Apply three MARL algorithm in VMAS

Zhen Wu
Sun Yat-Sen University, China

Abstract

Multi-Agent Reinforcement Learning(MARL) is gaining increasing attention in the robotics community as a promising solution to tackle multi-robot coordination problems.We apply three MARL algorithm in Vectorized Multi-Agent Simulator(VMAS). It is comprised of a vectorized 2D physics engine written in PyTorch and a set of twelve challenging multi-robot scenarios. Additional scenarios can be implemented through a simple and modular interface[1].The three MARL algorithm are CPPO, MAPPO(Chao Yu, 2021) and IPPO(Christian Schroeder de Witt, 2020).We apply those three MARL algorithms in serval games provided by VMAS benchmark.Besides, I have improved IPPO.I also compare three algorithms in different games;the results suggest that MAPPO's strong performance may be due to its robustness to some forms of environment nonstationarity.Also my experiments demonstrate its robust performance on a wide variety of task: balance, wheel and reverse transport.

Code is available at:
`wuzh77/MARLinVMAS`

1. Introduction

Many real-world problems require coordination of multiple robots to be solved.However, coordination problems are commonly computationally hard.Examples include pathplanning[12], task assignment[15], and area coverage[23].Multi-Agent Reinforcement Learning (MARL) can be used as a scalable approach to find near-optimal solutions to thes problems[21].All of the algorithms I use are based on PPO algorithm[11].PPO is a novel objective with clipped probality ratios, which forms a pessomistic estimate(i.e., lower bound) of the performance of the policy[10].

Most algorithms for policy optimization can be classified into three broad categories:(1)policy iteration methods, which alternate between estimating the value function under the current policy and improving the policy (Bertsekas, 2005);(2)policy gradient methods, which use an estimator of the gradient of the expected return (total reward) obtained from sample trajectories (Peter & Schaal, 2008a)(and which, as we later discuss, have a close connection to policy iteration);and (3) derivative-free optimization methods, such as the cross-entropy metho (CEM) and covariance matrix adaptation (CMA), which treat the return as a black box function to be optimized in terms of the policy parameters(Szita,2006)[9].

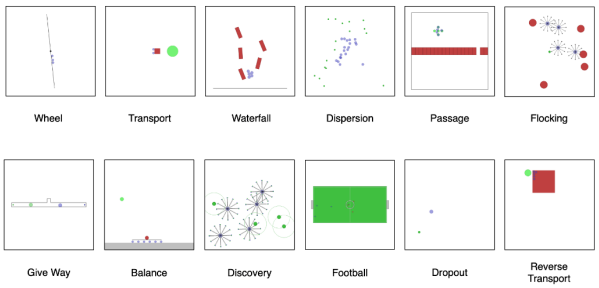


Figure 1 – As shown in the left image, VMAS(Matteo Bettini, 2022) include many multi-robot scenarios. Scenario creation is made simple and modular to incentivize contributions.VMAS simulates agents and landmarks of different shapes and supports rotations, elastic collisions, joints, and custom gravity.VMAS has an interface compatible with OpenAI Gym(Greg Brockman, 2016) and with the RLib library, enabling out-of-the-box integration with a wide range of RL algorithms.

In this paper we use VMAS, a vectorized multi-agent simulator.VMAS is a vectorized 2D physics simulator written in Pytorch[14], designed for effient MARL benchmarking.The models compared are:

- **CPPO:**This model uses a centralized critic and actor.It treats the multi-agent problem as a single-agent problem with one super-agent.
- **MAPPO[22]:**This model uses a centralized critic and a decentralized actor.Therefore,the agents act independently,with local decentralized policies,but are trained with centralized information.
- **IPPO[2]:**This model uses a decentralized critic and actor. Every agent learns and acts independently. Model parameters are shared among agents so they can benefit from each other's experiences.

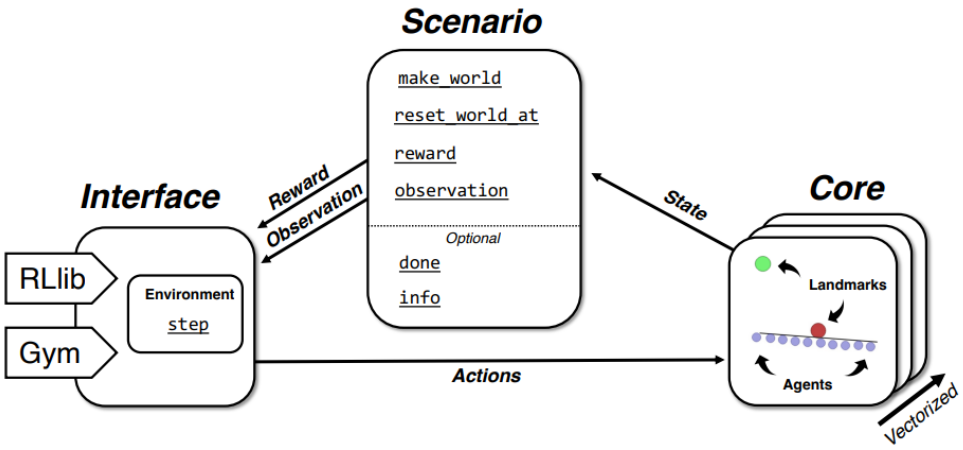


Figure 2 – VMAS structure.VMAS has a vectorized MARL interface (left) with wrappers for compatibility with OpenAI Gym[4] and the RLlib RL library[5].The default VMAS interface uses PyTorch and can be used for feeding input already on the GPU.Multi-agent tasks in VMAS are defined as scenarios (center).To define a scenario, it is sufficient to implement the listed functions. Scenarios access the VMAS core (right), where agents and landmarks are simulated in the world using a 2D custom written physics module.

2. Related Work

Actor-critic algorithms. Actor-critic algorithms have been shown to benefit from learning centralized joint critics alongside decentralised policies[7].As the critics are not needed during execution, this approach is inherently decentralizable.COMA extends this approach with a counterfactual multi-agent critic baseline based on temporal difference errors in order to facilitate multi-agent credit assignment. Joint Q-learning can also be made decentralizable.Value Decomposition Networks [20] decompose joint state-action value functions into sums of decentralised utility functions that can be used during greedy execution.QMIX [16] extends this additive decomposition to arbitrary centralized monotonic mixing networks. Both centralized joint critics and factored joint value functions can reap some benefits of centralized joint learning, while bypassing the joint action space explosion, imposing an effective prior on multi-agent credit assignment, and mitigating practical learning pathologies associated with centralized joint learning on popular benchmark environment StarCraft II [18].

MARL algorithms. MARL algorithms generally fall between two frameworks: centralized and decentralized learning.Centralized methods [3, Caroline & Craig] directly learn a single policy to produce the joint actions of all agents. In decentralized learning, each agent optimizes its reward independently; these methods can tackle general-sum games but may suffer from instability even in simple matrix games[6].Centralized training and decentralized execution (CTDE) algorithms fall in between these two frameworks. Several past CTDE methods [17] adopt actor-critic structures and learn a centralized critic which takes global information as input. Value-decomposition (VD) methods are another class of CTDE algorithms which represent the joint Q-function as a function of agents' local Q-functions [20] and have established state of the art results in popular MARL benchmarks.

3. Our Approach

In this section, I first introduce three MARL algorithms(CPPO, IPPO, MAPPO). Next, I will show the pseudocode of each MARL algorithms. Also, I will explain why I use VMAS as a MARL benchmark.Finally, I present my proposed method of improving IPPO algorithm.

3.1. MAPPO algorithm

Preliominaries I study decentralized partially observable Markov decision processes (DEC-POMDP)with share.d rewards.A DEC-POMDP is defined by $< \mathcal{S}, \mathcal{A}, O, R, P, n, \gamma >$.$\mathcal{S}$ is the state space.$\mathcal{A}$ is the shared action space for each agent i.$o_i = O(s; i)$ is the local observation for agent i at global state s.$p(\hat{s}|s, A)$ denotes the transition probability from s to $\hat{s}$ given the joint action $A = (a_1, a_2, \cdots, a_n)$ for all n agents. $R(s, A)$ denotes the shared reward function.γ is the discount factor.Agents use a policy $\pi_\theta(a_i|o_i)$ parameterized by θ to produce an action a_i from the local observation o_i, and jointly optimize the discounted accumulated reward $J(\theta) = \mathbb{E}_{A^t, s^t}[\sum_t \gamma^t R(s^t, A^t)]$ where $A^t = (a_1^t, \cdots, a_n^t)$ is the joint action at time step t.

Realization Our implementation of PPO in multi-agent settings closely resembles the structure of PPO in single agent settings by learning a policy π_θ and a value function $V_\phi(s)$; these functions are represented as two separate neural networks. $V_\phi(s)$ is used for variance reduction and is only utilized during training; hence, it can take as input extra global information not present in the agent's local observation, allowing PPO in multi-agent domains to follow the CTDE structure.Suppose there are $n(2 \leq n)$ agents, the input state of agent i is s_i.So the input of $V_\phi(s)$ is s_i and the $V_\phi(s)$ outputs the value of state s_i.The rest are the same as PPO.The algorithm details are as follows.

Algorithm 1 Recurrent-MAPPO.

Initialize θ, the parameters for policy π and ϕ, the parameters for critic V, using Orthogonal initialization
Set learning rate α
while $step \leq step_{max}$ **do**
  set data buffer $D = \{\}$
  for $i = 1$ to batchsize **do**
    $\tau = []$ empty list
    initialize $h_{0,\pi}^{(1)}, \cdots, h_{0,\pi}^{(n)}$ actor RNN states
    initialize $h_{0,V}^{(1)}, \cdots, h_{0,V}^{(n)}$ critic RNN states
    for $t = 1$ to T **do**
      for all agents a **do**
        $p_t^{(a)}, h_{t,\pi}^{(a)} = \pi(o_t^{(a)}, h_{t-1}^{(a)}; \theta)$
        $u_t^{(a)}$ $p_t^{(a)}$
        $v_t^{(a)}, h_{t,V}^{(a)} = V(s_t^{(a)}, h_{t-1}^{(a)}; \phi)$
      end for
      Execute actions u_t, observe r_t, s_{t+1}, o_{t+1}
      $\tau += [s_t, o_t, h_{t,\pi}, h_{t,V}, u_t, r_t, s_{t+1}, o_{t+1}]$
    end for
    Compute advantage estimate $\hat{A}$ via GAE on ϕ, using PopArt
    Compute reward-to-go $\hat{R}$ on τ and normalize with PopArt Split trajectory τ into chunks of length L
  end for
  for mini-batch $k = 1, \cdots, K$ **do** b $\leftarrow$ random mini-batch from D with all agent data
    for each data chunk c in the mini-batch b **do**
      update RNN hidden states for π and V from first hidden state in data chunk
    end for
  end for
  Adam update θ on $L(\theta)$ with data b
  Adam update ϕ on $L(\phi)$ with data b
end while

3.2. IPPO algorithm

Background We consider a fully cooperative multi-agent task A decentralised partially observable Markov decision process describes multi-agent tasks where a team of cooperative agents chooses sequential actions under partial observability and environment stochas ticity.Dec-POMDPs can be formally defined by a tuple $< \mathcal{N}, \mathcal{S}, \mathcal{U}, P, r, \mathcal{Z}, O, \rho, \gamma >$. Here $s \in S$ describes the state of the environment, discrete or continuous, and $N := \{1, \cdots, N\}$ denotes the set of N agents. s_0 ρ, the initial state, is drawn from distribution ρ. At each time step t, all agents $a \in \mathcal{N}$ simultaneously choose actions $u_t^a \in \mathcal{U}$ which may be discrete or continuous. This yields the joint action $u_t := \{u_t^a\}_{a=1}^N$. The next state s_{t+1} $P(s_t, u_t)$ is drawn from transition kernel P after executing the joint action u_t in state s_t. Subsequently, the agents receive a scalar team reward $r_t = r(s_t, u_t)$.

Realization To achieve the IPPO algorithm, we use PPO to learn decentralized policies π^a for agents with individual policy clipping where each agent's independent policy updates are clipped based on the objective defined in Equation 1.

$$\mathcal{L}(\theta) = \mathbb{E}_{s_t,u_t}[min(\frac{\pi_{\theta(u_t|s_t)}}{\pi_{\theta_{old}}(u_t|s_t)}A(s_t,u_t), clip(\frac{\pi_\theta(u_t|s_t)}{\pi\theta_{old}(u_t|s_t)}, 1-\epsilon, 1+\epsilon)A(s_t,u_t))] \tag{1}$$

We consider a variant of the advantage function based on independent learning, where each agent a learns a local observation based critic $V_\phi(z_t^a)$ parameterised by ϕ using Generalized Advantage Estimation(GAM)[19] with discount factor $\gamma = 0.99$ and $\lambda = 0.95$.The network parameters ϕ, θ are shared across critics, and actors, respectively.We also add an entropy regularization term to the final policy loss[13].For each agent a, we have its advantage estimation as follows:

$$A_t^a = \sum_{l=0}^{h} (\gamma\lambda)^l \delta_{t+l}^a \tag{2}$$

where $\delta_t^a = r_t(z_t^a, u_t^a) + \gamma V_\phi(x_{t+1}^a) - V_\phi(z_t^a)$ is the TD error at time step t.I use the team reward $r_t(s_t, a_t)$ to approximate $r_t(z_t^a, u_t^a)$. The final policy loss for each agent a becomes:

$$\mathcal{L}^\alpha(\theta) = \mathbb{E}_{z_t^a,u_t^a}[min(\frac{\pi_{\theta(u_t^a|s_t^a)}}{\pi_{\theta_{old}}(u_t^a|s_t^a)}A_t^a, clip(\frac{\pi_\theta(u_t^a|s_t^a)}{\pi\theta_{old}(u_t^a|s_t^a)}, 1-\epsilon, 1+\epsilon)A_t^a)] \tag{3}$$

Essentially, IPPO is consisting of n agents using PPO.The input to my NNcomprises of stacked obervations for the past few times steps, which is pass through three linear layers and Tanh activations.We use a discount factor of γ = 0.99, a learning rate of 0.00005 ,and clipping parameter 0.2. The algorithm details are as follows.

Algorithm 2 IPPO.

Initial θ, the parameters for policy π and ϕ, the parameters for critic V for each agent.
set learning rate α
while $step \leq step_{max}$ **do**
    All agents interact with the environment to obtain their own track data.
    For each agent, GAE is used to calculate the estimation of the dominance function based on the current value function.
    For each agent, update its policy by maximizing its PPO-clipped target.
    For each agent, its value function is optimized by mean square error loss function.
end while

3.3. CPPO algorithm

Background CPPO use a centralized critic and actor.It treats the multi-agent problem as a single-agent problem with one super-agent.In this paper,I use $< \mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{R}, P >$ to describe a multi-agents problem.N denotes the number of agents.$S = S_1 \times S_2 \times S_3 \cdots S_N$ denotes state set of all agents.$\mathcal{A} = A_1 \times A_2 \times \cdots A_N$ denotes action set of all agents.$\mathcal{R} = r_1 \times \cdots r_N$ denotes reward function set of all agents.The goal of general multi-agent reinforcement learning is to learn a strategy for each agent to maximize its own cumulative rewards.

Realization CPPO is just a super agent that receive all states of agents and return the actions of all agents.To realize CPPO algorithm, I just stack the states of oberservations and return each agent's action.For example, we have two agents play wheel games provided by VMAS.The dimension of each agent's state is 13 and action dimension is 2.We name two agents A and B.So the input of CPPO is $< \mathcal{S}_A, \mathcal{S}_B >$ and the output of the CPPO is $< \mathcal{A}_A, \mathcal{A}_B >$.In fact, CPPO is PPO expending the input and output dimensions.

3.4. A Vectorized Multi-Agent Simulator

A Vectorized Multi-Agent Simulator is a vectorized 2D physics simulator written in PyTorch, designed for efficient MARL benchmarking. Vectorization in PyTorch allows VMAS to perform simulations in a batch, seamlessly scaling to tens of thousands of parallel environments on accelerated hardware. [1]With the term GPU vectorization we refer to the Single Instruction Multiple Data (SIMD) execution paradigm available inside a GPU warp.VMAS supports inter-agent communication and customizable sensors, such as LIDARs.This platform has the following advantages:

- **Vectorized:**VMAS vectorization can step any number of environments in parallel.
- **Simple:**Complex vectorized physics engines exist [8], but they do not scale efficiently when dealing with multiple agents.This defeats the computational speed goal set by vectorization. VMAS uses a simple custom 2D dynamics engine written in PyTorch to provide fast simulation.
- **General:**The core of VMAS is structured so that it can be used to implement general high-level multi-robot problems in 2D.
- **Extensible:**VMAS is not just a simulator with a set of environments. It is a framework that can be used to create new multi-agent scenarios in a format that is usable by the whole MARL community.
- **Compatible:** VMAS has multiple wrappers which make it directly compatible with different MARL interfaces.

In general, VMAS can step any number of environments in parallel.Besides, GPU acceleration is available.Its speed is much faster than other MARL algorithm benchmark.

3.5. Improvement

I now list the main contributions of this work:

- I use a vectorized multi-agent simulator vectorization which can step any number of environments in parallel and use GPU to accelerate.
- While training, using variance decay to modify Gaussian kernel function.As the training goes on, this method can enable agents to perform functions with high sampling probability.
- Add IPPO input dimension.In initial IPPO, each agent only accepts its own state vector.In this paper, I splice all state vectors as input of IPPO.Through this way each agent can observe all agents' state so that they can cooperate better.

In IPPO [2, Christian], each agent only pay attention to its own state.In this paper, each agent not only focus on its own state, but it also focus on the states of other agents.By this method, all the agents in the same environment can pay attention to all agents' states so that they can cooperate better.But this is different from CPPO, because the critic network is not shared and each agent takes actions independently.And I use VMAS platform to run parallel environments useing GPU.I can get the result of my algorithm quickly.Finally, I use variance decay to modify Gaussian kernel function in the in the select action section.In this way, along with the experimental training, the agent will prefer to choose more rewarding actions.

4. Experiments

My experiments are conducted on the Vectorized Multi-Agent Simulator (VMAS)[1].There a set of 12 multi-robot scenarios in VMAS. These scenarios contain various multi-robot problems, which require complex coordination.Each scenario delimits the agents' input by defining the set of their observations. This set typically contains the minimum observation needed to solve the task (e.g., position, velocity, sensory input, goal position).I apply three algorithms and improved IPPO to the games named balance and wheel.

- **Wheel:**N agents have to collectively rotate a line. The line is anchored to the origin and has a parametrizable mass and length. The team's goal is to bring the line to a desired angular velocity.Each agent observes its position, velocity, the current angle of the line module pi, the absolute difference between the current angular velocity of the line and the desired one, and the relative position to the two line extrema.
- **Balance:**N agents are spawned at the bottom of a world with vertical gravity.A line is spawned on top of them. The agents have to transport a spherical package, positioned randomly on top of the line, to a given goal at the top.The observations for each agent are: its position, velocity, relative position to the package, relative position to the line, relative position between package and goal, package velocity, line velocity, line angular velocity, and line rotation mod pi.

Parameter setting. Unless specified, our actor and critic network both use three linear layers and tanh function as activate function.Max timesteps in one episode is 200.The max training timesteps is 10^6 in balance and $2 * 10^6$ in wheel.The starting std for action distribution is 0.6.In this paper I use linearly decay action std with action std decay rate 0.05.The minimum action std is 0.1.The frequency of updating policy is every 800 timesteps.Clip parameter of PPO is 0.2.And γ is 0.99.The learn rate of actor and critic are both 0.00005.The parameters setting is shown in Table 1.

parameter	value
Network type	Linear layer
Number of layers	2
max timesteps	200
max training timesteps	10^6
start action std	0.6
std decay rate	0.05
min action std	0.1
clip	0.2
γ	0.99
actor learn rate	0.00005
critic learn rate	0.00005
environment1	balance
environment2	wheel

Table 1 – This table describe the common hyperparameters for three algorithm.

Training Details. I firstly forward the environment observation state through the network and obtain the agents' action and value function.Then I send the actions to the environment to get the next states and rewards.When training, the $RolloutBuffer$ will continuously save the states, actions and rewards.Those $< \mathcal{S}, \mathcal{A}, \mathcal{R} >$ saved in the $RolloutBuffer$ will be taken out for updating actor-critic network.I will adjust variance periodically between training.

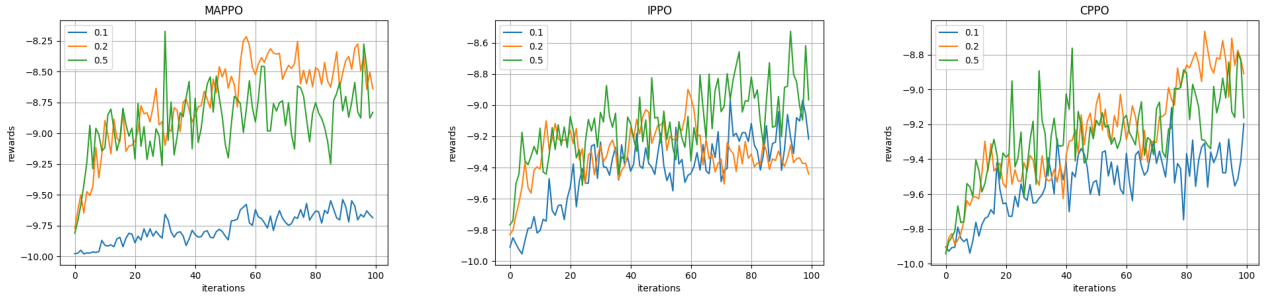
There are some differences between three algorithms.In CPPO, I splice the observations of all agents together.While in IPPO, I create n agents and each agent receives its own observation.MAPPO algorithm is a little different.Each agent receives different states, but when updating network it should splice all observations of agents then send it to the critic network.

In the environment parameters setting, there are 2 agents in all algorithms.I set 300 parallel environments in balance and wheel.To accelerate my training, I use GPU to building environments and training.In the environment, the observation of agent is not two-dimensional, so I use MLP network architectures.The action space of two environments is continuous action space.During the experiment, I find that continuous action is better than discrete action in complex environment.So that's why I use continuous action.

Another core feature of PPO is the use of clipped importance ratio and value loss to prevent the policy and value functions from drastically changing between ieration.Clipping strength is controlled by the ϵ hyperarameter:large ϵ values allow for larger updates to the policy and value function.For small ϵ, agents' policies are likely to change less per update, which we posit improves overall learning stability at the potential expense of learning speed.The result is shown in Figure 3.For the best PPO performance, maintain a clipping ratio ϵ under 0.2;within this range tune ϵ as a trade-off between training stability and fast convergence.

	MAPPO	CPPO	IPPO	IPPO$_{improve}$
Balance	60	80	60	80
Wheel	-8.0	-8.5	-9.0	-8.5

Table 2 – The table describe the final approximate reward after convergence.In the table,we can find that in wheel game MAPPO performs best.In balance, the result of IPPO and CPPO almost the same.



(a) (b) (c)

Figure 3 – Effect of different clipping strengths on MAPPO's performance in SMAC.The picture is from MAPPO paper[22, Chao Yu]. For the best PPO performance, maintain a clipping ratio ϵ under 0.2;within this range tune ϵ as a trade-off between training stability and fast convergence.I change a clipping ratio ϵ to 0.1,0.2 and 0.5.All experiments run in wheel games.In picture (a), $\epsilon = 0.2$ is best for MAPPO.And $\epsilon = 0.2$ is similar to the result of 0.5.But the rewards curve of $\epsilon = 0.2$ is more stable.In the experiment of PPO, $\epsilon = 0.5$ show better performance.While in CPPO, $\epsilon = 0.2$ and $\epsilon = 0.5$ are basically the same.According to the above experiment, the results suggest that when ϵ is high, the rewards curve has a great changes.But when ϵ is low, the rewards curve rises more slowly and steadily.That is because for small ϵ, agents' policies are likely to change less per update, which we posit improves overall learning stability at the potential expenseof learning speed.

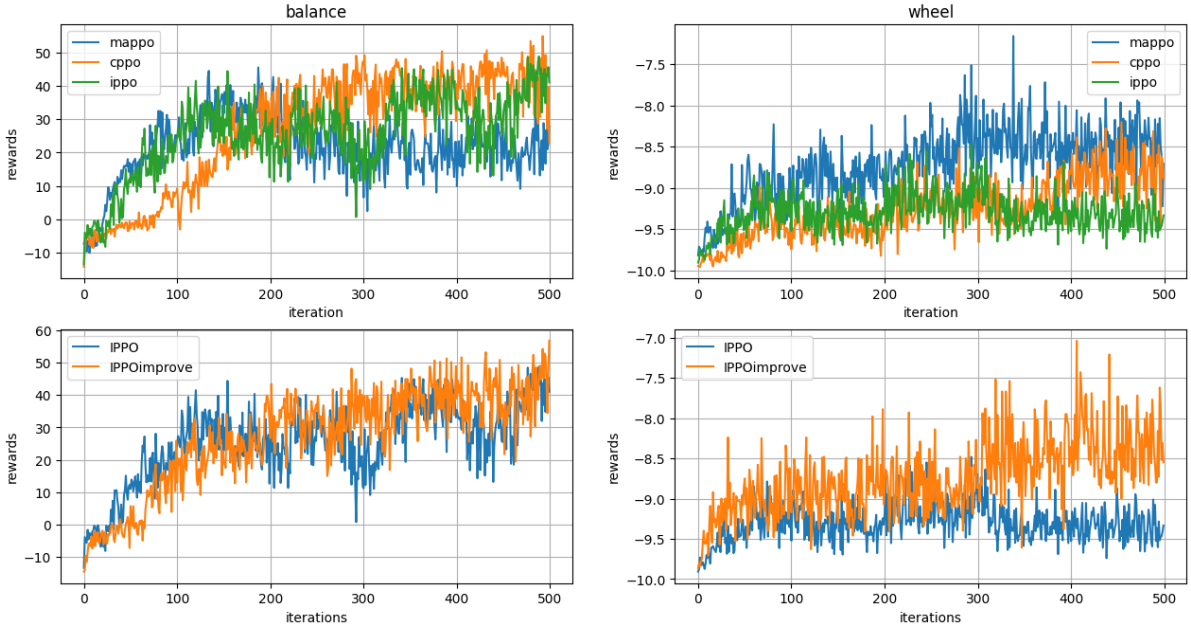


Figure 4 – The figure describe three different algorithms return curve.Picture in the upper left corner shows the result of three algorithms in balance.The curves show that MAPPO algorithm perform don't perform very well.CPPO and IPPO performe similarly.The picture in the upper right corner describes the result of three different algorithms in wheel.The results suggest that MAPPO has better performance.The following two pictures show the result of IPPO and IPPO improved.To show the improvement results, I put IPPO and it together separately.We can find that IPPO improved is more stable than IPPO.Besides, IPPO improved in wheel performs much better than IPPO.

5. Conclusion

This work demostrates that PPO, an on-policy gradient RL algorithm,achieves strong results in both final returns and sample efficiency that are comparable to the state-of-the-art methods on a variety of cooperative multi-agent challenges, which suggests that properly configured PPO can be a competitive baseline for cooperative MARL tasks.I also use PPO algorithms to build CPPO, IPPO and MAPPO which show excellent results in VMAS.I run a set of training experiments to benchmark the performance of MARL algorithms on 2 VMAS scenarios.Thanks to VMAS, I can run the environment using GPU to accelerate, so that I can get the results of my MARL algorithms more quickly. The runs reported in this section all took under 3 hours to complete.All the models compared are all based on Proximal Policy Optimization, an actor-critic RL algorithm.And all models use full connection layer to build network.In the experiment, I find that IPPO perform better in balance, while MAPPO perform better in wheel.And CPPO algorithm is more stable.MAPPO converges faster.

Improvement In this paper, I propose a method which enter the observations of multiple agents to IPPO.As shown in Fig4, the method outperforms ordinary IPPO.In multi-agent collaboration problems, this method enable each agents obtain other agents' observations, so that they can cooperate better.But this way increases the weight of training, compared with IPPO.

References

[1] Matteo Bettini, Ryan Kortvelesy, Jan Blumenkamp, and Amanda Prorok. Vmas: A vectorized multi-agent simulator for collective robot learning. *The 16th International Symposium on Distributed Autonomous Robotic Systems*, 2022.

[2] Denys Makoviichuk Viktor Makoviychuk Philip H.S. Torr Mingfei Sun Shimon Whiteson Christian Schroeder de Witt, Tarun Gupta. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

[3] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998(746-752):2, 1998.

[4] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[5] Philipp Moritz Robert Nishihara Roy Fox Ken Goldberg Joseph E. Gonzalez Michael I. Jordan Ion Stoica Eric Liang, Richard Liaw. Rllib: Abstractions for distributed reinforcement learning. *arXiv preprint arXiv:1712.09381*, 2017.

[6] Jakob N. Foerster, Richard Y. Chen, Maruan Al-Shedivat, Shimon Whiteson, P. Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. *ArXiv*, abs/1709.04326, 2017.

[7] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *ArXiv*, abs/1705.08926, 2017.

[8] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021.

[9] Philipp Moritz Michael Jordan Pieter Abbeel John Schulman, Sergey Levine. Trust region policy optimization. *arXiv preprint arXiv:1502.05477*, 2015.

[10] Prafulla Dhariwal Alec Radford Oleg Klimov John Schulman, Filip Wolski. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[11] Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`, 2018.

[12] Qingbiao Li, Fernando Gama, Alejandro Ribeiro, and Amanda Prorok. Graph neural networks for decentralized multi-robot path planning. *arXiv preprint arXiv:1912.06095*, 2019.

[13] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *ArXiv*, abs/1602.01783, 2016.

[14] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[15] Amanda Prorok. Robust assignment using redundant robots on transport networks with uncertain travel time. *IEEE Transactions on Automation Science and Engineering*, 17(4):2025–2037, 2020.

[16] Tabish Rashid, Mikayel Samvelyan, C. S. D. Witt, Gregory Farquhar, Jakob Nicolaus Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *J. Mach. Learn. Res.*, 21:178:1–178:51, 2020.

[17] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

[18] Mikayel Samvelyan, Tabish Rashid, C. S. D. Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob N. Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *ArXiv*, abs/1902.04043, 2019.

[19] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.

[20] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech M. Czarnecki, Vinícius Flores Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning. In *Adaptive Agents and Multi-Agent Systems*, 2017.

[21] Binyu Wang, Zhe Liu, Qingbiao Li, and Amanda Prorok. Mobile robot path planning in dynamic environments through globally guided reinforcement learning. *IEEE Robotics and Automation Letters*, 5(4):6932–6939, 2020.

[22] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games, 2021.

[23] Xiaoming Zheng, Sven Koenig, David Kempe, and Sonal Jain. Multirobot forest coverage for weighted and unweighted terrain. *IEEE Transactions on Robotics*, 26(6):1018–1031, 2010.