Model Predictive Control - EL2700

Linear Model Predictive Control

Assignment 4

Division of Decision and Control Systems
School of Electrical Engineering and Computer Science
Kungliga Tekniska Högskolan

Gregorio Marchesini, Samuel Erickson Andersson, and Mikael Johansson

Figure 1: Gilles Villeneuve in 1978, driving with team Ferrari.

Introduction

In previous assignments, your team has tested your capabilities as a driver. You have implemented some "old school" controllers, using Linear Quadratic Regulator and the Pole Placement, but it is now time to show that you are not just any other pilot. In this task, you will finally implement your first linear MPC controller within the same racetrack we used in the previous assignment. Similar to the previous assignment, our continuous time model of the vehicle linearized around the optimal raceline is given by

$$\begin{aligned} \dot{x}(s) &= \frac{1}{\bar{v}} A^c x(s) + \frac{1}{\bar{v}} B^c u(s) + \frac{1}{\bar{v}} B_w^c \kappa(s) \\ y(s) &= Cx(s) = \left[e_d(s), e_\psi(s)\right]^T \end{aligned} \tag{1}$$

where

$$\begin{bmatrix} \dot{e}_d(t) \\ \dot{e}_\psi(t) \\ \dot{v}_y(t) \\ \dot{r}(t) \\ \dot{\delta}(t) \end{bmatrix} = \begin{bmatrix} 0 & \bar{v} & -1 & -L_p & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & \frac{c_1}{m\bar{v}} & (\frac{c_2}{m\bar{v}^2} - 1) & \frac{C_f}{m} \\ 0 & 0 & \frac{c_2}{I_z \bar{v}} & \frac{c_3}{\bar{v} I_z} & \frac{C_f l_f}{I_z} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} e_d(t) \\ e_\psi(t) \\ v_y(t) \\ r(t) \\ \delta(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \bar{v} \\ 0 \\ 0 \\ 0 \end{bmatrix} \kappa(t) + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{I_z} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u_\delta(t) \\ u_r(t) \end{bmatrix}. \tag{2}$$

We also discretize such model in order to obtain the discrete-time model

$$x_{k+1} = Ax_k + Bu_k + B_w \kappa_k \tag{3}$$

where κ_k is the curvature of the raceline we are following. We encourage you to go to the previous assignment to refresh your understanding of the model (2). For system (3), we also already derived a feedforward input/state trajectory u^{ff} and x^{ff} that track the reference trajectory with minimum error, and that is dynamically feasible, in the sense that

$$x_{k+1}^{\text{ff}} = Ax_k^{\text{ff}} + Bu_k^{\text{ff}} + B_w \kappa_k. \tag{4}$$

If we define the state and input errors

$$e_k^x = x_k - x_k^{\text{ff}},\ e_k^u = u_k - u_k^{\text{ff}}$$

We want to design an MPC controller that stabilizes the state error $e_k^x = x_k - x_k^{\text{ff}}$, whose dynamics are given by

$$e_{k+1}^x = Ae_{k+1}^x + Be_k^u. \tag{5}$$

The MPC controller you will define takes the following form

$$\text{minimize} \sum_{k=0}^{N-1} (e_k^x)^T Q e_k^x + (e_k^u)^T R e_k^u + (e_N^x)^T Q_T e_N^x, \tag{6a}$$

$$\begin{aligned}
e_k^x &= x_k - x_k^{\text{ff}} && \forall k = 0, \dots N && \text{(6b)}\\
e_k^u &= u_k - u_k^{\text{ff}} && \forall k = 0, \dots N-1 && \text{(6c)}\\
e_{k+1}^x &= A e_k^x + B e_k^u && \forall k = 0, \dots N-1 && \text{(6d)}\\
u_{lb} &\leq u_k \leq u_{ub}, && \forall k = 0, \dots N-1 && \text{(6e)}\\
x_{lb} &\leq x_k \leq x_{ub}, && \forall k = 0, \dots N && \text{(6f)}\\
x_0 &= \hat{x}_0 && && \text{(6g)}
\end{aligned}$$

where u_{lb}, u_{ub} represent vectors of input lower/upper bounds, and x_{lb}, x_{ub} represent vectors of state lower/upper bounds.

Design Task

In the assignment `.zip` file of the assignment, you will find the following Python files

1. **`linear_car_model.py`**: Contains a class that stores the parameters of your vehicle. This is the same as the previous assignment. You will **not** need to touch this file for this assignment.
2. **`road.py`**: This file contains an instance of the racetrack object that we use to store information about the racetrack. You will **not** need to touch this file for this assignment.
3. **`sim.py`**: This file contains the routines you will be using to simulate the performance of your MPC controller in tracking the optimal raceline. You will **not** need to touch this file for this assignment.
4. **`mpc_controller.py`**: In here, you will find the classes `MPC` and `MPCdual`, in which you will implement a standard MPC controller and an MPC controller with dual mode horizon. You will mainly work with these scripts.
5. **`task3.py`**: This is the main file that you will run to visualize the results of your simulations.
6. **`u_ff.npy`**, **`x_ff.npy`**, **`raceline.npy`**: These are pre-stored matrices that contain the feedforward action for your system and the receline.

Your task will be to answer the following questions by appropriately **implementing** the code, where required, and by **motivating** your design choices. It is suggested that you use plots from your code to answer the questions!

Q1: In the file `mpc_controller.py` we are going to implement the class `MPC`. Namely, you will implement the method `setup_mpc_problem` which defines (7) as an optimisation problem in `cvxpy` which will be saved in the variable `self.problem`. To define the problem, you will use the following recipe, which is valid for any general MPC controller (the skeleton of the code will accompany you in these steps).

1. Define the **variables** of you problem as
   (a) $x \in \mathbb{R}^{n\times(N)}$ (state variables),
   (b) $u \in \mathbb{R}^{m\times(N-1)}$ (input variables),
2. Define the **parameters** of your problem as
   (a) $x_0 \in \mathbb{R}^n$ (initial state variable)
   (b) $x^{\text{ff}} \in \mathbb{R}^{n\times N}$ (state feedforward reference)
   (c) $u^{\text{ff}} \in \mathbb{R}^{m\times(N-1)}$ (input feedforward reference)

3. Based on this information,**define the state and input error** $e^x = x - x^{\text{ff}} \in \mathbb{R}^{n\times N}$ and $e^u = u - u^{\text{ff}} \in \mathbb{R}^{m\times(N-1)}$.

4. Compute the **cost** of your MPC controller. Recall that you can access the cost matrices using the syntax `self.Q`, `self.R` and `self.Qt`.

5. Define the **dynamic constraint** of your MPC controller based on the errors e^x and e^u as per (6d). Remember to check the column/row consistency of the terms you use.

6. Define the **state/input constraints** for your system. In your case, you can access the lower/upper bound vectors `self.ubu`, `self.ulb`, `self.ubx`, and `self.lbx`. You will define the constraint element by element as, for example,

$$\begin{aligned} x_k[i] \leq x_{\text{ub}}[i] \text{ and } x_k[i] \geq x_{\text{lb}}[i] \qquad \forall i = 0, \dots n-1. \\ u_k[i] \leq u_{\text{ub}}[i] \text{ and } u_k[i] \geq u_{\text{lb}}[i] \qquad \forall i = 0, \dots m-1. \end{aligned}$$

   As you will see in the code, we will avoid including constraints that are not bounding (i.e., when $x_{\text{ub}}[i] = +\infty$ and $x_{\text{lb}}[i] = -\infty$) since not all the solvers accept these unbounded constraints (e.g. Mosek).

7. Define the **initial state constraint** $x_0 = \hat{x}_0$ where $\hat{x}_0$ is the initial state of your system, which we will update at every iteration in which we apply our MPC controller.

With these steps, you have now completed your first MPC controller! You will see in the future that most MPC controllers follow very similar steps to the ones you have, with different variations on the constraints/cost depending on the application. Now pass to Q2 to finish your controller.

Q2: You will now fill the final bit of your controller. Namely, you will implement the method `solve_mpc` inside the class `MPC`, where the input to the method is

1. $\hat{x}_0 \in \mathbb{R}^n$ (initial state of your system)
2. $x^{\text{ff}} \in \mathbb{R}^{n\times N}$ (state feedforward reference over the horizon)
3. $u^{\text{ff}} \in \mathbb{R}^{n\times(N-1)}$ (input feedforward reference over the horizon)

and the output will be your input to the system given by $u_0^\star$ (the optimal first input found by your solver). Recall that after calling the solver, you can access the optimal input array as the array `self.u.value`, from which you can select the first element.

Q3: We are now done with the class `MPC`. You can now go to the file `task4.py` and look into the section marked as Q3. You will create your MPC controller with the syntax

```
mpc_controller = MPC(model      = system,
                     N          = 20,
                     Q          = Q,
                     R          = R,
                     Qt         = Qt,
                     warm_start = True,
                     solver     = 'CLARABEL')
```

For this question, a pair of matrices Q and R will be given to you, and **you will not add** any input or state constraints to the MPC controller. Moreover, the terminal cost matrix Q_T will be set to be equal to the *cost-to-go* matrix P of a standard LQR controller such that P solves the Riccati equation

$$P = Q + A^\top PA - A^\top PB(R + B^\top PB)^{-1}B^\top PA,$$

and we thus set $Q_T = P$ (this is done for you in the code already).

At this point, run your MPC controller and comment on the difference, at each step, between the LQR optimal input and the MPC optimal input. Are they the same? why? Does your controller render your closed-loop system asymptotically stable? Does the log-scaled plot of the MPC cost agree with your intuition? (Hint: if you coded the MPC correctly, then the difference between the LQR controller and the MPC controller input should be zero up to numerical precision. But then you should tell us why this is the case)

Q4: Select a pair of matrices Q, R that gives you a response that you deem appropriate. You are encouraged to reuse the same settings you might have selected from the previous assignment,

where you used the Bryson rule to tune your controller. Briefly explain according to which criterion you made your choice.

Q5: It is now the time to add some constraints. Namely you can add constraints on your MPC controller using the syntax

```
mpc_controller.add_upper_bound_on_state(state_index: int, upper_bound: float)

mpc_controller.add_lower_bound_on_state(state_index: int, lower_bound: float)

mpc_controller.add_upper_bound_on_input(input_index: int, upper_bound: float)

mpc_controller.add_lower_bound_on_input( input_index: int, lower_bound: float)
```

With these available methods, set the constraints

$$|\delta| \leq 35 \cdot \frac{\pi}{180},\ |v_y| \leq 3.1,\ |u_\delta| \leq 50,\ |u_r| \leq 60$$

Q6: As you have learned in class, the MPC controller you have designed **after** introducing the constraints is not necessarily stabilizing. This is why **terminal sets** offer a valuable approach to enforce the stability of your controller even after adding the given state/input constraints. Unfortunately, when tracking a time-varying trajectory, the computation of terminal sets is often not practical. But luckily, it is still possible to stabilize our controller by providing a sufficiently high terminal cost Q_T. The most widely used approach to do this is to set $Q_T = \lambda P$ with $\lambda \geq 1$. Computing the exact value of $\bar{\lambda}$ that renders your controller stabilizing is difficult except for some specific cases, but you can often obtain it by trial and error. Test the performance of your controller when you sets

$$\lambda_1 = 2 \quad \lambda_2 = 10 \quad \lambda_3 = 50 \quad \lambda_3 = 100.$$

Explain in your own words how the controller is affected by the choice of λ. After testing the provided values, feel free to choose a value that seems to stabilize your system and provide a good response.

Q7: Now that we have explored some theoretical aspects of our controller, we want to go to real implementation issues. Namely, to run on real hardware, your controller needs to be fast. Hence, as an engineer, you will often have to play with the **solver** that you apply to solve (7), the **number of horizon steps** N, and the **warm starting** of the solver (see Computer Exercise 2).

1. Simulate your MPC controller with $Q_T = P$ and try the different values

   $$N_1 = 20 \quad N_2 = 40 \quad N_3 = 80.$$

   Comment on the difference in performance for different values of N based on the computational time and the LQR VS MPC control difference. Do you think that in general bigger N is better for stability? Considering your controller should be run at 10 HZ, what is the maximum N you can use to stay within this limit?

2. Now try to change your solver. Namely, try to solve your problem using "MOSEK" and "CLARABEL". Which one is better in terms of computational time?

3. Repeat the experiment setting the `warm_start` first to True and then to False. What do you notice?

Q8 (Bonus): A second method you have seen in class to provide some stability guarantees for your MPC controller is by defining a *dual mode horizon*. With this horizon, your MPC formulation will look something like

$$\text{minimize} \sum_{k=0}^{N-1} (e_k^x)^T Q e_k^x + (e_k^u)^T R e_k^u + (e_N^x)^T Q_T e_N^x \tag{7a}$$

$$e_k^x = x_k - x_k^{\text{ff}} \qquad \forall k = 0, \dots N \tag{7b}$$
$$e_k^u = u_k - u_k^{\text{ff}} \qquad \forall k = 0, \dots N \tag{7c}$$
$$\hat{e}_k^x = \hat{x}_k - x_k^{\text{ff}} \qquad \forall k = N, \dots N + 2 + N_d \tag{7d}$$
$$e_{k+1}^x = Ae_k^x + Be_k^u \qquad \forall k = 0, \dots N \tag{7e}$$
$$\hat{e}_{k+1}^x = (A - BL)\hat{e}_k^x \qquad \forall k = N, \dots N + N_d - 1 \tag{7f}$$
$$u_{lb} \leq u_k \leq u_{ub}, \qquad \forall k = 0, \dots N \tag{7g}$$
$$x_{lb} \leq x_k \leq x_{ub}, \qquad \forall k = 0, \dots N \tag{7h}$$
$$x_{lb} \leq \hat{x}_k \leq x_{ub}, \qquad \forall k = N, \dots N + N_d \tag{7i}$$
$$\hat{x}_N = x_N \tag{7j}$$
$$x_0 = \hat{x}_0 \tag{7k}$$

where $\hat{x}_k$ are the so called *dual state variables*. Try to implement this controller by *copy-paste* your code for the standard MPC controller and then change the method `setup_mpc_problem` to account for the new dual variables. Critically, you will have to consider creating the additional dual variable and implement the dual constraints (7f) and (7i). Try to set $Q_T = P$ and increase the dual mode horizon N_d until your MPC controller is stable. How does this compare with the standard MPC controller with you have implemented in Q6 for the value of λ you have selected there?

Submission Your submission should be a .ZIP file, named with your Group number (e.g., Group1.zip), containing:

1. a PDF document (can be a scanned document, generated from Word, Latex or others) containing the answers and motivation to the questions on to the assignment. Make sure to name your document Assignment4_STUDENT1_STUDENT2.pdf;
2. the *.py code files with a working task4.py that shows your solution on the plot output.

Good Luck!