H0E76A: Model Predictive Control

— Homework assignments —

Andrea Alboni

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`.

Done.

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.

Zio pera, grafico sembra giusto. uno converge dal basso perchè considera solo il costo per fare gli N (horizon) steps dell'mpc. non considera tutto ciò che avverrà successivamente all'horizon. l'altro converge dall'alto perchè considera il costo di tutti gli steps, infatti per horizon < 4 sono instabili -> costo infinito. linea costante perchè costo infinito è quello.

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.

TO FIX. CHECK CODE TOO. ha detto che grafico sembra ok, ha consigliato di non discretizzare il confine (cosa che già non faccio, quindi indagare), gli sembra strano che alcuni punti sotto il nostro constraint risultino comunque rossi. per il tempo di esecuzione ha consigliato di ridurre la griglia di discretizzazione (from 40x40 to 30x30).

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.

pagine appunti 315-320.

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 $u_t = Kx_t$.

Assignment 3.3

Formulate a constraint on P and K that guarantees that

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

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.

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)

Assignment 3.6

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

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

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

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.)

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.

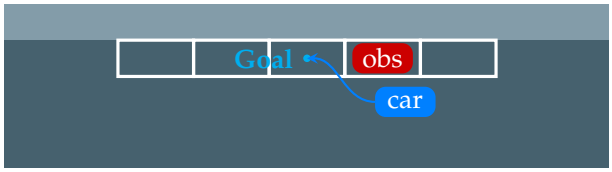

Figure 1: Autonomous parking task with obstacle

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. 2. 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.

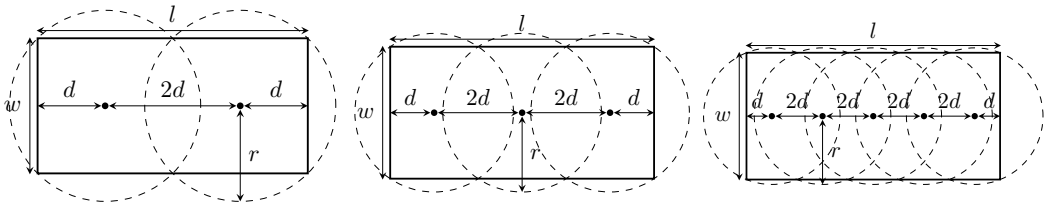

Figure 2: Vehicle covering with 2, 3 and 5 circles respectively.

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.

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.

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.

Assignment 4.4

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

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

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`=$(0.25, 0.12)$

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.

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

```
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.

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.

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?

```
<Paste the output of the program here.>
```

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?

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?

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?

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.

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`!