1 Problem formulation

The `MPCProblem` class permits to solve Model Predictive Control (MPC) problems. The class considers the following state space representation of the system

$$\boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k \tag{1}$$
$$\boldsymbol{y}_k = C_1\boldsymbol{x}_k \tag{2}$$

where $\boldsymbol{x}_k \in \mathbb{R}^n$, $\boldsymbol{y}_k \in \mathbb{R}^q$, and $\boldsymbol{u}_k \in \mathbb{R}^p$ represent the state of the system, the output, and the control at time step k, respectively. $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times p}$, and $C_1 \in \mathbb{R}^{q\times n}$ are the state, the control, and the output matrices. The matrices given to the class are the discrete time matrices, so if you have the continuous time state space representation of your system, you should first discretize it for a particular sampling time.

In general, the `MPCProblem` class solves the following quadratic optimization problem over the time horizon T:

$$\min_s s^\intercal Q s \tag{3}$$

with

$$s = [\boldsymbol{e}, \boldsymbol{u}, \dot{\boldsymbol{u}}, \epsilon_y, \epsilon_y, \epsilon_{\dot{u}}, \epsilon_T] \tag{4}$$

subject to

$$\boldsymbol{u}_0 = \boldsymbol{u}^* \tag{5}$$
$$\boldsymbol{e}_k = C_1\boldsymbol{x}_k - \boldsymbol{r}_k, \quad k = 0, \dots, T \tag{6}$$
$$-\epsilon_T \leq \boldsymbol{e}_T \leq \epsilon_T \tag{7}$$
$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \dot{\boldsymbol{u}}_k \cdot t_s, \quad k = 0, \dots, T-1 \tag{8}$$
$$C_2\boldsymbol{x}_{k+1} \leq \boldsymbol{y}_{\max} + \epsilon_y, \quad k = 0, \dots, T-1 \tag{9}$$
$$C_2\boldsymbol{x}_{k+1} \geq \boldsymbol{y}_{\min} - \epsilon_y, \quad k = 0, \dots, T-1 \tag{10}$$
$$\boldsymbol{u}_{k+1} \leq \boldsymbol{u}_{\max} + \epsilon_u, \quad k = 0, \dots, T-1 \tag{11}$$
$$\boldsymbol{u}_{k+1} \geq \boldsymbol{u}_{\min} - \epsilon_u, \quad k = 0, \dots, T-1 \tag{12}$$
$$\boldsymbol{u}_{k+1} - \boldsymbol{u}_k \leq \dot{\boldsymbol{u}}_{\max} \cdot t_s + \epsilon_{\dot{u}}, \quad k = 0, \dots, T-1 \tag{13}$$
$$\boldsymbol{u}_{k+1} - \boldsymbol{u}_k \geq \dot{\boldsymbol{u}}_{\min} \cdot t_s - \epsilon_{\dot{u}}, \quad k = 0, \dots, T-1 \tag{14}$$
$$\epsilon_y \geq 0 \tag{15}$$
$$\epsilon_u \geq 0 \tag{16}$$
$$\epsilon_{\dot{u}} \geq 0 \tag{17}$$
$$\epsilon_T \geq 0 \tag{18}$$
$$\tag{19}$$

where

$$\boldsymbol{x}_k = A^k\boldsymbol{x}_0 + \sum_{i=0}^{k-1} A^{k-i-1}B\boldsymbol{u}_i \tag{20}$$

In Eq. (3), s is the minimization variable and Q a diagonal matrix, which can be modified by the user to weight the minimization terms. The Q matrix must be positive definite, so all entries on the diagonal must be strictly positive. This is required because of the resolution method used by the solver, which requires problems to be strictly convex. This kind of problem is simpler to solve, so the solution can be computed very efficiently. In turn, we cannot set diagonal entries to 0, which requires the problem to be expressed in a non-intuitive way. For example, in standard MPC problems $\boldsymbol{x}_k$ is part of the state so, for example, the error constraint is formulated as

$$\boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k \tag{21}$$
$$\boldsymbol{e}_k = C_1\boldsymbol{x}_k - \boldsymbol{r}_k, \tag{22}$$

which is much easier to read than the combination of Eqs. (6) and (20). Clearly, $\boldsymbol{x}_k$ should not be part of the minimization: you want to minimize the error, not the state. With a solver accepting semi-definite programs, this is not a problem, as we can simply set the diagonal entries of the Q matrix regarding the $\boldsymbol{x}_k$ terms to 0. With the QuadProg++[1] solver used in here this is not possible, and this is the reason why the problem is formulated in a more "complex" way.

The s variable is composed by a set of subvariables:

[1] `https://github.com/liuq/QuadProgpp`

- $\boldsymbol{e}$, output error: this is the set of output errors for each time step k in the considered time horizon. $\boldsymbol{e}_k \in \mathbb{R}^q$ is the difference between the output of the system ($C_1\boldsymbol{x}_k$) and the target reference value ($\boldsymbol{r}_k$) (constraint in Eq. (6)) for $k = 0, \ldots, T$.
- $\boldsymbol{u}$, control: this is the set of control actions for each time step k in the considered time horizon, with $\boldsymbol{u}_k \in \mathbb{R}^p$ for $k = 0, \ldots, T-1$. The first value ($\boldsymbol{u}_0$) is fixed and it is a parameter of the problem (Eq. (5)).
- $\dot{\boldsymbol{u}}$, control derivative: this is the set of control derivatives, i.e., $\dot{\boldsymbol{u}}_k \in \mathbb{R}^q$ is the difference between $\boldsymbol{u}_{k+1}$ and $\boldsymbol{u}_k$ (constraint in Eq. (8)) for $k = 0, \ldots, T-1$. This term is optional can can be disabled when instantiating the problem, disregarding the control derivative when minimizing.
- ϵ_y: slack variable for the violation of output constraints. This is optional and can be disabled when instantiating the problem. This removes the slack variable from the optimization problem and allows no constraint violation.
- ϵ_u: slack variable for the violation of control constraints. This is optional and can be disabled when instantiating the problem. This removes the slack variable from the optimization problem and allows no constraint violation.
- $\epsilon_{\dot{u}}$: slack variable for the violation of control derivative constraints. This is optional and can be disabled when instantiating the problem. This removes the slack variable from the optimization problem and allows no constraint violation.
- ϵ_T: slack variable for the terminal constraint. This is optional and can be disabled when instantiating the problem. This removes the slack variable from the optimization problem and transforms the terminal constraint into $\boldsymbol{e}_T = 0$.

With respect to the constraints, the interpretation is the following:

- Eq. (7): this is called the terminal constraint and indicates that, at the end of the time horizon, we want our output error to be zero, or very close to it. Notice that this might make the problem unfeasible, e.g., when setting a very small time horizon. For this reason, the constraint can be disabled. Alternatively to disabling the constraint, it is possible to enable a slack variable for it, so that the constraint can be violated but at a certain cost.
- Eq. (8): this constraint simply defines that two successive control actions differ by the control derivative. When the minimization on the control derivative is disabled, this constraint is not used.
- Eqs. (9) and (10): constraints on output limiting the maximum and the minimum output value. Notice that we use a different output matrix than the one of the error constraint (Eq. (6)), which we call C_2. The reason is simple: imagine that our state is composed by the acceleration and the speed of a vehicle and that we want to reach a target speed while limiting the acceleration. If we use the same output matrix this is not possible (if not with some ugly hack), but by using two different output matrices we increase the flexibility while lowering the computational effort. In the document and in the code, $C_2 \in \mathbb{R}^{q_2 \times n}$. These constraints can be violated if the slack variable for the output is enabled. If the slack variable on output is not enabled, then the ϵ_y variable is not considered.
- Eqs. (11) and (12): constraints on control limiting the maximum and the minimum control value. These constraints can be violated if the slack variable for the control is enabled. If the slack variable on control is not enabled, then the ϵ_u variable is not considered.
- Eqs. (13) and (14): constraints on control derivative limiting the maximum and the minimum control derivative value. The limit is multiplied by the sampling time t_s, so changing the sampling time doesn't require the user to change the $\dot{\boldsymbol{u}}_{\max}$ and the $\dot{\boldsymbol{u}}_{\min}$ values. These constraints can be violated if the slack variable for the control derivative is enabled. If the slack variable on control derivative is not enabled, then the ϵ_u variable is not considered.
- Eqs. (15) to (18): these constraints simply force the slack variables to be positive.

Finally, Eq. (20) defines $\boldsymbol{x}_k$ in terms of the variables of the problem. $\boldsymbol{x}_0$ and $\boldsymbol{u}_0$ are the initial state and control action, respectively, while $\boldsymbol{u}_k$ for $k = 1, \ldots, k-1$ are the control actions computed by the algorithm.

2 Sample problem

The library comes with a sample program (`test-mpc.cc`) that solves an MPC problem where the state is described by the following system of differential equations:

$$\dot{\boldsymbol{x}} = \begin{cases} \dot{a} = -\frac{1}{\tau}a + \frac{1}{\tau}u \\ \dot{v} = a \end{cases} \tag{23}$$

The system state represents the acceleration and the speed of a vehicle, where the acceleration is subject to an actuation lag modeled as a first order lag with a time constant τ. By writing the system in the standard state-space representation we obtain

$$\dot{\boldsymbol{x}} = G\boldsymbol{x} + H\boldsymbol{u} \tag{24}$$

$$\boldsymbol{y} = C_1\boldsymbol{x} \tag{25}$$

where

$$G = \begin{bmatrix} -\frac{1}{\tau} & 0 \\ 1 & 0 \end{bmatrix}, \quad H = \begin{bmatrix} \frac{1}{\tau} \\ 0 \end{bmatrix}, \quad C_1 = \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{26}$$

We discretize the continuous time representation by transforming H and G into A and B, respectively, so that the state-space representation now becomes

$$\boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k \tag{27}$$

$$\boldsymbol{y}_k = C_1\boldsymbol{x}_k \tag{28}$$

where

$$A = e^{G\cdot t_s}, \quad B = \int_0^{t_s} e^{G\lambda} d\lambda H \tag{29}$$

obtaining

$$A = \begin{bmatrix} e^{-\frac{t_s}{\tau}} & 0 \\ \tau\left(1 - e^{-\frac{t_s}{\tau}}\right) & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 - e^{-\frac{t_s}{\tau}} \\ t_s + \tau\left(e^{-\frac{t_s}{\tau}} - 1\right) \end{bmatrix} \tag{30}$$

A, B, and C are thus the matrices we pass to the `MPCProblem` class given a particular choice of the time constant τ and the sampling time t_s.

In the provided example, we set $t_s = 0.1\,\mathrm{s}$ and $\tau = 0.5\,\mathrm{s}$, obtaining

$$A = \begin{bmatrix} 0.8187 & 0 \\ 0.09063 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0.1812 \\ 0.009365 \end{bmatrix} \tag{31}$$

The initial state $\boldsymbol{x}_0$ and the initial control $\boldsymbol{u}_0$ are set to $[0, 0]^\intercal$ and $[0]$, respectively, the reference $\boldsymbol{r}_k$ to $[1]$, the time horizon T to 60 steps (6 s), the terminal constraint is enabled, and there is no bound for output, control, and control derivative variables. Finally all the weights in the Q matrix are set to 1.

In the root folder you find a bash script (`test-mpc.sh`) that runs the test application under different output parameters. If the scripts finds an `R` installation, it will also plot the results to some PDF files.

The first example (Fig. 1) shows the results for the default parameters, i.e., the ones previously described. The graph shows the different quantities considered in the optimization problem, i.e., the actual speed and the target speed, the acceleration, the control input, and the control derivative. As expected, the solution brings the system to the target speed minimizing also the effort on control and control derivative.

In the second example (Fig. 2) we lower the weights for the control and the control derivative terms in the minimization, setting them to 0.01 instead of 1. Control and control derivative are now much larger, as we told our solver that we do not care too much about minimizing them. This results in a faster settling time.

In the third example (Fig. 3) we use the parameters of the second but we add upper and lower bounds on control actions, i.e., $1\,\mathrm{m/s^2}$ and $-1\,\mathrm{m/s^2}$, respectively. The result is very similar to the second, with the only difference that the control action is "truncated" at $1\,\mathrm{m/s^2}$ as per constraint. This causes a slightly larger settling time with respect to the second example.

In the fourth example (Fig. 4), w.r.t. the third, we add a constraint on the control derivative as well, i.e., $-0.5\,\mathrm{m/s^3} \leq \dot{\boldsymbol{u}}_k \leq 0.5\,\mathrm{m/s^3}$. The result is pretty evident, both in terms of control derivative bounds and in terms of control input. The bound on control derivative causes a "linear" increase and decrease of the control action. The additional bound causes the settling time to increase.

The fifth example (Fig. 5) is the same as the second (i.e., we lower the weights for control and control derivative) but, in addition, we add a bound to the maximum acceleration ($0.6\,\mathrm{m/s^2}$). The plot shows that the system reaches the target speed, but the acceleration never exceeds the bound set by the constraint.

The final example (Fig. 6) enables the slack variable for the output constraint set in the fifth example, setting the weight penalization for the slack variable in the Q matrix to 10. In this case the solver is allowed to violate the constraint on acceleration, as it can bee seen in the plot. The constraint bound is highlighted by the gray dashed line, and the solver exceeds this limit reaching roughly $0.75\,\mathrm{m/s^2}$.

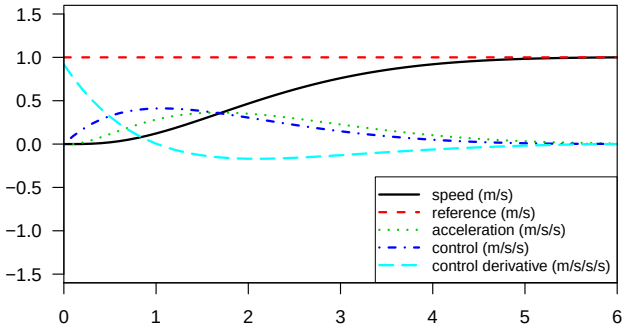

Figure 1: Results for default parameters.

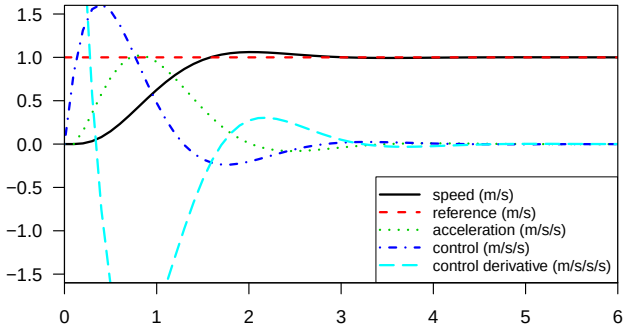

Figure 2: Results for lower weights on control and control derivative.

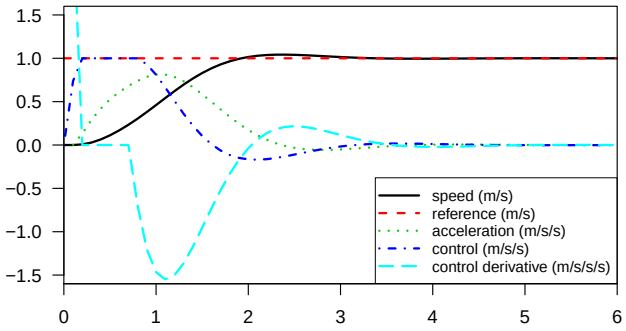

Figure 3: Results for lower weights on control and control derivative, plus bounded control action.

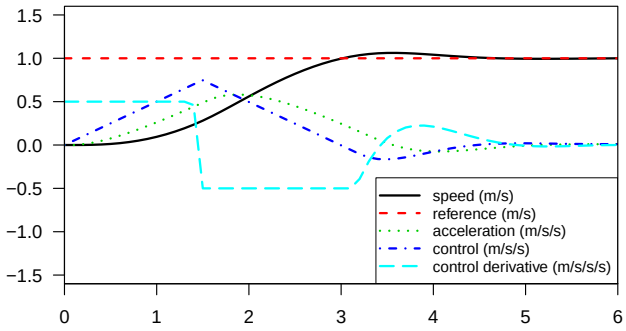

Figure 4: Results for lower weights on control and control derivative, plus bounded control and control derivative action.

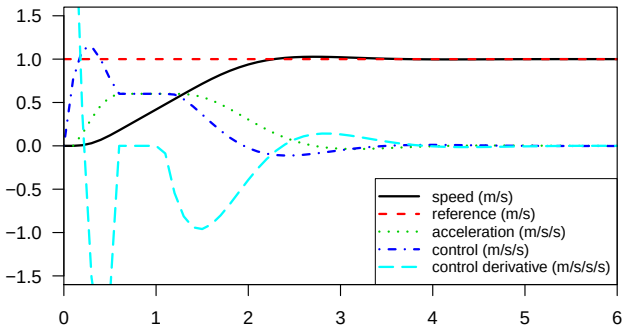

Figure 5: Results for lower weights on control and control derivative, plus bound on acceleration.

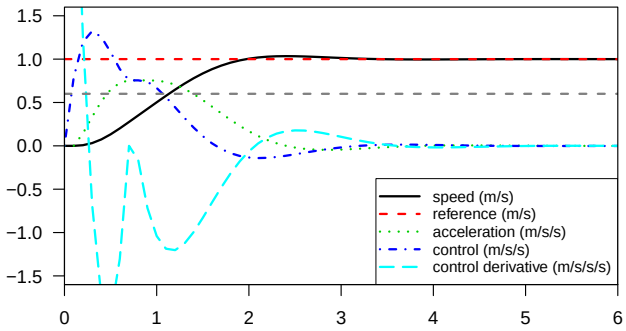

Figure 6: Results for lower weights on control and control derivative, plus bound on acceleration and slack variable for output bounds enabled.