MADDPG with Ensemble and UDR for Robust Multi-Agent Coordination

Francesco Mastrosimone
DAUIN
Politecnico di Torino
Turin, Italy
s348159@studenti.polito.

Brian Facundo Condorpocco Morales
DAUIN
Politecnico di Torino
Turin, Italy
s346581@studenti.polito.

Salvatore Nocita
DAUIN
Politecnico di Torino
Turin, Italy
s346378@studenti.polito.

Luciano Scarpino
DAUIN
Politecnico di Torino
Turin, Italy
s346205@studenti.polito.

***Abstract*—This project explores Reinforcement Learning (RL) for continuous control tasks, with a focus on the sim-to-real transfer challenge. The agents were trained in a source domain with altered dynamics and evaluated in a target domain representing unmodified real-world conditions.**

We used Proximal Policy Optimization (PPO) as our main RL algorithm and applied Uniform Domain Randomization (UDR) to improve the robustness of the policy. UDR significantly reduced the performance gap between training and deployment domains, outperforming non-randomized baselines.

As a project extension, we implemented Multi-Agent Deep Deterministic Policy Gradient (MADDPG) in a cooperative environment and introduced ensemble policy strategies to address non-stationarity. The resulting multi-agent models, especially when combined with UDR, achieved strong generalization under previously unseen dynamics, matching or surpassing the robustness of single-agent PPO policies, suggesting MARL as a viable alternative to traditional domain randomization.

Here you can find the project repository.

***Index Terms*—Reinforcement Learning, Policy Gradient Methods, REINFORCE, Actor-Critic, Proximal Policy Optimization (PPO), Sim-to-Real Transfer, Domain Randomization, Multi-Agent Reinforcement Learning (MARL), MADDPG, Ensemble Policies**

I. Introduction

This project explores Reinforcement Learning (RL) techniques for robotic control in simulation, with a focus on sim-to-real transfer—the challenge of training policies in a virtual environment and successfully deploying them in the real world. Using the Hopper environment, a one-legged robot tasked with learning to hop forward without falling, we studied policy robustness across two domains: a **source domain** where the torso mass is reduced by 30% to simulate the gap with the real world, and a **target domain** representing the unaltered dynamics.

The work proceeded in stages: we first implemented two classic RL algorithms—REINFORCE and Actor-Critic—to gain insight into policy gradient methods. We then trained agents using Proximal Policy Optimization (PPO) in both domains to establish lower and upper performance bounds, before introducing Uniform Domain Randomization (UDR) to increase robustness to domain shifts.

As an extension, we explored Multi-Agent Reinforcement Learning (MARL) as an alternative path to robustness. Specifically, we implemented MADDPG in a cooperative balancing scenario using the VMAS simulator, where agents must coordinate to stabilize and lift a shared object. We further enhanced this setup with ensemble policies to address instability from agent interactions. The MARL models trained with UDR achieved strong generalization under perturbations in mass, gravity, and object position. These findings reinforce the idea that multi-agent variability can act as an implicit source of regularization—helping agents develop more robust, transferable policies even in unpredictable environments.

II. REINFORCE Algorithm

A. Objective and Formulation

REINFORCE is a Monte Carlo policy gradient algorithm that maximizes the expected return of a stochastic policy $\pi_\theta(a|s)$ [1]. The objective is:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[G_t], \quad G_t = \sum_{k=t}^{T} \gamma^{k-t} R_k \tag{1}$$

Policy parameters are updated via gradient ascent:

$$\theta \leftarrow \theta + \alpha G_t \nabla_\theta \log \pi_\theta(A_t|S_t) \tag{2}$$

which reinforces actions in proportion to their returns.

B. Algorithm Overview

The REINFORCE algorithm collects complete trajectories by interacting with the environment using the current stochastic policy. Once an episode terminates, it computes the return associated with each action and updates the policy parameters using the policy gradient estimate.

The update is performed retrospectively at the end of each episode. As a stochastic gradient ascent method, REINFORCE guarantees improvement in expected performance for sufficiently small learning rates, and converges to a local optimum under standard assumptions [2].

C. Baseline for Variance Reduction

While REINFORCE is theoretically sound, in practice it suffers from high variance in gradient estimates and slow convergence. A common remedy is introducing a baseline function $b(s_t)$, subtracted from the return G_t, yielding the *advantage* $A_t = G_t - b(s_t)$. This transformation reduces

variance without introducing bias, as long as the baseline is action-independent. By centering the return, the advantage helps stabilize updates: actions yielding higher-than-expected outcomes are reinforced, while others are suppressed.

D. Experimental Results

We implemented two baseline variants: a *constant scalar* baseline ($b = 20$) and a *learnable* one modeled as a neural network $V(s)$, trained jointly with the policy. These were compared against a *no-baseline* version using the same hyperparameters.

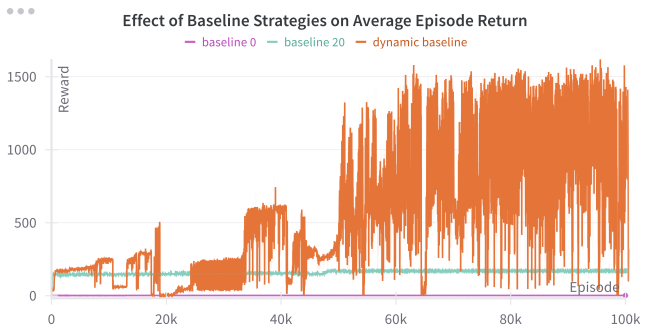

Fig. 1. Average return over training episodes for REINFORCE agents using no baseline, constant baseline, and dynamic baseline.

Figure 1 shows the average episode return during training for the three baseline strategies. The agent trained without a baseline (purple) failed to learn effectively, with returns stagnating near zero. The constant baseline (light blue) improved stability and accelerated learning, enabling the agent to reach average returns of approximately 150. This confirms the expected effect and validates the variance reduction role of the baseline in practice. The learnable dynamic baseline (orange) achieved substantially higher returns—peaking above 1500—but exhibited considerable variance and training instability.

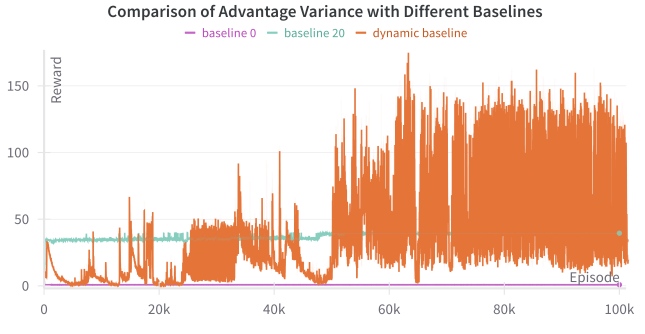

Fig. 2. Standard deviation of the advantage term during training.

Figure 2 shows that the constant baseline effectively reduced variance, as indicated by the consistently low standard deviation of the advantage. In contrast, the dynamic baseline led to persistent fluctuations due to the instability of learning the value function online.

III. Actor-Critic Algorithm

A. Objective and Formulation

The Actor-Critic algorithm merges value-based and policy-based approaches by maintaining two networks: an *actor* $\pi_\theta(a|s)$ that selects actions, and a *critic* $V_w(s)$ that evaluates the current policy [2].

The critic is trained via Temporal Difference (TD) learning to minimize prediction error:

$$\delta_t = r_t + \gamma V_w(S_{t+1}) - V_w(S_t) \tag{3}$$

$$\mathcal{L}_{\text{critic}} = \delta_t^2 \tag{4}$$

The actor uses the TD error as an advantage estimate:

$$\hat{A}(s_t, a_t) \approx \delta_t = r_t + \gamma V_w(S_{t+1}) - V_w(S_t) \tag{5}$$

to adjust its policy via gradient ascent:

$$\theta \leftarrow \theta + \alpha_\theta \hat{A}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{6}$$

This simultaneous update—gradient descent for the critic, ascent for the actor—enables online learning and continuous policy improvement.

B. Algorithm Overview

Actor-Critic is an online, incremental algorithm in which both the actor and the critic are updated at each timestep. The actor selects an action according to the current policy, the environment returns a reward and the next state, and the critic updates its value estimate using a bootstrapped target. The actor then adjusts the policy based on the estimated advantage. This process is repeated continuously, allowing the agent to learn and adapt during interaction with the environment.

C. Experimental Results

After extensive hyperparameter tuning, we selected the configuration with $\gamma = 0.99$, $\alpha_\theta = 0.0005$, and critic learning rate $\alpha_w = 0.0001$, as it provided faster learning progress and better variance reduction.

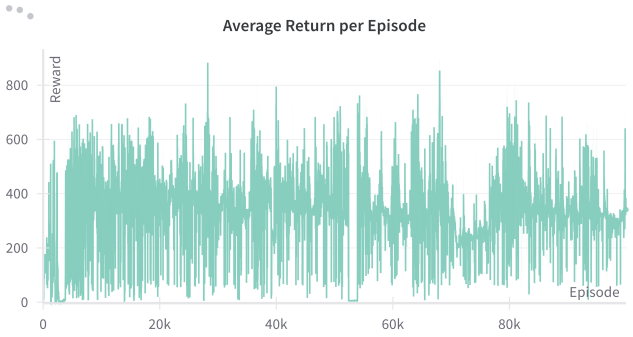

Fig. 3. Average return per episode during training.

Figure 3 shows the evolution of the average return. Compared to the REINFORCE agents, Actor-Critic outperformed both the no-baseline and constant-baseline configurations. Although it did not reach the peak performance achieved by

REINFORCE with a learned dynamic baseline, it displayed significantly more stable learning dynamics.

To further assess this stability, Figure 4 reports the standard deviation of the reward over a moving window. A clear downward trend is visible, indicating progressive reduction in return variability as training progresses.

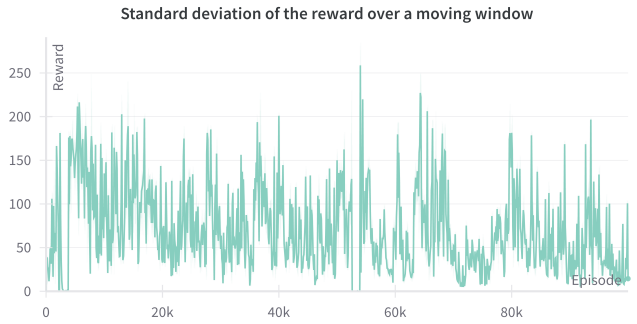

Fig. 4. Standard deviation of the episode return.

In terms of time consumption, Actor-Critic required more wall-clock time to complete 100,000 episodes than REINFORCE with a learned baseline. Nonetheless, it achieved faster progress in the early phases of training and maintained more stable performance throughout, with significantly lower variance in returns.

Overall, Actor-Critic offers a compelling balance between performance, stability, and convergence speed, especially when compared to REINFORCE baselines.

IV. PPO Algorithm

A. Objective and Formulation

Proximal Policy Optimization (PPO) is a state-of-the-art policy gradient algorithm that balances sample efficiency, performance, and implementation simplicity. It improves training stability by constraining the size of policy updates through a clipped surrogate objective [3].

The PPO objective is defined as:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{7}$$

where:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new and old policies,
- ϵ is a small hyperparameter (e.g., 0.2) that defines the clipping range.

The clipping operation prevents excessively large updates, thus improving the stability and reliability of the learning process.

B. Algorithm Overview

PPO alternates between two phases: (1) collecting trajectories by interacting with the environment using the current policy, and (2) updating the policy via multiple epochs of minibatch stochastic gradient ascent. The advantage estimates $\hat{A}_t$ are used to weight the policy updates, and the clipped surrogate objective ensures that the new policy does not deviate too far from the previous one, avoiding destructive updates.

C. Experimental Results – Lower and Upper Bounds

We trained the PPO agent using the implementation provided by the `stable-baselines3` library. The training pipeline was designed to ensure both efficient data collection and systematic performance monitoring. To accelerate experience gathering, we employed vectorized environments via `make_vec_env`, which allowed the agent to interact with multiple environment instances in parallel, while evaluation rewards were tracked using the `EvalCallback` utility.

Extensive hyperparameter tuning was conducted to maximize training stability and performance in the source domain. Table I summarizes the final configuration used for the best-performing run.

TABLE I
HYPERPARAMETERS FOR PPO TRAINING ON THE SOURCE DOMAIN

Hyperparameter	Value
Learning rate	5e-4
Batch size	128
Number of steps (n_steps)	1024
Clip range	0.2
Entropy coefficient	0.001
GAE lambda	0.95
Discount factor (γ)	0.99
Number of epochs	10

Training progress was tracked via the average return computed on rollout episodes. Figure 5 shows the resulting learning curve, which highlights steady improvement and convergence toward high-reward policies.

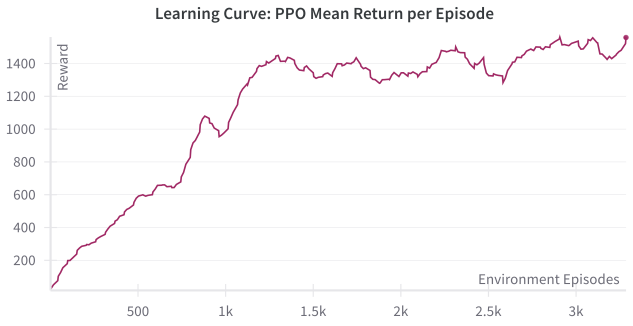

Fig. 5. Learning curve for PPO training in the source domain: mean episode reward over environment steps, computed on rollouts during training.

D. Training and Testing Across Domains

To assess the cross-domain generalization of the trained PPO agents, we evaluated their performance across three configurations, as summarized in Table II, which reports the mean reward and standard deviation over 50 evaluation episodes per setting.

TABLE II
CROSS-DOMAIN EVALUATION RESULTS FOR PPO (50 EVALUATION EPISODES PER CONFIGURATION)

Train Domain	Test Domain	Mean Reward	Reward Std. Dev.
Source	Source	1726.33	77.46
Source	Target	707.16	5.13
Target	Target	1729.82	30.99

The sharp drop in performance when testing the source-trained agent in the target domain illustrates the challenge of *domain shift*: policies optimized for specific state-transition dynamics often fail when deployed in environments with different physical characteristics.

This configuration (**source** $\rightarrow$ **target**) acts as a lower bound, while **target** $\rightarrow$ **target** defines the upper bound achievable with direct access to the target domain.

While training directly in the target domain may yield optimal performance, it is often impractical. As a result, sim-to-real transfer—training in simulation and deploying in the real world—remains the most viable approach for real-world robotics and control applications [5].

V. UNIFORM DOMAIN RANDOMIZATION (UDR)

A major challenge in sim-to-real RL is the *domain gap* between simulation (source) and deployment (target), which can cause significant performance drops (see Table II).

To improve robustness, we applied **Uniform Domain Randomization (UDR)** during training by varying the Hopper robot's link masses:

- **Scope:** The *thigh*, *leg*, and *foot* masses were randomized independently each episode. The torso mass remained fixed to preserve the source-target mismatch.
- **Distribution:** Masses were sampled uniformly within $\pm 20\%$ of their nominal values. This interval was selected after testing multiple ranges, offering the best tradeoff between robustness and training stability.
- **Protocol:** Training used PPO in the source domain, with mass randomization applied per episode.

A. Results and Analysis

After training, the UDR-trained policy was evaluated on both the source and target domains (Table III), following the same 50-episode testing protocol as used in earlier tasks.

TABLE III
EVALUATION RESULTS FOR THE UDR-TRAINED PPO AGENT

Train Domain	Test Domain	Mean Reward	Reward Std. Dev.
Source (UDR)	Source	1698.34	3.27
Source (UDR)	Target	1708.81	8.61

Compared to the baseline *source* $\rightarrow$ *target* performance (707.16 ± 5.13), the UDR-trained agent achieved a substantial improvement when tested on the target domain (1708.81 ± 8.61), nearly matching the *target* $\rightarrow$ *target* upper bound (1729.82 ± 30.99). This demonstrates that UDR effectively enhances policy robustness and transferability, despite not directly addressing the torso mass mismatch during training.

Overall, these results confirm that UDR can mitigate the impact of unmodeled dynamics in sim-to-real transfer settings, provided that the randomization ranges are carefully tuned.

VI. PROJECT EXTENSION: MULTI-AGENT REINFORCEMENT LEARNING

A. Introduction

Most Reinforcement Learning (RL) research has focused on single-agent settings, where the agent interacts with a stationary environment. However, many real-world tasks are inherently *multi-agent*, requiring collaboration or competition among multiple autonomous entities.

To address these challenges, Multi-Agent Reinforcement Learning (MARL) has emerged as a dedicated field. A particularly successful paradigm within MARL is *centralized training with decentralized execution* (CTDE), where critics access global state and joint action information during training and agents act based on local observations. Among the most prominent CTDE algorithms is MADDPG [12], which extends DDPG by using centralized critics for each agent.

In parallel, *ensemble policy strategies* have been proposed as a means to combat the instability and overfitting introduced by non-stationary dynamics. By training multiple sub-policies per agent on diverse experiences and aggregating their outputs or representations, ensemble methods aim to increase policy diversity and stability.

This work investigates how these two approaches—domain randomization and ensemble learning—can be combined in a multi-agent context to improve policy robustness and generalization.

B. Limitations of Single-Agent

Traditional single-agent RL algorithms prove inadequate when applied to tasks involving multiple interacting agents, due to several structural issues. First and foremost, the simultaneous learning processes of multiple agents render the environment inherently *non-stationary* from the perspective of each individual agent. State transitions and rewards no longer depend solely on the agent's own actions, but also on the continuously evolving policies of its counterparts. This violation of the Markov assumption compromises the convergence guarantees of classical algorithms such as Q-learning and invalidates core techniques like experience replay, which presuppose a stationary environment. Additionally, in cooperative scenarios, policy gradient methods are subject to *high variance*. Since rewards often depend on the *joint actions* of all agents, the learning signal becomes noisy—especially when each agent updates its policy independently, without accounting for the behavior of others. This leads to inefficient learning and unstable convergence.

Another key challenge is *partial observability*. In many real-world applications, agents lack access to the global state of the environment and must instead rely on local, incomplete observations. This makes it significantly harder to infer the

causal relationship between an agent's actions and their outcomes, especially in the presence of other autonomous agents whose behavior also influences the environment. Moreover, single-agent algorithms are not designed to explicitly model interactions between agents. They treat other agents as part of the environment—static or at best stochastic—rather than as adaptive entities with their own objectives and policies. This abstraction limits the expressiveness of learned strategies and undermines the development of effective coordinated or competitive behaviors.

C. Related Works

Many real-world problems are intrinsically *multi-agent*, where coordination among several autonomous entities is essential for effective task execution. This is evident in domains such as multi-robot systems [6], where cooperation is required to navigate dynamic environments or carry out search-and-rescue operations; in advanced communication [7] and energy networks [8], where distributed control strategies govern power allocation and resource management; and in military applications involving UAV swarms [9], where agents must adapt to both local and global conditions. A further challenge arises in collaborative scenarios involving humans and artificial agents [10], where the unpredictability of human behavior demands greater adaptability from machine policies.

D. Methodology

1) DDPG: Before introducing its multi-agent extension, we first briefly review Deep **Deep Deterministic Policy Gradient** [11], as it serves as the foundation for MADDPG.
DDPG is a single-agent, off-policy actor-critic algorithm designed for continuous action spaces. Understanding its core structure is essential for appreciating the modifications needed to adapt it to multi-agent settings.

DDPG learns a deterministic policy $\mu_\theta(s)$ to maximize the action-value function $Q^\mu(s,a)$ using the deterministic policy gradient:

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)} \nabla_\theta \mu_\theta(s)\right] \tag{8}$$

To improve stability, DDPG uses experience replay, target networks, and exploration noise. While effective in some settings, DDPG is sensitive to hyperparameters and lacks robustness in complex or multi-agent environments—limitations addressed by its extension, MADDPG.

2) MADDPG: **Multi-Agent Deep Deterministic Policy Gradient** [12] extends DDPG to multi-agent scenarios with interacting agents and non-stationary dynamics. MADDPG operates under three constraints: (1) policies must rely only on local observations at execution time, (2) the method does not require a differentiable model of environment dynamics, and (3) no assumptions are made about differentiable communication between agents. To meet these goals, MADDPG adopts a *centralized training with decentralized execution* (CTDE) paradigm. During training, each agent i learns a deterministic policy $\mu_{\theta_i}(o_i)$ using a centralized critic Q_i that conditions on the joint actions $a_1,\ldots,a_N$ and global state information x. The policy gradient for agent i is expressed as:

$$\nabla_{\theta_i} J(\mu_i) = \mathbb{E}_{x,\,a\sim\mathcal{D}}\big[\nabla_{\theta_i}\mu_i(a_i \mid o_i) \cdot \nabla_{a_i} Q_i^\mu(x,a_1,\ldots,a_N)\big|_{a_i=\mu_i(o_i)}\big] \tag{9}$$

The centralized critic is updated using the Bellman loss:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{x,a,r,x'}\left[\left(Q_i^\mu(x,a_1,\ldots,a_N) - y\right)^2\right] \tag{10}$$

$$y = r_i + \gamma Q_i^{\mu'}(x',a'_1,\ldots,a'_N)\big|_{a'_j=\mu'_j(o'_j)} \tag{11}$$

where $Q_i^{\mu'}$ and μ'_j are the respective target networks. This design allows the critic to model the true multi-agent dynamics by conditioning on the full joint action, effectively restoring stationarity to the environment from each agent's perspective. Despite this centralized structure during training, each policy remains decentralized and depends only on local observations at the test time. This balance enables MADDPG to scale to complex cooperative and competitive tasks where agents may have heterogeneous and potentially conflicting reward functions.

The complete training loop, including optional ensemble support, is outlined in the pseudocode in Algorithm 1 (Appendix).

3) Ensemble Policy Strategy: In dynamic multi-agent settings, the learning signal each policy sees changes as other agents update, which makes single policies brittle. To improve stability and performance under this non-stationarity, we use an *ensemble*: each agent maintains K sub-policies that are trained on partially disjoint experience streams and kept in separate replay buffers to preserve their specialization [12]. The ensemble provides a diversified set of behaviors—some tuned to different partner strategies or environment regimes —so at decision time it is aggregated, reducing variance, improving robustness to distribution shifts, and mitigating catastrophic regressions during training.

A common approach is to aggregate the neural network weights of the sub-policies into a single policy. However, this method is computationally intensive and fragile to architectural mismatches. Instead, we propose a simpler alternative: compute the action from each sub-policy for the same observation s, average these actions, and use the result as a supervision target for a distilled policy $\hat{\mu}$:

$$\begin{aligned} a^{(k)} &= \mu^{(k)}(s), \quad k = 1,\ldots,K \\ \bar{a} &= \frac{1}{K}\sum_{k=1}^{K} a^{(k)} \end{aligned} \tag{12}$$

The distilled policy is trained using mean squared error:

$$\mathcal{L}_{\text{MSE}} = \|\hat{\mu}(s) - a_{\text{avg}}\|^2 \tag{13}$$

This method consolidates the ensemble into a single executable policy while retaining the diversity and robustness

gained through ensemble training. It improves adaptability to varying agent behaviors and promotes more stable coordination.

VII. Experiments and Results

A. Environment: VMAS Balance

We evaluated our MARL approach using the **Balance** scenario from the *VMAS* library [13], a vectorized PyTorch-based simulator for scalable multi-agent training. VMAS supports parallelized rollouts, accelerating data collection.

In this continuous-control cooperative task, N agents are aligned beneath a beam that supports a spherical package of mass `package_mass`. The system spawns at a random position near the bottom of the environment, under gravity. A goal zone is placed in the upper region (see Figure 6). Agents must coordinate to balance the package and lift it toward the goal.

Each agent receives partial observations, including: its own position, the relative positions to the beam, to the package, the relative position between package and goal, the velocities and beam rotation. A shared reward encourages progress toward the goal and penalizes failure (e.g., beam or package hitting the ground, with a -10 penalty). Episodes end either when the package or the beam fall or when the package touches the goal.

We used the official VMAS implementation via `make_env` to configure agent count, duration, and action space. Vectorization boosted training speed but didn't reduce the number of episodes needed for convergence.

1) UDR: To improve robustness in our multi-agent setting, we extended **Uniform Domain Randomization (UDR)** (see Section V) to the VMAS Balance environment. We randomized both environmental and task-specific dynamics.

Specifically, we applied the following randomizations during training:

- **Package Mass:** Uniformly sampled in the range $[3, 7]$ (default: 5), altering the load agents must stabilize and lift.
- **Gravity:** Sampled from a uniform distribution over $\vec{g} \in \{(0, -0.01), (0, -0.1)\}$ (default: $(0, -0.05)$), changing the difficulty of vertical lifting and balance.
- **Package Positioning:** Enabled `random_package_pos_on_line`, causing the package to spawn at a different location along the beam in each episode. This introduces high variability in the coordination challenge from the very start.

These randomized parameters were applied on a per-episode basis during training..

B. Architecture

We implemented a customized version of the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm within a *centralized training, decentralized execution* (CTDE) framework. Each agent maintains an ensemble of up to K deterministic actor networks, with one policy sampled at each episode to enhance exploration diversity. A global critic evaluates joint actions based on the concatenated observations of all agents, enabling coordinated learning across agents. Both actor and critic networks are fully connected (64–64 units) and trained using the Adam optimizer, with stability ensured through soft updates of target networks.

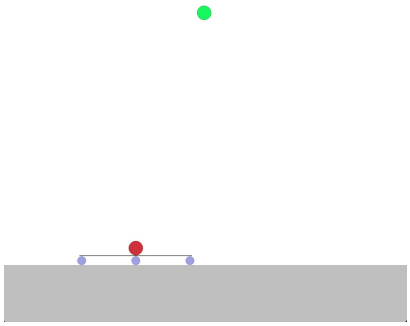

Fig. 6. Illustration of the VMAS *Balance* environment. Agents (bottom) must lift and stabilize a beam with a package on top, under gravity.

Exploration is driven by additive Gaussian noise, scaled by a decaying coefficient (H, ρ). Transitions are collected from N parallel VMAS environments and stored in a shared multi-agent replay buffer. During updates, the critic minimizes the mean squared error (MSE) between the target function (11) and $Q^{\mu}(s, a)$, while each actor maximizes the critic's output obtained by replacing a newly computed action in the critic's input joint action vector a.

To reduce execution-time overhead, ensemble policies are distilled post-training into a single student actor per agent via supervised regression on state-action pairs $(s, \bar{a})$, where $\bar{a}$ denotes the ensemble-averaged action (12). The resulting architecture integrates ensemble-based exploration, knowledge distillation, and CTDE, and is validated on continuous control tasks in the VMAS multi-agent benchmark.

C. Hyperparameter Optimization

We optimized the MADDPG hyperparameters using Optuna, which applies Bayesian optimization with early stopping to efficiently explore the search space. Each trial involved a full training run on the VMAS Balance scenario, with performance tracked via Weights & Biases. The best configuration, shown in Table IV, revealed that sustained exploration—enabled by a moderate initial noise level and slow decay—was crucial to avoid premature convergence. Learning rates for actor and critic were tuned to ensure stable updates, while a high discount factor favored long-term coordination. The soft-update parameter τ and large batch size contributed to stable target estimation and low-variance gradients, respectively, improving both convergence and final reward consistency.

D. Train

Training was conducted over 7500 episodes, corresponding to more than 2 million timesteps. This decision was based

TABLE IV
TUNED HYPERPARAMETERS FOR MADDPG

Hyperparameter	Value
Noise scale (H)	0.274
Actor learning rate (α)	2.91e-4
Batch size	512
Critic learning rate (β)	3.55e-4
Noise decay rate	0.998
Discount factor (γ)	0.976
Learn every (episodes)	10
Soft update coefficient (τ)	0.025

on empirical evidence indicating that the learning process consistently plateaued around a mean reward score of 400. Beyond this point, no significant improvement in performance was observed, suggesting that further training would yield diminishing returns.

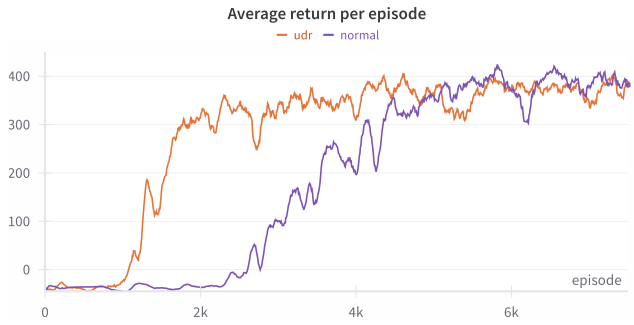

Fig. 7. Learning curves of baseline model and UDR version.

Figure 7 reports the learning curves obtained from the baseline model trained without ensemble methods, both with domain randomization and not. The average score over the last 100 episodes (*Average return per episode*) shows a consistent improvement during training, reaching a plateau around 400, which suggests that the agents are able to learn a reasonably effective policy. However, in figure 9 are shown standard deviation of all models: it is possible to notice a considerable increase in performance variance as training progresses, especially after 3000 steps, with fluctuations stabilizing around a high value due to fast learning in this phase.

Figure 7 also illustrates the training performance of the model when Uniform Domain Randomization (UDR) is applied during training, without the use of ensemble methods. The inclusion of UDR does not significantly alter the overall trend of the learning curves in terms of convergence speed, as the *average reward* again rapidly increases and stabilizes around 400.

Figure 8 compares the training performance of the ensemble model with and without domain randomization. The version without UDR (blue) shows faster initial learning and reaches higher returns earlier, though with greater variability. In contrast, the UDR-enhanced model (green) learns more gradually but exhibits more stable performance over time. Both approaches eventually converge to similar average returns, indicating that UDR does not significantly impacts final performance.

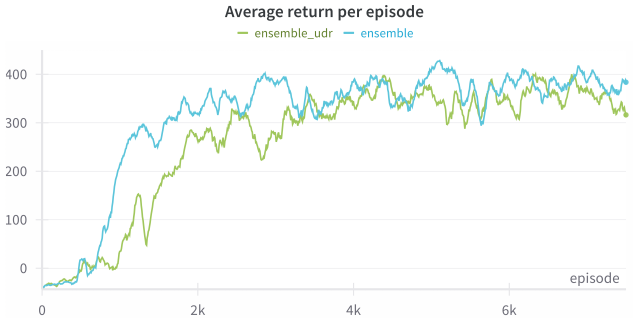

Fig. 8. Learning curves of ensemble model and UDR version.

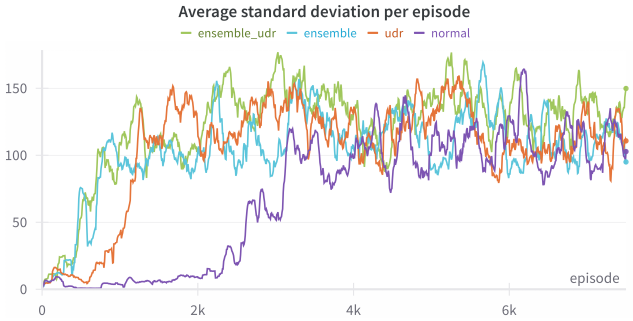

Fig. 9. Learning curves of compared standard deviation.

It is evident in Figure 9 that variances of all models remain high, despite UDR fluctuations are more regular, suggesting that the model trained under randomized domain conditions can generalize better across varying environment dynamics. This is a desired outcome of domain randomization, which exposes the policy to a wider variety of configurations, promoting the learning of behaviors that are less sensitive to overfitting specific dynamics.

Overall, the adoption of UDR enhances the robustness and generality of the learned policy, reducing the risk of failure in unseen scenarios, despite a similar mean performance level. These findings support the hypothesis that domain randomization contributes to more resilient policy learning in multi-agent control tasks involving variable dynamics.

E. Test

The test phase was carried out over 200 episodes. To approximate the sim-to-real gap, the environment was deliberately modified to include variations not encountered during training. Specifically, the test conditions involved:

- a 60% increase in the package mass,
- a 60% reduction in gravity,
- a lateral displacement of the package from the center of the bar.

These perturbations were introduced to evaluate the robustness and generalization capability of the learned policies under unseen dynamics.

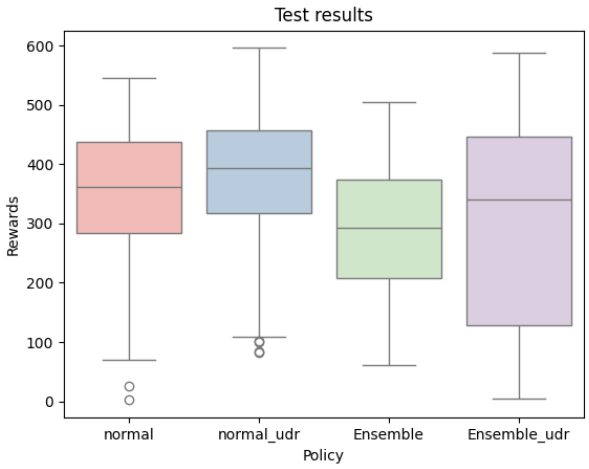

Fig. 10. Learning curves of compared standard deviation.

Figure 10 presents the distribution of rewards obtained under four different policies: *normal*, *normal_udr*, *Ensemble*, and *Ensemble_udr*. Overall, policies incorporating UDR (Uniform Domain Randomization) tend to exhibit higher median rewards and greater robustness across episodes. The normal_udr policy achieves the highest median reward and demonstrates a relatively compact interquartile range, suggesting consistent performance with limited variability. In contrast, the Ensemble_udr policy shows the widest reward distribution, indicating increased variability but also achieving some of the highest reward values among all tested configurations. The standard normal and Ensemble policies, both trained without UDR, display lower median rewards and more pronounced presence of outliers, particularly in the normal case. These results indicate that UDR contributes positively to policy robustness and that, when combined with ensemble methods, it can enhance peak performance, although with increased variance.

F. Discussion

The experimental results validate the correctness and effectiveness of our MADDPG implementation. Across all baseline configurations, agents learned to solve the Balance task with stable training curves, low variance, and convergence within a relatively short number of episodes. These findings indicate that the learning dynamics were well captured, and that the centralized training with decentralized execution (CTDE) paradigm was successfully applied in our setup.

Among all tested configurations, **MADDPG with Uniform Domain Randomization (UDR) and no ensemble consistently achieved the best performance**, both in terms of average reward and stability during test time. Despite being exposed to shifted dynamics not seen during training, this model exhibited strong generalization and resilience, confirming the utility of UDR in multi-agent settings. Moreover, it is important to note that the test configuration was fixed for this analysis. A more comprehensive test protocol involving broader randomization of environmental parameters would further highlight the superior generalization capabilities of the UDR trained policy.

In contrast, **ensemble-based models underperformed during testing**, regardless of whether UDR was used. While training behavior was comparable to non-ensemble models, test rewards were approximately 20% lower, and standard deviation was paradoxically higher. This contradicts the theoretical advantages of ensemble methods, which are expected to increase robustness and reduce performance variance.

We hypothesize that these results stem primarily from challenges in the *distillation phase*. The student actor may have lacked the capacity or architectural compatibility to faithfully replicate the behavior of the ensemble. Additionally, averaging actions from multiple sub-policies may have diluted specialized strategies learned by individual actors. Fragmented experience across separate replay buffers may have also led to less consistent learning signals.

VIII. Conclusion and Future Works

Despite some limitations, this project successfully achieved its main objective: **developing a robust multi-agent reinforcement learning pipeline** capable of managing coordination, partial observability, and domain variability. Our implementation of MADDPG, combined with Uniform Domain Randomization (UDR), proved effective in learning stable and generalizable policies, even under unseen test-time conditions.

While ensemble strategies introduced architectural richness, their benefits did not fully materialize in our experiments. Future work will focus on improving ensemble integration—particularly the policy distillation process. Promising directions include attention-based aggregation mechanisms, adaptive sub-policy selection, or bypassing distillation entirely in favor of online ensemble execution. These approaches could help better retain the diversity and robustness offered by ensemble training, ultimately enhancing performance in complex multi-agent tasks.

References

[1] L. Weng, “Policy Gradient Algorithms,” *Lilian Weng’s Blog*, Apr. 8, 2018. [Online]. Available: https://lilianweng.github.io/posts/2018-04-08-policy-gradient/

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., Cambridge, MA, USA: MIT Press, 2018.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” *arXiv preprint*, arXiv:1707.06347, 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[4] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, “High-Dimensional Continuous Control Using Generalized Advantage Estimation,” *arXiv preprint*, arXiv:1506.02438, 2015. [Online]. Available: https://arxiv.org/abs/1506.02438

[5] F. Muratore, M. Gienger, M. Traversaro, and J. Peters, “Robot Learning from Randomized Simulations: A Review,” *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7781–7788, 2021. [Online]. Available: https://doi.org/10.1109/LRA.2021.3095540

[6] L. Matignon, L. Jeanpierre, and A.-I. Mouaddib, “Coordinated multi-robot exploration under communication constraints using decentralized Markov decision processes,” in *Proc. AAAI Conf. Artif. Intell.*, 2012, pp. 563–568.

[7] T. Li, K. Zhu, N. C. Luong, D. Niyato, Q. Wu, Y. Zhang, and B. Chen, "Applications of Multi-Agent Reinforcement Learning in Future Internet: A Comprehensive Survey," *arXiv preprint*, arXiv:2110.13484, 2022. [Online]. Available: https://arxiv.org/abs/2110.13484

[8] S. S. Ghazimirsaeid, M. Selseleh Jonban, M. Wijesooriya Mudiyanselage, M. Marzband, J. L. Romeral Martinez, and A. Abusorrah, "Multi-agent-based energy management of multiple grid-connected green buildings," *J. Build. Eng.*, vol. 74, p. 106866, 2023. [Online]. Available: https://doi.org/10.1016/j.jobe.2023.106866

[9] J. Cui, Y. Liu, and A. Nallanathan, "Multi-agent reinforcement learning based resource allocation for UAV networks," *arXiv preprint*, arXiv:1810.10408, 2018. [Online]. Available: https://arxiv.org/abs/1810.10408

[10] P. Sen and S. M. Jakkaraju, "Modeling AI-human collaboration as a multi-agent adaptation," *arXiv preprint*, arXiv:2504.20903, 2025. [Online]. Available: https://arxiv.org/abs/2504.20903

[11] 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," *arXiv preprint*, arXiv:1509.02971, 2015. [Online]. Available: https://arxiv.org/abs/1509.02971

[12] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," *arXiv preprint*, arXiv:1706.02275, 2017. [Online]. Available: https://arxiv.org/abs/1706.02275

[13] M. Bettini, R. Kortvelesy, J. Blumenkamp, and A. Prorok, "VMAS: A Vectorized Multi-Agent Simulator for Collective Robot Learning," arXiv preprint arXiv:2207.03530, 2022. [Online]. Available: https://github.com/proroklab/VectorizedMultiAgentSimulator

APPENDIX

Algorithm 1: MADDPG training procedure for N agents, with optional ensemble policies per agent.

Initialize replay buffer $\mathcal{D}$;
for *each agent* i **do**
  Initialize actor μ_{θ_i}, critic Q_{ϕ_i};
  Initialize target networks μ'_{θ_i}, Q'_{ϕ_i};
  if *with_ensemble* **then**
    for $k \leftarrow 1$ **to** K **do**
      Initialize sub-policy $\mu_{\theta_i^{(k)}}$, target $\mu'_{\theta_i^{(k)}}$, and buffer $\mathcal{D}_i^{(k)}$;
for *each episode* **do**
  Observe initial global state x, local observations $\{o_1, \ldots, o_N\}$;
  for $t \leftarrow 1$ **to** *episode length* **do**
    for *each agent* i **do**
      if *with_ensemble* **then**
        Sample $k \sim \mathcal{U}(1, K)$;
        Select action $a_i = \mu_{\theta_i^{(k)}}(o_i) + \mathcal{N}_t$;
      else
        Select action $a_i = \mu_{\theta_i}(o_i) + \mathcal{N}_t$;
    Execute joint action $a = (a_1, \ldots, a_N)$, observe r, x', $\{o'_1, \ldots, o'_N\}$;
    Store (x, a, r, x') in $\mathcal{D}$ (or $\mathcal{D}_i^{(k)}$ if ensemble);
    $x \leftarrow x'$;
  for *each agent* i **do**
    if *with_ensemble* **then**
      for *each sub-policy* k **do**
        Sample minibatch from $\mathcal{D}_i^{(k)}$;
        Compute target $y = r_i + \gamma Q'_{\phi_i}(x', a'_1, \ldots, a'_N)$;
        Update critic by minimizing $(Q_{\phi_i}(x, a_1, \ldots, a_N) - y)^2$;
        Update actor using policy gradient: $\nabla_{\theta_i^{(k)}} J = \nabla_{\theta_i^{(k)}} \mu_{\theta_i^{(k)}}(o_i) \nabla_{a_i} Q_{\phi_i}(x, a)$;
    else
      Sample minibatch from $\mathcal{D}$;
      Compute target $y = r_i + \gamma Q'_{\phi_i}(x', a'_1, \ldots, a'_N)$;
      Update critic by minimizing $(Q_{\phi_i}(x, a_1, \ldots, a_N) - y)^2$;
      Update actor using policy gradient: $\nabla_{\theta_i} J = \nabla_{\theta_i} \mu_{\theta_i}(o_i) \nabla_{a_i} Q_{\phi_i}(x, a)$;
    Update target networks: $\theta'_i \leftarrow \tau \theta_i + (1 - \tau)\theta'_i$;