UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Optimal Control

OPTCON AIRCRAFT PROJECT REPORT

Professor: **Giuseppe Notarstefano**

Students:
Mengozzi Riccardo
Vespignani Lorenzo
Francesconi Nicola

Academic year 2023/2024

Abstract

The objective of this project is to design the optimal control law of a fictitious Supersonic Aircraft. In order to absolve this tasks, once the dynamic of the aircraft has been discretized, the trajectories of the Aircraft must be optimized through the Newton's method algorithm, with a variable step size selection obtained through Armijo algorithm.
In the first task we applied the optimization algorithm to a step-reference trajectory, then the second task was the same but with a different reference trajectory, which was smooth instead of a step.
Then, in the third task, an optimal feedback controller has to be defined using the LQR (Linear Quadratic Regulator) algorithm in order to be able to track the optimal trajectory obtained in previous tasks even in case of perturbation of the initial condition. In the fourth task we tried a different control technique to track the optimal trajectory: MPC (Model Predictive Control). Finally we created an animation that shows the real behaviour of our Aircraft compared to a "Ghost Aircraft" which follow the desired (non feasible) trajectory.

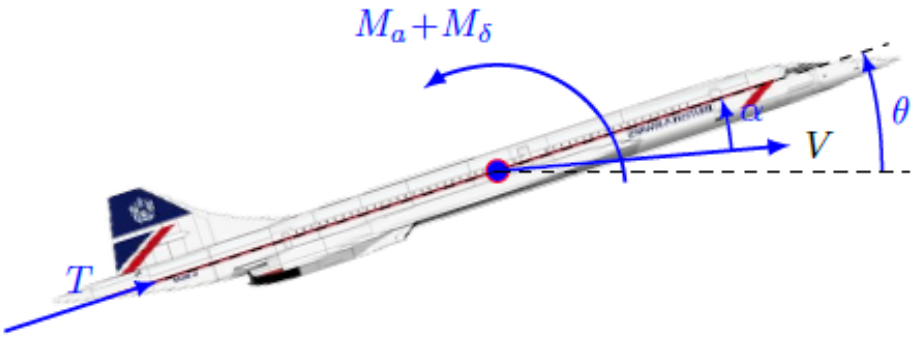


Figure 1: Supersonic Aircraft

Contents

Task 0 - Aircraft Dynamics

0.1 Aircraft Dynamics

In order to accomplish this task, we have created the file **"dynamics.py"**. In this file is present a function: **dynamics()** that takes as input parameter the actual state vector at time t **xx**, the applied inputs vector **uu** at time t and the discretization time **dt**. The function then simulates the discretized behaviour of the aircraft and returns the state vector at next dt and the linearization of the A_t and B_t which are respectively the jacobians of the dynamics with respect to states vector and inputs vector.

The given continuos time dynamics of the system were represented by the following differential equations:

$$
\begin{aligned}
m\dot{V} &= T(V,\delta_T)\cos(\alpha) - D(V) - mg\sin(\theta-\alpha) \\
\dot{\alpha} &= q - \frac{1}{mV}\big(T(V,\delta_T)\sin(\alpha) + L(V) + L_\delta(V,\delta_c,\delta_e) - mg\cos(\theta-\alpha)\big) \\
\dot{\theta} &= q \\
J\dot{q} &= M_a(V) + M_\delta(V,\delta_c,\delta_e).
\end{aligned}
$$

Figure 2: Continuous Time dynamic model of the Aircraft

For the discretization we used **Forward Euler discretization** for which

$$x_{t+1} = x_t + dt \cdot f(x,u)$$

Chapter 1

Task 1 - Optimal Trajectory given a Step Reference

1.1 Introduction

In order to accomplish this task we have created the folder **Task1** which contains all the files necessary for the simulation. In particular the file **task1.py** is the one to run in order to see the results.

In this task is required to generate a step reference both for states and inputs and, once a cost function has been designed, optimize the sequence of inputs to apply to the system in order to minimize the cost.

1.2 Step Reference Trajectory Definition

In order to define a Step Reference Trajectory we first of all computed two different equilibria. To compute the equilibria we decided the values of the four state of the system and through a zero-finding problem we found out the optimal input u_e to apply to the system in order to keep the Aircraft in the desired x_e given as initial condition $x = x_e$

Once the two input vectors of the two equilibria has been computed, we generated a step reference where for the first half of the trajectory we want to keep the first equilibrium while in the second half we commute to the second equilibrium.

1.3 Cost Function Definition

Once a reference has been designed, we had to define a cost function to minimize. For such problem we defined a quadratic cost function:

$$J(x,u) = \frac{1}{2}[\sum_{t=0}^{T-1}(x_t - x_t^{Ref})^T Q(x_t - x_t^{Ref}) + (u_t - u_t^{Ref})^T R(u_t - u_t^{Ref}) + (x_T - x_T^{Ref})^T Q_T(x_T - x_T^{Ref})]$$

Where Q, Q_T, R are diagonal weight matrices designed as:

$$Q = Q_T = \begin{bmatrix} 1e-2 & 0 & 0 & 0 \\ 0 & 5e2 & 0 & 0 \\ 0 & 0 & 5e2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

1.4 Newton Algorithm

In order to minimize our cost function we were asked to use the **Newton Method for Optimal Control**. Our problem becomes an optimal control that reads:

$$\min_{\mathbf{x}\in R^{4\times T}, \mathbf{u}\in R^{3\times T}} \quad J(\mathbf{x},\mathbf{u})$$

$$\text{s.t.} \quad x_{t+1} = f_t(x_t, u_t) \quad \forall t = 1, \ldots, T-1$$

The solution of the problem for a given $Q, Q_T, R, x^{Ref}, u^{Ref}$ is obtained by an **Affine LQR Problem** which returns the sequence of Δu^k used to update the input at the following iteration. At each iteration of the newton algorithm the sequence of input to apply at the system is updated as:

$$u_t^{k+1} = u_t^k + \gamma \Delta u_t^k$$

Where γ is the step size that has been chosen with Armijo. For Armijo algorithm we have used the following parameters: $\beta = 0.7, c = 0.5, \gamma^0 = 1$

Once the new sequence of inputs u_t^{k+1} has been updated, the new state trajectory is updated aswell as:

$$x_{t+1}^{k+1} = f(x_t^{k+1}, u_t^{k+1}) \quad \text{with} \quad x_0^{k+1} = x_{init}$$

1.5 Results

Starting from an initial control input exactly equal to a sequence of inputs that keep the airplane in the first equilibrium, it is possible to see in fig:1.1 and fig:1.2 how at each iteration the control input and the relative state trajectory evolve.

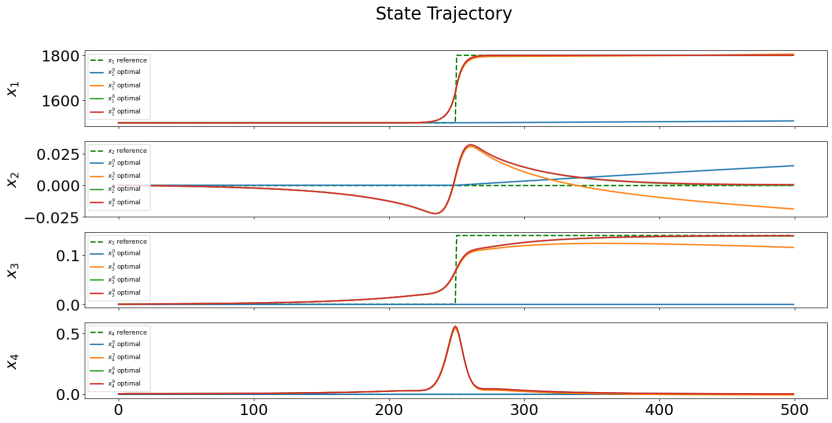


Figure 1.1: Plot of intermediate desired and optimal state trajectories

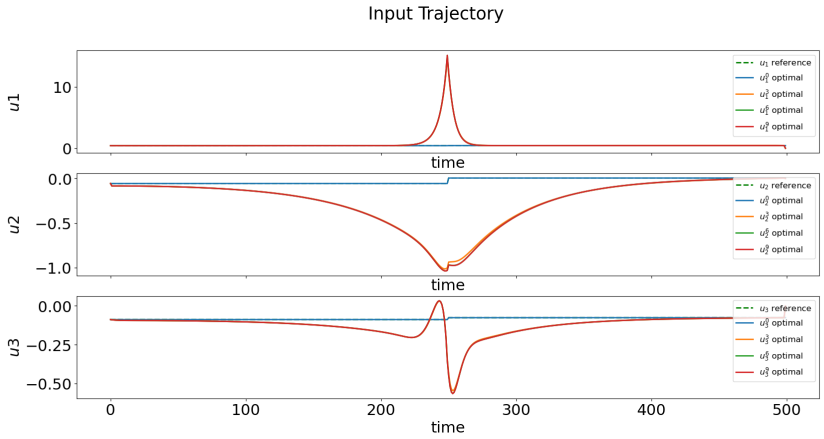


Figure 1.2: Plot of intermediate desired and optimal input trajectories

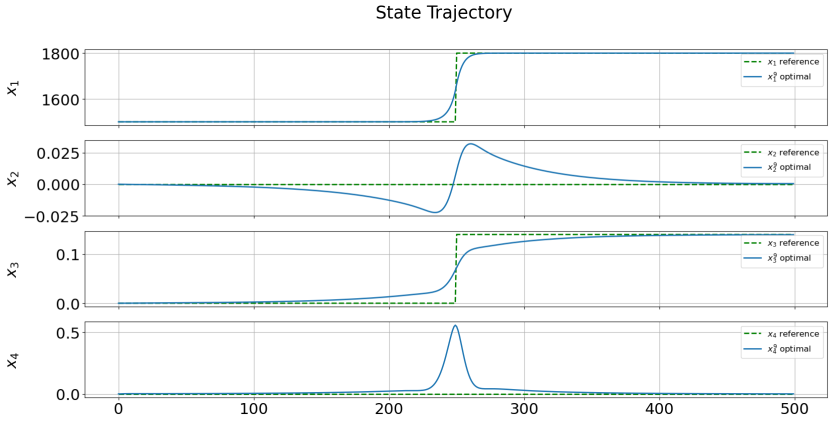


Figure 1.3: Plot of desired and optimal state trajectory

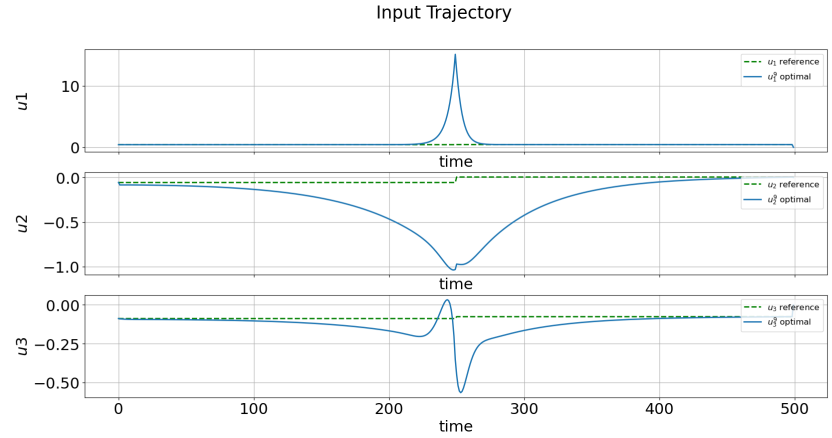


Figure 1.4: Plot of desired and optimal input trajectory

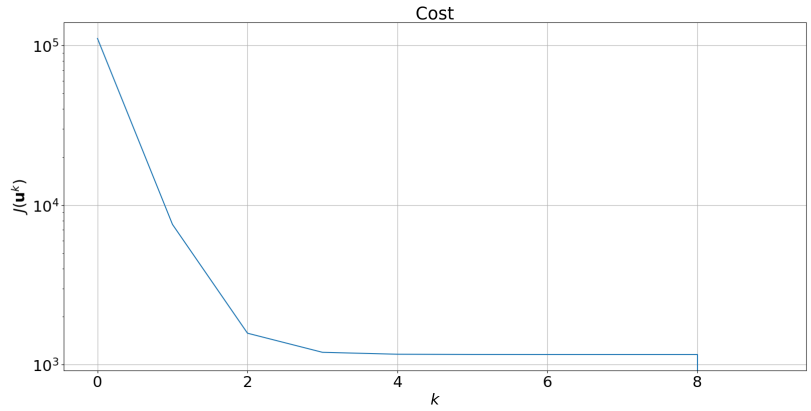


Figure 1.5: Plot of the cost for every iteration

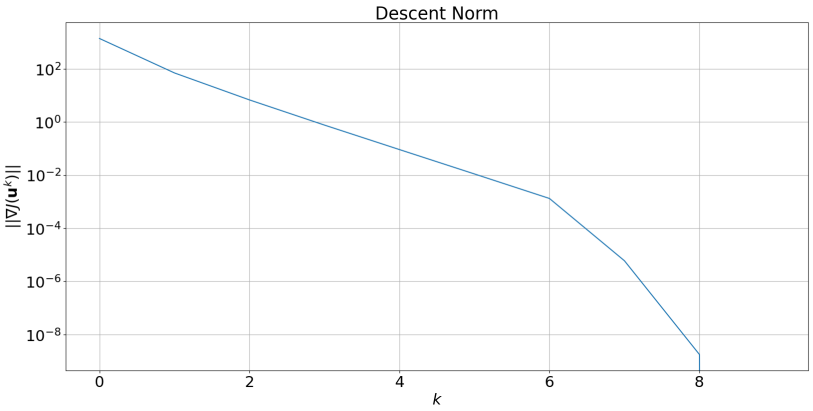
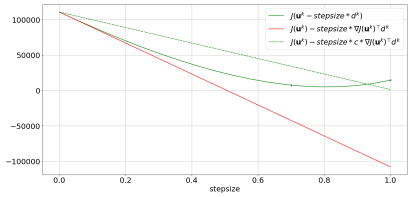
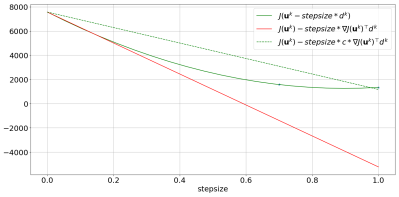


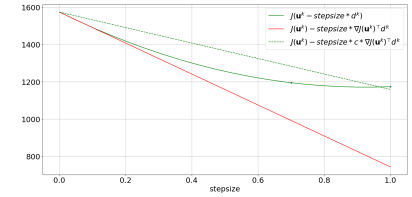
Figure 1.6: Plot of the norm of the descent for every iteration



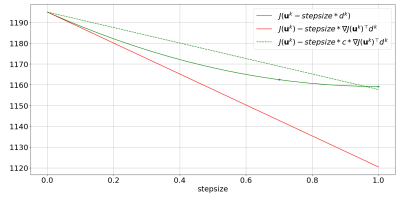
(a) $k = 1$



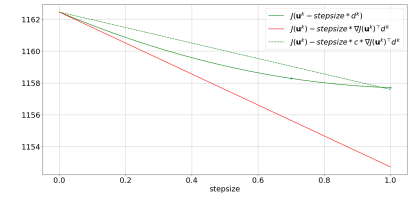
(b) $k = 2$



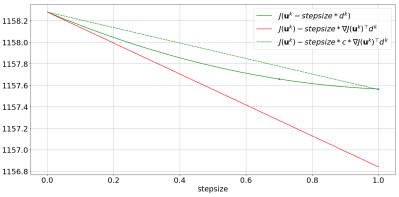
(c) $k = 3$



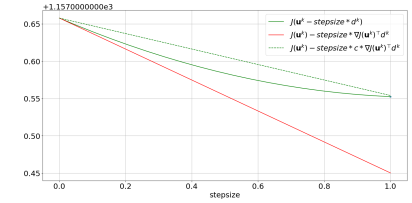
(d) $k = 4$



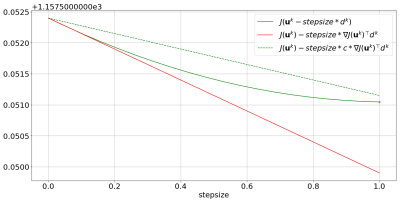
(e) $k = 5$



(f) $k = 6$



(g) $k = 7$



(h) $k = 8$

Figure 1.7: Armijo plot for every iteration

Chapter 2

Task 2 - Optimal Trajectory given a Smooth Reference

2.1 Introduction

In order to accomplish this task we have created the folder **Task2** which contains all the files necessary for the simulation. In particular the file **task2.py** is the one to run in order to see the results.

The goal of the Task 2 is to use a smooth reference trajectory instead of a step reference.

2.2 Smooth Reference Trajectory Definition

A smooth reference trajectory is defined as a collection of steps transitions between equilibria of the system, that means a **quasi-static trajectory**. To define the curve, first an initial and final state have been set, the states in between, that are part of the transition, are defined according to a polynomial. In our case a cubic or quintic polynomial have been used. Then, the corresponding input reference trajectory is defined as the collection of points that keep every correspondent point in the state trajectory in equilibrium. Also, to simulate an altitude change, two transitions have been defined. The following figure will show an example of smooth reference trajectory for the velocity:

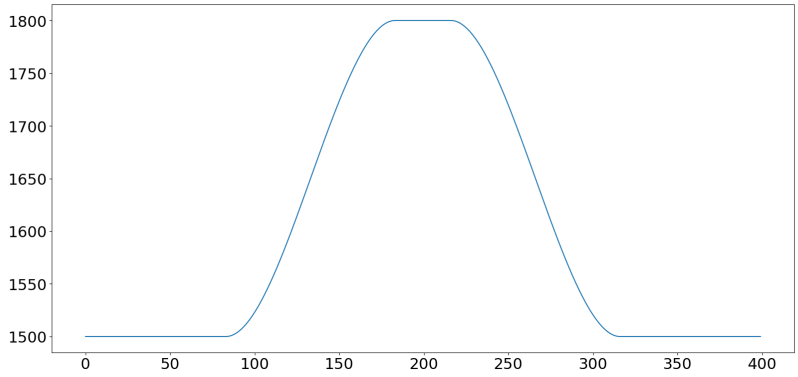


Figure 2.1: Plot of an example of smooth trajectory

As one can imagine, this type of trajectory is much easier for the system to follow. A part from the different reference trajectory definition, the Task2 is equal to the Task1.

2.3 Results

In the following figures the results are shown.

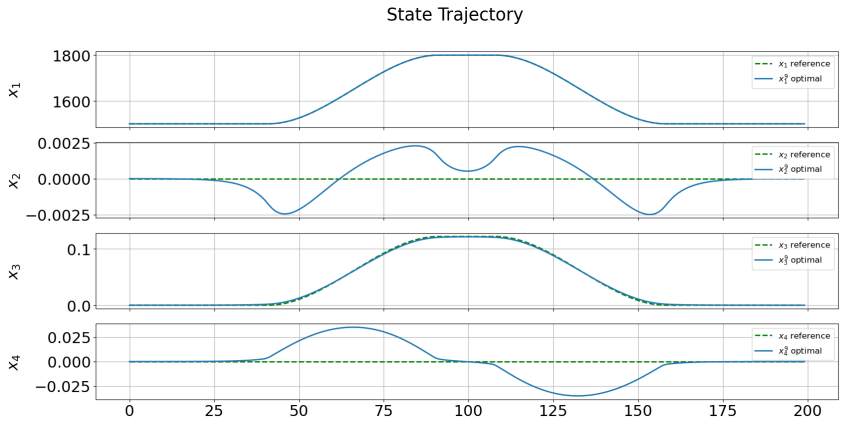


Figure 2.2: Plot of desired and optimal state trajectory

It is possible to see from fig:2.2 how the reference trajectory for the velocity and the angle θ are followed almost perfectly, differently from the Task 1 reference trajectory.

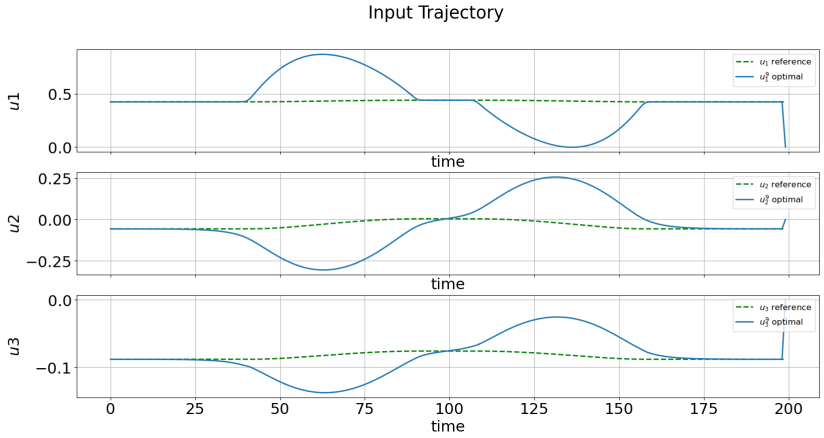


Figure 2.3: Plot of desired and optimal input trajectory

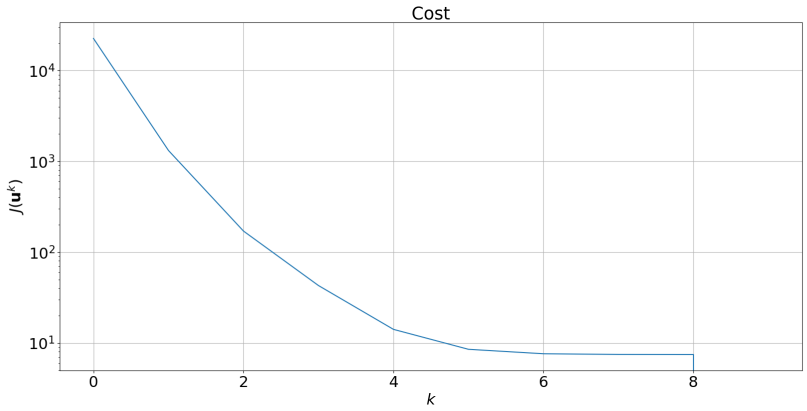


Figure 2.4: Plot of the cost for every iteration

Also, form fig:2.4, it can be seen how the cost is much smaller than the one in Task 1 (fig:1.5).

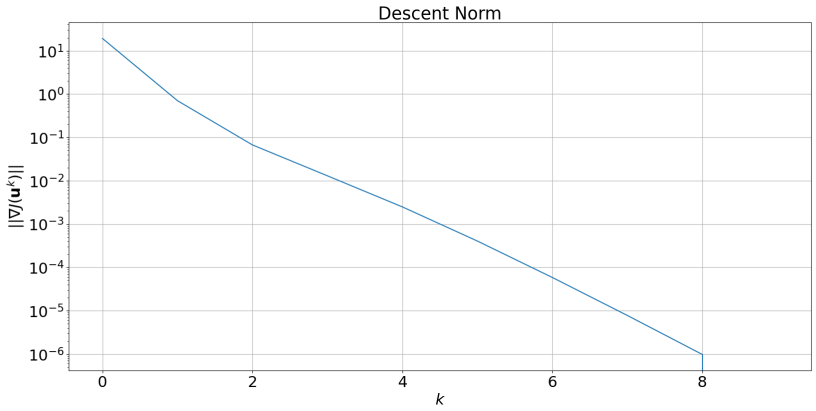


Figure 2.5: Plot of the norm of the descent for every iteration

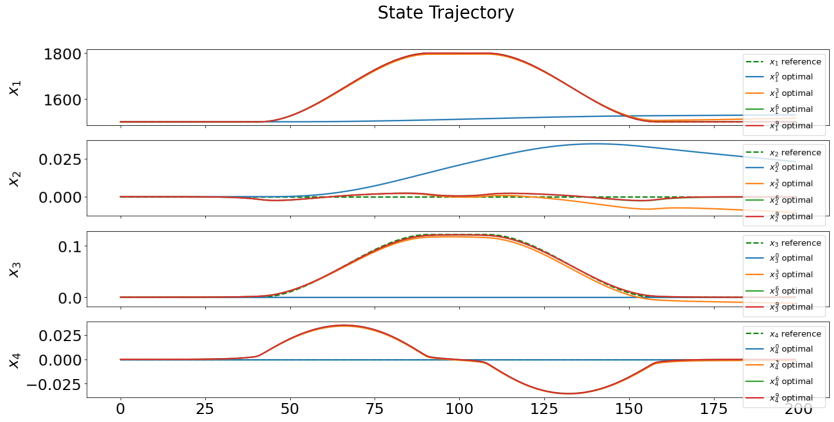


Figure 2.6: Plot of intermediate desired and optimal state trajectories

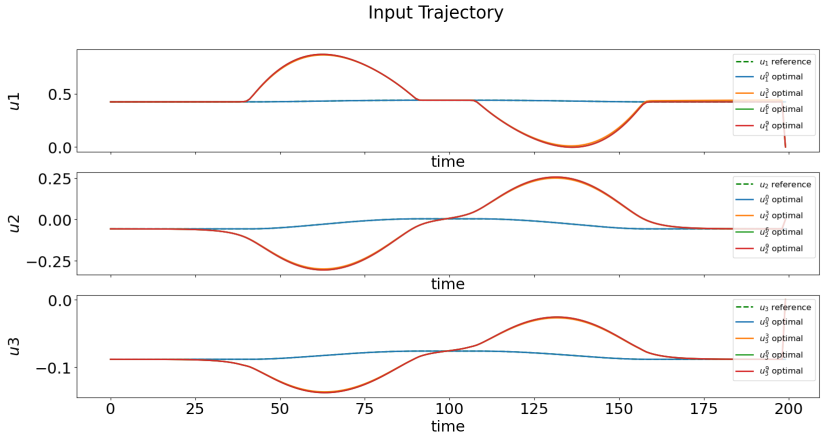
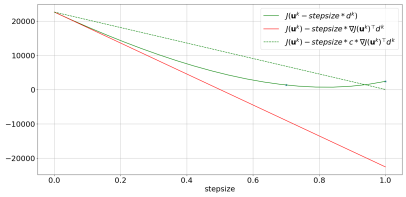
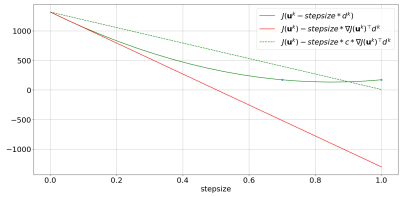


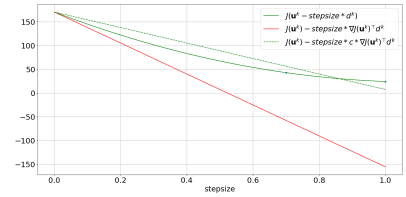
Figure 2.7: Plot of intermediate desired and optimal input trajectories



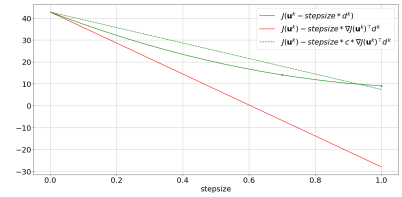
(a) $k = 1$



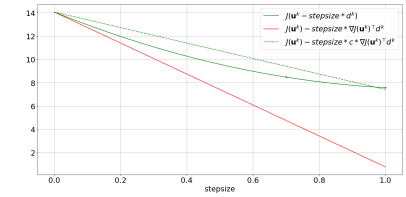
(b) $k = 2$



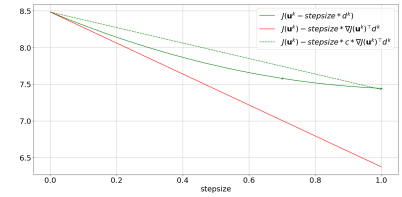
(c) $k = 3$



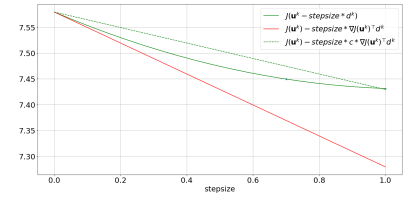
(d) $k = 4$



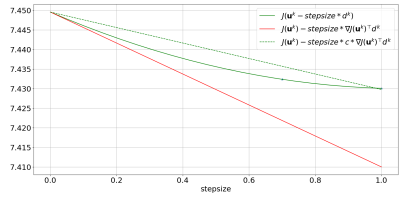
(e) $k = 5$



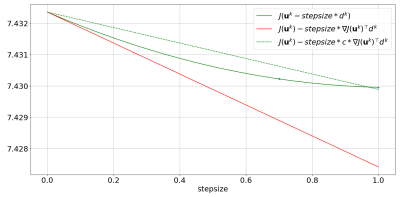
(f) $k = 6$



(g) $k = 7$



(h) $k = 8$



(i) $k = 9$

Figure 2.8: Armijo plot for every iteration

Chapter 3

Task 3 - LQR-based Trajectory Tracking

3.1 Introduction

In order to accomplish this task we have created the folder **Task3** which contains all the files necessary for the simulation. In particular the file **task3.py** is the one to run in order to see the results.

The goal of Task 3 is to track the optimal trajectory computed in the previous task with a LQR - based controller . To show if the system actually follows this trajectory, perturbation to the system need to be added, like a initial condition different to the one defined from the optimal trajectory.

3.2 LQR Problem

To solve the task, the input and the state need to be update as follows:

$$u_t = u_t^{opt} + K_t^{reg}(x_t - x_t^{opt})$$
$$x_{t+1} = f_t(x_t, u_t)$$

Where K_t^{reg} is computed solving the Riccati Difference Equation of the following LQ problem.

$$\min_{\substack{\Delta x_1,\dots,\Delta x_T \\ \Delta u_0,\dots,\Delta u_{T-1}}} \sum_{t=0}^{T-1} \Delta x_t^\top Q^{\text{reg}} \Delta x_t + \Delta u_t^\top R^{\text{reg}} \Delta u_t + \Delta x_T^\top Q_T^{\text{reg}} \Delta x_T$$
$$\text{subj.to } \Delta x_{t+1} = A_t^{\text{opt}} \Delta x_t + B_t^{\text{opt}} \Delta u_t \qquad t = 0, \dots, T-1$$
$$x_0 = 0$$

Figure 3.1

The weight matrices Q^{reg},R^{reg} and Q_T^{reg} have been set as in task 1.

3.3 Results

In the following pictures (fig:3.2,fig:3.3) the result of the state and input trajectory and error given a random initial condition are shown.

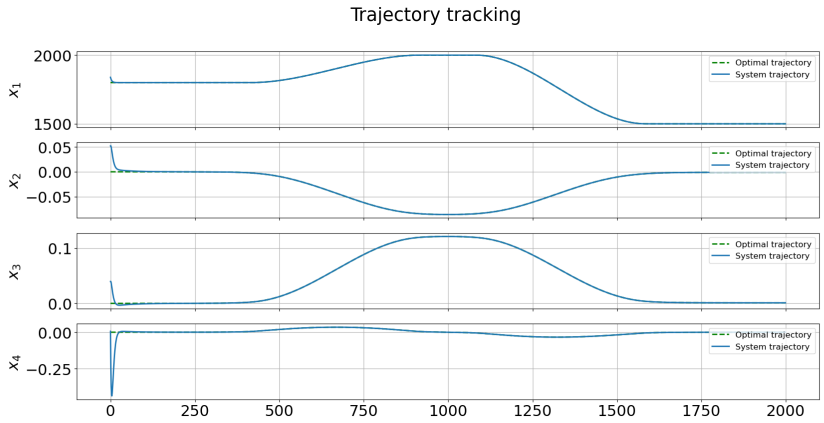


Figure 3.2: Plot of optimal and tracking state trajectory

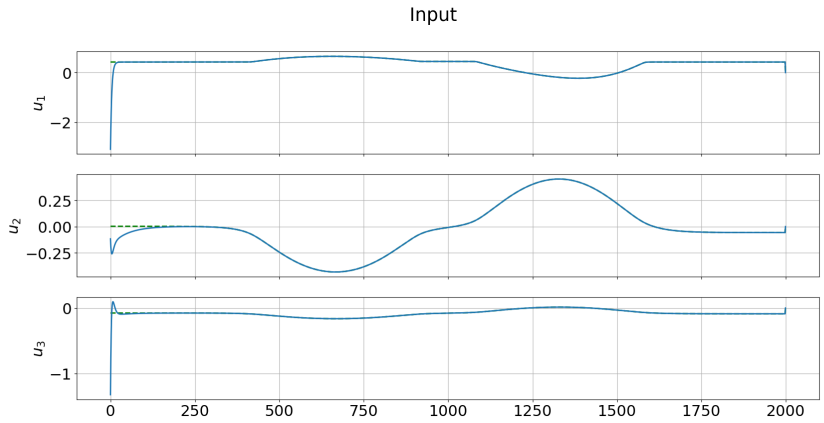


Figure 3.3: Plot of optimal and tracking input trajectory

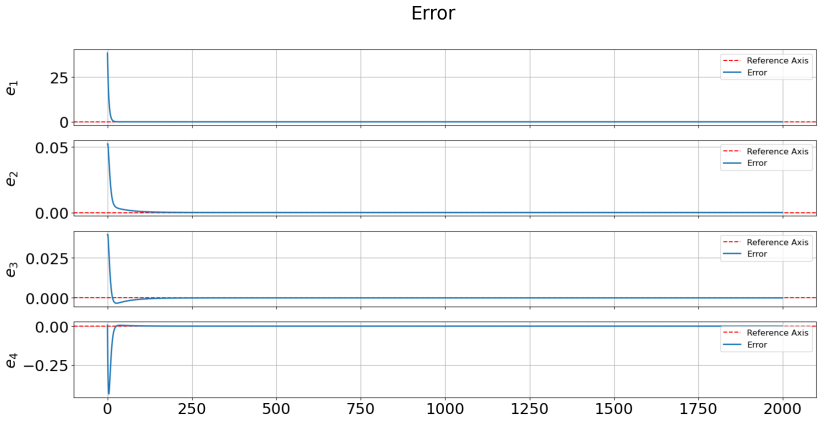


Figure 3.4: Plot of the error

As it can be seen, after a very short transient, the trajectory are tracked perfectly and the error goes to zero.

In the next figure (fig:3.5), the error given 4 different initial conditions with increasing magnitude of perturbation is shown.

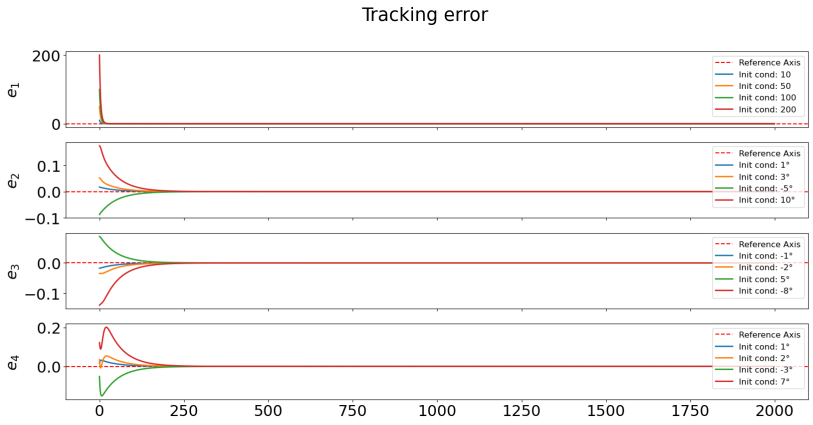


Figure 3.5: Plot of the error given different initial conditions

Chapter 4

Task 4 - MPC

4.1 Introduction

In order to accomplish this task we have created the folder **Task4** which contains all the files necessary for the simulation. In particular the file **task4.py** is the one to run in order to see the results.

The goal of this task is to design a MPC to track the optimal trajectory of Task2 even in case of perturbation of the initial condition or/and a perturbation of the nominal parameters of the dynamic model.

4.2 Model Predictive Control

Considering a disturbance acting on the system, if the optimal control input trajectory evaluated in previous task is applied to the system open loop, the overall system would be deviated with respect to the desired behaviour, as we can see in the next figure.

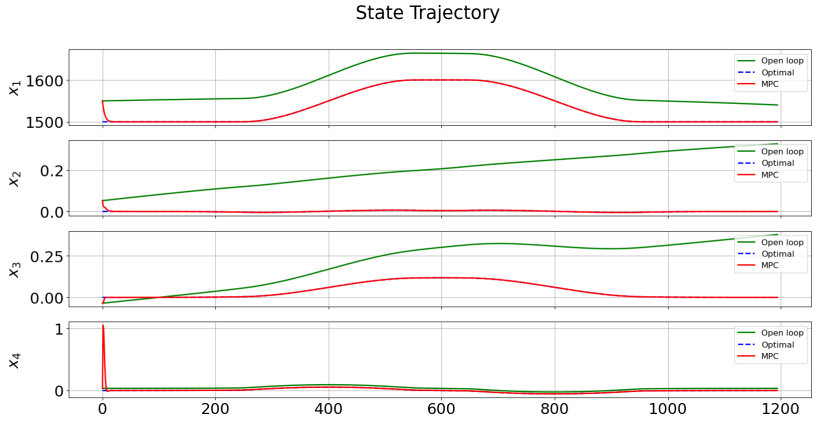


Figure 4.1: Plot of the open loop versus the MPC state trajectory

At each time t we measure the state error x_t^{meas} then we solve the minimization problem:

$$\min_{\substack{x_t,\dots,x_{t+T} \\ u_t,\dots,u_{t+T-1}}} \quad \sum_{\tau=t}^{t+T-1} x_\tau^\top Q_\tau x_\tau + u_\tau^\top R_\tau u_\tau + x_{t+T}^\top Q_T x_{t+T}$$

$$\begin{aligned} \text{subj.to} \quad & x_{\tau+1} = A_\tau x_\tau + B_\tau u_\tau && \forall \tau = t, \dots, t+T-1 \\ & x_\tau \in \mathcal{X},\ u_\tau \in \mathcal{U} && \forall \tau = t, \dots, t+T \\ & x_t = x_t^{\text{meas}} \end{aligned}$$

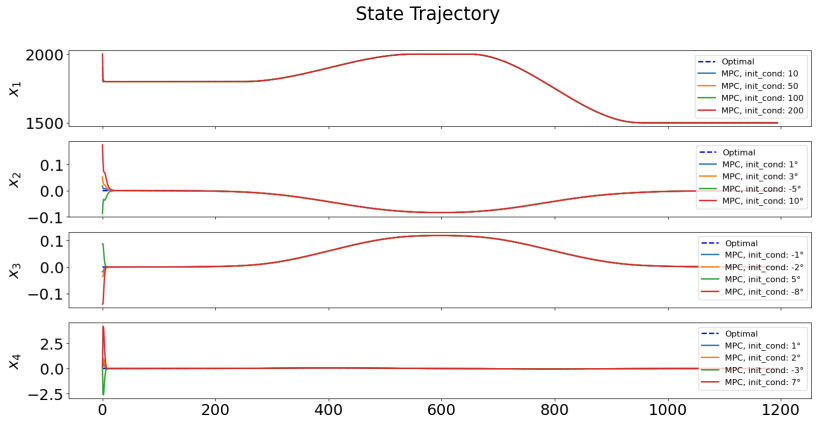
Figure 4.2: Minimization Problem where T is the prediction horizon, A_τ and B_τ are the linearized matrix of the system along the predicted trajectory.

The solution of this minimization problem gives us a sequence of inputs $[\mathbf{u_0}, \mathbf{u_1}, \ldots, \mathbf{u_{Tpred-1}}]$ that should bring the system to the optimal trajectory.

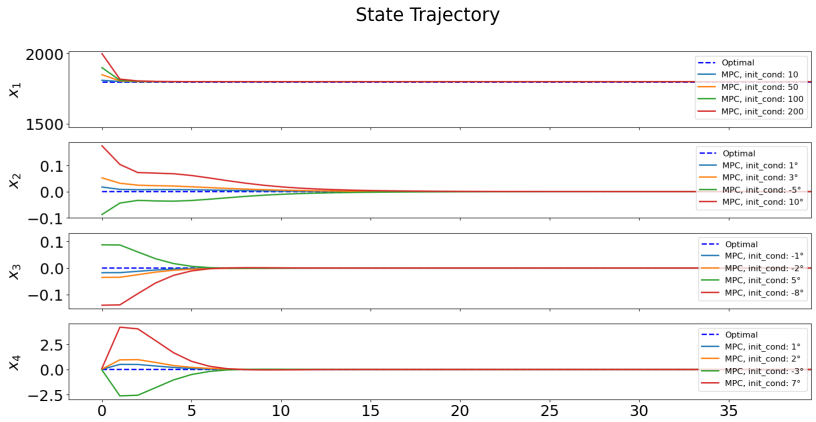
Once the sequence of inputs is updated, the first input $\mathbf{u_0}$ is applied to the system and a prediction of how the system will evolve in the next Tpred is done. Then the procedure is repeated.

4.3 Results

In figures 4.3,4.4 it's possible to see how starting from a perturbed initial condition the system reach the optimal trajectory after few iterations.

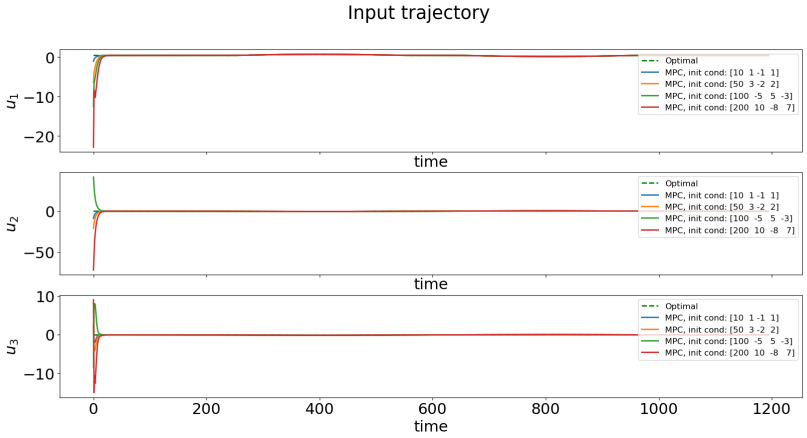


(a) Normal plot

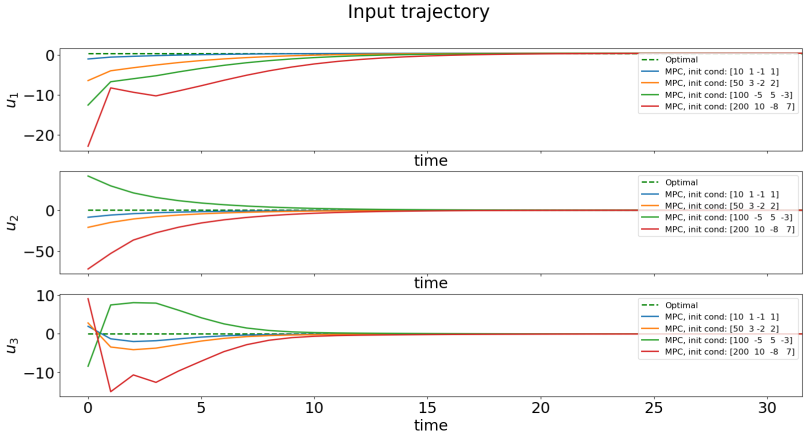


(b) Zoomed plot

Figure 4.3: Plot of the optimal and MPC state trajectories for different initial conditions



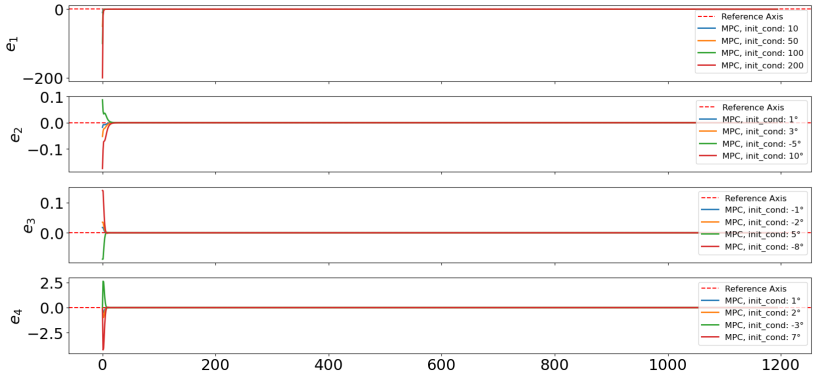
(a) Normal plot



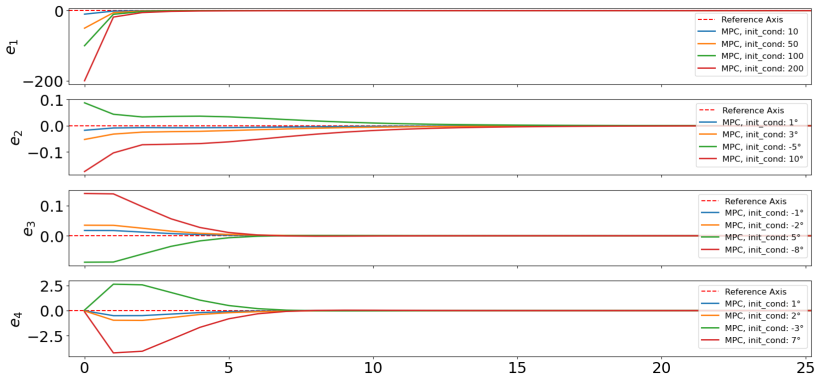
(b) Zoomed plot

Figure 4.4: Plot of the optimal and MPC input trajectories for different initial conditions

In figure 4.4 it's possible to see how, once the Aircraft reach the optimal trajectory, the applied input is exactly equal to the optimal one, since we are on the optimal trajectory and the optimal input, if no disturbances are present, keep the aircraft on the optimal state trajectory.



(a) Normal plot



(b) Zoomed plot

Figure 4.5: Plot of the error for different initial conditions

Then we tried to add model uncertainties modifying the key parameters of the aircraft dynamic model by a percentage error. In such a way, even if the aircraft reach the optimal trajectory, the optimal input will not keep it on it since the real dynamic is different from the nominal dynamic for which the optimal input sequence has been computed.

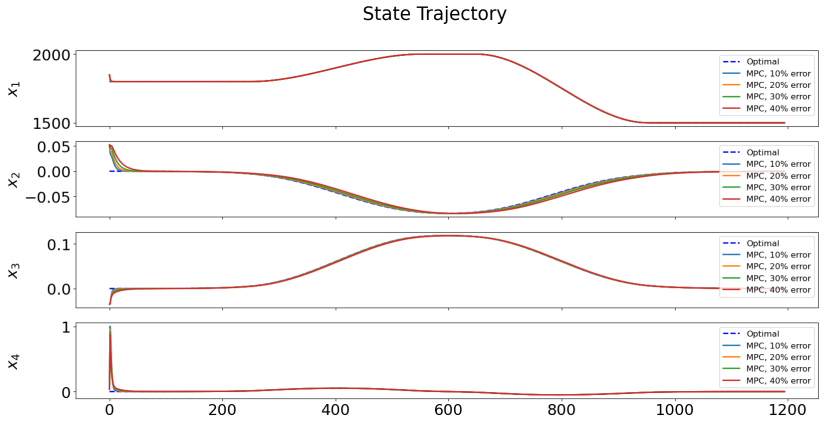


Figure 4.6: Plot of the optimal and MPC state trajectories given different uncertainties on the dynamics

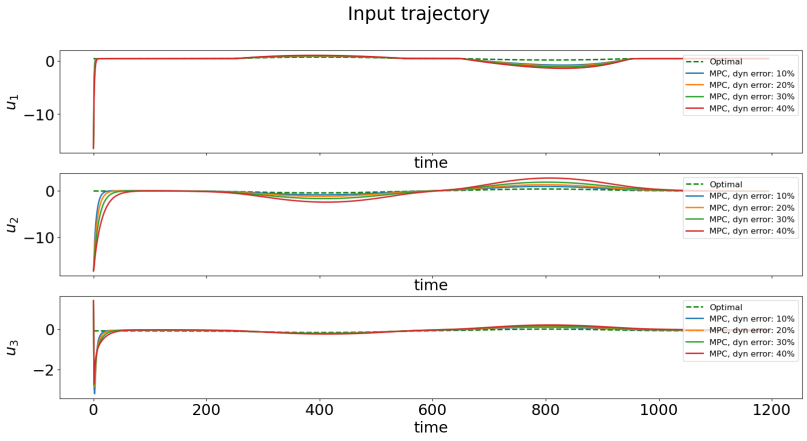


Figure 4.7: Plot of the optimal and MPC input trajectories given different uncertainties on the dynamics

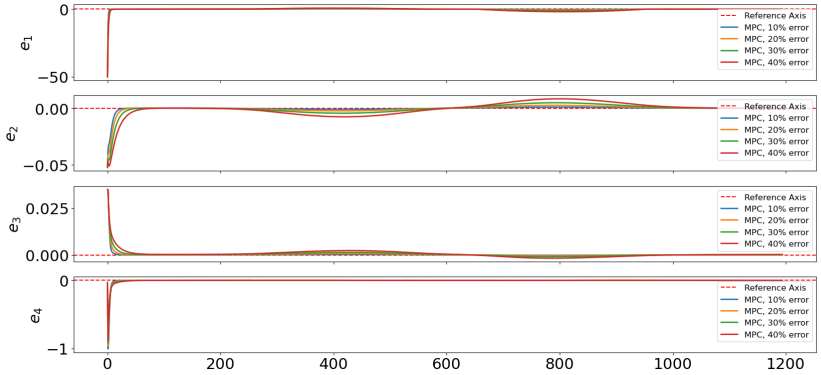


Figure 4.8: Plot of the error given different uncertainties on the dynamics

Conclusions

4.4 Task5 - Animation

To conclude the project, two animation have been created:

1. Animation 1: Created from Task1. Simulates a takeoff procedure: given a step reference in the velocity and in the inclination of the aircraft with respect to the ground (angle θ), the animation shows the optimal/feasible trajectory versus the reference/infeasible one.

2. Animation 2: Created from Task3. Simulates an altitude change: given a optimal trajectory created in Task 2 and a perturbed initial condition, the animation shows how the aircraft, after a short transition, track the optimal trajectory with a certain offset given by the not optimal initial condition.

The animations can be found in the Project folder as .gif files.

4.5 Remarks

4.5.1 Cvxpy vs Riccati Difference Equation

The reason behind the use of *cvxpy* library to solve the LQR problem for optimal trajectory generation instead of the Riccati Difference Equation has been mainly for a higher stability in the execution of the code. Mainly, with Riccati, some simulation parameters were limited (such as simulation length, dt, Q, R) as the execution would crash otherwise. With *cvxpy* the range of values for all parameters is much wider: the simulation can be arbitrarily long, dt can be chosen as arbitrarily small and also the weight matrices Q and R can be chosen in much more different ways.

Also, armijo seems to work much better, as the descent direction is plotted right until a norm of $1e-7$ more or less, which is much smaller than the descent norm until which the armijo plot was correct for the Riccati solution.

Bibliography