UNIVERSITY OF BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Optimal Control

OPTIMAL CONTROL OF A VEHICLE

Professor: **Giuseppe Notarstefano**

Students: **Daniele Simonazzi**
Nicoló Musolesi
Giulia Cutini

Academic year 2022/2023

Abstract

In this report we are going to discuss how we managed to develop an optimal control strategy for a simplified car-like vehicle, which has to accomplish two different maneuvers: a lane change and the tracking of an 8-shaped trajectory.

Each chapter is organized in a similar fashion: first we introduce the framework of the task we have to fulfill, then we describe the algorithms, and finally we discuss the results.

Contents

Chapter 1

Problem setup

We start from the following set of equations, which describe the dynamics of the car-like vehicle schematized in figure 1.1:

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

The state space is defined as:

$$\mathbf{x} = [x\ y\ \psi\ V_x\ V_y\ \dot{\psi}]^T \tag{1.2}$$

where $(x\ y\ \psi)$ are the cartesian coordinates of the centre of mass of the vehicle and its yaw in a global reference frame, while $(V_x\ V_y\ \dot{\psi})$ are the vehicle velocities and yaw rate in the vehicle's body fixed reference frame.

The control inputs are:

$$\mathbf{u} = [\delta\ F_x]^T \tag{1.3}$$

where δ is the steering angle of the vehicle and F_x is the force applied by the front wheel.

The terms F_y represent the lateral forces and can be found as:

$$\begin{aligned} F_{y,f} &= \mu F_{z,f} \beta_f \\ F_{y,r} &= \mu F_{z,r} \beta_r \end{aligned} \tag{1.4}$$

where β_f are β_r are the front and rear slip angles:

$$\begin{aligned} \beta_f &= \delta - \frac{V_y + a\dot{\psi}}{V_x} \\ \beta_r &= -\frac{V_y - b\dot{\psi}}{V_x}. \end{aligned} \tag{1.5}$$

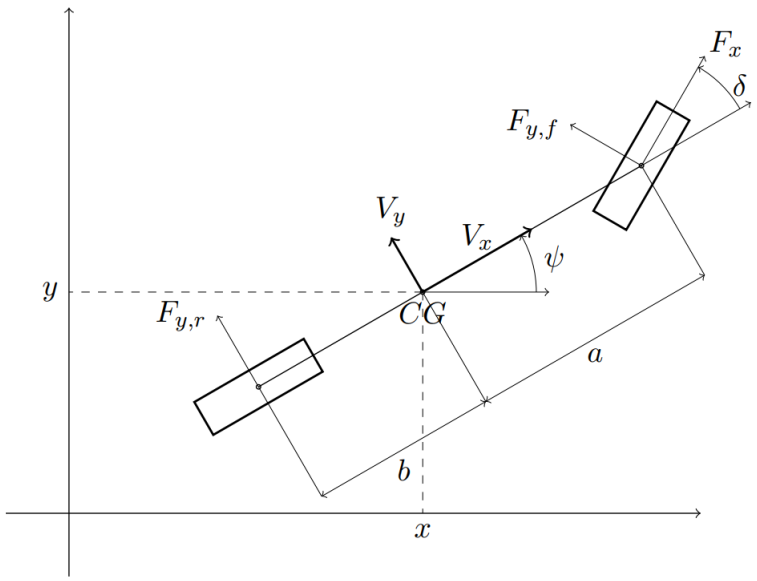

Figure 1.1: Bicycle model of the car-like vehicle

The vertical forces on the front and the rear wheel can be found as:

$$\begin{aligned} F_{z,f} &= \frac{mgb}{a+b} \\ F_{z,r} &= \frac{mga}{a+b}. \end{aligned} \tag{1.6}$$

1.1 Discretization of the dynamics

In order to discretize the continuous-time dynamics we used the forward Euler method:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + dt \cdot f_{CT}(\mathbf{x}_t, \mathbf{u}_t) \tag{1.7}$$

where dt is the discrete time step used to sample the continuous time dynamics, $(\mathbf{x}_t, \mathbf{u}_t)$ are the state vector and the input vector at current time instant and $\mathbf{x}_{t+1}$ is the state vector at the next time instant.

In compact notation, we can also write:

$$\mathbf{x}_{t+1} = \mathbf{f}_t(\mathbf{x}_t, \mathbf{u}_t) \tag{1.8}$$

Given $\mathbf{x}_t$ and $\mathbf{u}_t$, the computation of $\mathbf{x}_{t+1}$ is carried out by the `dynamics` function inside the Python script `dynamics.py`.

Since we will need to linearize the dynamics equations around specific operational points (see next tasks) and we will need to implement the Newton's method algorithm, `dynamics` also returns:

- the Jacobian of function $\mathbf{f}$ at time t, splitted into matrices A_t and B_t.

$$A_t^T = \frac{\partial \mathbf{f}(\cdot)}{\partial \mathbf{x}}|_{\substack{\mathbf{x}=\mathbf{x}_t \\ \mathbf{u}=\mathbf{u}_t}}, \qquad B_t^T = \frac{\partial \mathbf{f}(\cdot)}{\partial \mathbf{u}}|_{\substack{\mathbf{x}=\mathbf{x}_t \\ \mathbf{u}=\mathbf{u}_t}} \tag{1.9}$$

- the Hessian-terms needed in the implementation of the non-regularized version of the Newton's method

All the derivatives have been computed by using the symbolic toolbox `Python SymPy`.

Chapter 2

Lane change maneuver

The first task consists in making the car perform a lane change maneuver as shown in figure 2.1.

The framework we chose is the following: a car running on an highway with constant forward velocity of $20\frac{m}{s}$ has to perform a lateral shift of $4m$ within a time horizon of $5s$.

It is recommended to use the **Newton's method algorithm** in order to find the optimal trajectory between initial and final states, both characterized by a null yaw angle.

Initial and final states are identical in the sense that they both represent the car running forward with constant velocity.

2.1 Reference and Initial-Guess trajectories

In order to properly define the cost function we choose a step as **reference curve**:

- $4m$-jump along the y-axis
- a constant velocity along the x direction
- zero inputs (we deduced from the equations that the force necessary to keep the car moving forward at constant velocity is zero)

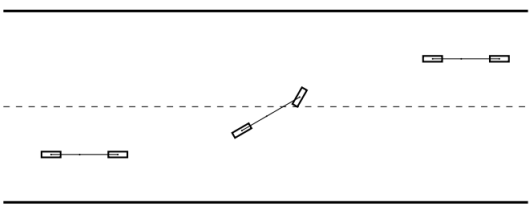

Figure 2.1: Lane change maneuver

Such reference curve represents the optimum since the maneuver takes place instantaneously and with smallest inputs.

It's easy to see that the step is not a feasible trajectory, yet we do not care since feasibility is not a requirement for the reference trajectory: it should just possess the good features we want the real trajectory to tend to.

Once the reference has been defined it's time to decide where to initialize the algorithm.

We know that the Newton's method algorithm is quite sensitive to initialization errors, thus, in order to obtain a meaningful **initial guess trajectory**, we did the following:

- we defined a sigmoid function in the $(x,\ y)$ plane with $x = V_0 \cdot t$ and, adopting the same discretization step dt we used for the dynamics equations, we sampled the components of the state vector (`utils sigmoid.py` file):
$$[x\ y\ \psi\ V_x\ V_y\ \dot{\psi}] \tag{2.1}$$
The choice of the sigmoid function as mathematical representation of the lane change maneuver was driven by the need to have a smooth transition between initial and final states.

- at each sampled time instant, given the velocity-state of the car (obtained from the sigmoid), we used a root-finding function (`fsolve`[1]) to compute the pair of inputs u_{0_t}. These inputs are called quasi-static since they succeed in keeping the velocity-state of the car constant.

- we plugged the sequence of quasi-static inputs $\mathbf{u}_0$ into the dynamics to get the real state-trajectory performed by the car, $\mathbf{x}_0$ [2] (`test dynamics.py` file).

The system trajectory $(\mathbf{x}_0,\ \mathbf{u}_0)$ represents the meaningful initial guess we were looking for.

In figure 2.2 we can see both the initial-guess and reference trajectory.

[1] `fsolve` has been applied to the third and fourth dynamics equations, we neglected the fifth equation in order to avoid a potential failure of the root-finding method (3 equations in 2 unknowns)

[2] For a meaningful computation of the cost at iteration 0, $\mathbf{x}_0$ must be a feasible state trajectory of the system. This is why we did not use the quasi static states to initialize the Newton's method, instead we computed a real trajectory by feeding the quasi-static inputs to the dynamics.

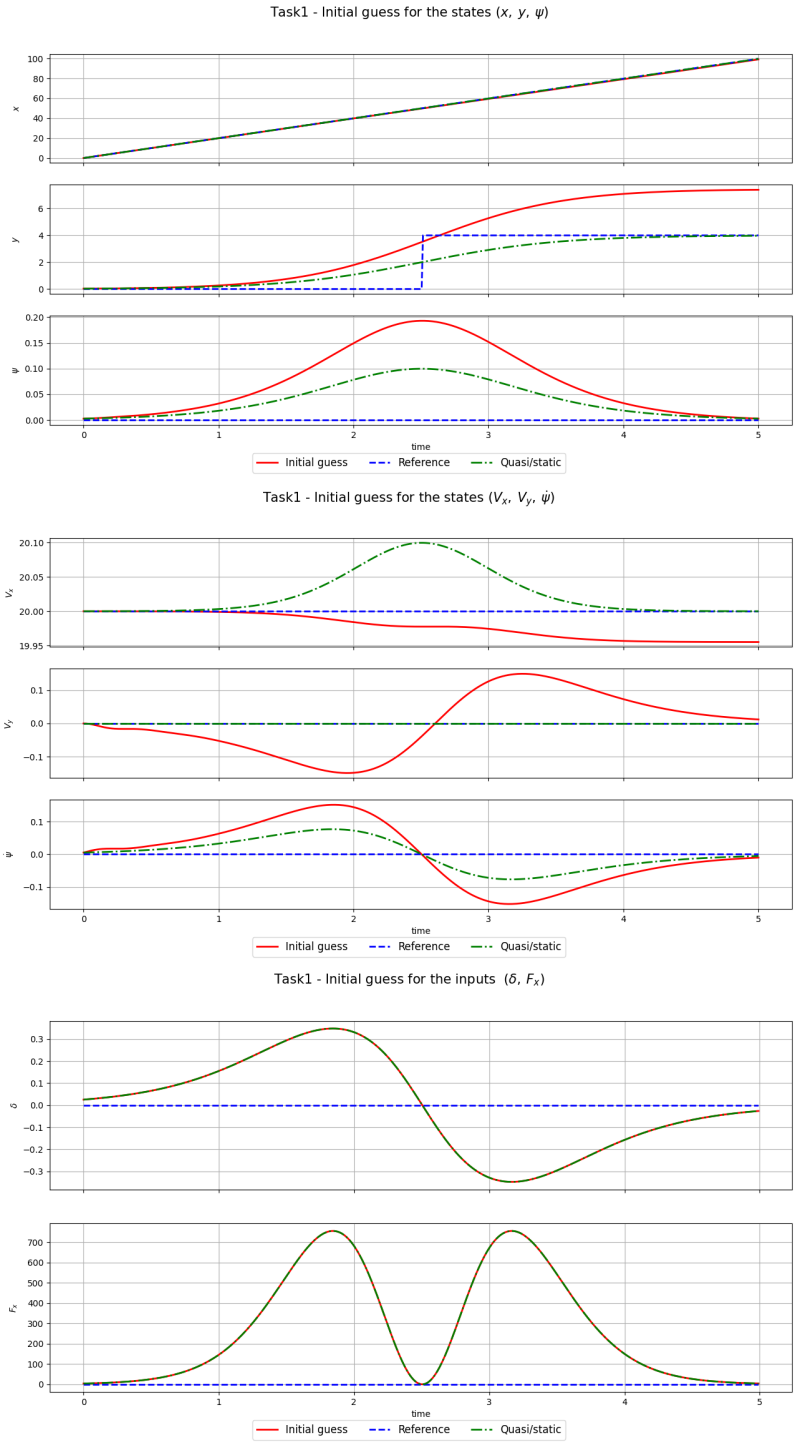

Figure 2.2: Reference, quasi-static and initial-guess trajectories

2.2 Newton's method algorithm

Implementation:

Algorithm 1 Newton's Algorithm for optimal control

Initialization with the trajectory $(\mathbf{x}^0,\ \mathbf{u}^0)$;
for $k = 0, 1, 2, ...$ **do**
  $x_0^{k+1} = x_{init}$
  Set ${p_T}^k = \nabla_x l_T$ and ${P_T}^k = \nabla_{xx} l_T$
  for $t = T-1, ..., 1$ **do**
    Compute:
    $q_t^k = \nabla_{x_t} l_t(x_t^k, u_t^k)$
    $r_t^k = \nabla_{u_t} l_t(x_t^k, u_t^k)$
    $\tilde{Q}_t^k = \nabla^2_{x_t x_t} l_t(x_t^k, u_t^k)$
    $\tilde{R}_t^k = \nabla^2_{u_t u_t} l_t(x_t^k, u_t^k)$
    $K_t^k = -(R_t^k + {B_t^k}^T P_{t+1}^k B_t^k)^{-1}(S_t^k + {B_t^k}^T P_{t+1}^k A_t^k)$
    $\sigma_t^k = -(R_t^k + {B_t^k}^T P_{t+1}^k B_t^k)^{-1}(r_t^k + {B_t^k}^T p_{t+1}^k)$
    $p_t^k = (q_t^k + {A_t^k}^T p_{t+1}^k) - {K_t^k}^T (R_t^k + {B_t^k}^T P_{t+1}^k B_t^k)\sigma_t^k$
    $P_t^k = (Q_t^k + {A_t^k}^T P_{t+1}^k A_t^k) - {K_t^k}^T (R_t^k + {B_t^k}^T P_{t+1}^k B_t^k) K_t^k$
  end for
  for t=0,1,....,T-1 **do**
    Compute:
    $u_t^{k+1} = u_t^k + K_t^k(x_t^{k+1} - x_t^k) + \sigma_t^k$
    $x_{t+1}^{k+1} = f(x_t^{k+1}, u_t^{k+1})$
  end for
end for

Let's notice that we implemented the regularized version of the Newton method, where the matrices ${\tilde{Q}_t}^k$, ${\tilde{S}_t}^k$ and ${\tilde{R}_t}^k$ are computed neglecting the dynamics-terms:

$$\begin{aligned} {\tilde{Q}_t}^k &= \nabla^2_{x_t x_t} l_t({x_t}^k, {u_t}^k) \\ {\tilde{S}_t}^k &= \nabla^2_{x_t u_t} l_t({x_t}^k, {u_t}^k) \\ {\tilde{R}_t}^k &= \nabla^2_{u_t u_t} l_t({x_t}^k, {u_t}^k) \end{aligned} \tag{2.2}$$

Initially, we tried to use the complete update-rule for matrices ${\tilde{Q}_t}^k$, ${\tilde{S}_t}^k$ and ${\tilde{R}_t}^k$, however we realized that most of the times the hessian-terms didn't play a role (being negative semi-definite) and, when they did, they created instabilities in the algorithm.

For this reason we decided to stick to the regularized version of the Newton's method, which still retains the good feature of **quadratic convergence**.

2.3 Cost function and weights

Moving on to the **cost function** (`cost.py`), a quadratic one was considered:

$$l(\mathbf{x}, \mathbf{u}) = \sum_{t=0}^{T-1} \frac{1}{2} \left[x_t^T Q_t x_t + u_t^T R_t u_t \right] + \frac{1}{2} x_T^T P_t x_T \tag{2.3}$$

Our goal was to design the **weight-matrices** in such a way to perform the best possible maneuver with the lowest possible inputs.

Realizing that such inputs have different measurement units, we initially designed the weights in order to make the mismatches (with respect to the reference quantities) on δ and F_x comparable.

Starting from the fact that the forces, expressed in Newtons, have mismatches of the order 10^2, whereas the angles δ, expressed in radians, have mismatches of the order 10^{-1}, and taking into account that such mismatches enter the cost function in a quadratic way, our first guess on the R_t matrix had been:

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

Later, we realized that by applying a smaller weight on F_x, thus by pushing on the minimization of the forces, we were able to get noticeably smaller thrust values with almost no change on the behaviour of δ and the other state variables.

A possible explanation may have to do with the fact that the F_x enters the dynamics equations divided by the mass of the car, resulting in an almost negligible term (the algorithm can push on the minimization of F_x without affecting the other optimization variables).

As a consequence, it made sense to us to opt for this second choice of weights:

$$R_t = \begin{bmatrix} 10^3 & 0 \\ 0 & 10^{-1} \end{bmatrix}$$

In choosing the weights for the remaining optimization variables, we adopted the following reasoning:

- x is the least weighted optimization variable since we are not so interested in closely following the x-reference
- y is weighted less in the stage cost and more in the terminal cost. We made this choice since we want the car to reach the expected endpoint of the maneuver, while we are less interested in closely following the y-reference
- ψ is weighted a lot since it's expressed in radians and we want the associated mismatches to be comparable with the ones on x and y

- V_x is given a medium weight in order to allow the algorithm to deviate from V_0
- V_y is given a high weight as we want to reduce the presence of skidding
- $\dot{\psi}$ is weighted a lot since it's expressed in rad/sec and we want the associated mismatches to be comparable with the ones on V_x and V_y

The matrices are the following:

$$Q_t = \begin{bmatrix} 10^{-1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^3 \end{bmatrix}$$

$$Q_T = \begin{bmatrix} 10^{-1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 10^3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^3 \end{bmatrix}$$

2.4 Newton results and plot

The results obtained at the last iteration of the Newton's algorithm are plotted in figure 2.3.

We can clearly see that when ψ becomes different from 0, V_y appears and as a consequence V_x diminishes (we have imposed a constant velocity along the x-axis and not a constant V_x)

In figure 2.4 we present the lane change maneuver at different iterations of the algorithm.

It can be noticed how, iteration after iteration, the maneuver gets better and better.

As far the inputs are concerned, we notice that both F_x and δ have the expected behaviour.

In particular, F_x has two spikes in correspondence of the highest and smallest values of the steering angle: this is fine since the friction forces to win are higher when the steering angle is larger.

The **cost**, shown in figure 2.5, converges around the fifth iteration, when the **descent** (in figure 2.6) has a value of order 10^2.

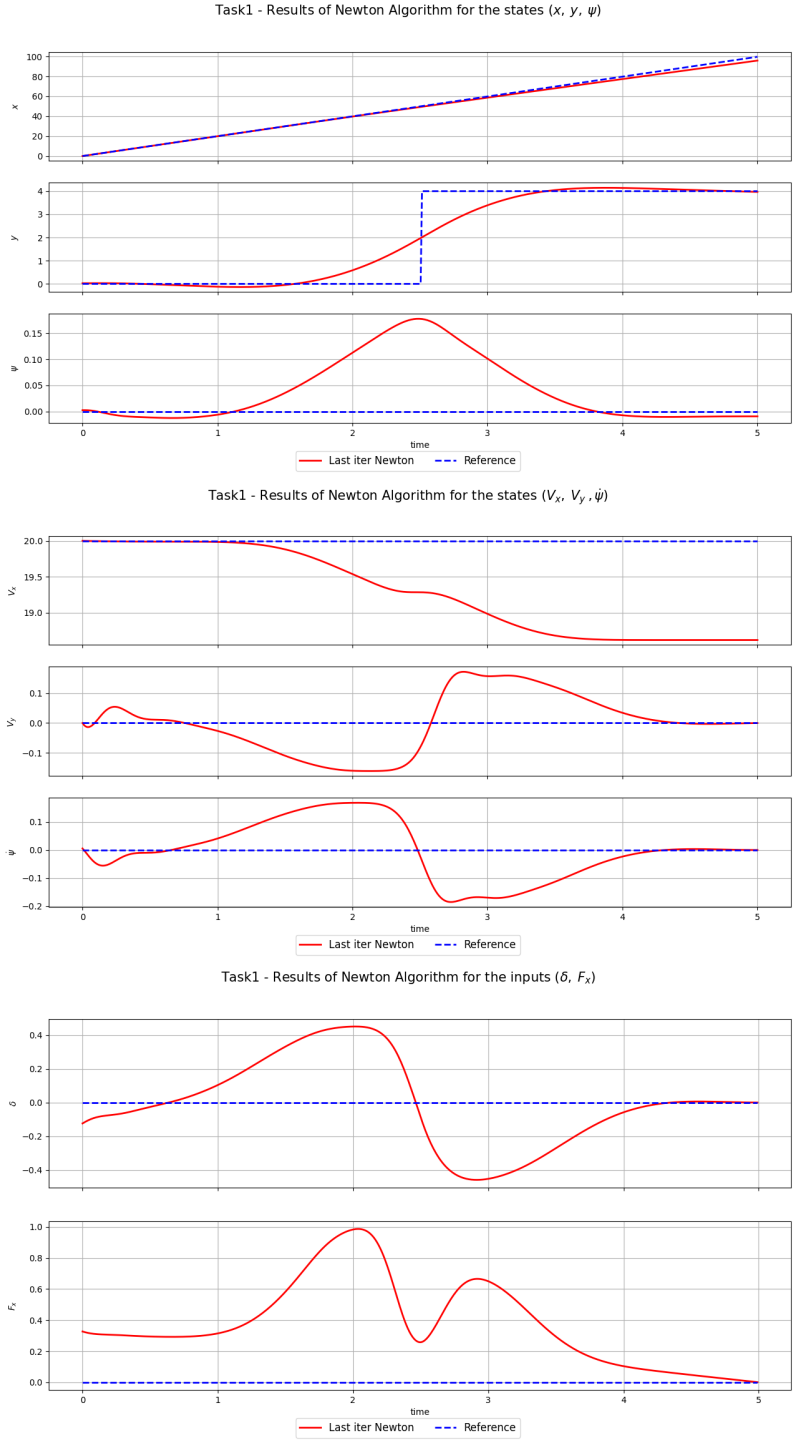

Figure 2.3: Newton's results for task 1

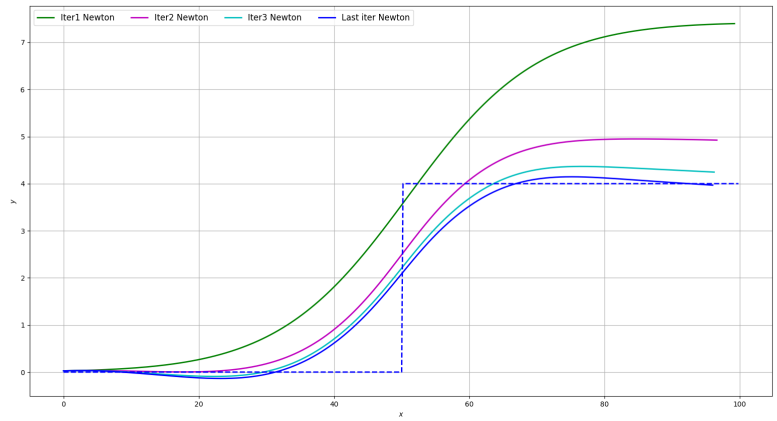

Figure 2.4: Lane-change maneuver at different iterations

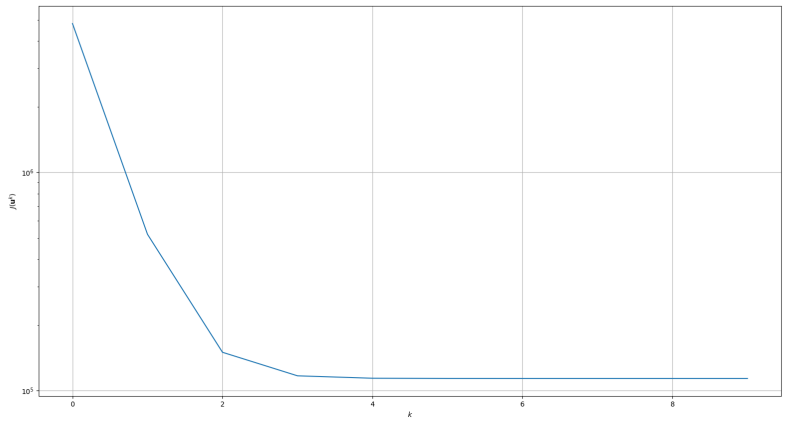

Figure 2.5: Time evolution of the cost function

As a consequence, a meaningful **stopping criterion** for the algorithm may be when the descent drops below 100, or maybe 10.

However, since the descent continues diminishing, we chose as stopping criterion 10^{-2}, to show the good behaviour of the code.

Last but not least, let's point out that we used the **Armijo's method for stepsize computation** at each iteration of the algorithm.

In figure 2.7 we present the Armijo plot associated to the last iteration of the Newton algorithm, we can see that the red line is always tangent to the green one at $\gamma = 0$, as expected.

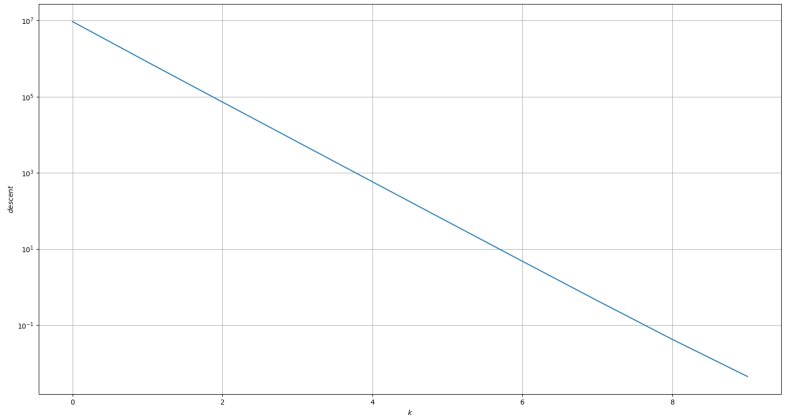

Figure 2.6: Time evolution of the Descent

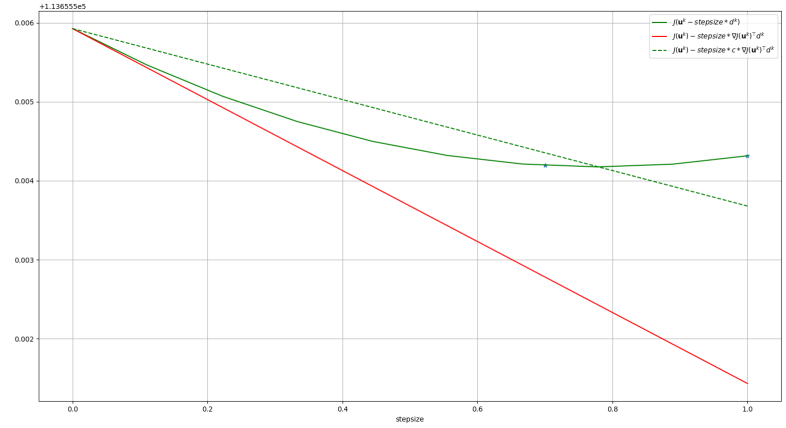

Figure 2.7: Armijo plot at the last Newton iteration

2.5 Case with acceleration

We have also implemented the case in which the car is subjected to a constant acceleration of $1.5\frac{m}{s^2}$ along the x-axis.

We added $mass \cdot 1.5\frac{m}{s^2}$ as reference on F_x since, in a purely forward motion, F_x must be the force which causes the aforementioned acceleration.

The Newton results associated to this case can be found in figure 2.8, let's notice that forces are larger than before since the car accelerates.

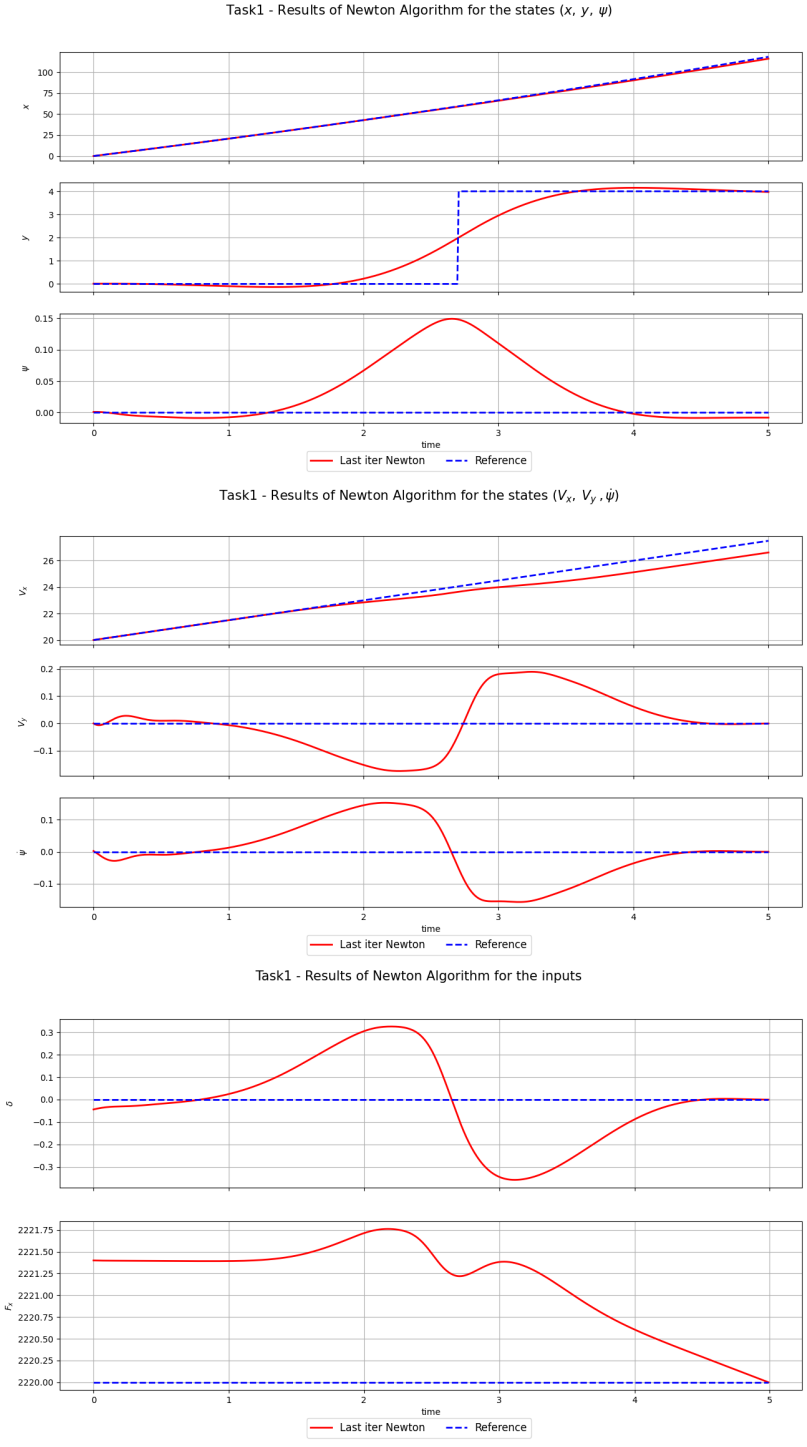

Figure 2.8: Task 1 results in accelerated case

Chapter 3

Second task - Skidpad

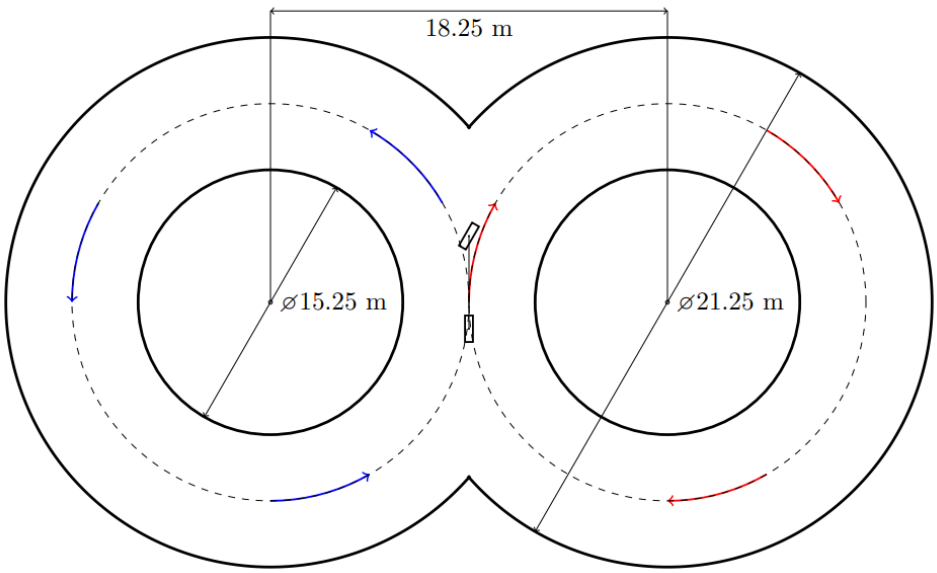

Figure 3.1: Skidpad

In the second task we were asked to use the Newton method previously implemented to make the car move along an "8-shaped" trajectory, the skidpad in figure 3.1.

We treated the case in which the car moves along the trajectory with constant V_x.

3.1 Reference and initial guess trajectory

To compute the **reference trajectory** we imposed a motion law on theta, the angle spanned by the car, and by adopting the same discretization step

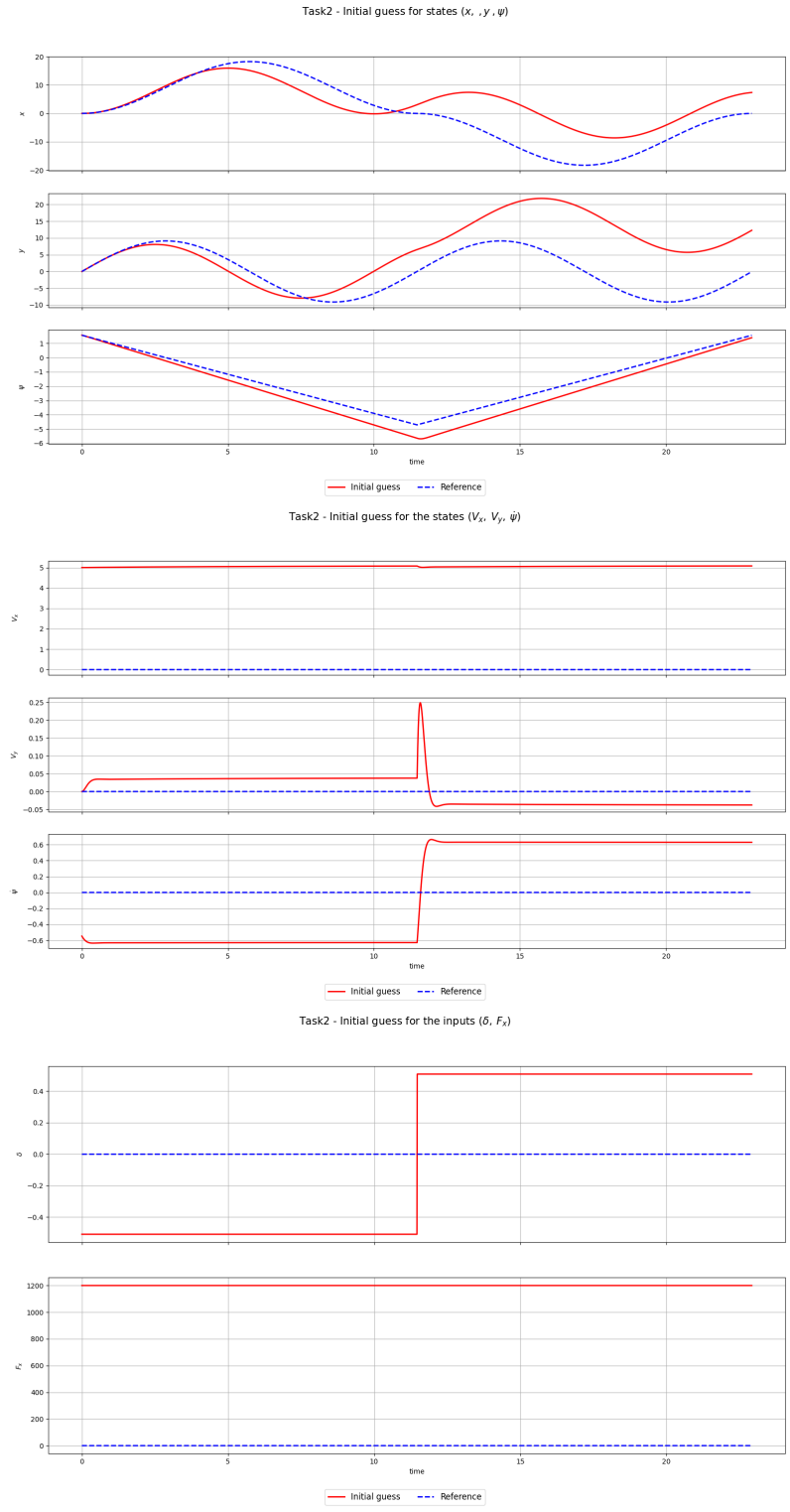

Figure 3.2: Reference and initial guess trajectories

we used for the dynamics equations, dt, we sampled the components of the state vector (`skidpad.py`file):

$$[x\ y\ \psi\ V_x\ V_y\ \dot{\psi}] \tag{3.1}$$

We gave as reference only x, y and ψ, the other state variables are left free.

Moving on to the **initial guess trajectory**, it was obtained by applying the same methodology used before:

- given the velocity state of the car (obtained from the sampling of the centerline), we used a root-finding function (`fsolve`) to compute, at each sampled time instant, the pair of inputs u_{0_t}.
- we plugged the sequence of quasi-static inputs $\mathbf{u}_0$ into the dynamics to get the real state-trajectory $\mathbf{x}_0$ performed by the car (`test dynamics.py` file).

The reference curve and the initial-guess trajectory are plotted figure 3.3.

It is worth pointing out that the initial-guess trajectory is quite different from the reference one, as a consequence of the fact that the reference trajectory is not a feasible one.

3.2 Cost function and weights

In choosing the **weights** we followed the same reasoning done for task 1:

- small weight on F_x as the mismatches are high and caused cost-overflow.
- high weight on δ since it is expressed in radians and we want to make the δ-mismatches comparable with the F_x-mismatches

$$R_t = \begin{bmatrix} 10^2 & 0 \\ 0 & 10^{-5} \end{bmatrix}$$

- In the stage-cost, x and y are given same weight since they have same measurement unit and their ranges of variation ($x \in [0, 21.25]m, y \in [-10.625, 10.625]m$) have same amplitude. Once again, we care a lot about the end-point point of the maneuver and thus we game high weight to x and y in the terminal cost.
- ψ is weighted a lot since it's expressed in radians and we want the associated mismatches to be comparable with the ones on x and y

- V_x is given a medium weight since we want to give the possibility to deviate from V_0
- V_y is given a high weight as we want to reduce the presence of skidding
- $\dot{\psi}$ is weighted a lot since it's expressed in rad/sec and we want the associated mismatches to be comparable with the ones on V_x and V_y

$$Q_t = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10^{-1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^2 \end{bmatrix}$$

$$Q_T = \begin{bmatrix} 10^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^3 \end{bmatrix}$$

We tried several different weight-configurations and we deduced that the aforementioned one is the best in terms of number of Newton iterations before convergence.

Once again it can be seen that radians and meter should be weighted differently and properly.

3.3 Newton results and plots

The Newton results are plotted in figure 3.3.

We can notice that, since we gave a high weight to x, y and ψ, their evolutions over time closely follow the reference.

Let's highlight the fact that F_x is approximately constant as the car moves along the two circumferences with almost constant velocity and constant steering angle.

Moreover, the car terminates the maneuver by going forward and, as a consequnce, δ and F_x tend to zero as t gets closer and closer to the time horizon.

In figure 3.4 we present the maneuver at different iterations of the Newton algorithm.

It can be noticed how, iteration after iteration, the maneuver gets better and better.

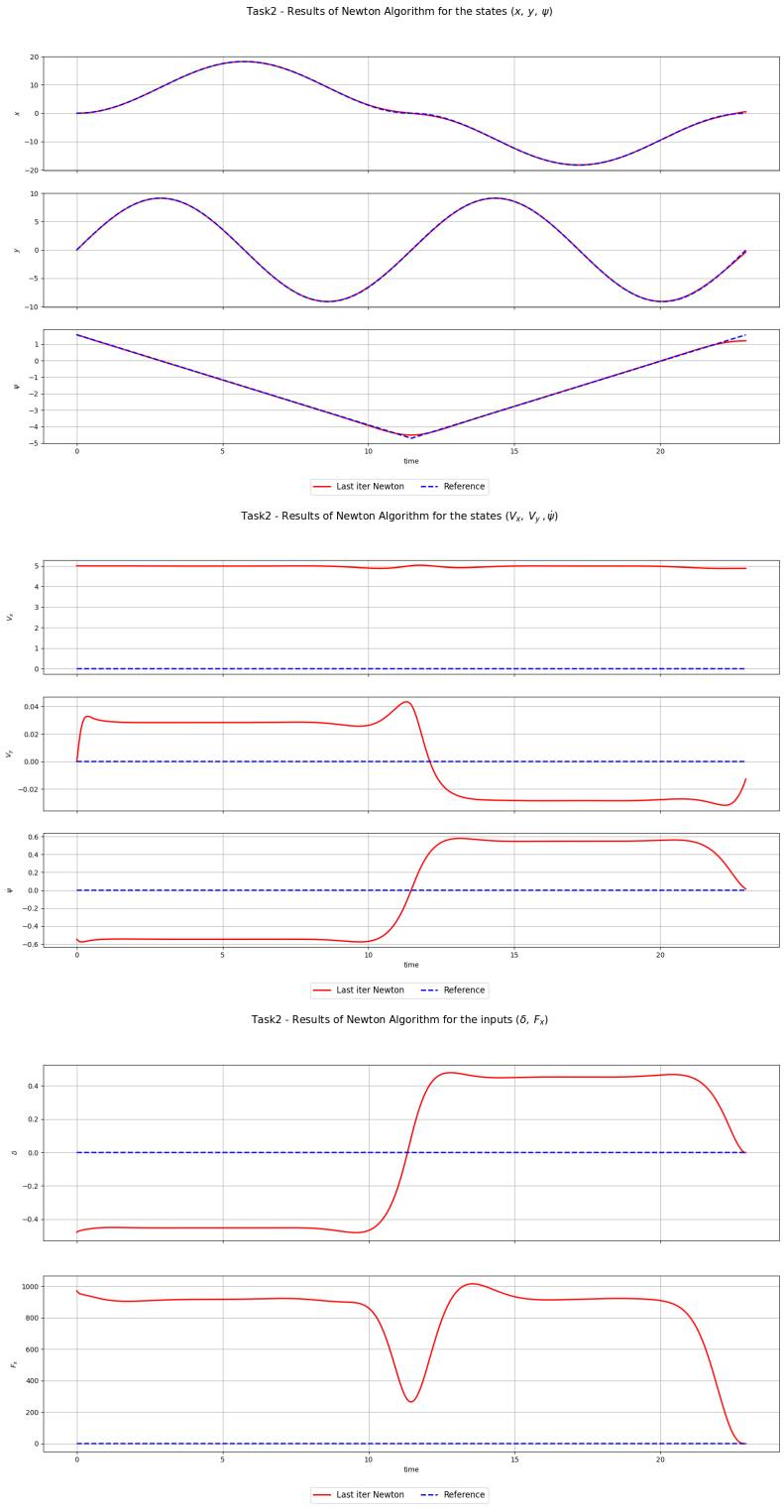

Figure 3.3: Results of Task2

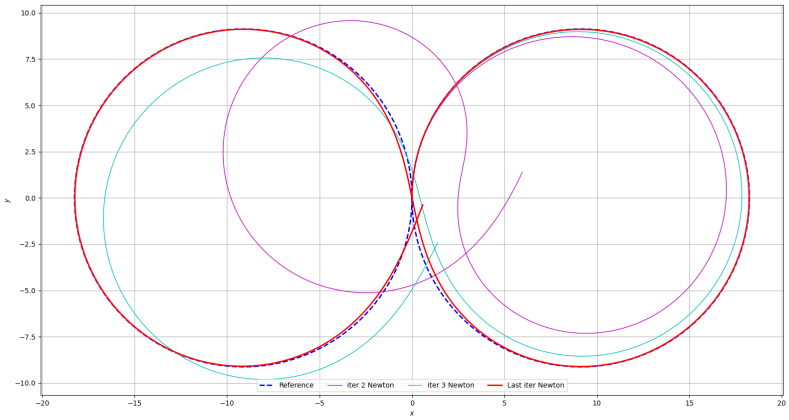

Figure 3.4: Evolution of results on skidpad

Chapter 4

Third task - Trajectory tracking

In the third task we were asked to track the (previously computed) Optimal Trajectory by means of a feedback controller.

Such controller can be found as (part of the) solution of the following finite-horizon-LQ program:

$$\begin{aligned} \min_{\Delta\mathbf{x},\,\Delta\mathbf{u}} \quad & \sum_{t=0}^{T-1} (\Delta x_t^T Q_t^{\text{reg}} \Delta x_t + \Delta u_t^T R_t^{\text{reg}} \Delta u_t) + \Delta x_T^T Q_T^{\text{reg}} \Delta x_T \\ \text{s.t.} \quad & \Delta x_{t+1} = A_t^{\text{opt}} \Delta x_t + B_t^{opt} \Delta u_t \quad t = 0, ..., T-1, \\ & x_0 \quad given \end{aligned} \tag{4.1}$$

where A_t^{opt} and B_t^{opt}t are the matrixes which come from the linearization of the dynamics around optimal points.

The pseudocode of the LQP-solver algorithm is displayed in 2 (next page).

4.1 Cost function and weights

In trying different initial conditions we realized that, to closely follow the optimal trajectory, we need to adjust the weights of the cost function.

Therefore, the weights we provided in this report are specific for the chosen initialization point (small displacements about such point are accepted).

The idea behind weight selection is: lower the weight of an optimization variable the bigger the initial mismatch is, in order to prevent cost-overflow.

We also tried to vary the module of V_x, however we found out that we can't request a V_x higher that $5m/s$.

A possible explanation may have to do with the fact that the trajectory is a complex one and cannot be tracked with high velocities.

Algorithm 2 LQ Optimal Controller

Given an optimal state-input trajectory $(\mathbf{x}_{\text{opt}}, \mathbf{u}_{\text{opt}})$ obtained via Newton's method;
Linearize the system around $(\mathbf{x}_{\text{opt}}, \mathbf{u}_{\text{opt}})$:
for $t = 0, ..., T$ **do**
    $A_t^{\text{opt}} = \nabla_{x_t} f(x^{\text{opt}}, u^{\text{opt}})^T$
    $B_t^{\text{opt}} = \nabla_{u_t} f(x^{\text{opt}}, u^{\text{opt}})^T$
    $\Delta x_{t+1} = A_t^{\text{opt}} \Delta x_t + B_t^{opt} \Delta u_t$
end for
Compute the LQ Optimal Controller
Set $P_T = Q_T^{reg}$ and
for $t = T - 1, ..., 0$ **do**
    Compute:
    $P_t = Q_t^{reg} + A_t^{\text{opt}^T} P_{t+1} A_t^{\text{opt}} - (A_t^{\text{opt}^T} P_{t+1} B_t^{\text{opt}})(R_t^{\text{reg}} + B_t^{\text{opt}^T} P_{t+1} B_t^{\text{opt}})^{-1} (B_t^{\text{opt}^T} P_{t+1} A_t^{\text{opt}})$
end for
Compute the feedback gain
for t=0,1,....,T-1 **do**
    Compute:
    $K_t^{\text{reg}} = -(R_t^{\text{reg}} + B_t^{\text{opt}^T} P_{t+1} B_t^{\text{opt}})^{-1} (B_t^{\text{opt}^T} P_{t+1} A_t^{\text{opt}})$
end for
Track the generated Optimal Trajectory
for $t = 0, ..., T - 1$ **do**
    $u_t = u_t^{\text{opt}} + K_t^{\text{reg}} (x_t - x_t^{\text{opt}})$
    $x_{t+1} = f(x_t, u_t)$
end for

4.2 Results and plots

Figure 4.2 shows the robustness of the algorithm, as it is able to track the optimal trajectory even if we give as initial condition a point which is far away from the origin of the reference system.

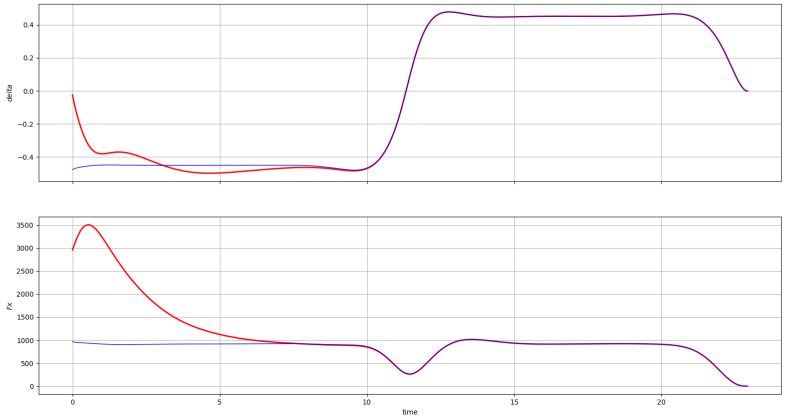

Figure 4.1: Inputs task 3

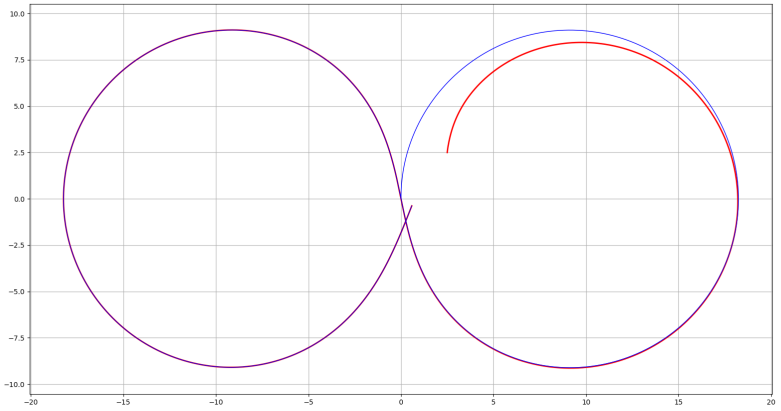

Figure 4.2: Trajectory tracked in task 3

Conclusions

To conclude our work, we would like to highlight some crucial points observed during the development of the project:

- First of all, we noticed that the behaviour of the Newton's algorithm strongly depends on the chosen weight matrices, we didn't expect such a high sensitivity to the weights.
- Moreover, we saw that the convergence depends a lot on the choice of the stepsize: we initially used a small constant stepsize, however the convergence was either very slow or a failure. We observed a big improvement in performance when we put in place the Armijo step size selection method.
- Another important aspect to highlight when it comes to convergence is initialization: without the correct initialization the algorithm was not able to converge.
- Throughout the whole report we adopted a discretization time step dt equal to 10^{-2}, we also tried smaller time steps but we didn't experience a tangible improvement in the results. In turn, what we experienced was a higher computational cost.

In conclusion, making use of all the knowledge accumulated in doing several trials, we obtained the expected results. The inputs we found are coherent with the chosen initialization