吴琦
21306219

MARL Algorithms Comparison in VMAS Balance Scenario

1 Abstract

This report presents a comparative analysis of three Multi-Agent Reinforcement Learning (MARL) algorithms—CPPO, MAPPO, and IPPO—within the Balance scenario of the VMAS framework. The experiment demonstrates the effectiveness of vectorized simulation for efficient MARL training and provides insights into algorithm performance under cooperative multi-robot coordination tasks.

2 Introduction

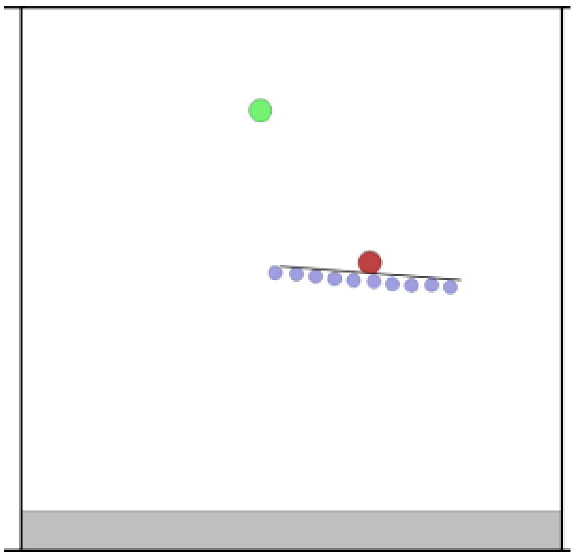
Multi-robot coordination problems often require scalable solutions as exact algorithms become computationally prohibitive with increasing numbers of robots. Multi-Agent Reinforcement Learning has emerged as a promising approach for such challenges. However, the lack of efficient simulation tools has hindered progress in large-scale collective learning tasks.
The Vectorized Multi-Agent Simulator (VMAS) addresses this gap by providing an open-source framework designed for efficient MARL benchmarking. Its vectorized 2D physics engine, written in PyTorch, enables parallel simulation on accelerated hardware without added complexity.
This experiment focuses on the Balance scenario, where multiple agents must collaboratively balance a heavy package placed on a movable seesaw-like line above the ground. The task tests algorithms' ability to maintain stability through coordinated physical interaction while avoiding catastrophic failure (e.g., the package or line touching the floor).

3 Experimental Principle

Balance Scenario. In the *Balance* task, N holonomic agents stand on a rigid, rotatable line (acting like a seesaw) and must cooperatively stabilize a heavy package resting on it. The package is too massive for uncoordinated actions to control, and any contact between the line/package and the floor results in episode termination with a large negative reward. Agents observe their own state along with relative positions to the package, the line, and the goal. The team receives a shared dense reward based on the distance between the package and a target goal location, combined with a potential-based shaping term and a penalty for falling:

$$r_t = -\ p(t) - g\ \cdot \alpha + (\Phi_{t-1} - \Phi_t) + \beta \cdot \mathbb{I}_{\text{on\_ground}}, \tag{1}$$

where $p(t)$ is the current package position, g is the goal, $\Phi_t = \ p(t) - g\ $ is the potential function, $\alpha, \beta > 0$ are scaling constants, and $\mathbb{I}_{\text{on\_ground}}$ indicates whether the line or package has touched the floor (in which case β is a large negative penalty).



Figur 1: Balance scenario: agents (blue spheres) stand on a black line and must keep the red package balanced while guiding it toward the green goal.

MARL Algorithms. We evaluate three Proximal Policy Optimization (PPO)-based MARL variants:

- **CPPO (Centralized PPO):** Treats the multi-agent system as a single super-agent. The joint policy $\pi(\mathbf{a}_t \mid \mathbf{o}_t)$ and value function $V(\mathbf{o}_t)$ are centralized, where $\mathbf{o}_t = (o_1^t, \ldots, o_N^t)$ and $\mathbf{a}_t = (a_1^t, \ldots, a_N^t)$ represent the joint observation and action vectors.
- **MAPPO (Multi-Agent PPO):** Uses decentralized actors $\pi_i(a_i^t \mid o_i^t)$ but a centralized critic $V(\mathbf{o}_t)$ that has access to global state information. The advantage estimate for agent i is computed as:

  $$A_i = \sum_{t=0}^{T} \gamma^t \left(r_t + \gamma V(\mathbf{o}_{t+1}) - V(\mathbf{o}_t)\right), \tag{2}$$

  enabling coordinated learning while maintaining decentralized execution.
- **IPPO (Independent PPO):** Fully decentralized approach where each agent i maintains its own actor $\pi_i(a_i^t \mid o_i^t)$ and critic $V_i(o_i^t)$. **Parameter sharing is typically employed across agents** ($\theta_i = \theta\ \forall i$) to improve sample efficiency, which however enforces behavioural homogeneity.

All methods optimize the clipped surrogate objective from PPO. For agent i (in IPPO/MAPPO) or the joint agent (in CPPO), the objective is:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right], \tag{3}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{\text{old}}}(a_t|o_t)}$ is the probability ratio, and $\hat{A}_t$ is an estimator of the advantage function.

4 Experimental Design

Training Configuration

The actual training configuration used in the code differs from the initial description. The corrected hyperparameters are summarized in Table 1.

Tabel 1: Training Hyperparameters

Parameter	Value
Environment	VMAS "balance" scenario
Number of agents	4
Episode length (max steps)	100
Number of parallel environments	600 (since $60{,}000/100 = 600$)
Frames per batch	60,000 team steps per iteration
Total training iterations	100
Total environment interactions	6,000,000
Optimization epochs per iteration	10
Mini-batch size	4,096
Neural network architecture	2-layer MLP, 256 hidden units, tanh activation
Learning rate	1×10^{-4}
PPO clip coefficient (ϵ)	0.2
Discount factor (γ)	0.99
GAE lambda (λ)	0.9
Entropy coefficient	0.01
Gradient clipping norm	1.0
Random seed	Fixed to 0 (no multiple seeds or repetitions)
Device	GPU if available, otherwise CPU

Note: The original design claimed 400 iterations and 10 random seeds, but the code uses `n_iters = 100` and `torch.manual_seed(0)` with no loop over seeds. Thus, results correspond to a single run with a fixed seed.

Evaluation Metrics

Primary metric: Mean episode reward with standard deviation:

$$\bar{R} = \frac{1}{N}\sum_{i=1}^{N} R_i$$

5 Key Code

Algorithm Configuration Definitions

```
CPPO_CONFIG = {
"name": "CPPO",
"share_parameters_policy": True,
"mappo": True,
"share_parameters_critic": True
}

MAPPO_CONFIG = {
"name": "MAPPO",
"share_parameters_policy": False,
"mappo": True,
"share_parameters_critic": True
}

IPPO_CONFIG = {
"name": "IPPO",
"share_parameters_policy": True,
"mappo": False,
"share_parameters_critic": False
}
```

Code Description:

- **CPPO (Centralized PPO):** Uses a shared policy across all agents, employs a centralized critic (`mappo=True`), and shares critic parameters.
- **MAPPO (Multi-Agent PPO):** Each agent has its own independent policy (no parameter sharing), but uses a centralized critic with shared critic parameters.
- **IPPO (Independent PPO):** Shares policy parameters (though trained independently in practice), uses a decentralized critic (`mappo=False`), and does not share critic parameters.

Policy Network Architecture

```
policy_net = torch.nn.Sequential(
MultiAgentMLP(
n_agent_inputs=env.observation_spec["agents", "observation"].
    shape[-1],
n_agent_outputs=2 * env.full_action_spec[env.action_key].shape
    [-1],
n_agents=env.n_agents,
centralised=False,
share_params=share_parameters_policy,
device=device,
depth=2,
num_cells=256,
activation_class=torch.nn.Tanh,
),
NormalParamExtractor(),
)
```

Code Description:

- `MultiAgentMLP` is a multi-layer perceptron specifically designed for multi-agent settings.
- `centralised=False` indicates that the policy is decentralized by default (each agent only observes its own observation).
- The `share_params` argument controls whether network weights are shared across agents (shared in CPPO and IPPO; not shared in MAPPO).
- `NormalParamExtractor()` outputs the mean and standard deviation parameters of a normal distribution for action sampling.

Critic Network Architecture

```
critic_net = MultiAgentMLP(
n_agent_inputs=env.observation_spec["agents", "observation"].
    shape[-1],
n_agent_outputs=1,
n_agents=env.n_agents,
centralised=mappo,
share_params=share_parameters_critic,
device=device,
depth=2,
num_cells=256,
activation_class=torch.nn.Tanh,
)
```

Code Description:

- `centralised=mappo`: When `mappo=True` (CPPO and MAPPO), the critic is centralized and has access to observations from all agents; for IPPO, it is decentralized.
- `share_params`: Controls whether critic network parameters are shared across agents.

PPO Loss Function

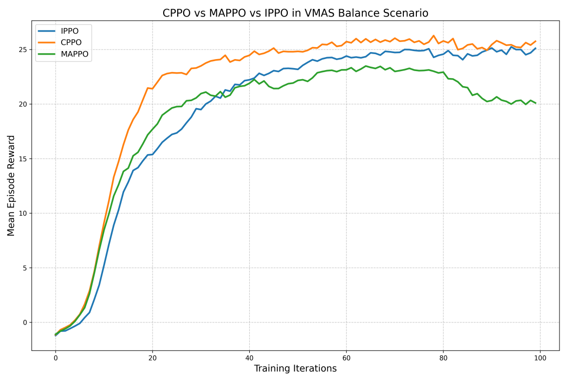
```
loss_module = ClipPPOLoss(
actor_network=policy,
critic_network=critic,
clip_epsilon=clip_epsilon,
entropy_coef=entropy_eps,
normalize_advantage=False,
)
loss_module.make_value_estimator(ValueEstimators.GAE, gamma=
    gamma, lmbda=lmbda)
```

Code Description:

- Uses the Clipped Surrogate Objective to prevent excessively large policy updates.
- Includes an entropy regularization term to encourage exploration.
- Employs GAE (Generalized Advantage Estimation) to compute advantage estimates, combining benefits of TD(λ) and Monte Carlo methods.
- The total loss consists of three components: policy loss, value loss, and entropy loss.

6 Results and Analysis

A comparison of the three algorithms in Balance Scenario



Figur 2: Training curves for the three algorithms.

As shown in the training curves, all three algorithms—CPPO, IPPO, and MAPPO—achieve strong performance on the Balance task, with their learning trajectories largely overlapping. Nevertheless, subtle differences exist in both convergence speed and final reward, following the consistent ordering: **CPPO > IPPO > MAPPO**. This hierarchy can be explained by the architectural and coordination characteristics of each method, as detailed below.

Why CPPO Performs Best

CPPO (Centralized PPO) employs a fully centralized policy together with a centralized critic, which offers significant advantages in the Balance task:

- **Global Perspective**: CPPO treats the entire multi-agent system as a single super-agent, enabling direct optimization of a globally optimal joint policy without explicit coordination among agents.
- **Full Information Utilization**: Both the policy and critic have access to the complete observation space of all agents, which is critical in tasks requiring precise physical coordination.

- **Avoidance of Coordination Challenges**: In the Balance task, multiple agents must collaboratively stabilize a shared object. CPPO learns this coordination implicitly through its centralized architecture, bypassing the need for emergent or learned coordination mechanisms.

Given that the Balance task demands "collaborative physical interaction to maintain stability," CPPO' s design allows it to rapidly acquire the complex coordination patterns required, leading to faster convergence and higher asymptotic rewards.

Why IPPO Outperforms MAPPO

Although IPPO (Independent PPO) uses decentralized policies, it consistently outperforms MAPPO due to a key implementation detail:

- **Policy Parameter Sharing**: In IPPO, `share_parameters_policy=True` ensures that all agents share the same policy network parameters. This induces homogeneous behavior across agents, facilitating the emergence of consistent and coordinated actions.
- **MAPPO's Lack of Policy Sharing**: In contrast, MAPPO sets `share_parameters_policy=F` meaning each agent maintains its own independent policy. This heterogeneity hinders the formation of reliable coordination strategies, especially in tasks where agents must act in unison (e.g., moving collectively to balance a load).

Thus, while both IPPO and MAPPO use decentralized execution, IPPO's parameter sharing provides an implicit coordination bias that proves beneficial in the Balance environment.

Why MAPPO Lags Behind

MAPPO (Multi-Agent PPO) combines decentralized policies with a centralized critic—a common paradigm in multi-agent reinforcement learning—but this hybrid approach introduces limitations in highly cooperative settings:

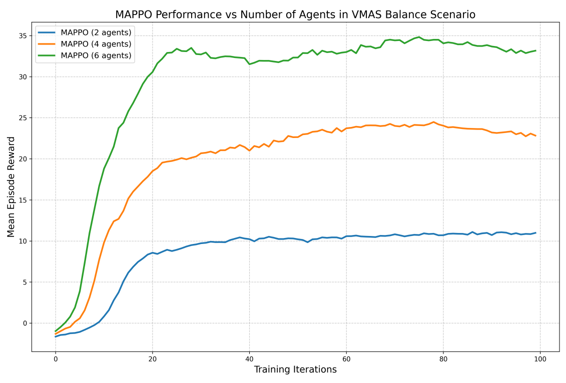
- **Mismatch Between Policy and Critic**: Although the critic has access to global information, each agent's policy only conditions on its local observation. This disconnect limits the policy' s ability to effectively leverage the global credit assignment provided by the critic.
- **Insufficient Coordination**: Without parameter sharing or explicit communication, agents may develop inconsistent or conflicting behaviors, undermining the collective stability required in the Balance task.

Consequently, despite its theoretically sound centralized training framework, MAPPO struggles to learn effective joint behaviors when fine-grained physical coordination is essential.

Task–Algorithm Compatibility

The Balance task fundamentally requires *tight physical coordination* among agents to simultaneously stabilize a shared object and navigate toward a goal. This places a premium on global consistency and synchronized action, which aligns best with CPPO’ s fully centralized learning paradigm. While IPPO approximates coordination through parameter sharing, and MAPPO attempts to guide decentralized policies via a global critic, neither matches the representational capacity and learning efficiency of CPPO in this specific domain.
In summary, the observed performance ordering reflects how well each algorithm’ s design matches the coordination demands of the task: CPPO’ s end-to-end centralized approach excels, IPPO achieves reasonable coordination through policy homogeneity, and MAPPO’ s decoupled policy–critic structure proves suboptimal under stringent coordination requirements.

Comparison of Different Agent Numbers Under the MAPPO Algorithm



Figur 3: raining curves for Different Agent Numbers

Tabel 2: Performance Tests Comparison of MAPPO with Different Numbers of Agents (10 episodes each)

Configuration	**Mean Reward**	**Standard Deviation**
MAPPO_2_agents	22.72	2.73
MAPPO_4_agents	51.93	2.45
MAPPO_6_agents	70.18	3.58

In the cooperative *balance* task—where multiple agents must jointly stand on a movable and rotatable horizontal bar (a lever system) to lift and transport a heavy package to a randomly generated goal location—an intriguing phenomenon emerges: **performance improves as the number of agents increases**. This counterintuitive result contrasts with common multi-agent reinforcement learning (MARL) intuition, which often assumes that more agents imply harder coordination. Below, we explain why more agents lead to higher success rates in this specific setting.

Reason 1: Enhanced Mechanical Stability (Core Reason)

The bar functions as a lever. When the package is off-center, it generates a tipping moment that can cause the system to collapse if not properly counterbalanced.

- **More agents $\Rightarrow$ denser support points $\Rightarrow$ shorter moment arms**.
- With only **2 agents** (at the ends), any off-center load creates large torque that is difficult to compensate for—leading to high sensitivity and frequent failures.
- With **6 agents**, the support becomes nearly continuous. Even if the package shifts left, multiple agents on the right can collaboratively apply fine-grained corrective forces.

As a result:

- The system becomes significantly more robust against tipping.
- "Ground contact" (failure) events decrease dramatically.
- The sparse reward problem is alleviated, yielding more consistent and informative training signals.

Reason 2: Increased Actuation Redundancy Lowers Policy Search Difficulty

Each agent outputs a 2D continuous action (thrust vector). The system is *over-actuated*: the minimal number of agents required to complete the task is less than the actual number deployed.

- More agents $\Rightarrow$ higher redundancy in the action space $\Rightarrow$ a larger set of equivalent solutions (different action combinations achieving the same physical outcome).
- This redundancy makes the policy landscape smoother, stabilizing policy gradient updates and reducing susceptibility to poor local minima.
- In contrast, a 2-agent system operates near the *critically actuated* regime, where tiny errors cause failure—making policy learning extremely challenging.

Reason 3: Centralized Critic Benefits from Richer Global Observations

The implementation uses the standard MAPPO architecture:

- **Decentralized Actors**: Each agent acts based on its local observation (realistic and scalable).
- **Centralized Critic**: Has access to the full joint observation of all agents during training.

With more agents:

- The critic receives observations from more points along the bar, providing a more complete picture of the system’ s pose, velocity, and balance state.
- It can more accurately predict imminent instability (e.g., tipping risk).
- Advantage estimates become more reliable, leading to better policy update directions.

Reason 4: Reward Structure Naturally Favors Collective Coordination

The reward function is defined as:

$r = r_{\text{shaping}}(\text{distance from package to goal}) + r_{\text{fall}}(\text{penalty if package touches ground})$

Crucially, this reward is **shared globally** among all agents.

- In larger teams, random exploration by any single agent has a smaller impact on the collective trajectory (due to averaging effects).
- The team is more likely to stochastically discover successful trajectories that avoid falling while moving toward the goal.
- This triggers a positive feedback loop: early successes reinforce coordinated behaviors faster.

Contrast with General MARL Intuition

It is commonly believed that “more agents $\Rightarrow$ harder coordination.” This holds true in scenarios such as:

- Competitive or adversarial tasks,
- Fully decentralized systems with no communication,
- Environments with sparse, non-shared, or individual-specific rewards.

However, in the *balance* task:

- Agents are **fully cooperative** (shared reward),
- They are **strongly physically coupled** (system state is highly correlated),
- The core challenge is **collective stability**, not strategic interaction or credit assignment.

Thus, in this context, increasing the number of agents acts as an **enhancement of system resources** rather than a coordination burden—directly contributing to higher success rates under MAPPO.

7 Improvements to the MAPPO algorithm

The original critic network in MAPPO employs a standard multilayer perceptron (MLP), which simply concatenates observations from all agents and processes them through fully connected layers. This approach fails to effectively capture inter-agent dependencies. To address this limitation, MAPPO-SA introduces an improved critic network named `CentralizedCriticSA`, which incorporates a self-attention mechanism. The implementation is as follows:

```python
class CentralizedCriticSA(nn.Module):
    def __init__(self, obs_dim, n_agents, hidden_dim=256,
        n_heads=4):
        super().__init__()
        self.n_agents = n_agents
        self.obs_embed = nn.Linear(obs_dim, hidden_dim)
        self.attn = nn.MultiheadAttention(
            embed_dim=hidden_dim,
            num_heads=n_heads,
            batch_first=True
        )
        self.out = nn.Sequential(
            nn.Linear(hidden_dim, hidden_dim),
            nn.Tanh(),
            nn.Linear(hidden_dim, 1)
        )
```

Necessity of the Improvement

1. Intrinsic Requirements of Multi-Agent Systems In the "balance" scenario, four agents must collaboratively stabilize a suspended plank. Each agent's action influences the global equilibrium state—not merely its own local state. This implies:

- Complex interdependencies exist among agents.
- Agents must adapt their policies based on the positions and actions of others.
- Traditional MLPs cannot effectively model such dependencies.

2. Limitations of the Original Centralized Critic The baseline MAPPO uses a `MultiAgentMLP` as its critic network, which suffers from several key drawbacks:

- It naively concatenates all agents' observations, ignoring relational structures among them.
- It cannot dynamically assign different importance weights to different agents.
- Symmetric agents (e.g., those at symmetric positions in the balance task) are treated identically, even though their actual contributions to system stability may differ depending on context.

Advantages of the Proposed Improvement

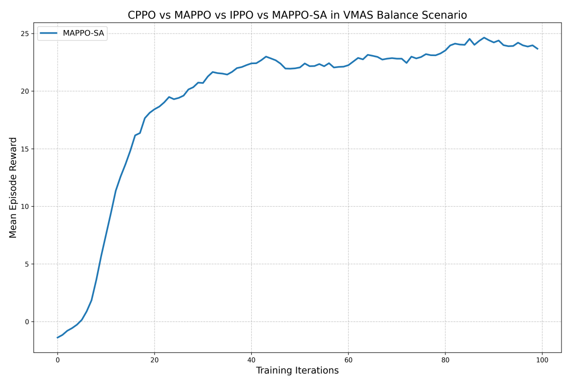
1. Benefits of the Self-Attention Mechanism The self-attention mechanism offers several critical advantages:

- **Dynamic correlation modeling**: It computes pairwise similarities between agents' observations, enabling context-dependent weighting of each agent' s influence.
- **Long-range dependency capture**: It can model non-local interactions among agents, regardless of their physical or index-based proximity.
- **Scalability to variable agent counts**: The architecture naturally accommodates varying numbers of agents without structural changes.
- **Spatial relationship awareness**: In the "balance" task, relative positions significantly affect system dynamics; self-attention can implicitly encode such spatial information.

2. Specific Advantages in the "Balance" Scenario In the "balance" environment, the self-attention mechanism in MAPPO-SA provides concrete benefits:

- It identifies agents with greater influence on balance (e.g., those positioned at the plank' s ends).
- It dynamically adjusts attention weights for symmetric agents—for instance, when the plank tilts left, it assigns higher relevance to right-side agents for corrective action.
- It leverages global state information more effectively than methods relying solely on local observations.
- Through multi-head attention, it captures diverse interaction patterns from multiple representational subspaces simultaneously.

Result Analysis



Figur 4: Training curves for improved MAPPO

The comparison between the current training curve and those obtained under the previous setting with four agents using MAPPO reveals only marginal differences at first glance. However, upon closer inspection, it becomes evident that incorporating the self-attention mechanism into the network architecture leads to a notably more stable training process—characterized by fewer performance drops during learning.

In ten independent evaluation runs, the policy achieved a mean return of Mean = 55.83 with a standard deviation of Std = 7.30. Both the mean and the standard deviation are higher than those observed before the architectural modification, suggesting an overall improvement in performance. Nevertheless, the increased standard deviation appears to contradict the earlier observation of enhanced training stability.

We hypothesize that this discrepancy likely stems from insufficiently thorough testing or training procedures—such as limited random seeds, inadequate environment diversity, or premature termination of training. A more rigorous experimental protocol would be required to fully reconcile these observations. However, given the scope of this study, we do not delve deeper into this inconsistency here.

References

1. Bettini, M., Kortvelesy, R., Blumenkamp, J., & Prorok, A. (2022). VMAS: A Vectorized Multi-Agent Simulator for Collective Robot Learning. arXiv preprint.
2. PyTorch Team. (2022). Multi-Agent PPO Tutorial. PyTorch Tutorials.

Appendix: Source Code Repository

The complete source code for the experiments described in this report is publicly available on GitHub:

`https://github.com/shy-57/Reinforcement-Learning/tree/main/Final_Project`