Adapting to Robot Malfunctions: A Multi-Agent Reinforcement Learning Approach

September 26, 2024

Hunter M. Hasenfus

Mentor

Dr. Reza Ahmadzadeh, Miner School of Computer Information Sciences

Committee Member

Yasin Findik, Miner School of Computer Information Sciences

Miner School of Computer Information Sciences
University of Massachusetts: Lowell
HONORS PROJECT

Hunter M Hasenfus
Date:
Mathematics and Computer Science

1 Abstract

This project delves into reinforcement learning, a learning paradigm focusing on how agents predict and interact within their environments. We are exploring multi-agent reinforcement learning, which, compared to single-agent scenarios, introduces unique complexities. One of the major challenges in single-agent contexts is choosing between value-based and policy-based learning algorithms. Notably, the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) approach has shown promise in multi-agent settings and is adept at handling continuous environments. Meanwhile, value-based algorithms have not been adapted for continuous domains. Mixed Q-Functionals (MQF) is a value-based algorithm extended into the continuous domain. The aim is to analyze how dynamics, reward structures, and other properties of multi-agent reinforcement learning function in continuous environments. The core of this project is to evaluate MQF in these settings and benchmark its performance against that of MADDPG across various continuous environments.

2 Introduction

In the evolving landscape of robotics, reinforcement learning algorithms stand at the forefront, driving advancements with their capacity to adapt and learn from dynamic environments. The evolution of algorithmic development has lent itself to a comprehensive taxonomy that delineates reinforcement learning algorithms into classifications based upon various properties and means of achieving the intended outcome. This paper investigates the comparative performance of value-based versus policy-based reinforcement learning algorithms in continuous action and observation spaces, with a specific focus on robotic applications."

A focal point of this analysis is the consideration of strengths and weaknesses relative to the format of the environment upon which the algorithm is being tested. The environment can be split into two spaces: action space and observation space, each of which can take up a discrete or continuous format. Within the fully discrete domain, value-based methods are more advantageous; tabular methods suffice in these simpler environments. However, as we shift focus to the continuous domains prevalent in robotics, the inadequacy of discrete methods becomes evident, necessitating advanced algorithms capable of handling the complexities inherent in such environments. The switch from discrete to continuous can be analogized to moving from mathematical logic to geometry, the number of components approaches infinity, and thus continuity is a means of understanding overwhelming complexity through approximation. As environments grow in complexity, necessitating continuous action and observation spaces, the limitations of value-based methods become apparent, highlighting the need for function approximation to achieve generalizability and efficiency. These techniques become integral to achieving both acceptable sample efficiency and sufficient generalizability. By utilizing neural networks, RL algorithms navigate continuous spaces by approximating the value functions or policies directly, where traditional tabular methods would fail due to the curse of dimensionality.

Building off these notions, the experiments conducted fall within the realm of robotics, addressing three critical aspects of adaptability and algorithm efficiency. First, we examine agents' ability to overcome malfunctions, employing a diverse set of metrics to assess resilience against motor impairments. This investigation not only tests the robustness of the algorithms but also their capacity for recovery in disrupted environments. Second, we explore the potential for knowledge transfer among agents, particularly how an agent's experience with malfunctions can inform and enhance collective learning and behavior adaptation. This aspect evaluates the algorithms' capability to facilitate learning and adaptation in a multi-agent setting. Lastly, we delve into the scalability of these algorithms by incrementally increasing the number of agents, aiming to understand how algorithm performance and cooperation dynamics evolve in larger, more complex systems.

Conducting these experiments in a continuous action and observation space enables a nuanced comparison between value-based and policy-based algorithms. Beyond assessing their relative strengths and weaknesses, this experimental approach illuminates pathways for algorithmic improvements and provides insights into optimal algorithm selection and development for specific robotic environments.

3 Background Concepts and Related Work

3.1 Reinforcement Learning

A key conceptual framework for understanding how agents and environment interact.[1] Known as a Markov decision process (MDP) is represented by a tuple $\langle S, A, T_1, T, R\rangle$, which includes the state space S, action space A, a (possibly stochastic) reward function $R : S\times A\times S \to \mathbb{R}$, the start state distribution $T_1 \in P(S)$, and the transition function $T : S \times A \to P(S)$, where $P(X)$ denotes the set of probability distributions over the set X. The expected value of R is denoted by $\bar{R}$. The interactions of the agent with its environment produce a trajectory $(S_1, A_1, R_1, S_2, \ldots)$ where $S_1 \sim T_1$, $S_{t+1} \sim T(\cdot|S_t, A_t)$ and $R_t = R(S_t, A_t, S_{t+1})$. Each agent aims to maximize their expected cumulative discounted reward with a discount factor $\gamma, R_t := \sum_{t=1}^{\infty} \gamma^{t1} R_t$.

In an environment where the number of agents is greater than 1, various frameworks have been proposed. A fully cooperative multi-agent sequential decision-making task can be modeled as a decentralized partially observable Markov decision process (Dec-POMDP),.[2] This is formalized by the tuple $G = (S, U, P, r, Z, O, n, \gamma)$, where $s \in S$ represents the actual state of the environment. At every time step, each agent a from a set of agents $A = \{1, \ldots, n\}$ selects an action $u_a \in U$, leading to a collective action $u \in U^n$. This action induces a transition in the environment according to the state transition function $P(s'|s, u) : S\times U^n \times S \to [0, 1]$. All agents are assumed to share a common reward function $r(s, u) : S\times U \to \mathbb{R}$, and $\gamma \in [0, 1)$ serves as the discount factor. This framing is very similar to Markov Games[3]

The agent chooses actions according to a policy: a function : S → P(A) from states to probability distributions over A. An optimal policy is one which maximizes expected cumulative reward. In partially observed environments, the policy usually incorporates past agent observations from the history $h_t = a_1 o_1 r_1, ..., a_{t1} o_{t1} r_{t1}$ (replacing st).

3.2 Value-based

Value-based methods have garnered significant interest due to their advantageous characteristics for off-policy methods. Due in large part to sample efficiency and speed of convergence. The journey to where value-based methods are today is worthy of knowledge as it aids in understanding. An on-policy value function with its respective action-value function is seen below,

$$V^{\pi}(s) = E_{\tau\sim\pi}[R(\tau)\,|s_0 = s\,], \quad Q^{\pi}(s, a) = \mathbb{E}[R|s_t = s, a_t = a] \tag{1}$$

Further building off of this, the use of dynamic programming and bellman equations led to understanding implementations of TD(0), TD(λ), and Q-learning.

Q-Learning and DQN. Q-learning uses the action-value function defined above with a MSE and a epsilon-greedy action choice. In other words, the objective function is built upon the difference in action value of present policy and optimal policy, as well as the epsilon greedy correlates to a stochastic choice of maximal or random behavior. Shown below are the objective function with respect to the target function $\overline{Q}^*(s', a')$,

$$L(\theta) = \mathbb{E}_{s,a,r,s'}\left[(Q^*(s, a|\theta) - y)^2\right], y = r + \gamma \max_{a'} \overline{Q}^*(s', a'), \tag{2}$$

3.3 Policy-based

Policy-based methods are designed to optimize the policy itself, effectively learning a parameterization of the policy. While a value function may play a role in these calculations, it's not always a necessary component.

Policy-based methods can be further categorized into gradient-based and gradient-free methods. Gradient-based methods directly employ the gradient of the parameterized policy to update these parameters, while gradient-free methods achieve performance improvements through alternative techniques. In this discussion, we'll primarily focus on gradient-based algorithms.

Policy Gradient Algorithms. These algorithms directly adjust parameters θ for instrumentation in the direction of the objective function $J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}[R]$ by making updates with the gradient, $\nabla J(\theta)$. Through the policy gradient theorem which utilizes the action-value function as defined in 3.2 the gradient of the loss function can be defined as,

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)\right] \tag{3}$$

Gradient-based methods typically utilize techniques such as stochastic gradient descent or natural gradient methods to update the policy's parameters.

It is quite interesting to understand the advantages of policy-based gradient-based methods relative to value-based methods, in that the former can attain much finer-grained understanding over the action space.

3.4 Multi-agent Architecture

Within the jump from single agent to multi-agent, various approaches to training and execution have been considered. The range is between fully decentralized learning and executing agents who struggle from lack of communication, and fully centralized learning who struggle from the curse of dimensionality. There appears to be a convergence in the literature through the growing effectiveness of centralized training and decentralized execution (CTDE). During the training phase information from the entire environment is included in the learning process whereas, during execution, the individual agents have access only to their individual observations.

4 Methodology

This section discusses the algorithms used in the experiments. Four algorithms were chosen to be compared within this setting to provide a means of exemplifying the various directions of algorithmic development.

4.1 Multi-agent deep deterministic policy gradient (MADDPG)

MADDPG[4] takes advantage of the learning paradigm of centralized training and decentralized execution. Building off the concept of actor-critic,[1] learning a centralized critic that then utilizes a action-value function $Q(s;a)$ to 'critique' the agents. Each agent then learns a parameterized policy, this parameterization is achieved through the use of a deep neural network and policy updates $\nabla log_\pi(s|a)$. Within the centralized phase, the centralized action value function takes as input actions from all agents in addition to some state information and outputs values for each agent. The agents learn deterministic policies that select optimal action, analogous to a deep Q-learning for continuous action spaces. The gradient of the expected return for each agent is,

$$\nabla_{\theta_i} J(\mu_i) = \mathbb{E}_{x,a\sim D}\left[\nabla_{\theta_i}\mu_i(a_i|o_i)\nabla_{a_i} Q^{\mu_i}(x,a_1,\ldots,a_N)|_{a_i=\mu_i(o_i)}\right], \tag{4}$$

MADDPG can be considered a hybrid algorithm between policy based and value based, as it utilizes an action-value function for the critic but ultimately learns a policy. The objective function that the centralized critic optimizes is,

$$L(\theta_i) = \mathbb{E}_{x,a,r,x'}\left[\left(Q^\mu(x,a_1,\ldots,a_N) - y\right)^2\right], y = r_i + \gamma Q^{\mu'}(x', a'_1,\ldots,a'_N)_{a'_j=\mu'_j(o'_j) \text{ for } j=1,\ldots,N}. \tag{5}$$

4.2 Heterogenous Agent Trust Region Policy (HATRPO) & Proximal Policy Optimization (HAPPO)

Two algorithms being experimented upon are Heterogenous Agent Trust Region Policy Optimization (HATRPO) and Heterogeneous Agent Proximal Policy Optimization (HAPPO). These algorithms were chosen due to their proclivity in addressing heterogeneous agents and the relationship between malfunction changes and natural gradient optimization methods.

Trust region optimization and proximal policy optimization techniques are policy optimization strategies, sprouting from optimization built upon natural gradient techniques. Defining the surrogate $L_\pi(\pi^*)$ as,

$$L_\pi(\pi^*) = J(\pi^*) + \mathbb{E}_{s\sim\rho,a\sim\pi^*}[A_\pi(s,a)], \qquad D_{KL}^{max}(\pi,\pi^*) = \max_s D_{KL}(\pi(\cdot|s),\pi^*(\cdot|s)) \tag{6}$$

$$J(\pi^*) \geq L_\pi(\pi^*) - CD_{KL}^{max}(\pi,\pi^*) \tag{7}$$

TRPO[5] is formulated on the basis that as the distance between the current policy and the candidate policy decreases, $L_\pi(\pi^*)$ becomes an increasingly accurate measure of the actual performance metric $J(\pi^*)$. Therefor based upon Equation (6), a trust region algorithm based upon iteration can be derived below,

$$\pi_{k+1} = \arg\max_\pi (L(\pi) - CD_{\max}(\pi,\pi')) \tag{8}$$

Within the original paper, the KL-penalty was approximated with the KL-constraint **PPO**[6] is an approximate solution to TRPO using only first order derivatives, the PPO-clip objective is as follows,

$$L^{\pi_{\theta_k}}(\pi_\theta) = \mathbb{E}_{s\sim\rho_{\pi_{\theta_k}},a\sim\pi_{\theta_k}}\left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1\pm\epsilon\right)A^{\pi_{\theta_k}}(s,a)\right)\right] \tag{9}$$

HATRPO/HAPPO[7] algorithms define such surrogate equations for a joint policy and then proceed with sequentially updating the individual policies.

4.3 Mixed Q-Functionals (MQF)

MQF[8] is the multi-agent extension of Q-Functionals.[9] The algorithm takes a novel approach towards the formulation of the action-value function (3.2); the change in perspective enables the handling of continuous action space. This is accomplished by currying or splitting the input parameters into a composition of functions, as in one function that when passed in a state provides a function that can be evaluated with an action.

$$Q(s,a) : (\mathcal{S}\times\mathcal{A}) \to \mathbb{R} \quad \rightsquigarrow \quad Q_{FUNC}(s,a) : \mathcal{S} \to (\mathcal{A}\to\mathbb{R}) \tag{10}$$

Each state defines parameters over the action space, and the resulting function can be used to find the optimal action. MQF leverages CTDE by utilizing a centralized loss function and then factorizing such through a mixing function for individual agents. The algorithm is off-policy and has superior sample efficiency through the use of a replay buffer. MQF identifies optimal action through a uniform sampling of the action space where each sample is evaluated over the function derived from passing the state into the Q_{FUNC}.

5 Experiments

As delineated in the introduction, the experiments center on robotics specifically the MaMuJoCo suite of environments. To test *adaptability* fixed malfunctions were staged during the middle of the training bout and learned behavior and performance were compared on a slew of metrics. This experiment was run on the Ant 4x2 environment as the faculty for investigation allowed for sufficient configurability and control in the testing procedure; each of the legs were conceptually homogeneous, having identical action and observation

spaces, and thus it was easy to split the malfunctions into two categories and observe the behavior. Building off of this experiment, the next aim was to test *transferability* of knowledge; the curiosity was whether or not the experience of malfunction on one limb would lead to some form of advantageous learning on an adjacent limb. Lastly as a metric for comparison between the value and policy characteristics, *scalability* was tested concerning the number of agents. This experiment took the same Ant environment, however, it measured the effectiveness of algorithms at 2, 4, and 8 agents.

5.1 MaMuJoCo

MaMuJoCo is a multi-agent configuration of MuJoCo, which stands for Multijoint dynamics with contact. The environments allow however many controllable joints there are to be configured into separate agents. The environment that will be experimented upon is Ant.

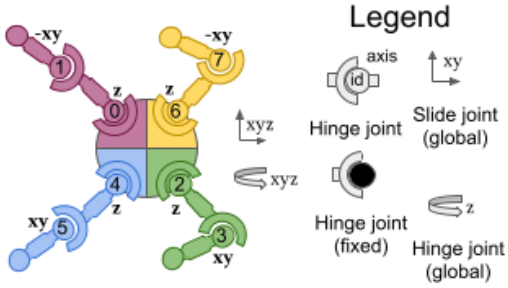


Figure 1: Ant 4x2 configuration

5.1.1 Ant

The ant is a 3D robot consisting of one torso (free rotational body) with four legs attached to it with each leg having two body parts. The goal is to coordinate the four legs to move in the forward (right) direction by applying torques on the eight hinges connecting the two body parts of each leg and the torso (nine body parts and eight hinges).

4x2 Partitioning Scheme

- Partitioning Approach: The 4x2 partitioning divides the environment into four parts, with each part corresponding to a single leg of the ant. This scheme creates a more granular control environment, with four agents each responsible for a specific leg's movements.
- Agent [0..3] Action Space: Controls torque applied to the hip and the angle between the leg links. The action space consists of two actions in a continuous box range from -1 to 1.

5.2 Algorithmic configurations

When configuring the following algorithms and tuning hyperparameters, it was necessary to make certain algorithms work properly within the continuous action space. For MADDPG, the algorithm is originally configured towards a discrete action space due to the nature of MPE (Multi-Agent Particle Environment), for which it was originally designed. The change to allow utilization necessitated the critic to utilize activation functions of ReLU and the actor to have hidden layer activation functions of ReLU and an output layer of Tanh, so that the output was within the MaMuJoCo's action space domain. A note to add is that, although Tanh limits the bounds of the output between -1 and 1, the environment was distributing negative rewards associated with control costs, which signified out-of-domain action values.

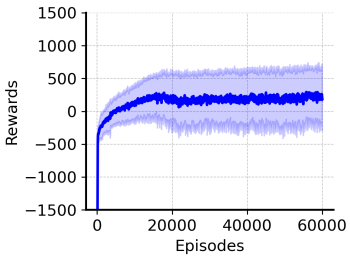
5.3 Environment & Training parameters

For the Ant environment, several initialization parameters were set for training purposes. The ctrl_cost_weight was set to 0, terminate_when_unhealthy was set to False, and healthy_rewards set to 0.1. Control cost was eliminated as the other training algorithm QMIX was trained on an earlier version of MuJoCo that didn't utilize this reward mechanism. Terminating when unhealthy was chosen to standardized training steps between algorithms. Healthy_rewards was set to 0.1 because MADDPG was converging towards stasis behavior beforehand.

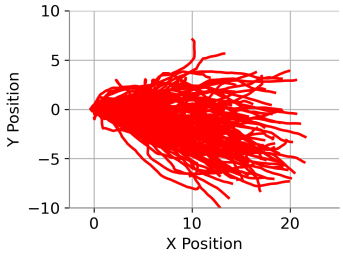
An important note is that for each of the training runs, the episodes were instantiated with the parameters of max_episode_steps being set to 100, and terminate_when_unhealthy being set to False. This effectively cuts the amount of time in a single episode from the default 1000 to 100 and then disallows any faulting of the environment. It creates a form of level playing field whereby both total training steps and training episodes is held fixed.

5.4 Malfunction

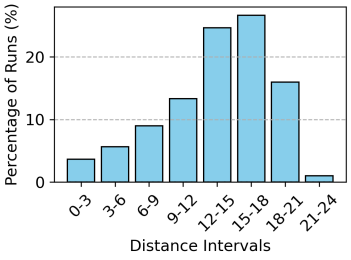
Within the experiment, the front left leg and the back left leg of the agent were chosen to experimented on independently. The experiment went as follows, the agent would learn in the regular environment for 30,000 episode (3,000,000 training steps), and then at the 30,000th episode the agent's motor would experience a malfunction. The implementation of the motor malfunction could take a plethora of directions, however the choice of malfunction in our experiment is a fixed start state, where for each environment step the agent's actions are all zeros.



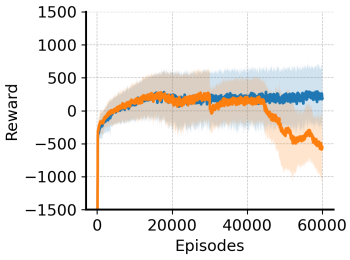
(a) Reward function - normal



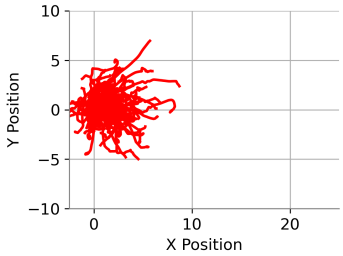
(b) Trajectories



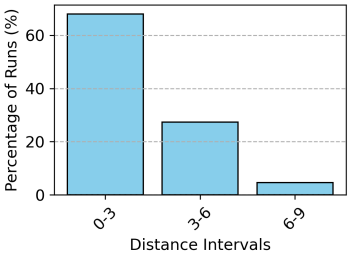
(c) Distance Distribution



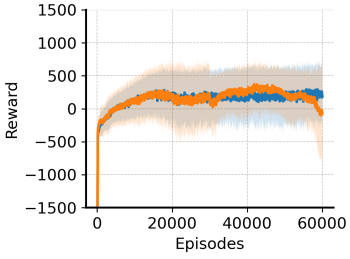
(d) Reward function - agent 0 malfunction



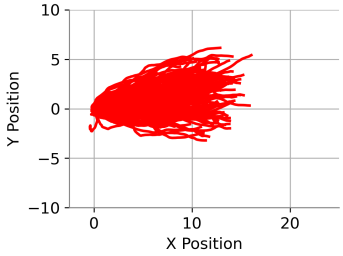
(e) Trajectories



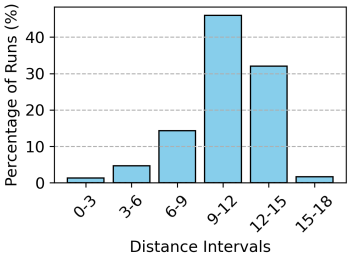
(f) Distance Distribution



(g) Reward function - agent 2 malfunction



(h) Trajectories



(i) Distance Distribution

It is apparent that the location of the malfunction with respect to the given experiment is important to be controlled upon. As the learned behavior within front left leg malfunction is much less optimal compared to that of the hind leg.

5.4.1 Changing reward functions — Ant 4x2

When comparing QMIX and MADDPG, the learned behavior appeared to be similar through the resemblance of trajectory diagrams, however, the reward graphs of each converged to different values. The corresponding reward function parameters are as follows:

- R1 := {use_contact_forces=True}
- R2 := {use_contact_forces=False, healthy reward = 0.1,}

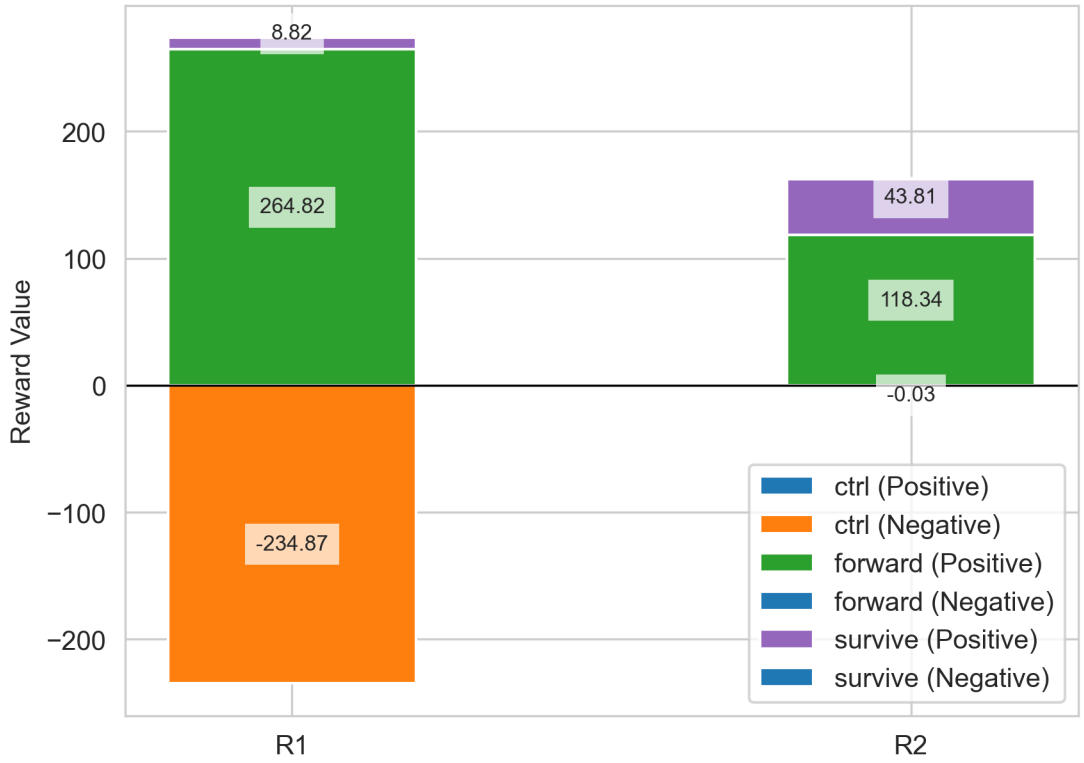


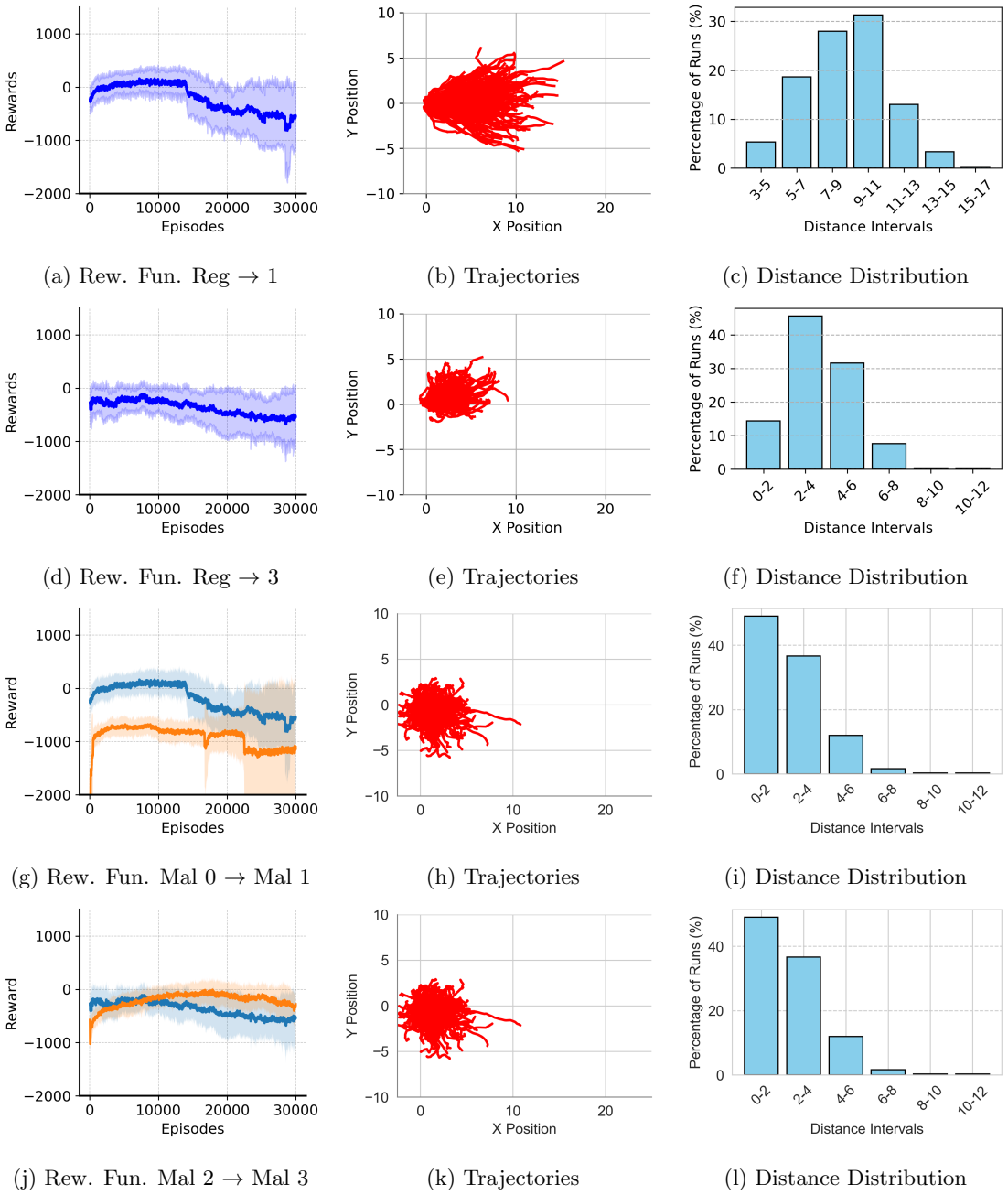
Figure 3: Caption

Above is the comparison of the learned behavior of MADDPG under each of the differing reward functions. This is tested with the MADDPG model that was trained with R2; R2 allowed for MADDPG to move beyond the suboptimal behavior of standstill. By lowering the healthy reward, the positive reward to movement in the positive x direction was reinforced. It is also apparent through the control cost that MADDPG suffers tremendously, it makes sense that the value is very high as the value was not

5.5 Tranfer Learning

With the models derived from the malfunction experiment, transfer learning was tested via further training on adjacent limb. The goal was to identify if previous experience with a similar scenario allowed for advantageous learning ability.

- Front right trained with {regular, front left malfunction}
- Back right trained with {regular, back left malfunction}



(a) Rew. Fun. Reg → 1 (b) Trajectories (c) Distance Distribution

(d) Rew. Fun. Reg → 3 (e) Trajectories (f) Distance Distribution

(g) Rew. Fun. Mal 0 → Mal 1 (h) Trajectories (i) Distance Distribution

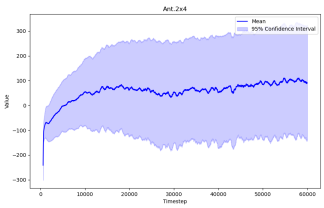
(j) Rew. Fun. Mal 2 → Mal 3 (k) Trajectories (l) Distance Distribution

Figure 4: MADDPG Transfer learning scenarios

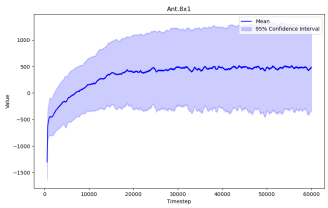
The

5.6 Scalability

This experiment took advantage of the ease of simplicity in the scaling of agents within the Ant environment. The goal was to see how each algorithm compared with the Ant environment when it was factored into 2, 4, and 8 agents.



(a) 2x1



(b) 8x1

6 Conclusions and Future work

The intentions of the experiments still have much to build upon in terms of completing the experiments. Going further, experiments with multiple runs would be done utilizing HATRPO and HAPPO on the malfunction scenario; due to time constraints the runs were limited to ¡ 2, which isn't ideal for gathering adequate approximations of mean and variance. In addition to finishing experiments, the HATRPO, HAPPO, and MQF would be ran on the transfer learning experiments.

The experiments have various aspects that would benefit from further experimentation and investigation. For the malfunction experiment, a point to control would be malfunction timing; does the length of training affect the capacity to adapt? With the transfer learning experiment, a similar curiosity regarding the length of training is present. For the scalability experiment as well as the other experiments the change in loss function ultimately led to a less rich environment and it seems to have a lesser capacity to compare optimal learned behaviors truly.

Various avenues seem to sprout off the idea of malfunction. Instead of actuator or motor malfunctions, to consider sensor impairments; introducing noise to the continuous observation space. Such introduction of noise could also be implemented in the motor impairments as a more soft or mild form of malfunction.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[2] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs*, 1st ed., ser. SpringerBriefs in Intelligent Systems. Springer Cham, Jun 2016, book Title: A Concise Introduction to Decentralized POMDPs. Series Title: SpringerBriefs in Intelligent Systems. Number of Illustrations: 14 b/w illustrations, 22 illustrations in colour. Topics: Artificial Intelligence, Control, Robotics, Mechatronics, Optimization.

[3] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proceedings of the Eleventh International Conference on Machine Learning*, vol. 157, 1994, pp. 157–163.

[4] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," 2020.

[5] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[7] J. G. Kuba, R. Chen, M. Wen, Y. Wen, F. Sun, J. Wang, and Y. Yang, "Trust region policy optimisation in multi-agent reinforcement learning," *CoRR*, vol. abs/2109.11251, 2021. [Online]. Available: https://arxiv.org/abs/2109.11251

[8] Y. Findik and S. R. Ahmadzadeh, "Mixed q-functionals: Advancing value-based methods in cooperative marl with continuous action domains," 2024.

[9] S. Lobel, S. Rammohan, B. He, S. Yu, and G. Konidaris, "Q-functionals for value-based continuous control," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 7, pp. 8932–8939, Jun. 2023. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/26073