Aditya Pulipaka

COE379L – Joe Stubbs

Project 3 Final Report

Developing Simulation Environment and RL Pipeline for Hybrid EM Levitation

Introduction and Problem Statement

The transportation of tomorrow will be defined by efficiency and speed. To maximize these two factors, the reduction of friction is paramount, resulting in the conception of the magnetic levitation train, and, soon after, the hyperloop. This idea gave birth to Texas Guadaloop, UT Austin's very own hyperloop engineering student organization. As the lead of the levitation sub-team, I have been pondering for months on how to develop not only a robust control algorithm, but a system that would allow for repeated success in HEMS (Hybrid Electromagnetic Suspension) control, which utilizes a u-shaped 'yoke' with two permanent magnets covered with copper coils as shown in Fig. A. The overarching goal is to be able to control systems of various shapes and sizes with relatively minimal recalculation, something that I believe is made significantly more possible by machine learning and ANNs/DNNs.

To lay the goals out clearly, this project aimed to do the following. These steps will be further clarified in Data Sources / Methods Employed section. Firstly, a simulation was conducted to generate a dataset on which to base our agent. A basic physics simulation environment was also set up in python which takes in four PWM (Pulse-width-modulation) values and applies forces to a simulated test rig (shown in Fig. B) based on Ansys sim results which were interpolated using a SciKit-Learn Linear Regression model using polynomial features. Once this groundwork was laid out, a cost function was developed to optimize for successful levitation. Finally, a neural network architecture was designed and implemented to take physical state as input and output PWM controls, then trained to optimize on the cost function.

Data Sources + Technologies Used

The data used in this project was a mix of real-world observations and Ansys Maxwell sparse simulation data, interpolated using scikit-learn's polynomial fitting functionality. Reference observations made in the lab were compared with Ansys calculations to perform a basic sanity check of the viability of this project. The polynomial fit had inputs of: (1) an average gap height axis, (2) a roll degree axis, (3) a left-coil current axis, and (4) a right-coil current axis. These variables correspond to the variable parameters of the Ansys model, allowing a CSV to be populated by Ansys initially with grid points. This CSV was then loaded using pandas and a polynomial was fit using scikit-learn with outputs of force and torque.

The actual training data for the neural network was provided in real-time by the python simulation script (using pyBullet). The reinforcement learning (RL) algorithm consists of two neural networks that share a base network. These two are known as the policy and value networks, and are trained on trajectories, which are populated as the robot (or magnetic levitation pod, in our case), moves throughout the environment for a set number of steps or causes some

boundary condition to trigger, ending the simulation early. There is a reward function attached to the state of the pod at any given point in the simulation, which gets stored as part of the trajectory along with the observed state (sensor gaps and velocities) and the output action of the agent. The value network is trained on the rewards at different states, to best predict the value of taking certain actions from a given state. This is the critic part of the actor-critic network in the proximal policy optimization (PPO) RL method. The actor is trained on the values for a given action given a certain state. This trains the actor to output actions that form a solid policy that generates the best value for any state.

To simplify the scope of this project, time-dependence of control outputs was captured by passing the following input state vector: [Front_mm, Back_mm, Left_mm, Right_mm, *Front_velocity, Back_velocity, Left_velocity, Right_velocity*]^T (velocities capture the necessary scope of time-dependence).

Methods Employed

The focus of this project was experimenting with RL in the hopes of creating an agent that could effectively control a magnetic levitation gap height to sustain a stable hover. To obtain the data, as touched on in the previous section, an Ansys parametric sweep simulation was used, from which the data was exported to CSV, analyzed in python, and fit with a polynomial regression from scikit-learn. This was incorporated into a simulation environment that provided feedback and rewards for the agent to learn from.

To track the agent's performance, two main methods were used. Firstly, the actual gap height was tracked over different trials to see if the actual empirical performance was improving with time. This allowed for a direct evaluation of the model's ability to perform the task at hand. Additionally, the reward over time was tracked to take note of the model's ability to optimize the reward it received over several trials. This helped to monitor the reward structure, since if the reward was increasing steadily towards the maximum theoretical reward (or reaching it) but the gap height error was not trending toward zero, it was obvious there was an issue with the reward structure.

Finally, viewing the pod's behavior through pyBullet's GUI viewer provided a last-resort evaluation solution, where the pod's stability and response to gap height was visible in real-time.

Results

By the end of the time period allotted for this project, the RL agent was unable to find an optimal solution to the control problem. This could be due to a reward shaping issue or due to incorrect architecture of the actor-critic network.

The most common type of issue observed on the first run of the training loop was that the RL agent learned to stick to the track and avoid moving. This allowed it to accumulate relatively low penalties but gain survival bonus rather than falling off and getting a negative reward. To combat

this problem, a penalty for contact with the track was added, proportional to the number of bolts touching the track at a given time. While this helped discourage the pod from sticking to the track, the pod simply switched back and forth between sticking and pushing off too hard (often the issue with traditional control algorithms, as well). Striking the balance between strong repulsion to break free of magnetic attraction and strong attraction afterwards to maintain the gap height before coming quickly to a steady-state gap will be a problem to wrestle with going forward, even with the RL algorithm.

The results from a few trials are shown near the bottom of this report, along with short descriptions of the reward structure used and the pod structure in simulation. Overall, the gap height error plots are visibly non-impressive, with mean gap error never approaching 0.

Next steps to improve RL performance include reward shaping work, neural network architecture tweaking, and exploring RL architectures other than PPO

References

PyBullet for custom RL environments:

https://www.youtube.com/watch?v=kZxPaGdoSJY&t=282s

This video aided greatly in setup and connection of the pyBullet simulation with the RL environment.

Let's Code Proximal Policy Optimization

https://www.youtube.com/watch?v=HR8kQMTO8bk&t=1888s

This video was the source for most of the code in my lev_PPO.ipynb notebook, informing the model architecture and the dataflow.s

Reinforcement Learning Made Simple

https://youtube.com/playlist?list=PL_49VD9KwQ_OML1Knh-Yb7FUFkhTLS0jL&si=cyn4ub9vXDj-abjf

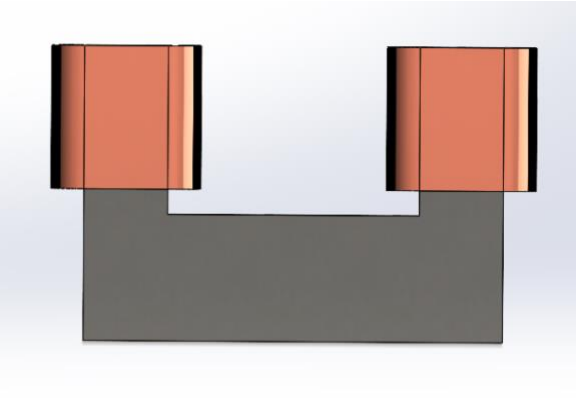
This playlist gave me a decent bit of background and helped me grasp the ideas behind RL. I used the intuition I developed by watching this series to tweak various parameters and conduct reward shaping in my own RL environment.

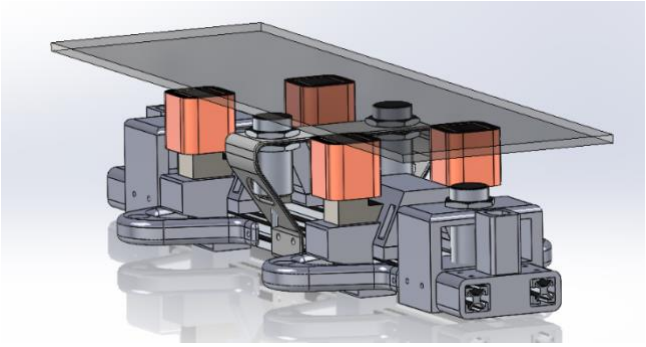
Proximal Policy Optimization Algorithms (Paper)

https://arxiv.org/pdf/1707.06347

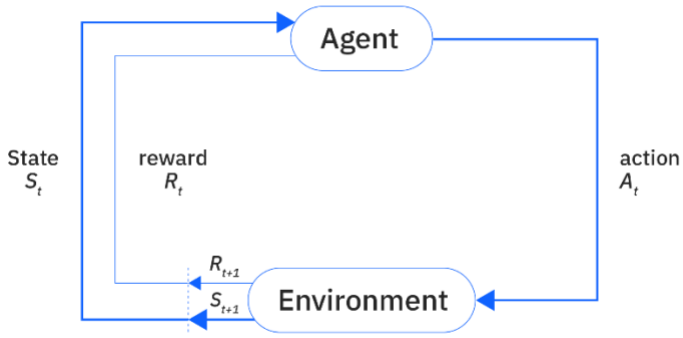
Helped gain a more solid understanding of the math behind the algorithm and develop intuition about parameters.

Fig. A **Fig. B**

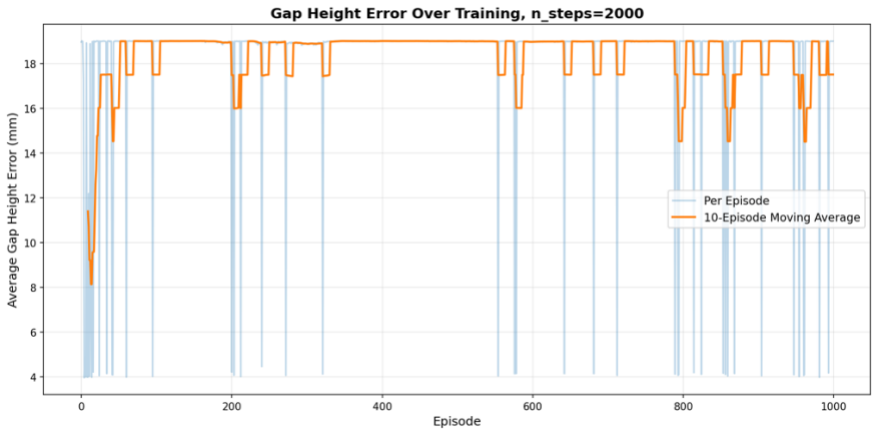




RL Pipeline Visualization

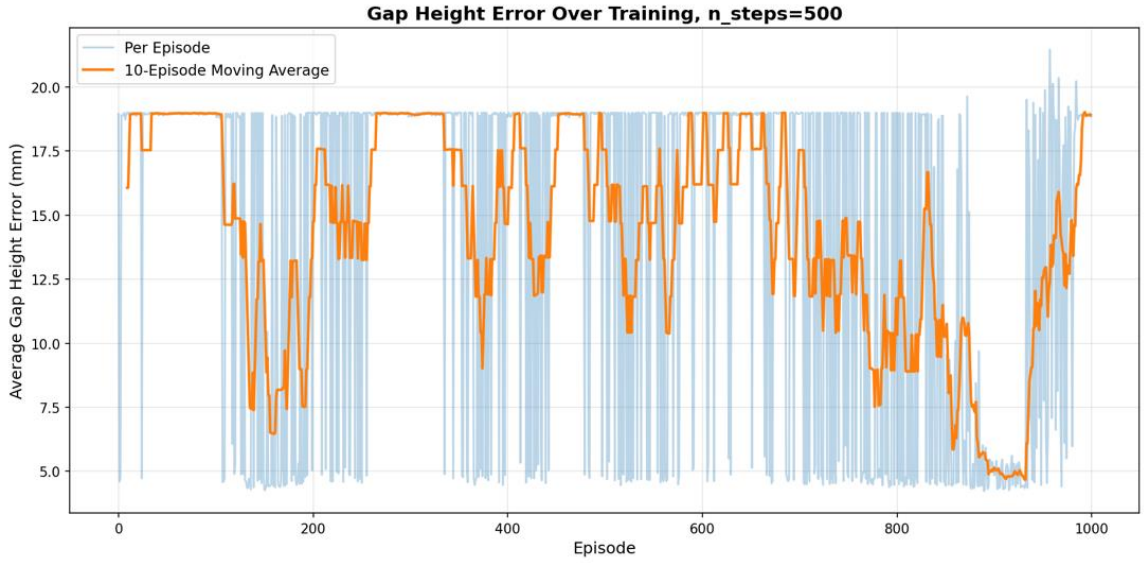


Gap Error Plot #1



This plot was recorded after bottom-stops were added to the pod, meaning that the pod could only drop so far before hitting a mechanical stop. The goal was to give it a guardrail to safely learn to maximize reward, but it instead learned to consistently bang against this bottom boundary to minimize unpredictable error penalties.

Gap Error Plot #2

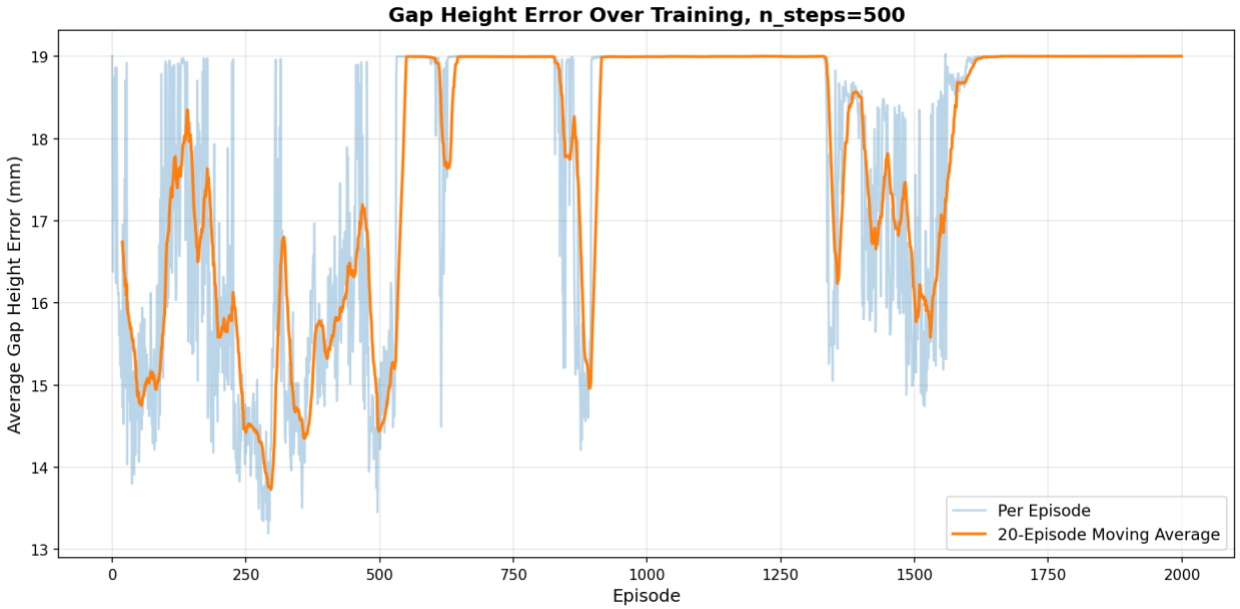


This was recorded from a trial with contact penalties, discouraging the pod from touching the track or bottom-stop in any manner. As is visible, there is more variability in the agent's behavior, but keeping in mind that this is the absolute value of error of the gap height, the agent was still unable to find the optimal gap, instead oscillating seemingly randomly.

Gap Error Plot #3

This was recorded after tweaking the lambda value (how much future rewards weigh into previous actions' evaluations) to be higher (hopefully encouraging planning of actions by the agent) and setting the initial gap to the true equilibrium height. We get much less predictable action, but still outcomes that inevitably oscillate between falling and sticking, no hover.

Gap Error plot #4



This was recorded after removing bolts entirely from the pod structure and trying some tweaked hyperparameters including entropy (exploration factor for RL agent).