H0E76A: Model Predictive Control

— Homework assignments —

Andrea Alboni - r1030437

2024–2025

The assignments below are homework assignments related to the exercise sessions that are organized throughout the semester. The assignments count for 7/20 of the total marks on this course, and are graded based on the following deliverables

Deliverables

- A report containing the answers to the questions and any numerical values or plots that are asked in the assignments. You can write your solutions directly in this `tex`-file. Make sure to fill in your name above.
- A single zip-archive containing all the code to generate the plots included in your report. Make sure to add a readme file describing which script(s) to run (and the working directory from which to do so) to reproduce your results. It is recommended that you have at least one separate script per session.

Most of the assignments build upon the solutions obtained from the sessions. Solutions for the exercise sessions will be made available on Toledo shortly after each session. Before submitting your final report, make sure that you verify your solutions from the sessions if you reuse them here.

Session 1 LQR and Dynamic programming

Assignment 1.1

Compare the finite horizon controllers K_N you computed during the session with the infinite horizon LQR controller.

- Implement the infinite horizon LQR controller $u = K_\infty x$ (with initial state $x_0 = (10, 10)$).[a] Write the obtained value for K_∞. **Remark.** Report your answers with at least four decimal places.
- Plot the simulated closed-loop trajectory of this controller with the trajectories of the finite-horizon controller. What happens as you increase N?
- Explain this observation based on the recursion relation used earlier and the discrete-time algebraic Riccati equation (DARE).

[a] Use `solve_discrete_are` from `scipy.linalg`.

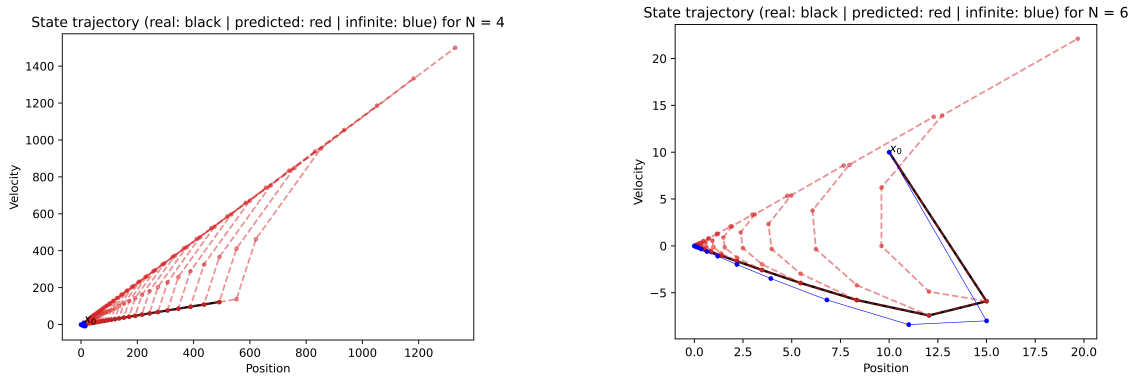
The goal of this analysis is to compare the performance of finite-horizon controllers K_N, computed for various prediction horizons N, with the infinite-horizon Linear Quadratic Regulator (LQR) controller K_∞. The infinite-horizon controller is obtained by solving the Discrete Algebraic Riccati Equation (DARE), which provides the optimal feedback gain K_∞ for minimizing the infinite-horizon quadratic cost.

- The obtained value of K_∞ is:

```
Kinf:  [[1.28645 2.31256]]
```

- The analysis demonstrates, and the plots 1 confirm, that increasing the prediction horizon N improves the performance of the finite-horizon MPC. For small N, the controller struggles to approximate the long-term cost-to-go, leading to poor decision-making and significant deviation from the infinite-horizon LQR solution. However, as N increases, the finite-horizon MPC gradually captures more of the system's future behavior, reducing the mismatch between its terminal cost approximation and the true cost-to-go function. Ultimately, for sufficiently large N, the finite-horizon MPC converges to the infinite-horizon LQR solution, achieving optimal control performance.

  The black curves correspond to the real closed-loop trajectories under the finite-horizon controller, while the red curves represent the predicted trajectories at each time step. For comparison, the blue trajectory represents the infinite-horizon Linear Quadratic Regulator (LQR) controller, which serves as the optimal solution for the system.



(a) State trajectory for $N = 4$

(b) State trajectory for $N = 6$

(c) State trajectory for $N = 10$

(d) State trajectory for $N = 20$

Figure 1: State trajectories for different horizons

For $N = 4$, shown in Figure 1a, the MPC trajectories show a significant deviation from the infinite-horizon LQR solution. The real trajectory and predictions fail to converge toward the desired optimal behavior. This happens because a short prediction horizon limits the controller's ability to account for the system's long-term evolution.

As the prediction horizon increases to $N = 6$, Figure 1b, it improves noticeably. The predicted trajectories (red curves) begin to better align with the infinite-horizon LQR solution (blue curve), and the real trajectory starts to resemble the desired behavior. Although some visible deviation remains, the increased horizon length allows the controller to approximate the terminal cost more accurately, providing better foresight into the system's future behavior. This improvement reflects the recursive nature of the cost-to-go function: with a longer prediction horizon, the controller reduces the mismatch between the finite-horizon cost approximation and the true long-term cost-to-go.

For N=10, Figure 1c, the finite-horizon MPC achieves near-optimal behavior, as evidenced by the close overlap between the real and predicted trajectories with the infinite-horizon LQR trajectory. At this horizon length, the MPC solution has almost converged to the optimal control

law. The terminal cost and the resulting feedback gain closely approximate those obtained from solving the Discrete Algebraic Riccati Equation (DARE), which characterizes the infinite-horizon optimal solution.

As N increases, Figure 1d shows a simulation where $N = 20$, the terminal cost used in the finite-horizon MPC converges to the true cost-to-go, enabling the MPC to replicate the infinite-horizon behavior.

Assignment 1.2

Numerically compare the quality of the finite-horizon LQR controller to the infinite horizon controller. For the same fixed x_0 as before and for N ranging from 1 to 10[a]:

- Plot $V_N = x_0^\top P_N x_0$ versus N
- Plot $V_\infty = x_0^\top P_\infty x_0$ on the same axis (this is just a horizontal line).
- Approximate the infinite horizon cost for the finite-horizon controller using a long state and input trajectory:

$$\hat{V}_N = \sum_{k=0}^{\infty} (x_k^\top Q x_k + x_k^\top K^\top R K x_k) \approx \sum_{k=0}^{100} (x_k^\top Q x_k + x_k^\top K^\top R K x_k).$$

  and add a plot of $\hat{V}_N$ vs. N on the same figure.
- Describe what you observe: Do you observe convergence? Which quantities observe to which and in what direction? **Briefly** explain these observations.

[a] **Hint:** manually set the y-limit of the plot to $[0, 2000]$. Otherwise, the costs of unstable controllers will dominate the figure.

Figure 2 illustrates the behavior of the value function V_N, V_∞ and $\hat{V}_N$ as a function of the horizon N. Each curve represents a different computation of the cost. V_N, shown in cyan, which is converging from below represents the cost of following the MPC policy over a finite horizon N. Since it only considers the cost over the horizon without accounting for what happens beyond it, V_N tends to underestimate the total cost. As N increases, V_N includes the effects of more steps and approaches the infinite-horizon cost V_∞. The value function $\hat{V}_N$, shown in green, represents the cost computed for the infinite sequence of actions when applying the MPC policy derived using horizon N. Initially, for small horizons (e.g. $N < 4$), the MPC policy produces an unstable trajectory, leading to an infinite cost. However, as N increases, the policy stabilizes, and $\hat{V}_N$ converges from above to the infinite-horizon cost V_∞. The magenta line represents the infinite-horizon optimal cost V_∞, which is constant and provides a benchmark for comparison.

The figure highlights the importance of selecting an adequate prediction horizon in MPC. For small horizons, the controller may produce unstable trajectories, as evidenced by the infinite cost. Once the horizon is sufficiently long (e.g. $N \geq 4$), the MPC policy stabilizes, and both V_N and $\hat{V}_N$ approach V_∞, ensuring optimal and stable control.

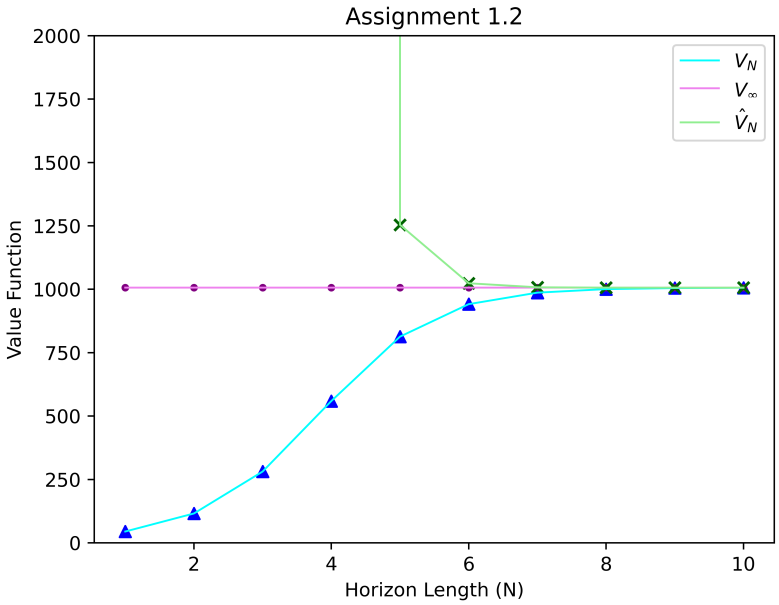


Figure 2: Plot

Session 2 Linear MPC and convex optimization

See the session 2 sheet for the set-up.

The goal of this assignment is to look more closely at what goes wrong in the final exercise of the session, and to gain some intuition about ways to fix it. In essence, the problem that occurs is a symptom of the myopic nature of MPC, that is, it can only take into account what happens in the very near future. To rectify this, we have to provide it with information about the long-term behavior of the system under the applied controls.

Remark. The assignments in this session should be performed in discrete time.

Assignment 2.1

Consider the vehicle at a fixed position and velocity (p_0, v_0). Suppose we apply the maximum brake, i.e., $u_t = u_{\min}$ for $t = 0, \dots, T$. Write an expression of T as a function of v_0, such the vehicle can come to standstill in T steps.

Given the system dynamics:

$$\begin{aligned} x_{t+1} &= \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ T_s \end{bmatrix} u_t \\ \text{where } x_t &= \begin{bmatrix} p_t \\ v_t \end{bmatrix} \end{aligned} \tag{1}$$

it is possible to develop the evolution of the state's elements over time:

$$\begin{cases} p_1 = p_0 + T_s v_0 \\ v_1 = v_0 + T_S u_{min} \end{cases} \tag{2}$$

$$\begin{cases} p_2 = p_1 + T_s v_1 = p_0 + 2T_s v_0 + T_s^2 u_{min} \\ v_2 = v_1 + T_S u_{min} = v_0 + 2T_s u_{min} \end{cases} \tag{3}$$

Therefore we have that:

$$\begin{cases} p_T = p_{T-1} + T_s v_{T-1} = p_0 + T_s v_0 T + \frac{1}{2} T(T-1) T_s^2 u_{min} \\ v_T = v_0 + T_s T u_{min} \end{cases} \tag{4}$$

Since the objective is for the vehicle to come to a standstill in T steps:

$$v_T = v_0 + T_s T u_{min} = 0 \implies T = -\frac{v_0}{T_s u_{min}} \tag{5}$$

Assignment 2.2

For a given T, compute the total braking distance, i.e., the distance travelled during the deceleration: $p_T - p_0$.

As previously stated, the vehicle's position after T steps can be expressed as:

$$p_T = p_{T-1} + T_s v_{T-1} = p_0 + T_s v_0 T + \frac{1}{2} T(T-1) T_s^2 u_{min} \tag{6}$$

Therefore, the total braking distance is given by the difference between the final and initial positions:

$$p_T - p_0 = T_s v_0 T + \frac{1}{2} T(T-1) T_s^2 u_{min} \tag{7}$$

Assignment 2.3

Use the result of the previous exercises to formulate a constraint on a state $x = (p, v)$ that will guarantee that there will always exist a control input that will keep future states in the set $S = \{x \mid p \leq p_{\max}\}$.

Given the total braking distance expression derived in the previous exercise, where T can be expressed as a function of the initial velocity v_0, we can reformulate the total breaking distance as follows:

$$\begin{aligned} p_T - p_0 &= \frac{-v_0}{T_s u_{min}} T_s v_0 + \frac{1}{2}\left(\frac{-v_0}{T_s u_{min}}\right)\left(\frac{-v_0}{T_s u_{min}} - 1\right) T_s^2 u_{min} \\ &= \frac{-v_0^2}{u_{min}} + \frac{1}{2} v_0 \left(\frac{v_0 + T_s u_{min}}{u_{min}}\right) \end{aligned} \tag{8}$$

Therefore, the constraint that guarantees that there will always exist a control input that will keep future states in the set S, states within the breaking distance to p_{max}, is:

$$\begin{aligned} p_0 + (p_T - p_0) &\leq p_{max} \\ p_0 \leq p_{max} - (p_T - p_0) &= p_{max} + \frac{v_0^2}{u_{min}} - \frac{1}{2} v_0 \left(\frac{v_0 + T_s u_{min}}{u_{min}}\right) \end{aligned} \tag{9}$$

Assignment 2.4

Write a function that takes the horizon N as an input and that for all initial states $[-10, 0] \leq x_0 \leq [1, 25]$ (sampled in a grid), checks whether your MPC controller is feasible at this initial state. Set $u_{\min} = -5$. Leave the other settings at the same values as the ones in the session. For all $N \in \{2, 5, 10\}$, make a plot showing the set of states that are feasible (this can be a simple scatter plot), and draw the boundary of the constraint you derived in Assignment 3.
Remark. You only have to check the initial feasibility of the MPC problem, not recursive feasibility.

The cyan curve, in figure 3, 4 and 5, represents the derived constraint, denoting the boundary for feasible initial points given the initial velocities. States below this boundary ensure that the vehicle can decelerate and come to a complete stop within the allowed constraints, having a maximum final position $p_{max} = 1$. Thus, it is expected that only states below this line are classified as feasible initial states, as they satisfy the physical and control limitations of the system.

From figure 3, 4 and 5 can be seen that the number of feasible initial states decreases as the prediction horizon N increases. This phenomenon arises because, with a longer prediction horizon, the Model Predictive Control algorithm accounts for future constraints over a longer period. Specifically, the system becomes increasingly aware that with the minimum allowable braking input of $u_{min} = -5$, the car will not be able to stop in time if the initial velocity exceed a certain threshold, cyan line, with respect to the initial position.

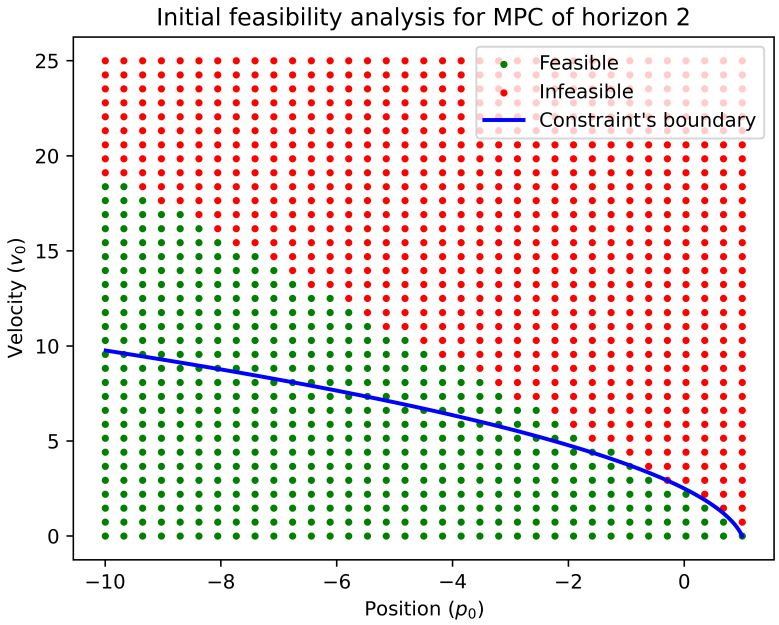


Figure 3: $N = 2$

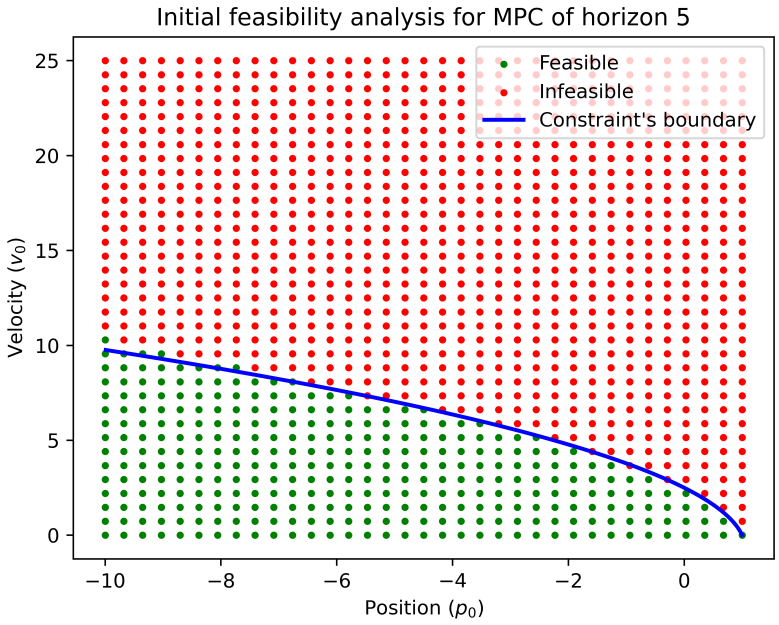


Figure 4: $N = 5$

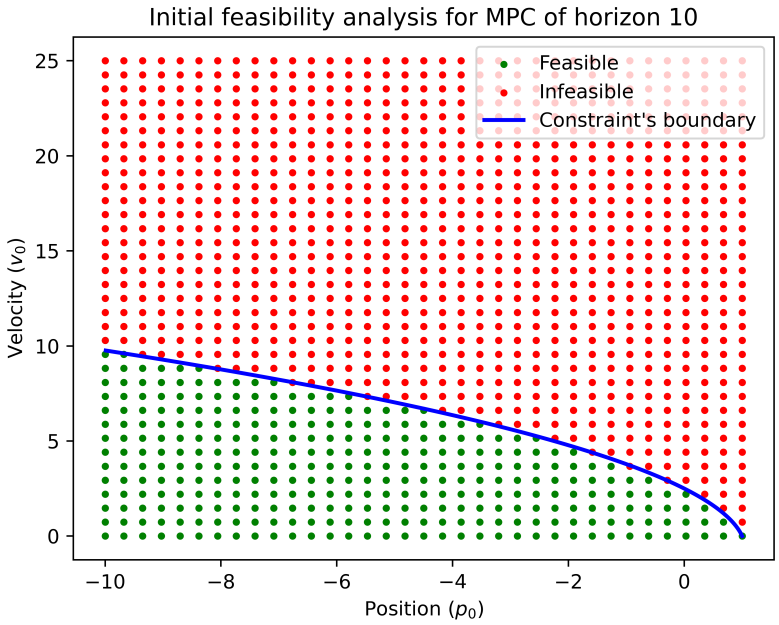

Initial feasibility analysis for MPC of horizon 10
Feasible
Infeasible
Constraint's boundary
Velocity (v_0)
Position (p_0)
25
20
15
10
5
0
−10
−8
−6
−4
−2
0

Figure 5: $N = 10$

Session 3 MPC theory: terminal ingredients

During the sessions, we have computed polyhedral invariant sets to use for the terminal ingredients in our MPC controller. Alternatively, we can use ellipsoidal sets. The goal of this assignment is to compute such sets in a few different ways and compare them with the invariant sets we've computed during the session.

Assignment 3.1

Let K, P denote the optimal, infinite-horizon LQR gain and the solution of the discrete-time Riccati equation, respectively. As shown in the lectures, any level set

$$\text{lev}_\alpha V_\infty = \{x \mid V_\infty(x) = x^\top P x \leq \alpha\}$$

is a positive invariant set for the system under the policy $u = Kx$. However, not all states $x \in \text{lev}_\alpha V_\infty$ necessarily satisfy the state and input constraints.
Write down the expression for the largest value of α such that

$$\begin{cases} x \in X \\ Kx \in U \end{cases} \qquad \forall x \in \text{lev}_\alpha V_\infty,$$

where $X = \{x \in \mathbb{R}^n \mid H_x x \leq h_x\}$ and $U = \{u \in \mathbb{R}^m \mid H_u u \leq h_u\}$ are the (polyhedral) sets of feasible states and inputs, respectively.
Hint. The notion of a support function might be of use.

Given $X_f = \left\{x \in \mathbb{R}^n \mid x^T P x \leq \alpha\right\}$, pick $P \in S_+^{n_x}$ and $\alpha > 0$ such that X_f is positive invariant for $x^+ = A_k x$ and $X_f \subseteq X_k$. We will pick $P = P_f$: $x_f = \left\{x \in \mathbb{R}^{n_x} \mid V_f(x) \leq \alpha\right\}$ since V_f is a Lyapunov function: $V(x^+) \leq V_f(x) - \alpha(\|x\|)$. Now we should find the largest α such that X_f is as large as possible but still within X_k, $X_f \subseteq X_k$.

$$\begin{aligned} X_k &= \{x \in X \mid K_x \in U\} \\ &= \{x \in R^{n_x} \mid H_x x \leq h_x, H_u K x \leq h_u\} \\ &= \left\{x \in R^{n_x} \mid \begin{bmatrix} H_x \\ H_u K \end{bmatrix} x \leq \begin{bmatrix} h_x \\ h_u \end{bmatrix}\right\} \end{aligned}$$

Our objective is therefore to find the largest α:

$$\begin{aligned} \alpha &= \max\left\{\alpha \mid X_f(\alpha) \subseteq X_k\right\} \\ &= \max\left\{h_i^T x \leq g_i \mid x \in X_f(\alpha)\right\} \end{aligned}$$

which, recalling that $X_f(\alpha) \subseteq X_k = \left\{x \mid h_i^T x \leq g_i, \quad i = 1, \ldots, m\right\}$ and implementing the support function $\sigma_{X_f(\alpha)}(h_i) \leq g_i$, can be written as:

$$\max\left\{h_i^T x \mid x^T P x \leq \alpha\right\}$$

Therefore, the lagrange function is:

$$\mathcal{L}(x, \mu) = -h_i^T x + \mu g(x) \quad = -h_i^T x + \mu(x^T P x - \alpha)$$

Setting the gradient to zero, we obtain:

$$\frac{\partial \mathcal{L}(x, \mu)}{\partial x} = -h_i + 2\mu P x = 0 \implies x(\mu) = \frac{1}{2\mu} P^{-1} h_i$$

Substituting $x(\mu)$ back into the lagrange function, we obtain the dual function:

$$\begin{aligned} q(\mu) = \mathcal{L}(x(\mu), \mu) &= -\frac{1}{2\mu} h_i^T P^{-1} h_i - \mu\left(\frac{1}{(2\mu)^2} h_i^T P^{-1} P P^{-1} h_i - \alpha\right) \\ &= -\frac{1}{2\mu} h_i^T P^{-1} h_i + \frac{1}{4\mu} h_i^T P^{-1} h_i - \mu\alpha \\ &= -\frac{1}{4\mu} h_i^T P^{-1} h_i - \mu\alpha \end{aligned}$$

Solving the dual problem leads to:

$$\dot{q}(\mu) = \frac{1}{4\mu^2} h_i^T P^{-1} h_i - \alpha = 0 \implies \mu^* = \sqrt{\frac{1}{4\alpha} h_i^T P^{-1} h_i}$$

Finally the largest value of α can be obtained by solving:

$$\max\left\{\alpha \mid \sqrt{\alpha h_i^T P^{-1} h_i} \leq g_i, \quad i = 1, \ldots, m\right\}$$

which gives:

$$\alpha = \min\left\{\frac{g_i}{h_i^T P^{-1} h_i}, \quad i = 1, \ldots, m\right\}$$

Assignment 3.2

Plot the ellipsoidal set (You can use the given `Ellipsoid` class we provide, together with the function `visualization.plot_ellipsoid` for this) together with the polyhedral set $\{x \in \mathbb{R}^n \mid x \in X,\ Kx \in U\}$. Is the obtained ellipsoidal set a valid terminal set for the MPC problem?

Alternatively, we can more directly encode the requirements for the invariant set and solve a convex optimization problem to find it. Let $E = \{x \in \mathbb{R}^n \mid x^\top P x \leq 1\}$ denote our candidate set, where now P is the positive definite shape matrix. Let $K \in \mathbb{R}^{m \times n}$ furthermore be a candidate state feedback gain. Our goal is to determine P and K to obtain the "largest" possible positive invariant set for $x_{t+1} = Ax_t + Bu_t$ under the policy $u_t = Kx_t$.

The ellipsoidal set E, depicted in Figure 6 in yellow, is centered in $[0, 0]$ and its size is defined by a shape matrix P, obtained from the Riccati recursion. Furthermore, by appropriately scaling the ellipsoid using the parameter α, the largest possible ellipsoidal set that remains entirely within the polyhedral constraints is obtained. The result is a maximized terminal set that touches precisely one boundary of the feasible polyhedral set, depicted in black, ensuring optimal usage of the allowable state space.
This ellipsoid satisfies the necessary stability and feasibility requirements, since its entirely within X_k, additionally its positive invariance ensures that it serves as a valid terminal set for the Model Predictive Control (MPC) probelm.

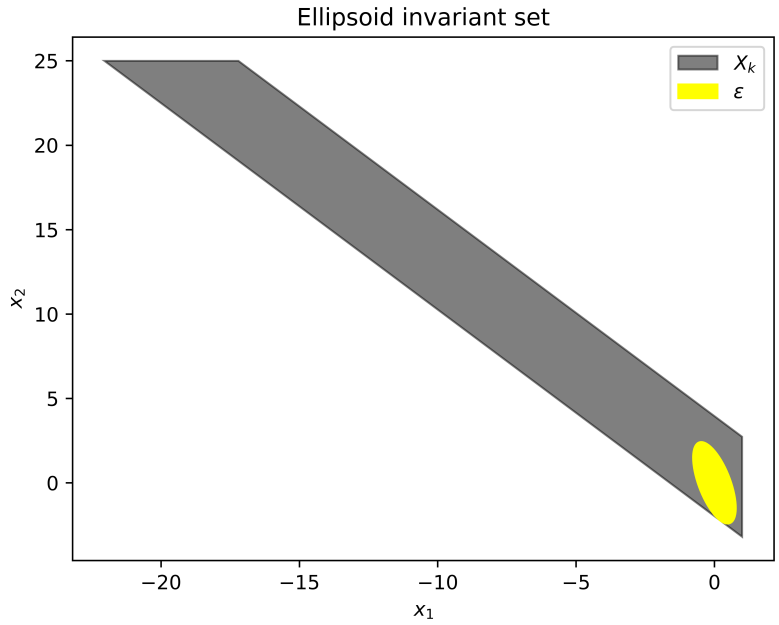


Figure 6: Ellipsoidal set

Assignment 3.3

Formulate a constraint on P and K that guarantees that

$$x^\top Px \leq 1 \implies x^\top (A+BK)^\top P(A+BK)x \leq 1 \quad \forall x \in \mathbb{R}$$

To enforce positive invariance, if $x \in E$, then $x_{k+1} = (A+BK)x_k \in E$ too, we need:

$$x^\top (A+BK)^\top P(A+BK)x \leq x^\top Px, \quad \forall x \in E \tag{10}$$

Subtracting $x^\top Px$ from both sides, we obtain:

$$x^\top \Big[(A+BK)^\top P(A+BK) - P\Big]x \leq 0 \tag{11}$$

This implies:

$$(A+BK)^\top P(A+BK) - P \leq 0 \tag{12}$$

The term $(A+BK)^\top P(A+BK) - P$ represents the "transformed" shape of the ellipsoid E under the system dynamics, $u = Kx$. The constraint ensures that the transformed ellipsoid is contained within the original ellipsoid E, preserving invariance.

Assignment 3.4

Show that the previous constraint can be written as the linear matrix inequality (LMI)

$$\begin{bmatrix} S & (AS+BF)^\top \\ AS+BF & S \end{bmatrix} \geq 0$$

Hint. Doing this involves the following steps.

1. multiply the inequality you obtained from the left and the right by P^{-1}
2. Introduce the change of variables
   - $S = P^{-1}$
   - $F = KS$
3. Apply the Schur complement lemma.

Given $P = P^\top$ by construction, we can multiply the inequality by P^{-1} from the left and the right:

$$\begin{aligned} P^{-1}\Big[(A+BK)^\top P(A+BK) - P\Big]P^{-1} \preceq 0 \\ P^{-1}(A+BK)^\top P(A+BK)P^{-1} - P^{-1} \preceq 0 \end{aligned} \tag{13}$$

Let's introduce the change of variables:

$$\begin{aligned} S = P^{-1} \implies P = S^{-1} \\ F = KS \implies K = FS^{-1} \end{aligned} \tag{14}$$

Therefore, we have that:

$$\begin{aligned} S\Big[A + B(FS^{-1})\Big]^\top S^{-1}\Big[A + B(FS^{-1})\Big]S - S \preceq 0 \\ S\Big[A^\top + S^{-1}F^\top B^\top\Big]S^{-1}\Big[A + B(FS^{-1})\Big]S - S \preceq 0 \\ \Big[AS + BF\Big]^\top S^{-1}\Big[AS + BF\Big] - S \preceq 0 \\ S - \Big[AS + BF\Big]^\top S^{-1}\Big[AS + BF\Big] \succeq 0 \end{aligned} \tag{15}$$

Finally, applying the Schur complement lemma, we obtain the desired LMI:

$$\begin{bmatrix} S & (AS+BF)^\top \\ AS+BF & S \end{bmatrix} \succeq 0 \tag{16}$$

Assignment 3.5

Recall from the lectures that

$$E \subseteq \{x \in \mathbb{R}^n \mid Hx \leq g\} \iff \sigma_E(H_i) \leq g_i,\ \forall i = 1, \dots, p,$$

where σ_E denotes the support function of the ellipsoid E and H_i the ith row of H. Use this fact to show that all states in our candidate invariant set E satisfy the state constraints if and only if

$$H_{x,i}^\top S H_{x,i} \leq h_{x,i}^2, \quad i = 1, \dots, m_x,$$

with $H_{x,i}$ the ith row of H_x and $h_{x,i}$ the ith coordinate of h_x (see Assignment 1) and $S = P^{-1}$ as before.

Hint. An expression for the support function of an Ellipsoid is given in the slides. (It's also a useful exercise to derive it yourself from the definition, but this is not part of the assignment)

Developing the definition of the ellipsoid E:

$$E \subseteq \{x \in \mathbb{R}^n \mid Hx \leq g\} \equiv \left\{x \in \mathbb{R}^n \mid \begin{bmatrix} H_x \\ K^\top H_u \end{bmatrix} x \leq \begin{bmatrix} h_x \\ h_u \end{bmatrix}\right\}$$

The support function of the ellipsoid E is given by:

$$\sigma_E(v) = \alpha^{1/2} \|P^{1/2} v\|_2$$

Therefore, knowing that $P^{-1} = S$, we can write the support function as:

$$\begin{aligned} \sigma_E(H_i) &= \alpha^{1/2} \|S^{1/2} H_i\|_2 \\ &= \alpha^{1/2} \Big[\Big(S^{1/2}H_i\Big)^\top \Big(S^{1/2}H_i\Big)\Big]^{1/2} \\ &= \alpha^{1/2} \Big(H_i^\top S H_i\Big)^{1/2} \end{aligned} \tag{17}$$

Since we are interested only in the states and $x^\top P x \leq 1 \implies \alpha = 1$:

$$\sigma_E(H_{x,i}) = \Big(H_{x,i}^\top P H_{x,i}\Big)^{1/2} \leq g_{x,i} = h_{x,i} \implies \Big(H_{x,i}^\top S H_{x,i}\Big) \leq {h^2}_{x,i}, \quad \forall i = 1, \dots, m_x \tag{18}$$

Assignment 3.6

Similarly, show that $u = Kx$ satisfies the input constraints for all $x \in E$, if and only if

$$h_{u,i}^2 - H_{u,i}^\top F P F^\top H_{u,i} \geq 0, \quad \forall i = 1, \ldots, m_u.$$

Use the Schur complement lemma to write this constraint as an LMI in F and S.

Developing the definition of the ellipsoid E:

$$E \subseteq \{x \in \mathbb{R}^n \mid Hx \leq g\} \equiv \left\{ x \in \mathbb{R}^n \mid \begin{bmatrix} H_x \\ K^\top H_u \end{bmatrix} x \leq \begin{bmatrix} h_x \\ h_u \end{bmatrix} \right\}$$

As previously seen, knowing that $P^{-1} = S$, we can write the support function as:

$$\sigma_E(H_i) = \alpha^{1/2} \left(H_i^\top S H_i \right)^{1/2} \tag{19}$$

Since we are interested only in the inputs and knowing that $K = FS^{-1}$:

$$\begin{aligned} \sigma_E(K^\top H_{u,i}) &= \sigma_E(S^{-1} F^\top H_{u,i}) \\ &= \left(H_{u,i}^\top F S^{-1} S S^{-1} F^\top H_{u,i} \right)^{1/2} \leq g_{u,i} = h_{u,i} \\ &= \left(H_{u,i}^\top F S^{-1} F^\top H_{x,i} \right) \leq {h^2}_{u,i} \\ &\implies h_{u,i}^2 - H_{u,i}^\top F P F^\top H_{u,i} \geq 0, \quad \forall i = 1, \ldots, m_u \end{aligned} \tag{20}$$

Applying the Schur complement lemma, we obtain the desired LMI:

$$\begin{bmatrix} h_{u,i}^2 & H_{u,i}^\top F \\ F^\top H_{u,i} & S \end{bmatrix} \geq 0 \tag{21}$$

Assignment 3.7

Since the volume of the ellipsoid is proportional to $\log\det P^{-1}$, we can now compute the largest (in terms of volume) invariant set by maximizing $\log\det S$ subject to the constraints in Assignments 4 to 6.
Implement this problem using `cvxpy` and plot the resulting ellipsoid (using the given utility code as before.)

The largest invariant set, represented by the ellipsoid, has been computed by:

$$\begin{aligned} \underset{S}{\text{minimize}} \quad & -\log\det S \\ \text{subject to} \quad & \text{constraints in Assignments 4 to 6} \end{aligned} \tag{22}$$

The constraints, elaborated in the previous sections of this assignment, have been formulated as LMI. The resulting ellipsoid is shown in the Figure 7. Notably, this ellipsoid touches two boundaries of the polyhedral set, demonstrating that it has been scaled to its maximum possible size while remaining entirely within the feasible region. This ensures the ellipsoid fully satisfies the constraints, serving as the largest admissible invariant set for the MPC problem.

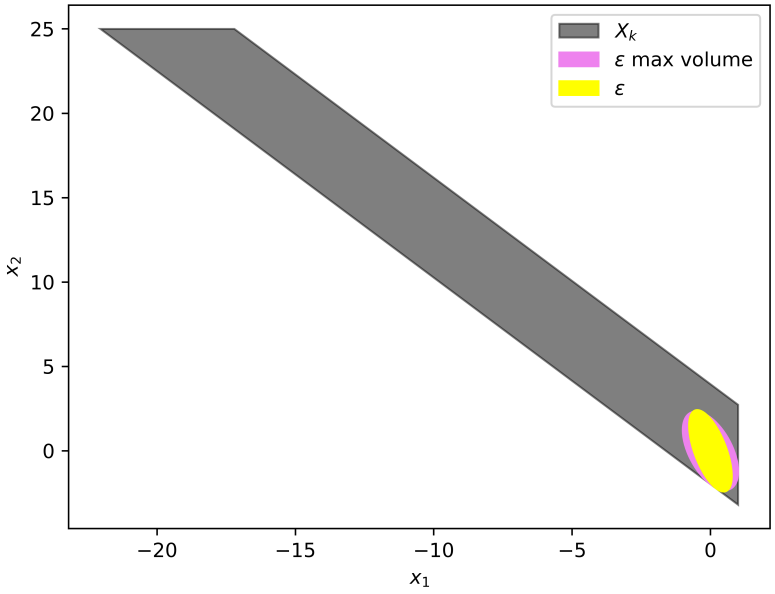


Figure 7: Ellipsoidal sets

Session 4 Nonlinear MPC

We revisit the autonomous parking task but in a slightly more challenging situation. Suppose that there is a vehicle in the adjacent parking spot.

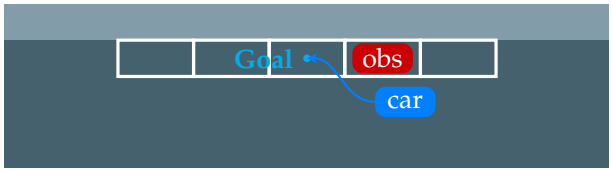


Figure 8: Autonomous parking task with obstacle

Naturally, the vehicle can in no circumstance be allowed to collide with the stationary obstacle. The goal of this assignment is to modify the MPC formulation from the exercise session to include collision avoidance constraints.

One way of formulating the collision avoidance constraint is to cover each vehicle with a single row of circles as illustrated in fig. 9. Here, d denotes the horizontal distance between the center of each circle and its intersection with the rectangle, r denotes the radius of the circles, l is the length of the vehicle and w is the width of the vehicle.

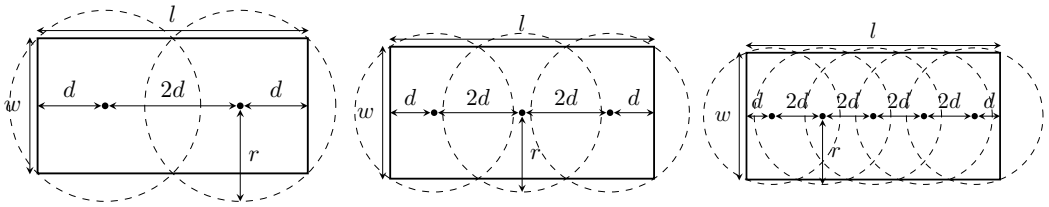


Figure 9: Vehicle covering with 2, 3 and 5 circles respectively.

Assignment 4.1

Let n_c denote the number of circles used to cover the vehicle. Based on the figure, derive an expression for the **centers** $c_i \in \mathbb{R}^2$, $i = 0, \ldots, n_c - 1$ and the radius $r \in \mathbb{R}_+$ of the circles as a function of l, w and n_c. Assume that the origin lies in the center of the vehicle (i.e., the rectangle).
Consider a vehicle of $l = 4$ and $w = 2$, and take $n_c = 3$. To verify your results, use `Circle` and `Rectangle` from `matplotlib.patches` to plot it for this example.
Hint: This task involves only basic geometry.

Studying the figures, we can see derive an expression to compute d:

$$d = \frac{l}{2 * nc} \quad \text{where } nc \text{ is the number of circles}$$

Therefore the position of the center of the circles can be expressed as, taking into consideration the desired refernce frame:

$$c_i = \begin{bmatrix} -\frac{l}{2} + (1 + 2i)d \\ 0 \end{bmatrix} \quad \text{where } c_i \text{ is the center of the i-th circle}$$

Moroever, the radius of the circles, applying the Pitagora theorem, can be expressed as:

$$r = \sqrt{d^2 + \frac{w^2}{4}}$$

Below, Figure 10, a picture of the vehicle covered by 3 circles is shown.

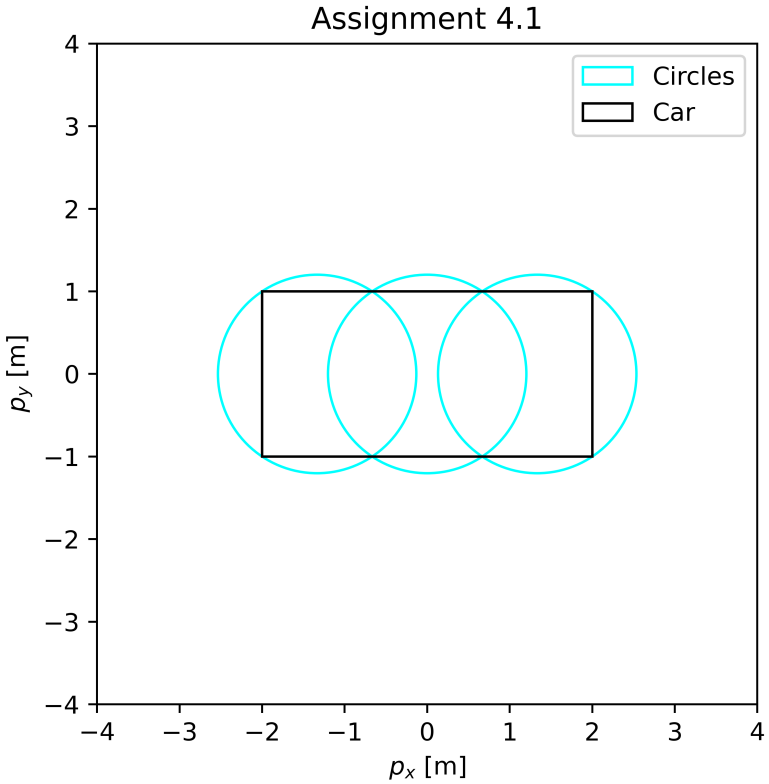


Figure 10: Vehicle covered by 3 circles

Assignment 4.2

Given that the current vehicle state is $x = (p_x, p_y, \psi, v)$, express the center $\bar{c}$ of a circle in global coordinates, given that its local coordinates (aligned with the vehicle, as in the previous exercise) are given by c.
For a vehicle of length $l = 4$, width $w = 2$, with its center at $p = (2, 2)$ and heading angle $\psi = \frac{\pi}{4}$, use your result (combined with the previous assignment) to plot the covering circles on this vehicle for $n_c = 3$.
Hint: You can write your result in terms of the rotation matrix $R(\psi)$ defined by the heading angle ψ. There is no need to work out everything into individual components.

In order to go from local coordinates to global coordinates, we implement the rotation matrix $R(\psi)$:

$$R(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{bmatrix}$$

And we sum to the circles' local coordinates obtained previously the global coordinates of the vehicle:

$$c_{i,global} = Rc_{i,local} + p \quad \text{where } p \text{ is the position of the center of the car } \begin{bmatrix} p_x \\ p_y \end{bmatrix}$$

Below, Figure 11, a picture of the vehicle having a yaw, $\psi = \pi/4$, positioned in $p = [2, 2]^\top$ covered by 3 circles is shown.

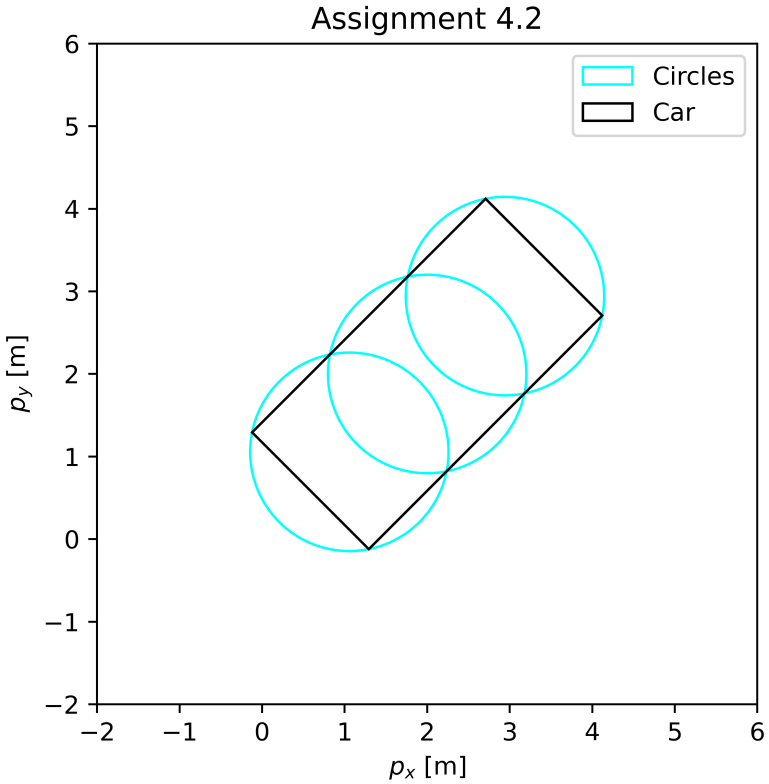


Figure 11: Vehicle global coordinates covered by 3 circles

Assignment 4.3

Given that the controlled vehicle is covered by circles with centers $\bar{c}_i$ $i = 0, \ldots, n_c - 1$ (expressed in global coordinates) and radius r, and similarly the obstacle vehicle is covered by circles with centers $\bar{c}'_i$ and radius r'. Formulate a set of *smooth* constraints

$$g(\bar{c}_i, \bar{c}_j) \leq 0$$

on $\bar{c}_i$ and $\bar{c}'_j$, $i, j \in 0, \ldots, n_c - 1$. which – when satisfied – guarantees that the intersection between the two vehicle rectangles is empty.
Additionally, show whether this set of constraints is convex or not.
Hint: Squaring can get rid of isolated points of non-smoothness.

To guarantee that the intersection between the two vehicles is empty, using circles, we can formulate a constraint based on the distance between the centers of the circles and the sum of their radii. To guarantee the smoothness of the constraint, we can square both distances.

The squared distance between the centers of the circles is:

$$d_c^2 = \|\bar{c}_i - \bar{c}'_j\|_2^2 = (\bar{c}_i - \bar{c}'_j)^\top (\bar{c}_i - \bar{c}'_j) \quad \forall i, j$$

The squared sum of the radii is:

$$d_r^2 = (r + r')^2$$

Therefore, the constraint can be formulated as:

$$\begin{aligned} g(\bar{c}_i, \bar{c}'_j) &= d_r^2 - d_c^2 \leq 0 \\ &= (r + r')^2 - (\bar{c}_i - \bar{c}'_j)^\top (\bar{c}_i - \bar{c}'_j) \leq 0 \end{aligned} \tag{23}$$

The euclidean norm squared is a convex function, the negative of a convex function is a concave function. Therefore, the constraint is concave because it is the sum between a concave function and a constant.

Assignment 4.4

Implement the constraint you derived into your MPC controller formulation. That is, enforce that

$$g(\bar{c}_i(x_t), \bar{c}'_j) \leq 0, \quad \forall i, j \in \{0, \dots, n_c - 1\}, \forall t \in 1, \dots, N.$$

where $\bar{c}_i(x_t)$ is the center (in global coordinates) of the i'th circle on the controlled vehicle expressed as a function of its state at time step t, $\bar{c}'_j$ is the center (in global coordinates) of the j'th circle on the obstacle vehicle and g is the function you derived in the previous exercise. Using the following settings, Simulate your MPC controller for 100 time steps in closed loop. You can use the MPC model dynamics for the simulation (no model mismatch):

x_0	$(0.3, -0.1, 0, 0)$
N	30
T_s	0.08
Obstacle pos.	$(0.25, 0)$ (heading 0)

Use `plot_state_trajectory` from `given.plotting` to visualize the state trajectory, including the obstacle (simply pass the function a list with only one state, since it is stationary). Is the trajectory collision-free? Does the car converge to a point that is entirely within the parking spot [a]?
Optionally, you can also generate an animation using `given.animation`, but this is not a deliverable. (see the solution of the exercise session for some example code. You can use the argument `obstacle_positions` to visualize the obstacle.)

[a] defined in the code as a rectangle with dimensions `PARK_DIMS`=(0.25, 0.12)

As can be seen in Figure 12, using the default values for Q, R, and N, the trajectory is collision-free but the vehicle is not fully parked within the parking spot.

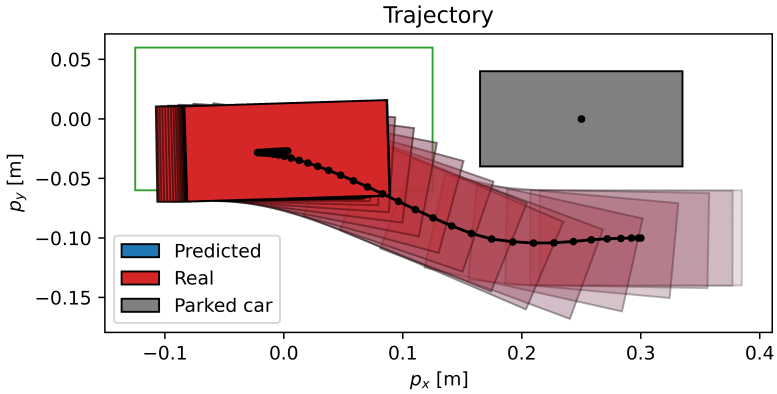


Figure 12: Parking trajectory using default parameters

Assignment 4.5

Play around with the tuning of Q, R, N or other controller parameters (perhaps even n_c, feel free to get creative) to obtain a controller which can (without collision) park the vehicle such that it is fully enclosed in the parking spot. Briefly describe your reasoning while tuning, report the final parameters that you changed from the provided defaults and illustrate the final result with a plot of the trajectory.

These parameters and how they affect the behavior of the car have been thought through, and then some experiments have been performed to see how they work in practice. Here's the results:

- n_c: This parameter affects how closely the car passes by the parked car. Increasing the number of circles improves the approximation of the car by the circles, allowing the car to pass closer to the parked car.
- N: A larger prediction horizon N allows the car to get closer to the desired position (the center of the parking spot $(0, 0)$). However, this also increases the computational cost and time.
- Q: Modifying the first two elements on the diagonal of Q helps by increasing the cost associated with the position in x and y. Consequently, the car will tend to improve along those directions. Additionally, changing the third element (related to the *yaw*) can also be beneficial.
- Q_N: Modifying its elements leads to a greater penalization of the final position of the car additionally pushing the car to reach the origin.

Given the observations, the following parameters were changed from the provided defaults and the final tuning is shown in the code below. Using these values, the final position reached by the car is $[x, y, \text{yaw}, \text{vel}] = [-2.68\,\text{exp}\,{-06}, -3.65\,\text{exp}\,{-04}, -1.28\,\text{exp}\,{-03}, -6.91\,\text{exp}\,{-07}]$. The final parking trajectory is illustrated in Figure 13.

```
Q = cs.diagcat(5, 30, 0.05, 0.01)
R = cs.diagcat(0.5, 0.007)
Q_add = cs.diagcat(0, 15, 10, 0)
Q_N = 5 * Q + Q_add
```

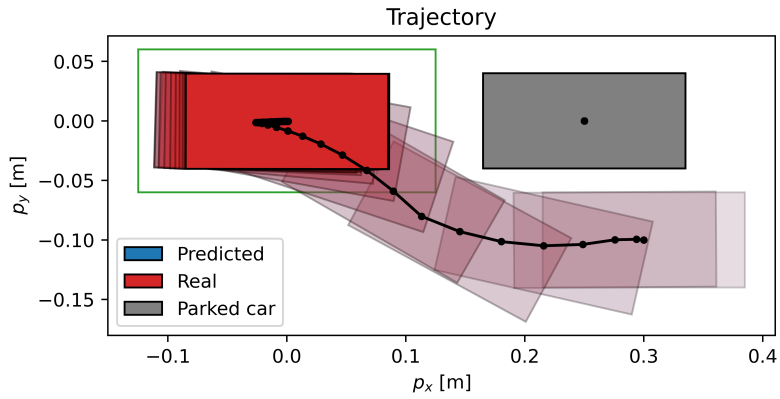


Figure 13: Parking trajectory using custom parameters

Assignment 4.6

Measure your solver time. A quick and easy way to do this is to invoke `perf_counter` from the `time` module before and after your solver call, within the `__call__` method of your controller. In general, if a solver returns its own time, its usually better to use this instead of measuring the time yourself, in order not to account for overhead in the modeling language. For this assignment, however, you can just use your own timing. You can log the solver time by creating a log class

```python
from dataclasses import dataclass
from rcracers.simulator.core import BaseControllerLog, list_field
@dataclass
class ControllerLog(BaseControllerLog):
    solver_time: list = list_field()
```

and passing it as `log` to the `simulate` method:

```python
from rcracers.simulator import simulate
```

```
log = ControllerLog()   # Initialize an empty log
x = simulate(..., log=log) # Simulate
print(log.solver_time)  # Now the solver times have been written
```

Within your solver call, you can then log it like this:

```
from time import perf_counter
def __call__(self, y, log) -> np.ndarray:
    start = perf_counter()
    solution = self.solve(y)
    stop = perf_counter()
    u = self.reshape_input(solution)
    log("solver_time", stop-start)
    return u[0]
```

Visualize the resulting solver time. Is your MPC controller real-time capable? How did you verify this? If it isn't, finetune your controller to obtain one that is real-time capable and still successfully parks into the parking area. Report your final tuning (any value that you changed from the provided defaults) and show the timings and state trajectory of the final controller.

I assumed real-time execution to be any solver time below 75% of the sampling time t_s (i.e. $0.75t_s$), this has been implemented in code as can be seen below. To achieve this, I adjusted, from the previously custom parameters, the prediction horizon length N to 8, which reduced the solver time to 0.5 seconds, comfortably meeting the real-time requirement ($0.5 < 0.6 = 0.75t_s$). Below, in Figure 14 the car's trajectory with this tuning applied can be seen.

```
if np.max(log.solver_time) > 0.75 * ts:
    print(f"Solver time: {np.max(log.solver_time):.2f} s")
    print("Not real time execution")
else:
    print(f"Solver time: {np.max(log.solver_time):.2f} s")
    print("Real time execution")
```

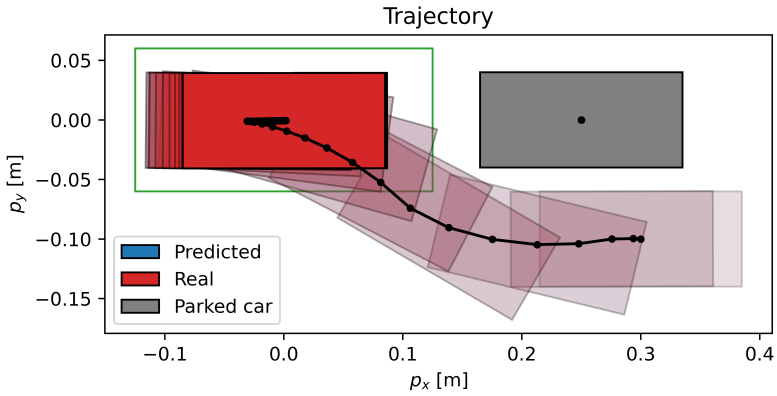


Figure 14: Parking trajectory using custom parameters and real-time execution

Session 5 State estimation

The goal of this assignment is to further improve the MHE you developed in the associated exercise session. Specifically right now we only use the data provided within the horizon of the MHE optimization problem. Instead we would like to use past information as well, which we can do using priors.

Assignment 5.1

We will implement a prior update. Since you already have the EKF iterates available, these can be used to add the *filtering prior update* as described in the slides:

$$\Gamma_{T-N}(z) = \frac{1}{2} \|z - \hat{x}_{T-N}\|_{(P^-_{T-N})^{-1}} .$$

1. Note that the `prior` can already be passed to the provided `build_mhe` method:

```
fs, hs = get_system_equations(symbolic=True, noise=True)
loss = lambda w, v: w.T @ w + v.T @ v
solver = build_mhe(loss, f, h, 10, lbx=0.0, ubx=10.0, use_prior=True)

# you can solve the optimization problem as follows:
x, w = solver(P=np.eye(3), x0=np.zeros(3), y=np.zeros((10, 1)))
```

2. Alter your `MHE` class to use `EKF` internally. Keep in mind that you should produce $\hat{x}_{T-N}$ using only measurements available before time step $T - N$ and the same is true for P^-_{T-N}. How many past estimates $\hat{x}_{T-N}$ and P^-_{T-N} should you store internally?
3. Verify, both for longer horizons ($N = 25$) and shorted ($N = 10$) ones that the filtering prior improves the behavior. Plot the estimated trajectories.
4. Does clipping the state estimates in the *EKF* aid the performance? Plot the result.

1. The prior can be passed to the `build_mhe` method as follows:

```
def build(self, horizon: int):
    return build_mhe(
        self.loss,
        self.f,
        self.h,
        horizon,
        lbx=self.lbx,
        ubx=self.ubx,
        use_prior=True
    )
```

2. The `MHE` has been modified in the following way to implement internally an `EKF` that to produce $\hat{x}_{T-N}$, and P^-_{T-N}, uses only the measurements available before time $T - N$. The number of past estimates to store should be equal to the horizon length, N.

```
    def __call__(self, y: np.ndarray, log: LOGGER):
    # Store the new measurement
    self.y.append(y)
    if len(self.y) > self.horizon:
        self.y.pop(0)

    # Update EKF with the new measurement
    for y in self.y:
        self.ekf(y)

    # Store past estimates of x and P
    self.x_past.append(self.ekf.x.copy())
    self.P_past.append(self.ekf.P.copy())
    if len(self.x_past) > self.horizon:
```

```
            self.x_past.pop(0)
            self.P_past.pop(0)

        # Use EKF state as initial guess for MHE
        if len(self.y) < self.horizon:
            solver = self.build(len(self.y))
            initial_state = self.x_past[0]
            P = self.P_past[0]
        else:
            solver = self.solver
            initial_state = self.x_past[-self.horizon]
            P = self.P_past[-self.horizon]

        # Update MHE
        x, _ = solver(P, initial_state, self.y)

        # update log
        log("x", x[-1, :])
        log("y", y)
```

3. The obtained plots for $N = [10, 25]$ are shown in Figures 15 and 16.

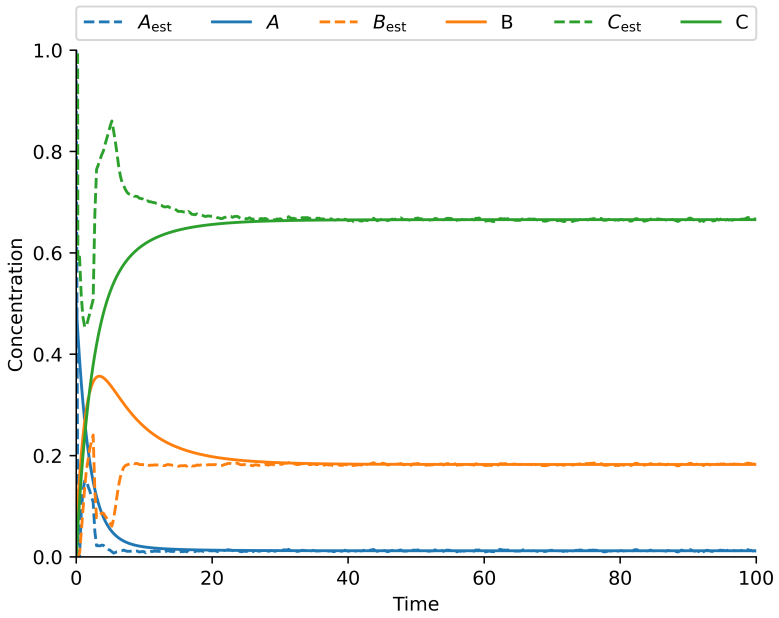


Figure 15: States and their estimation evolution - N = 10

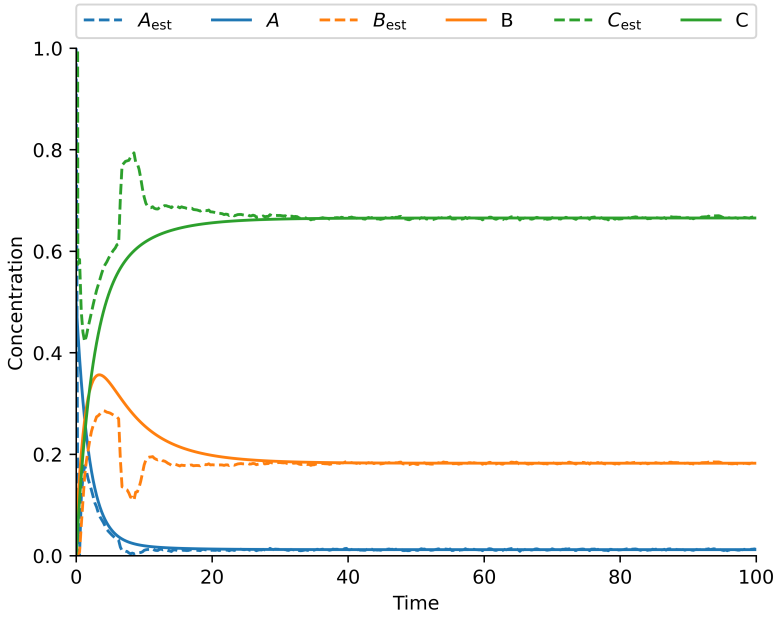


Figure 16: States and their estimation evolution - N = 25

4. Clipping the state estimates in the `EKF` aids the performance making sure that non positive actions are avoided. Figures 17 and 18 show the state evolution for both the horizon lenghts with clipping.

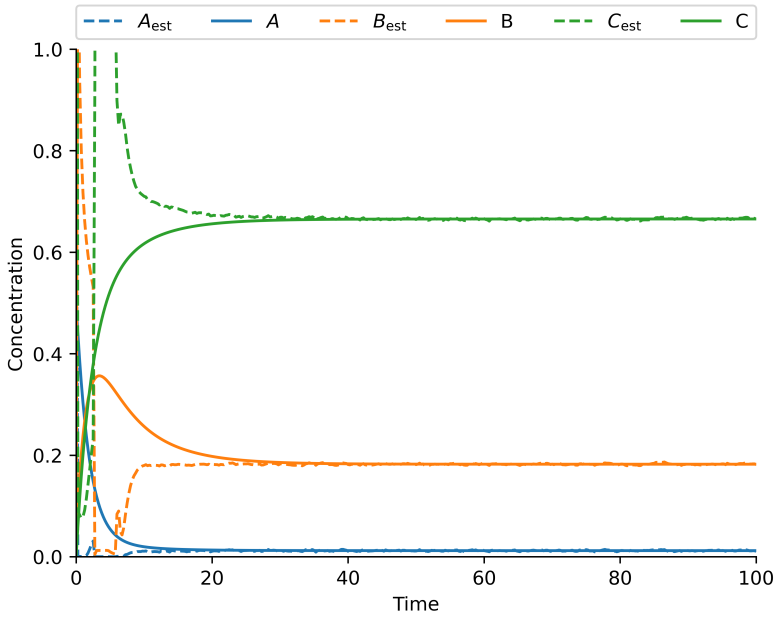


Figure 17: States and their estimation evolution - N = 10 + Clipping

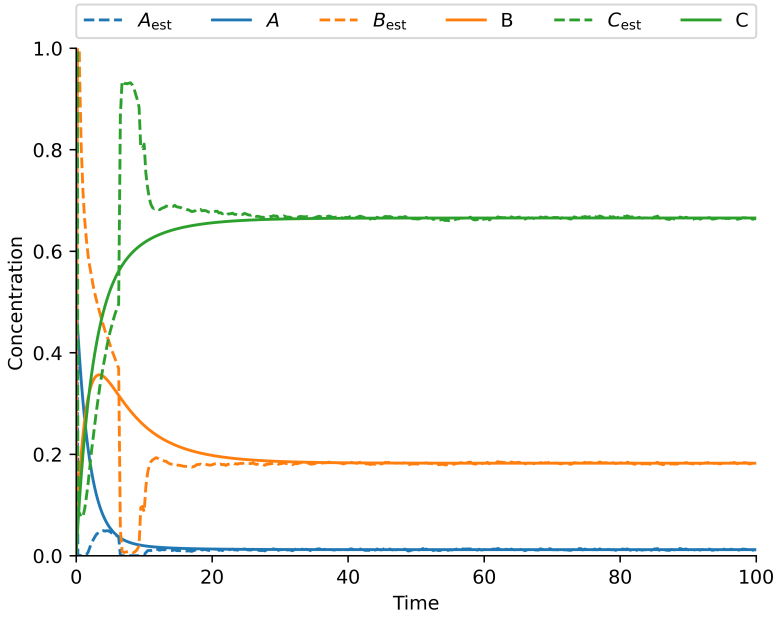


Figure 18: States and their estimation evolution - N = 25 + Clipping

Session 6 Optimization

The goal of this assignment is to get familiar with methods for solving nonlinear MPC problems. Due to the nonlinearity of the dynamics, the OCP will be nonconvex and therefore, the methods from during the session will no longer be applicable.

The given code (`problem.py`), defines a few test problems and some data-structures that you can use to build your nonlinear solver. Have a brief look at this file, and read the comments therein to get acquainted with the provided ingredients. You will not be required to make any changes in this file.

A scaffolding for your own implementation is given in `template.py`. Throughout this assignment, you will fill in the blanks in this file, to finally obtain a solver for nonlinear optimal control problems.

6.1 Building the solver

We will build a Newton-Lagrange method for multiple shooting (i.e., simultaneous approach).

Assignment 6.1

Implement the Newton-Lagrange method (SQP) by completing the auxiliary functions used by `newton_lagrange` in the provided template. Particularly, the functions `lqr_factor_step`, `lqr_solve_step` and `update_iterate` need to be implemented. The former two functions should implement Algorithms 2 and 3 in the related lecture slides. In `update_iterate`, you can for now assume that the option `linesearch` is `False`.
Run the function `test_linear_system` and present the output. This function calls your SQP solver on a problem involving a linear system. Is the output what you expect? Why?

Expected convergence in 1 iteration as the system is linear, but reached in 2 iterations.

```
it.    0 | JN = 1.51e+04 | ||h||2 = 1.25e-13
it.    1 | JN = 1.51e+04 | ||h||2 = 1.07e-13
Converged
```

6.2 Nonlinear test case

Now that you have tested your implementation on a linear system, we will move on to a nonlinear example.

Consider the continuous time dynamics

$$\begin{aligned}\dot{x}_1 &= 10(x_2 - x_1)\\ \dot{x}_2 &= x_1(u - x_3) - x_2.\\ \dot{x}_3 &= x_1 x_2 - 3x_3\end{aligned}$$

which are discretized using the foward Euler scheme. This system is already implemented in `problem.py` under `ToyDynamics`. The problem that you can pass to your SQP solver is described by `ToyProblem`.

Assignment 6.2

Construct an initial guess of all zeros for the inputs, the states and the costates. However, for the states, keep in mind that the first entry must be the provided initial state, i.e., `problem.x0`. Run your solver with this initial state. Give the cost and constraint violation of the last iterate (printed automatically using the `Logger` class in `given.problem`). Use the provided `animate_iterates` function to animate the states and inputs over the iterates and to export a figure of the final solution. Add this plot in your report. Does your solution look plausible?

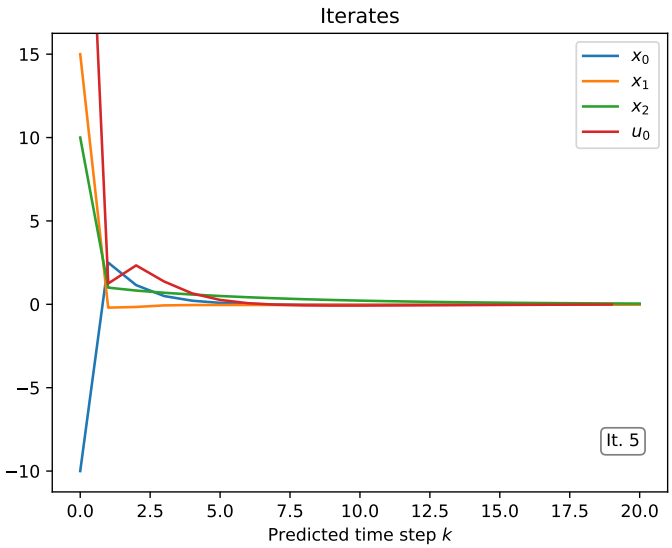


Figure 19: Final solution of the nonlinear test case - Initial guess of all zeros.

Assignment 6.3

Simulate the system and use this as an initial guess.
You may find that the solver now fails to converge. Why do you think this happens?
Hint. Plot the state trajectory you used as an initial guess. Does it look like the solution you just obtained in the previous exercise? Why is this important?

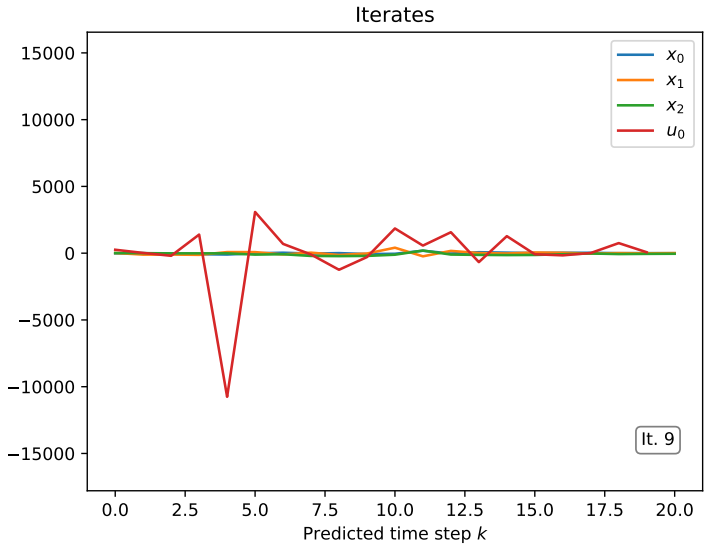


Figure 20: Final solution of the nonlinear test case.

6.3 Adding Line Search

Assignment 6.4

Implement line search and the experiment of Assignment 3. Use

$$\phi(z) = J_N(z) + c\,\|h(z)\|_1$$

as a merit function (see `problem.build_cost_and_constraint`).
Hint. Make sure to set `linesearch=True` in the `cfg` argument of the solver, which is expected to be an instance of `NewtonLagrangeCfg`.
Now does it converge? Give the output of the method at the final iterate and use this to argue that your method indeed converged to something meaningful. Do you notice any difference with respect to the solution in Assignment 2? How do you explain this?

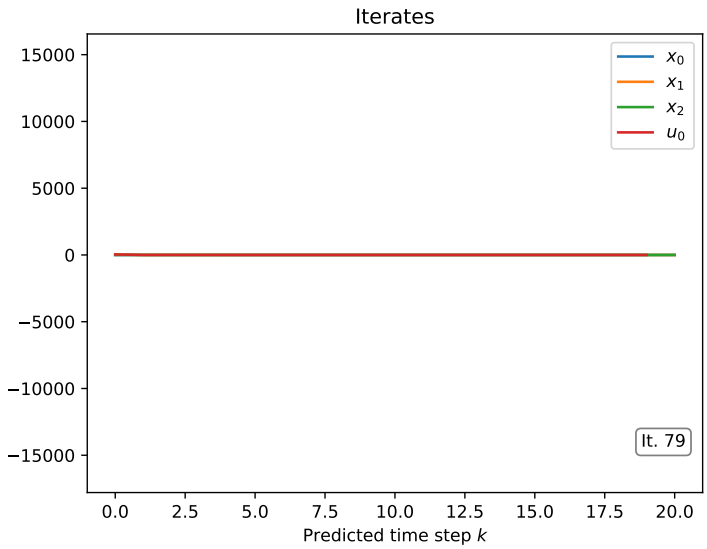


Figure 21: Final solution of the nonlinear test case using linesearch.

Assignment 6.5

Run it on the parking example. Use initial condition all zeros (except x_0 =`problem.x0`). Does it converge? Why?
Hint. During the Newton-Lagrange iterations, check whether the Hessian of the QP cost is positive definite.

The solver does not converge because the Hessian of the cost is not positive definite. This is proven by a check performed in the code which will print in yellow "Warning: Qk is not positive definite!". Figure 22 shows the trajectory of the car, it is obvius that the orgin, parking spot, is not reached.

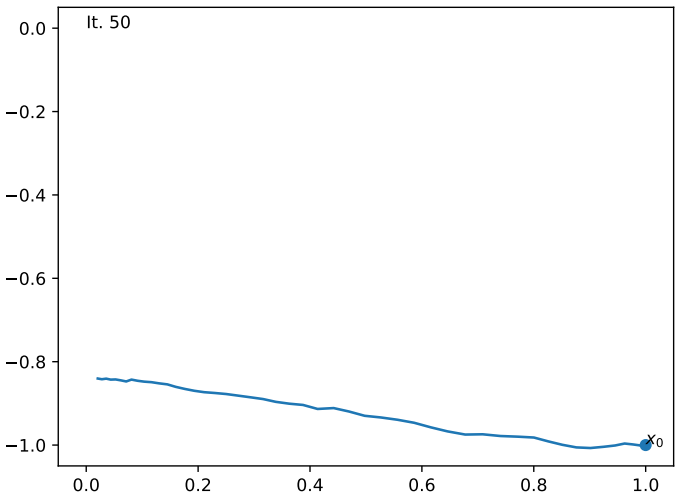


Figure 22: Final solution of the parking case without using regularization.

6.4 Adding Regularization

Assignment 6.6

Implement regularization in the function `regularize`. This method should check whether the QP is convex. If not, it should add λI to the Hessian $\overline{Q}$ of the cost function, where λ is (approximately) the smallest constant such that $\overline{Q} + \lambda I$ is positive definite. To do so set $\lambda = 10^{-6}$ and double it every time the result is not positive definite.
With this modification, run the method again on the Parking problem. Does it converge?
Hint. Make sure to set `cfg.regularize` to `True`!

With the regularization implemented, the solver converges in 57 iterations. The final solution is shown in Figure 23 and the trajectory followed by the car to reach the parking spot is shown in 24.

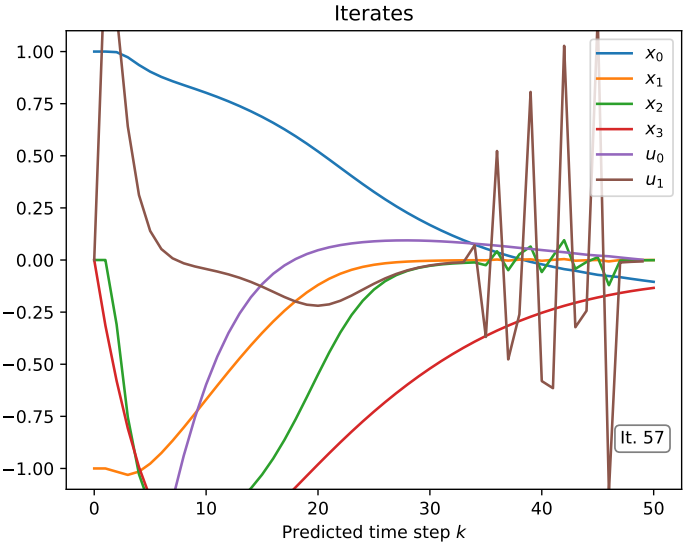


Figure 23: Final solution of the parking case using regularization.

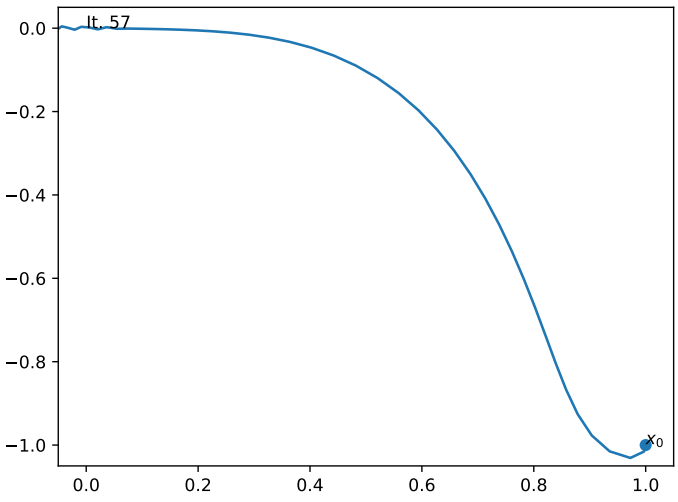


Figure 24: Final solution of the parking case using regularization.