Final Project For ECE228 Track Number 1

Group Number 50: Pranav Maddireddy **Akhilan Gurumoorthy**

Abstract

A very prominent topic in robotics is the ability to move across a variety of terrains and environments. A common way to do this is to train a robot to move in a simulation environment and transfer that robot's policy into the real world. However, this can run into the issue of the robot not being able to fully grasp the environments that exist in the real world, as they are trained fully in a simulation environment. We expand upon a previous paper that trained a policy for a four-legged robot using motion capture data of animals. We introduce a physics constraint term to the baseline reward function, which only contains terms that imitate motion capture data in a simulation environment. We find that the model with this additional term provides more constraints and performs very similarly to the base model, and we believe that it would allow for more effective transfer to a real-world robot.

1 Introduction

1.1 Problem Background and Motivation

Robotics is a growing field that is developing quickly and is focused on solving a large set of problems. One of these problems is moving over different types of terrain and environments. There have already been many solutions to this problem in which people have manually designed control policies or had robots learn motion from scratch. While promising, these methods lack the smoothness and dynamic movement we see in humans and animals. One solution to this problem is modeling robot movement after that of humans and animals that can already move through environments fairly well. We can do this by using motion capture technology to help us create reference motions from animal movement and then use reinforcement learning to learn control policies that would allow robots to move similarly. Typically, reinforcement learning in robotics cases is difficult due to the large amount of episodes needed to learn optimal policies. We can get around this issue by simulating our training robot using physics engined like pybullet and then transferring learned policies to real robots (sim to real transfer). In this project, we will examine a previous paper that used reinforcement learning to teach a four-legged robot to imitate the movement of animals through motion capture footage.

1.2 Project Overview

In our project, we are replicating and building off of the work done in the paper "Learning Agile Robotic Locomotion Skills by Imitating Animals"(Peng et al. [2020]). This paper utilizes motion capture footage of dogs and reinforcement learning to teach a quadruped robot how to move. This paper showed promising results for agile movement but still has some trouble translating dynamic movements like jumping from the sim to real space. In our paper, we are trying to improve on the transfer from the simulation environment to the real-world environment. We believe that the most effective way to gain improvement in this facet of the project is to modify the reward function used to train the policy. The reward function only uses imitation learning factors, and we believe that adding physics-based components to the calculation of the reward function will allow the learned policy to better capture the aspects of the real world that may not be present in a simulation environment. We believe that this will lead to less efficient training results due to the extra constraints, but will generalize better in the sim to real transfer.

Preprint. Under review.

1.3 Contributions

- Modifying the reward function to increase stability
- Utilizing physics constraints to create better sim to real transfer

2 Related work

The paper we are building off of, as mentioned earlier, is "Learning Agile Robotic Locomotion Skills by Imitating Animals" (Peng et al. [2020]) . This paper trains a policy using reinforcement learning, and their reward function for this training is calculated by taking the weighted sum of five sub-rewards: the pose reward (r^p), the velocity reward (r^v), the end-effector reward (r^e), the root pose reward (r^{rp}), and the root velocity reward (r^{rv}). More specifically, the reward function at any timestep is calculated with the following:

$$r_t = w^p r_t^p + w^v r_t^v + w^e r_t^e + w^{rp} r_t^{rp} + w^{rv} r_t^{rv}$$

$$w^p = 0.5, w^v = 0.05, w^e = 0.2, w^{rp} = 0.15, w^{rv} = 0.1$$

All of these rewards are calculated by taking the simulated robot and comparing it to motion capture data of an actual animal. This allows the robot to train a policy similar to that of the animal, and in theory, this would mean that the robot would imitate the animal. One limitation of this approach is that it is only able to imitate the motion capture data of the animal in a simulation environment. As a result, it is only able to gain information about the physics of the real world through whatever latent information is passed on through the motion capture data of the animal. Our work attempts to address this issue by directly appending physical constraints to the reward function, as this would help the robot learn real-world physics directly, rather than latently.

Furthermore, Xue Bin Peng has another paper Sim-to-Real Transfer of Robotic Control with Dynamics Randomization (Peng et al. [2018], which also addresses the issue of a sim-to-real transfer. In this paper, they perform domain randomization to try to generate different environments to train a robotic arm to better handle a real-world environment. They randomize several parameters, such as the mass of the robot, friction and damping of the puck that the arm is attempting to push, and the table height. We feel that a limitation with this method is that it is more computationally expensive, and the parameters that are being randomized do not necessarily always have to do with the physics of the real world environment. For example, one of the parameters being randomized is the mass of each link in the robot's body. In our model, we focus solely on physical constraints that are independent of the size and shape of the robot, but instead focus on position, velocity, acceleration and torque.

3 Methodology

3.1 Physical Constraints

To generate our physical constraint term, we utilized four physical concepts: dynamics, energy efficiency, stability, angular momentum. We examine these four terms in greater depth below.

3.1.1 Dynamics Term

We wanted to make sure that the acceleration of the robot was similar to that of gravity, as this is something that exists in the real world that may not be effectively represented in the simulation environment. To calculate this term, we use the central difference over the last two timesteps to calculate the acceleration in 3 dimensions. We then compare this to the $(0, 0, -9.8)$ tuple that represents gravity, and we represent the dynamics term r^d as the sum of the square of the difference of each element.

3.1.2 Energy Efficiency Term

We want to reward movements that are energy efficient, as that is more representative of the real world. Let us define the term τ_j as the torque at the joint j and the term v_j as the velocity at the joint

j. We calculate the energy efficiency term r^{ee} as follows:

$$r^{ee} = \sum_j |\tau_j v_j|$$

3.1.3 Stability Term

We want a term that checks if at least one foot is in contact with the ground at all times. This term is simply calculated by seeing if any of the feet are in contact with the ground. This term r^s is 1 if no foot is touching the ground and 0 otherwise.

3.1.4 Angular Momentum Term

We want to minimize the angular momentum to lower the presence of jittery movements. We get the base angular velocity v^a and calculate the angular momentum term as follows:

$$r^a = \sum_{dim} (v^a)^2$$

3.1.5 Physical Constraint Calculation

Now, we combine these terms together to get the singular physical constraint term r^{ph}. This term is calculated as follows:

$$r^{ph} = e^{-0.1r^d} + e^{-0.05r^{ee}} + (1 - r^s) + e^{-0.05r^a}$$

3.2 Updated Reward Function

We scale the physical constraint r^{ph} by a factor of 0.01 and add it to the reward function. Therefore, the new reward function that we use is as follows:

$$r_t = w^p r_t^p + w^v r_t^v + w^e r_t^e + w^{rp} r_t^{rp} + w^{rv} r_t^{rv} + w^{ph} r_t^{ph}$$
$$w^p = 0.5, w^v = 0.05, w^e = 0.2, w^{rp} = 0.15, w^{rv} = 0.1, w^{ph} = 0.01$$

3.3 Experimental Design

We modified the code base to utilize our new physics-enhanced reward function. We then used the reference motion from the baseline paper to train our policy for 10 million iterations. This reference motion was generated by taking motion capture data of an animal and using motion re-targeting and inverse kinematics to convert this to a reference motion that could be used for training. Once we trained our policy, we tested our model in the simulation environment and compared it to the policy trained by the baseline paper. This comparison was done both qualitatively through visual observation and quantitatively through the reward values of the models. We performed this experiment with different w^{ph} values to determine which value would be best. The reinforcement learning method we used is a hybrid imitation learning plus reinforcement learning approach that combines reference motion tracing with physics inspired reward terms. The algorithm we used is Proximal Policy Optimization (PPO) (Schulman et al. [2017]), this algorithm uses an On-policy actor-critic method with clipped objectives for stable training. We used a low discount factor $\gamma = 0.1$ to emphasize short-term agility over long-term motion planning. We had the PPO uses 8 parallel workers to collect trajectories that where fed into a two hidden layer neural network with ELU activation functions to learn the control policy. The trajectory information that was used was the pose of each joint, pose of reference joints, phase(location within repeated movement), and last action. The total loss function for PPO is:

$$L^{\text{CLIP+VF+E}}(\theta) = \mathbb{E}_t \Bigg[\underbrace{\min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t, \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right) A_t\right)}_{\text{Clipped policy loss}} - \underbrace{c_1 \cdot (V_\theta(s_t) - V_{\text{target}})^2}_{\text{Value function loss}} + \underbrace{c_2 \cdot \mathcal{H}(\pi_\theta(\cdot|s_t))}_{\text{Entropy bonus}} \Bigg] \quad (1)$$

where:

- π_θ: Current policy network parameters
- $\pi_{\theta_{\text{old}}}$: Policy before update
- A_t: Advantage estimate (GAE with $\lambda = 0.95$)
- ϵ: Clip range (typically 0.2)
- V_θ: Value function estimate
- V_{target}: Empirical return
- $\mathcal{H}$: Entropy of the policy
- c_1, c_2: Weighting coefficients ($c_1 = 0.5$, $c_2 = 0.01$ in the paper)

The training terminates once the change in policy falls below a defined threshold.

One important factor to consider is that we were only able to train our model for 10 million iterations, while the original experiment was trained for two billion iterations. This was due to computing limitations and GPU incompatibility issues. In the future, training our model for more iterations would allow us to generate stronger conclusions about our enhancement.

4 Experiments

4.1 Visual Results

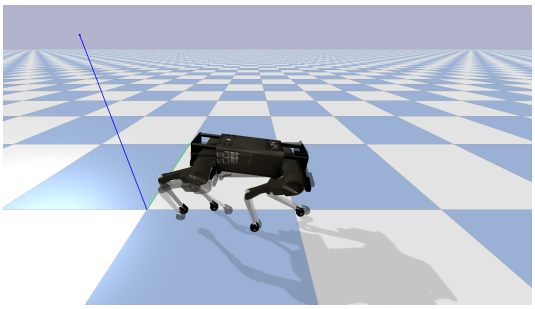

Figure 1: Baseline Policy test after 10 million timesteps

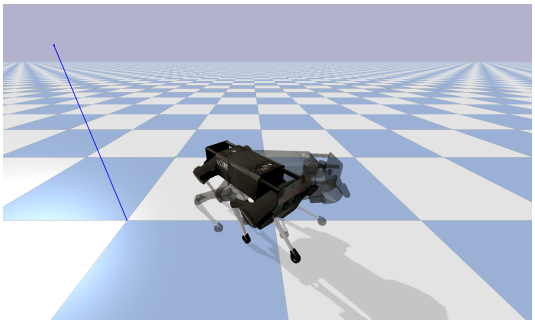

Figure 2: Physics Augmented Policy test after 10 million time-steps

The Baseline model seems to stick to the reference path better but falls over and fails after only going slightly further than the Physics based model. The Physics based model performs comparatively to the baseline but tends to deviate more to the sides than the Baseline model. We believe that this behavior is due to the constraints from the physics based model that prevent the policy from training to match the reference motion as quickly as the baseline model. So far, these results are within our expectation and require further training time for more conclusive results.

4.2 Quantitative Results

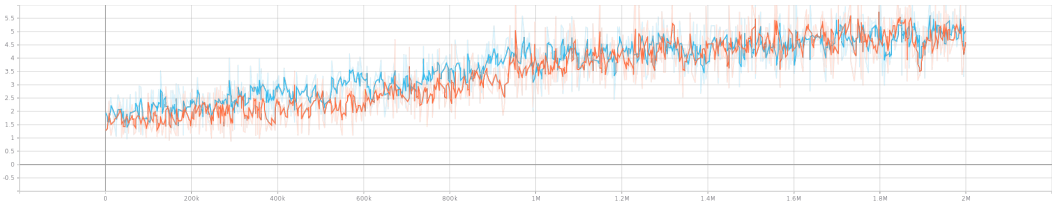

Figure 3: Baseline (orange) vs Physics (blue) reward values after 2 million time-steps

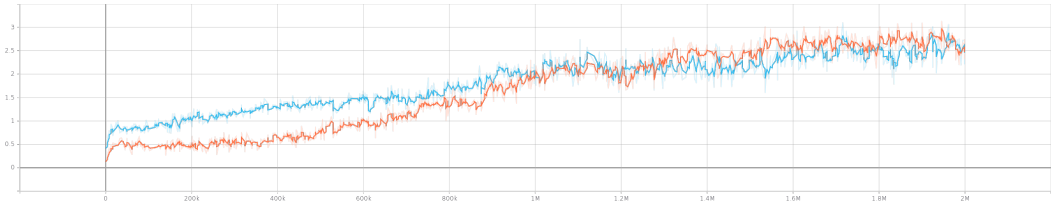

Figure 4: Baseline (orange) vs Physics (blue) discounted reward values after 2 million time-steps ($\gamma = 0.1$ for aggressive PPO training)

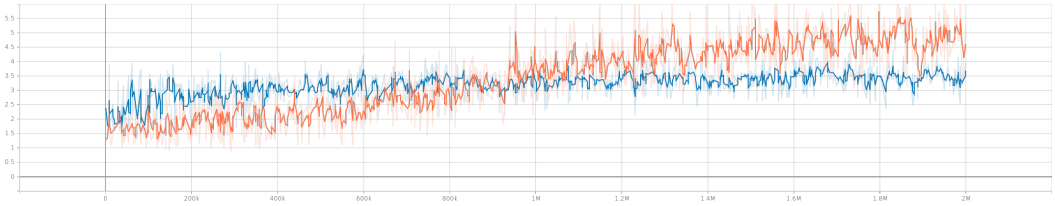

Figure 5: Baseline (orange) vs Physics (blue) physics term dominated reward values after 2 million time-steps

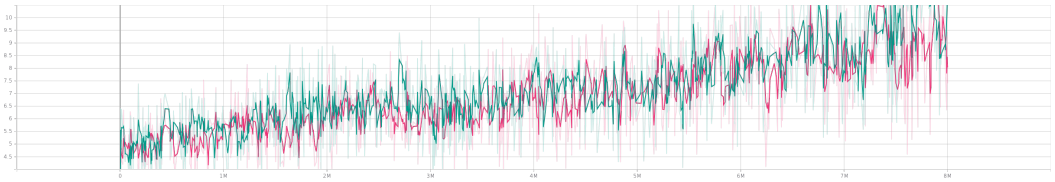

Figure 6: Baseline (green) vs Physics (red) reward values between 2 million and 10 million iterations

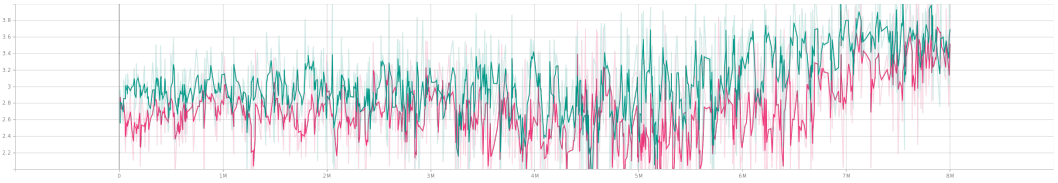

Figure 7: Baseline (green) vs Physics (red) discounted reward values ($\gamma = 0.1$) between 2 and 10 million iterations

Figures 3 and 4 compare the reward functions of our physics-based model and the baseline model over the first two million training time-steps. Our model initially trains faster but converges to a similar trend as the baseline by around 1.2 million time-steps. Figures 6 and 7 track reward evolution from 2 million to 10 million time-steps. Here, our model performs marginally worse than the baseline but remains close in performance as time-steps increase. These results follow our initial hypothesis as well as what we saw in our visual results. Figure 5 shows us a training result in which we used a weight of 0.058 for the physics-based terms instead of the value we used for the final model (0.01). This resulted in an overly constrained reward function that caused our robot to find a policy at a local maximum that prioritized minimizing movements rather than following the reference motion. The visual test showed us a model that refused to move and would constantly tighten up and fall over. We believe that is likely caused by the physics part of the reward function dominating the imitation terms and prioritizing minimal movements for efficiency. Another compounding reason could be because we are using an on-policy training method so the lack of exploration makes it impossible to escape the local maximum.

5 Conclusion

The goal of our paper was to recreate and improve on an existing project in which animal motion capture data was used to train a four-legged robot. We wanted to improve on this existing model by introducing physics as a factor in the training of the motion policy. We introduced a physics constraint term into the reward function, as we felt that this would help the robot better simulate real-world conditions.

Visually, our model and the base model seem to perform fairly similarly in the simulation environment. Our model has slight deviations from the reference motion, but we believe this is due to the extra constraints in the reward function and that the real-world robot will benefit from these added constraints. Furthermore, we see that the reward values on both models are quite similar, showing that the constraint does not have a significant negative impact on the model. However, both qualitatively and quantitatively, we are unable to conclude if our model is an improvement without more training time and a physical robot to test with.

In the future, it would be highly beneficial for this codebase to be updated to a modern Python version, as it currently runs on Python 3.7. Doing so would provide computing benefits, while also making it more accessible for others to perform their own experiments on this model. We also believe that it would be good to try to implement more variability in the simulation environment. It already implements randomness such as pushing forces on the robot, variable ground resistances, and varied starting positions. but we think adding more randomness in the form of objects that the robot may have to step over or more aggressive random forces on the robot would help increase real world generalization. Finally, as previously mentioned, it would be ideal to have the chance to test this new model on a real-world robot. We believe that our model is much more effective in sim to real transfer, and to ensure this, we would like our model to be tested on a robot that is subject to real world conditions.

Appendix

Github Link

Project repository: Motion Imitation GitHub

If above link doesn't work, repository is at https://github.com/pmaddire/228_Final

Team Member Contribution

Pranav: Novelty conceptualization, baseline code analysis, methodology creation, experiment implementation, poster preparation, report writing
Akhilan: Novelty conceptualization, baseline paper analysis, methodology creation, poster preparation, poster printing, report writing

Course Evaluation

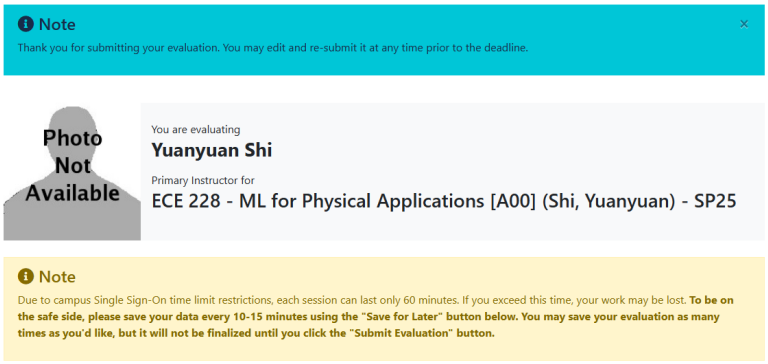

Figure 8: Pranav's Proof of evaluations

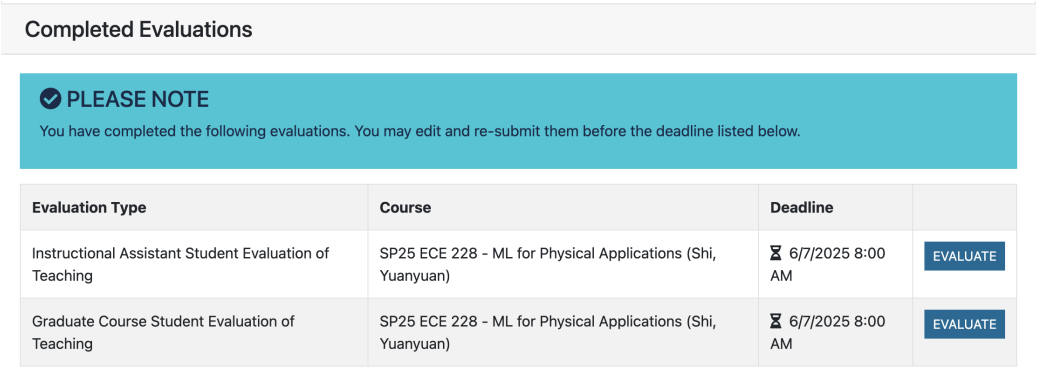

Figure 9: Akhilan's Proof of evaluations

References

X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3803–3810, 2018. doi: 10.1109/ICRA.2018.8460528.

X. B. Peng, E. Coumans, T. Zhang, T.-W. E. Lee, J. Tan, and S. Levine. Learning agile robotic locomotion skills by imitating animals. In *Robotics: Science and Systems (RSS)*, 2020. doi: 10.15607/RSS.2020.XVI.011.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.