CHEM-E7225 Advanced Process Control D — Winter 2026

Lecture 07 – Model Predictive Control w/ Constraints

Otacilio "Minho" Neto (otacilio.bneto@pm.me)

In this lecture, we consider the problem of designing model-predictive controllers when the system to be controlled is subject to *operational constraints*. We introduce the classes of input-constrained and state-constrained problems.

1 Linear-quadratic optimal control problems w/ constraints

The *constrained linear-quadratic regulator* problem is the constrained QP

$$\underset{x[\cdot],\ u[\cdot]}{\text{minimize}} \quad \frac{1}{2}\sum_{k=0}^{N-1}\begin{bmatrix}x[k]\\u[k]\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}Q & S\\S^{\mathsf{T}} & R\end{bmatrix}\begin{bmatrix}x[k]\\u[k]\end{bmatrix}+\frac{1}{2}x[N]^{\mathsf{T}}Q_f x[N] \tag{1a}$$

$$\text{subject to} \quad x[k+1]=A_{\Delta t}x[k]+B_{\Delta t}u[k], \quad k=0,\dots,N-1, \tag{1b}$$

$$H_x x[k]+H_u u[k]\preceq h, \quad k=0,\dots,N-1, \tag{1c}$$

$$x[0]=x_0. \tag{1d}$$

The LQ Problem (1) is now defined by the *tuning matrices* (Q,R,S,Q_f), *constraint matrices/vector* (H_x,H_u,h), and the *system matrices* $(A_{\Delta t},B_{\Delta t})$. In this lecture, for ease of notation, we use $u[n]=K(x[n])$ to refer to the receding-horizon policy based on this problem. While the unconstrained LQ Problem has a closed-form solution (via dynamic programming), its constrained version can only be solved numerically (e.g., via Newton-type algorithms). Moreover, Lagrangian duality tells us that the optimal value achieved by a constrained problem is always smaller or equal than that of the unconstrained case. In conclusion, *inequality constraints add a special challenge to the design of optimal controllers*. However, they cannot be ignored when the real system is subjected to those constraints, and it's our job to know how to encode different operational requirements in the form of linear inequality constraints.

when the constraints can be expressed by two independent inequalities,

$$H_x x[k]+H_u u[k]\preceq h \quad\Longleftrightarrow\quad H_x x[k]\preceq h_x, \quad H_u u[k]\preceq h_u, \tag{2}$$

they are said to be *separable*. This is quite common in practice, so it is natural to study input-constrained and state-constrained problems separately; as we'll see shortly, each have their own "particularities".

In setpoint tracking problems, the constraints must be shifted *by the references:* $H_x(x^\delta[n]+x_{ref})\preceq h_x$ *and* $H_u(u^\delta[n]+u_{ref})\preceq h_u$.

1.1 Input-constrained problems

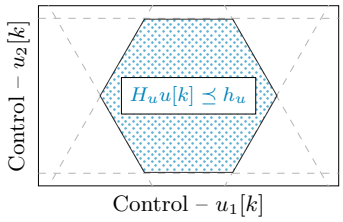

Figure 1. Polyhedral constraints.

The general constraint $H_u u[k]\preceq h_u$ describe a *polyhedral constraint set*

$$\mathcal{U}=\{u\in\mathbb{R}^{N_u}\ \mid\ H_{ui}u-h_{ui}\leq 0,\quad i=1,\dots,N_h\}, \tag{3}$$

with (H_{ui},h_{ui}) being the ith row of the matrix $H_u\in\mathbb{R}^{N_h\times N_u}$ and vector $h_u\in\mathbb{R}^{N_h}$, respectively. Thus, at each kth timestep, the controls $u[k]$ must lie in the polyhedron $\mathcal{U}$, which consists of an intersection of a finite set of *halfspaces* (Figure 1). The dimension $N_h>0$ defines the number of inequality constraints per timestep, meaning that an input-constrained LQ Problem has (at least) a total of NN_h inequality constraints.

The polyhedral constraint includes many familiar special cases. A very common type of polyhedral constraints are *bound constraints*, that is, *lower and upper bounds* that the controls must satisfy at each timestep. They are expressed as

$$u_{\min}\preceq u[k]\preceq u_{\max} \tag{4}$$

given the vectors of lower $(u_{\min}\in\mathbb{R}^{N_u})$ and upper $(u_{\max}\in\mathbb{R}^{N_u})$ bounds. The set $\mathcal{U}=\{u\in\mathbb{R}^{N_u}\mid u_{\min,i}\leq u_i\leq u_{\max,i},\ i=1,\dots,N_u\}$ describes a N_u-dimensional box. For this reason, they are sometimes referred to as *box constraints* (Figure 2).

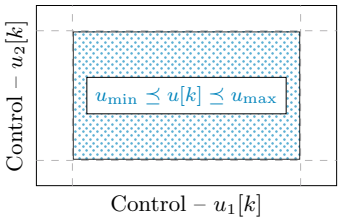

Figure 2. Box constraints.

Bound constraints are equivalent to a single polyhedral constraint, since

$$\begin{matrix} u[k] \preceq u_{\max} \\ u[k] \succeq u_{\min} \end{matrix} \quad \iff \quad \begin{bmatrix} I \\ -I \end{bmatrix} u[k] \preceq \begin{bmatrix} u_{\max} \\ u_{\min} \end{bmatrix}.$$

In practice, these constraints arise from the physical limits of the actuation equipment. However, they can also include *desirable* requirements, such as enforcing limits for which the system's response is predictable or well-understood. Due to being ubiquitous, there are special methods for dealing with bound constraints, often allowing the LQ Problem (1) to be solved faster than if a general polyhedral constraint is considered.

A class of input constraints which are common in (bio)chemical process are those representing *the allocation of a finite resource*. If the control input $u(t) = (u_1(t), \dots, u_{N_u}(t))$ represents the distribution of some finite resource (with limit $s_{\max} > 0$) into N_u *slots*, the LQ Problem must include the constraints

$$u[k] \succeq 0, \quad u_1[k] + u_2[k] + \cdots + u_{N_u}[k] \leq s_{\max} \quad \iff \quad \begin{bmatrix} -I \\ \mathbf{1}^\mathsf{T} \end{bmatrix} u[k] \preceq \begin{bmatrix} 0 \\ s_{\max} \end{bmatrix},$$

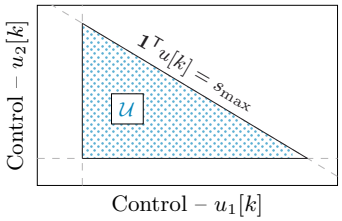

Figure 3. Simplex constraints.

where $\mathbf{1}$ is a N_u-dimensional vector of 1's. A similar constraint relates to controls representing *percentages* or *mixture coefficients* [1, 2]. For instance, let the feed to a reactor be a perfect mixture of N_u components, whose relative concentrations are $0 \leq u_i(t) \leq 1$ $(i = 1, \dots, N_u)$. The LQ Problem must then include

$$u[k] \succeq 0, \quad u_1[k] + u_2[k] + \cdots + u_{N_u}[k] = 1 \quad \iff \quad \begin{bmatrix} -I \\ \mathbf{1}^\mathsf{T} \\ -\mathbf{1}^\mathsf{T} \end{bmatrix} u[k] \preceq \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix},$$

where the equality constraint is represented in inequality form (using the fact that $\mathbf{1}^\mathsf{T} u[k] \leq 1 \iff 1 \leq \mathbf{1}^\mathsf{T} u[k] \leq 1$). Note that the constraint $u[k] \preceq \mathbf{1}$ is not needed, as it is implicitly satisfied by the other two. Geometrically, a perfect mixture represents a N_u-dimensional simplex (Figure 3). For this reason, these are sometimes referred to as *simplex constraints*. Finally, note that such constraints only make sense for $N_u > 2$. If $N_u = 2$, then it's much easier to simply define $u_2(t) = 1 - u_1(t)$ and rewrite the dynamics in terms of $0 \leq u_1(t) \leq 1$; the mixture constraint becomes implicit.

†*Street fighting* techniques

Previously, we learned that (convex) inequality-constrained problems can be solved via *interior-point methods* [3]. While these are numerically stable and easy to implement (thanks to modern software), it is still common to find in industry/academia some "*street-fighting*" tricks to try and get rid of the inequality constraints. The idea is to design an *unconstrained* MPC, but then modify the control policy somehow to ensure that the input constraints are not violated. We present two of such tricks below, but advise against using them in critical applications.

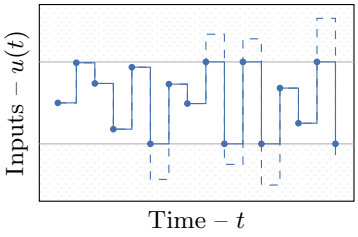

Figure 4. Input clipping.

A common, but very crude, method to deal with input constraints consists of enforcing bound constraints by *saturating the solution of the unconstrained problem* (Figure 4). For each ith control input $(i = 1, \dots, N_u)$, this takes the form

$$u_i[n] = \begin{cases} u_{\max,i} & \text{if } K_i(x[n]) \geq u_{\max,i}, \\ K_i(x[n]) & \text{if } u_{\min,i} < K_i(x[n]) < u_{\max,i}, \\ u_{\min,i} & \text{if } K_i(x[n]) \leq u_{\min,i}, \end{cases} \tag{5}$$

where $K_i(x[n])$ is the ith component of the L-MPC policy without input constraints. This is known as *clipping the control inputs*, and formally represented as $u[n] = \texttt{clip}(u_{\min}, K(x[n]), u_{\max})$. The most common clipping is used to enforce positivity constraints $(u[n] \succeq 0)$, written compactly as $u[n] = \max(0,\ K(x[n]))$.

Since this leads to a (nonlinear) closed-form control policy, the clipping-based L-MPC policy is perhaps the fastest receding-horizon control policy that still enforces input constraints. However, it is clear that this approach leads to suboptimal solutions. Modern solvers are efficient and they guarantee optimality, thus clipping is really not a good practice in L-MPC design; unless for open-loop stable systems that *need* to be operated at megahertz frequencies.

Another interesting *street-fighting* technique consists of directly incorporating the input constraints into the cost

function using a *fixed barrier function*. The common case consists of solving the approximated LQ Problem

$$\underset{x[\cdot],\ u[\cdot]}{\text{minimize}} \quad \frac{1}{2}\left(\sum_{k=0}^{N-1}\begin{bmatrix}x[k]\\u[k]\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}Q & S\\ S^{\mathsf{T}} & R\end{bmatrix}\begin{bmatrix}x[k]\\u[k]\end{bmatrix} - \frac{1}{\tau}\sum_{i=1}^{N_h}\log(h_{ui} - H_{ui}u[k])\right) + \frac{1}{2}x[N]^{\mathsf{T}}Q_f x[N] \tag{6a}$$

$$\text{subject to} \quad x[k{+}1] = A_{\Delta t}x[k] + B_{\Delta t}u[k], \quad k = 0,\ldots,N{-}1, \tag{6b}$$

$$x[0] = x_0, \tag{6c}$$

for a fixed value of $\tau > 0$. Despite the intimidating expression, the optimization problem above is unconstrained and can be solved directly using Newton-type algorithms, that is, without the need for interior-point methods. Thus, we can define a suboptimal L-MPC policy $u[n] = K_\tau(x[n])$ based on solving the above problem instead of the original, constrained, one. Due to the log-barrier function, this policy always satisfies the input constraints.

Using Lagrangian duality, we can show that the control policy $K_\tau(\cdot)$ is (NN_h/τ)-suboptimal with respect to the $K(\cdot)$. We can get an arbitrarily good controller by choosing a large $\tau > 0$, while still having the computational benefits of solving an unconstrained problem. Unfortunately, the nonlinearity incurred by the barrier function becomes very expressive for large values of τ, leading to poor convergence for the Newton-type algorithms. The barrier parameter τ thus works as a tuning parameter that controls the trade-off between optimality and performance. Interestingly, there is practical evidence that computing $K_\tau(\cdot)$ is (sometimes) orders-of-magnitude faster than computing $K(\cdot)$, and that this speed-up allows even such suboptimal policies to outperform optimal policies, as they enable very-large control frequencies [4]. Consequently, such *suboptimal MPC policies* $K_\tau(\cdot)$ are actually very popular in practice.

1.2 State-constrained problems

In general, the state constraints $H_x x[k] \preceq h_x$ describe a *polyhedral constraint set*

$$\mathcal{X} = \{x \in \mathbb{R}^{N_x} \ \mid \ H_{xi}u - h_{xi} \leq 0, \quad i = 1,\ldots,N_h\}, \tag{7}$$

with (H_{xi}, h_{xi}) being the ith row of the matrix $H_x \in \mathbb{R}^{N_{hx}\times N_x}$ and vector $h_x \in \mathbb{R}^{N_{hx}}$, respectively. At each kth timestep, the state $x[k]$ must lie in the polyhedron $\mathcal{X}$, which consists of an intersection of a finite set of *halfspaces*. Again, the dimension $N_{hx} > 0$ defines the number of inequality constraints per timestep, meaning that an state-constrained LQ Problem has (at least) a total of $(N{+}1)N_{hx}$ inequality constraints (in addition to possible input-constraints).

The previous discussion about input constraints should, in principle, be applicable to state constraints. However, there is a practical issue with state constraints. During operation, there could be constraint violations caused by inaccurate predictions and/or external disturbances *"kicking"* the state to the unfeasible region (Figure 5). This is specially the case when the constraints are not describing physical limits (as these would be satisfied by the dynamics), but rather *desirable specifications* (that is, conditions that we want to enforce for economical/environmental reasons).

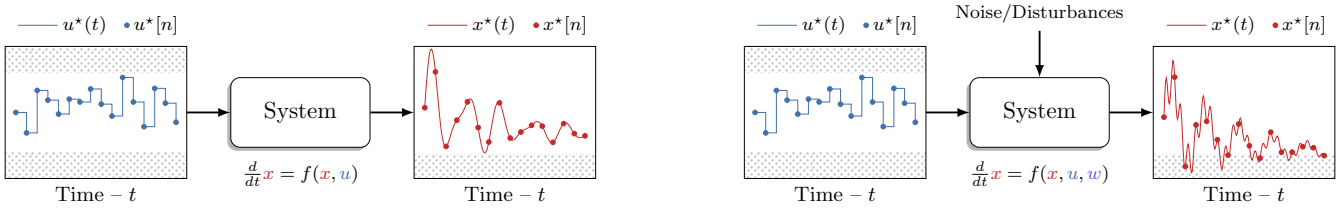

Figure 5. Based on the predictive model, the nominal state response strictly satisfies the state constraints (left). In reality, due to external disturbances, the true state response deviates from the predicted response and can violate the constraints (right).

Violating the state constraints is critical not only for economical reasons: if the LQ Problem becomes unfeasible, then the MPC cannot solve it and thus crashes. The task of ensuring that the MPC remains feasible under closed-loop operation is known as the *recursive feasibility* problem. In this course, we deal with this problem using an approach common in real-world practice: *softening the state constraints*. The main idea is to modify the LQ Problem (1) such that the MPC can itself verify whether the state constraints are achievable or not, and then relax them in some satisfactory manner, if necessary. The focus is on a *softnening* approach based on *slack variables*.

> *The discipline of* robust model predictive control *[5] concerns the design of controllers that ensure recursive feasibility even in the presence of disturbances and model uncertainty.*

Soft constraints using slack variables

The state constraints can be relaxed as

$$H_x x[k] \preceq h_x, \quad (\forall k) \quad \xrightarrow[\varepsilon[k] \succeq 0]{\text{Relaxation w/}} \quad H_x x[k] \preceq h_x + \varepsilon[k], \quad (\forall k). \tag{8}$$

The components of $\varepsilon = (\varepsilon[0], \ldots, \varepsilon[N-1])$ are user-defined parameters known as *slack variables*. When $\varepsilon = 0$, these constraints are equivalent, that is, they must be enforced–they are *hard state constraints*. When $\varepsilon \succ 0$, then some constraint violation is permitted–they become *soft state constraints*. We would like the slack variables to be zero when the state constraints are achievable, and small as possible otherwise. The task now is to redesign the LQ Problem (1) such that the our controller becomes responsible for *determining* the best value of these slack variables in real-time.

The natural procedure would be to include $\varepsilon = (\varepsilon[0], \ldots, \varepsilon[N-1])$ into the LQ Problem (1) as a new decision vector, then augment the cost and inequality functions appropriately. For bound constraints, the problem can be written as

$$\underset{x[\cdot],\ u[\cdot],\ \varepsilon[\cdot]}{\text{minimize}} \quad \frac{1}{2}\sum_{k=0}^{N-1}\left(\begin{bmatrix} x[k] \\ u[k] \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} Q & S \\ S^{\mathsf{T}} & R \end{bmatrix} \begin{bmatrix} x[k] \\ u[k] \end{bmatrix} + \varepsilon[k]^{\mathsf{T}} R_\varepsilon \varepsilon[k]\right) + \frac{1}{2} x[N]^{\mathsf{T}} Q_f x[N] \tag{9a}$$

$$\text{subject to} \quad x[k+1] = A_{\Delta t} x[k] + B_{\Delta t} u[k], \quad x[0] = x_0, \quad k = 0, \ldots, N-1, \tag{9b}$$

$$-\varepsilon[k] + x_{\min} \preceq x[k] \preceq x_{\max} + \varepsilon[k], \quad k = 0, \ldots, N-1, \tag{9c}$$

$$u_{\min} \preceq u[k] \preceq u_{\max}, \quad k = 0, \ldots, N-1, \tag{9d}$$

$$0 \preceq \varepsilon[k]. \quad k = 0, \ldots, N-1, \tag{9e}$$

Let's detail what happened: (i) The stagewise cost $L(\cdot)$ was augmented with the term $\varepsilon[k]^{\mathsf{T}} R_\varepsilon \varepsilon[k]$, given the weighting matrix R_ε which is set to be diagonal with very large coefficients (e.g., $R_\varepsilon = 10^6 I$); (ii) For each kth state constraint, the lower bound is relaxed by $-\varepsilon[k]$ and the upper bound is relaxed by $\varepsilon[k]$; (iii) The input bound constraints remain unchanged (they don't need to be relaxed, as their feasibility is not affected by disturbances); (iv) The slack variables ε are restricted to be nonnegative, as per their definition. Under this formulation, the state constraints can be relaxed as needed to ensure feasibility. Due to the penalty term $\varepsilon[k]^{\mathsf{T}} R_\varepsilon \varepsilon[k]$ with very-large R_ε, the MPC will determine $\varepsilon[k] = 0$ $(\forall k)$ whenever possible–the constraints remain as desired. When a constraint violation is inevitable, the MPC does not crash but instead determine (as small as possible) slacks $\varepsilon[k] \succeq 0$ $(\forall k)$ to relax the state bounds. Note that we can use the same slack variable $\varepsilon[k]$ for both the lower and upper bounds, as they cannot be both active anyways.

The slack variable is very popular in practice because, for many relevant problems, it works very well. Moreover, due to the large penalties on the slack variables, this soft-constrained MPC permits constraint violations but tries to correct them immediately. Of course, this approach does not come without disadvantages. Firstly, we cannot used it when state constraints cannot be violated at any circumstance; for instance, when they can cause catastrophic consequences, like a reactor blowing up. In such cases, we would need more advanced robust MPC designs. Secondly, the slack variables add complexity to the LQ Problem (1), affecting the controller's performance. Specifically, the above method introduces at least $N_h N_x$ decision variables and inequality constraints (or $N_h N_x/2$, in the case of bound constraints), and augmenting the stagewise cost $L(\cdot)$ with very-large penalties can lead to ill-conditioned problems prone to numerical issues.

The LQ Problem (9) can be rewritten in the original form by augmenting the controls as $\widetilde{u}[k] = \texttt{col}(u[k], \varepsilon[k])$, then changing the cost and inequality matrices accordingly (you should not even be surprised at this point). However, in most software, the formulation presented above is easier to implement and solve.

References

[1] D. E. Seborg et al. *Process dynamics and control*. John Wiley & Sons, 2016.

[2] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl. *Model Predictive Control: Theory, Computation and Design*. 2nd ed. Nob Hill Publ., LLC., 2020. ISBN: 978-0-9759377-5-4.

[3] S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004. ISBN: 978-0-52-183378-3.

[4] Y. Wang and S. Boyd. "Fast model predictive control using online optimization". In: *IEEE Transactions on control systems technology* 18.2 (2009), pp. 267–278.

[5] F. Borrelli, A. Bemporad, and M. Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017. ISBN: 978-1-13-906175-9.