Designing a Learning-Based Controller for a Bipedal Robot

Julian Millan
Faculty Mentor: Soon-Jo Chung
Co-Mentor: Sorina Lupu

***Abstract*—This project aims to design and train a machine learning model that can be trained and implemented onto a controller. The controller (referred to as the policy) will be tested in simulation and have its robustness verified before being implemented on the bipedal robot hardware. The effectiveness of the policy will be verified in comparison with an "expert", which allows the robot to walk via a capture-point inverse kinematics system and inverse dynamics system. A successful test and implementation of this controller will result in the robot being able to walk autonomously and be able to learn how to walk on different surfaces other than the smooth and firm lab surface.**

I. INTRODUCTION

A. *Background Information*

The primary goal of this project was to establish an effective learning based controller to control a bipedal robot's walking gait in a novel direction. The novel direction chosen was to use behavior cloning on a capture-point inverse kinematics system and inverse dynamics system. The end goal was to have successful tests in simulation before transferring the model to hardware. Traditional control methods for bipedal robots like inverse kinematics (IK) and inverse dynamics (ID) provide effective solutions but lack flexibility in adapting to dynamic environments. A learning-based controller has the potential to adapt better by learning directly from expert demonstrations.

The current hardware is a passive ankle robot focused on leg motion planning and testing learning based controllers [1].

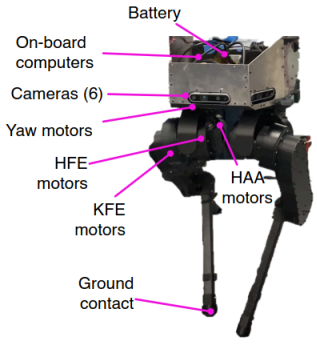

Figure 1: Current Hardware

A "capture point" is a desired point for the robot's foot to be placed during the walking cycle. The capture point system can be discretized in the below equation.

$$\mathbf{u}_m = -\boldsymbol{\xi}_{m+1}^{\text{des}} + \boldsymbol{\xi}_m e^{\omega T_s}. \tag{1}$$

We have *m* as a whole number, ξ as the capture point, **u** as the control input, T_s as the step time (taken to be constant), and ω as the natural frequency of the inverted pendulum model. This model will be referred to henceforth as the "expert".

B. Approach

Compared to kinematics, machine learning (ML) offers much more adaptability and robustness. Kinematics works well on known surfaces, but as focus in robotics shifts to unknown surfaces with varying texture. The adaptability of a learning controller offers more success in a rougher terrain.

We chose to focus on learning in legged robots as they are more robust to different environments such as rough terrain and steep hills [2]. This project involves developing and testing such a controller for a bipedal robot, improving its ability to walk and balance in varied terrains. Initial issues in policy convergence, data scaling, and pipeline stability have presented challenges but also opportunities for optimization.The project leverages behavior cloning, training a neural network to learn the expert policy . Data from simulations of the bipedal robot is collected, scaled, and fed into the model. The model is trained using various architectures, loss functions, and optimizers to improve its learning performance. Key goals include improving simulation results and ensuring consistent controller behavior in both simulation and hardware environments.

The two main types of ML used in the project were Behavior Cloning (BC) and Reinforcement Learning (RL). In particular, BC was used to imitate a dataset of proper walking behavior demonstrated by the robot in simulation using the expert. A form of offline RL known as "Conservative Q-Learning" (CQL) was used to attempt to improve upon the BC performance.

II. WORK DONE

A. Prior Work

Work done prior to my start on the project included initial robot simulation setup, basic BC model setup and pipeline setup. The robot simulation was done in RVIZ using Mujoco and ROS2. I would have to redo the ROS2 and Mujoco setup on my own device to do the project.

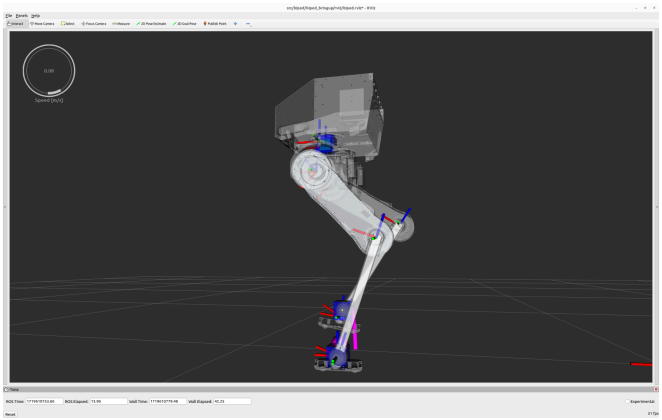

Figure 2: Image of the Robot in Simulation

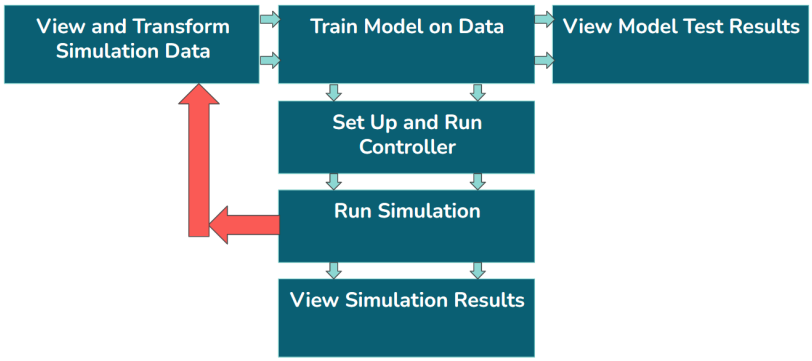

Figure 3: Old/Initial Pipeline Structure

B. Progress With Behavior Cloning

After further setup (downloading Linux, necessary code, and repositories), the first step was to verify the integrity of the pipeline. A few small changes were made to get the pipeline working and producing sensible results. The model consists of 41 joints or "states", each with their own sets of actions for a given time. Initial model test results indicated strong performance, while the simulation showed convergence issues.

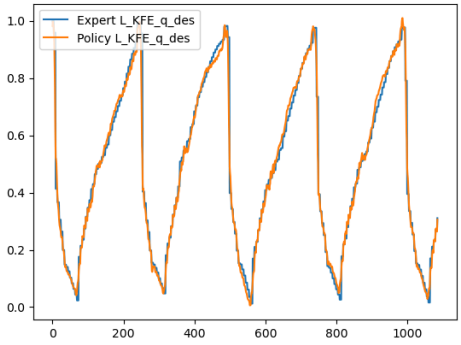

Figure 4: Test Results for One Joint

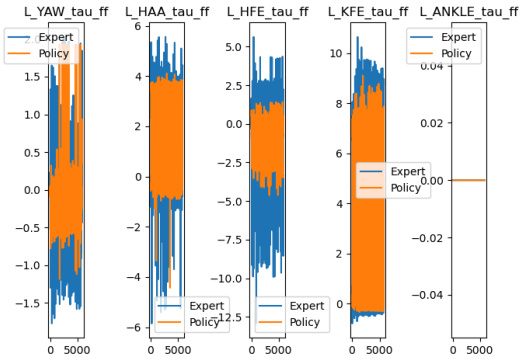

Figure 5: Simulation Results for Five Joints

Links to videos of the robot in simulation under different circumstances are provided below.

Simulation Conditions	YouTube Video Link
Biped Running with Expert Only	https://youtu.be/QXrxSlsH6GA
Biped Running with Expert and Policy	https://youtu.be/nL_OXZj_FPo
Biped "Running" with only Policy	https://youtu.be/SgNpyKnjCDs
Simulation with Mlpfile (Explained Later)	https://youtu.be/hKPzngbcWBk

Table 1: Simulation Conditions and YouTube Visualization

C. Major Issues and Fixes

As I started working on improving the model and looking for standout issues, I became aware of a variety of flaws in the project. One of the most obvious issues was the relatively slow simulation speed of the robot's walking. This was solved by introducing the mlpfile format, a more efficient BC model saving format that sped up the robot's walking in simulation. This can be seen in the last video in Table 1. Introducing this new format also resulted in noticeable simulation performance.

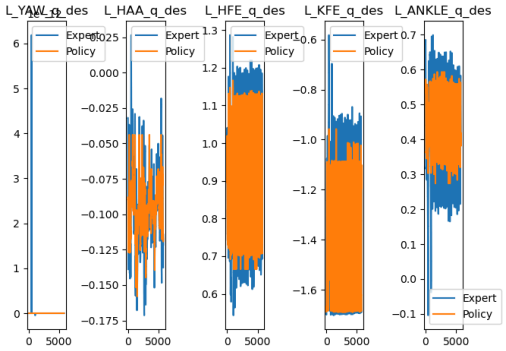

Figure 6: Before mlpfile on Old Policy Format

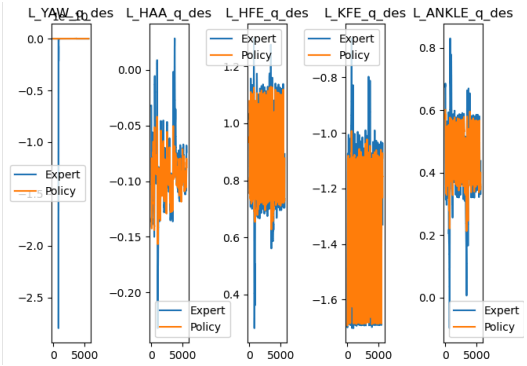

Figure 7: With mlpfile Format

Another issue identified was the poor data visualization for the robot performance post-simulation. I reprogrammed the visualization code to better set up the test data and simulation graphs.

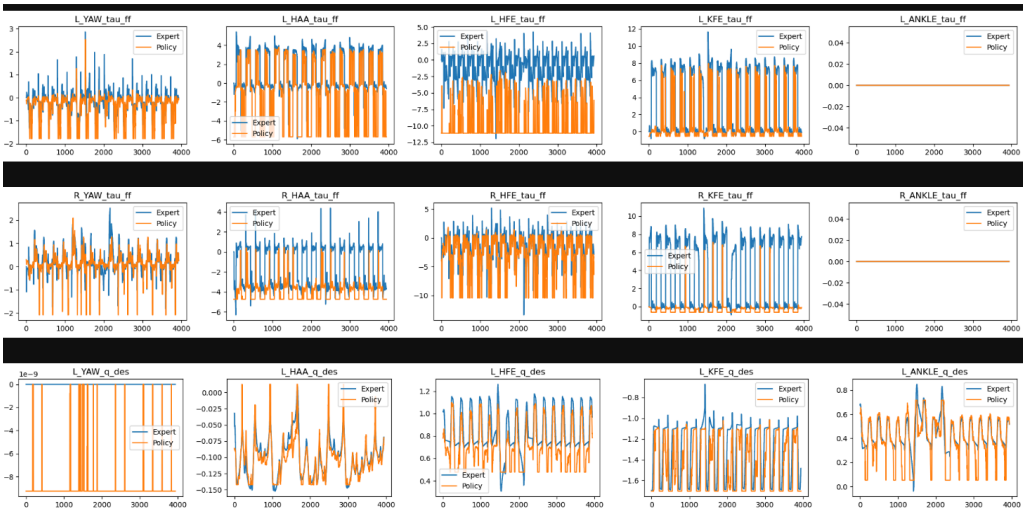

Figure 8: Improved Simulation Data Visualization

Progress in these areas then led me to investigate deeper issues: the model quality and pipeline integrity. The simulation at this point was still showing the robot crashing and post-simulation data still showed convergence issues (visible in Figure 8). Using TensorBoard, I was able to visualize the outputs of the loss functions on both the training and test data.

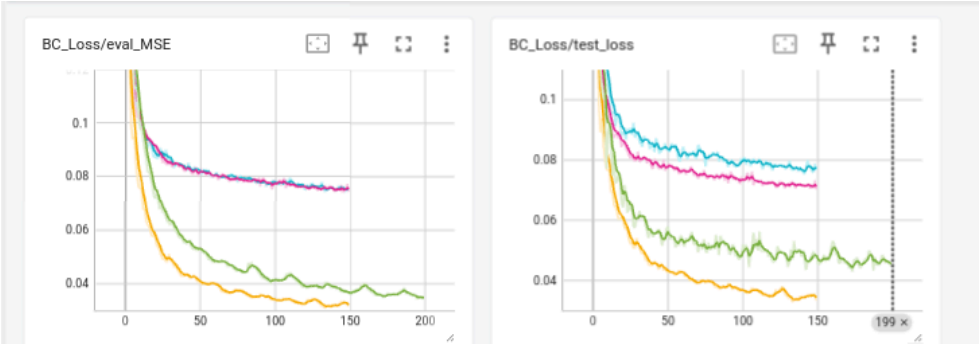

Figure 9: Visualization of the BC Model Test Loss and Training Loss Functions

The main point of this visualization is to help identify cases of over-fitting, that is if the loss function ever starts to increase after plateauing. This means we have essentially trained our model too much on the test data, and it won't perform well on new data outside of the test sample. TensorBoard results, which show this wasn't the case. Furthermore, the test data performance, with enhanced visualization, still indicated strong model performance.

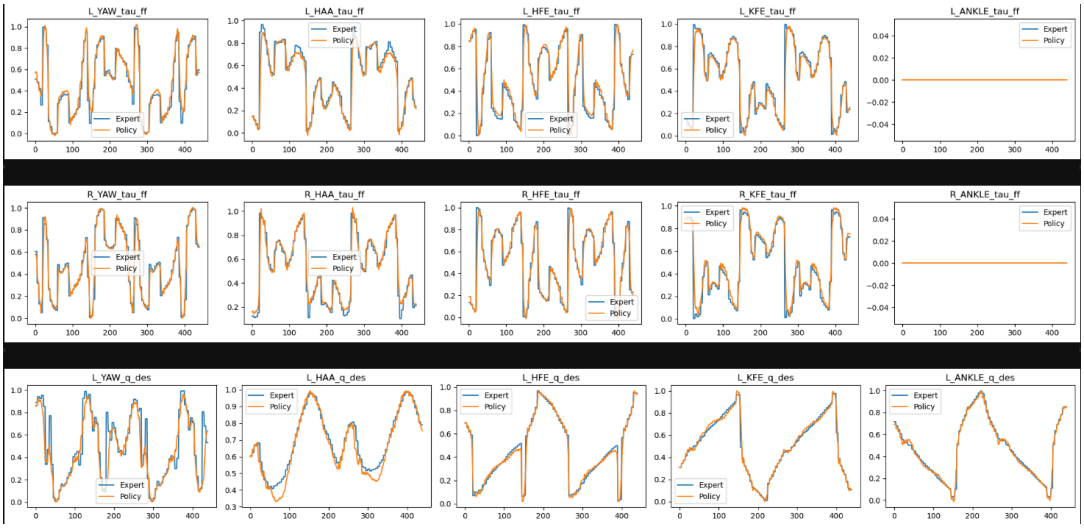

Figure 9: Improved Test Data Visualization

This information would then indicate that the poor simulation performance must have to do with the pipeline itself. Further investigation would show that running the pipeline, as shown in Figure 3, multiple times *without* changing any code resulted in different simulation results. This would mean somehow, repeatedly training or visualizing the robot's performance affects future iterations, which shouldn't happen. The results of the model training should be repeatable and identical.

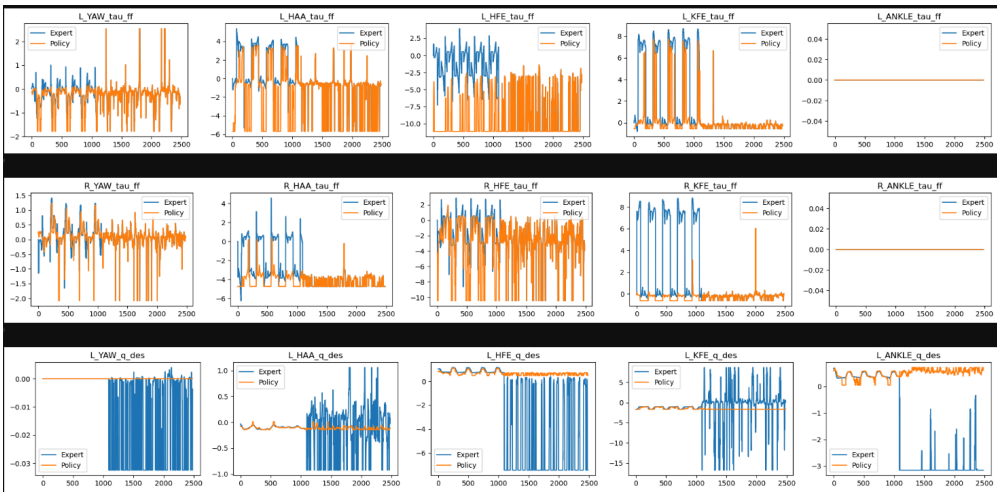

Figure 10: Robot Crashing in Simulation Data

D. Pipeline Verification

My first idea was to try running the simulation for longer to see if this would produce more training data or increase training time. If so, this would prove that simulation was being mistakenly recycled into future model training.

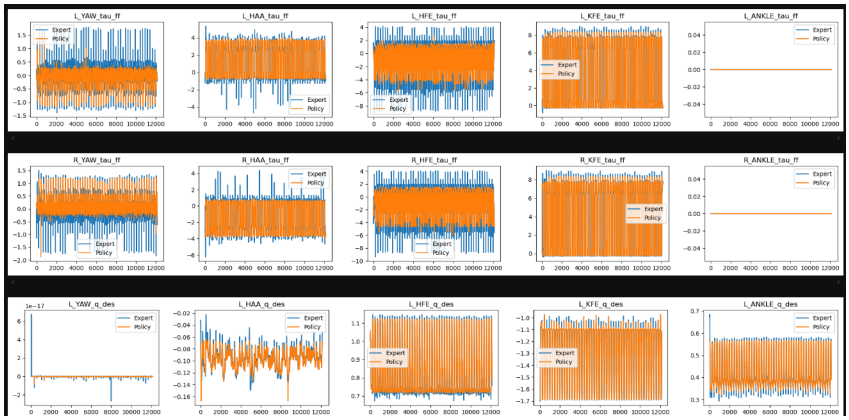

Figure 11: Longer Simulation Runtime

Running the simulation much longer than usual indeed resulted in much slower training time as suspected. This shouldn't happen, this was indicative of a pipeline error! The solution in this case would be to remove the red arrows as depicted in Figure 3, and have the pipeline run linearly instead of cyclically.

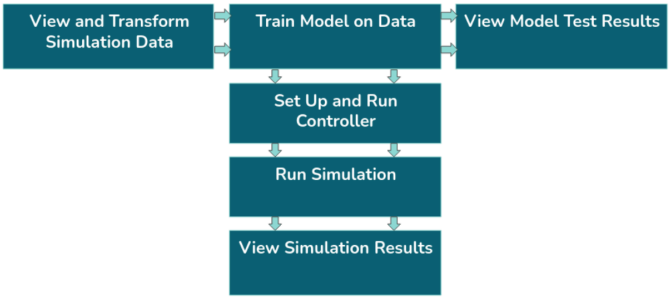

Figure 12: New Pipeline

This new pipeline would also require a new, static, training data file that includes much more robust behaviors. I manually created a new file with ~370,000 lines of data (the typical simulation output used to train was around ~7,000 lines). This file had behaviors beyond just walking in a straight line at a constant speed, such as walking in all directions at different speeds as well as quick direction changes. After this, training time went from ten minutes to four to five hours. Despite these changes, the model performance remained poor.

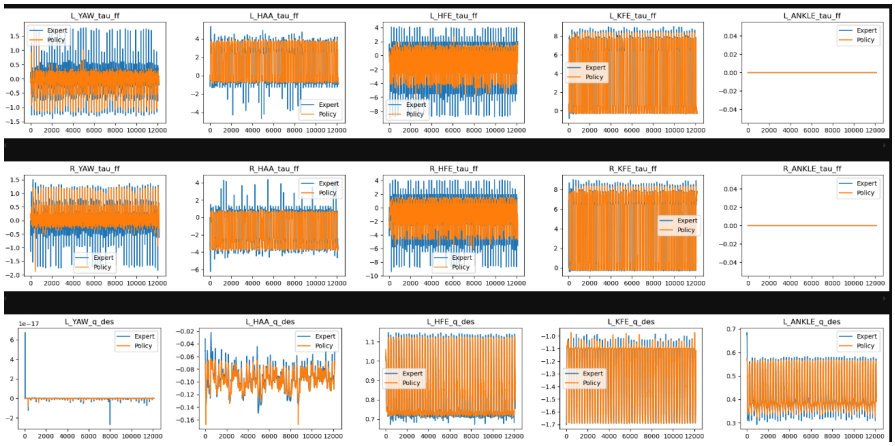

Figure 12: Model Convergence Issues Persist

III. WORK WITH CQL

A. *CQL Introduction*

After hitting a metaphorical wall with the BC model, it was time to try to use a more robust model type to see if the robot could be taught to walk in a different offline manner. Offline RL is a form of learning that uses a static dataset and attempts to use concepts like rewards and random action sampling to learn a behavior. Conservative Q-Learning (CQL) is a form of offline RL that introduces penalties for actions outside the training data distribution.

Algorithm 1 Conservative Q-Learning (both variants)

1: Initialize Q-function, Q_θ, and optionally a policy, π_ϕ.
2: **for** step t in $\{1, \ldots, N\}$ **do**
3: Train the Q-function using G_Q gradient steps on objective from Equation 4
$\theta_t := \theta_{t-1} - \eta_Q \nabla_\theta \mathrm{CQL}(\mathcal{R})(\theta)$
(Use $\mathcal{B}^*$ for Q-learning, $\mathcal{B}^{\pi_{\phi_t}}$ for actor-critic)
4: (only with actor-critic) Improve policy π_ϕ via G_π gradient steps on ϕ with SAC-style entropy regularization:
$\phi_t := \phi_{t-1} + \eta_\pi \mathbb{E}_{\mathbf{s}\sim\mathcal{D}, \mathbf{a}\sim\pi_\phi(\cdot|\mathbf{s})}[Q_\theta(\mathbf{s}, \mathbf{a}) - \log \pi_\phi(\mathbf{a}|\mathbf{s})]$
5: **end for**

Figure 13: Explanation of CQL

B. *Work Done with CQL*

While working with CQL, there immediately became clear and obvious differences in how the model would perform and how to tune it. CQL is much harder to use as it has many more parameters to tune and a more sensitive loss function. CQL also relies on more architecture. Instead of a policy and policy manager, it requires a sample buffer, a policy, a Q-Network, and a manager. One of the more parameters to turn is the "cql_alpha" parameter, which scales the penalty for out of distribution actions. In Figure 14, this parameter is set to 0.5 for the green line, and 0.1 for the purple line.

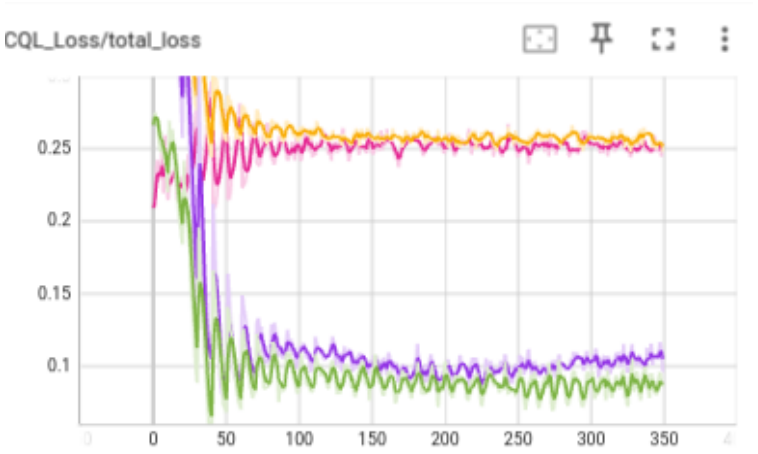

Figure 14: CQL Loss Function Performance

Overfitting was a stronger concern for this model, but once resolved performance issues still persisted.

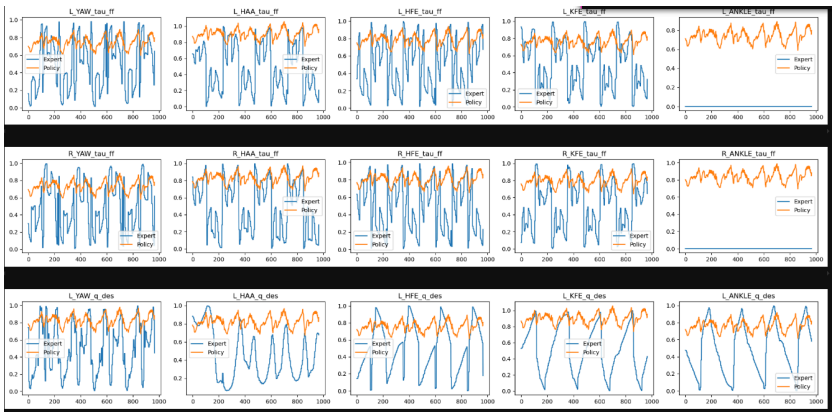

Figure 15: CQL Test Results

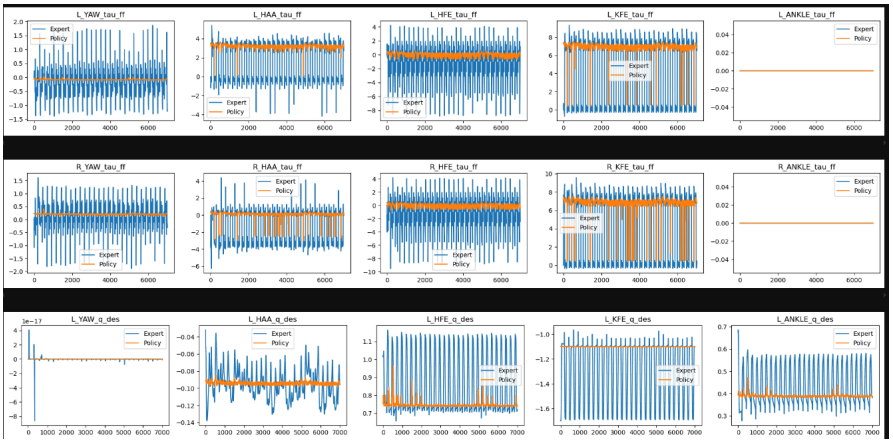

Figure 16: CQL Simulation Results

At this point, the time for working on the project was up.

IV. CONCLUSIONS

A. General Conclusions

Overall, improvements and progress were made on the pipeline and model training. However, getting the data to converge enough for consistent walking remained elusive. I also learned more about the different types of offline ML models. While more robust and "better" at learning behaviors, offline RL is generally much more difficult to use and isn't necessarily best suited for this task. When a BC model encounters slightly out of distribution states, it has a strong potential to diverge from the typical behavior completely (generally avoided by CQL). This causes the robot to crash. When it comes to these models, testing on simulation is much more convenient than testing on hardware. These models are more difficult to implement than online RL, the most common ML model in walking robotics.

B. Personal Takeaways

I came into this project with no prior ML experience as my undergraduate major is in Mechanical Engineering and a lot of my focus up to this point has been on hardware. It was difficult to learn ML from scratch and pick up all sorts of new topics during this project, but I am grateful to have gained so much knowledge in an area of robotics I have never worked in before. I walk away with a better understanding of ML in the robotics field, such as how very "black box" it is in nature and the value of discrete, kinematic, robotic control systems.

C. Future Work

Both the CQL and BC models have room for improvement, whether it's continuing to adjust the model parameters or modifying the model architecture. There may be some further issues with the simulation or pipeline that clamp the values of the BC model incorrectly that need to be fixed. The training data collection wasn't done autonomously, it was done via me steering the robot on an Xbox controller. An autonomous training data collection method that is more steady and robust could improve the model performance. Perhaps the right model hasn't been identified yet, and a tertiary offline ML model could be used that would do better than both BC and CQL. Once the biped gets walking, it could be taught novel offline behaviors such as hurdling.

ACKNOWLEDGEMENTS

I would like to thank my mentor Soon-Jo Chung, Co-Mentor Sorina Lupu, the SURF/SFP program and organization, and give a special thanks to the Class of '52 Fellowship for their extra support.

REFERENCES

[1] Elena-Sorina Lupu, Learning and Control of Legged Robots Traversing Unstructured Terrain for Agile Space Exploration. 2023.

[2] C. Gehring, P. Fankhause, L. Isler, R. Diethelm, S. Bachmann, M. Potz, L. Gerstenberg, and M. Hutter, "Anymal in the field: Solving industrial inspection of an offshore hvdc platform with a quadrupedal robot,' in *Field and Service Robotics* (G. Ishigami and K. Yoshida, eds.) (Singapore), pp. 247-260, Springer Singapore, 2021.