Advanced Robot Control Project Report

La Scala Giovanni Maria Francesco 241561
Di Lorenzo Andrea 239221

1 Introduction

Our final project, Project A, focuses on learning a value function $V(x)$ that can then be used as a terminal cost in an Model Predictive Control (`MPC`) formulation with shorter horizon time to show that the introduction of said cost compensates the decrease of the horizon length.

In this section we exploit the steps followed during the project:

1. Formulation the problem using CasaDi (a software library used for solving OCPs);
2. Training the Neural Network;
3. Formulation of the MPC;

The work was done at first on a single pendulum and then, once the algorithm developed has been validated, we moved the problem to a double pendulum.

2 Formulation of the problem using CasaDi

CasaDi is software framework for dynamic optimization and control in the field of computational mathematics. It provides a user-friendly environment for modeling and solving complex optimization problems, particularly within the context of optimal control and dynamic optimization applications.

As said, at first, we want to apply this a single pendulum whose dynamics is:

$$f(x,u) = \begin{cases} \dot{q} = v \\ \dot{v} = g\sin q + u \end{cases} \tag{1}$$

The problem we need to implement in our code is:

$$\begin{aligned} \min_{u[i]} \; & J = \sum_{i=0}^{N} \left((x[i] - x_d)^T W_x (x[i] - x_d) + w_u u[i]^2 \right) \\ \text{s.t.:} \; & x_{k+1} = f(x_k, u_k) \quad \text{for } k = 0, \dots, N-1 \\ & x_0 = x_{\text{init}} \end{aligned} \tag{2}$$

For the sake of this project, we'll be dealing with unconstrained `OCP`s, except for the dynamic constraint and the initial state constraint.

[1] We have chosen a quadratic cost to make the problem convex.

From the formulation in (2) we want to create a dataset containing the initial state x_0 and the corresponding optimal cost $J(x_0)$. For simplicity of calculations we are considering an `OCP` without the path bounds. For the sake of this project we have set $q_{\text{target}} = 5/4\pi$. The relative cost map is shown in Figure 1.

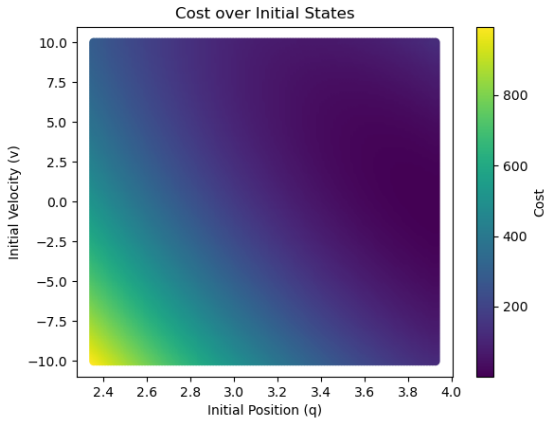

Figure 1: Cost map

As we may see, for the given problem, we pay less cost as the system starts closer to the target.

Since the system is relatively simple we can also show the cost function associated as described in (2) which is a quadratic cost[1].

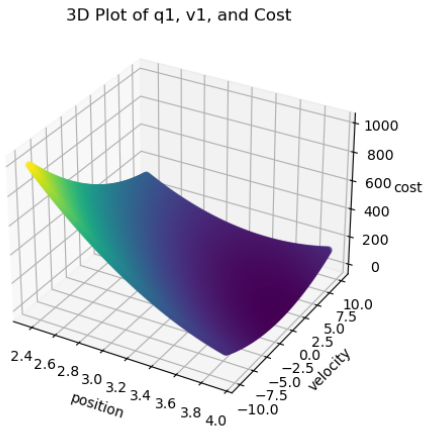

Figure 2: Cost Functional

3 Neural network

Now, we train a neural network to learn the optimal cost associated with the initial state.

In the file `nn_SP.py`, we process the dataset, which is loaded from a `.csv` file containing the collected states alongside the corresponding initial cost $J(x_0)$. After splitting the data into training (80% of the dataset) and testing sets (20% of the dataset), and normalizing the features using `StandardScaler`, we proceed with building the neural network model.

The model consists of an input layer with dimensions corresponding to the feature space, followed by three hidden layers containing 64, 32, and 16 neurons, respectively, each using the ***tanh*** activation function. The final output layer is a fully connected (**Dense**) layer with a single neuron, as the output is expected to be a scalar (the cost value).

This architecture is commonly used for regression tasks, which is the goal in this case.

The script trains the neural network over 600 epochs using the **Adam optimizer** which is widely used for stochastic optimization (requires minimal memory and computationally efficient) and **mean squared error** as the loss function.

The learning rate α is a hyper parameter that controls the step size for adjusting model weights after each iteration during training. A lower learning rate leads to smaller, more precise updates, reducing the risk of overshooting the optimal solution but slowing convergence. A higher learning rate accelerates training but can cause instability. In this case, $\alpha = 0.0005$ was chosen to balance precision and training speed.

A summary of the model is reported in `Table 1`.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 2)	0
dense (Dense)	(None, 64)	192
dense_1 (Dense)	(None, 32)	2080
dense_2 (Dense)	(None, 16)	528
dense_3 (Dense)	(None, 1)	17

Table 1: Model Summary

The performance of the trained neural network and the resulting model quality is demonstrated by the plot shown in `Figure 3` and the corresponding metrics in Table 2.

The plot illustrates a near-perfect linear relationship between the true and predicted values, indicating that the model has learned to predict the target variable with high accuracy. The line of points closely follows the $y = x$ line, suggesting minimal error in the predictions.

Figure 3: True Values vs. Predictions

	Training Set	Test Set
`MSE`	0.61634	0.40148
`RMSE`	0.78507	0.63363

Table 2: Metrics summary for trained nn

4 `MPC` formulation

As previously stated, the goal is to implement a Model Predictive Control algorithm to control the single pendulum (1).

At a theoretical level, MPC consists in solving an infinite horizon OCP using the current state as initial state. Though it is not possible to compute an infinite horizon OCP, there are some challenges we need to face.

The aim of this project is in fact to show that with the addition of a **terminal cost** we can reduce the `OCP` time horizon recovering the performance of the controller obtained via full `MPC`.

The problem we want to solve now looks like this:

$$\begin{aligned} \min_{u[i]} J = &\sum_{i=0}^{N-1} \left((x[i] - x_d)^T W_x (x[i] - x_d) + w_u u[i]^2 \right) \\ &+ V_F\left(x[N]\right) \\ \text{s.t.:} \quad & x_{k+1} = f(x_k, u_k) \quad \text{for } k = 0, \dots, N-1 \\ & x_0 = x_{\text{init}} \end{aligned} \tag{3}$$

Again, we are dealing with a non-constrained `MPC` problem; the only constraints considered are the dynamic constraint, where $f(x_k, u_k)$ is the dynamics of the single pendulum reported in (1), and the initial state constraint.

To the sum of the stage cost we add the term $V_F(x_N)$ which represents the **terminal cost** learned from the neural network explained before.

4.1 Reduction of the time horizon

Adding a terminal cost in `MPC` formulation in (3) has significant implications for the optimization process, particularly in terms of reducing the required time horizon for effective control.

Incorporating a terminal cost in MPC reduces the required prediction horizon by focusing on long-term performance objectives through the final state. This allows for shorter horizons, easing computational demands and improving real-time responsiveness, while maintaining system performance and stability.

In the table below we report the fundamental hyper parameters used during the implementation of the project.

	With `TC`	**Without `TC`**
Time Horizon T	0.01	1.0
`OCP` time steps dt	0.01	0.01
Horizon steps N	1	100
Target position q_{des}	$5/4\,\pi$	$5/4\,\pi$
Position weight w_q	10^2	10^2
Velocity weight w_v	10^{-1}	10^{-1}
Input weight w_u	10^{-4}	10^{-4}

Table 3: Parameters

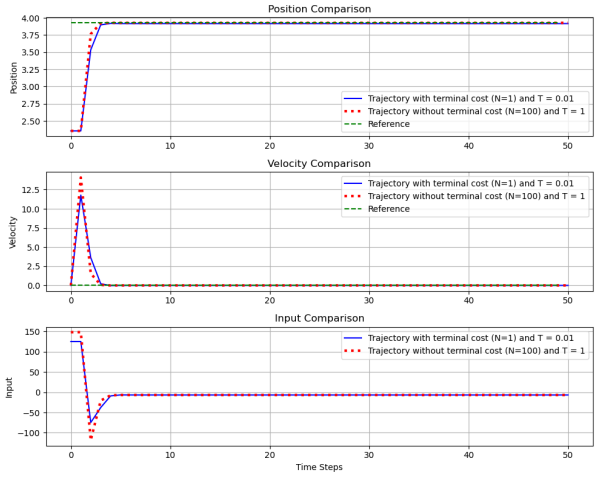

Figure 4: State Trajectories compared

Figure 4 shows the trajectory computed for both problem formulation[2] (with and without the terminal cost). As we may see, the introduction of the terminal cost compensates for the reduction of the time horizon and the number of horizon steps.

Further analysis regards the error e between the predicted terminal cost $\hat{V}_1(x_1)$ by the neural network and the true terminal cost $V_1(x_1)$ associated to the terminal state x_N because it directly reflects the accuracy of the neural network in approximating the optimal cost-to-go function.

[2]For this plot an initial state $[3/4\pi, 0]$ was chosen.

Minimizing this error is crucial for ensuring the reliability and performance of the control system in predicting future states and optimizing control actions.

$$e = \hat{V}_1(x_1) - V_1(x_1) \tag{4}$$

To better exploit the characteristics and behaviors of the error function above, we simulate the system over $N_{\text{sim}} = 50$ steps, starting from $N_{\text{init}} = 15$ initial states sampled randomly.

Figure 5 illustrates the system's state trajectory from different initial states, comparing the different `MPC` formulations (solid line `MPC` with terminal cost, dashed line without).

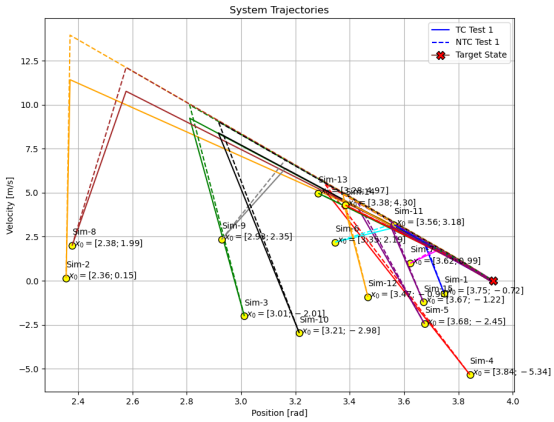

Figure 5: State trajectories from an initial state x_0

In the following we report the behavior of the error function $e(\cdot)$ across the conducted tests. For better visualization we only report 10 time steps out of the 50 computed during the simulation.

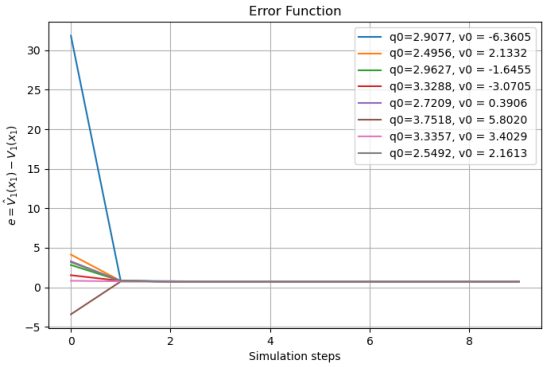

Figure 6: Error function across tests

Starting from Figure 6 we've conducted an analysis that aims to estimate the quality of the model, thus investigating mean value $\hat{e}(\cdot)$, standard deviation $\sigma_{\hat{e}(\cdot)}$ and median $Me_{\hat{e}(\cdot)}$ for all tests.

Figure 7: Error statistics

	Mean	Median	StD
Average (across tests)	0.8114	0.7360	0.7522

Table 4: Summary of statistics

The error analysis across the tests reveals consistent performance of the model, with most tests showing low mean errors and small variability.

The majority of the mean error values are close to zero, indicating that the model is generally unbiased, as confirmed by the red dashed line representing the median of the mean errors. However, test 1 and test 10 stand out due to their significantly larger standard deviations, suggesting higher variability under given initial conditions.

Overall, the model demonstrates reliable accuracy with minimal bias, but some test scenarios exhibit increased uncertainty.

The cost-to-go estimated $\hat{V}_1(x_1)$ by the neural network associated to the state trajectories in Figure 5 is reported in Figure 8 which also shows the "true" value of the terminal cost associated to the terminal state x_N and the total cost trend for the same initial conditions tested for the full `MPC`.

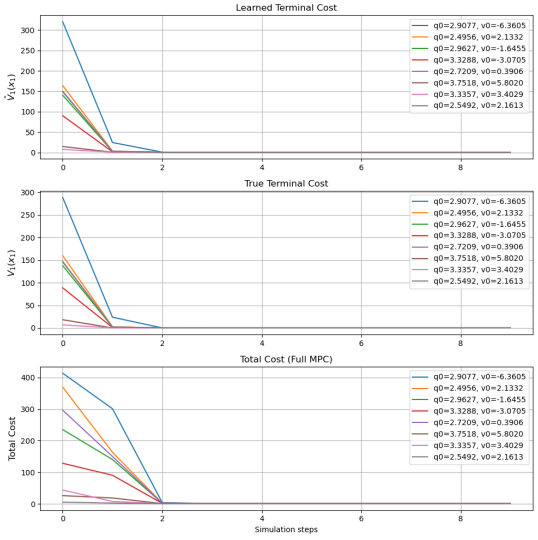

Figure 8: Costs behavior

4.2 Robustness with respect to noise

In this study, we extend the previous `MPC` formulation (3) by introducing controlled noise into the system, $\mathcal{N}(0,\ 0.1)$, to assess its robustness and performance under uncertainty.

We will only analyze the error function $e(\cdot)$ as previously discussed.

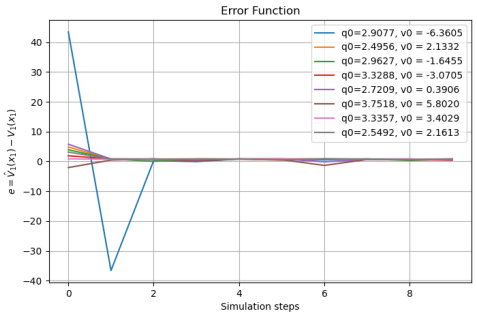

Figure 9: Error across tests with $\epsilon = \mathcal{N}(0,\ 0.1)$

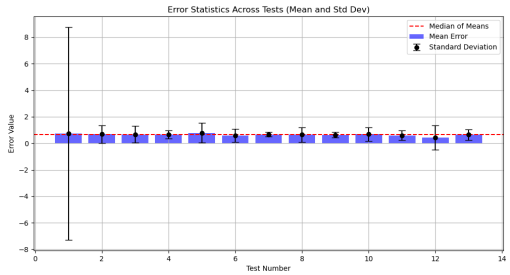

Figure 10: Error Statistics with $\epsilon = \mathcal{N}(0,\ 0.1)$

	Mean	Median	StD
Average (across tests)	0.6576	0.7403	0.9268

Table 5: Summary of statistics with $\epsilon = \mathcal{N}(0,\ 0.1)$

The relatively high standard deviations, particularly in `Test 1` ($\sigma_e = 8.0284$), highlight significant variability in performance, indicating that the system may be sensitive to certain conditions or disturbances, but overall it works as expected.

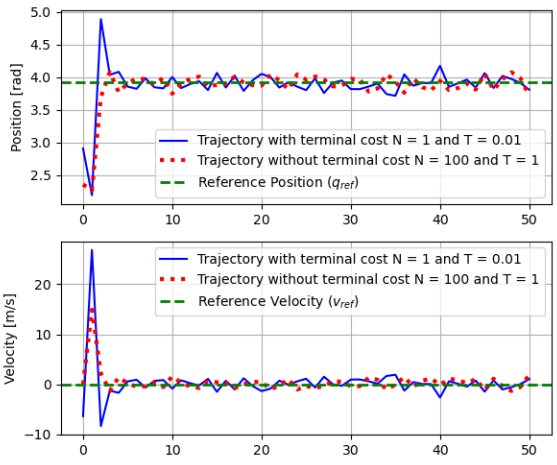

Figure 11: State Trajectories with $\epsilon = \mathcal{N}(0,\ 0.1)$

5 Problem formulation

After having examined the properties and the implications when formulating a Model Predictive Control with terminal cost, we try to exploit the behaviour of a **double pendulum** with the same formulation.

The implementation steps we took are basically the same. We just need to modify the codes accordingly to the new dynamics and based on the fact that the state vector we are interested in analyzing is now a 4-Dimensional states ($x \in \mathbb{R}^4$) while the control input vector is 2-Dimensional ($u \in \mathbb{R}^2$):

$$x = \begin{bmatrix} q_1 & v_1 & q_2 & v_2 \end{bmatrix} \qquad u = \begin{bmatrix} u_1 & u_2 \end{bmatrix}$$

We start again by generating the dataset by solving a series of OCPs with initial states sampled from a grid (or randomly) and saving, for a viable configuration, the initial state and the corresponding optima cost $J(x_0)$ using the double pendulum dynamics $f(x, u)$ reported in the file `DP_dynamics`.

The Optimal Control Problems to solve have the following formulation:

$$\begin{aligned} \min_{u[i]} \; & J = \sum_{i=0}^{N-1} \left((x[i] - x_d)^T W_x (x[i] - x_d) + u[i]^T W_u u[i] \right). \\ \text{s.t.:} \; & x[i+1] = f\left(x[i], u[i]\right) \\ & x[0] = x_{\text{init}} \end{aligned} \tag{5}$$

Where W_x and W_u are diagonal matrices representing the weights used for the states of the system and for the control inputs.

$$W_x = \begin{bmatrix} w_{q1} & 0 & 0 & 0 \\ 0 & w_{v1} & 0 & 0 \\ 0 & 0 & w_{q2} & 0 \\ 0 & 0 & 0 & w_{v2} \end{bmatrix} \qquad W_u = \begin{bmatrix} w_{u1} & 0 \\ 0 & w_{u2} \end{bmatrix}$$

In this case, we are also excluding any inequality constraints (path constraints) to maintain simplicity. As for the Single Pendulum here we have set $q_{1d} = \pi$ and $q_{2d} = \pi$.

6 Neural Network

For what concerns the neural network, the reasonment at the basis is the same, we just need to put the input layer dimension to 4.

Again we built the neural network with three hidden layers while the output dimension is 1 since we want to solve a linear regression problem of the cost with respect to the states stored in the DataFrame file.

In the following we evaluate the quality of the model obtained via the following:

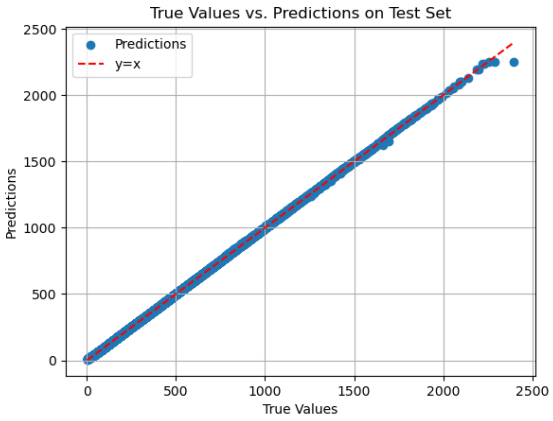

Figure 12: Linear Regression

	Training Set	Test Set
`MSE`	3.5097	2.9796
`RMSE`	1.8734	1.7262

Table 6: Metrics summary for trained nn

For the formulation of the optimal control problems we have discarded the path constraints on the system the limits on the control action.

The neural network was then trained with 300 epochs using 80% of the dataset for the training and 20% for testing which gave the plot seen in `Figure 12`.

We are using the same structure of the neural network described in Table 1.

7 MPC formulation

As said previously, we are interested in exploiting the behaviour of the system when a terminal cost is introduced in the formulation of the MPC as in (6).

The complete formulation can be expressed by:

$$\begin{aligned} \min_{u[i]} J = & \sum_{i=0}^{N-1} \left((x[i] - x_d)^T W_x (x[i] - x_d) + u[i]^T W_u u[i] \right) \\ & + V_F(x_N) \\ \text{s.t.: } & x[i+1] = f\left(x[i], u[i]\right) \\ & x_0 = x_{\text{init}} \end{aligned} \tag{6}$$

To simplify the problem, we assigned identical weights to the position, velocity, and control actions for both links. To achieve faster convergence, we applied a heavier penalty on the positions compared to the velocity and control inputs.

Furthermore, the introduction of the terminal cost aims to enhance the controller's performance in comparison to a simulation with $T = 1$, which excludes $V_F(x_N)$ from the formulation.

Once again, we are addressing a non-constrained MPC problem; the only constraints considered are the dynamic constraints, represented by $f(x_k, u_k)$, which govern the dynamics of the double pendulum, along with the initial state constraint.

To the cumulative stage cost, we add the term $l_F(x_N)$, which signifies the **terminal cost** derived from the neural network discussed earlier.

Incorporating a terminal cost in MPC helps reduce the necessary prediction horizon by concentrating on long-term performance objectives via the final state. This approach allows for shorter horizons, enhancing real-time responsiveness, all while maintaining system performance and stability.

Table 7 below lists the significant hyper-parameters utilized in the simulation.

	With TC	**Without TC**
Time Horizon T	0.05	1.0
OCP time steps dt	0.01	0.01
Horizon steps N	5	100
Target link 1 $q_{1_{\text{des}}}$	π	π
Target link 2 $q_{2_{\text{des}}}$	π	π
Position weight w_q	10^2	10^2
Velocity weight w_v	10^{-1}	10^{-1}
Input weight w_u	10^{-4}	10^{-4}

Table 7: Parameters

Figure 13 shows the trajectory computed for both problem formulations.

As we may see, the introduction of the terminal cost compensates for the reduction of the time horizon and the number of horizon steps, with a random initial state: [3.7255, 1.1769, 2.4238, 0.6009].

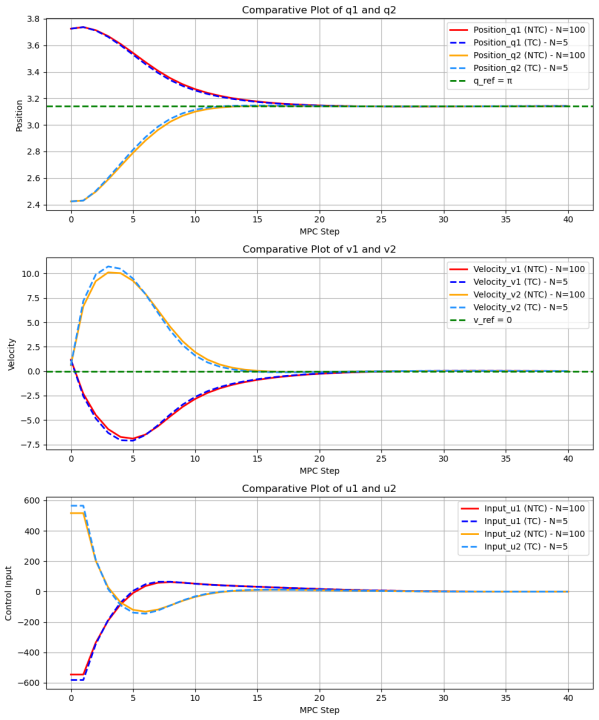

Figure 13: Trajectory compared

In analyzing the double pendulum, we again consider the error e between the predicted terminal cost $\hat{V}_1(x_1)$ and the true terminal cost $V_1(x_1)$.

To effectively analyze the characteristics and behaviors of the error function, we simulate the system over $N_{sim} = 40$ steps, beginning with $N_{init} = 15$ randomly sampled initial states. This randomness in the initial states demonstrates the robustness of the algorithm across different scenarios.

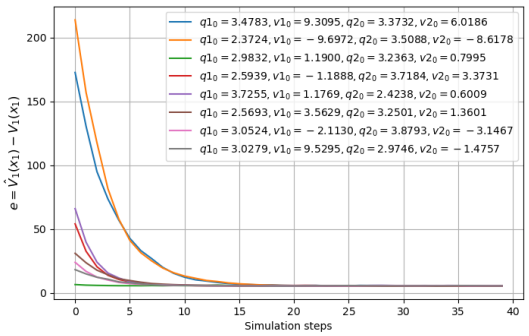

Figure 14: Error over test

In Figure 14 and 15 we present an analysis of the error function in (4), detailing its statistical properties for each test conducted summarized in Table 8.

	Mean	**Median**	**StD**
Average (across tests)	11.0033	5.5264	15.6081

Table 8: Summary of statistics

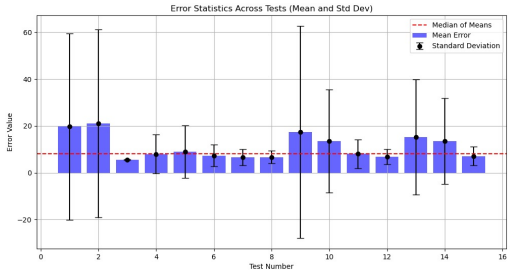

Figure 15: Error statistics

The graph shows the error statistics between the predicted and actual terminal costs across 15 different tests.

The mean errors are close to zero for most tests, indicating that the model's predictions are generally unbiased. The median error (highlighted by the red dashed line) reinforces this, with most values clustering near 5, suggesting consistent, low errors.

However, tests 1, 2, 9, 10, 13, and 14 exhibit much larger standard deviations, showing greater variability and uncertainty in these predictions. This indicates that under specific initial conditions in these tests, the model struggles to predict terminal costs with high accuracy.

The cost-to-go estimated $\hat{V}_1(x_1)$ by the neural network associated to the state trajectories tested is reported in Figure 16 which also shows the "true" value of the terminal cost associated to the terminal state x_N and the total cost trend for the same initial conditions testes.

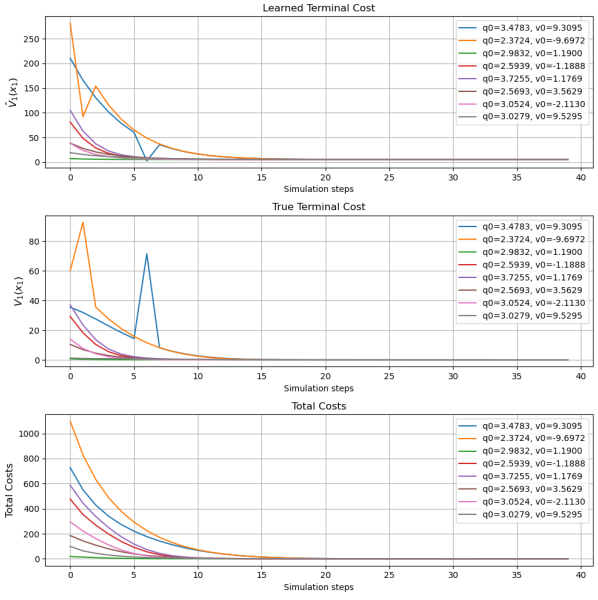

Figure 16: Costs behavior

For better visualization of the graphs we've only reported 8 tests, while the statistical results come from an analysis on the complete dataset.

7.1 Robustness with respect to noise

In this study, we extend the previous `MPC` (6) framework by introducing controlled noise, modeled as $\mathcal{N}(0,\ 0.01)$.

Figure 17 shows the error $e = \hat{V}_1(x_1) - V_1(x_1)$ over multiple simulation steps, instead 18 provides a statistical summary of error across tests.

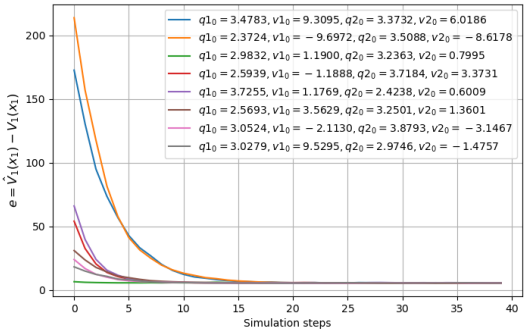

Figure 17: Error over tests with $\epsilon = \mathcal{N}(0,\ 0.01)$

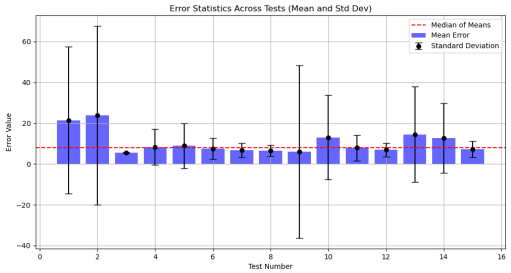

Figure 18: Error statistics with $\epsilon = \mathcal{N}(0,\ 0.01)$

	Mean	Median	StD
Average (across tests)	10.4766	5.6381	15.2164

Table 9: Summary of statistics with $\epsilon = \mathcal{N}(0,\ 0.01)$

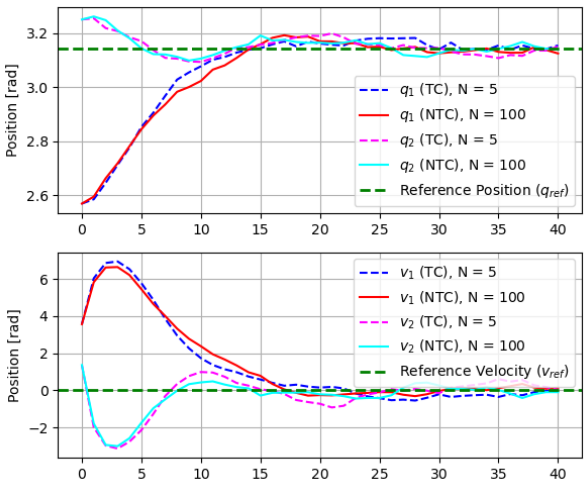

Figure 19: State Trajectories with $\epsilon = \mathcal{N}(0,\ 0.01)$

8 Conclusions

The conclusions from this project indicate that the inclusion of a neural network-trained terminal cost in Model Predictive Control formulations enables a significant reduction in prediction horizon length while maintaining control performance.

This adjustment lowers computational demands (as shown in Figure 20) and achieves improved responsiveness, as demonstrated in both single and double pendulum scenarios.

Error analysis confirmed that the terminal cost function approximation is generally reliable, though some scenarios exhibited higher error variability, suggesting areas for further tuning.

The noise robustness tests showed that while the system performs well under nominal conditions, it is sensitive to larger disturbances.

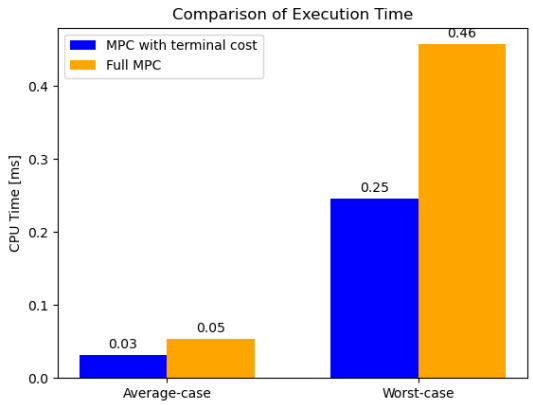

Figure 20: Computational Times

The bar chart above presents a comparison of execution times for the two different Model Predictive Control formulations.

The results indicate that in both the average-case and worst-case scenarios, the `MPC` with terminal cost formulation, achieves faster computation times, highlighting its efficiency.

9 Possible Improvements

Improvements related to this project focus on:

- **Double Pendulum neural network**: as reported in Table 6, the errors both on Training and Test sets are quite high, therefore we should more on reducing more the error (i.e. closer to 0). The challenge here was to find a suitable series of learning rates α. The main adjustment should be on the number of epochs used in training phase.

- **Constraints**: in the current implementation, constraints on states and control actions were omitted to simplify the problem formulation.

  While this allowed for a streamlined model, it does not fully capture the limitations that would be present in real-world applications, such as actuator saturation, safety boundaries, or physical constraints of the system.

  Introducing constraints into the `MPC` framework would enhance the controller's realism and robustness, particularly in complex environments where such limitations are critical for safe and efficient operation.

- Lastly, at a more theoretical level, we could demonstrate that the learned terminal cost is a **Lyapunov Function**, hence it must be a decreasing, continuous, and positive definite function. This would help us proving the asymptotic stability of the controller.

  For a function $V(\cdot) : \mathbb{R}^n \to \mathbb{R}$ to be *Lyapunov*, for a given **Positive Invariant Set** $\mathcal{W}$, it should satisfy the following conditions:

$$\begin{aligned} V(x) &\geq \alpha_1 |x| \\ V(x) &\leq \alpha_2 |x| \\ V(f(x)) - V(x) &\leq -\alpha_3 |x| \end{aligned} \tag{7}$$

  If there exists α_1, α_2, α_3 greater than 0, $V(\cdot)$ is a Lyapunov function. Further, if said Lyapunov function exists, the origin is **exponentially stable** in $\mathcal{W}$.