Humanoid Locomotion by Q-Learning, Discounted Cost Infinite Horizon, and PID Control

James Utley
utleyj@bu.edu

Project Code Repository: https://github.com/jam-utley/ec710_humanoid_locomtion

Figure 1: 16-headed training results displayed in IsaacSim

1. Introduction

Bipedal locomotion is difficult to simulate, let alone deploy in a physical system. However, this has not stopped researchers from creating ever more agile humanoid robots capable of tasks from boxing to acrobatics to menial labor in warehouse environments. One need not look further than Boston's own backyard where Boston Dynamics has been developing convincing humanoid movement for the past 30 years. A brief search on YouTube will show dozens of videos of their robot, Atlas, performing a number of complex tasks including parkour, item identification and sorting, and even dancing.

While this work is fascinating, these feats of engineering and agility fall out of the possibility for a term project, limited by both time and money. The goal of this paper is not to recreate high-flying robotic stunts, but rather to analyze the mathematics and algorithms behind this well-documented engineering problem, and to implement these algorithms into NVIDIA's IsaacSim and deploy them in a simulation.

2. Related Work

Before diving into the heart of the project, it is important to note the supporting work that has been done by earlier researchers into the simulation and deployment of reinforcement learning algorithms into robots.

<u>Reinforcement Learning in Simulation:</u>
Learning to walk in Minutes Using Massively Parallel Deep Reinforcement Learning : Rudin, Hoeller, Reist, Hutter

In this paper from 2021, Rudin et al. use NVIDIA's Isaac Gym (a variant of Isaac Sim used in this project) to parallelize the reinforcement learning of a quadripedal robot in a matter of minutes using GPUs

instead of programs running on the CPU. Using thousands of identical robots, they simulated many different terrain types for the robot to walk on. While the focus of this paper is on the efficacy of parallelising reinforcement learning, its applications and similarities with this project made it both applicable and informative to read. The team even introduced a number of familiar concepts like infinite horizon, discounted cost, and Proximal Policy Optimization (PPO) which, though not completely known from EC710, bears many similarities to policy updates and optimizations like Q-Learning.

Proximal Policy Optimization Algorithms : Schulman, Wolski, Dhariwal, Radford, Klimov

Interested in the previous paper, the author did more research on PPOs and found this 2017 paper. In it, Schulman et al., all researchers at OpenAI, lay out the groundwork for the optimization algorithms. Often they made comparisons to Q-Learning, laying out the failures of Q-Learning and proposing PPOs as an alternative. While this project does not use PPOs since its purpose is to demonstrate knowledge learned in EC710, but future directions of the project would analyze the deployment of PPOs and consider their use as a replacement for Q-Learning.

DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills : Peng, Abbeel, Levine, Van De Panne

In research published in 2018, Peng et al. describes the process of deploying reinforcement learning in a physics-based simulator. The paper uses motion tracking of human motions as a model for kinematic movements to be trained to the simulated robots, as well as a defined reward function. These models also use Proportional-Derivative (PD) control as a means of determining system torques.

However, unlike in other deployments reviewed in this project, this paper uses a Neural Network for policy training and determination. This is mainly a byproduct of using mocap as a primary training dataset, requiring deeper networks to process and translate the data to something usable by the robot. While this is slightly out of the scope for this project, the author wanted to include it as a fascinating deployment of reinforcement learning.

Finally, this paper outlined many topics that were familiar from EC710, including a detailed, mathematical interpretation of the algorithm for policy determination and highlighting the Bellman equation.

Learning Agile and Dynamic Motor Skills or Legged Robots : Hwangbo et al.

This 2019 paper lays out the process of performing Reinforcement Learning in simulation and its deployment in a real system. Hwangbo et al. engages in an analysis of contemporary advances in reinforcement learning and the applications of Proportional-Integral-Derivative (PID) control. The paper describes how deployment in real systems is limited by the quality of the simulations as well as the physical systems and perception in the real world that lead to further difficulties, even when starting with a fully trained policy. Overall, the paper was interesting, with many topics inspiring ideas about the direction of this project.

<u>SAC</u>

Soft Actor-Critic (SAC) is a derivative of Q-Learning and exists as a part

of a long line of algorithmic alterations to Q-Learning for different applications. Q-Learning was originally proposed in 1989 [9], but had significant limitations in its applications, namely that the algorithm required discrete valued states in order to calculate the argmin of the matrix Q (see further description of this problem in Section 3, below). Improvements using neural networks were made to the algorithm in 2015 in Mnih et al. [5] and Lillicrap et al. [4] which sought to use neural networks as well as introducing an Actor [4]. Then in 2018 Fujimoto et al. [1] added two critics to the method. Finally Haarnoja et al. [2] added entropy regularization to create SAC. Therefore, SAC is a descendant algorithm from Q-Learning and utilizes similar mechanisms adapted for continuous values. For a more in depth look at the structure of SAC, see Section 3.

3. Method

To apply reinforcement learning to the complex task of a humanoid, bipedal robot walking, the theory is primarily grounded in Q-Learning and infinite horizon problems with a discounted rate. Q-Learning is a good selection for this project because on top of being the best example of reinforcement learning present in EC710, it also allows for the updating of policy decisions from a pre-defined course. However, as will be described below, naive Q-Learning is intractable for this problem, so a variant of Q-Learning must be used: Soft-Actor-Critic (SAC). Infinite horizon theory is also applicable to this problem because this project does not seek to train walking where the end state is known, but in the more generalizable walking action where the destination is not necessarily predetermined by the algorithm.

With the grounding of the theory as described in depth in the following section, this project seeks to apply this learning schedule in IsaacSim, a physics-based simulated environment used for reinforcement learning.

4. Description of the Theory

Description of the problem

Bipedal humanoid locomotion is a complex task. The state space exists of around 43 continuous and discrete value states for the leg motion alone. A detailed accounting for each of these states is given below:

- Joints (12 states * 2 legs = 24 states)
  - Angles [-π,π] rad
    - Hip: Roll, Pitch, Yaw
    - Knee: Pitch
    - Ankle: Pitch, Roll
  - Ang. Velocity [-20,20] rad/s
    - Hip: Roll, Pitch,Yaw
    - Knee: Pitch
    - Ankle: Pitch, Roll
- Body (9 states)
  - Orientation [-π,π] rad
    - Roll, Pitch, Yaw
  - Angular Velocity
    - ω_x, ω_y, ω_z
  - Linear Velocity
    - v_x, v_y, v_z
- Feet/Ground Contact (6 states)
  - Contact Binary {0, 1}
    - Left foot
    - Right foot
  - Ground Reaction Force [0, 800] Newtons
    - Left foot
    - Right foot
  - Zero Moment Point (Balance) [-0.1,0.1] m
    - Left foot
    - Right foot

- Task and Context (5 states)
  - Target Velocity Command [-1.5,1.5 m/s]
    - $v_{x,cmd}$
    - $v_{y,cmd}$
    - $yaw_{rate,\,cmd}$
  - Gait Phase (Cyclical)
    - $sin(\phi)$
    - $cos(\phi)$

The allowable controls for this problem perform four key functions: maintaining desired joint position, center of mass balancing, zero moment point control (also balance), and maintaining foot contact when needed. The allowable controls of these four loops modify the state spaces defined above to keep them as close as possible to the desired path.

Q-Learning - SAC

Q-Learning is the heart of this project, and the theory that the reinforcement learning aspects of this project primarily utilizes. Q learning uses the titular $Q_{\mu}(i,u)$, where the transitional probability p_{ij} is known, which is defined as the cost of applying the policy u at time i, and then following the stationary policy μ until the terminal state is encountered. This can be expressed as the following:

$$Q_{\mu}(i,u) \equiv \sum_{j} p_{ij}(u)[g(i,j,u) + J_{\mu}(j)]$$

After each epoch, the optimal stationary policy can be updated according to the following equation:

$$\mu^{new}(i) = \arg\min_{u \in U(i)} Q_{\mu^{old}}(i,u)$$

The optimal Q function can be found through a value iterative algorithm according to the following equation:

$$Q^{new}(i,u) = \sum_{j}[g(i,j,u) + \min_{u' \in U(j)} Q^{old}(j,u')] \; \forall i,u$$

However, this value iteration of the matrix Q implies that the transitional probabilities between the states are known, which is not possible for the scenario of this project. Instead, the Q matrix can be iteratively updated after each epoch according to the following equation:

$$Q^{new}(i,u) = Q^{old}(i,u) + \gamma[g(i,j,u) + \dots$$
$$\dots + \min_{u' \in U(j)} Q^{old}(j,u') - Q^{old}(i,u)]$$

In the equation above, γ must be selected such that it satisfies the Robbins-Monro criteria, which ensures result monotonicity, bounded noise, and a reasonable step size (small enough to ensure the disappearance of noise, but big enough to reach the root of the function). These conditions can be met by setting γ as a function of the iteration:

$$\gamma_k = \frac{b}{m(k)}$$

Even this process is intractable for this project. Namely, the optimal stationary policy μ requires the maximum Q to be knowable by enumerating all possible actions, which cannot be done with continuous variables. There are a few solutions to this, like using a Deep Q-Network (DQN) to enumerate values of each state combination, but even a DQN is unable to faithfully compute the argmax of Q.

Therefore, this project considers the use of a Soft Actor-Critic (SAC) as a replacement to the pure Q-Learning algorithm. SAC can be split into three components: the Critic, the Actor, and the entropy regularization. The Critic component is the most similar to Q learning, because it uses two Q networks with implicit Bellman equations to reduce overestimation bias. SAC can be defined according to the following equation:

$$y = r + \gamma[\min_{u \in U(i)} (Q_1(i,u), Q_2(i,u)) - \ldots$$

$$\ldots - \beta \log \mu(u,i)]$$

$$loss = E[(Q_1(i,u) - y)^2 + (Q_2(i,u) - y)^2]$$

Where r is a scalar reward and β is a scalar hyperparameter that controls the penalty for confidence. A higher β leads to a more spread out PMF and slower convergence, and a lower β is more deterministic, resembling standard Q-Learning as β approaches 0. In many deployments of SAC, there is a flag in the function that allows the algorithm to optimize automatically before the main algorithm begins.

The Actor component adjusts pure Q-Learning for continuous states. Instead of attempting to compute $\arg\max_{u \in U(i)} Q(u,i)$, the SAC Actor uses a separate neural network to compute the probability distribution over particular actions, maximizing for the expected value of the Q-values.

$$\mu_\theta(u \mid i) \sim N(mean_\theta(i), \sigma_\theta(i))$$

$$Loss = E_{b \in \mu}[\alpha log(\mu(b|i) - min(Q_1(i,b), Q_2(s,a)$$

Because argmax is not used, the SAC Actor makes the problem of bipedal locomotion feasible for continuous values.

The third and final component of SAC is the entropy regularization which prevents premature convergence and increases system stability. The entropy regularization is defined as follows:

$$J(\mu) = E[\sum \gamma^t (r(u_t, i_t) + \alpha H(\mu(u|i_t)))]$$

Where $H(\mu)$ is the entropy of the policy μ. The regularization improves the adaptability and generalization of the reinforcement learning training and allows for easier transition from simulation to reality.

This understanding of Q-Learning and SAC allowed for the deployment of this project into simulation successfully.

Infinite Horizon, Discounted Rate

Discounted cost infinite horizon theory allows for systems without a finite runtime to use the Bellman equation. Infinite Horizon problems utilize two important operators, T and T_μ which are defined as:

$$T_\mu(J)(x) \equiv E_w[g(x, \mu(x), w) + aJ^k(f(x,u,w))]$$

$$T(J)(x) \equiv \min_{u \in U(x)} T_\mu(J)(x)$$

With these definitions of the T operator, one can rewrite the DP Algorithm to be:

$$J^{k+1}(x) = T(J^k)(x)$$

These two operators must also meet the following three criteria:

1. They must converge to a unique fixed point
2. The cost satisfies a fixed point stationarity property
3. They must satisfy contraction and geometric convergence

Infinite horizon cost is therefore very helpful when applied to bipedal robotic locomotion. With the discount factor, a, because it allows the system to predict values for time steps greater than the current time step, but the discount factor ensures that after a certain number of steps, the discount factor decreases to 0 and can be neglected in calculations. With a limited number of future values for it to predict, the discounted cost infinite horizon becomes computationally possible and therefore gives similar benefits that the system might know if it were a finite horizon.

5. Simulated Deployment

The deployment of this project rested strongly on the example and repository set up and maintained by Rudin et al. [7] which has become an industry standard for reinforcement learning. The code they wrote works primarily for

quadrupedal robots, but there have been deployments of it using the prebuilt model, Cassie, a bipedal robot preset in IsaacLab.

There are a number of preset bipedal robot models available in IsaacLab and IsaacSim. The model deployed in this project is Unitree's H1. The choice of model was based on the available literature and ease of deploying SAC and infinite horizon theory. Moreover, the actuators of the H1 have PD control running as its own internal loop in the model which allows them to self-correct. Therefore, this model was ideal for the deployment of this project. The state space was slightly different from the projected theory, with the model requiring 46 states for the robot model, and 27 environmental states, for a total of 73 states.

This project was deployed on an AMD Ryzen 9 9900X 12-Core processor with an RTX5070ti GPU. Since there was only one CPU and GPU available to this project, and in order that the project could be performed and produce results in a reasonable amount of time, the scope of the training was limited to a single surface. As such, training took 45 minutes and 18 seconds.

For ease of deployment, this project uses the Python library SKRL, a reinforcement learning library for Python. This project uses SKRL for implementing SAC as well as implementing infinite horizon, discounted cost theory. SKRL offers a config-file setup that helps run the SAC actor, twin critics, a discount factor γ, as well as target updates.

6. Results

Training was run on 16 models at once for a total of 100000 steps in IsaacLab, NVIDIA's terminal based version of IsaacSim that streamlines reinforcement learning in the physics environment and creates results that can be displayed in IsaacSim. Data was collected in Tensorboard and reprinted in Python using matplotlib into the subplots listed below. All figures appear at the end of the document in larger form so as to be more easily seen.

As can be seen in Figure 2, the result of this training is inherently flawed. The estimates for Q1 and Q2 rise, monotonically increasing in an ascent resembling exponential growth.

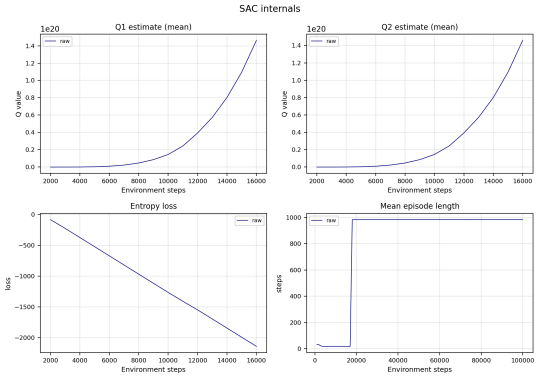

Figure 2: Results from SAC Q1, Q2, Entropy Loss and Mean Episode Length as a function of steps

As it increased, the mean episode length plateaued 20% of the way through the total 100000 step training regiment. Both of these point to an improperly tuned reward system that does not penalize negative actions strongly enough and doesn't reward good actions well enough to cause meaningful change in the performance of the models.

This result is further supported in Figure 3, which shows the SAC training curves.

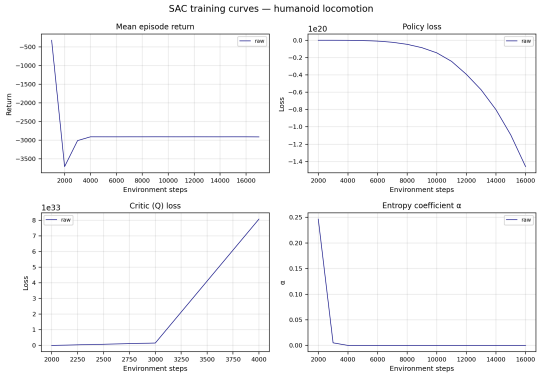

Figure 3: SAC Training Curves

These training curves further the idea that the reward function was not working properly. The critic loss is shown slowly growing the first several thousand steps, and then increasing rapidly after environmental step 3000. The Policy Loss likewise shows divergence from a converged point, meaning that the reward system, instead of pulling the system to converge to the desired goal, is instead pushing it further from proper performance.

The final Figure finalizes the idea that the reward function was flawed.

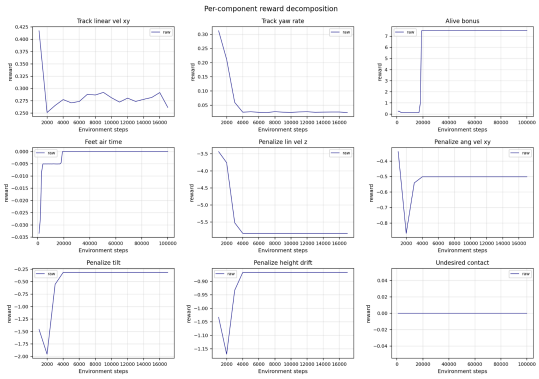

Figure 4: Per-Component reward decomposition across steps

Figure 4 shows the reward for each component as a function of the environmental steps. Rather than the smooth curves we would expect towards convergence, these graphs behave erratically in the absence of chaotic or erratic elements in the training environment itself, such as rough terrain. The penalties for linear velocity in the z direction and height drift max out with little variation after step 4000. The other graphs similarly settle at most at step 20000, as is expected, however the way that they stabilize with no perturbations after step 20000 implies that the reward system is either maxing out and the limits of the penalties need to be adjusted, or the rewards are not being properly applied and are improperly emphasizing the model's actions.

Ultimately, however, this is what was expected. Other implementations, namely Rudin et al [7], use many more steps and any more model instances to refine their simulations. This project is limited to a small number of models run at the same time (16) and a limited number of environmental steps (100000) to make the training time tractible for the purposes of this project. Rudin et al. [8] use varying numbers of policy updates and number of models, with one figure displaying 4000 models running 1000 policy updates. This was possible for the team due to their access to multiple higher performance GPUs than the single GPU this project ran on.

Therefore, even though the performance was not what was hoped upon the initiation of this project, the result was not unexpected and is not altogether a failure. Future directions will tweak this project's reward system as well as experimenting with the number of models that can be used on this machine to increase the throughput of the simulation.

7. Bibliography

1. Fujimoto, S., van Hoof, H., & Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. arXiv. https://doi.org/10.48550/arxiv.1802.09477
2. Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. arXiv. https://doi.org/10.48550/arxiv.1801.01290
3. Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., & Hutter, M. (2019). Learning agile and dynamic motor skills for legged robots. Science Robotics, 4(26), eaau5872. https://doi.org/10.1126/scirobotics.aau5872
4. Lillicrap, T. P., Hunt, J. J., Pritzel, A., et al. (2015). Continuous control with deep reinforcement learning. arXiv. https://doi.org/10.48550/arxiv.1509.02971
5. Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015). https://doi.org/10.1038/nature14236
6. Peng, X. B., Abbeel, P., Levine, S., & van de Panne, M. (2018). DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. ACM Transactions on Graphics, 37(4). https://doi.org/10.1145/3197517.3201311
7. Rudin, N., Hoeller, D., Reist, P., & Hutter, M. (2022). Learning to walk in minutes using massively parallel deep reinforcement learning. In Proceedings of the 5th Conference on Robot Learning (CoRL 2021). arXiv:2109.11978. https://doi.org/10.48550/arXiv.2109.11978
8. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv:1707.06347. https://doi.org/10.48550/arXiv.1707.06347
9. Watkins, C.J.C.H., Dayan, P. Q-learning. Mach Learn **8**, 279–292 (1992). https://doi.org/10.1007/BF00992698

8. List of Figures

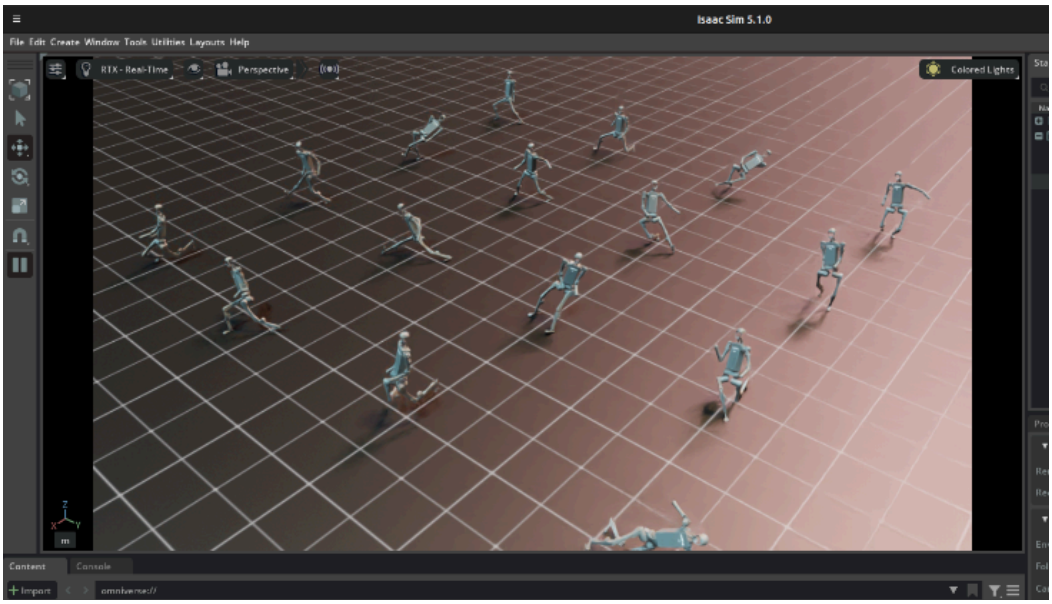

Figure 1: 16-headed training results displayed in IsaacSim

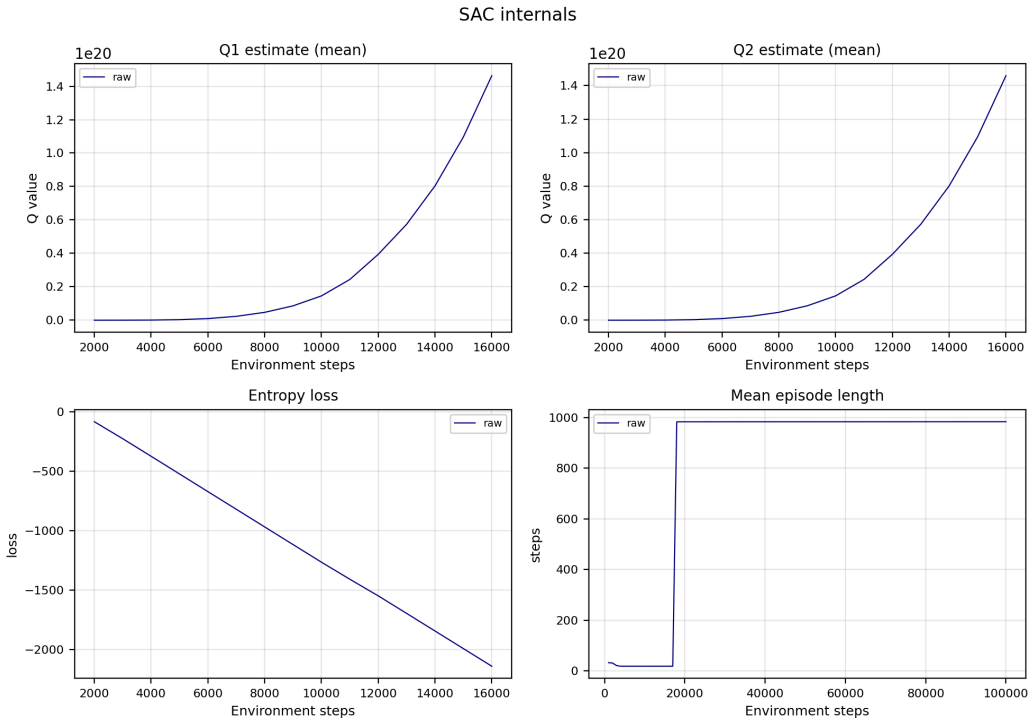

Figure 2 - Results from SAC Q1, Q2, Entropy Loss and Mean Episode Length as a function of steps

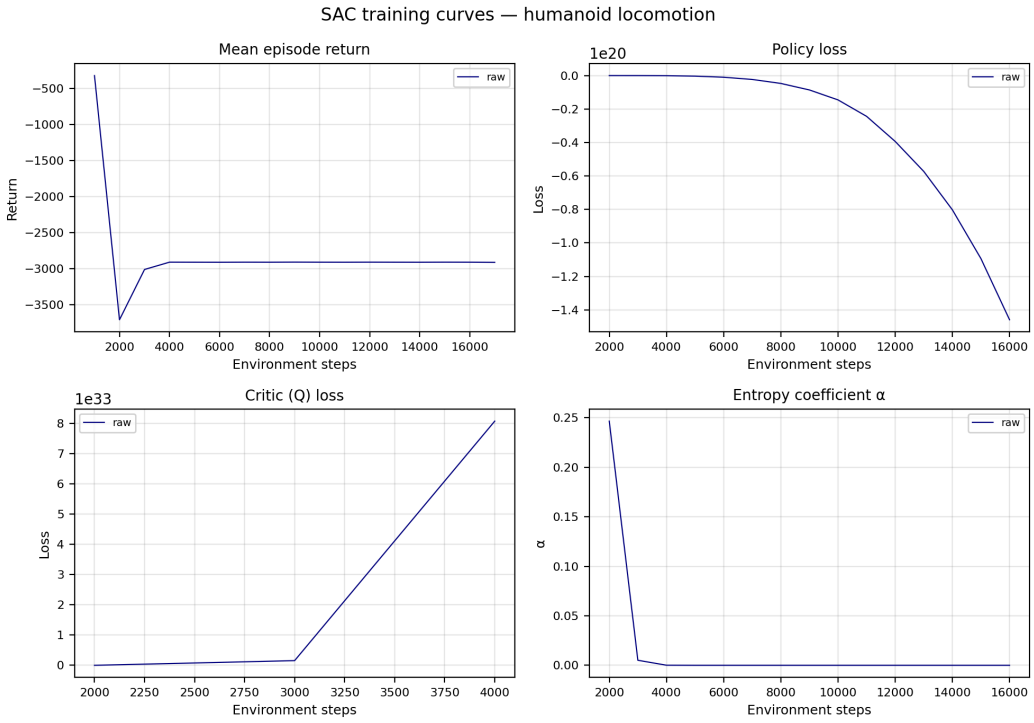

Figure 3 - SAC Training Curves

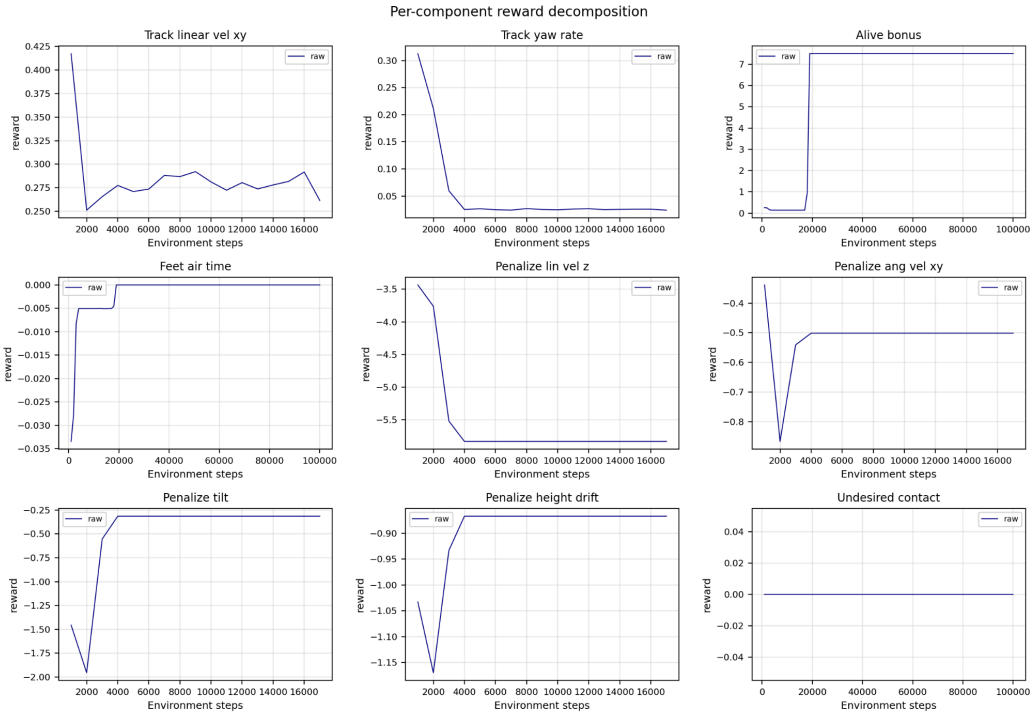

Figure 4 - Per-Component reward decomposition across steps