H0E76A: Model Predictive Control

— Homework assignments —

Muhammad Saad Bin Amin (r1016396)

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`.

Solution

- The K_∞ controller is implemented by solving the DARE using `scipy.linalg.solve_discrete_are` for P. Then K_∞ is calculated by

$$K_\infty = -(R + B^T P B)^{-1} B^T P A$$

The value for K_∞ is [1.2864, 2.3126]

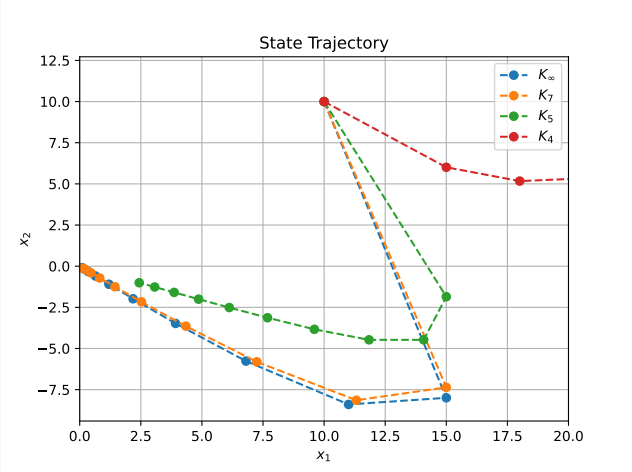

Figure 1: State trajectories for different controllers starting at $(10, 10)$

- Figure 1 shows that increasing the horizon N brings the trajectories closer and closer to the infinite horizon controller's trajectory. For $N < 5$ the controllers are unstable as can be seen.
- The recursion relation used earlier is

$$P_+ = Q + A^T(P - PB(R + B^TPB)^{-1}B^TP)A$$

  As N is increased, the recursion converges to the DARE given by

$$P = Q + A^T(P - PB(R + B^TPB)^{-1}B^TP)A$$

  This behavior is because by increasing N the finite horizon cost can approximate the infinite horizon cost. For a stabilizing control sequence, the stage costs decrease to zero exponentially so the Riccati recursion reaches a steady state. Thus adding more terms does not change the cost significantly and convergence occurs.

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.

Solution

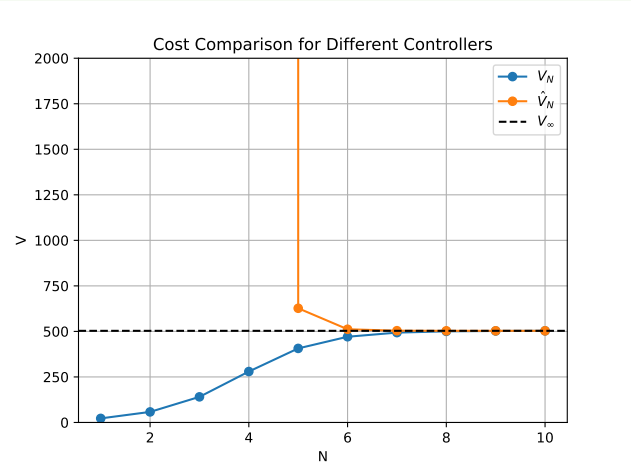

Figure 2: Optimal cost and Infinite Horizon cost for different horizons for $x_0 = (10, 10)$

It can be seen that the Optimal cost converges to the infinite horizon cost from below as N increases whereas the total infinite horizon cost converges from above. This is because increasing N adds more terms of stage cost $l(x, u) \geq 0$ to V_N which for a stable system go to zero exponentially. So as $N \to \infty$, $V_N \to V_\infty$ from below. For the approximated infinite horizon cost $\hat{V}_N$, small N do not result in stabilizing sequence so stage cost grows exponentially. Increasing N makes the system stable making the stage costs smaller. Therefore as $N \to \infty$, $\hat{V}_N \to V_\infty$ from above.

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.

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.

Solution

Based on the provided model

$$x_{k+1} = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ T_s \end{bmatrix} u_k$$

with $x = (p, v)$, the velocity equation can be written as

$$\begin{aligned} v_1 &= v_0 + u_{\min} T_s \\ v_2 &= v_1 + u_{\min} T_s = v_0 + 2u_{\min} T_s \\ &\vdots \\ v_T &= v_0 + T u_{\min} T_s \\ \Rightarrow T &= -\frac{v_0}{u_{\min T_s}} \end{aligned} \tag{1}$$

Assigment 2.2

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

Solution

Proceeding similarly as before, this time with p

$$\begin{aligned} p_1 &= p_0 + v_0 Ts \\ p_2 &= p_1 + v_1 Ts = p_0 + 2v_0 T_s + u_{\min} T_s^2 \\ &\vdots \\ p_T &= p_0 + T v_0 T_s + \sum_{k=0}^{T-1} (k) u_{\min} T_s^2 \\ \Rightarrow p_T - p_0 &= T v_0 T_s + \frac{T(T-1)}{2} u_{\min} T_s^2 \end{aligned} \tag{2}$$

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

Solution

Minimum braking distance happens when maximum brakes are applied ($u_k = u_{\min}$). Therefore under maximum braking, as calculated in the previous exercise,

$$p_T = p_0 + Tv_0T_s + \frac{T(T-1)}{2}u_{\min}T_s^2$$

$$T = -\frac{v_0}{u_{\min T_s}}$$

$$\frac{T(T-1)}{2} = \frac{v_0^2 + u_{\min}v_0T_s}{2u_{\min}^2T_s^2}$$

$$\Rightarrow p_T = p_0 - \frac{v_0^2}{2u_{\min}} + \frac{1}{2}v_0T_s \tag{3}$$

Thus equation 3 relates the maximum braking distance with initial state. Combining it with the set constraint gives us the maximal control invariant set S_∞

$$S_\infty = \{x \mid p - \frac{v^2}{2u_{\min}} + \frac{1}{2}vT_s \leq p_{\max}\} \tag{4}$$

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.

Solution

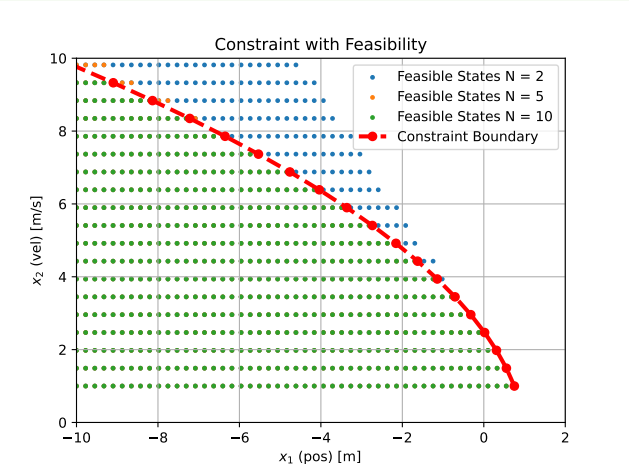

Figure 3: The feasible region for MPC controllers with different N along with the constraint boundary for S_∞

Figure 3 shows the initial feasibility for MPC controller with different prediction horizons along with the constraint boundary for calculated S_∞. As N increases the feasibility converges to the calculated region. Since this is only initial feasibility, short horizons cannot look ahead in the future far enough to detect infeasibility. Therefore their feasibility region is larger.

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

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

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.

Solution

Let X_κ be the set of all states which satisfy $x \in X$, $Kx \in U$. We want to maximize α while respecting the constraints. This can be written as the optimization problem

$$\begin{aligned} \max \quad & \alpha \\ \text{subject to} \quad & \text{lev}_\alpha V_\infty \subset X_\kappa \end{aligned}$$

Since given X and U are polyhedral sets, X_κ is also a polyhedral set. We can write

$$X_\kappa = \left\{ x \mid \underbrace{\begin{bmatrix} H_x \\ H_u K \end{bmatrix}}_{H \in \mathbb{R}^{d \times n}} x \leq \underbrace{\begin{bmatrix} h_x \\ h_u \end{bmatrix}}_{g} \right\}$$

Let $sigma_\alpha(v)$ be the support function of the ellipsoid

$$\sigma_\alpha(v) = \sqrt{\alpha} \left\| P^{-1/2} v \right\|_2$$

Then the optimization problem can be written as

$$\begin{aligned} \max \quad & \alpha \\ \text{subject to} \quad & \sigma_\alpha(h_i) \leq g_i \quad i = 1, \cdots, d \end{aligned}$$

The constraint can be expanded applying the definition of the support function. This leads to a constraint on $\alpha \leq \frac{g_i^2}{h_i^T P^{-1} h_i}$. Thus the largest alpha can be found by taking the smallest over $i = 1, \cdots, d$, i.e.

$$\alpha^* = \min \left\{ \frac{g_i^2}{h_i^T P^{-1} h_i} \mid i = 1, \cdots, d \right\}$$

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?

Solution

Figure 4 shows the computed ellipse along with the polyhedral set X_κ. Since the elliptical set is positive invariant, it is a valid terminal set for the MPC problem.

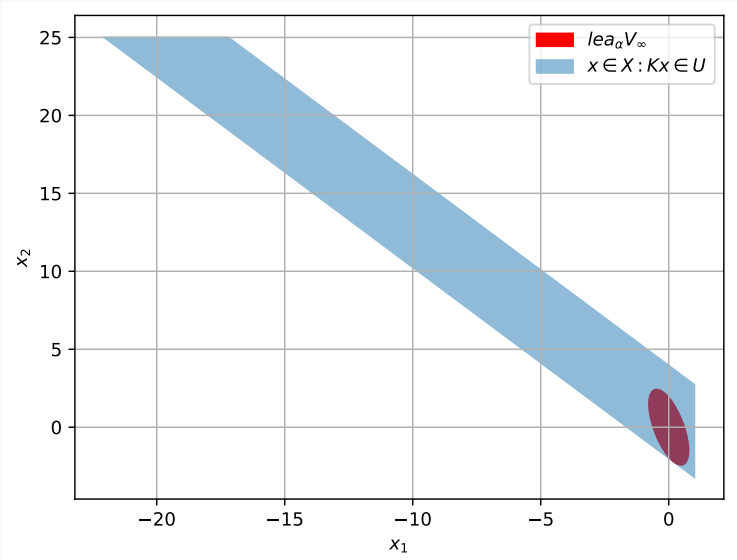

Figure 4: The set X_κ and the maximal ellipse contained in the set

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 $u_t = Kx_t$.

Assigment 3.3

Formulate a constraint on P and K that guarantees that

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

Solution

A constraint on P and K that guarantees the given requirement holds is the Lyapunov decrease constraint. i.e.

$$x^T (A + BK)^T P (A + BK) x \leq x^T P x$$

This holds when

$$(A + BK)^T P (A + BK) \leq P$$

or

$$P - (A + BK)^T P(A + BK) \geq 0$$

Assigment 3.4

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

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

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.

Solution

Multiplying LHS and RHS by P^{-1} gives,

$$P^{-1} - P^{-1}(A + BK)^T P(A + BK)P^{-1} \geq 0$$

The direction of inequality does not change since P is positive definite. Using $S = P^{-1}$, the inequality becomes

$$S - (SA + SK^T B^T)S^{-1}(AS + BKS) \geq 0$$

Using $F = KS$

$$S - (AS + BF)^T S^{-1}(AS + BF) \geq 0$$

Applying the Schur complement lemma,

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

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)

Solution

The support function of an ellipsoid with shape matrix P is given by

$$\sigma(H_{x,i}) = \sqrt{H_{x,i}^T P^{-1} H_{x,i}} = \sqrt{H_{x,i}^T S H_{x,i}}$$

Using this definition and squaring the support function constraint we get

$$H_{x,i}^T S H_{x,i} \leq h_{x,i}^2$$

Which is the required result.

Assigment 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.

Solution

The constraint on input is

$$H_{u,i}^T u \leq h_{u,i}$$

Using $u = Kx$

$$H_{u,i}^T K x \leq h_{u,i} \Rightarrow (H_{u,i} K^T)^T x \leq h_{u,i}$$

This constraint on state can be written in terms of shape matrix as derived in previous section

$$\begin{aligned} H_{u,i}^T K P^{-1} (K^T H_{u,i}) &\leq h_{u,i}^2 \\ H_{u,i}^T F P F^T H_{u,i} &\leq h_{u,i}^2 \quad (\text{Using } K = FP,\ P = P^T) \\ \Rightarrow h_{u,i}^2 - H_{u,i}^T F P F^T H_{u,i} &\geq 0 \end{aligned}$$

Using the Schur complement, we can get our LMI

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

For $i = 1, \cdots, m_u$.

Assigment 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 Assigments 4 to 6.
Implement this problem using `cvxpy` and plot the resulting ellipsoid (using the given utility code as before) together with the new polyhedral set $\{x \in \mathbb{R}^n \mid x \in X,\ FS^{-1}x \in U\}$.

Solution

The problem can be solved with `cvxpy.log_det(S)` routine. The summary of constraints required are

- One LMI constraint for stability (Assignment 4)
- m_x constraints for state constraints (Assignment 5)
- m_u LMI constraints for inputs (Assignment 6)
- Total 38 constraints for the given case

The summary of the variables in the optimization are

- S: $n_x \times n_x$ Symmetric Positive Definite matrix
- F: $n_u \times n_x$ general matrix
- Total 6 variables

The problem was solved in 75,625 iterations. From the solution, the controller matrix K and ellipsoid shape matrix P are calculated as $P = S^{-1}$ and $K = FS^{-1} = FP$. From these the new polyhedral set and ellipse are calculated. Figure 5 shows the sets together. As expected, the sets are larger than the sets obtained using the LQR solution.

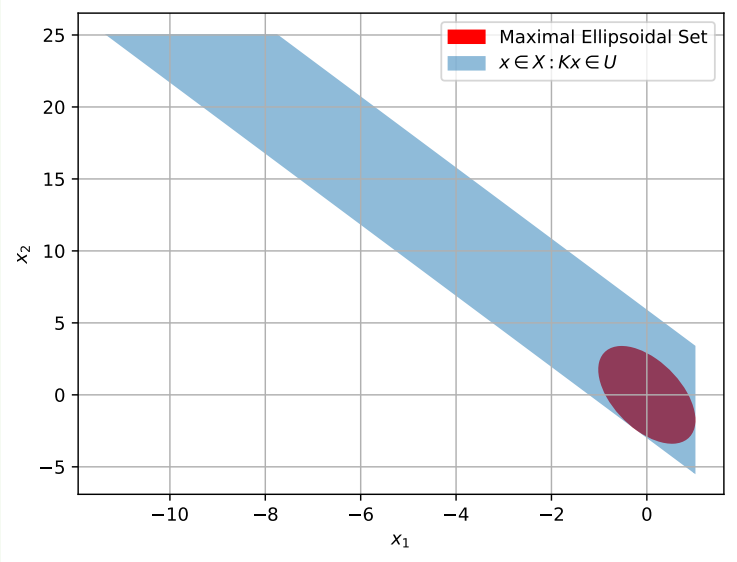

Figure 5: Positive invariant set calculated with the LMI optimization along with the maximal ellipsoidal set

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.

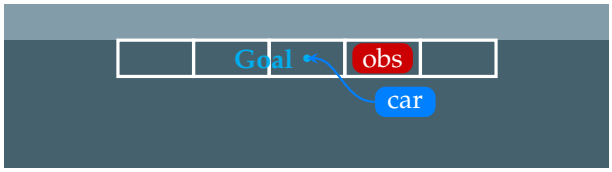

Figure 6: Autonomous parking task with obstacle

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. 8. 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.

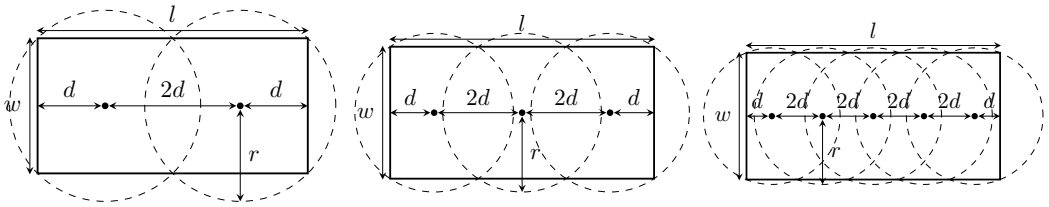

Figure 7: Vehicle covering with 2, 3 and 5 circles respectively.

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.

Solution

- **Radius Calculation:** Given that the first circle is from a distance d from the edge, radius r can be calculated as the smallest circle that contains the edge. Applying the Pythagoras theorem
$$r = \sqrt{\frac{w^2}{4} + d^2}$$
- **Edge Distance d Calculation:** The entire length l is divided into n_c segments of length $2d$. Therefore
$$d = \frac{l}{2n_c}$$
- **Center Calculation:** The centers are placed on the x axis of the origin at the vehicle center. The coordinates of the intersection of left edge and the x axis are $(-l/2, 0)$. The

centers are placed at odd multiples of d from this point. Therefore

$$c_i = \left(-\frac{l}{2} + (2i+1)d, 0\right)$$

Using the above calculations, a rectangle with $l = 4$, $w = 2$ and $n_c = 3$ is shown below.

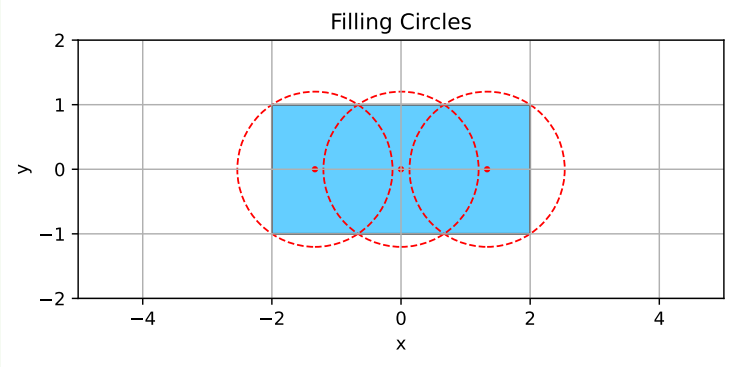

Figure 8: Application of the derived formulas

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.

Solution

The rotation matrix to rotate a point on 2D plane counter-clockwise by ψ radians is given by

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

The centers of the circles need to undergo a rotation and a shift (in that order) to go from local coordinates to global coordinates. The centers in global coordinates are given by

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

Where $p = (p_x, p_y)$ is the shift vector which points to the center of the car. Figure 9 shows the result of this rotating and shifting.

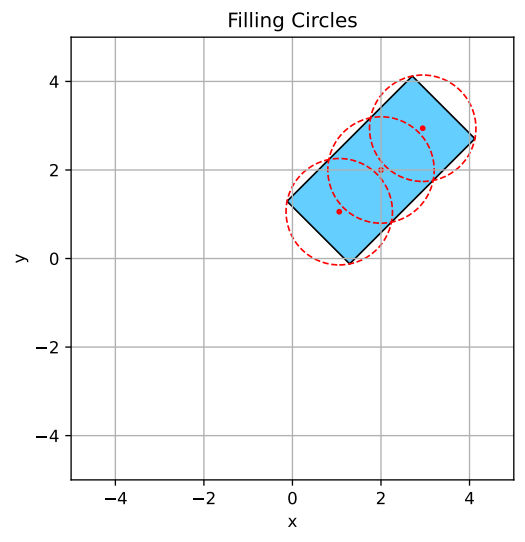

Figure 9: The centers shifted and rotated

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.

Solution

No collision translates to no overlap between circles. So, the condition can be written as

$$\|\bar{c}_i - \bar{c}'_j\|_2 \geq r - r'$$

The ℓ_2 norm is non-smooth at the origin. Squaring both sides

$$\|\bar{c}_i - \bar{c}'_j\|_2^2 - (r - r')^2 \geq 0$$

Rearranging to get the desired form

$$g(\bar{c}_i, \bar{c}'_j) = (r - r')^2 - \|\bar{c}_i - \bar{c}'_j\|_2^2 \leq 0$$

Since the obstacles do not move, there centers are constant. Visually a negative sign on the quadratic function means it is not convex. However to confirm, the Hessian w.r.t $\bar{c}_i$ is

$$\nabla^2 g = \begin{bmatrix} -2 & 0 \\ 0 & -2 \end{bmatrix}$$

Which has negative eigenvalues thus the constraint is not convex.

Assigment 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)

Solution

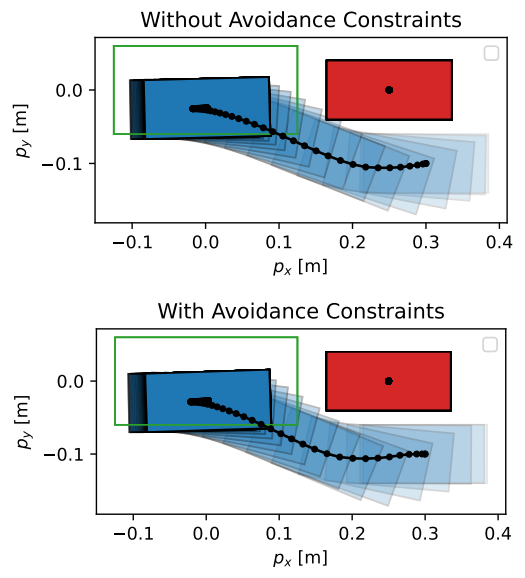

Figure 10: Obstacle avoidance with the constraint

The derived constraint was implemented in the MPC controller. Weights of the controller were $Q = \text{diag}(1, 3, 0.1, 0.01)$, $Q_N = 5Q$, $R = \text{diag}(1, 0.01)$, Figure 10 shows that without the constraint, the rectangles overlap meaning a collision occurs. With the constraint the rectangles do not overlap and the collision is avoided. However the car does not converge to a point that is within the parking spot. It shows a little offset in y-axis.

Assigment 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.

Solution

Figure 10 shows that there is an offset in y and the car is not parked fully horizontal ($\psi \neq 0$). Increasing the weight of y in Q ($Q = \text{diag}(1, \mathbf{15}, 0.1, 0.01)$) makes the controller more aggressive, reducing the offset by increasing the velocity and angles in the trajectory as shown in Figure 11.

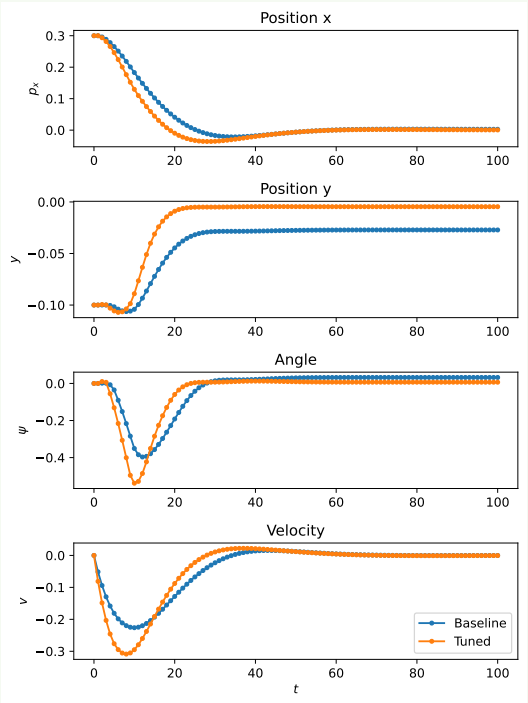

Figure 11: Improvement with $Q = \text{diag}(1, \mathbf{15}, 0.1, 0.01)$. (Baseline with original weights)

However the parking angle was found to be a little off still. Further improvement was found by realizing that the offset and angle had to be zero only at the end of the trajectory. Therefore this time, Q was kept the original and Q_f was changed to $Q_f = \text{diag}(5, \mathbf{50}, \mathbf{50}, 5)Q$. This creates a cost function that penalizes last states much more aggressively thus forcing the controller to make a trajectory that ends closer to the origin, thus removing the offsets. Figure 12 shows that this controller is not as aggressive during the maneuver and still stops at the center. This controller's performance was deemed acceptable and iterations were stopped. Figure 13 shows the trajectory of the controller with obstacle showing successful avoidance.

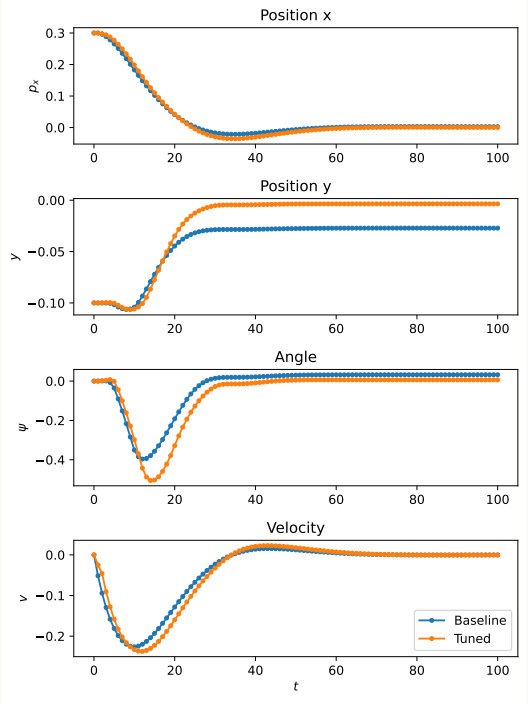

Figure 12: Improvement with $Q_f = \text{diag}(5, \mathbf{50}, \mathbf{50}, 5)Q$. (Baseline with original weights)

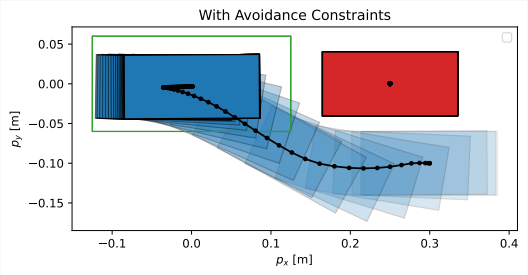

Figure 13: Trajectory of improved controller

Assigment 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

```
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:

```
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.

Solution

The solver time was logged using a global variable and `perf_counter`. Figure 14 shows the time per iteration in blue. The controller violates the real-time constraint (time per iteration $< T_s = 0.08$s) before 40 iterations. To fix this, the horizon was reduced from $N = 30$ to $N = 25$ without sacrificing performance (Figure 15). The updated controller now satisfies the real-time constraint.

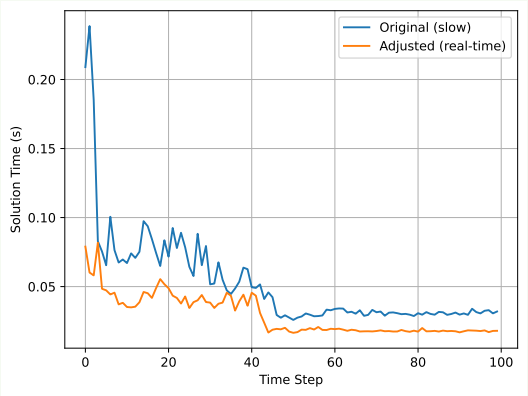

Figure 14: Time taken per iteration

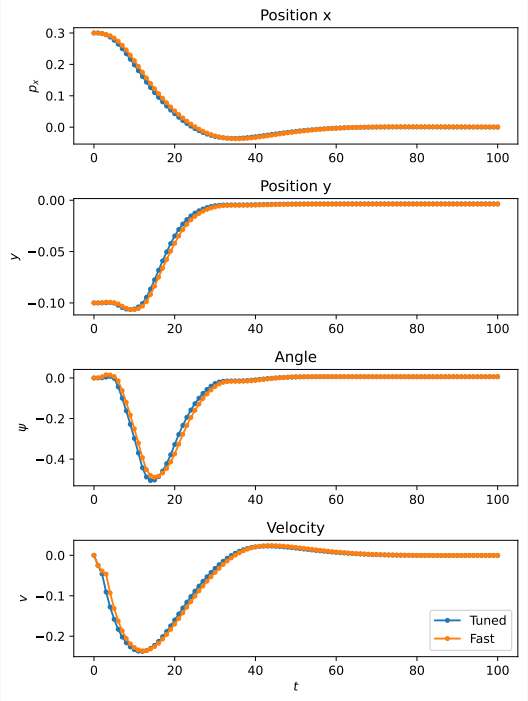

Figure 15: Comparison of $N = 30$ (Tuned) and $N = 25$ (Fast)

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

Solution

1. Using the EKF class developed in the exercise session and the option to `use_prior` in the provided solver code, the filtering update was implemented.
2. To use P^-_{T-N} and $\hat{x}_{T-N}$, N previous samples of $\hat{x}$ and P^- (a priori) are needed to be stored in a running window fashion. EKF updates are computed recursively as new data comes in and a buffer provides a delay of N time steps to get P^-_{T-N}.
3. For both $N = 10$ and $N = 25$, filtering reduces the variance in the state estimates as can be seen in Figure 16, 17. As can be seen, shorter horizon without filtering produces estimates with very large variance. It is known that for a large enough N, the estimation error can go to zero without any prior filtering under few assumptions. This can be seen that $N = 25$ produces estimates with much lower variance and bias than $N = 10$ without filtering. Adding filtering adds information about the samples outside the horizon window thus improving the estimates.
4. Figure 18 shows the effect of clipping on the estimates. Since clipping in this case improves the EKF estimates, it also improves the MHE estimates. However this is a heuristic and only applicable to this problem.

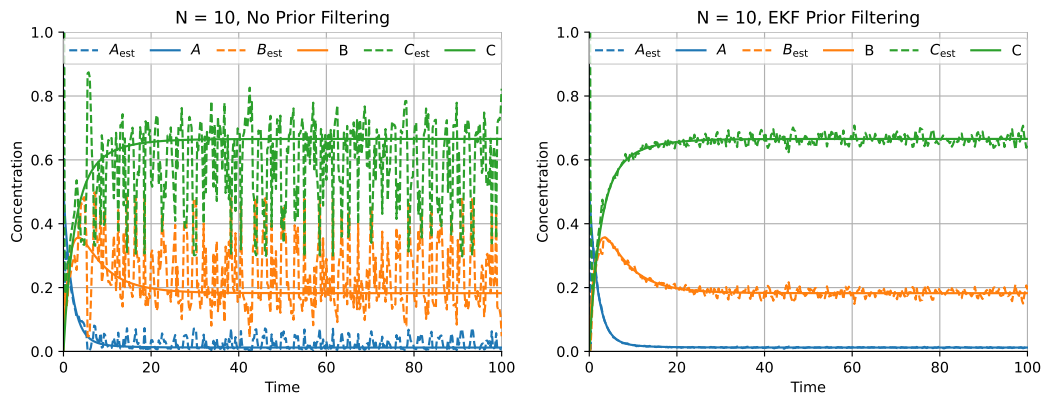

Figure 16: Effect of filtering on state estimates (no clipping in EKF)

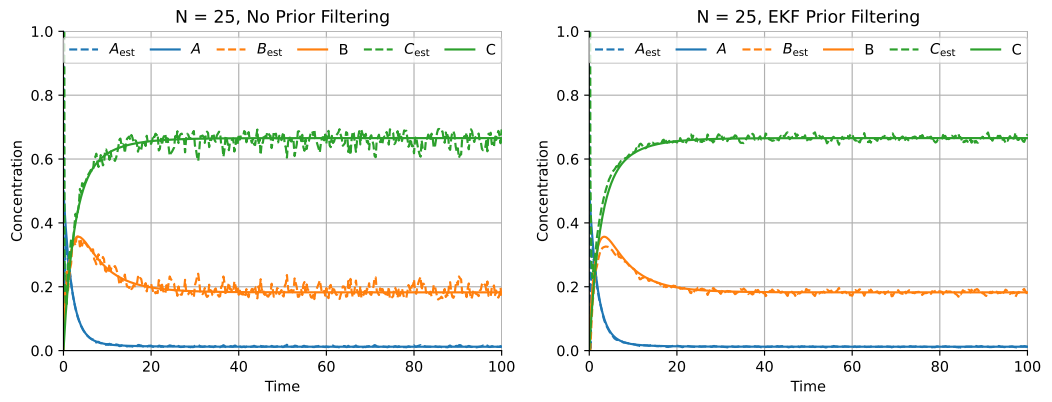

Figure 17: Effect of filtering on state estimates (no clipping in EKF)

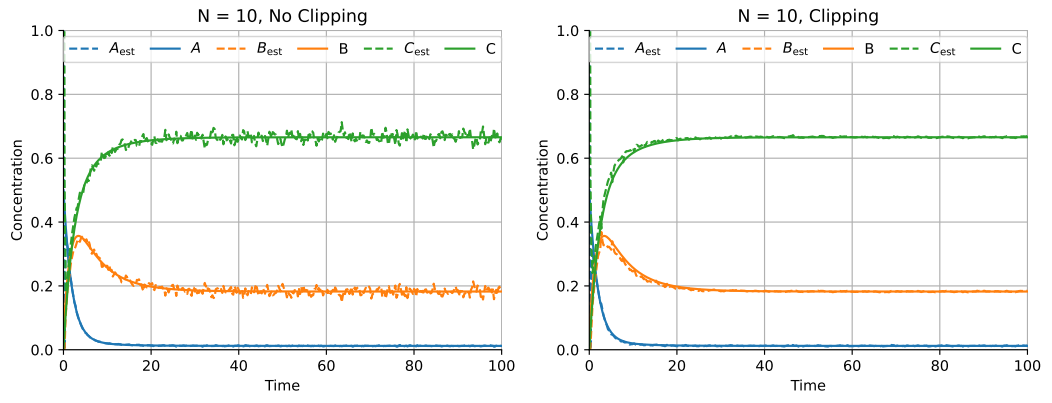

Figure 18: Effect of clipping on state estimates (with filtering)

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 = 1.25e-13
it.    1 | JN = 1.51e+04 | ||h||2 = 1.41e-13
```

Solution

The method converges after first iteration. For linear case, the OCP is an equality constrained quadratic program (QP). Newton-Lagrange works by approximating the OCP as a QP at each step. Since the true problem is a QP itself, this approximation is exact resulting in the true solution after one iteration.

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`.

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?

Solution

Using an initial guess of all zeros except x_0, the solver converges after three iterations. The final cost is $J_N = 226$ and constraint violation is $\|h\|_2 = 1.57 \times 10^{-9}$. The control sequence u_k and the corresponding state sequence x_k are shown in Figure 20. As can be seen, the found control sequence is stabilizing as it brings the state back to 0. Therefore the solution is plausible.

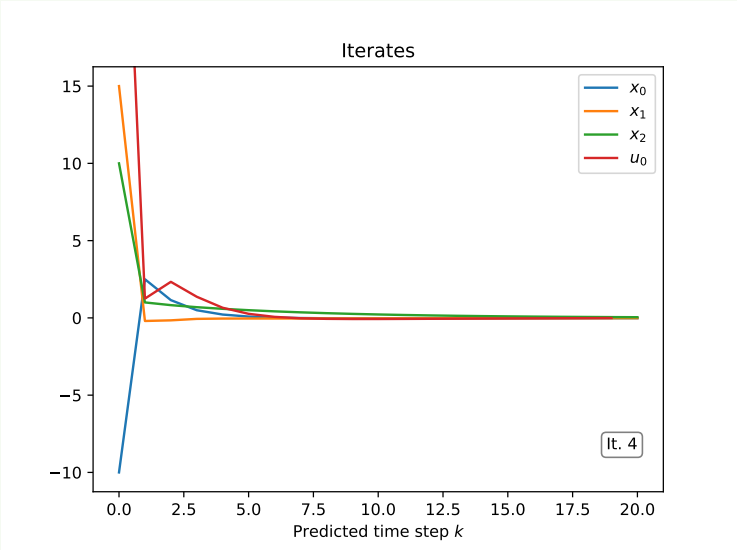

Figure 19: u_k and x_k solution for an all zero guess

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?

Solution

Using initial state x_0 and $u_k = 0$ the system was simulated and the resulting state sequence x_k was used as an initial guess. This resulted in the solver diverging. Figure 20 shows that the state trajectory is very different from the previous exercise with states taking very large values. This can cause the solver to diverge because Newton-Lagrange is a local method which works by minimizing a quadratic approximation around the current point. When started far from the minimum, it may diverge, which is what is happening here. Line search is needed to damp the update to guarantee global convergence.

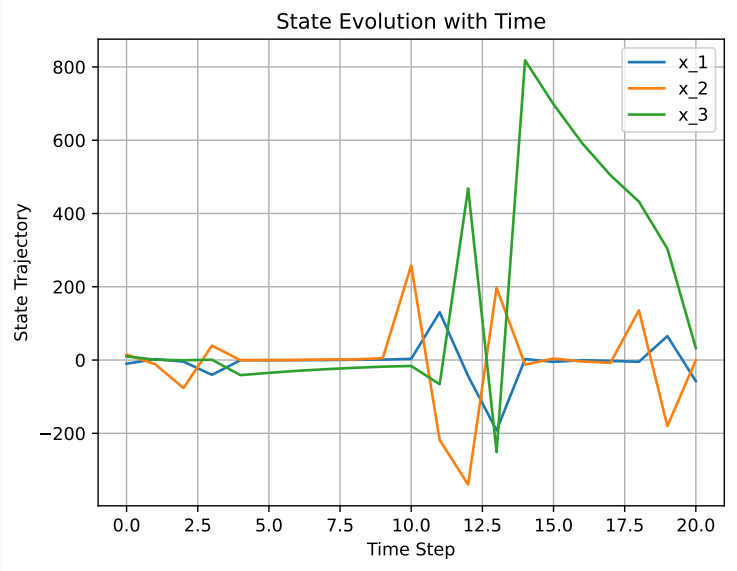

Figure 20: State trajectory for x_0

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?

Solution

With Armijo line search and dividing α by 0.8 for every Armijo condition failure made the algorithm converge after 25 iterations. Penalty c_v is chosen as $10\max(p^{v+1})$. The final cost was $J_N = 226$ and constraint violations were 1.01×10^{-10}. The violations are small and the cost is the same as Assignment 6.2. Figure 21 shows that the found solution exactly matches the previously found solution in Assignment 6.2. The method converges now thanks to the fact that line search can make Newton Lagrange from a locally converging to a globally converging algorithm.

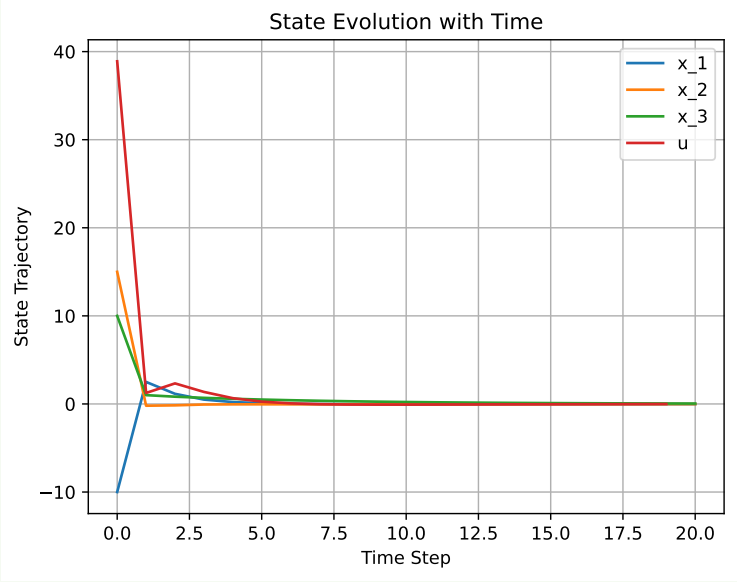

Figure 21: Solution with Armijo line search

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.

Solution

The solver does not converge and runs out of iterations. Figure 22 shows the solution. Upon checking the eigenvalues of $\begin{bmatrix} Q_k & S_k^\top \\ S_k & R_k \end{bmatrix}$ for $k = 0, \cdots, N-1$ and Q_N it was found that after first iteration the matrices started having negative eigenvalues which meant the QP is non-convex. Without convexity Newton step is not guaranteed to be a direction of descent thus causing the failure of convergence.

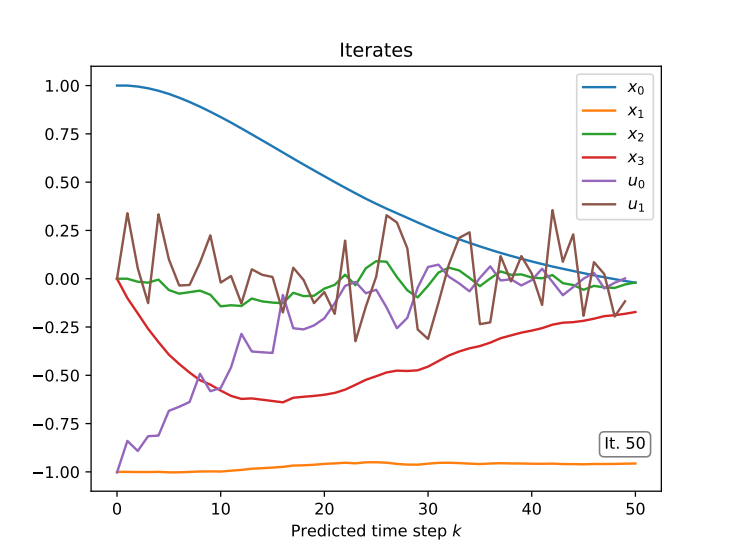

Figure 22: Solution of the parking problem

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`!

Solution

With regularization the problem converges to a solution in 65 iterations. The cost is $J_N = 83.3$ and the constraint violations are 2.16×10^{-7}. This shows that the solution is valid. Figure 23, 24 show the solutions on a plot. This time the solution is much smoother than before and the state converges to 0 further validating the solution.

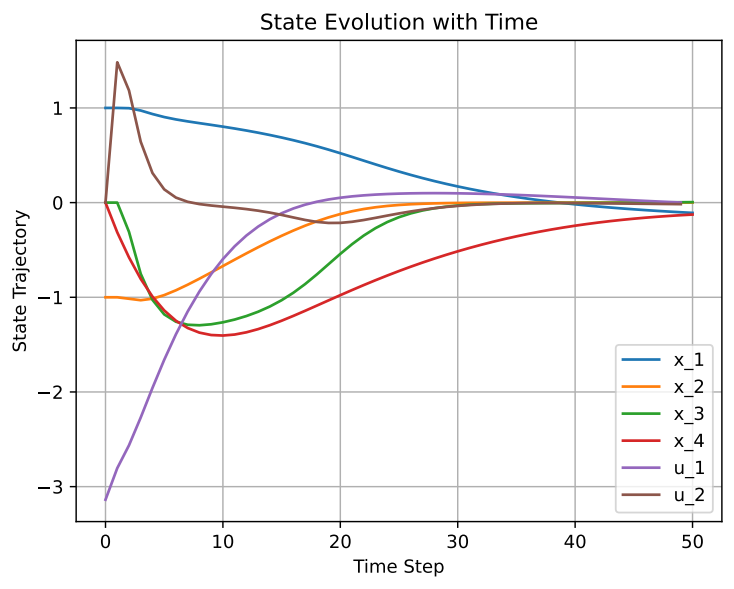

Figure 23: Solution of the parking problem with regularization

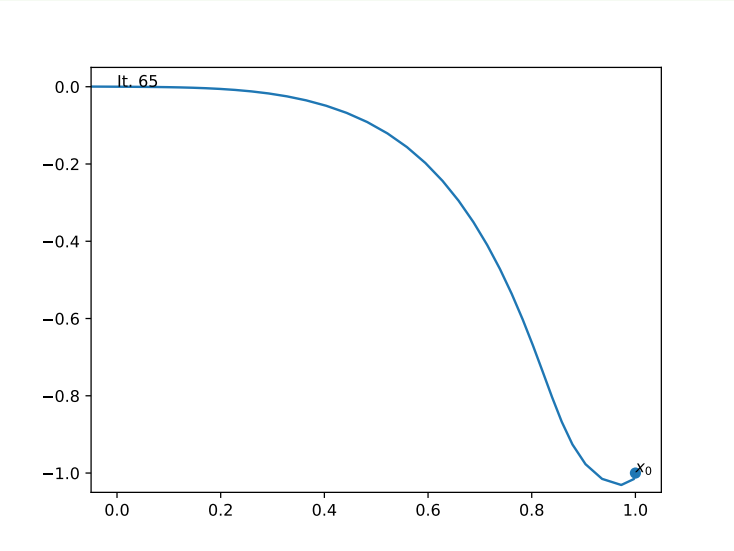

Figure 24: (x, y) positions of the parking problem solution