# Pawlicy - Learning to walk

Author: Roshin Rajan Panackal, Abdul Rahman ElKebbi, Emrald Abraham Thadathil
Supervisor: Baohe Zhang

04/08/2022

## 1 Objectives

In the original locomotion paper by Peng et. al.[1], they taught a four-legged robot to perform actions similar to that of an animal by using imitation learning. The algorithm they used to train the agent was Proximal Policy Optimization(PPO)[2]. In this project, We try to benchmark the results of different Reinforcement learning algorithms for a simple locomotion task at two levels of control. The algorithms that we chose to benchmark are the Soft Actor Critic algorithm (SAC)[3], the Twin Delayed Deep Deterministic Policy Gradient algorithm (TD3)[4] and the Proximal Policy Optimization algorithm. The github repository containing the final implementation can be found here[1].

In practice, people often choose reward function on one's intuition, which involves the risk of encouraging undesirable agent behavior. So we would also like to study the aspects of reward formulations and it's influence on the gaits learned by agent.

## 2 Environment Configuration

Unitree Robitcs' A1 robot, a quadruped with 3 actuators on each leg, was setup in Bullet physics simulation environment.

**Observation Space**: A 45D vector composed of trunk displacement, trunk orientation, trunk angular velocity, motor angles, motor velocities and motor torques.

**Action Space**: A 12D vector for motor angles, one in each of its 3 joints in 4 legs.

**Task**: Walk along a path with minimum deviation.

**Benchmarking**: Algorithms benchmarked are SAC, PPO, and TD3. They all use ADAM optimizer for training and a detailed list of hyperparameters are mentioned in the Appendix section.

**Terrain**: We setup randomized terrains to test how well policies learned of flat terrain extends to coarse terrains.

**Controls Modes**: Two control modes are setup, position control (high level) and torque control. Action space remains the same in both setup. Position control translates to manipulating motor angles in our use-case. In torque control setup post-processing is applied, by using a motor model to convert predicted target positions to motor torques.

[1]https://github.com/EmraldT95/pawlicy

Actions, observations and rewards were normalized to the range [-1, 1]. Actions are also smoothed to avoid jerkiness and clipped to constraint it within the action space.

## 3 Challenges Faced

The effectiveness of your reward function is intimately tied up with how you specify the observation space, as inadequate observation cripples learning. Initially, then agent behavior didn't exhibit presumed gaits or correlation with the formulated reward functions due to inadequate information in observation space.

High level control (position-based) on Bullet physics environment failed to enforce the velocity and torque limits, leading to agent exceeding physical limits of a quadruped animal and learning unnatural gaits.

## 4 Reward Engineering

### 4.1 Approach

Pick an objective as the pivot and scale other objectives accordingly around it. We identified orientation based reward objective as the best suited as a pivot based on mean episode length and reward during evaluation. The objective weights were hand-tuned.

**Observed Trend:** Positive rewards encourage reward accumulation and Negative rewards (penalties) encourage reaching a terminal state.

### 4.2 Our Reward Function

$$\mathbf{r}(\mathbf{s}, \mathbf{a}, \hat{\mathbf{s}}) = w_0 * cos(\alpha) * cos(\beta) + w_1 * \Delta x - w_2 * ||a||_2 \quad (1)$$

where $\alpha$ and $\beta$ are Euler angles about x and y axes, $\Delta$x – displacement along x-axis, a – action vector and $w_{0:2}$ are objective weights.

The reward weight were as follows, $w_0 = 1$, $w_1 = 800$ and $w_2 = 4 \times 10^{-3}$

### 4.3 Strengths

- Leverage implicit constraints of a reward objective to control the complexity of the reward function.

- Adopt a continuous and differentiable reward function to avoid sparse environment feedback and streamline modeling of value function.

- Correlate the reward with the observations to facilitate information for the agent to base its decisions on.

- Avoid accumulation of penalties over rewards to discourage suicidal behavior – reward shaping.

- Scale the reward objectives considering if one objective outweighs the others, the agent will exploit that, even at the expense of failing the other objectives – reward scaling.

## 5 Experiment Results

Each algorithm was trained for 5e5 steps. As shown in figure 1 the critic loss of SAC and TD3 algorithms drops quickly. During training, the neural net policy was evaluated every 5000 steps. The best model was saved using criteria of mean evaluation reward. Another criteria using inverse coefficient of variation, $C_v^{-1}$, in-order to checkpoint policies that scores consistently well, was also experimented with but discarded due to time constraints.

$$C_v^{-1} = \mu/\sigma, \tag{2}$$

where $\mu$ and $\sigma$ are mean and standard deviation of evaluation returns.

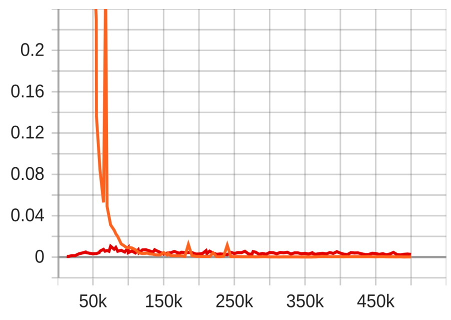

Figure 1: Critic loss of SAC (orange) and TD3 (red) algorithms

In figure 2, we can seen that, under the same configuration, high level actuator control performed poorly. There is negligible improvement is distance gained from initial point.

Laikago motor model in [5] was used to convert target position to motor torques in low level actuator control. There is a noticeable improvement in maximum distance gained from initial point and best evaluation reward.

The final scores of each of the algorithms in both control modes are given below in table 1.

| Mode | SAC | PPO | TD3 |
|---|---|---|---|
| High-Level control | 647.59 | 897.67 | 1,141.55 |
| Low-level control | 10,044.8 | **11,214.83** | 6,052.4 |

Table 1: Comparison of best evaluation rewards

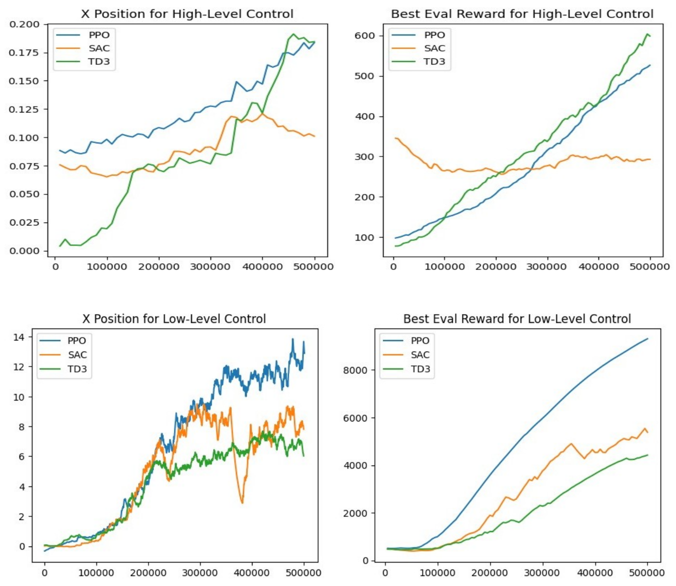

Figure 2: *(left)*: Shows agent progress along designated path across training steps . *(right)*: Shows how agent performance improved based on best evaluation rewards

## 6 Summary

We saw minimum drift from path, without imposing explicit reward or penalty, by leveraging implicit constraints through displacement reward.

All algorithm performed relatively well, while PPO scored the best in terms of mean episode length and rewards. Noticeably, PPO also achieved highest trunk velocity while maintaining a small range of applied actuator torques.

Actuator control modes of Bullet physics has significant impact on modeling of value functions. In our experiments we observed low-level actuator control promote learning and outperformed high-level control. Utilizing suitable actuator model to produce low-level control inputs can help resolve the situation.

## 7 Future Scope

Learn a policy on randomized terrains and study how well they generalize. We may also use approaches like Inverse Reinforcement Learning to learn the reward function. Hand tuning the reward scales are not only tedious but difficult to track. Hyperparameter optimization approaches may be used for reward scaling, instead of hand engineering the reward weights.

## References

[1] 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, 07 2020.

[2] 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

[3] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," CoRR, vol. abs/1801.01290, 2018. [Online]. Available: http://arxiv.org/abs/1801.01290

[4] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," CoRR, vol. abs/1802.09477, 2018. [Online]. Available: http://arxiv.org/abs/1802.09477

[5] Y. Yang, "Locomotion simulation : Python environments for unitree a1 robot," Sep 2021, accessed: 2022-8-7, https://github.com/yxyang/locomotion_simulation.

[6] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," Journal of Machine Learning Research, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[7] A. Raffin, "Rl baselines3 zoo," https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

## 8 Appendix

### 8.1 Hyperparameters

The implementation of algorithms in package Stable baseline were used to setup the training interface. The hyperparameters values are mainly adopted from Stable baseline's[6] and RLZoo[7]. Therefore, we chose to use their namespace for reporting the hyperparameter configurations.

**Twin Delayed DDPG (TD3)** : *n_timesteps*=$5 \times 10^5$, *eval_freq*=$5 \times 10^3$, *learning_rate*=$1 \times 10^{-3}$, *gradient_steps*=1, *batch_size*=100, *buffer_size*=$1 \times 10^6$, *n_critics*=2, *gamma*=0.99, *tau*=0.005, *policy*=MlpPolicy, *net_arch*=[400, 300], *activation_fn*=relu.

**Soft Actor Critic (SAC)** : *n_timesteps*=$5 \times 10^5$, *eval_freq*=$5 \times 10^3$, *learning_rate*=$3 \times 10^{-4}$ with cosine scheduling, *gradient_steps*=1, *batch_size*=256, *buffer_size*=$1 \times 10^5$, *ent_coeff*=auto, *n_critics*=2, *gamma*=0.99, *tau*=0.005, *policy*=MlpPolicy, *net_arch*=[256, 256], *activation_fn*=relu.

**Proximal Policy Optimization (PPO)** : *n_timesteps*=$5 \times 10^5$, *eval_freq*=$5 \times 10^3$, *learning_rate*=$2.5 \times 10^{-4}$, *batch_size*=64, *ent_coeff*=0, *gamma*=0.99, *clip_range*=0.2, *vf_coef*=0.5, *policy*=MlpPolicy, *net_arch*=[64, 64], *activation_fn*=tanh.

## 9 Contributions

Five different environments (1 Mujoco and 4 Bullet physics) were configured and 700+ hours of training was performed in total as a team for experimentation and reward engineering. Given below are the contributions of each member.

**Abdul Rahman ElKebbi** – Randomized terrain setup. Poster creation and printing. Reward Engineering. Plotting. Training of PPO and TD3 algorithms.

**Emrald Abraham Thadathil** – Environment setup(2). Utility functions incl. Tensorboard logging and checkpointing of best model. Github integration. Training of SAC, TD3 and PPO algorithms. Training interface.

**Roshin Rajan Panackal** – Environment setup(3). Reward engineering. Sensor configuration. Github integration. Training of SAC algorithm. Command line interface. Project report.