended to 30,000 episodes for long-term stability analysis
- **Hyperparameters:**
  Learning rate: 0.01; Discount factor (γ): 0.99; Soft update coefficient (κ): 0.01; Replay buffer size: 10^6; Batch size (B): 1024; Episode length (T): 25
- **Hardware:** Tesla T4 GPU (16 GB) with 4 vCPUs

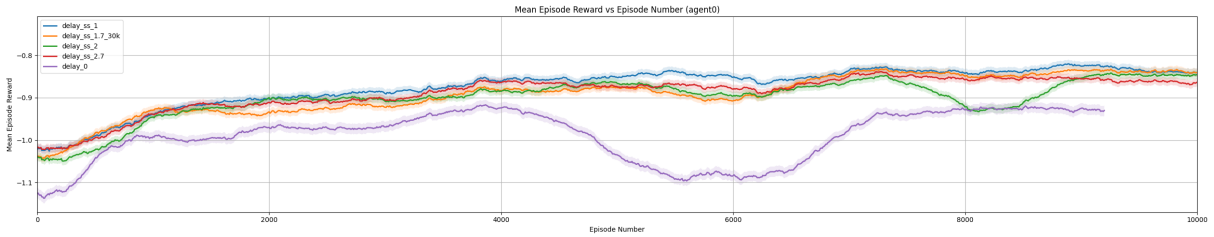

Figure 1: Training performance comparison of delay-unaware, integral-delay, and fractional-delay MARL models.

Each model was trained for 10,000 episodes, with additional extended runs up to 30,000 episodes for stability checks.

Results:

The delay-aware models consistently outperformed the delay-unaware baseline, achieving higher rewards and smoother convergence. The fractional delay (non-integral) variant achieved performance comparable to the integral-delay model, validating the interpolation-based formulation for handling non-integer delays. In contrast, the delay-unaware approach exhibited unstable learning and slower convergence beyond approximately 6,000 episodes. Incorporating delay-awareness improved coordination and robustness in delayed feedback scenarios.

5 Conclusion

This project explored the impact of both integer and non-integer action delays on multi-agent reinforcement learning (MARL), with the objective of evaluating the robustness of delay-aware algorithms. The work was carried out in two phases: replication and validation of the existing Delay-Aware Multi-Agent Deep Deterministic Policy Gradient (DAMA-DDPG) algorithm, and an extension to handle fractional (non-integer) delays through the use of virtual effective actions.

In the first phase, the DAMA-DDPG framework was successfully replicated and validated on standard cooperative environments such as `Simple Spread`, confirming its correctness and reliability in scenarios involving fixed integer-step delays. In the second phase, the framework was extended to address non-integer delays by introducing a linear interpolation mechanism that blends consecutive past actions to emulate continuous-time effects.

The results indicate that the proposed non-integral delay formulation performs on par with the integral-delay model, demonstrating that the interpolation-based virtual effective action approach effectively handles fractional delay scenarios without performance degradation.

6 Future Work

Future extensions of this research could focus on developing mechanisms to handle **heterogeneous delays** across agents. Additionally, expanding the framework to support **time-varying or stochastic delays** would further enhance its applicability to real-world multi-agent systems, where communication and action delays are often dynamic and uncertain.

References

[1] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in Neural Information Processing Systems*, 30, 2017.

[2] Yuyang Zhang, Shusen Li, Yang Yu, and Shusen Li. Delay-aware multi-agent reinforcement learning. *IEEE Access*, 8:85486–85496, 2020.

[3] E. Schuitema, L. Busoniu, R. Babuska, and P. Jonker. Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach. In *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2521–2526, 2010.

[4] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint*, 2015.

[5] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 29, pages 2137–2145, 2016.

[6] Kenji Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000.

[7] Y. Altıntaş, U. Büyükşahin, and A. Ersak. Analysis of a predictor-based controller for time-delay systems with some applications. *IEEE Transactions on Control Systems Technology*, 10(4):499–507, 2002.