Interim Report 1

By Jade Millan

Background

The current project is focused on designing a learning based controller for a bipedal walking robot. This robot is motivated by the advantages of legged robots [1]. Legged robots, in particular compared to wheeled robots, are better at navigating complex terrain, such as terrain with lots of bumps or hills. Recently, legged robots have been applied in cave exploration, package delivery, and inspection tasks for offshore power plants [2]. Legged rovers would also be a great boon for space exploration. The surfaces of Mars and the Moon are rocky and mountainous, meaning that current wheeled rovers have to take great care when planning their paths, leading to more inefficient and ineffective exploration. Current work in the Autonomous Robotics and Control Laboratory (ARCL) has consisted of designing a bipedal robot robust to rough terrain. It is a passive ankle robot focused on leg motion planning and testing learning based controllers. It is depicted below:

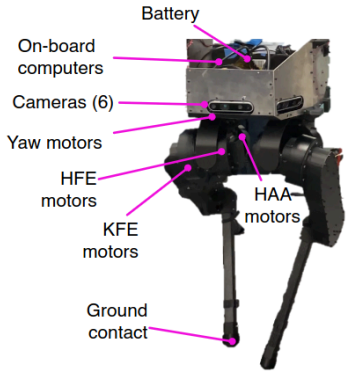

Fig. 1: Biped

Currently, the robot walks via a capture-point inverse kinematics system and inverse dynamics system. This was derived from modeling the leg movements as an inverted pendulum. A “capture point” is a desired point for the robot’s foot to be placed during the walking cycle, and it consists of both position and velocity dimensions. The series of capture points the robot walking cycle will follow 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 type of controller and programming certainly is effective for the flat and firm lab surface, but it lacks the adaptability and robustness a learned controller would have on different surfaces.

Approach

Currently, some software architecture is in place to create a learning controller, but there are still a few issues with its implementation. It interfaces well with the simulation, but post-simulation analysis indicates that the learning controller is not converging well enough to the data. Below is one example of the graphs generated to analyze the controller performance post-simulation.

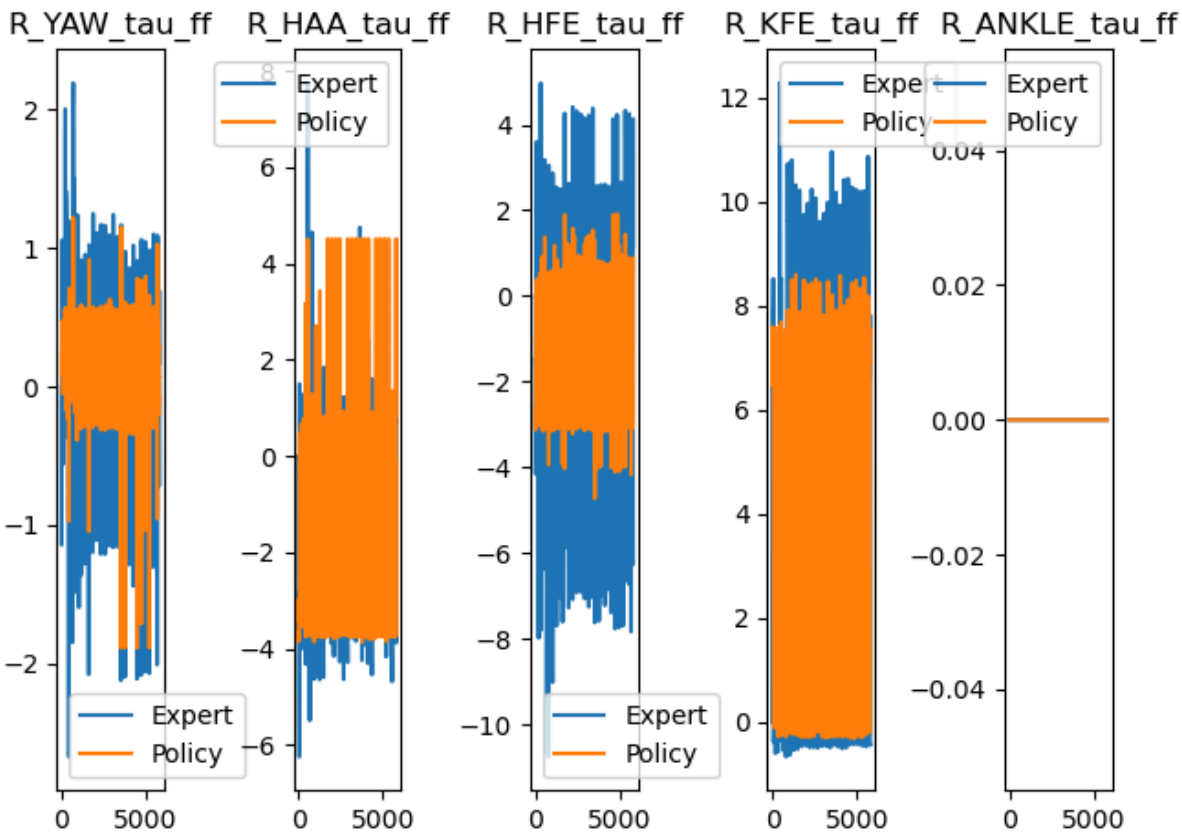

Figure 2

The above figure compares the feedforward torques generated during the simulation by both the inverse kinematics (“Expert”) and the learning controller (“Policy”). While the general shape of the policy is right, it appears to be incorrectly scaled to follow the expert. I am expected to find out the scaling issue and reformat the data so that the policy converges better onto the expert. Additionally, I am expected to speed up the code process as the controller can take too long and cause simulation lag. The simulation being slowed down is less of an issue, as focusing on producing the correct results should take priority.

Work Done

The past couple of weeks have been dedicated mostly to setup and learning key background information. Setup consisted of dual booting Linux and Windows, downloading necessary software and repository on the new Linux computer (such as ROS2 and Mujoco), and cloning the current biped repository with all of the code for running the learning controller on the simulation. This has included successfully running the simulation and the learning controller while doing basic training on it. Further work has included reading the current biped paper by the ARCL as well as starting some reading on imitation learning. Some progress has been made on improving the learning controller, but a clear issue with convergence remains. Attempts to fix the learning controller have mostly been centered around changing how the data is scaled and have mostly been unsuccessful. Attempts have included: using different scalers such as Normalizer and StandardScaler instead of QuantileTransformer from sklearn (a Python machine learning library) on the input data, unscaling the input data completely, using a MinMax scale on the input data *after* scaling it with QuantileTransformer. The last idea was motivated by the fact that the data maintains a good shape horizontally, but perhaps needs to be scaled vertically to better fit the Expert.

Challenges and Problems

The main problem of the scaling issue remains, as I have yet to resolve the issue. I have reached out to Leo Zhang, the student who worked on this project last year, for his thoughts as he was the one who originally suggested that this issue might be the cause of the inaccurate learning. A further challenge is my general lack of machine learning knowledge. While I have taken multiple robotics courses and an introductory controls course, I have yet to learn machine

learning formally. Learning about model training at the same time as trying to *fix* a machine learning model is proving to be difficult. I anticipate that, assuming the software gets resolved, further issues may be encountered when testing the algorithm on the robot's hardware, or on different surfaces. Changing the scaling of the data to better help the Policy converge to the Expert may also result in unexpected issues.

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.