Final Project Report: Learning a Terminal Cost for Optimal Control of a Double Pendulum Dynamics

Submitted to: Professor Andrea Del Prete

Student Name
1. Tesfaye Meberate Anteneh (240691)
2. Abdul Hannan Chowdhury (238846)

1 Optimal Control Problem (OCP) Formulation

The double pendulum system is a classic example of a nonlinear dynamical system. The state of the system is described by the angles and angular velocities of the two pendulum arms. The control inputs are the torques applied to the joints. The goal is to find the control inputs u that minimize a cost function while satisfying the system (double pendulum) dynamics and constraints.

2 Objective Function

The objective function J is defined as:

$$J = \sum_{k=0}^{N-1} \left(\mathbf{x}_k^T Q \mathbf{x}_k + \mathbf{u}_k^T R \mathbf{u}_k \right) + V(\mathbf{x}_N) \tag{1}$$

where:

- $\mathbf{x}_k$ is the state vector at time step k.
- $\mathbf{u}_k$ is the control vector at time step k.
- Q is the state weighting matrix.
- R is the control weighting matrix.
- $V(\mathbf{x}_N)$ is the terminal cost.

3 Constraints

3.1 Dynamics Constraints

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \cdot f(\mathbf{x}_k, \mathbf{u}_k) \tag{2}$$

This ensures that the system evolves according to the discretized dynamics of the double pendulum.

3.2 Initial State Constraint

$$\mathbf{x}_0 = \mathbf{x}_{\text{init}} \tag{3}$$

This ensures that the optimization starts from the specified initial state.

3.3 Control Input Constraints

$$\mathbf{u}_{\min} \le \mathbf{u}_k \le \mathbf{u}_{\max} \tag{4}$$

This ensures that the control inputs are within allowable limits.

4 Implementation Steps

4.1 Solve OCP for Different Initial States

- Use CasADi to solve the OCP for a range of initial states x_0.
- Store the initial states and corresponding optimal costs.
- Test with an initial state to ensure that the setup and CasADi dynamics are functioning as expected.

4.2 Train Neural Network

- Generate a dataset of initial states and their corresponding optimal costs by solving the OCP for a variety of initial states.
- Use np.random.uniform to generate random initial conditions x_0 within a specified range.
- For each randomly generated initial condition, attempt to solve the OCP and store the results.
- If the solver encounters a RuntimeError, skip that initial condition.

4.3 Use Neural Network (NN) to Find Value Function

- Incorporate the learned neural network as the terminal cost in a new OCP formulation with a shorter horizon.
- Compare the performance of the shorter horizon OCP with the neural network terminal cost to the longer horizon OCP.

4.4 Parallel Computation

- Use multiprocessing to parallelize the OCP solutions for faster computation.
- The neural network is a feed-forward neural network with the following layers:
  - Input Layer: Takes the initial state x_0 as input.
  - Hidden Layers: Four hidden layers with ReLU activation functions.
  - Output Layer: A single neuron that outputs the predicted optimal cost $J(x_0)$.

5 Results

5.1 Empirical Comparison

- For Initial Condition: $[-0.8, 0, 0, 0]$
  - Optimal cost with full horizon (N=50): 190.91108126560303
  - Optimal cost with reduced horizon (N=25) and terminal cost: 45.433086838816784
- For Initial Condition: $[0.22751964, -0.71200835, 0.67767919, -0.47294348]$
  - Optimal cost with full horizon (N=50): 151.22367829019802
  - Optimal cost with reduced horizon (N=25) and terminal cost: 27.275761504212568
- For Initial Condition: $[-0.53530764, -1.82229739, -0.47224108, -1.79558604]$
  - Optimal cost with full horizon (N=50): 273.3500588299718
  - Optimal cost with reduced horizon (N=25) and terminal cost: 124.78782880989273

5.2 Hyper-parameters

- Adam optimizer
- Learning rate: 1×10^{-3}
- Mini-batch size: 32
- Number of data points: 2000
- Full Control horizon N: 50
- Control input bounds: [-2, 2]
- Sampling time: 0.01 sec
- Training epochs: 500 (early stopping with patience of 10 epochs)

- The model was trained for up to 500 (early stopping with patience of 10 epochs), with the loss decreasing over time. This is expected behavior, showing that the model is learning from the data. The neural network was trained for 500 epochs, taking approximately 0.57 minutes on an 4 core CPU machine.

6 Value Function and Policy Function Plots

The color maps of the value function $V(x)$ and policy function $\pi(x)$ are shown below.

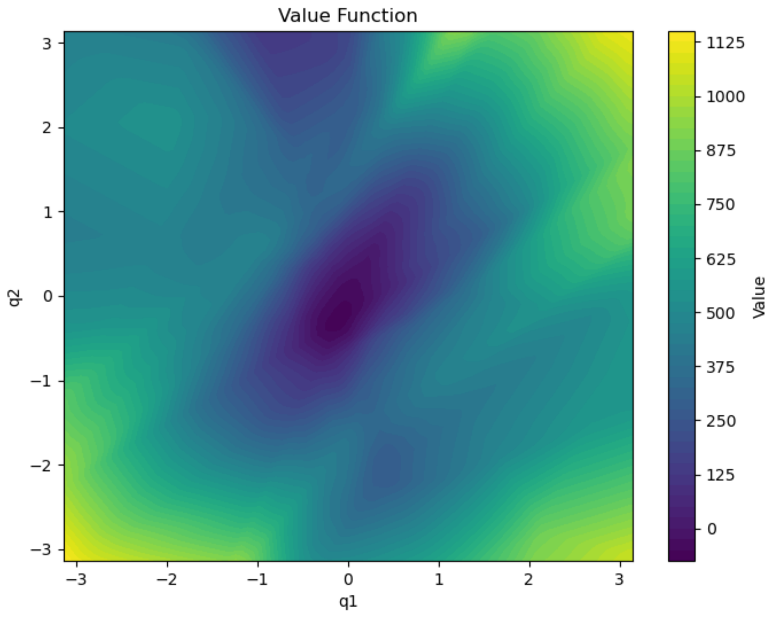

Figure 1: Value Function Plot

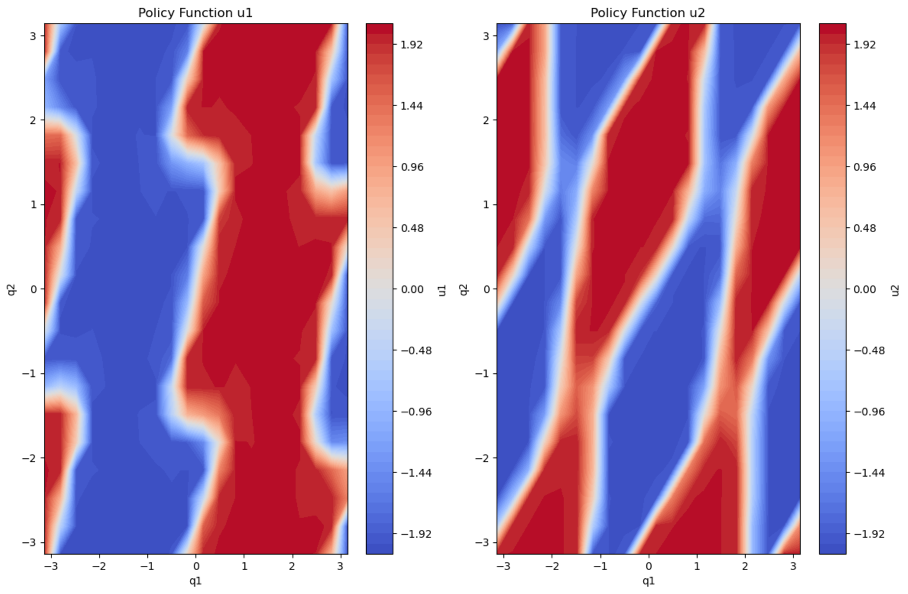

Figure 2: Policy Function Plots for $u1$ and $u2$

7 value function and policy function

- The plots show the control strategy and state dependence of the policy functions.
- The distinct red and blue regions show where positive and negative controls are applied, respectively.
- The central region represents states with the lowest cost-to-go, suggesting these are likely near the desired equilibrium or goal state.
- The policy functions reveal how control inputs $u1$ and $u2$ are applied across different states.
- The trained neural network model significantly reduces the optimal cost when used as a terminal cost in a shorter horizon OCP.

8 Appendix

8.1 Training Process

- The training process for the neural network showed a significant reduction in both training loss and validation loss over the epochs.

- The losses were still relatively high, indicating potential for improvement with additional fine-tuning of the model architecture or hyperparameters.
- Further, test functionality of the code in real-time for the double pendulum.

8.2 Additional Plots

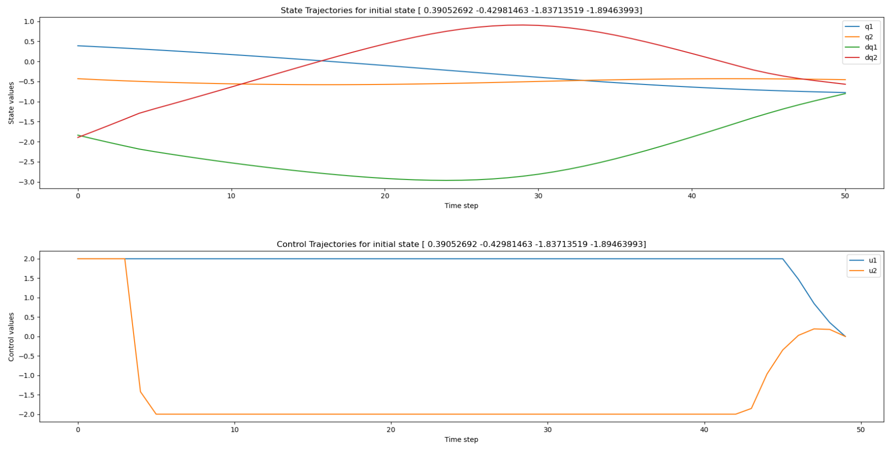

Figure 3: Generated state and control trajectories for a given initial state

based on figure 3 we test our initial model, and trajectories and control inputs are changes over time.

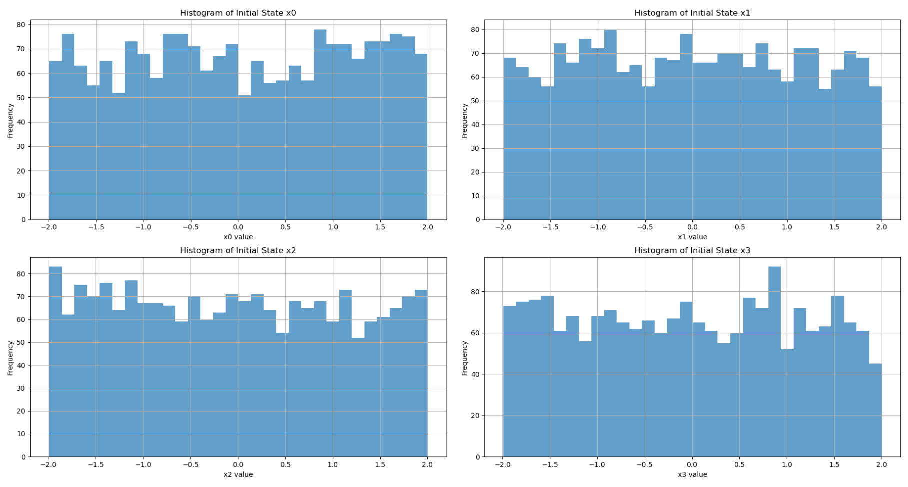

Figure 4: Histogram of Initial States

The histograms in figure 4 for each state variable (x0, x1, x2, x3) show a relatively uniform distribution. This confirms that the initial states are sampled uniformly within the range of [2,2].

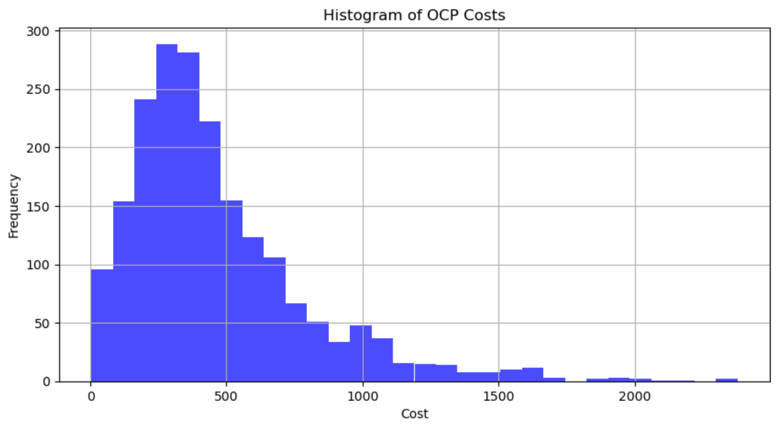

Figure 5: Histogram of OCP Costs

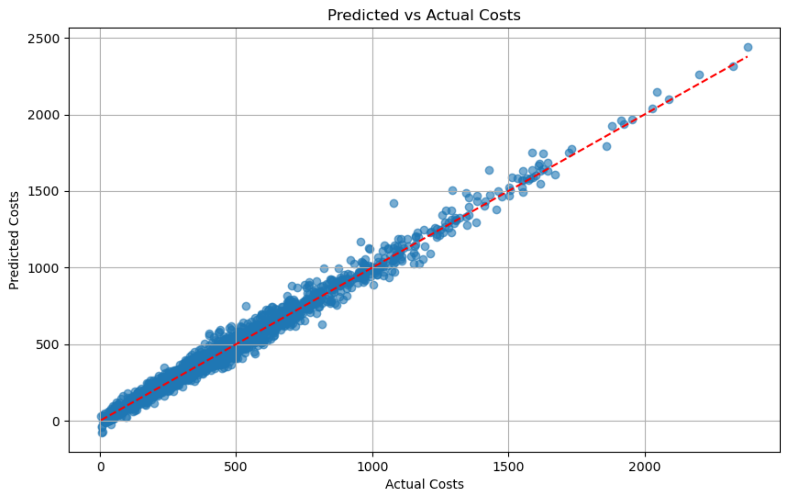

Figure 6: Scatter Plot of Predicted vs. Actual Costs

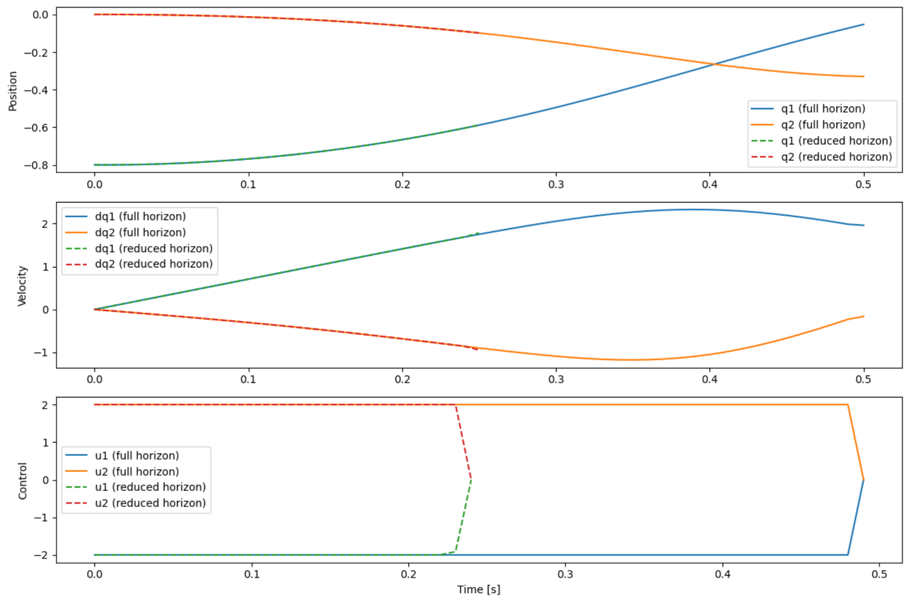

Figure 7: Initial Condition: [-0.8 0. 0. 0.] trajectories and control inputs

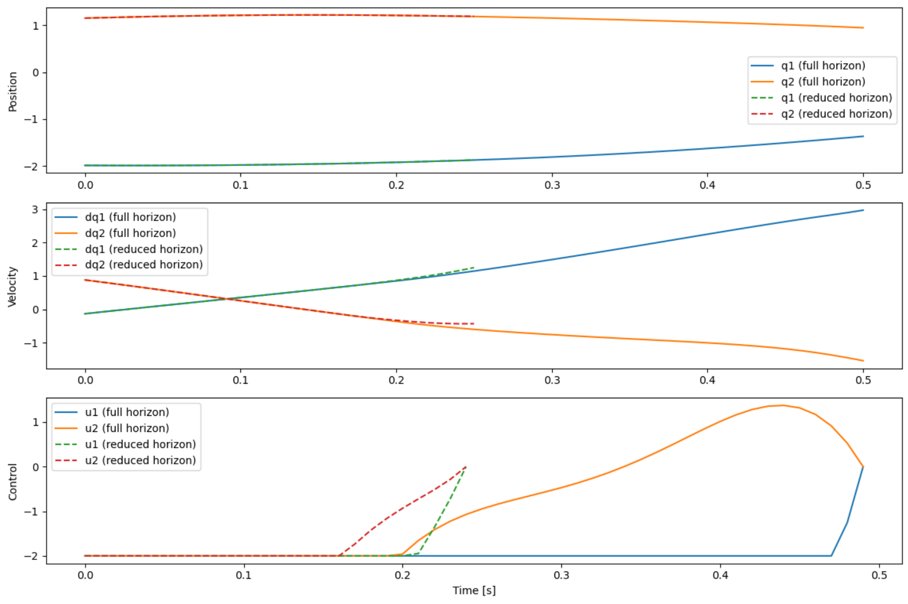

Figure 8: Initial Condition: [-1.98562809 1.15334051 -0.13378199 0.88065746] trajectories and control inputs

order of the code.

double_pendulum_casadi.py$\longrightarrow$ *ocp_double_pendulum_casadi.py*$->$ *train_value_network.py*-$->$ *ocp_with_terminal_cost.py*.