Assignment 3.B

Advanced optimization-based robot control

Daniele Schlagenauf
ID number: 236548

Sara Endrizzi
ID number: 241011

August 2024

1 Introduction

In this assignment is asked to learn the Viability kernel of a double pendulum and use it as terminal constraint in an Model Predictive Control (MPC) framework to ensure recursive feasibility. For simplicity the initial approach is made by considering a single pendulum.

As a first step it is implemented an Optimal Control Problem (OCP), develop with ***CasADi*** library[1], in order to create a dataset that will be used to train a neural network able to classify the viability of a state. This network is then implemented as the terminal constraint of the MPC in order to ensure that last state of each OCP falls into the invariant set $\mathcal{X}_f$.

2 Single Pendulum

The dynamics of the pendulum is obtained by exploit the equation of motion 1, considering unitary both mass and length. From now on the state x is define by position q and velocity $\dot{q}$ and the control input torque u is τ.

$$m\,l^2\,\ddot{q} = -m\,g\,l\,\sin q + \tau \tag{1}$$

2.1 Optimal Control Problem

Resuming the conclusion of the *Assignment 2* the sampling strategy applied to collect the data is the grid, whose allow a better vision of the viable kernel inside the state space.

The discrete OCP, shown in Equation 2, has a time horizon of 0.5 seconds and its cost function is defined as a weighted least square function whose try to penalizing large speed variations, oscillation and high torque with same importance, hence same weights (10^{-3}). For what concern the constraints, it is impose to follow the system dynamics, to remains in the user define bounds for both state and control and to start from a given initial point.

[1]https://web.casadi.org/

$$\begin{aligned}
\text{Minimize:} \quad & \sum_{k=0}^{N-1} w_v \dot{q}_k^2 + w_u u_k^2 \\
\text{Subject to:} \quad & x_{k+1} = f(x_k, u_k) \quad k = 0, ..., N-1 \\
& u_k \in \mathcal{U} \quad k = 0, ..., N-1 \\
& x_k \in \mathcal{X} \quad k = 1, ..., N \\
& x_0 = x_{initial}
\end{aligned} \tag{2}$$

where $f(x_k, u_k)$ is the dynamics while the sets $\mathcal{X}$ and $\mathcal{U}$ are defined respectively $[q_{min} \quad q_{max}] \times [\dot{q}_{min} \quad \dot{q}_{max}]$ and $[u_{min} \quad u_{max}]$.

The dataset created is composed by the initial state and its viability outcome which is 1 if viable and 0 if not. Remembering that a state is viable if, starting from it, the system always remain inside the feasible set without violating the constraints.

To compute a grid of 10000 points it is required around 45 minutes, thus to speed up the computation a multi-process approach has been implemented leading the time to 15 minutes. The computed viability kernel is reported in Figure 1.

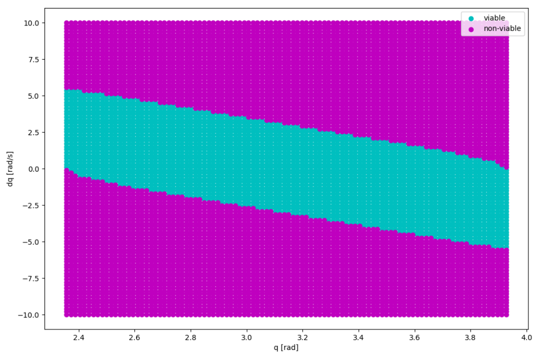

Figure 1: OCP viable set with 100x100 points

2.2 Neural Network

After some research on how to build a neural network[2] it has been decided to use the TensorFlow library[3].

Since the model perform a binary classification, the loss function chosen is the **binary cross-entropy**, which quantifies the difference between the predicted output and the actual target. For the same reason, the activation function chosen for the output layer is the **sigmoid** while, for the hidden layers, following the suggestion, it has been used the **ReLu** function. Similar results can be obtained by implement ReLU in all layers.

The neural network created, after some tries changing the number of layers and neurons, is characterized by a decreasing number of neurons per layers and it is composed by:

- Input layer: that takes the dimensions of the feature's input (2);
- Hidden layers: three layers with 64, 32 and 16 neurons each;
- Output layer: returns a value between 0 and 1.

The dataset collected from OCP problems is scaled to ensure the same magnitude for both the input features, which lead to a faster convergence. It has been implemented by means of *StandardScaler* which remove the mean and divide by the standard deviation each state.

The model created is compiled using **ADAM** optimizer algorithm which is robust to the hyper parameters and is able to dynamically adapts the learning rates base on the magnitude of the gradient. Given the simplicity of the system, the ADAM learning rate is left by default to 0.001. Another possibility could be **Nesterov-ADAM** which present better performance but increase the computational complexity.

For the train of the model it is used an 80/20 division of the dataset, with 80% allocated to the training and 20% for the testing. Two important parameters of the training are the number of epochs and the batch size, which represent the number of times that the entire dataset will be iterate and how many samples are taken in each iteration. For this model the number of epochs is set to 300 and the batch size is left the default 32.

The number of epochs can be set high thanks to the implementation of the early stopping callback, this option allow to stop the training when the performance, obtained from the validation data, stops improving for a number of epochs defined by the *patience* parameter, sets at 10, avoiding the over fitting and minimizing the time.

An important parameter to evaluate the performance of a neural network is the accuracy, or dually the loss, which represents the error between the model's predictions and the actual values of the training dataset. The model created has in average an accuracy of 0.99 and a loss value of 0.01.

A graphical way to visualize how the network behave is by means of the *confusion matrix*, shown in Figure 2, in which the False positive and False negative are very few respect the total amount of states.

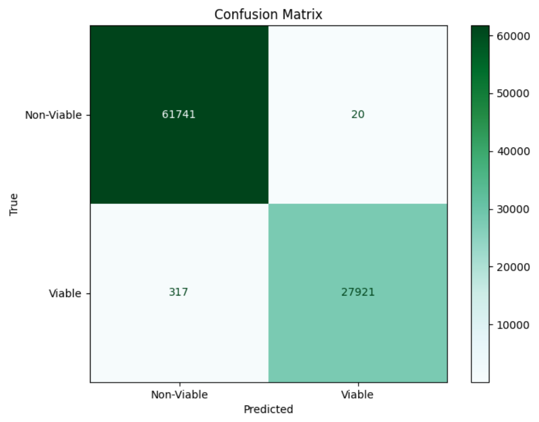

Figure 2: Confusion matrix with 90000 test data

Given the high value of accuracy it is necessary to verify if is due to an over-fitting of the Neural Network which lead the model to works good only with the train data but fails with new ones. This can be seen by inspecting its pattern over the epochs where the training accuracy will continuously increase while the validation one will reach a plateau. In the model created this behaviour does not happen, as shown in Figure 3, and thus the network is well fitted and the high accuracy is due to the specificity of the problem.

The plot can be used also to tune some hyper parameters such as the learning rate which effects the speed of convergence (the slope), the batch size which effect the smoothness of the curve and the memory required. Moreover, the choice of the optimizer influence the convergence properties. The used values produce good results.

Compare to the 15 minutes required to the OCP, the Neural Network takes only around 20 seconds to classify the viability of all points and the result with 10000 points is the same as the one computed with the OCP reported in Figure 1.

[2]How to design a neural networks
[3]TensorFlow website

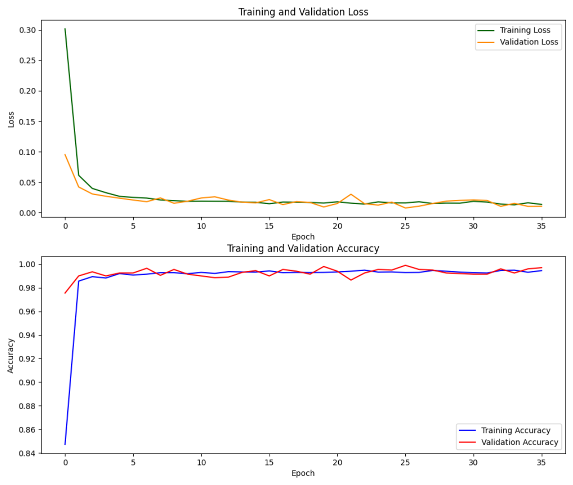

Figure 3: Loss and accuracy

2.3 Model Predictive Control

The MPC is an algorithm that solve finite-horizon OCPs in order to predict the future behavior of the system. The control problems are solved iteratively, in particular the first prediction of the overall trajectory is computed starting from the initial conditions but only the first step is actually executed and the reached state is then taken as initial condition for a new OCP. The rest of the solution is used to create a guess (vector) for the next iteration in order to speed up the calculation of the new trajectory which must be provided in time.

One of the main problem is to ensure the recursive feasibility of the problem. This would not be an issue if the OCPs have an infinite time horizon, which lead each problem to see the same time window, hence for the Bellman's optimally principle each computed trajectory will be a sub-arc of the optimal one. In order to mimic this behaviour a terminal constrain is added to the problem formulation by imposing that the last step must be in the control invariant set ($\mathcal{X}_f$).

In particular in the assignment the latter is manage by giving the last state as input of the Neural Network and forcing the result to be close to one which ensure the viability. Closer to one farther the actual trajectory is from the limits.

The problem formulation, similarly to the OCP one and shown in Equation 3, has a cost function defined as previously but now it has been add a target position. As before, the weight chosen should push the system to the joint limits thus, higher weight on the a target position (10^3) and smaller to joint velocities and torques (10^{-3}).

$$\begin{aligned}
\text{Minimize:} \quad & \sum_{k=0}^{N-1} w_q(q_k - q_{target})^2 + w_v\dot{q}_k^2 + w_u u_k^2 \\
\text{Subject to:} \quad & x_{k+1} = f(x_k, u_k) \quad k = 0, ..., N-1 \\
& u_k \in \mathcal{U} \quad k = 0, ..., N-1 \\
& x_k \in \mathcal{X} \quad k = 1, ..., N \\
& x_0 = x_{initial} \\
& x_N \in \mathcal{X}_f
\end{aligned} \tag{3}$$

Considering always a grid of 10000 points, the MPC has been performed at first without the final constrain and then adding it. The result without the terminal constraint is reported in Figure 4 where it is possible to see, as expected, that since the initial state is inside the viable set the MPC performs some steps but after a while it reach a point out the kernel and the process stops immediately due to infeasibility.

By elongating the time horizon T from 0.5 to 2 seconds, it is possible to notice that the system can reach the position without violating any bounds. A possible explanation for this behaviour could be the achievement of a value of N "large enough" to mimic the infinite horizon, in according with the *maximum output admissible theory.* Note that with an higher time horizon the MPC must compute more iterations for each step, increasing the total time.

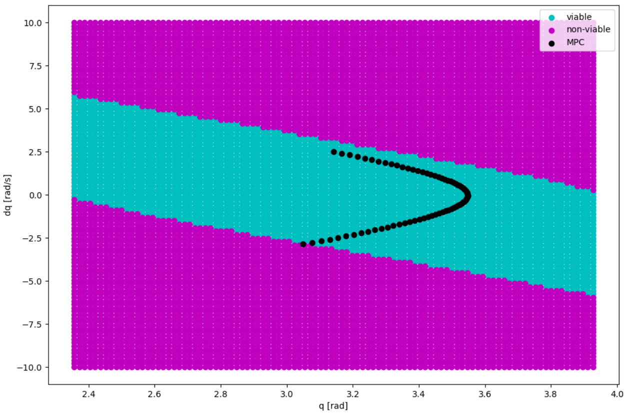

Figure 4: MPC result with no terminal constraint. T = 0.5s, initial state = [π, 2.5], target position = $\frac{3}{4}\pi$. Animation *Single_withoutTC*

Instead activating the final constraint the recursive feasibility is ensured also with T = 0.5, as shown in Figure 5 where the MPC's trajectory stay inside the viable kernel meaning that the target can be reached without violating the bounds.

By changing the value of the terminal constraint it has been notice that values too close to 1 (greater then 0.99999) the problem can not be solved since it exceed the maximum number of iteration, which should not be too high in order to not violate the time constraint.

This behavior is probably due to a too high severity of the constraint which results in a smaller $\mathcal{X}_f$.

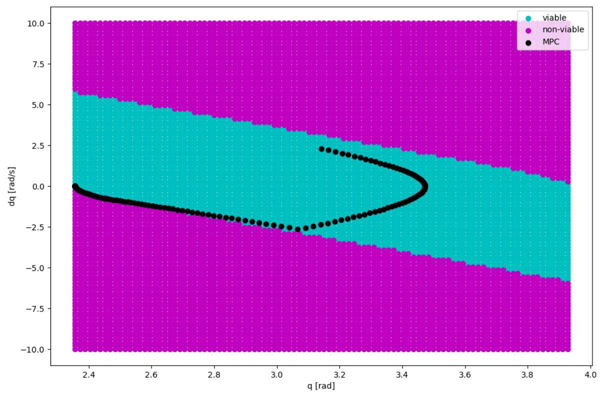

Figure 5: MPC result with terminal constraint. T = 0.5s, initial state = [π, 2.5], target position = $\frac{3}{4}\pi$. Animation *Single_withTC*

For the latter case also the evolution of the torque is reported in Figure 6, position and velocity in section 5 Figure 14 and an animation[4]of the single pendulum is attached.

From these graphs it is possible to notice that the system reach the imposed target in about 130 MPC steps after which the velocity remain zero and the torque constant to the value necessary to counteract the gravitational force. Decreasing the number steps lead to a fast execution but if too low the target will not be reached. To complete 200 steps with T = 0.5 the system takes about 30s.

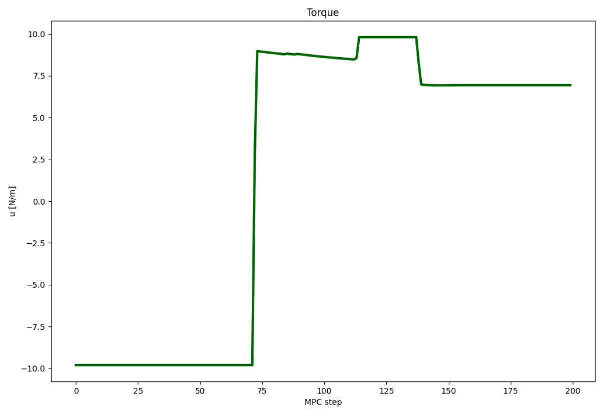

Figure 6: Torque evolution with terminal constraint. T = 0.5s, initial state = [π, 2.5], target position = $\frac{3}{4}\pi$. Animation *Single_withTC*

3 Double Pendulum

Now that the problem has been solved for a single pendulum it is possible to implement the double pendulum following basically the same steps, using the provided file for the dynamics.

[4]Reference taken for the animation

3.1 Optimal Control Problem

For what concern the OCPs formulation it is necessary to augment the state of two dimensions since now there are two positions and two velocities, therefore $x = [q_1, \dot{q}_1, q_2, \dot{q}_2] \in \mathbb{R}^4$ and also the input will be augmented of one dimension becoming $u = [u_1, u_2] \in \mathbb{R}^2$, thus the new formulation became as reported in Equation 4:

$$\begin{aligned} \text{Minimize:} \quad & \sum_{i=1}^{2} \sum_{k=0}^{N-1} w_{v,i}\dot{q}_{k,i}^2 + w_{u,i}u_{k,i}^2 \\ \text{Subject to:} \quad & x_{k+1} = f(x_k, u_k) \quad k = 0, ..., N-1 \\ & u_k \in \mathcal{U} \quad k = 0, ..., N-1 \\ & x_k \in \mathcal{X} \quad k = 1, ..., N \\ & x_0 = x_{initial} \end{aligned} \tag{4}$$

To compute a grid of 10000 points this time the OCP require around 1 hour and 40 minutes, due to the increased complexity. For this reason it has been decided to implement the possibility of adding new data on existing file, allowing the dataset to be enlarged later with smaller runs and thus reducing the computation time. Implementing always a grid with different dimensions, leads to consider multiple time the same data, thus it has been decided to implement a random sample strategy, as seen in the *Assignment 2*, as a fancy way to augment the dataset in specific region of the state space.

Moreover, visualize the fourth dimension is not possible hence no plots are generated.

3.2 Neural Network

Following the same considerations done in the single pendulum, a new Neural Network should be implemented. As a starting point the model previously created has been considered, changing the input layer to 4, as the dimensions of the states. Testing it with 10000 data sampled using the grid the results obtained are 0.982 of accuracy and 0.042 of loss, hence it has been decided to keep this structure.

The dataset is then augmented, using the random strategy, and the neural network is retrained. This time, even with an accuracy of 0.979 it is possible to notice a slightly over-fitting (Figure 7) therefore, some changing must be performed. After some research and testing different models, the chosen one consider two more *dropout* layers which randomly set inputs units to 0. In this way the accuracy decrease to 0.971 but the network seems working well (Figure 8).

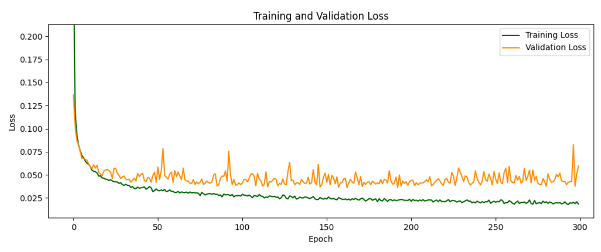

Figure 7: Accuracy with slight over-fitting

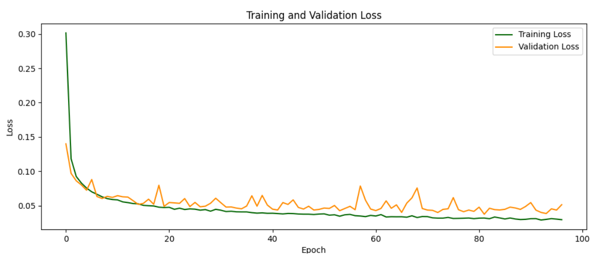

Figure 8: Accuracy with dropout at 50%

To improve more the network behaviour few tries have been performed changing the hyper parameters. A batch size of 64, a learning rate of 0.0005 and a dropout rate of 0.1 allow to have a smooth behaviour which converge slower but reach an accuracy of 0.992 and a loss of 0.022 (Figure 9). The time required to train the network with 10000 data sample from grid and 8000 random, is now around 45 seconds.

For completeness the confusion matrix is reported in Figure 10.

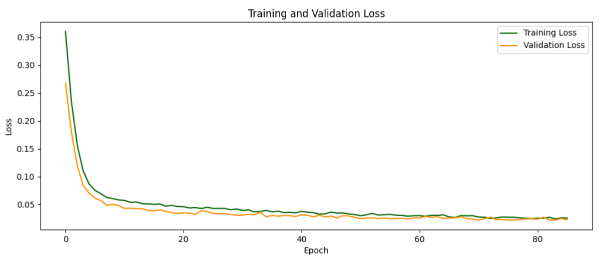

Figure 9: Accuracy after dropout with tuning of the hyper parameters

3.3 Model Predictive Control

As said before the problem idea remain the same but scaled for a double pendulum, thus it is necessary to consider double bounds, for the new dimensions and two target positions, one for each pendulum. For now the bound values are kept the same as in the single case. The formulation is reported in Equation 5.

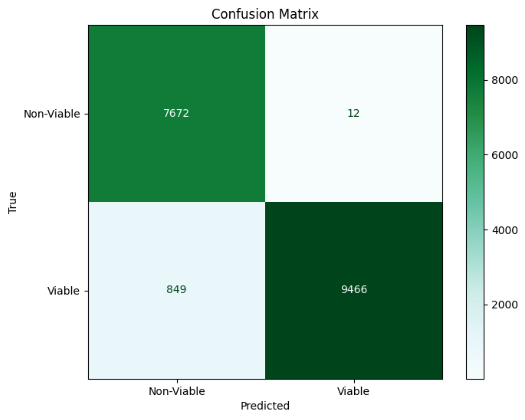

Figure 10: Confusion matrix double pendulum with 18000 random data

$$\begin{aligned}
\text{Minimize:} \quad & \sum_{i=1}^{2}\sum_{k=0}^{N-1} w_{q,i}(q_{k,i}-q_{target,i})^2 + w_{v,i}\dot{q}_{k,i}^2 + w_{u,i}u_{k,i}^2 \\
\text{Subject to:} \quad & x_{k+1} = f(x_k, u_k) \quad k = 0, ..., N-1 \\
& u_k \in \mathcal{U} \quad k = 0, ..., N-1 \\
& x_k \in \mathcal{X} \quad k = 1, ..., N \\
& x_0 = x_{sample} \\
& x_N \in \mathcal{X}_f
\end{aligned} \tag{5}$$

Maintaining the same limits as for the first pendulum, it has been decide to test the system with some interesting cases which involve the bounds, as required, and to visualize the results has been created a torques plot, a velocity over position plot for each pendulum and an animation. In particular two cases were tested, one where both the pendulums start from π with zero velocity and try to reach the lower bound $\frac{3}{4}\pi$ and a more challenging case where both the pendulums start to $\frac{5}{4}\pi$ and reach $\frac{3}{4}\pi$.

With this limit's value, both the cases failed with and without the terminal constrain. In particular, in the first case, when the the terminal constrains is active the problem fails due to “maximum number of iteration” while when disabled it the problem becomes infeasible.

Considering the dynamics and by looking at the animations (Figure 11) it has been notice that the first pendulum, the one attached to ground, stay at its maximum torque's value. In order to counteract the gravity force and the inertia of the second pendulum an higher torque should be applied, as discussed in the *Assignment 2*. Note that the increase in limits leads to an extension of the set hence a new Neural Network must be trained to obtain the new weights.

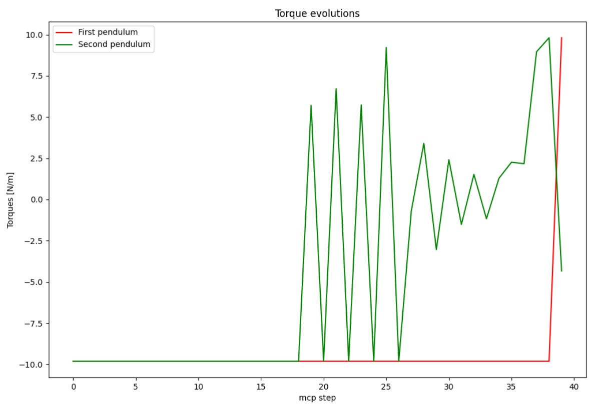

Figure 11: Torque behaviour of the double pendulum. T = 0.5s, initial state = [π, 0,π, 0], target positions = [$\frac{3}{4}\pi$, $\frac{3}{4}\pi$]. Animation *11_piTObound*

As first try, the input bounds of the first element are increased by five times and the accuracy of the new neural network became 0.981, with the hyper parameters previously chosen. Even in this case, the system is not able to solve the problem for the same reasons but now is the second element that saturates its torque, as can be seen in Figure 12.

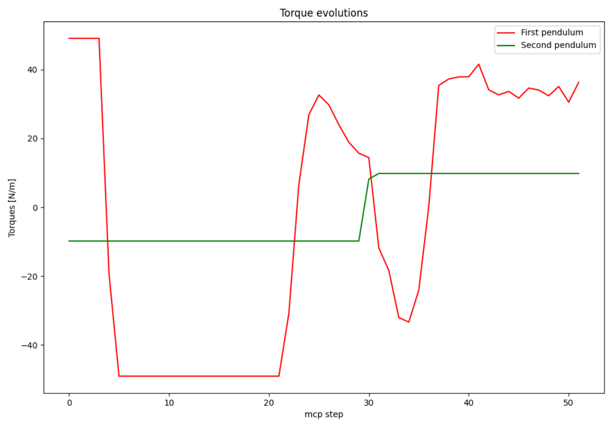

Figure 12: Torque behaviour of the double pendulum. T = 0.5s, initial state = [π, 0,π, 0], target positions = [$\frac{3}{4}\pi$, $\frac{3}{4}\pi$]. Animation *51_piTObound*

To overcome this problem also the torque's limits for the second pendulum has been increased to a value of four times. Moreover, to overcame the problem of the terminal constraint it has been thought to train the network with a smaller time horizon of 0.25 seconds and keeping 0.5 for the MPC. In this way the terminal constraint considers an enlarged viable set and thus the uncertain values of its bound will be considered non viable. With the new data the neural network has an accuracy of 0.966.

Solving now the MPC with the more conservative kernel, the system is able to reach mostly all the configurations, even if some initial conditions remain problematic and leads to exceed the maximum numbers of iterations. However, the system shows recursive feasibility with or without the terminal constrains. A possible explanation could be a too enlarged viable set or the time horizon values, as seen in the single pendulum case.

The results of the two cases analyzed are given in Figure 13, 15.

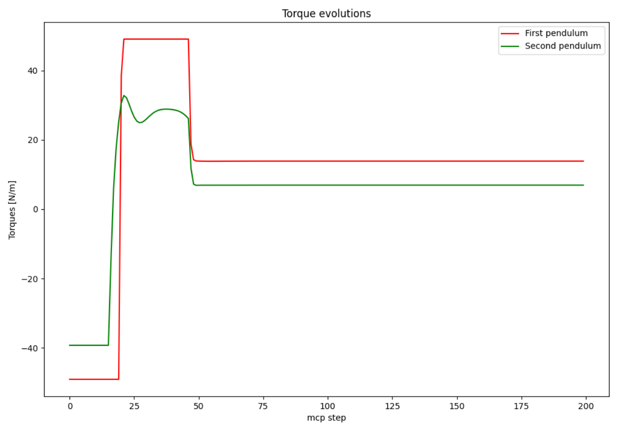

Figure 13: Torque behaviour *54_piTOboundTC* animation. T = 0.5s, initial state = [π, 0, π, 0], target positions = [$\frac{3}{4}\pi$, $\frac{3}{4}\pi$]

To completeness some animations of the interesting case are attached where is possible to visually see how the two system behaves.

4 Conclusions and possible implementations

The task of this project was to show that, by adding a constraint on the terminal state of an MPC, the problem remains recursively feasible even in situations where feasibility would be lost.

In the single pendulum case this goal can be reached, with a sufficiently small finite time horizon, changing a bit the bound of the terminal constraint base on the initial state chosen. In the case of the double pendulum instead, some problems has been encounter in proving the feasibility in fact, as said, often the problem exceed the maximum number of iteration.

Possible explanations of this behaviour are in the developed Neural Network since, respect the single pendulum case, is more complex due to the increased complexity of the system's dynamics. Another aspect of the network that needs to be considered is the output value which should be better explored and understood especially near the bounds where most of the problems probably happen. In fact, working with the neural network output for the double case, the predicted values are no more as for the single pendulum very close to 1 or to 0 but reach also intermediate value. Proof of this can be found in the last case where, using a larger viable set to train the neural network, the results slightly improve however for some initial condition the problem cannot even start with the terminal constraint but is able to reach the solution without.

Other problems outside the Neural network can be find in the proper set of the parameters as the time horizon, the input bounds, weights on the running cost etc.

Possible implementations that can be done are:

- Better dataset creation by pick more data near the bounds. This can be also implemented by using the Neural Network output, so taking points near the ones which produce uncertain value in exit;
- Create a better Neural Network by tuning more the hyper parameters or changing the optimizer;
- For a faster computation of the dataset a more advanced strategy could be applied to split the operations on other core when one has finished its own computation since at the moment they do not finish at the same time;
- Stop the MPC when the target configuration has been reached, so the system do not iterate when the pendulum is in position.

5 Attachments

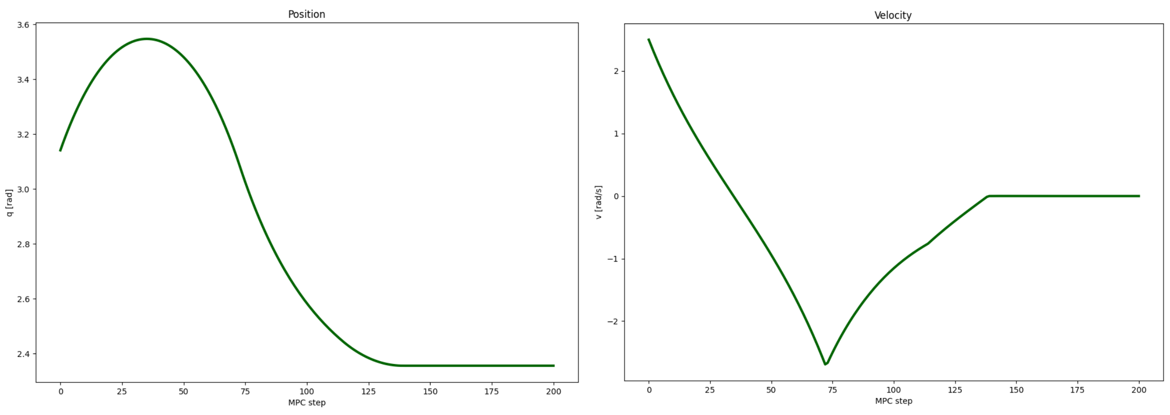

Figure 14: Position and velocity evolution with terminal constraint. T = 0.5s, initial state = [π, 2.5], target position = $\frac{3}{4}\pi$

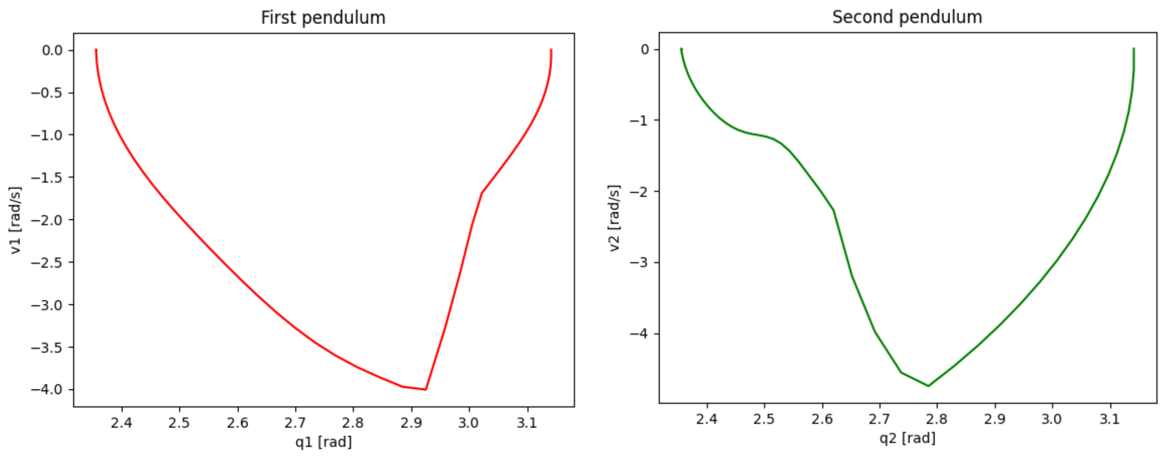

Figure 15: Velocity over position evolution for both pendulum. T = 0.5s, initial state = [π, 0, π, 0], target positions = [$\frac{3}{4}\pi$, $\frac{3}{4}\pi$]. Animation *54_piTOboundTC*.

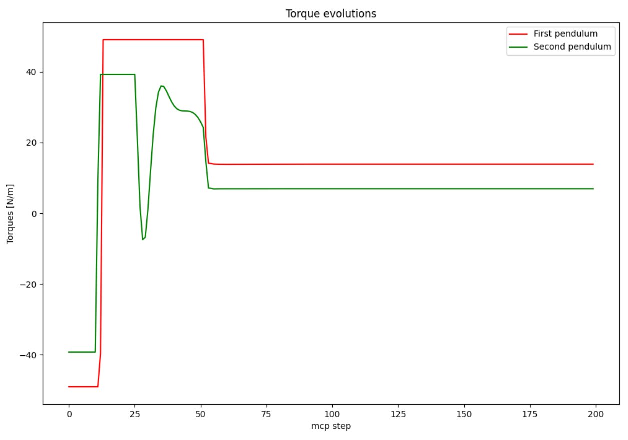

Figure 16: Torque behaviour. T = 0.5s, initial state = [$\frac{5}{4}\pi$, -2, $\frac{5}{4}\pi$, -6], target positions =[$\frac{3}{4}\pi$, $\frac{3}{4}\pi$]

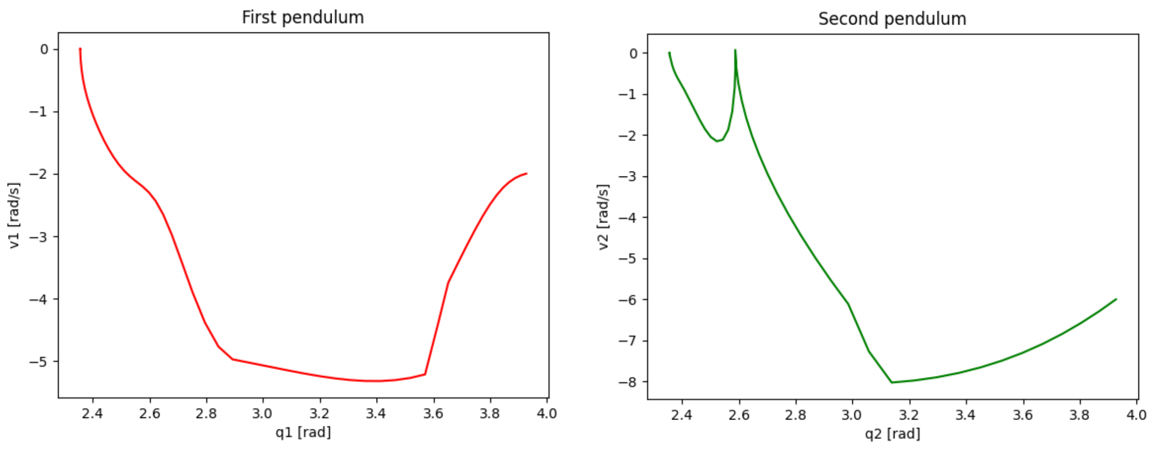

Figure 17: Velocity over position evolution for both pendulum. T = 0.5s, initial state = [$\frac{5}{4}\pi$, -2, $\frac{5}{4}\pi$, -6], target positions =[$\frac{3}{4}\pi$, $\frac{3}{4}\pi$]. Animation *54_AllboundTC*