Deep Reinforcement Learning for Single-Shot Diagnosis and Adaptation in Damaged Robots

Shresth Verma[1], Haritha Nair[2], Gaurav Agarwal[3], Joydip Dhar[4] and Anupam Shukla[5]

Abstract— **Robotics has proved to be an indispensable tool in many industrial as well as social applications such as warehouse automation, manufacturing, disaster robotics, etc. In most of these scenarios, damage to the agent while accomplishing mission-critical tasks can result in failure. To enable robotic adaptation in such situations, the agent needs to adopt policies which are robust to a diverse set of damages and must do so with minimum computational complexity. We thus propose a damage aware algorithm which diagnoses the damage prior to gait selection while incorporating domain randomization in the damage space for learning a robust policy. To implement damage awareness, we have used a Long Short Term Memory based supervised learning network which diagnoses the damage and predicts the type of damage. The main novelty of this approach is that only a single policy is trained to adapt against a wide variety of damages and the diagnosis is done in a single trial at damage time.**

I. INTRODUCTION

Robotic devices were introduced with the motive of providing a safe method of access and operation in environments that are hazardous and unreachable to humans. But very often these environments destabilize or damage the robot partially, often impairing them, leading to a mission failure or significant drop in performance. This is especially critical for robots deployed in manufacturing industries and warehouses [1], search and rescue missions [2] and disaster response [3]. Although this situation of partial damage is tackled in humans or animals by their learning of alternate ways to perform the action, this kind of learning in robots requires, what we call, intelligence. Hence the objective while designing robotic devices is not restricted to avoiding or tackling obstacles, it also includes the adaptation of the agent in presence of adversaries, both in the form of internal damages as well as external effects.

Deep Reinforcement learning (Deep RL) has been shown to be effective in modeling such navigation problems because of its online learning capability in high dimensional search spaces [4], [5], [6], [7]. But the environments and agents both are complex in nature as a result of which, retraining the RL policy every time a change occurs in either of them is highly impractical. This points to the necessity of having a single robust policy which can help the agent adapt in varying adversarial conditions.

To remove the bottleneck of single policy, several approaches have tried to learn multiple policies and then choosing from them at the time of damage. However, models which have made progress in this domain require reset of the agent to initial state [8], or multiple hardware trials to be performed to help the agent recover or adapt [8], [9], [10]. Although this is intuitive, it is inefficient considering the overhead. To make a smart decision, rather than choosing from a set of high performing gaits, the agent will need to understand the damage prior to adaptation.

We propose Damage Aware-Proximal Policy Optimization (DA-PPO), combining damage diagnosis with deep reinforcement learning. The algorithm performs damage diagnosis on multiple damage cases using a Long Short Term Memory (LSTM) [11], based supervised learning network, which uses the difference between the gaits of a healthy and a damaged robot as input and classifies the damage that has occurred if any. The diagnosed damage data is collected along with the current observation vector to create a modified observation space, which contains information of both state space observation as well as damage. This data is used to train our RL model, which is optimized using Proximal Policy Optimization (PPO) [12]. The trained model shall be able to understand the damage that has occurred and choose its gait accordingly. Since only a single policy is developed, this reduces the overhead of storing and choosing between multiple policies, making our algorithm effective in real time.

We intend to create an efficient control structure that can tackle single and multiple internal damages in robotic agents in real time. The major objectives of our work are:

1) To create a deep reinforcement learning based control architecture for enabling robots to accomplish mission-critical tasks even in presence of physical damages.
2) To optimize the control structure so that the robot adapts its gait using a single hardware trial.

II. RELATED WORK

Early work on automated recovery in robots was based on evolutionary algorithms and generally divided the process into two phases-damage estimation and recovery. Based in this idea, the algorithm introduced by Bongard et al. [9] proposed a continuous information flow between a physical robot and its simulation wherein the robot provides its current state information to the simulator, which returns neural controllers that are expected to handle its state or damage.

Rather than considering two separate phases for damage diagnosis and recovery algorithm generation, Cully et al. [8], proposed a method inspired from animals, who perform trial and error to determine the least painful alternate gait in presence of injury. The approach put forward, Intelligent

*This work was not supported by any funding organization

The authors are with the Department of Information Technology, Indian Institute of Information Technology and Management, Gwalior, India. (e-mail:ipg_2015095@iiitm.ac.in; ipg_2015035@iiitm.ac.in; jdhar.iiitmg@gmail.com).

Trial and Error (ITE), relies on a behavior-performance map space. This map enables the robot to try multiple behaviors which are predicted to perform well. The algorithm implements a Gaussian process, which uses current data to approximate a performance function, and a Bayesian optimization procedure, which uses this model to find the maximum of the performance function. Based on the trials conducted and their results, the estimated performance values are also updated in the map. The process converges when the best behavior possible has been estimated. This select-test-update loop continues until the right behavior is obtained.

Inspired by ITE, Chatzilygeroudis et al. [4], proposed a more optimized version of the algorithm. Similar to ITE, Reset free Trial and Error (RTE) pre-computes and generates a behavior performance map using MAP-ELITES [13]. It learns the robots model, especially when it is damaged and uses Monte Carlo Tree Search [14], to compute the next best action for the current state of the robot. Also, the method uses a probabilistic model to incorporate uncertainty of predictions and uses this data to correct the outcome of each action on the damaged robot. This culmination of algorithms makes sure that there is no reset required when damage occurs.

An approach was introduced by Kume et al. [15], using multi-policy mapping for a single behavior, unlike previous works where a single policy was used. Map-based Multi-Policy Reinforcement Learning (MMPRL) trains many different policies by collaborating a behavior-performance map and the concepts of deep reinforcement learning. It aims to search and store these multiple policies while maximizing expected reward. MMPRL saves all possible policies with different behavioral features, making it extremely fast and adaptable.

Some recent works have also experimented with randomization in simulation environments through domain and dynamics randomization [16], [17] so as to bridge the gap between simulation and real world. The idea is to create numerous variations in the simulation environment so that real world appears as just another sample from a rich distribution of training samples. In [16], the authors have experimented on object localization for the purpose of grasping in cluttered environment. They have shown impressive results randomizing in the visual domain to transfer learning from simulation to real world without requiring real world training images. On the other hand, in [17], the authors have randomized the dynamics of the environment such as mass,damping factor, friction coefficient and have shown that the policy learned in such dynamic environment is quite robust to calibration errors in the real world.

While most map-based methods are able adapt over a wide range of damages, their computational overhead in creating the behaviour-performance map is a significant drawback. In ITE and RTE, the complexity is further increased by the Gaussian process computations. Moreover, all these approaches require multiple hardware trials for adapting to a damage. We try to incorporate domain randomization approach in the context of damages so that damages in the real world are just another variation of training samples. Moreover, we further improve this approach by adding a single-hardware-trial control loop for diagnosing the damage so that policy learning is aware of damage type.

III. APPROACH

A. Overview

We consider the following scenario: A robot has been damaged while in a remote and hazardous environment. We require the robot to reach the destination by adapting its gait so as to overcome the damage. Rather than making the agent dependent on a pre-computed set of high performing gaits, it should be able to identify and adapt to its damage autonomously.

Thus we propose a self-diagnose network which can predict the type of damage that has occurred in the structure of the robot. With this damage awareness, we use an augmented observation space for learning a well-performing policy through a modified version of Proximal Policy Optimization (PPO) which we call Damage Aware-Proximal Policy Optimization (DA-PPO). In our work, we assume that internal damages, unlike environmental adversaries, do not keep changing constantly. Thus, we only need to perform the self-diagnosis step for determining damage class whenever the reward drastically drops below a certain threshold, indicating that damage has occurred.

B. Self-diagnose network

Self-diagnose network is an LSTM [11] based predictive model, which tries to classify the type of damage that has occurred in the robot using continuous feedback from its gait. In Automated Damage Diagnosis and Recovery for Remote Robotics, Bongard et al. [9], have used the difference between simulated robots behavior against the physical robots behavior in terms of forward displacement in order to classify damages. We extend this idea by measuring the difference in sensor values between the two for a fixed number of time steps. This results in a time series and our problem is reduced to classifying damage from this data. More specifically, the on-board computer of the robot can run a simulation of a healthy robot and compare its gait with the actual steps taken. Based on the difference between the two, the network can diagnose the class of damage.

Algorithm 1 Sample collection

```
Result: An array with collected samples
Initialize:
 Load an expert policy trained on healthy robot
 Run parallel threads
 for $i \leftarrow 0$ to $n\_rollouts$ do
     Set a random seed
     Initialize environments $env_d$, $env_h$ for healthy and
     damaged robots with same seed value
     for $damage\_class \leftarrow 0$ to $n\_damage\_classes$ do
         $env_d$.applyDamage(damage_class)
         for $n \leftarrow 0$ to $n\_timesteps$ do
             get action from predefined policy
             $a_h$ = policy_fn($obs_h$)
             $a_d$ = policy_fn($obs_d$)
             do simulation step in both environments
             $obs_d$, $rew_d$ = $env_d$.step($a_d$)
             $obs_h$, $rew_h$ = $env_h$.step($a_h$)
         end
         collect ($o_h$-$o_d$)
     end
 end
Concatenate collected samples
```

Since this time series is multivariate and high dimensional, we use LSTM hidden units which are powerful and increasingly popular models for learning from sequence data [18]. The network is trained using data obtained through the sample collection step explained in Algorithm 1. This step is also parallelizable and thus doesn't act as a bottleneck for the entire algorithm. The network, represented by Θ, can be accessed on demand to determine damage class μ within a single trial as shown in Fig. 1.

C. Encoding of Damage Indicators

The self-diagnose network predicts the damage class of the robot which can act as an additional state information about the environment. We thus concatenate it with the observation space of the original robot to form what we call an augmented observation space.

This poses a necessity to encode the output of the classifier so that the policy efficiently learns various gaits in accordance with the damage. If a random encoding scheme is used for creating the augmented observation space, it results in the algorithm perceiving the encoding as noise, and completely avoiding it during policy learning. We have thus used partial one hot encoding and it is observed to work well in practice as the damage information is not lost in training.

In our experiments, we have limited the number of damages that can occur simultaneously to two and have taken the assumption that only one damage can occur on a limb at a time. The number of damage classes can thus be calculated as sum of zero damage case, single damage cases and multiple damages cases occurring at various limbs. This is given by:

$$D = k^0\binom{n}{0} + k^1\binom{n}{1} + k^2\binom{n}{2}, \qquad (1)$$

where n represents the number of limbs in the agent and k represents the number of different damage types considered.

The encoded vector is of length $2n$ where the damage of i^{th} limb is represented by the values at indices $2i$ and $2i+1$ in the encoded vector. Thus, we have a tuple of size 2 associated with each limb where $[0,0]$ represents no damage, $[1,0]$ represents damage type 1 and $[0,1]$ represents damage type 2 at the limb. Note that $[1,1]$ can be used if we remove the assumption that two types of damages can't occur together at a single limb. Furthermore, the tuple size can be increased to model more types of damages.

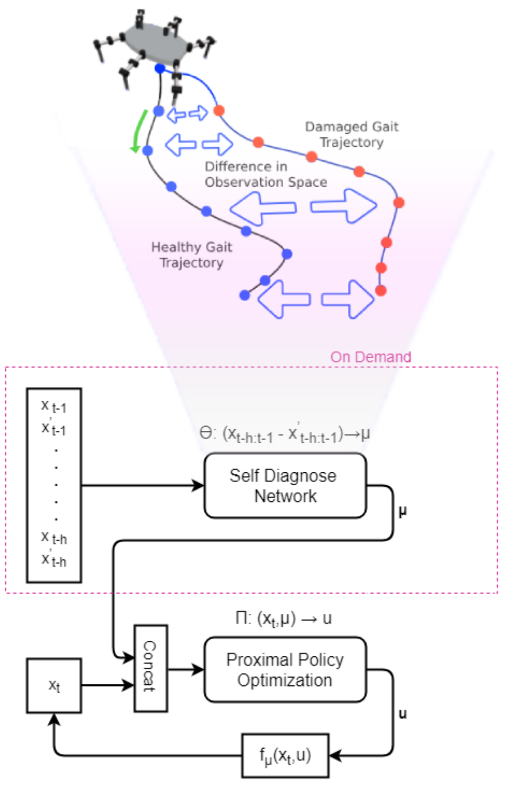


Fig. 1: Control Architecture

D. Proximal Policy Optimization

Since our task is that of continuous action control, we formulate it as a reinforcement learning problem, starting from initial state s_0, choosing a series of action $a \in A$ and obtaining state s_i and reward r_i at the i^{th} timestep while maximizing the expected sum of rewards by changing the parameter θ of the stochastic policy π_θ. But the use of large scale optimization is less widespread in continuous action spaces. An attractive option for such problems is to use policy gradient algorithms [19]. PPO improves standard policy gradient methods by including a clipped surrogate objective function. It is an improvement over Trust Region Policy Optimization [20], so as to enable the running of multiple epochs on collected data.

PPO represents the ratio between new policy and old policy as:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}. \tag{2}$$

The objective function [12]:

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

is a clipped objective which stabilizes training by constraining the policy changes at each step. This clipping approximates the gradient to a local value so that large steps are not taken between iterations, enabling multiple epoch training on collected samples. We use PPO to train our policy for learning continuous control tasks.

E. Damage Aware Proximal Policy Optimization

With the self-diagnose network in place, we can now use the policy learning algorithm on augmented observation space which encapsulates both environment state (through observation vector) and damage awareness (through damage encoding vector). We use the PPO algorithm for policy learning from the augmented observation space where x_t is the observation at timestep t, u is the action taken according to policy Π and f_μ is the environment in which damage μ has occurred (see Fig. 1). Note that we only run self diagnose network when reward during a run falls below a certain threshold. At other times, the damage is considered to be the same as diagnosed in the last run.

IV. Experimental Setup

A. Simulation Setup

To evaluate our approach, we have conducted experiments on two environments, *Ant*, a quadrupedal locomotive robot and *Hexapod*, a six-legged locomotory robot. We have used OpenAI gym toolkit [21], for performing simulations in combination with MuJoCo physics engine [22]. The Ant is an already implemented environment in OpenAI Gym while the Hexapod is implemented using the configuration and model described in ITE [8].

The two environments used in our experiments are discussed below:

Ant (Quadrupedal bot): Ant is a simple quadrupedal robot with 12 degrees of freedom (DoF) and 8 torque actuated joints. The joint has maximum flex and extension of 30 degrees for both from their original setting and also has a force and torque sensor. The observation includes features containing joint angles, angular velocity, the position with respect to the center of mass and force and torque sensor of each joint forming 111-dimension vector. The target action values are the motor torque values which are limited to the range -1.0 to 1.0. We limit an episode to at most 1000 timesteps and the episode will end whenever it crosses this limit or robot falls down on its legs or jumps above a certain height. The reward function is defined as follows:

$$R_t = \Delta x_t + s_t - w_0 C_t - (w_1||\phi_t||_2)^2, \tag{4}$$

where Δx_t is the covered distance of the robot in the current time step since the previous time step, s_t is the survival reward, which is 1 on survival and 0 if the episode is terminated by the aforementioned conditions. The variable C_t is the number of legs making contact with the ground, $\phi_t \in R^8$ are the target joint angles (the actions), and w_n is the weight of each component with w_0 = 0.5, w_1 = 0.5.

Hexapod: There are three actuators on each leg of the Hexapod. In the neutral position, the height of the robot is 0.2 meters. In addition to this, the actions are taken to be the joint angle positions of all 18 joints, which ranges from -0.785 to 0.785 radians. As the observation space of the agent, a 53-dimension vector is given as input which consists of the position and velocity of all the joints as well as he center of mass. Along with this, the observation space contains booleans of touch sensors which indicate whether a leg is making contact with the ground or not. Again, we limit an episode to be at most 1000 timesteps and the episode will end whenever the robot falls down on its legs or jumps above a certain height or crosses the time limit.

The reward function R is defined as follows:

$$R_t = \Delta x_t + s_t - w_0 C_t - (w_1||\tau_t||_2)^2 - (w_2||\phi_t||_2)^2, \tag{5}$$

where Δx_t is the covered distance of the robot in the current time step since the previous time step, s_t is the survival reward, which is 0.1 on survival and 0 if the episode is terminated by the aforementioned conditions. The variable C_t represents the number of legs making contact with the ground, $\tau_t \in R^{18}$ is the vector of squared sum of external forces and torques on each joint, $\phi_t \in R^{18}$ are the target joint angles (the actions), and w_n is the weight of each component with w_0 = 0.03, w_1 = 0.0005, and w_2 = 0.05.

B. Damage Simulation

Since both the environments considered in our experiments are simulated in OpenAI gym, the damages are implemented by modifying the *xml* files of the 3D model. This can be done on the fly without affecting parallelly running experiments. In our work, we have simulated broadly two kinds of internal damages which are:

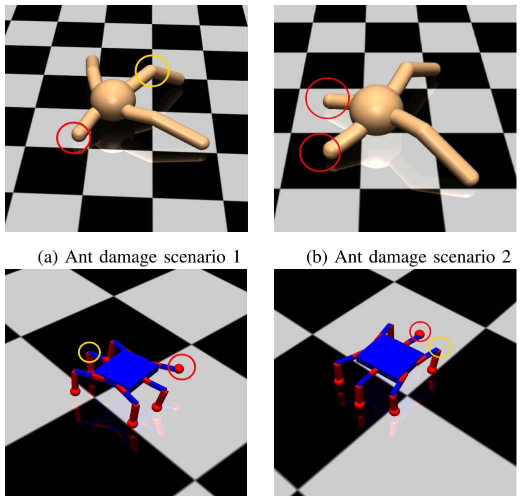
1) Jamming of joint such that it can't move irrespective of the amount of torsional force applied by the motor at that joint.
2) Missing toe, i.e., lower limb of the robot breaks off.

In both Ant and Hexapod, jamming of joint is modeled by restricting the angle range of the concerned joint to -0.1 to 0.1 degrees from default values of -30 to 30 degrees and -45 to 45 degrees respectively in the two environments.

Missing toe is modeled by reducing the lower limb size to 0.01 from the original values of 0.8 and 0.07 in the two environments. Note that in Hexapod, there are touch sensors on each of the lower limbs, so, whenever that limb breaks off, we consider that the touch sensor for that limb stops giving any signal and it is considered to output 0.

V. Results and Discussion

We evaluate the performance of our approach within the two elements involved : (1) Self-Diagnose network for


(a) Ant damage scenario 1 (b) Ant damage scenario 2

(c) Hexapod damage scenario 1 (d) Hexapod damage scenario 2

Fig. 2: Some of the damage scenarios in Ant and Hexapod. Yellow and red circles represent jammed joint and missing limb damage types respectively.

TABLE I: Classification accuracy in predicting damage class in Ant and Hexapod environment with varying number of timesteps and rollouts. Method A represents using plain observations as time series and method B represents using difference of observations between healthy robot and damaged robot as time series.

Classification Accuracy in Ant Environment				
Timesteps	Method	Number of Rollouts		
		1000	2000	7000
10	A	78.2±1.11	81.4±0.6	82.4±0.87
	B	**81.24±2.88**	**85.2±1.2**	**84.33±0.72**
30	A	82.17±1.7	87.1±1.8	88.17±1.3
	B	**83.62±2.03**	**90.8±0.9**	**91.5±1.067**
50	A	83.11±0.8	90.17±1.2	92.83±1.8
	B	**84.29±1.21**	**92.6±1.83**	**96.8±1.48**

Classification Accuracy in Hexapod Environment				
Timesteps	Method	Number of Rollouts		
		1000	2000	7000
10	A	22.2±0.6	33.1±1.23	44.6±0.9
	B	**32.6±0.8**	**38.5±1.1**	**47.8±1.13**
30	A	60.5±1.9	62.9±1.8	79.67±1.02
	B	**65.45±1.2**	**69.17±1.11**	**82.6±1.28**
50	A	65.23±1.3	69.7±1.1	82.2±1.8
	B	**68.83±1.8**	**72.17±1.29**	**87.6±0.86**

predicting class of damage (2) DA-PPO, which learns to adopt a policy given that a particular damage has occurred.

A. Self-diagnose network

For the comparison of performance, we consider different number of rollouts (amount of data to train on), length of history to look back into (timesteps) and what to give as observation data, i.e., our proposed approach of using difference of observations between healthy and damaged run or plain observations of only damaged run. Table I summarizes the validation accuracy for these modifiable parameters. We can observe that classifying using fewer timesteps results in faster diagnosis but at the expense of accuracy. Moreover, classification using the difference between observation vectors as input outperforms the use of plain damaged observations in most of the cases. If there is a constraint on computation power of on-board computer of the robot then former can be preferred over the latter one.

As for the network structure, we have used an LSTM embedding layer with embedding size 512 as our first hidden layer. It is followed by three dense layers of size 256, 128, 64 along with dropouts, so as to reduce overfitting. The output layer uses *softmax* as activation so that it outputs class probabilities. The loss function and optimizer used are *categorical crossentropy* and *adam* respectively. For Ant and Hexapod environments, the possible classes range from 0 to 32 and 0 to 72 respectively as calculated from equation 1.

B. Damage Aware-Proximal Policy Optimization

We start by comparing the performance of PPO policy which is trained on damage classes but without augmented observation space (i.e., without explicit knowledge of damage class), aka PPO-Unaware and Damage Aware PPO policy aka DA-PPO. The performance metric used is the forward reward of the agent. Fig. 4 shows the training curve comparison between PPO-Unaware and DA-PPO in Ant (See Fig. 4a) and Hexapod (See Fig. 4b) environments. It shows that average forward reward is consistently better in DA-PPO than PPO-Unaware.

For the Hexapod environment, we also use the concept of curriculum learning [23], by progressively training on cases which are more difficult. We implement this by increasing the percentage of damage classes in training examples and also progressively increasing the severity of damages (include multiple damages). In this way, we were able to encourage rapid learning progress.

We also do a per class performance analysis of the two approaches discussed across various damage classes in both Ant and Hexapod (See Fig. 3, 5). In the Ant environment, DA-PPO performs better in 82.84% of damage classes when compared to PPO-Unaware. Comparing between various damage classes, DA-PPO is seen to adapt really well when multiple damages occur on adjacent limbs, rather than when damage occurs on opposite limbs - which is a rarer case. In the Hexapod environment, DA-PPO performs better in 82.84% of damage classes when compared to PPO-Unaware (See Fig. 3).

VI. CONCLUSIONS

We have proposed and implemented a two-part algorithm for robotic damage adaptation. This is particularly useful when expensive robots are used in hazardous environments, where human intervention is nearly impossible.

Our algorithm enables the agent to autonomously identify and understand any damage that occurs to its physical structure and adapt its gait accordingly. Since the ultimate goal is the creation of intelligent machines, understanding the damage is as important as adapting from it, which has often been overlooked in previous literature.

On comparison with map-based approaches, DA-PPO doesn't require any pre-computation and thus the computation time is relatively much less. This is also enhanced by the fact that the algorithm adapts to the damage in a single trial itself, without trying multiple well-performing gaits or

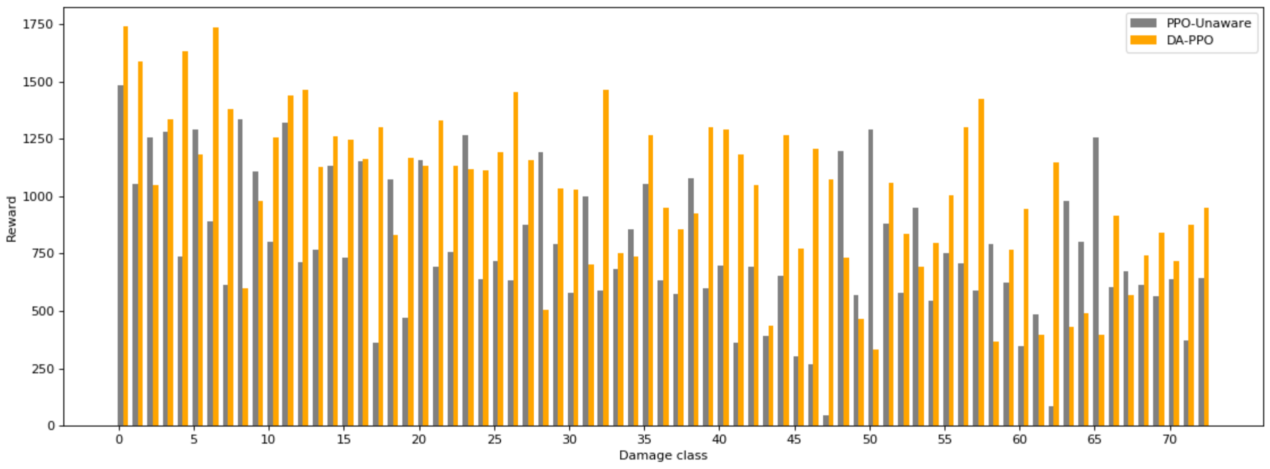
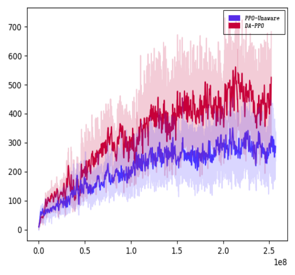
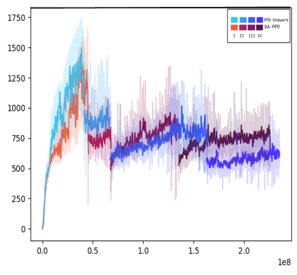


Fig. 3: Forward reward comparison between DA-PPO and PPO-Unaware across different damage classes in Hexapod



(a) Training curve of Ant



(b) Training curve of Hexapod

Fig. 4: Average reward comparison between DA-PPO and PPO-Unaware in Ant and Hexapod. In Hexapod, each piece-wise curve represents a stage (I, II, II or IV)Iin the curriculum learning process. I has 100% healthy cases, II has 60% healthy and 40% single damage cases, III has 70% healthy and single damage and 30% multiple damage cases and IV has all damages equally likely.

without having to be reset to the initial state to perform the trial.

The approach can be easily scaled to a larger number of damage classes too. Since no differentiation is made between the cause of damage, adaptation is possible in case of both morphological and external damages. Also, in the case of unknown damages, the network is expected to predict a damage class which resembles the actual damage the most and try to choose a gait accordingly. This implies a very low rate of complete failure.

Future work shall be focused on extending the algorithm to handle environmental adversaries, which is much desirable since real-world environments are not predictable. We also intend to work on DA-PPO for complex and dynamic environments, using SLAM [24]. Finally, we plan to extend our method and prove its effectiveness by applying it on a physical robot.

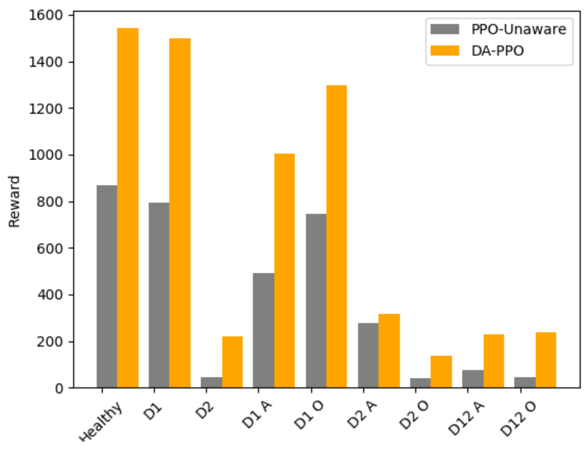


Fig. 5: Forward reward Comparison between PPO-Unaware and DA-PPO across different grouped damage classes in Ant. D1 and D2 refers to single jammed joint and single missing toe damages. D_{ij} A and O represents that damage type i and j are present in adjacent (A) or opposite (O) limbs.

References

[1] B. S. O. Khatib, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[2] R. R. Murphy, "Trial by fire [rescue robots]," *IEEE Robotics Automation Magazine*, vol. 11, no. 3, pp. 50–61, Sep. 2004.

[3] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima, and S. Kawatsuma, "Emergency response to the nuclear accident at the fukushima daiichi nuclear power plants using mobile rescue robots," *J. Field Robot.*, vol. 30, no. 1, pp. 44–63, Jan. 2013. [Online]. Available: http://dx.doi.org/10.1002/rob.21439

[4] K. Chatzilygeroudis, V. Vassiliades, and J.-B. Mouret, "Reset-free trial-and-error learning for robot damage recovery," *Robotics and Autonomous Systems*, vol. 100, pp. 236 – 250, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889017302440

[5] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, Oct 2017.

[6] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," *ICML*, 2017. [Online]. Available: https://arxiv.org/abs/1703.02702

[7] K. Lobos-Tsunekawa, F. Leiva, and J. Ruiz-del-Solar, "Visual navigation for biped humanoid robots using deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3247–3254, Oct 2018.

[8] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, May 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14422

[9] J. C. Bongard and H. Lipson, “Automated damage diagnosis and recovery for remote robotics,” in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA ’04. 2004*, vol. 4, April 2004, pp. 3545–3550 Vol.4.

[10] S. Koos, A. Cully, and J.-B. Mouret, “Fast damage recovery in robotics with the t-resilience algorithm,” *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1700–1723, 2013. [Online]. Available: https://doi.org/10.1177/0278364913499192

[11] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” 07 2017.

[13] J. Mouret and J. Clune, “Illuminating search spaces by mapping elites,” *CoRR*, vol. abs/1504.04909, 2015. [Online]. Available: http://arxiv.org/abs/1504.04909

[14] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, “Monte-carlo tree search: A new framework for game ai.” 01 2008.

[15] A. Kume, E. Matsumoto, K. Takahashi, W. Ko, and J. Tan, “Map-based multi-policy reinforcement learning: Enhancing adaptability of robots by deep reinforcement learning,” *CoRR*, vol. abs/1710.06117, 2017. [Online]. Available: http://arxiv.org/abs/1710.06117

[16] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30.

[17] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, “Sim-to-real transfer of robotic control with dynamics randomization,” *CoRR*, vol. abs/1710.06537, 2017. [Online]. Available: http://arxiv.org/abs/1710.06537

[18] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “Lstm: A search space odyssey,” *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[19] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML’14. JMLR.org, 2014, pp. I–387–I–395. [Online]. Available: http://dl.acm.org/citation.cfm?id=3044805.3044850

[20] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1889–1897. [Online]. Available: http://proceedings.mlr.press/v37/schulman15.html

[21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[22] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

[23] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML ’09. New York, NY, USA: ACM, 2009, pp. 41–48. [Online]. Available: http://doi.acm.org/10.1145/1553374.1553380

[24] H. Durrant-Whyte and T. Bailey, “Simultaneous localisation and mapping (slam): Part i the essential algorithms,” *IEEE Robotics and Automation Magazine*, vol. 2, p. 2006, 2006.