UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Optimal Control

Group 19 - Optimal Control of a Vehicle

Professor: **Giuseppe Notarstefano**

Students:
Riccardo Bassi
Simone Cenerini
Marco Sartoni

Academic year 2022/2023

Abstract

The objective of this project is to design the optimal control law of a fictitious bicycle. In order to absolve this tasks, the trajectories of the bicycle must be optimized through the Newton's method algorithm. Then an optimal feedback controller has to be defined using the LQR (Linear Quadratic Regulator) algorithm. An additional constrain for the input control is added in order to simulate input saturation. Finally, the bicycle movement will be shown in an animation.

Contents

Chapter 1

Task 0 - Vehicle dynamics

In order to accomplish this task we have created the files `Dynamics_Simpy.py` and `Dynamics_Numpy.py` . In the latter, you can find the function *dynamics()* which simulates the dynamics and returns the state at next instant of time and the jacobians with respect to the states and the inputs.

1.1 State space and control inputs

The model of the vehicle is described by these equations:

$$\begin{aligned}
\dot{x} &= V_x \cos\psi - V_y \sin\psi \\
\dot{y} &= V_x \sin\psi + V_y \cos\psi \\
m(\dot{V}_x - \dot{\psi} V_y) &= F_x \cos\delta - F_{y,f} \sin\delta \\
m(\dot{V}_y - \dot{\psi} V_x) &= F_x \sin\delta + F_{y,f} \cos\delta + F_{y,r} \\
I_z \ddot{\psi} &= (F_x \sin\delta + F_{y,f} \cos\delta)a - F_{y,r} b
\end{aligned} \tag{1.1}$$

To accomplish the job first of all the dynamic is brought in a state space model form.
So the reference position (x,y) with respect to the global reference frame are represented now as the first two states. While the yaw angle of the vehicle is described with the third state equation and it is derivative is described with the sixth state equation. Finally the fourth and fifth equations are the velocity with respect to the moving reference frame of the chassis. In total six equations are used to completely describe the state of our system.

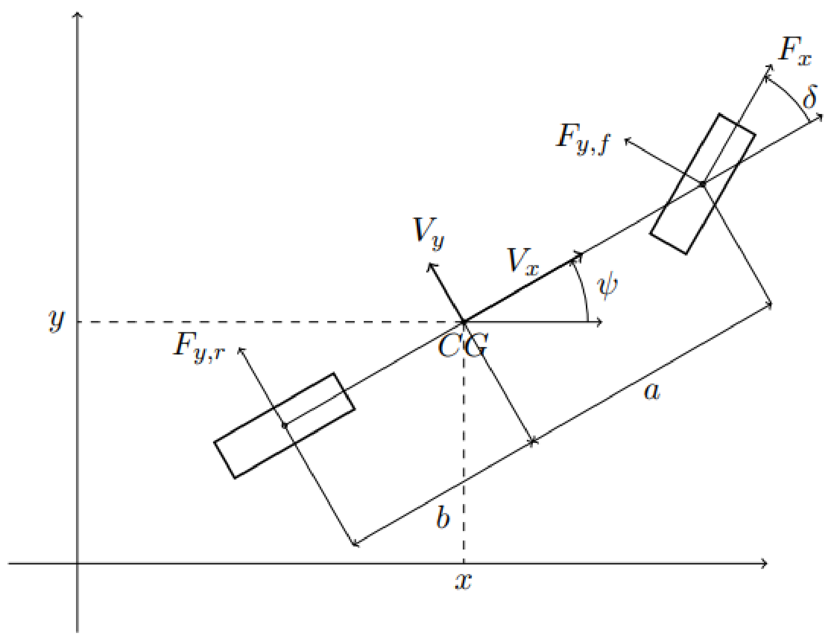

Figure 1.1: Bicycle vehicle

$$
\begin{aligned}
x_0 &= X \\
x_1 &= Y \\
x_2 &= \psi \\
x_3 &= V_x \\
x_4 &= V_y \\
x_5 &= \dot{\psi}
\end{aligned}
\tag{1.2}
$$

The first input variable is associated with the steering angle and the second one to the force applied to that wheel.

$$
\begin{aligned}
u_0 &= \delta \\
u_1 &= F_x
\end{aligned}
\tag{1.3}
$$

1.2 Forward Euler discretization

Forward Euler discretization is applied since the algorithm works in discrete time.

$$\begin{cases} x_{0_{t+1}} = x_{0_t} + dt(x_{3_t}\cos x_{2_t} - x_{4_t}\sin x_{2_t}) \\ x_{1_{t+1}} = x_{1_t} + dt(x_{3_t}\sin x_{2_t} + x_{4_t}\cos x_{2_t}) \\ x_{2_{t+1}} = x_{2_t} + dt(x_{5_t}) \\ x_{3_{t+1}} = x_{3_t} + dt(\frac{1}{m}(u_{1_t}\cos u_{0_t} - F_{y,f}\sin u_{0_t}) + x_{4_t}x_{5_t}) \\ x_{4_{t+1}} = x_{4_t} + dt(\frac{1}{m}(u_{1_t}\sin u_{0_t} - F_{y,f}\cos u_{0_t} + F_{y,r}) + x_{3_t}x_{5_t}) \\ x_{5_{t+1}} = x_{5_t} + dt(\frac{1}{I_z}(u_{1_t}\sin u_{0_t} - F_{y,f}\cos u_{0_t} + F_{y,r}b)) \end{cases} \tag{1.4}$$

$$\begin{cases} F_{y,f} = \mu F_{z,f}\beta_f \\ F_{y,r} = \mu F_{z,r}\beta_r \end{cases} \tag{1.5}$$

$$\begin{cases} \beta_f = \delta - \frac{V_y + a\dot{\psi}}{v_x} \\ \beta_r = -\frac{V_y - b\dot{\psi}}{v_x} \end{cases} \tag{1.6}$$

1.3 Jacobians computation

For future purposes the gradient with respect to state and input are calculated. To calculate the derivatives we used the Sympy library at first and then we copied the results into the function *dynamics()* since the functions that convert symbolic notation of Sympy to the numeric one of Numpy can create interference.

$$\nabla f_x(x,u)^T = \begin{bmatrix} \frac{\partial}{\partial x_1}f_1 \cdots \frac{\partial}{\partial x_1}f_m \\ \vdots \ddots \vdots \\ \frac{\partial}{\partial x_n}f_1 \cdots \frac{\partial}{\partial x_n}f_m \end{bmatrix}^T \tag{1.7}$$

$$\nabla f_u(x,u)^T = \begin{bmatrix} \frac{\partial}{\partial u_1}f_1 \cdots \frac{\partial}{\partial u_1}f_n \\ \vdots \ddots \vdots \\ \frac{\partial}{\partial u_k}f_1 \cdots \frac{\partial}{\partial u_k}f_n \end{bmatrix}^T \tag{1.8}$$

for the cost function, since it has been choosen quadratic, the gradients are given by:

$$\nabla l_x(x,u) = Qx = q \tag{1.9}$$

$$\nabla l_u(x,u) = Rx = r \tag{1.10}$$

and similarly the Hessians become the original choosen penality matreces.

$$\nabla^2 l_x x(x,u) = Q \tag{1.11}$$

$$\nabla^2 l_u u(x,u) = R \tag{1.12}$$

Chapter 2

Task 1 - Lane change manouver

In order to accomplish this task we have created the files `Task_1-2.py` . To launch this task, you have to impose the *Curve* variable equal to '*Step*' in "TO DO section".
You can also set the flags:

- *print_armijo*: to print in the terminal the Armijo's stepsize for each iteration.
- *visu_armijo*: to plot the Armijo graph at each iteration.
- *Save_output_opt*: to save the optimal trajectory in a `txt` file.

In the first task we were asked to perform a line change. We have created a step reference trajectory and we have given it to a Regularized Newton's method algorithm. The aim of the procedure is to track a reference curve given some weighting matrices in a way to selectively augment specific cost parts in which we are interested in. In our case for the first part we will use heavy weights for X and Y position to enforce the position tracking.

2.1 Algorithm structure

The algorithm is constructed in this way: as a cost function, we have chosen a quadratic form for both state and input. This because we wanted to track a reference state discrete evolution and also to set a saturation on actuators in order to achieve more realistic results.

Our problem becomes an optimal control problem that reads:

$$\begin{aligned} \min_{x_t,u_t} \quad & \sum_{t=0}^{T-1} x_t^T Q_t x_t + u_t^T R_t u_t + x_T^T Q_T x_T \\ \text{subject to} \quad & x_{t+1} = f(x_t, u_t) \qquad\qquad t = 0, ..., T-1 \\ & x_0 = x_{init} \end{aligned} \tag{2.1}$$

At starting time the position of the bicycle is at $(x, y) = (0, 0)$ and has initial velocity different from 0 otherwise the dynamic would be not feasible. So the initial guess trajectory will be a straight forward line.

We were asked to solve this problem via the Newton's method for optimal control algorithm.
The standard Newton's method expects the following matrices (we have omitted the iteration k for a more compact notation):

$$Q_t = \nabla^2_{x_t x_t} l(x_t, u_t) + \nabla^2_{x_t x_t} f(x_t, u_t)\lambda_{t+1} \tag{2.2}$$

$$R_t = \nabla^2_{u_t u_t} l(x_t, u_t) + \nabla^2_{x_t x_t} f(x_t, u_t)\lambda_{t+1} \tag{2.3}$$

$$S_t = \nabla^2_{x_t u_t} l(x_t, u_t) + \nabla^2_{x_t u_t} f(x_t, u_t)\lambda_{t+1} \tag{2.4}$$

Since we cannot guarantee that $Q \geq 0$ and $R > 0$, we have used the regularized version, which approximate those matrices as:

$$\tilde{Q}_t = \nabla_{x_t x_t} l(x_t, u_t) \tag{2.5}$$

$$\tilde{R}_t = \nabla_{u_t, u_t} l(x_t, u_t) \tag{2.6}$$

$$\tilde{S}_t = 0 \tag{2.7}$$

Namely solve the following affine LQR problem:

$$\begin{aligned} \min_{\Delta x_t, \Delta u_t} \quad & \sum_{t=0}^{T-1} \begin{bmatrix} q_t \\ r_t \end{bmatrix}^T \begin{bmatrix} \Delta x_t \\ \Delta u_t \end{bmatrix} + \begin{bmatrix} \Delta x_t \\ \Delta u_t \end{bmatrix}^T \begin{bmatrix} Q_t & 0 \\ 0 & R_t \end{bmatrix} \begin{bmatrix} \Delta x_t \\ \Delta u_t \end{bmatrix} + q_T^T X_T + \frac{1}{2} \Delta X_T^T Q_T \Delta X_T \\ \text{subject to} \quad & \Delta x_{t+1} = A_t \Delta x_t + B_t \Delta u_t \qquad t = 0, ..., T-1 \end{aligned} \tag{2.8}$$

where $q_t = \nabla_{x_t} l(x_t, u_t)$, $r_t = \nabla_{u_t} l(x_t, u_t)$, $A_t^T = \nabla_{x_t} f(x_t, u_t)$ and $B_t^T = \nabla_{u_t} f(x_t, u_t)$

So, at each iteration $k = 0, 1...max\_iters$:
we compute the time-varying q and r vectors and the matrices Q and R.

The quadratic program is efficiently solved using the augmented state:

$$\delta \tilde{x}_t = \begin{bmatrix} 1 \\ \Delta x_t \end{bmatrix} \tag{2.9}$$

Which allow us to derive the following equations to solve the associated LQR problem:

$$K_t^* = -(R_t + B_t^T P_{t+1} B_t)(S_t + B_t P_{t+1} A_t) \tag{2.10}$$

$$\sigma_t^* = -(R_t + B_t^T P_{t+1} B_t)^{-1}(r_t + B_t^T p_{t+1}) \tag{2.11}$$

$$p_t = q_t + A_t^T p_{t+1} - K_t^*(R_t + B_t^T P_{t+1} B_t)\sigma_t^* \tag{2.12}$$

$$P_t = Q_t + A_t^T P_{t+1} A_t - K_t^*(R_t + B_t^T P_{t+1} B_t) K_t^* \tag{2.13}$$

where: $p_T = q_T$ and $P_t = Q_T$

which can be used to solve our problem with the following closed loop update:

$$u_t^{k+1} = u_t^k + \gamma^k (K_t^k (x_t^{k+1} - x_t^k) + \sigma_t^k) \tag{2.14}$$

$$x_{t+1}^{k+1} = f(x_t^{k+1}, u_t^{k+1}) \tag{2.15}$$

This is iterated until the descent direction is lower than terminal condition. For the stepsize γ we have implemented the standard Armijo rule, with $\gamma_0 = 1$.

In the following pages, we have printed out several plots related to the algorithm results:

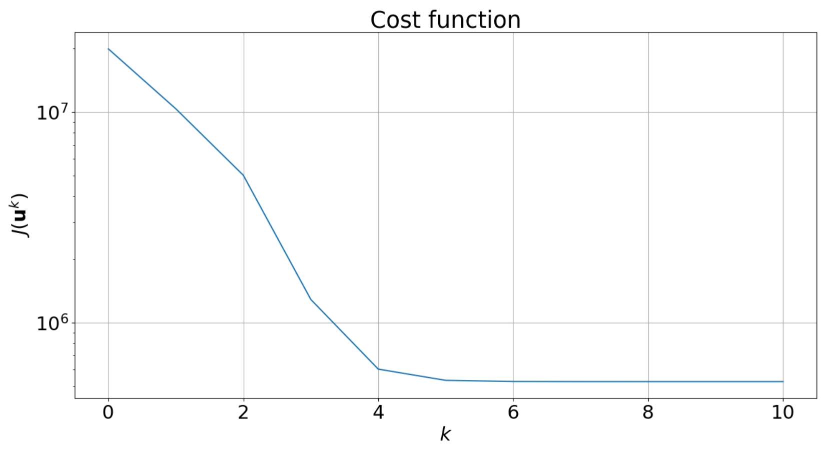

Figure 2.1: Plot of the cost function evolution

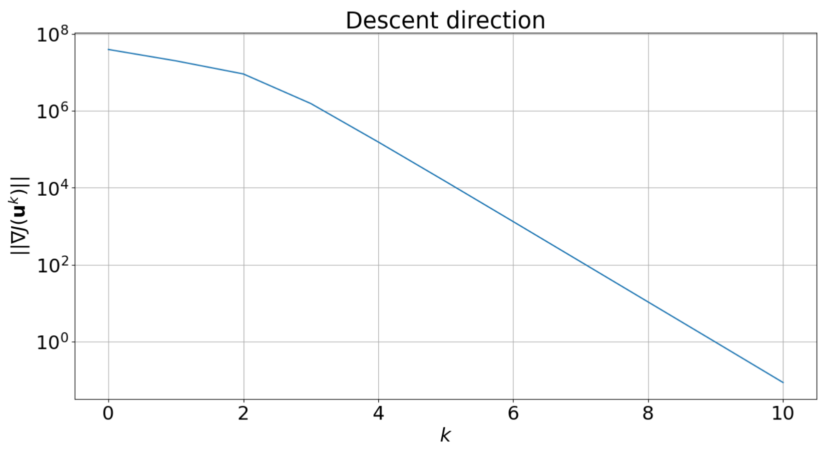

Figure 2.2: Plot of the descent direction evolution

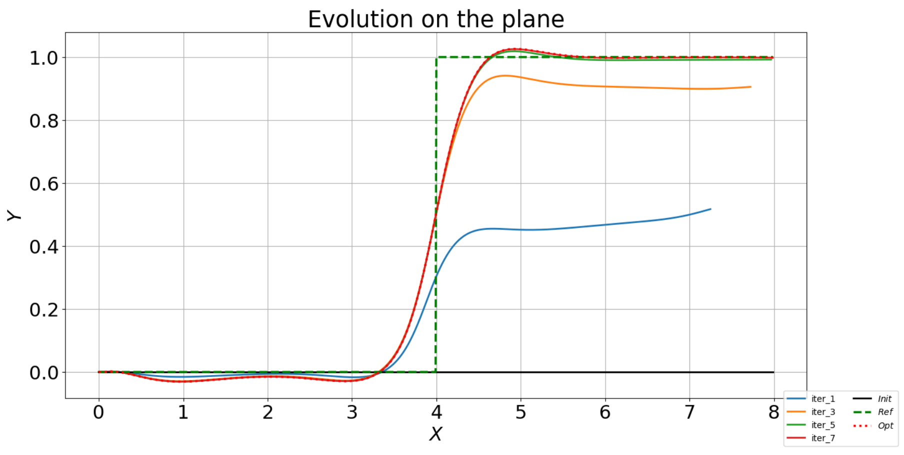

Figure 2.3: Evolution of the trajectory in different iterations

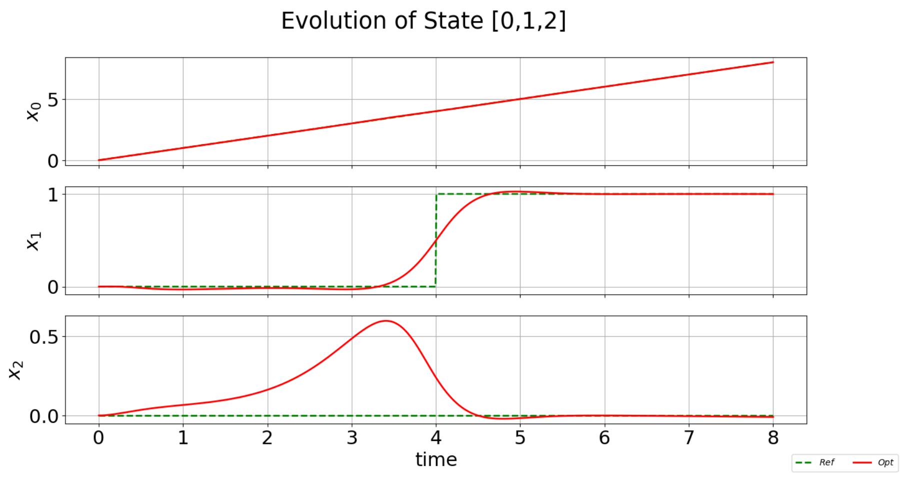

Figure 2.4: Plot of the $\{X_0, X_1, X_2\}$ reference and optimal states

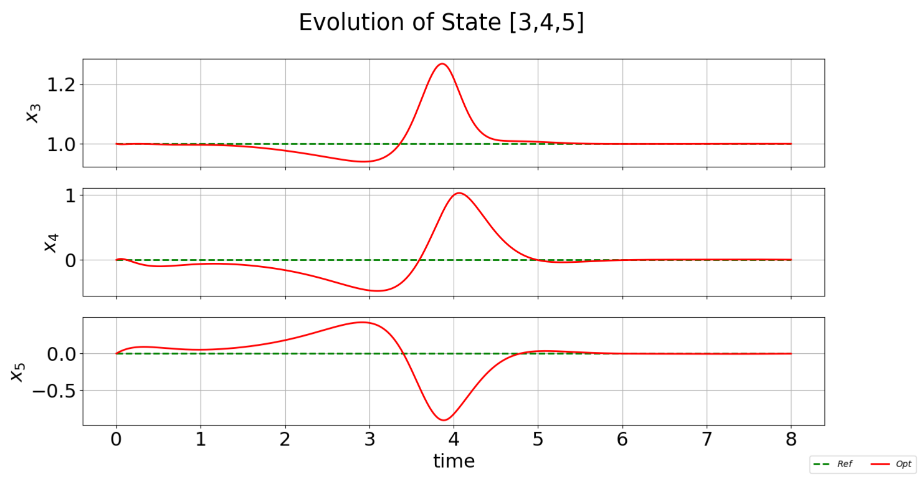

Figure 2.5: Plot of the $\{X_3, X_4, X_5\}$ reference and optimal states

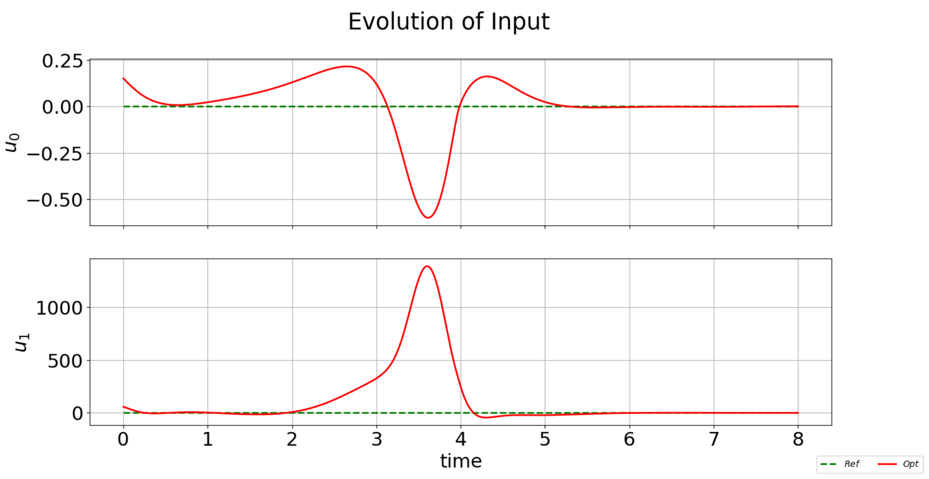

Figure 2.6: Plot of the reference and optimal inputs

Chapter 3

Task 2 - Skidpad

In order to accomplish this task we have created the files `Task_1-2.py` . To launch this task, you have to impose the *Curve* variable equal to '*Skidpad*' in "TO DO section".
You can also set the flags:

- *print_armijo*: to print in the terminal the Armijo's stepsize for each iteration.
- *visu_armijo*: to plot the Armijo graph at each iteration.
- *Save_output_opt*: to save the optimal trajectory in a `txt` file.

In the second task we are asked to make the vehicle to follow an 8-shaped track. Firstly we had to define a simple trajectory to perform this maneuver and than exploit the previously defined Newton's algorithm for optimal control to compute the optimal trajectory.

3.1 Skidpad reference curve definition

The reference curve has been defined in the function *Skidpad_Curve* in file `Reference_curve.py`

The curve in x and y directions has been defined with the following parametrization of the circumference:

$$\begin{cases} x = Rcos(\theta) + X_c \\ y = Rsin(\theta) + Y_c \end{cases} \tag{3.1}$$

The radius has been chosen as half the mean of the the inner and outer diameters. Thus: $R = \frac{1}{2} \cdot \frac{(21.25+15.25)}{2} = 9.125m$
In order to make the vehicle to follow the curve in the asked direction, the

first circumference (the one on the right) has been computed with $\theta \in [\pi, -\pi]$, while the second with $\theta \in [0, 2\pi]$.

The yaw angle ψ is equal to the circumference parameter θ shifted by an angle θ_0 which is equal to $\frac{\pi}{2}$ for the first circle and to $-\frac{3}{2}\pi$ for the second one. Hence:

$$\psi = \theta(t) + \theta_0 \tag{3.2}$$

The velocity V_x has been chosen to be the instantaneous tangential velocity of a circular motion. Thus:

$$V_x = R\frac{\Delta\theta}{\Delta t} \tag{3.3}$$

Where $\Delta\theta$ is the difference between the precedent θ and the actual one. The velocity V_y has been fixed to 0, while the yaw rate is:

$$\dot{\psi} = \frac{\Delta\theta}{\Delta t} \tag{3.4}$$

To summarize:

$$\begin{cases} x_{ref} = Rcos\theta(t) + X_c \\ y_{ref} = Rsin\theta(t) + Y_c \\ \psi_{ref} = \theta(t) + \theta_0 \\ V_{xref} = R\frac{\Delta\theta}{\Delta t} \\ V_y = 0 \\ \dot{\psi} = \frac{\Delta\theta}{\Delta t} \end{cases} \tag{3.5}$$

The initial state has been configured to have the vehicle facing toward the positive y direction. Thus:

$$\begin{cases} x_0 = 0 \\ y_0 = 0 \\ \psi_0 = \frac{\pi}{2} \\ V_{x0} = R\frac{\Delta\theta}{\Delta t} \\ V_{y0} = 0 \\ \dot{\psi}_0 = \frac{\Delta\theta}{\Delta t} \end{cases} \tag{3.6}$$

The plots of the designed reference curve are:

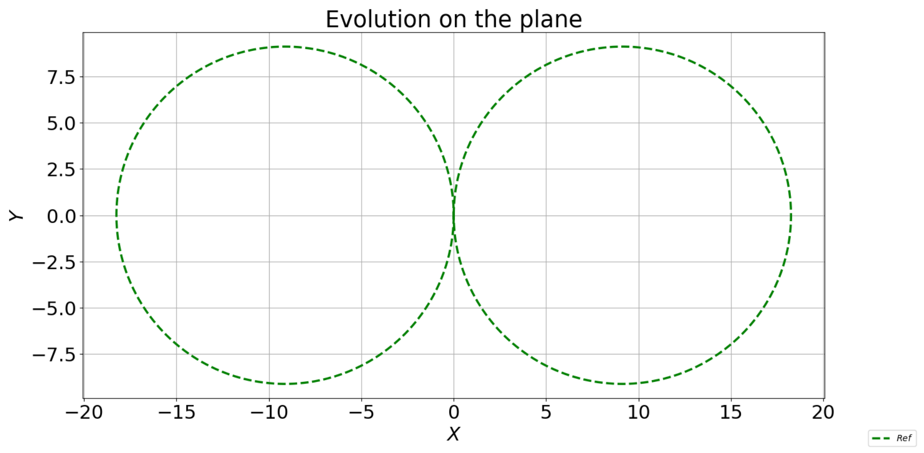

Figure 3.1: Plot in x and y

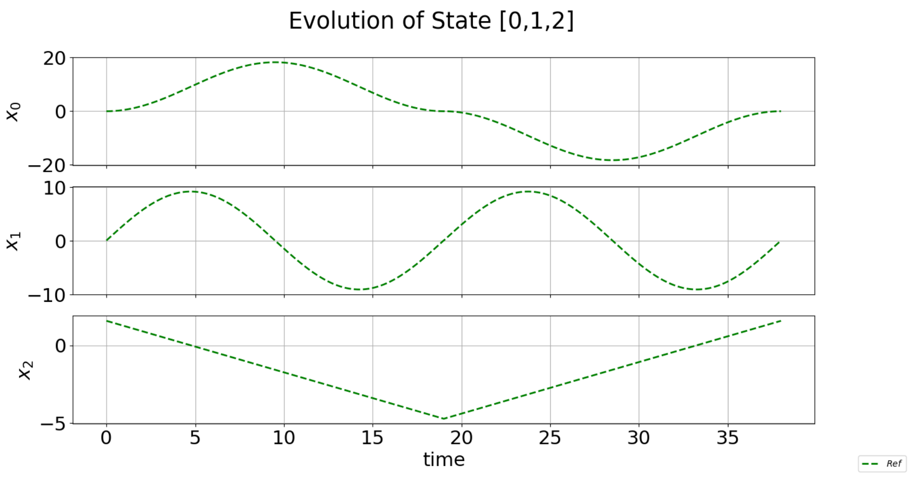

Figure 3.2: Plot of the $\{X_0, X_1, X_2\}$ reference curve's states

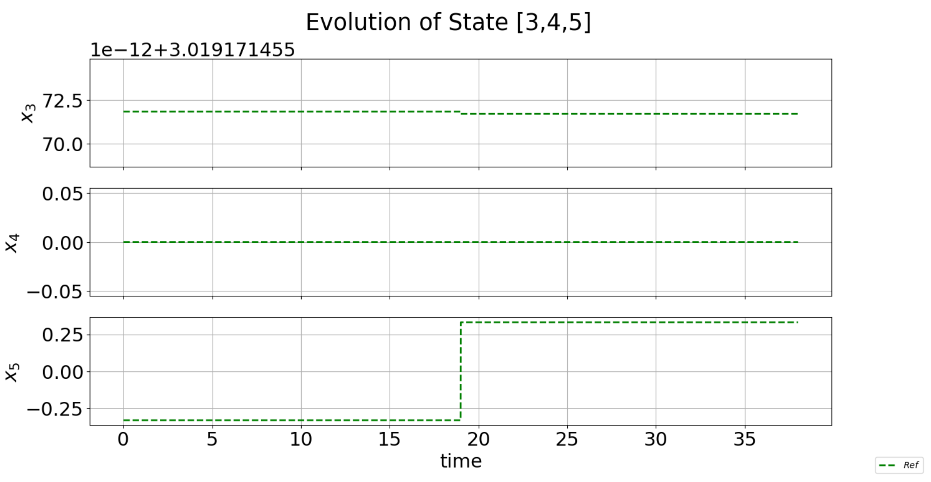

Figure 3.3: Plot of the $\{X_3, X_4, X_5\}$ reference curve's states

3.2 Initial trajectory - P regulator

The initial trajectory has been defined in the function *PID_Trj_2* in file `Initial_Trajectory.py`
In order to give to the Newton's method as initial states and inputs a real trajectory, we have designed a proportional regulator with the following equations:

$$\delta(t) = K_\delta \cdot (\psi_{ref}(t) - \psi(t)) \tag{3.7}$$

$$e(t) = \sqrt{(x_{0_{ref}}(t) - x_0(t))^2 + (x_{1_{ref}}(t) - x_1(t))^2} \tag{3.8}$$

$$F_x(t) = K_F \cdot (V_{x_{ref}}(t) - V_x(t)) \tag{3.9}$$

with $K_\delta = K_F/100$.

The 3.7 steering input aims to compensate the difference in the current yaw angle (which will turn out to be constant in time). The 3.9 force controller is a proportional regulator respect the error in position 3.8.
The designed inputs are then fed into the vehicle dynamics, in order to generate the trajectory. As we can see in the plots, the P regulators are able to produce a trajectory quite similar to the required skidpad.

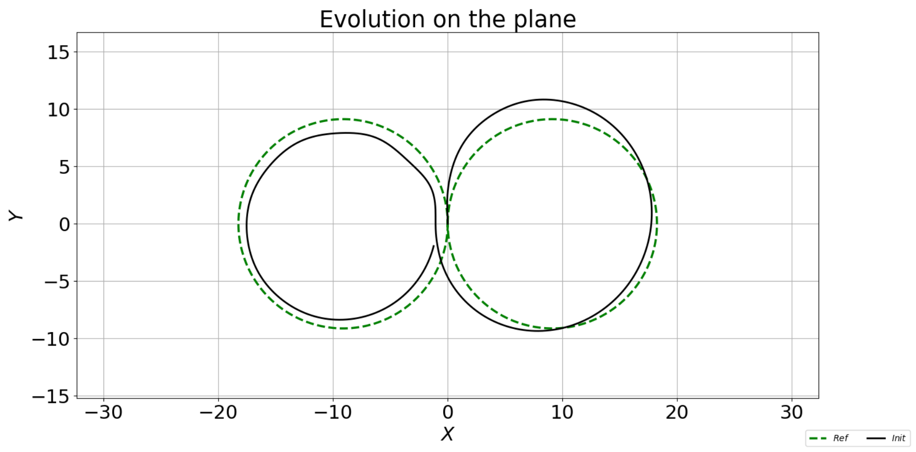

Figure 3.4: Plot in x and y of the initial trajectory

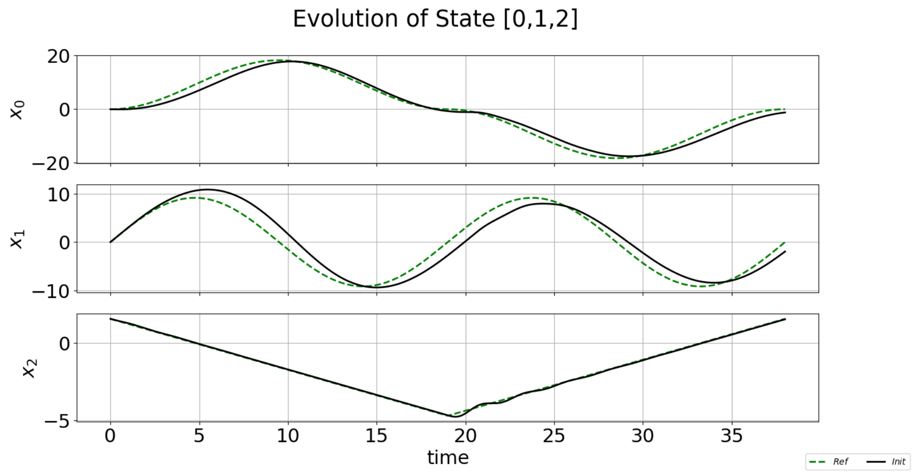

Figure 3.5: Plot of the $\{X_0, X_1, X_2\}$ reference and initial states

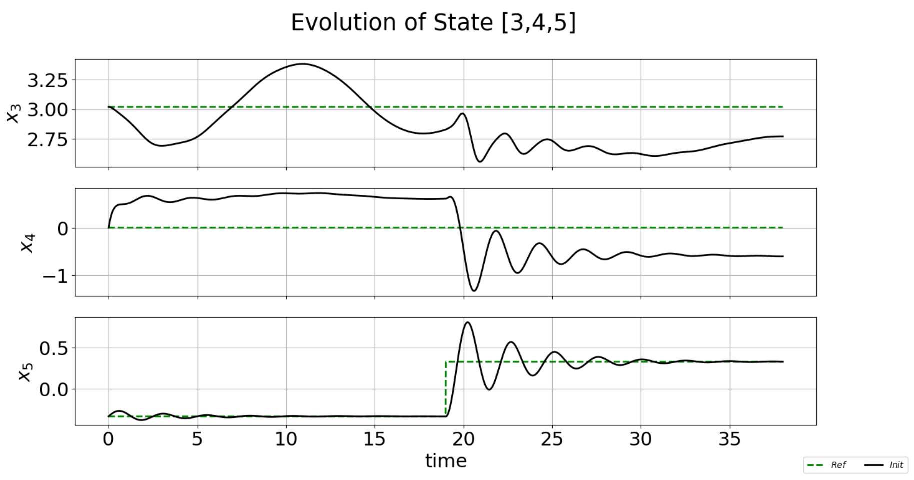

Figure 3.6: Plot of the $\{X_3, X_4, X_5\}$ reference and initial states

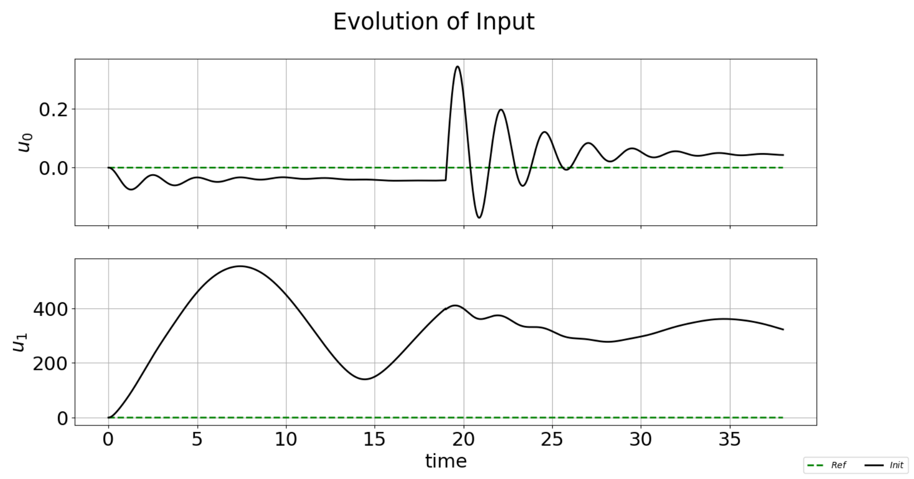

Figure 3.7: Plot of the inputs of the initial trajectory

Since it is a proportional regulator we see high oscillations in the inputs.

3.3 Newton's method

The chosen weights matrices are:

$$Q_t = \begin{bmatrix} 10^3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10^3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

$$R_t = \begin{bmatrix} 10^3 & 0 \\ 0 & 10^{-4} \end{bmatrix} \tag{3.11}$$

$$Q_T = \begin{bmatrix} 10^4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10^4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

Initially, we had several issues in tuning the weights, especially for inputs. So we have considered the difference in the order of magnitude of the two inputs given by the unit of measure. Then, we have computed the weights in order to make the weighted summation, as unitary as possible. While for the state weights, we have given more importance to the first 3 states, since our goal was to track the skidpad.

We have again checked if the Armijo rule implementation was correct, by examination of the Armijo's plots. As it is shown in the plot 3.8 the gradient of the cost function evaluated in $\gamma = 0$ is always tangent to the cost function itself, prooving the correctness of the algorithm.

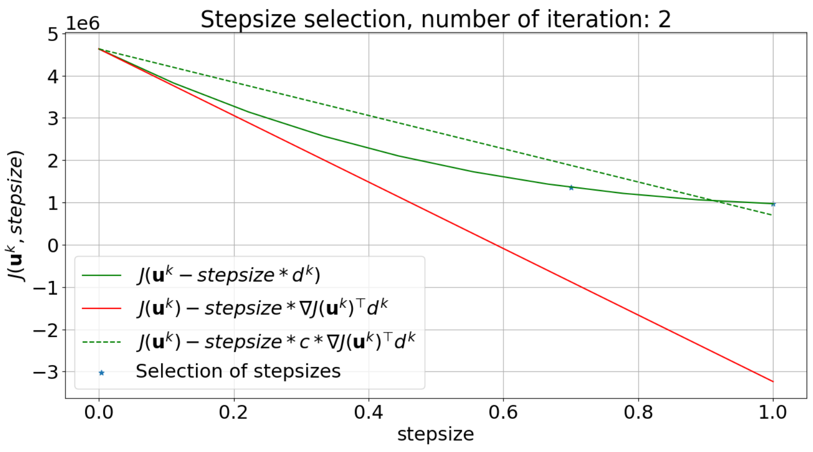

Figure 3.8: Armijo's plots

The Newton's method converges in 12 steps, tracking correctly the skid-pad. We have saved the trajectory in a `txt` file, in order to perform the next tasks.
In the following pages, we have printed out several plots related to the algorithm results:

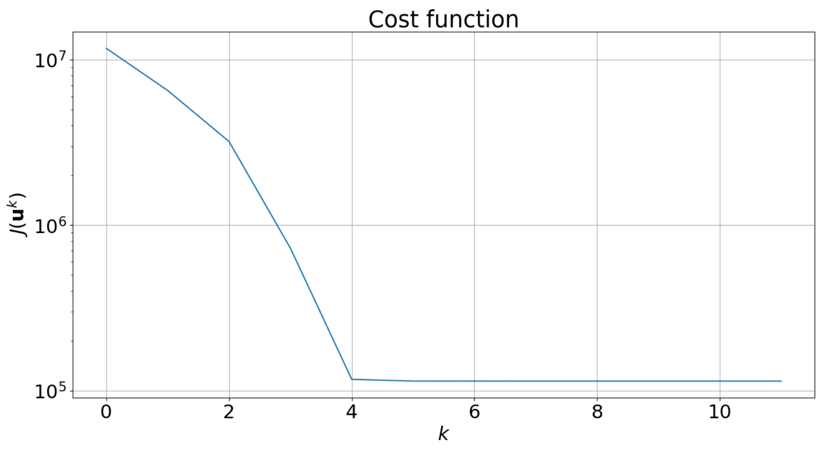

Figure 3.9: Plot of the cost function evolution

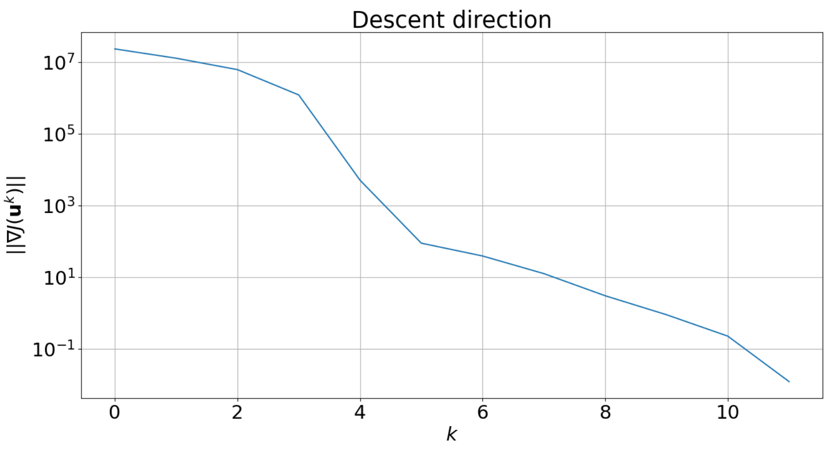

Figure 3.10: Plot of the descent direction evolution

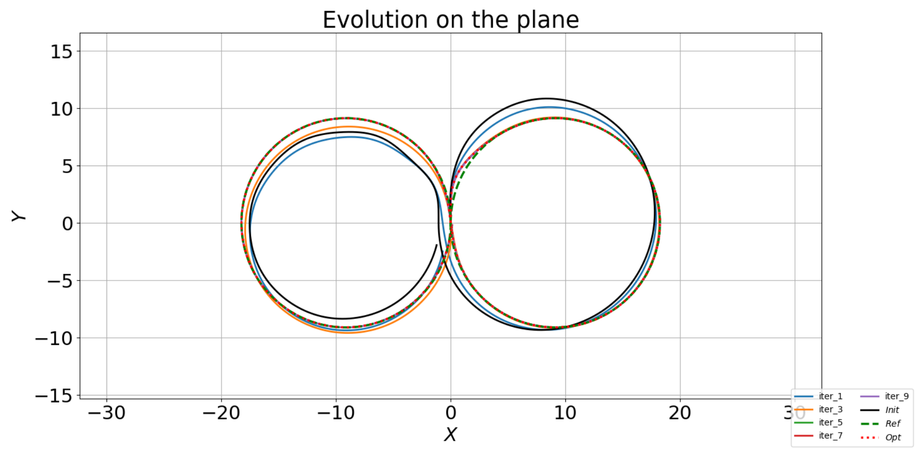

Figure 3.11: Evolution of the trajectory in different iterations

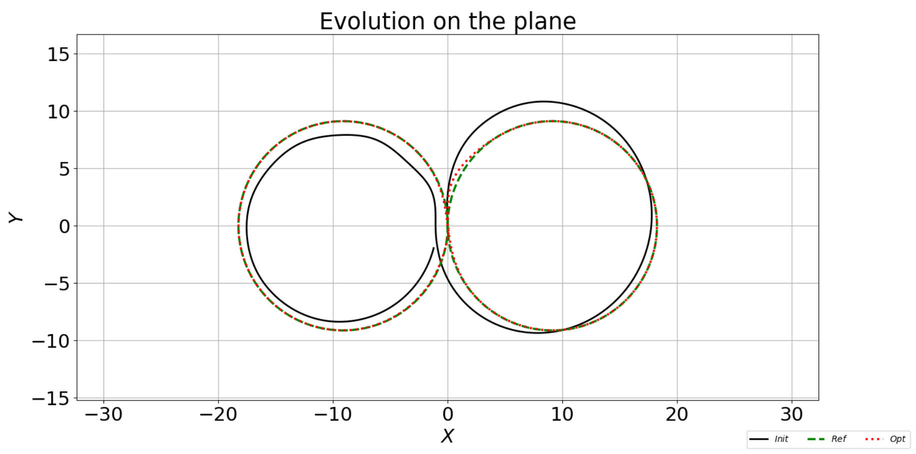

Figure 3.12: Evolution of the optimal trajectory

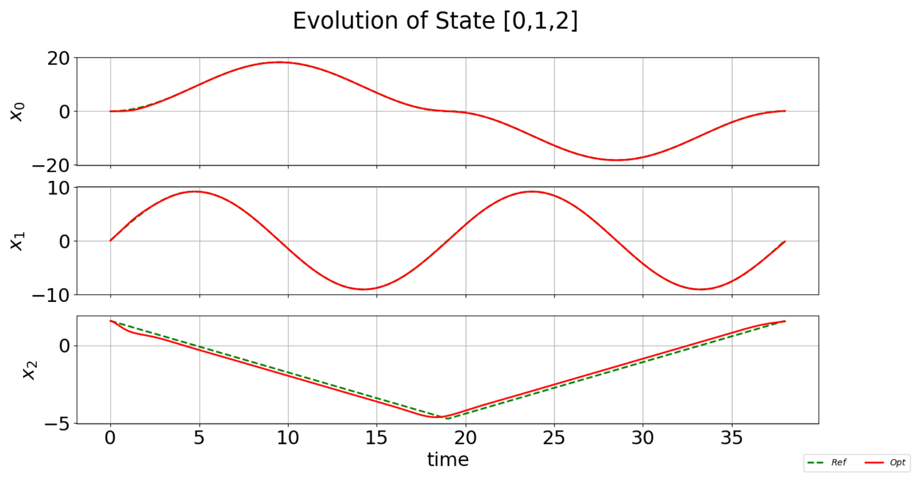

Figure 3.13: Plot of the $\{X_0, X_1, X_2\}$ reference, initial and optimal states

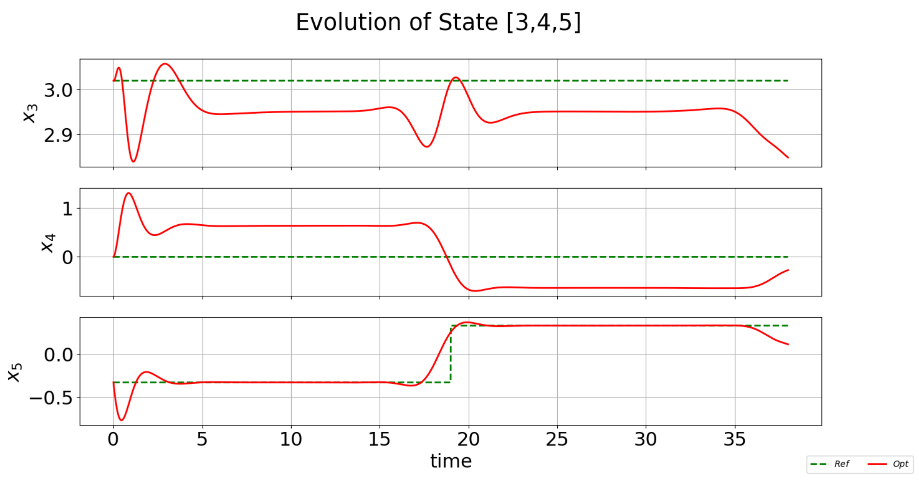

Figure 3.14: Plot of the $\{X_3, X_4, X_5\}$ reference, initial and optimal states

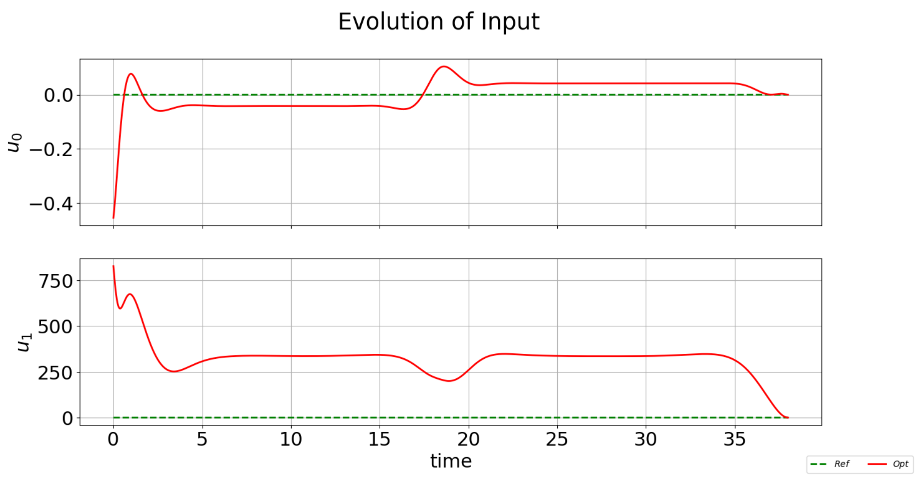

Figure 3.15: Plot of the reference, initial and optimal inputs

We can notice that the optimal control is much better than the proportional regulator.

Chapter 4

Task 3 - Trajectory Tracking

In order to accomplish this task we have created the files `Task_3-4.py` . To launch this task, you have to impose the *Trajectory* variable equal to '*Skidpad*' in "TO DO section". The program will automatically import the optimal trajectory generated in `Task_1-2.py` by loading the `txt` file.

In this task we are asked to linearize the vehicle dynamics about the optimal trajectory (**x***, **u***) computed in Task 2 and to exploit the LQR algorithm to define the optimal feedback controller to track this reference trajectory.

4.1 Linearization of the dynamics

We have approximated the dynamics about the optimal trajectory via the linear time-varying system:

$$\Delta x_{t+1} = A_t^{opt} \Delta x_t + B_t^{opt} \Delta u_t \tag{4.1}$$

where matrices A_t^{opt} and B_t^{opt} are:

$$A_t^{opt} := \nabla_{x_t} f(x_t^{opt}, u_t^{opt})^T \tag{4.2}$$

$$B_t^{opt} := \nabla_{u_t} f(x_t^{opt}, u_t^{opt})^T \tag{4.3}$$

This has been done for each time instant of the trajectory.

4.2 Calculation of the LQ optimal controller

Inside the function *ltv_LQP()* in the file `solver_LQ.py` we have solved the following LQ optimal control problem:

$$\begin{aligned} \min_{\Delta x_t, \Delta u_t} \quad & \sum_{t=0}^{T-1} \Delta x_t^T Q_t^{reg} \Delta x_t + \Delta u_t^T R_t^{reg} \Delta u_t + \Delta x_T^T Q_T^{reg} \Delta x_T \\ \text{subject to} \quad & \Delta x_{t+1} = A_t^{opt} \Delta x_t + B_t^{opt} \Delta u_t \qquad t = 0, ..., T-1 \\ & \Delta_{x_0} = 0 \end{aligned} \tag{4.4}$$

with the constant cost matrices $Q_t^{reg} \geq 0$ and $R_t^{reg} > 0$ that can be tuned in order to withstand some disturbances, as we have done later.

We have set $P_T = Q_T^{reg}$ and backward iterate $t = T-1, ..., 0$ to solve the Riccati equation:

$$\begin{aligned} P_t = {} & Q_t^{reg} + A_t^{opt,T} P_{t+1} A_t^{opt} - (A_t^{opt,T} P_{t+1} B_t^{opt})(R_t^{reg} + B_t^{opt,T} P_{t+1} B_t^{opt})^{-1} \cdot \\ & (B_t^{opt,T} P_{t+1} A_t^{opt}) \end{aligned} \tag{4.5}$$

Finally, for all $t = 0, ..., T-1$ we have computed the feedback gain K_t^{reg}

$$K_t^{reg} = -(R_t^{reg} + B_t^{opt,T} P_{t+1} B_t^{opt})^{-1} (B_t^{opt,T} P_{t+1} A_t^{opt}) \tag{4.6}$$

4.3 Tracking of the Optimal Trajectory

We have applied the feedback controller on the linearization of the non linear system in order to track the computed optimal trajectory. Hence, for all $t = 0, ..., T-1$ we have computed:

$$u_t^{reg} = u_t^{opt} + K_t^{reg}(x_t^{reg} - x_t^{opt}) \tag{4.7}$$

$$x_{t+1}^{reg} = f(x_t^{reg}, u_t^{reg}) \tag{4.8}$$

with x_0 equal to the initial state of the optimal trajectory

4.4 Trials with some disturbances

In order to test the effectiveness of our tracking, we have imposed some disturbances to the state.

In particular, we have applied a disturbance on the initial state of:

$$\begin{cases} x^{dist}(0) = +5 \\ y^{dist}(0) = -1 \\ \psi^{dist}(0) = 0 \\ V^{dist}(0) = 0 \\ V^{dist}(0) = 0 \\ \dot{\psi}^{dist}(0) = 0 \end{cases} \tag{4.9}$$

Furthermore, we have set a disturbance at time $t = \frac{3}{4}T$ of:

$$\begin{cases} x^{dist}(0) = 1 \\ y^{dist}(0) = -1 \\ \psi^{dist}(0) = 0 \\ V^{dist}(0) = 0 \\ V^{dist}(0) = 0 \\ \dot{\psi}^{dist}(0) = 0 \end{cases} \tag{4.10}$$

By tuning properly the weight matrices, despite the imposition of the disturbances, our LQR algorithm has been able to follow properly the skidpad.

In the following plots we can clearly distinguish the imposed disturbances. We can also see that the algorithm works successfully.

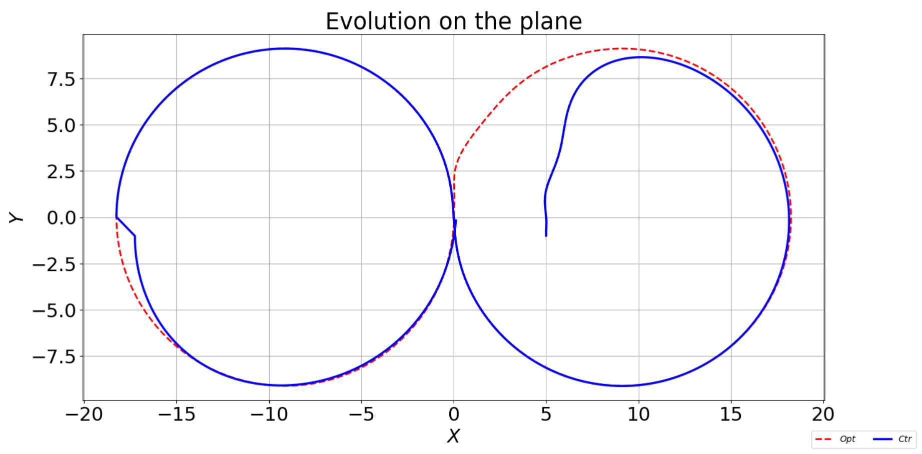

Figure 4.1: Evolution of the optimal trajectory and optimal feedback controller with disturbances

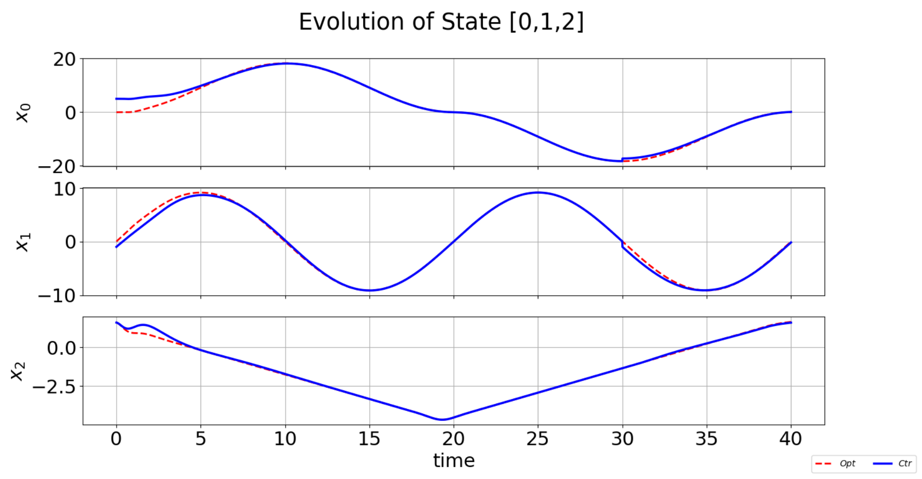

Figure 4.2: Plot of the $\{X_0, X_1, X_2\}$ optimal and controller states

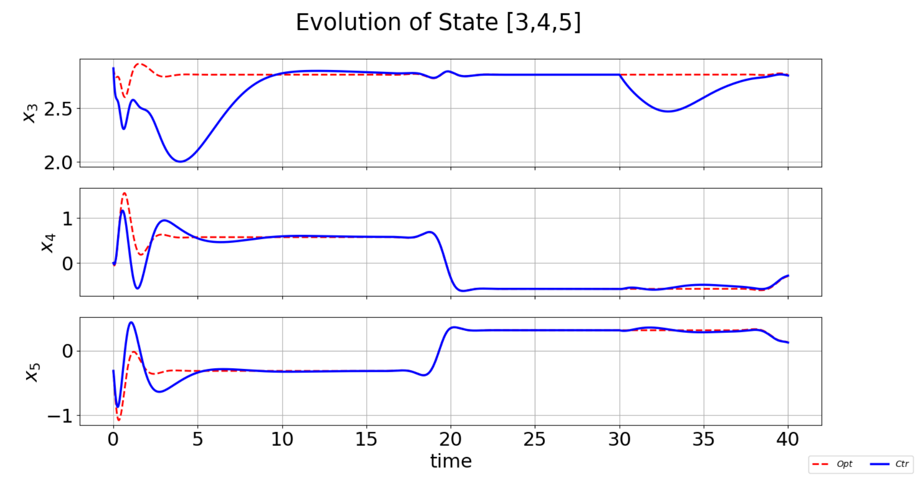

Figure 4.3: Plot of the $\{X_3, X_4, X_5\}$ optimal and controller states

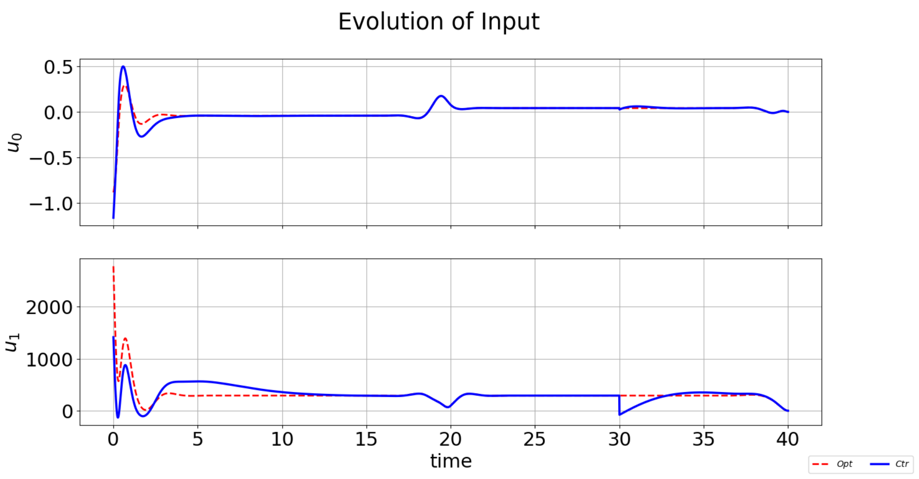

Figure 4.4: Plot of the optimal and controller inputs

Chapter 5

Task 4 - Animation

As required, we have implemented a simple animation, that will automatically starts at the end of `Task_3-4.py` execution.
We plots the original optimal trajectory without disturbance and we have animated the perturbed tracked trajectory.
In particular we have animated the chassis and the wheels of the vehicle.

Below you can find a frame of the animation along the skidpad curve.

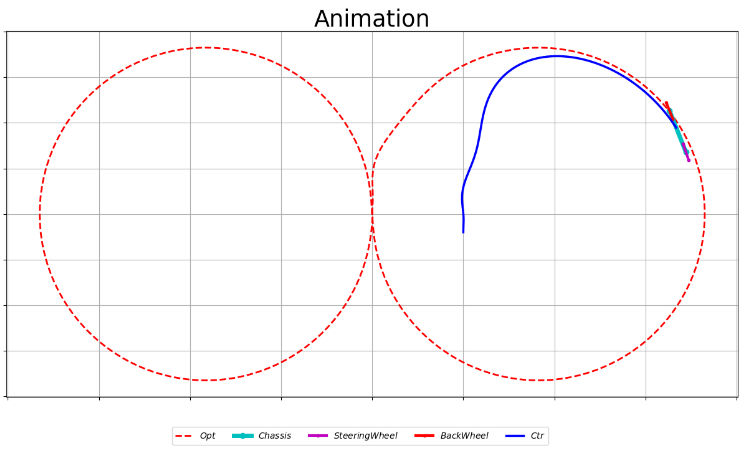

Figure 5.1: Animation's frame of skidpad curve

In the following figure is represented also a frame of the animation along the step curve, with different disturbances on the initial state and in the middle of lane change.

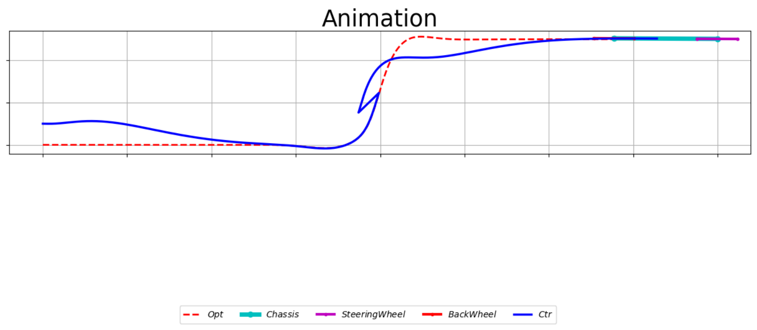

Figure 5.2: Animation's frame of step curve

Chapter 6

Task 5 - Physical limits

In order to accomplish this task we have created the files `Task_1-2-5.py` . To launch this task, you have to impose the *Curve* variable equal to '*Step*' or '*Skidpad*' in "TO DO section".
You can also set the flags:

- *print_armijo*: to print in the terminal the Armijo's stepsize for each iteration.
- *visu_armijo*: to plot the Armijo graph at each iteration.
- *Save_output_opt*: to save the optimal trajectory in a `txt` file.

The goal of this task was to apply physical constraints to the control variables, by inserting a saturation value U_{limit}. In order to do that, we have implemented the theoretical concept of a **barrier function** , which is necessary to impose an inequality constraint in the optimal control.

6.1 Barrier function

First of all, we have modified the file `cost.py` by integrating the new function *stagecost_barrier_function()* which has two new input variables U_{limit} and *Epsilon*.
Within the function, we have imposed the inequality constraint as:

$$\left(\frac{u_t^k}{U_{limit}}\right)^2 \leq 1 \tag{6.1}$$

so that it is equally lower and upper bounded.
Therefore, the input argument to the barrier function is:

$$\left(\frac{u_t^k}{U_{limit}}\right)^2 - 1 \leq 0 \tag{6.2}$$

As indicator function, we have used the barrier function represented by $-log_2$. This because it is the best representation of the theoretical step function.

We had also dealt with the problem of the logarithm argument < 0, by imposing it to be equal to 10^{-20} in the case it is lower than 10^{-20}. In this way, the function is able to work always in its feasible domain.

Therefore, we have applied the barrier function to both inputs u_0 and u_1, by summing it to the standard cost function.

6.2 Epsilon

The Epsilon coefficient is imposed as a multiplier for every barrier function. Theoretically speaking, it is needed to cancel out the effects of the barrier function at the n_th iteration. Thus leaving the algorithm at the last iterations with an unaffected cost function. In order to do that, the epsilon value is initialized equal to one and then decremented by an order of magnitude at each iteration k .

However, this implementation has not proved to be feasible, since it required the execution of the new cost function's derivatives in order to compute the descent direction. In fact, the derivative of the logarithm has caused us numerical problems.

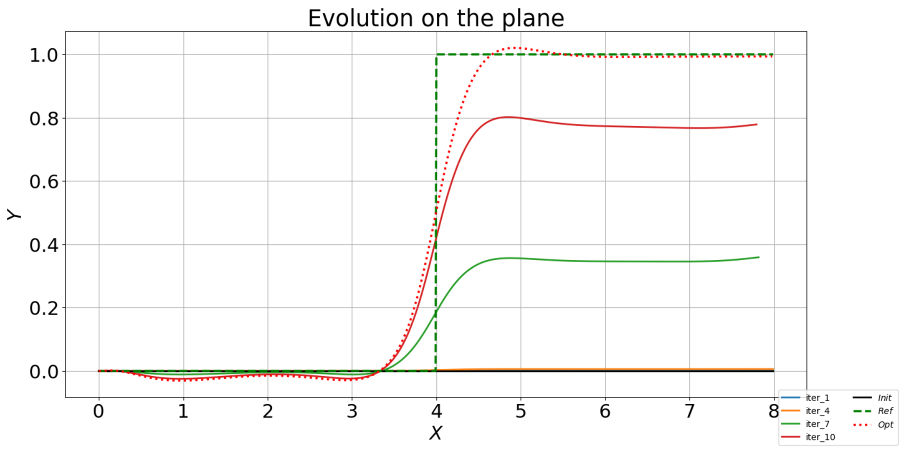

Figure 6.1: Evolution on the plane with a decremental epsilon

As we can notice from the plots, as long as the value of epsilon is substantial, there aren't relevant improvement of the optimal trajectory. Only when epsilon attains a negligible value, we come back to execute the standard Newton's method, which gives us a result without saturation constraints.

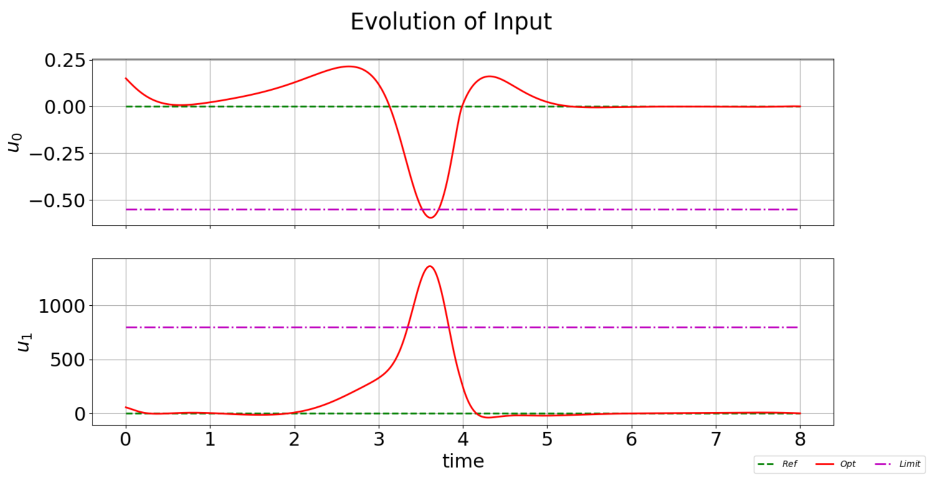

Figure 6.2: Plot of the inputs with a decremental epsilon

The inputs violate the saturation constraints.

6.2.1 Constant Epsilon

Therefore we have chosen to set a constant value for Epsilon equal to 1000, to compensate the high weights of the states in the cost function. In this way, the barrier function works correctly and the saturation constraints are attained.
As we expect, this cause a degradation of the performance in the optimal trajectory.

6.3 Results

The saturation values has been set to $F_{x_{limit}} = 800N$ (in task 1, we have got a peak value of about 1300N).

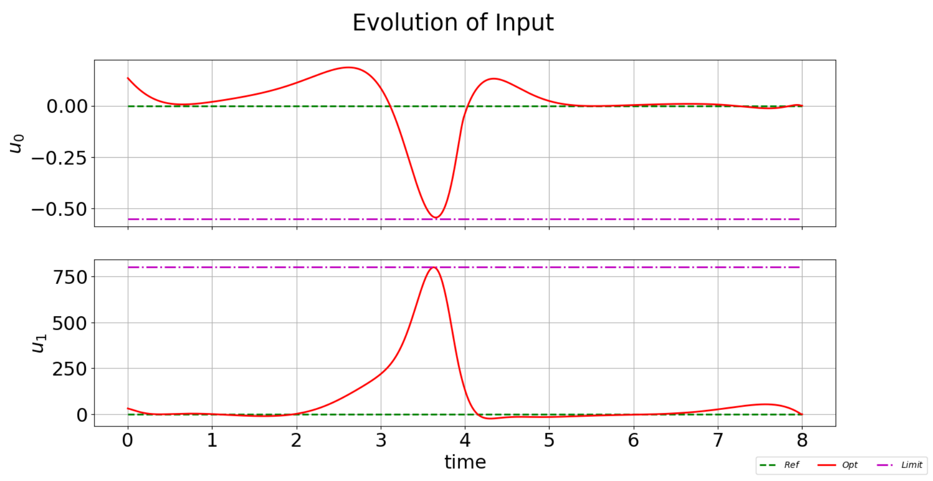

Figure 6.3: Plot of the inputs of a step curve with a constant epsilon

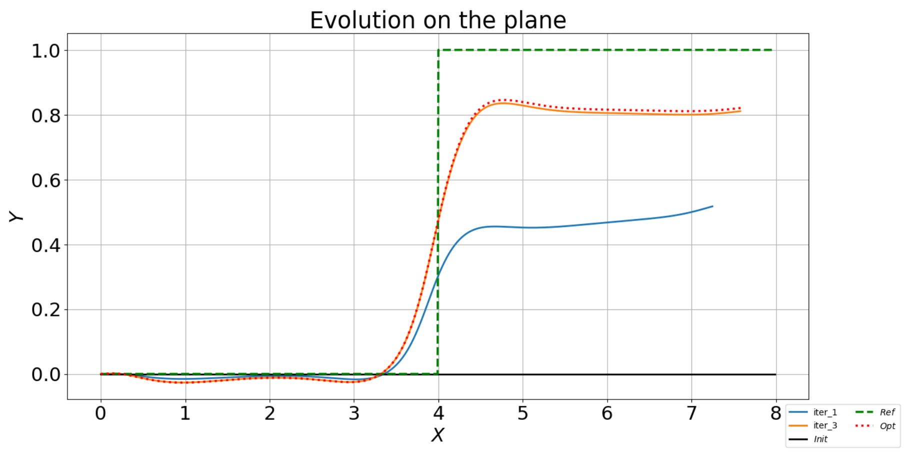

Figure 6.4: Evolution of the plane of a step

The saturation values has been set to $F_{x_{limit}} = 600N$ (in task 1, we have got a peak value of about 750N).

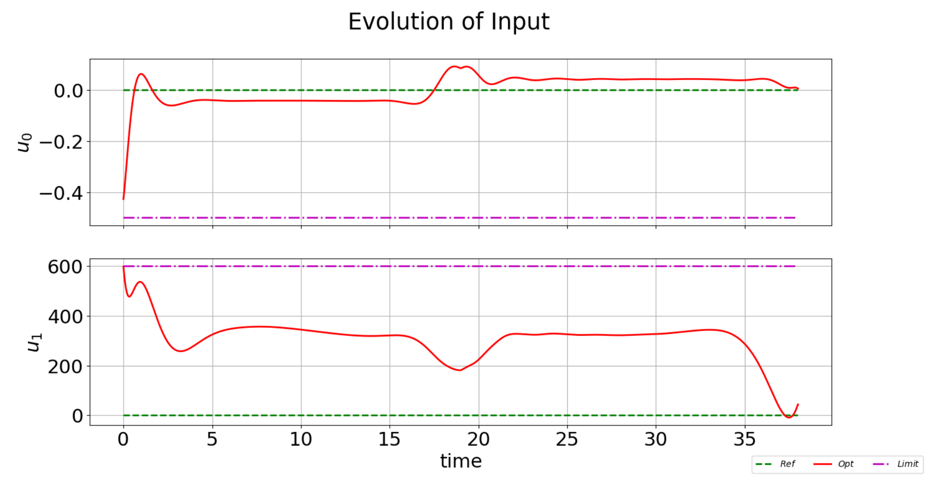

Figure 6.5: Plot of the inputs of a skidpad curve with a constant epsilon

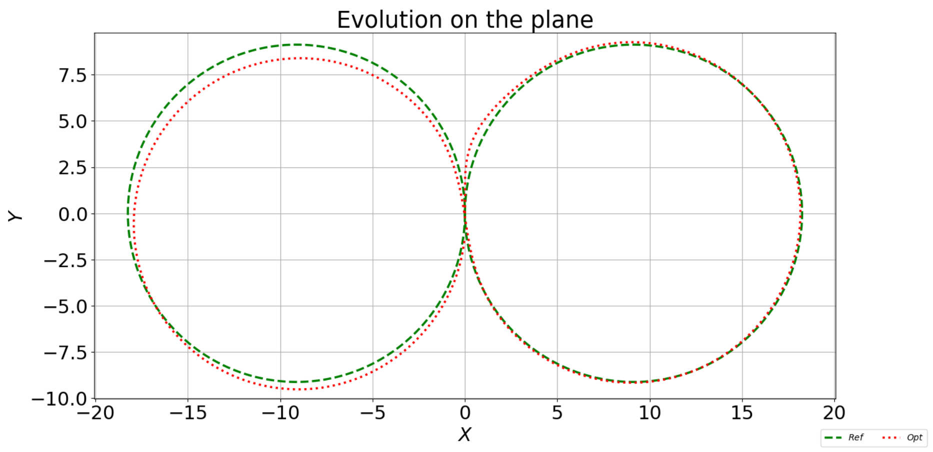

Figure 6.6: Evolution on the plane of a skidpad

Regarding the stopping criterion, we have modified it to be effective when the difference of the descent direction with respect to the previous iteration is below a certain threshold.

Conclusions

To conclude the report, we want to highlight some points of improvement:

- As we can notice, in both executions of step and skidpad, there are small errors with respect to the reference curve, especially for the first instants of time.

- We have noticed that with state x and y weights equal to 1, we get a perfect trajectory but shifted in position.
  We have tried to reduce the discretization step, without any variation in the algorithm's result.
  Then, we have tried to modify x_0, without gaining any improvements.
  Finally, we have implemented a new regulator, going from a proportional regulator in velocity to a proportional regulator in position. This got us an improvement in algorithm speed, but not in the trajectory results. Hence it can be an improvement point of the code.

- Another point of improvement could be the Epsilon parameter inside the physical constraints' task. As we have highlighted, we have set it as a constant value, on the opposite of what it is reported in the theory. As an improvement, we can implement the right derivatives of the cost function and analyze their effects on the descent direction

To conclude, we report that we have implemented the step reference also for tasks 3-4-5.
We have also implemented a sigmoid reference curve for the task 1, which works correctly.