Modern Control Paradigms:

Lecture 5: Model Predictive Control

MPC is a practical optimal control approach which can deal with multivariable systems with process constraints.

At each time instant, MPC uses a **current** measurement of the system output, and a **model** of the system, to **compute** and implement a new control input, which:

- minimizes some **cost function**, while
- guaranteeing that **constraints** are satisfied

LQR via Least Squares, Unconstrained MPC

To dig more in to MPC let us recall first the derivation of Discrete time finite horizon LQR.

Consider a discrete-time dynamical system described by:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$$

with a cost defined as:

$$J_c(\mathbf{x}_0, \mathbf{u}) = \frac{1}{2}\Big[\sum_{k=0}^{N-1}\left(\mathbf{x}_k^T\mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^T\mathbf{R}\mathbf{u}_k\right) + \mathbf{x}_N^T\mathbf{P}\mathbf{x}_N\Big]$$

But for now, instead of using Bellman principle of optimality, let us write the solution of discrete time dynamics as follows:

$$\begin{aligned}\mathbf{x}_1 &= \mathbf{A}\mathbf{x}_0 + \mathbf{B}\mathbf{u}_0\\ \mathbf{x}_2 &= \mathbf{A}\mathbf{x}_1 + \mathbf{B}\mathbf{u}_1\\ &\cdots\\ \mathbf{x}_N &= \mathbf{A}\mathbf{x}_{N-1} + \mathbf{B}\mathbf{u}_{N-1}\end{aligned}$$

By recursive substitutions ($\mathbf{x}_1$ to $\mathbf{x}_2$ etc). We may arrive to the following:

$$\begin{aligned}\mathbf{x}_1 &= \mathbf{A}\mathbf{x}_0 + \mathbf{B}\mathbf{u}_0\\ \mathbf{x}_2 &= \mathbf{A}^2\mathbf{x}_0 + \mathbf{A}\mathbf{B}\mathbf{u}_0 + \mathbf{B}\mathbf{u}_1\\ &\cdots\\ \mathbf{x}_N &= \mathbf{A}^N\mathbf{x}_0 + \mathbf{A}^{N-1}\mathbf{B}\mathbf{u}_0 + \cdots + \mathbf{B}\mathbf{u}_{N-1}\end{aligned}$$

Collecting the terms gives:

$$\mathbf{X} = \mathcal{A}\mathbf{x}_0 + \mathcal{B}\mathbf{U}$$

with state and control sequence and matrices $\mathcal{A}, \mathcal{B}$ defined as:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_{N-1} \end{bmatrix} \quad \mathcal{A} = \begin{bmatrix} \mathbf{I} \\ \mathbf{A} \\ \vdots \\ \mathbf{A}^N \end{bmatrix} \quad \mathcal{B} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{B} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{AB} & \mathbf{B} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}^{N-1}\mathbf{B} & \mathbf{A}^{N-2}\mathbf{B} & \cdots & \mathbf{B} \end{bmatrix}$$

Now lets rewrite the cost function:

$$J_c = \Big[\sum_{k=0}^{N-1} \left(\mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \right) + \mathbf{x}_N^T \mathbf{P} \mathbf{x}_N \Big] = \mathbf{X}^T \mathcal{Q} \mathbf{X} + \frac{1}{2} \mathbf{U}^T \mathcal{R} \mathbf{U}$$

where:

$$\mathcal{Q} = \begin{bmatrix} \mathbf{I} \otimes \mathbf{Q} & \mathbf{0} \\ \mathbf{0} & \mathbf{P} \end{bmatrix} \quad \mathcal{R} = \mathbf{I}_N \otimes \mathbf{R}$$

and $\otimes$ being Kronecker product

Substitution of the dynamics yields $\mathbf{X} = \mathcal{A}\mathbf{x}_0 + \mathcal{B}\mathbf{U}$ yields the following cost:
ParseError: KaTeX parse error: Unexpected character: '\' at position 140: …\mathbf{x}^T_0 _

where: $\mathcal{H} = 2(\mathcal{R} + \mathcal{B}^T \mathcal{Q} \mathcal{B})$, $\mathcal{G} = 2\mathcal{B}^T \mathcal{Q} \mathcal{A}$ and $\mathcal{F} = \mathcal{A}^T \mathcal{Q} \mathcal{A}$

To find the optimal solution we differentiate the cost function above w.r.t $\mathbf{U}$ and set it to zero:

$$\nabla J_c = \mathcal{H}\mathbf{U} + \mathbf{G}x_0 = 0$$

Solving for $\mathbf{U}$ yields **optimal control sequence**:

$$\mathbf{U}^* = -\mathcal{H}^{-1}\mathbf{G}x_0$$

since the matrix $\mathcal{H}$ is p.d. by construction, it is always invertible.

Question: How to ensure that the solution above provide minimum?

For any $\mathbf{x}_0$, a solution $\mathbf{U}^*$ is the optimal sequence that: **exists**, **unique** and is a **global minimizer** of J_c

Note that optimal sequence is in linear function on initial state $\mathbf{x}_0$:

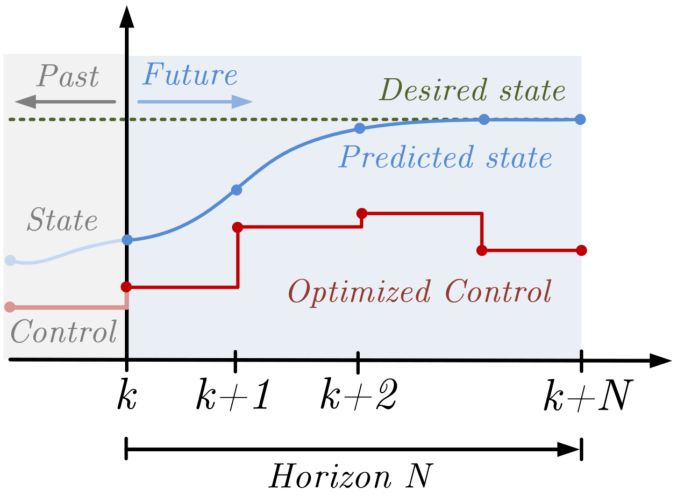
$$\mathbf{U}^* = -\mathcal{H}^{-1}\mathbf{G}\mathbf{x}_0 = \mathbf{K}_{MPC}\mathbf{x}_0$$

However in this sense we were able to calculate the optimal sequence on the finite horizon, in practice we need to apply control constantly. To do so usually the control input is implemented in a **receding-horizon** fashion.

We have feedback because a new control input is computed for each new measurement

Receding horizon control

The idea of receding horizon control is that at time $k+1$, **we solve a new problem** with the **new state** $x = \mathbf{x}_{k+1}$ but with the same prediction horizon length N.



The above cost function and optimal sequence apply to each time step k, then at each time instant k, a predictive controller uses

- Measurement/estimate of the current state $\mathbf{x}_0 = \mathbf{x}_k$ and
- Model of the system $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$ to predict future states
- Computes a **finite sequence of optimal control** inputs $\mathbf{U}^*$ that minimizes the cost J_c
- Implements only the **first input** in the optimal sequence: $\mathbf{u}_k = \mathbf{U}_0^*$

Then we shift to the next time instant $k+1$. When the optimal control sequence is implemented like this, it is called receding or moving horizon control (RHC).

It is worth noting that the resulting control is indeed feedback since it can be written as:

$$\mathbf{u}_k = \mathbf{U}_0^* = [\mathbf{I}, \mathbf{0}, \cdots, \mathbf{0}]\mathbf{U}^* = \mathbf{K}_{MPC}\mathbf{x}_k$$

This repeated solution of the finite horizon problem "approximates" the infinite horizon problem

Connection of RH MPC to LQR

As one can note the derivation of MPC is based on the very same cost and ideas of that we have in LQR, so there is shold be strong connection between them.

Indeed the solution to the infinite-horizon LQR problem is:

$$\mathbf{u}_k = \mathbf{K}_{LQR}\mathbf{x}_k$$

where

$$\mathbf{K} = (\mathbf{R} + \mathbf{B}^T\mathbf{P}\mathbf{B})^{-1}\mathbf{B}^T\mathbf{P}\mathbf{A}\mathbf{x}_k$$

and P is the solution to the Algebraic Riccati Equation (ARE)

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

And if the terminal weight P in the finite- horizon cost function J_c is the solution to above ARE, then:

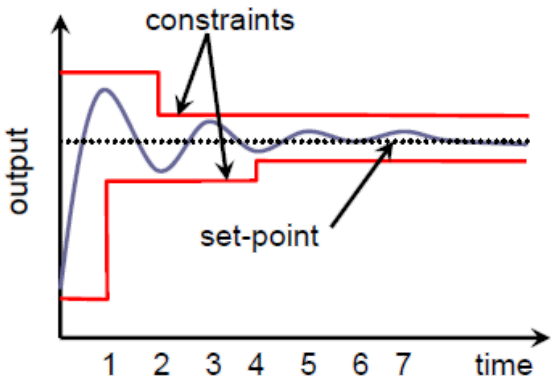
$$\mathbf{K}_{MPC} = \mathbf{K}_{LQR}$$

This is an important precursor in solving the problem of **infinite-horizon LQR with constraints**!

Incorporating Constraints

In practice, it is nearly always desirable to constrain system variables (e.g., system state/output) because of various reasons:

- **Physical limitations**, e.g. reservoir volume limits
- **Safety considerations**, e.g. critical temperatures
- **Performance specifications**, e.g. limit overshoot



The real power of MPC is in the ability to find the optimal sequence in the presence of constraints! Indeed let us rewrite the optimization problem above as:

$$\begin{aligned} \underset{\mathbf{U}_k}{\text{minimize}} \quad & \frac{1}{2}\mathbf{U}_k^T \mathcal{H} \mathbf{U}_k + \mathbf{x}_k^T \mathbf{G}^T \mathbf{U}_k + d \\ \text{subject to} \quad & \mathbf{M}\mathbf{U}_k \leq \mathbf{m} \\ & \mathbf{D}\mathbf{U}_k = \mathbf{c} \end{aligned}$$

If the cost is quadratic and constraints are linear in in $\mathbf{U}$, the problem above is written as quadratic program!

There are a lot of different solvers that can deal with QP problems. Among many others I suggest to use use osqp, ecos.

Stabilized Predictions MPC

Standard MPC makes open-loop predictions of the plant output, though the implementation of the control law in the receding horizon fashion provides some implicit feedback.

If the plant is unstable, problems may arise because of this open-loop nature of predictions:

- Numerical problems arising from large values of prediction matrices
- Amplification of model inaccuracies and disturbance effects

An effective way out is to ‘stabilize’ the predictions through a re-parametrization of the plant’s input signal.

Suppose that the default ‘do nothing’ policy is to apply the following linear state-feedback control.

$$\mathbf{u}_k = -\mathbf{K}\mathbf{x}_k$$

Here, we assume that the feedback gain $\mathbf{K}$ is stabilizing (i.e., the eigenvalues of $\mathbf{A} - \mathbf{B}\mathbf{K}$ lie inside the unit circle).

The default do-nothing predictions can be computed as follows:

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{A}_c \mathbf{x}_0 \\ \mathbf{x}_2 &= \mathbf{A}_c^2 \mathbf{x}_0 \\ &\cdots \\ \mathbf{x}_N &= \mathbf{A}_c^N \mathbf{x}_0 \end{aligned}$$

where $\mathbf{A}_c = \mathbf{A} - \mathbf{B}\mathbf{K}$

Now we can use optimization over a horizon N to modify these baseline 'do-nothing' predictions with the following objectives:

- Minimize the cost function
- Ensure that all constraints are satisfied

Let us consider the modified control input predictions in the form:

$$\mathbf{u}_k = -\mathbf{K}\mathbf{x}_k + \mathbf{c}_k$$

where $\mathbf{c}_k$ are chosen by the optimizer.

Using the same derivation that we have use for the standart MPC, we arrive to following prediction model:

With:

$$\mathcal{A} = \begin{bmatrix} \mathbf{I} \\ \mathbf{A}_c \\ \vdots \\ \mathbf{A}_c^N \end{bmatrix} \quad \mathcal{B} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{B} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{A}_c\mathbf{B} & \mathbf{B} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_c^{N-1}\mathbf{B} & \mathbf{A}_c^{N-2}\mathbf{B} & \cdots & \mathbf{B} \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} \mathbf{c}_0 \\ \mathbf{c}_2 \\ \vdots \\ \mathbf{c}_{N-1} \end{bmatrix}$$

So the regular MPC now can be applied to get the optimal control sequance $\mathbf{c}^*$, and controller is then become:

$$\mathbf{u}_k^* = -\mathbf{K}\mathbf{x}_k + \mathbf{c}_0^*$$

Note that instead of high powers of $\mathbf{A}$ in the standard MPC formulation, we have high powers of $\mathbf{A}_c = \mathbf{A} - \mathbf{B}\mathbf{K}$.
This is crucial since we choose $\mathbf{K}$ such that $\mathbf{A}_c$ is stable and hence $\mathbf{A}_c^k \to 0$ as $k \to \infty$.

It eliminates both numerical problems and amplification of errors.

In principle, we can use any stabilizing feedback, obtained, say, using pole-placement methods. However, in the case of predictive control with a quadratic cost function, it is natural to obtain $\mathbf{K}$ by solving the infinite horizon LQR with the same weight matrices as in the MPC problem.

Output Based Feedback

Another notable property of the MPC is that we may modify the cost function in order to perform the output stabilization/tracking with just a little change of the algorithm.
To do so let us introduce the output:

$$\mathbf{y}_k = \mathbf{C}_k \mathbf{x}_k$$

Then we may modify the cost to be:

$$J_c = \mathbf{Y}^T \mathcal{Q}_y \mathbf{Y} + \frac{1}{2}\mathbf{U}^T \mathcal{R} \mathbf{U}$$

which is identical to the standart LQR cost above if one choose $\mathbf{Q}_y = \mathbf{C}^T \mathbf{Q} \mathbf{C}$.

However, the control still will use the knowledge of the $\mathbf{x}_0$ to predict the states. An obvious way to deal with it is to introduce the state observer.

Simultaneous State/Control Optimization

Up to before we have analytically derived the solution of discrete dynamics as function of our parametrization. However a careful investigation in fact shows that the state predictions $\mathbf{x}_k$ are itself linear to the control $\mathbf{u}_k$. Thus we may incorporate them in to the QP constraints directly.

instead of solving descrite dynamics directly we can relly on power of optimization to solve following problem:

Recall that the finite horizon cost is given by:

$$J_c = \mathbf{X}^T \mathcal{Q} \mathbf{X} + \frac{1}{2}\mathbf{U}^T \mathcal{R} \mathbf{U}$$

Thus taking in to account linear dynamics one can write the MPC problem as following quadratic programm:

$$\begin{aligned}
\underset{\mathbf{U}_k, \mathbf{X}_k}{\text{minimize}} \quad & \mathbf{X}_k^T \mathcal{Q} \mathbf{X}_k + \frac{1}{2}\mathbf{U}_k^T \mathcal{R} \mathbf{U}_k \\
\text{subject to} \quad & \mathbf{X}_k = \mathbf{Z}\mathcal{A}\mathbf{X}_k + \mathbf{Z}\mathcal{B}\mathbf{U}_k \\
& \mathbf{M}_u \mathbf{U}_k \leq \mathbf{m}_u \\
& \mathbf{D}_u \mathbf{U}_k = \mathbf{c}_u \\
& \mathbf{M}_x \mathbf{X}_k \leq \mathbf{m}_x \\
& \mathbf{D}_x \mathbf{X}_k = \mathbf{c}_x
\end{aligned}$$

Where $\mathbf{Z}$ the block-downshift operator, i.e., a matrix with identity matrices along it's first block sub-diagonal and zeros elsewhere, and $\mathcal{A}, \mathcal{B}$:

$$\mathcal{A} = \begin{bmatrix} \mathbf{I}_{N-1} \otimes \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \quad \mathcal{B} = \begin{bmatrix} \mathbf{I}_{N-1} \otimes \mathbf{B} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$

The notable drawback of this approach is that we increase the number of decision variables and coinstraints in our optimization. However, the problem on other hand become sparse, and some solvers are in fact may smartly exploit these sparsity patterns in order to find solution in the very same time as it for the 'classical MPC'.

LTV Variants of MPC

The yet anothe power of MPC approach is in the ability to tackle LTV dynamics with no serious changes in the algorithm.

For instance for the joint state/control optimization one may just redifine the $\mathcal{A}, \mathcal{B}$:

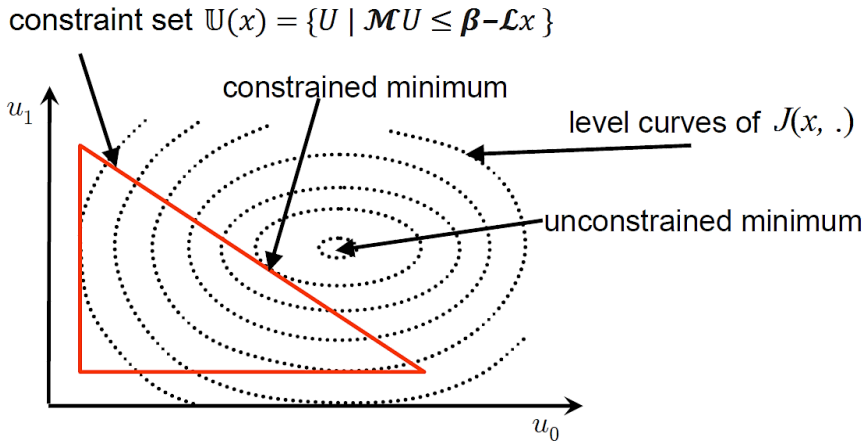
$$\mathcal{A}_k = \text{blockdiag}\{A_k, A_{k+1}, \cdots, A_{k+N-1}, \mathbf{0}\}, \quad \mathcal{B}_k = \text{blockdiag}\{B_k, B_{k+1}, \cdots, B_{k+N-1}, \mathbf{0}\}$$

In similar fashion one may always let constraints $\mathbf{M}_{(\cdot)}, \mathbf{m}_{(\cdot)}, \mathbf{D}_{(\cdot)}, \mathbf{c}_{(\cdot)}$ and cost $\mathbf{Q}, \mathbf{R}$ to be time variant.

This yields a pretty general formulation for constrained LTV problem, that can be used to approximate solution for the for optimal nonlinear tracking, if we let $\mathbf{A}_k, \mathbf{B}_k$ to represent the linearization of nonlinear dynamics along the predefined trajectory.

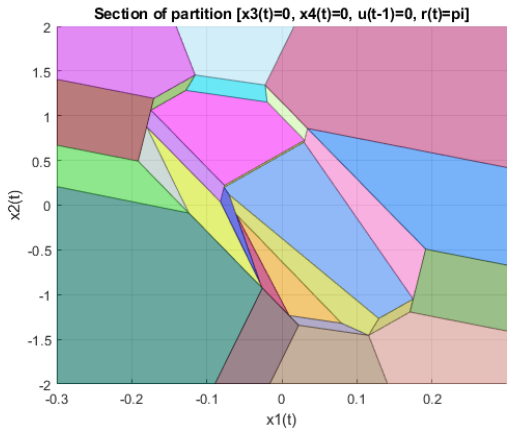
Explicit MPC

The optimal solution of the constrained MPC problem is often on a subset of constraints (called the set of active constraints).



Explicit MPC takes this approach to meet the above constraints and compute solution not in receding horizon but **offline**.

Thus it may be the case that we can somehow partition the state space, knowing constraints on controls and states in advance.



The derivation of explicit MPC is highly rely on multiparametric quadratic programming that is used to pre-solve the QP off-line therefore converting the MPC law into a continuous and piecewise-affine function of the parameter vector. The detailed derivation it is out of scope of this lecture, in order to get some infights please follow the works of Alberto Bemporad, who is leading researcher in this field.

CVXPY to facilitate control design

As we have stated above, there are a lot of solvers to deal with QP problems. However the problem definition itself may become a little tideous. Fortunately nowadays there are a lot of toolboxes dedicated to formulation of the optimization problems in natural way, as you would do with pen and paper. One of thess is `cvxpy`.

Note that cvxpy is not the solver itself, but set of tools that will build the problem for you and provide the interface to the solvers. You may even do the c-code generation with some additional routines like `cvxpygen`, resulting in self contained MPC that can be deployed even on embedded systems.

```
# The CVXPY example of parametric QP problem

import cvxpy as cp

# define dimensions
H, n, m = 10, 6, 3

# define variables
U = cp.Variable((m, H), name='U')
X = cp.Variable((n, H+1), name='X')

# define parameters
Psqrt = cp.Parameter((n, n), name='Psqrt')
Qsqrt = cp.Parameter((n, n), name='Qsqrt')
Rsqrt = cp.Parameter((m, m), name='Rsqrt')
A = cp.Parameter((n, n), name='A')
B = cp.Parameter((n, m), name='B')
x_init = cp.Parameter(n, name='x_init')

# define objective
objective = cp.Minimize(cp.sum_squares(Psqrt@X[:,H-1]) +
                        cp.sum_squares(Qsqrt@X[:,:H]) +
                        cp.sum_squares(Rsqrt@U))

# define constraints


constraints = [X[:,1:] == A@X[:,:H]+B@U,
               cp.abs(U) <= 1,
               X[:,0] == x_init]


# define problem
problem = cp.Problem(objective, constraints)
```

Example:

Kindly follow the following colab to checkout the MPC implemented for the cart pole system.

Pros and Cons of MPC

Pros

- Is based on **very intuitive concepts** and allows easy tuning.
- Can be used to control most processes, from **simple to complex** ones with long delay times, unstable modes etc.
- **Multivariable** case can be easily dealt with.

- Handling of **constraints** is conceptually simple.
- Can naturally compensate for **measurable disturbances**.
- Resulting controller is an **easy-to-implement** control law.
- Is useful when future references (robotics, batch processes etc.) are known.

Cons

- Derivation of control law is relatively more complex.
- Online **computation complexit**y may be significant.
- A **model of the system dynamics** must be available.

Where to go next

There is various extensions and applications of MPC, I highly suggest you to go over:

- LP formulations
- Stability and Feasibility
- Explicit MPC
- Moving Horizon Estimation
- System Level Synthesis