H0E76A: Model Predictive Control
— Homework assignments —

Matties Roofthooft

2025–2026

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

> **Assigment 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`.

$$K_\infty = \begin{bmatrix} 1.28645067 & 2.31256493 \end{bmatrix} \tag{1}$$

Figure 1 shows that increasing the prediction horizon N, makes the trajectory converge towards the infinite horizon trajectory (drawn in black).

At each time step in MPC K_N is used to calculate the input, the sequence K_N convergence towards K_∞ so therefore it logically the closed-loop trajectories will converge as well.
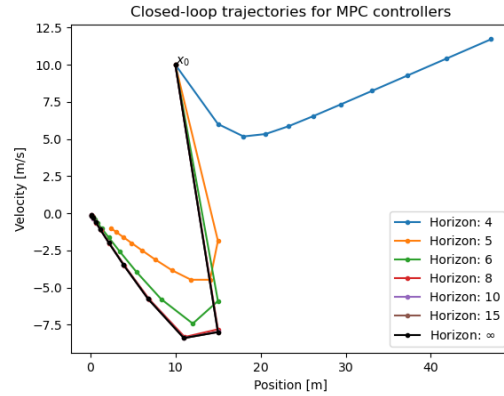
Figure 1: Closed loop trajectories of MPC controllers

Assigment 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 shows that the approximation of the sum bounds from above while the other estimation bounds from below. Both quantities converge towards the infinite horizon cost which is $V_\infty = 1006.40329$.
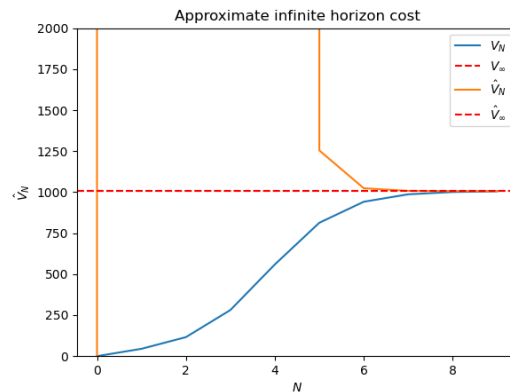

Figure 2: Approximation of infinite horizon cost

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 behaviour of the system under the applied controls.

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

Assigment 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.

Since the change in velocity each timestep is $T_s u_{min}$ the minimal timestep to stationarity is:

$$T = \left\lceil \frac{-v_0}{T_s u_{min}} \right\rceil \tag{2}$$

Assigment 2.2

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

$$p_{k+1} = p_k + T_s v_k \tag{3}$$

$$\Delta p_{k+1} = T_s v_k \tag{4}$$

We calculate distance travelled as $d = p_T - p_0$ as the sum of the position increments during the timesteps until a stationary velocity is reached:

$$d = \sum_{k=0}^{T} \Delta p_{k+1} = \sum_{k=0}^{T} T_s v_k = T_s \sum_{k=0}^{T} (v_0 + k \, T_s u_{min}) , \tag{5}$$

$$d = T_s v_0 \, T + T_s^2 u_{min} \sum_{k=0}^{T} k = T \, v_0 T_s + u_{min} T_s^2 \left(\frac{T+1}{2} \right). \tag{6}$$

Assigment 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 $S = \{x \mid p \leq p_{max}\}$.


The breaking distance given initial velocity v_0 should be lower than the distance to the wall to prevent a crash.

$$d(v_0) \leq p_{max} - p_0 \tag{7}$$

Assigment 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.
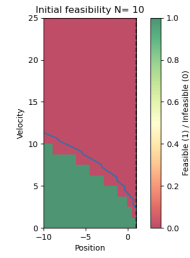

For all $N \in \{2, 5, 10\}$, make a plot showing the set of states that are feasible (this can be a simple scatter plot), and draw the boundary of the constraint you derived in Assigment 3.
Remark. You only have to check the initial feasibility of the MPC problem, not recursive feasibility.
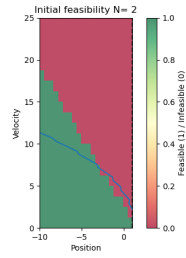
(a)
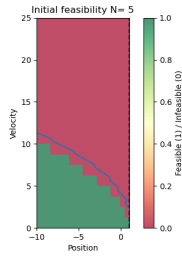
(b)

(c)

Figure 3

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.

> **Assigment 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
>
> $$\text{lev}_\alpha \, V_\infty = \{x \mid V_\infty(x) = x^\top P x \le \alpha\}$$
>
> 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
>
> $$\begin{cases} x \in X \\ Kx \in U \end{cases} \quad \forall x \in \text{lev}_\alpha \, V_\infty,$$
>
> where $X = \{x \in \mathbb{R}^n \mid H_x x \le h_x\}$ and $U = \{u \in \mathbb{R}^m \mid H_u u \le h_u\}$ are the (polyhedral) sets of feasible states and inputs, respectively.
> **Hint.** The notion of a support function might be of use.

The section of both sets remains a polyhedral set.

$$X_\kappa = \{x \in \mathbb{R}^n \mid H_x x \le h_x; \quad H_u K x \le h_u\} \tag{8}$$

$$X_\kappa = \{x \in \mathbb{R}^n \mid Hx \le h\} \tag{9}$$

With $H = \begin{bmatrix} H_x \\ H_u K x \end{bmatrix}$ and $g = \begin{bmatrix} h_x \\ h_u \end{bmatrix}$. The problem is a set inclusion problem by utilizing the support function for ellipsoids $\sigma_{\epsilon(\alpha)}(v)$ we can derive an expression.

$$\alpha^* = \min \left\{ \frac{g_i^2}{h_i^\top P^{-1} h_i} \mid i = 1, \dots, d \right\}. \tag{10}$$

With h_i every row in linear matrix H and g_i values in g of the combined set X_κ.

> **Assigment 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?

No the obtained ellipsoidal isn't a valid terminal set yet since it doesn't consider recursive feasibility but only one step.

Alternatively, we can more directly encode the requirements for the invariant set and solve a convex optimization problem to find it.

Let $E = \{x \in \mathbb{R}^n \mid x^\top P x \le 1\}$ 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 $u_t = Kx_t$.
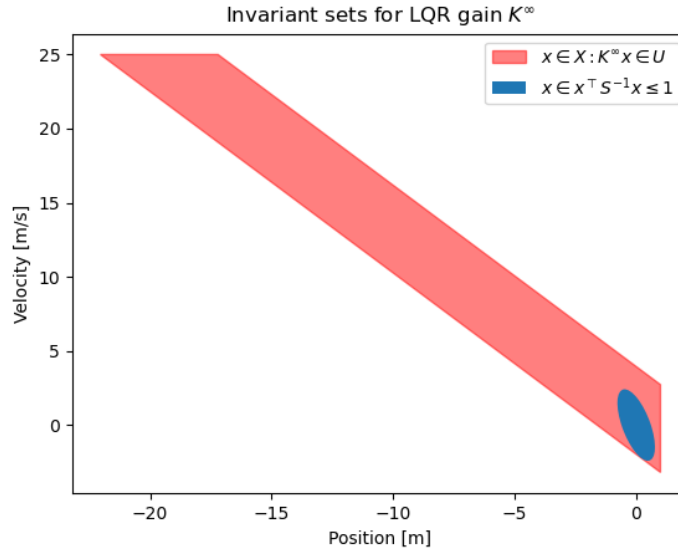
Figure 4: Ellipsoid set assignment 2

Assigment 3.3

Formulate a constraint on P and K that guarantees that

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

We can write this implication in terms of a matrix inequality since it holds $\forall x \in \mathbb{R}^\kappa$

$$(A + BK)^\top P(A + BK) \preceq P \tag{11}$$

And thus:

$$P - (A + BK)^\top P(A + BK) \succeq 0 \qquad \text{(PSD)} \tag{12}$$

Assigment 3.4

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

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

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.

Multiplying inequality from left and right by P^{-1}

$$P^{-1} - P^{-1}(A + BK)^\top P(A + BK)P^{-1} \succeq 0 \tag{13}$$

Introduce the change of variables:

$$S - S(A + BK)^\top P(A + BK)S \succeq 0 \tag{14}$$

6

$$S - S(A + BK)^\top P(AS + BF) \geq 0 \tag{15}$$

$$S - ((A + BK)S^\top)^\top P(AS + BF) \geq 0 \tag{16}$$

Since $S = S^\top$

$$S - (AS + BF)^\top P(AS + BF) \geq 0 \tag{17}$$

Applying Schur complement lemma, we arrive at the LMI:

$$M = \begin{bmatrix} S & (AS + BF)^\top \\ AS + BF & S \end{bmatrix} \geq 0 \tag{18}$$

Assigment 3.5

Recall from the lectures that

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

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

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

with $H_{x,i}$ the ith row of H_x and $h_{x,i}$ the ith coordinate of h_x (see Assigment 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)

Proof. Let the candidate invariant set be the ellipsoid

$$E = \{x \in \mathbb{R}^n \mid x^\top P x \leq 1\}, \qquad P > 0,$$

and denote $S = P^{-1}$. The state constraint set is the polyhedron

$$X = \{x \in \mathbb{R}^n \mid H_x x \leq h_x\},$$

where $H_{x,i}$ is the ith row of H_x and $h_{x,i}$ the ith component of h_x.
From the lecture result, we have

$$E \subseteq X \iff \sigma_E(H_{x,i}) \leq h_{x,i}, \qquad i = 1, \ldots, m_x,$$

where σ_E denotes the support function of E.

The support function of the ellipsoid $E = \{x \mid x^\top P x \leq 1\}$ is

$$\sigma_E(v) = \|P^{-1/2} v\|_2 = \sqrt{v^\top P^{-1} v} = \sqrt{v^\top S v}.$$

Applying this with $v = H_{x,i}^\top$ gives

$$\sigma_E(H_{x,i}) = \sqrt{H_{x,i}^\top S H_{x,i}}.$$

Thus the inclusion condition becomes

$$\sqrt{H_{x,i}^\top S H_{x,i}} \leq h_{x,i}, \qquad i = 1, \ldots, m_x.$$

Since both sides are nonnegative, squaring yields an equivalent constraint:

$$H_{x,i}^\top S H_{x,i} \leq h_{x,i}^2, \qquad i = 1, \ldots, m_x.$$

This proves the required equivalence. $\qquad\qquad\square$

Proof. Consider the control law

$$u = Kx, \qquad x \in E = \{x \in \mathbb{R}^n \mid x^\top P x \leq 1\}, \quad P > 0,$$

and let $S = P^{-1}$ and $F = KS$. The input constraint set is

$$U = \{u \in \mathbb{R}^{n_u} \mid H_u u \leq h_u\}, \quad H_{u,i} \text{ is the } i\text{th row of } H_u, \ h_{u,i} \text{ its } i\text{th entry.}$$

We require that all inputs satisfy the constraints:

$$H_{u,i} u = H_{u,i} Kx \leq h_{u,i}, \quad \forall x \in E.$$

The maximum value along each direction $H_{u,i}$ is given by the support function of the ellipsoid:

$$\max_{x \in E} H_{u,i} Kx = \sqrt{(H_{u,i} KS) P (H_{u,i} KS)^\top} \leq h_{u,i}.$$

Squaring both sides and transposing scalar gives the equivalent quadratic inequality:

$$H_{u,i}^\top F P F^\top H_{u,i} \leq h_{u,i}^2, \qquad i = 1, \ldots, m_u.$$

To express this as an LMI in F and S, we apply the Schur complement lemma. For a scalar $h_{u,i}^2$ and symmetric positive definite S, the inequality

$$h_{u,i}^2 - H_{u,i}^\top F P F^\top H_{u,i} \geq 0$$

is equivalent to

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

Thus, the input constraints for all $x \in E$ can be enforced by the following LMI for all $i = 1, \ldots, m_u$:

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

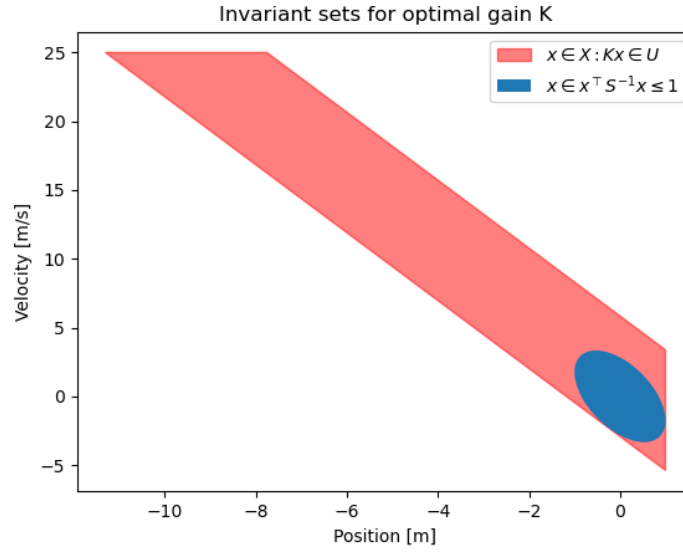with decision variables F and S. $\qquad \square$

Figure 5: Ellipsoid set

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.
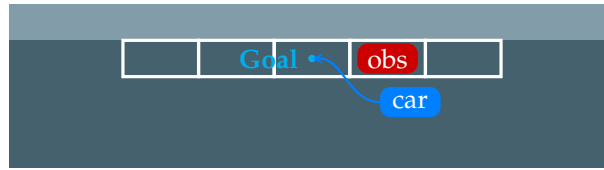

Figure 6: Autonomous parking task with obstacle

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. 7. 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.
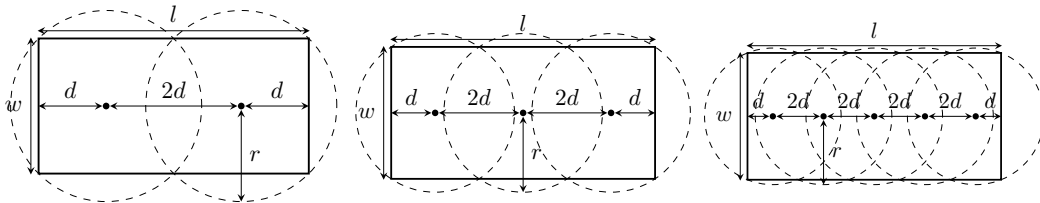

Figure 7: Vehicle covering with 2, 3 and 5 circles respectively.

Assigment 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.

We first calculate the horizontal distance d.

$$l = 2n_c d \tag{19}$$

$$d = \frac{l}{2n_c} \tag{20}$$

The centers of the circles then become

$$\mathbf{c}_i = \begin{pmatrix} -l/2 + d(1 + 2i) \\ 0 \end{pmatrix} \quad \text{for } i = 0, \ldots, n_c - 1 \tag{21}$$

Using pythagoren theorem we obtain:

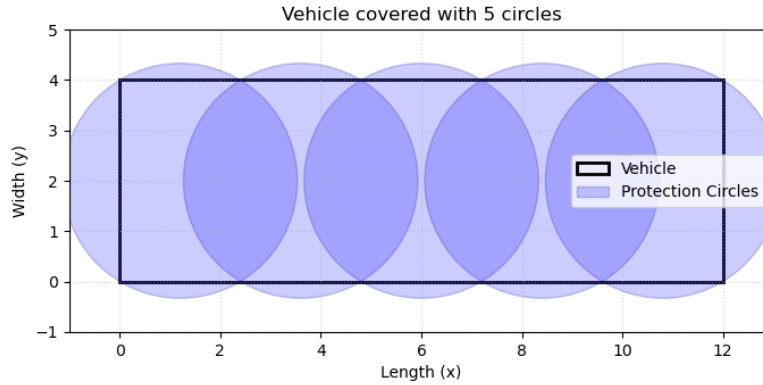$$r = \sqrt{\left(\frac{w}{2}\right)^2 + d^2} \tag{22}$$

Figure 8: Vehicle covered with 5 circles.

Assigment 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.

The center of a circle in global coordinates, $\bar{\mathbf{c}}$, is given by:

$$\bar{\mathbf{c}} = \mathbf{p} + \mathbf{R}(\psi)\mathbf{c} \tag{23}$$

where $\mathbf{p} = \begin{pmatrix} p_x \\ p_y \end{pmatrix}$ is the vehicle's global position, and $\mathbf{R}(\psi)$ is the 2D rotation matrix:

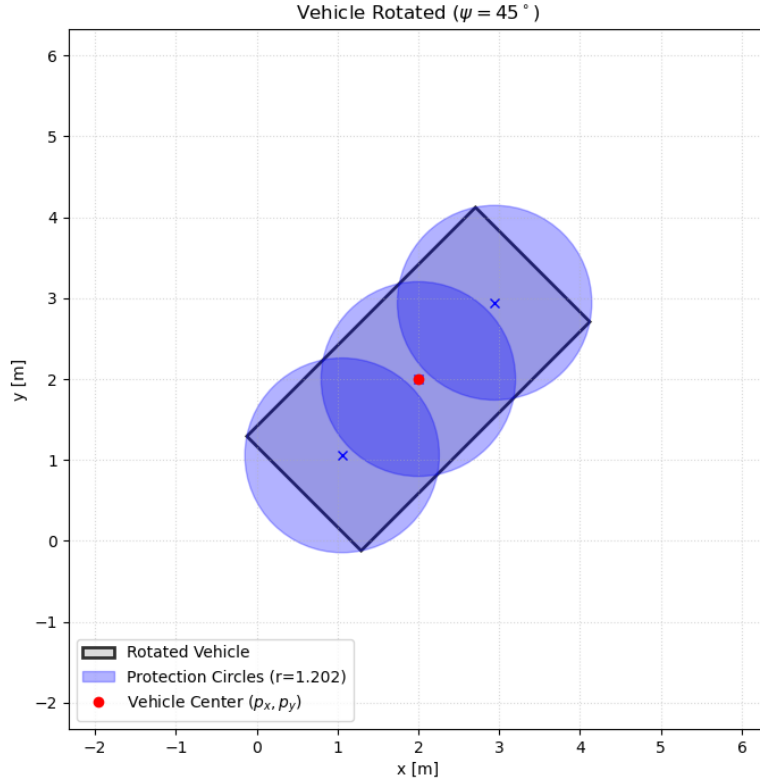$$\mathbf{R}(\psi) = \begin{pmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{pmatrix} \tag{24}$$

Figure 9: Rotated vehicle.

Assigment 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.

The vehicle will not crash if none of its circles overlap with the ones of the obstacle vehicle. Mathematically we can formulate this as follows:

$$\left\| \bar{c}_i - \bar{c}_{j'} \right\|_2 \geq r + r' \quad i, j \in 0, \ldots, n_c - 1 \tag{25}$$

Squaring gives:

$$g(\bar{c}_i, \bar{c}_{j'}) = (r + r')^2 - \left\| \bar{c}_i - \bar{c}_{j'} \right\|_2^2 \leq 0 \quad i, j \in 0, \ldots, n_c - 1 \tag{26}$$

The set is defined as the outerior of a disk. The set is nonconvex, since the set is not closed under convex combination.

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

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

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)`

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.

Without tuning the controller parameters the car wasn't able to park fully enclosed in the parking spot. The vertical positioning of the car was off. To punish this more we increased the terminal cost of the y-component. It is important to increase the terminal cost of the x-component of the controller as well. Otherwise the car decides to park before the other car instead of behind it. The tuned controller parameters are the following:

$$Q = \text{diag}(1, 3, 0.4, 0.001) \tag{27}$$

$$Q_f = 5 \cdot Q + \text{diag}(250, 3 \cdot 10^3, 100, 10) \tag{28}$$

$$R = \text{diag}(1.0, 10^{-4}) \tag{29}$$

$$N = 30 \quad n_c = 3 \tag{30}$$

The trajectory of the parking maneuver is shown in Figure 10.

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:

```python
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.
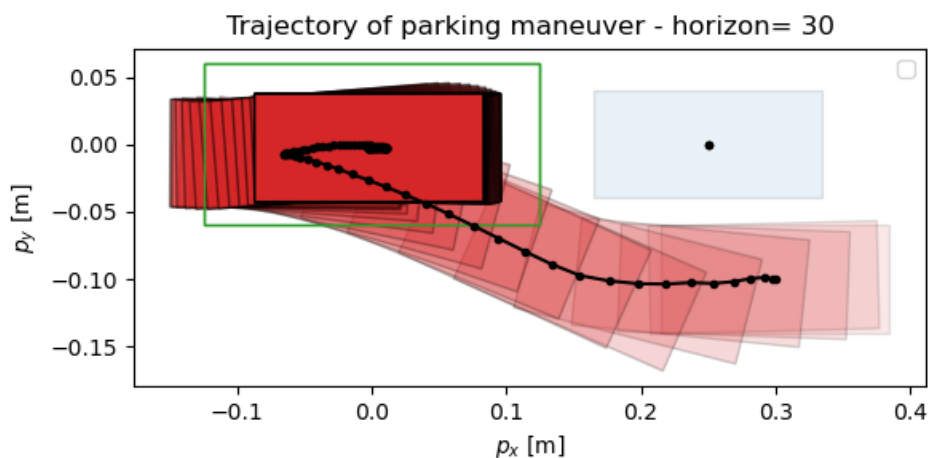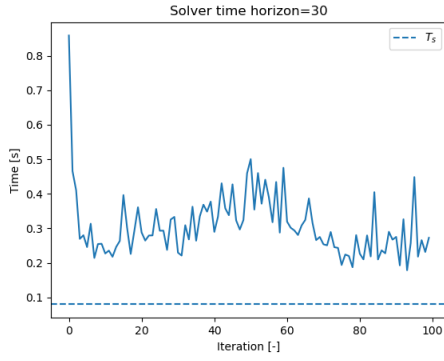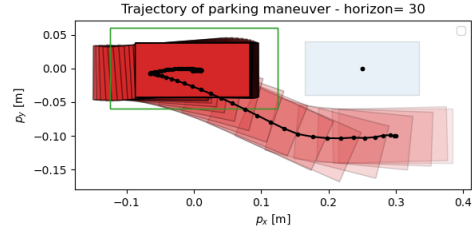

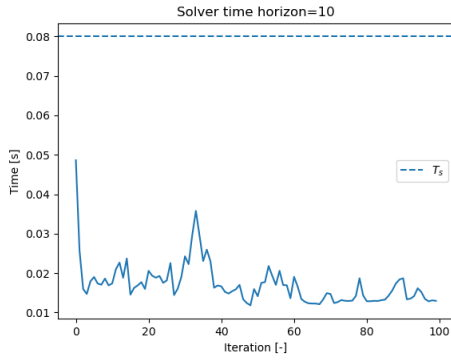Figure 10: Trajectory of parking maneuver

(a) Solver Time vs. Iteration

(b) Trajectory of Parking Maneuver

Figure 11: Analysis of the parking maneuver with a horizon of $N = 30$. (a) Solver time per iteration. (b) Optimal vehicle trajectory.

Decreasing the prediction horizon resolves this issue. The MPC controller is feasible in real-time and is still able to park fully enclosed in the parking spot.
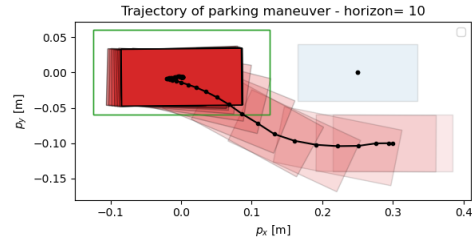

(a) Solver Time vs. Iteration

(b) Trajectory of Parking Maneuver

Figure 12: Analysis of the parking maneuver with a horizon of $N = 10$. (a) Solver time per iteration. (b) Optimal vehicle trajectory.

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.

Assigment 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. Initialize your EKF with the same values as during the exercise session. 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 behaviour. Plot the estimated trajectories.

4. Does clipping the state estimates in the *EKF* aid the performance? Plot the result.

First the EKF has to be initialized. We need to store $N + 1$ past estimates and past covariance estimates.

The results are shown in Figures **??** and 14 for a moving horizon of 10 and 25 respectively. The estimates are rapidly converging towards the actual states which indicate the improved performance. Clipping the state estimates slightly improves the performance as well, especially for the lower horizon case.
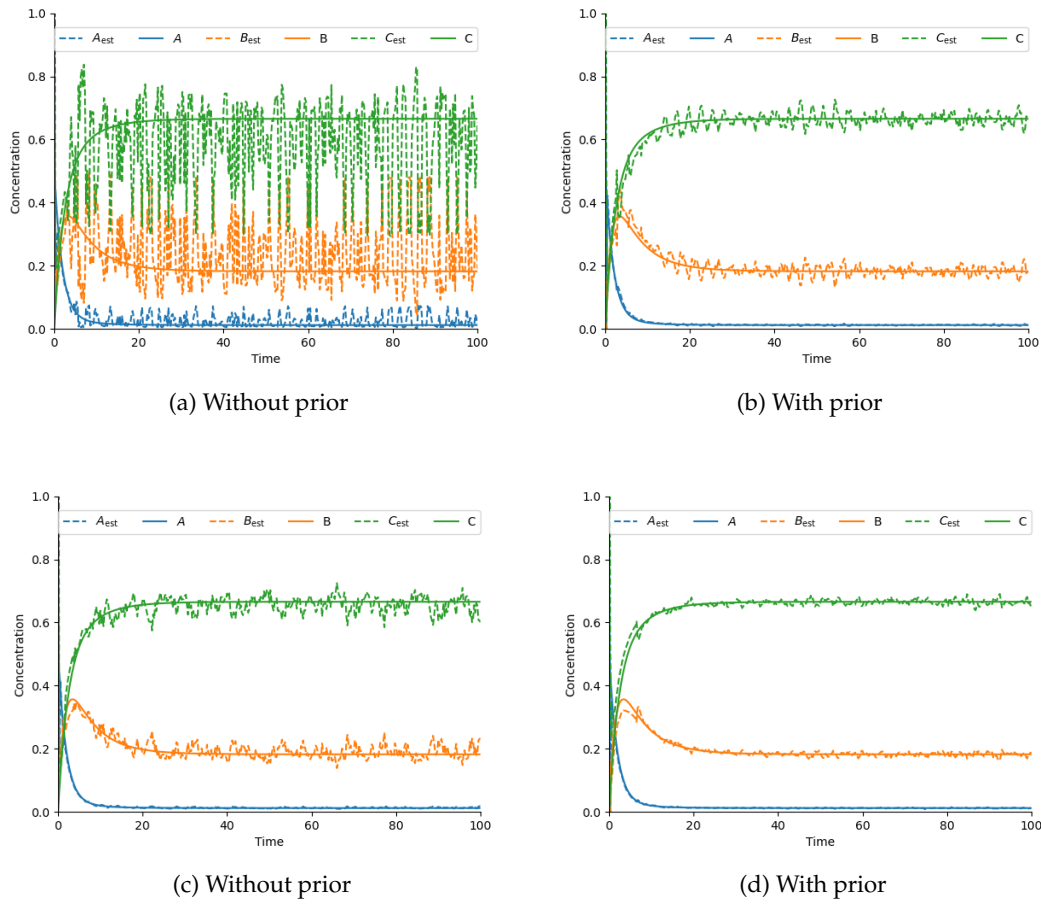
(a) Without prior

(b) With prior

(c) Without prior

(d) With prior

Figure 13: MHE with prior filtering for horizon 10 (upper) and horizon 25 (lower) without clipping.

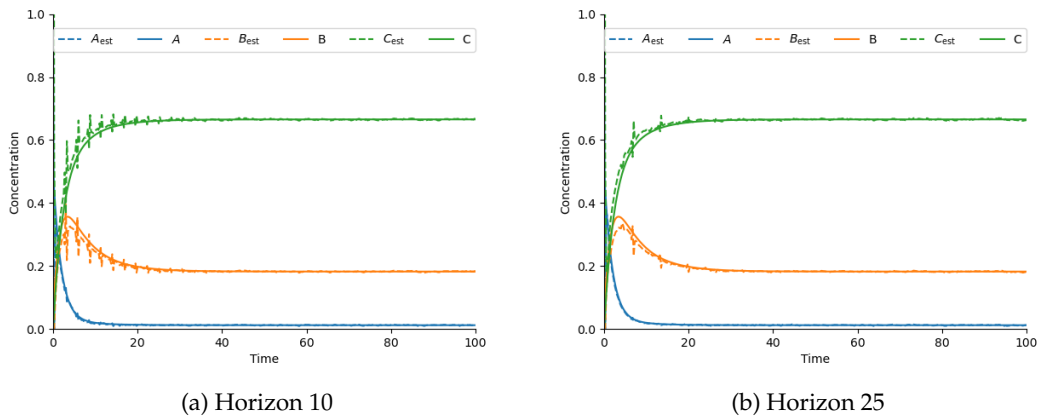

(a) Horizon 10

(b) Horizon 25

Figure 14: MHE with prior filtering and clipping enabled.

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).

Assigment 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?

```
it.    0 | JN = 1.51e+04 | ||h||2 = 1.15e-13
it.    1 | JN = 1.51e+04 | ||h||2 = 1.35e-13
Converged
```

The output is as expected. For a problem with linear dynamics and quadratic cost, the Newton-Lagrange QP coincides exactly with the original optimal control problem. Therefore, the Newton step computes the exact solution in a single iteration.
The solver performs one additional iteration because the stopping criterion is based on the norm of the update Δu_k, which becomes zero after the optimal solution has been reached.

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

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

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`.

Assigment 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?

The constraint violation decreases rapidly over the iterations, indicating that the discretized state dynamics (multiple-shooting constraints) are enforced correctly. Figure 15 shows the predicted state and input trajectories of the final iterate. The resulting trajectories remain bounded and

converge to zero, which suggests that the obtained solution is physically meaningful and satisfies the local optimality conditions. Since the states and inputs converge to zero, the regulation problem is successfully solved.

The iteration log clearly demonstrates the locally quadratic convergence of the constraint violation, as expected for the Newton-Lagrange method.

```
it.   0 | JN = 2.26e+02 | ||h||2 = 2.24e-01
it.   1 | JN = 2.26e+02 | ||h||2 = 2.11e-02
it.   2 | JN = 2.26e+02 | ||h||2 = 7.45e-05
it.   3 | JN = 2.26e+02 | ||h||2 = 9.11e-08
it.   4 | JN = 2.26e+02 | ||h||2 = 1.17e-10
Converged
```
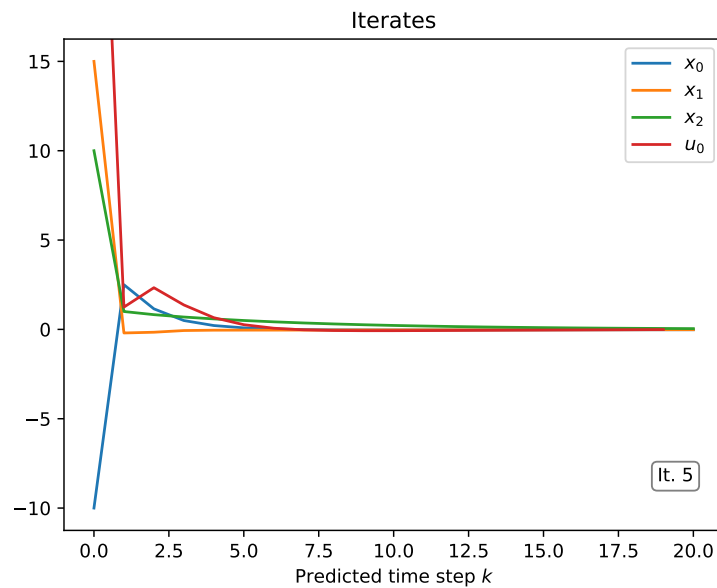

Figure 15: Predicted time evolution after 6 iterations.

Assigment 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?

The solver fails to converge, as shown in Figure 17, because the open-loop trajectory used as an initial guess (Figure 16) is far from the optimal solution obtained in the previous exercise. Due to the unstable open-loop dynamics, this trajectory quickly diverges, resulting in a poor initial guess. As a consequence, the local linearization of the dynamics provides a poor approximation of the true nonlinear system behaviour.

Since the Newton-Lagrange method relies on local second-order information, taking full Newton steps from such an initial guess leads to large and inaccurate updates, causing the iterates to diverge.

This illustrates the local nature of Newton-type methods and the importance of globalization techniques such as line search.

This behaviour is also reflected in the iteration log, where both the cost and the constraint violation oscillate and occasionally grow by several orders of magnitude, indicating the absence of a descent property without globalization.

```
it.    0 | JN = 2.86e+05 | ||h||2 = 5.00e+03
it.    1 | JN = 2.91e+04 | ||h||2 = 9.58e+02
it.    2 | JN = 2.71e+03 | ||h||2 = 2.24e+02
it.    3 | JN = 3.40e+05 | ||h||2 = 8.05e+04
it.    4 | JN = 1.85e+05 | ||h||2 = 2.06e+04
it.    5 | JN = 3.93e+04 | ||h||2 = 6.15e+03
it.      ----------------------------------
it.   47 | JN = 3.09e+05 | ||h||2 = 4.96e+04
it.   48 | JN = 4.53e+05 | ||h||2 = 2.10e+04
it.   49 | JN = 2.21e+05 | ||h||2 = 1.01e+04
it.   50 | JN = 5.35e+04 | ||h||2 = 3.25e+03
it.   51 | JN = 1.30e+07 | ||h||2 = 1.12e+06
Diverged
```
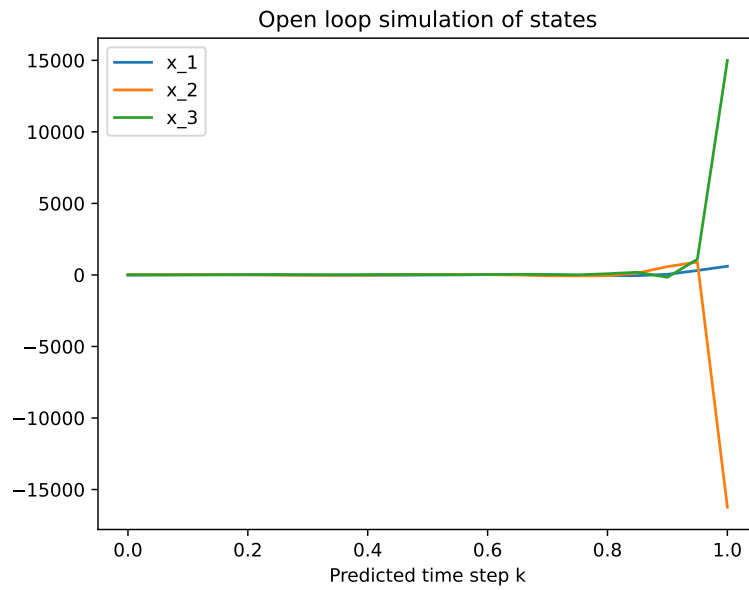

Figure 16: Openloop simulation of the states for zero input policy

6.3 Adding Line Search

Assigment 6.4

Implement line search and the experiment of Assigment 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 Assigment 2? How do you explain this?

Figure 18 shows the output of the method at the final iterate with linesearch enabled. With line search enabled, the method converges after 65 iterations to the same solution as in Assignment 2. The line search enforces sufficient decrease of the merit function

$$\phi(z) = J_N(z) + c \, \|h(z)\|_1 \tag{31}$$

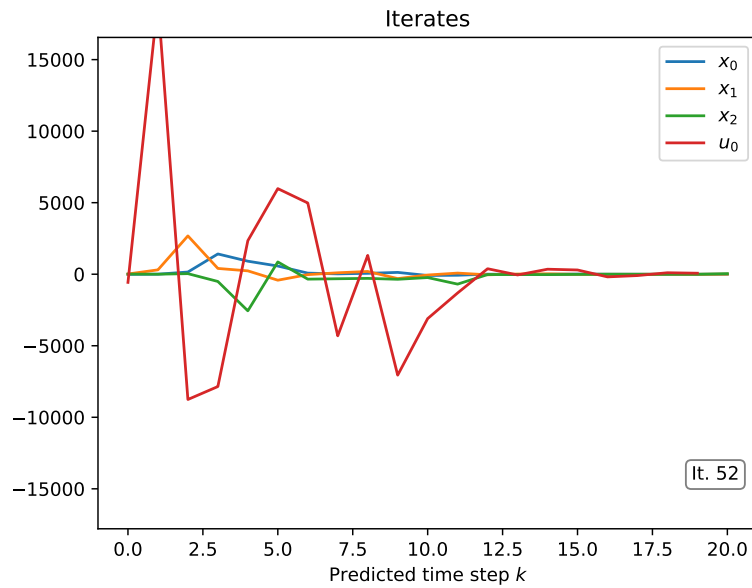which prevents overly aggressive Newton steps when the linearization is inaccurate.

19

Figure 17: Diverged solution without linesearch.

As a result, the solver is able to make progress even when starting far from the solution, illustrating the importance of globalization strategies in nonlinear optimization.

From the iteration log, it can be observed that the constraint violation decreases monotonically once the line search becomes effective, while the cost converges more gradually, which is typical for merit-function-based globalization. Notably, around iteration 62, as the iterate approaches the local optimum, the constraint violation begins to decrease at a locally quadratic rate, as expected for the Newton–Lagrange method, following the initial slower progress caused by globalization.

```
it.    0 | JN = 2.86e+05 | ||h||2 = 5.00e+03
it.    1 | JN = 2.91e+04 | ||h||2 = 9.58e+02
it.    2 | JN = 2.71e+03 | ||h||2 = 2.24e+02
it.    3 | JN = 2.72e+03 | ||h||2 = 2.24e+02
it.    4 | JN = 4.73e+03 | ||h||2 = 1.49e+02
it.    5 | JN = 1.01e+03 | ||h||2 = 3.22e+01
it.      --------------------------------
it.   61 | JN = 2.27e+02 | ||h||2 = 6.36e-01
it.   62 | JN = 2.26e+02 | ||h||2 = 1.43e-01
it.   63 | JN = 2.26e+02 | ||h||2 = 1.16e-02
it.   64 | JN = 2.26e+02 | ||h||2 = 1.47e-04
it.   65 | JN = 2.26e+02 | ||h||2 = 1.64e-07
Converged
```

> **Assigment 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.

Figure 19 shows the final iterate of the optimization process without regularization. The predicted state trajectories are highly non-smooth, and the resulting trajectory does not converge to a feasible parking maneuver.

The method fails to converge because the Hessian of the QP cost is frequently indefinite, resulting in nonconvex quadratic subproblems. In this situation, the Newton direction may correspond to a saddle point or even an ascent direction for the cost function, which prevents convergence despite
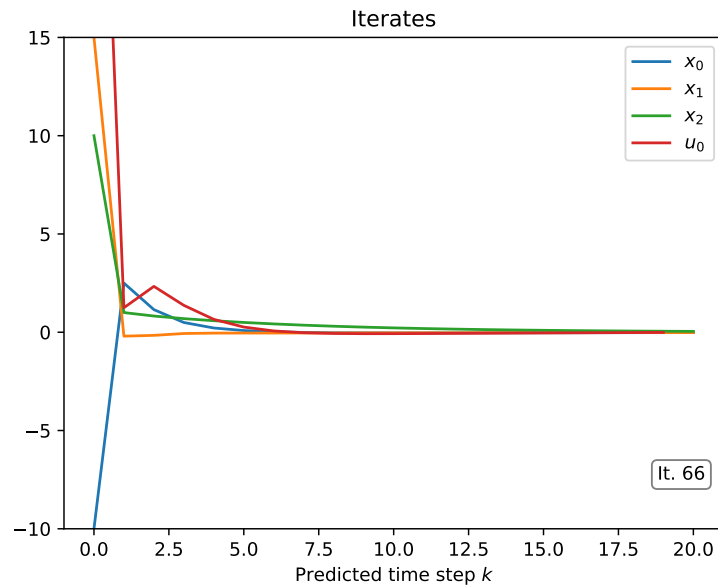
20

Figure 18: Converged solution with linesearch.

the use of line search. This behaviour is confirmed by the negative smallest eigenvalues of the QP Hessians shown in Figure 20.

The iteration log further illustrates this issue: both the cost and the constraint violation exhibit oscillatory behaviour and do not decrease monotonically, indicating that the Newton directions are unreliable when the QP Hessian is indefinite.

```
it.    0 | JN = 2.77e+02 | ||h||2 = 6.66e-16
it.    1 | JN = 2.77e+02 | ||h||2 = 5.12e-15
it.      -----------------------------------
it.   10 | JN = 2.71e+02 | ||h||2 = 1.25e-02
it.   11 | JN = 2.66e+02 | ||h||2 = 2.26e-02
it.      -----------------------------------
it.   89 | JN = 2.13e+02 | ||h||2 = 1.04e-01
it.   90 | JN = 2.14e+02 | ||h||2 = 1.04e-01
it.      -----------------------------------
it.  101 | JN = 2.19e+02 | ||h||2 = 7.42e-02
it.  102 | JN = 2.19e+02 | ||h||2 = 7.42e-02
it.      -----------------------------------
it.  140 | JN = 1.68e+02 | ||h||2 = 1.02e-01
it.  141 | JN = 1.66e+02 | ||h||2 = 1.02e-01
it.      -----------------------------------
it.  198 | JN = 1.08e+02 | ||h||2 = 1.04e-01
it.  199 | JN = 1.08e+02 | ||h||2 = 1.04e-01
Maximum number of iterations exceeded
```

6.4 Adding Regularization

Assigment 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!

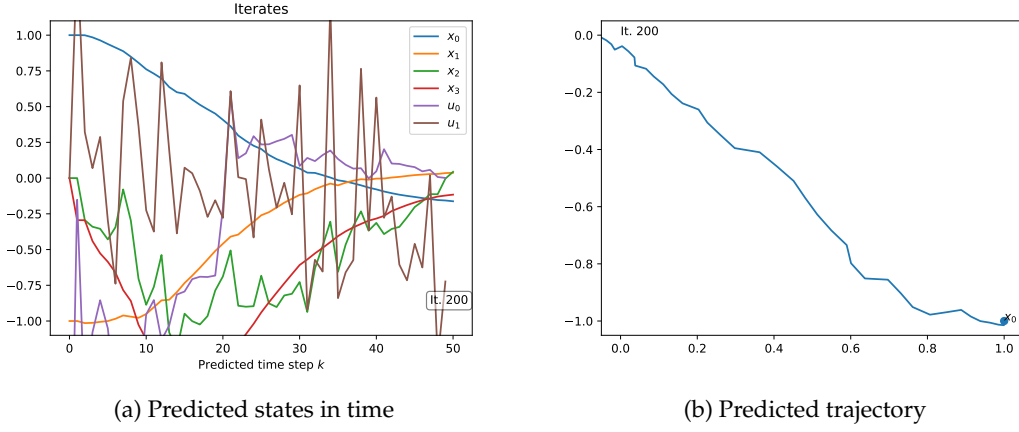(a) Predicted states in time   (b) Predicted trajectory

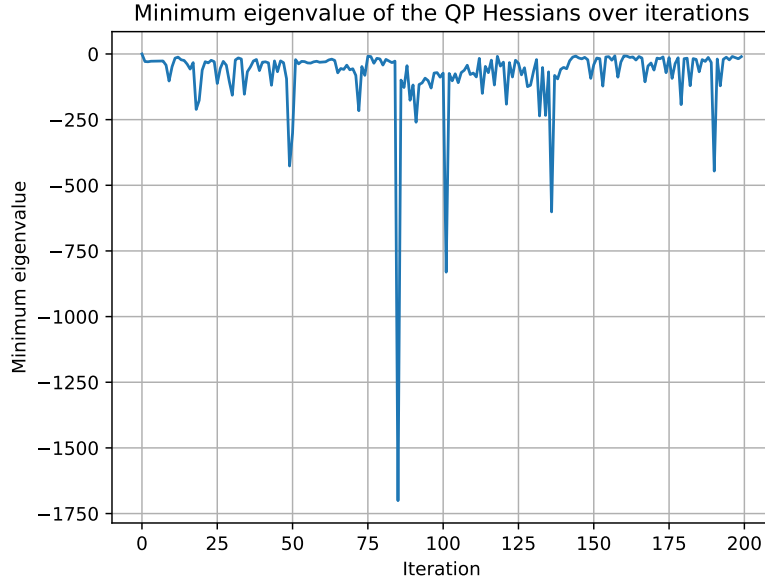Figure 19: Final iterates of the method without regularization.

Figure 20: Smallest eigenvalues of QP Hessians.

By adding a multiple of the identity matrix to the Hessian, all QP subproblems are rendered strictly convex. This guarantees that each Newton-Lagrange step is well-defined and corresponds to a descent direction for the merit function.

As a result, the solver converges reliably to a feasible and smooth solution, as shown in Figure 21. The smallest eigenvalues of the regularized Hessians remain positive throughout the iterations (Figure 22), confirming convexity of all QP subproblems.

After regularization, the iteration history shows a steady decrease in both the cost and the constraint violation, indicating that the QP subproblems are well-conditioned and the Newton directions remain reliable. However, regularization perturbs the Hessian of the Lagrangian, destroying the locally quadratic convergence of the Newton-Lagrange method and resulting in superlinear convergence instead.
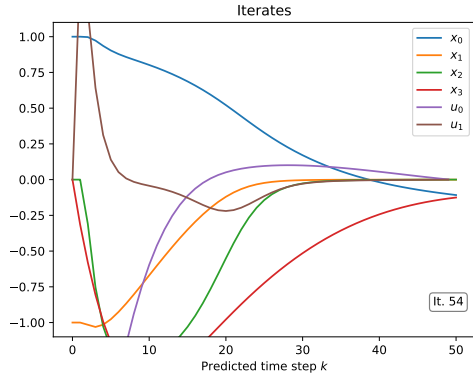
```
it.    0 |  JN = 2.77e+02 |  ||h||2 = 6.66e-16
it.    1 |  JN = 2.31e+02 |  ||h||2 = 6.92e-02
it.    2 |  JN = 1.70e+02 |  ||h||2 = 2.10e-01
it.    3 |  JN = 1.21e+02 |  ||h||2 = 1.42e-01
it.    4 |  JN = 1.01e+02 |  ||h||2 = 1.06e-01
it.    5 |  JN = 9.70e+01 |  ||h||2 = 8.31e-02
it.    ----------------------------------
```
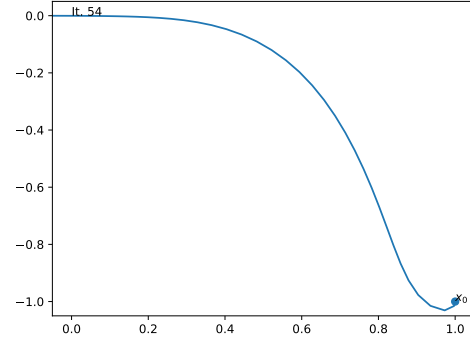
```
it.   49 | JN = 8.33e+01 | ||h||2 = 2.37e-08
it.   50 | JN = 8.33e+01 | ||h||2 = 1.82e-08
it.   51 | JN = 8.33e+01 | ||h||2 = 1.40e-08
it.   52 | JN = 8.33e+01 | ||h||2 = 1.07e-08
it.   53 | JN = 8.33e+01 | ||h||2 = 8.23e-09
Converged
```


(a) Predicted states in time   (b) Predicted trajectory

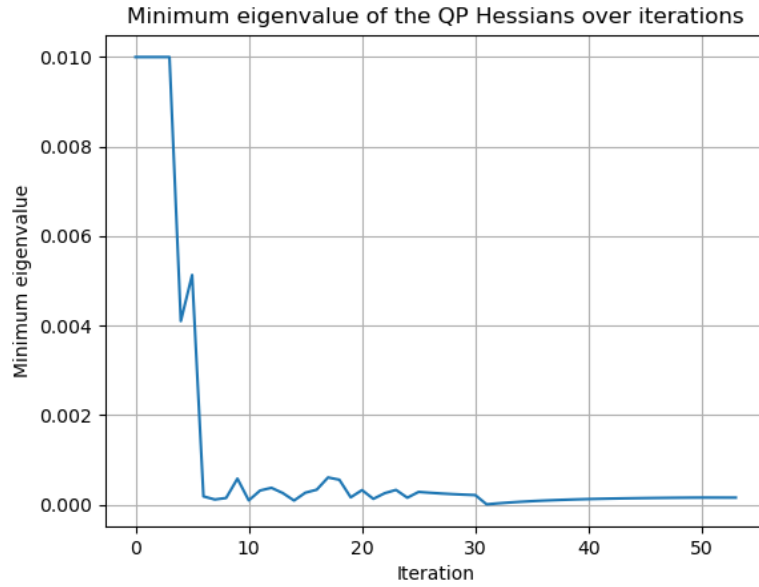Figure 21: Final iterates of the method with regularization.

Figure 22: Smallest eigenvalues of regularized QP Hessians.