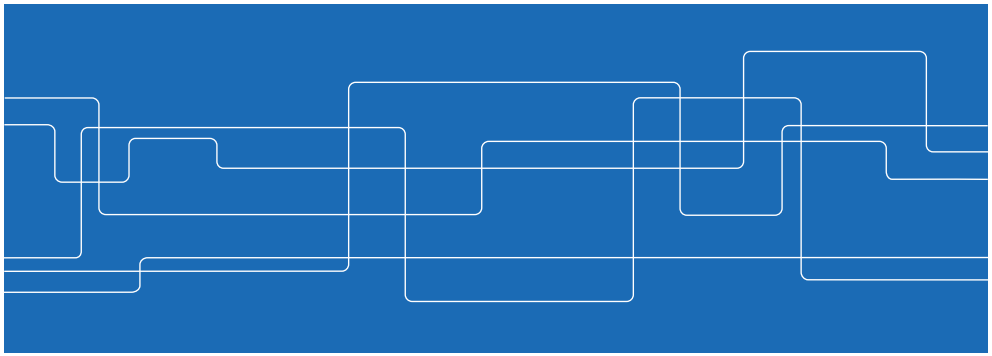
Lecture 12: Implementation issues

Mikael Johansson
KTH - Royal Institute of Technology



Outline

- Getting the model
- Tuning the controller
- Implementation platforms and solver technology

We will also spend some time discussing project presentations.

Getting the model

Throughout the course, we have assumed that a system model is given.

- in practice, both a model and its parameters have to be derived

Three basic approaches:

- *physical modeling*: use laws from physics, chemistry, etc., balance equations and measurements to derive model and its parameters
- *gray-box identification:* use physics to derive (part of) model structure, then estimate parameters from experiments
- *black-box identification*: estimate both model structure and its parameters from data

A rich and useful theory - you can learn much more in EL2820!

Challenges: model order, sufficient excitation, closed-loop

Many traps and pitfalls when estimating models from data:

- model order (number of states) typically not known
- input must excite all frequencies (steps, sinusoidals often not enough)
- identification in closed-loop difficult (models tend to be biased)

Important to validate estimated models on fresh data.

Example: sub-space identification

Given input and output sequences $\{u_t\}$, $\{y_t\}$, estimate A, B, C in

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t \\ y_t &= Cx_t \end{aligned}$$

If $\{x_t\}$ was known, then we could solve least-squares problem

$$\underset{A,B,C}{\text{minimize}} \sum_t \|x_{t+1} - Ax_t - Bu_t\|_2^2 + \|y_t - Cx_t\|_2^2$$

But x_t is typically not known, and neither is its dimension n.

Example: sub-space identification

We know that

$$y_t = CA^t x_0 + \sum_{k=0}^{t-1} CA^k B u_{t-1-k}$$

If $x_0 = 0$, or the system is stable and $t \gg 0$, then

$$y_t \approx \sum_{k=0}^{t-1} CA^k B u_{t-1-k} := \sum_{k=1}^{t} g_k u_{t-k}$$

where

$$g_k := CA^{k-1}B$$

is known as the k:th *impulse response* coefficient.

Can use least-squares to estimate g_k from $\{u_t\}$ and $\{y_t\}$.

Example: sub-space identification

To recover A, B, C from g_k, note that

$$\mathcal{H}_t = \begin{bmatrix} g_1 & g_2 & \dots & g_t \\ g_2 & g_3 & \dots & g_{t+1} \\ \vdots & & & \vdots \\ g_t & g_{t+1} & \dots & g_{2t+1} \end{bmatrix} = \begin{bmatrix} CB & CAB & \dots & CA^tB \\ CAB & CA^2B & \dots & CA^{t+1}B \\ \vdots & & & \vdots \\ CA^{t-1}B & CA^tB & \dots & CA^{2t}B \end{bmatrix} =$$
$$= \mathcal{O}_t \mathcal{C}_t$$

Since controllability and observability matrices have rank n, so does $\mathcal{H}_t$.

- we can therefore estimate the system order by the rank of $\mathcal{H}_t$.

Any X, Y such that $\mathcal{H}_t = XY$ yields (A, B, C) such that $X = \mathcal{O}_t$, $Y = \mathcal{C}_t$

- common to use singular-value decomposition: $\mathcal{H}_t = USV^T$.
- can take $\mathcal{O}_t$ as (first n rows of) US and $\mathcal{C}_t$ as (first n columns of) V^T.

Example: sub-space identification

Several ways to recover A, B and C from $\mathcal{C}_t$.

One approach uses that

$$\begin{aligned} x_{t+1} &= A^t x_0 + \sum_{k=0}^{t-1} A^k B u_{t-k-1} \\ &\approx \sum_{k=0}^{t-1} A^k B u_{t-k-1} = \mathcal{C}_t \begin{bmatrix} u_{t-1} \\ u_{t-1} \\ \vdots \\ u_0 \end{bmatrix} \end{aligned}$$

to create an estimate of $\{x_t\}$. Can then use least-squares to get A, B, C.

(many variations available, this plain vanilla version is not state-of-the art)

System identification for model-predictive control

Sub-space identification yields *one* state-space realization (out of many)

- no guarantee that state-vector corresponds to physical states

Under identification experiments, must measure

- actual outputs to be used by output feedback MPC, and
- process states which we would like to constrain

Often good for insight to transform identified A, B, C so that each output in the identification experiment maps to a single state.

Advised to scale model so that inputs and outputs have unit ranges, i.e.

$$\|u_t\|_\infty \leq 1, \qquad \|y_t\|_\infty \leq 1$$

Outline

- Getting the model
- Tuning the controller
- Implementation platforms and solver technology

Controller tuning

Many parameters to decide:

- sampling time
- performance weights, including final state penalty
- open-loop vs closed-loop prediction models
- control and prediction horizons
- slack variable weighting (when softening constraints)
- terminal set
- observer gains (possibly indirectly through weight matrices)
- disturbance models
- quadratic programming solver technology

Controller tuning: sampling time

Sampling time determined by fastest process dynamics.

Rules of thumb: at least $4 - 10$ samples per target rise time.

Advantage of long sampling interval

- reduced computational load.

Disadvantage of long sampling interval:

- increased phase loss, worse stability margins
- possible issues with inter-sample behavior

If hardware platform admits, better to use 8-10 samples per rise time.

Controller tuning: weight matrices

Convenient to consider unconstrained problem (linear operation) first.

Keep it simple:

- penalize critical signals only (as long as $(A, Q_1^{1/2})$ is observable)
- select initial weights by Bryson's rule, adjust in simulations
- scale control penalty with scalar to adjust bandwidth

Useful to analyze frequency domain properties and sensitivities to uncertainties of linear controller (not covered in class).

Simulate the effect of typical references, disturbances, measurement noises on the control signal and the system output.

Form terminal weight using Riccati-solution to corresponding LQ-problem.

Controller tuning: pre-stabilized predictions

When open-loop system is unstable and prediction horizons long:

- open-loop prediction matrices $A^{k-t-1}B$ "blow up"
- QP problem becomes numerically ill-conditioned, difficult to solve

Better to use pre-stabilized predictions, *i.e.*, let

$$u_t = -Lx_t + v_t$$

and optimize over v_t (deviations from linear control) instead of u_t.

Results in pre-stabilized prediction matrices on the form

$$(A - BL)^{k-t-1}$$

These do not blow up when closed-loop is stable.

Natural to use LQ-optimal control gain as L.

Controller tuning: control and prediction horizon

Determined by the slowest process dynamics

- need to be able to reach terminal set in N steps

In theory, "longer prediction horizon is better", but not always in practice.

- if models are inaccurate, larger horizons may worsen performance.

Suggestion: with $8 - 10$ samples per rise time use at least $N \geq 4 - 5$.

Controller tuning: constraint softening and slack penalities

Should soften constraints whenever allowed to ensure feasibility of QP.

Slack penalty selection related to matrices in QP-problem

- would like slacks to be zero whenever feasible control exists

If system is scaled so that controls and outputs have unit ranges and

$$\begin{array}{ll} \text{minimize} & U^T P U + q^T U + r + \kappa s^T s \\ \text{subject to} & U \in \mathcal{U}, \quad s \geq 0 \end{array}$$

is the QP resulting from our MPC problem, then we advise to set

$$\kappa = 100\lambda_{\max}(P)$$

Controller tuning: terminal set selection

Terminal set: a necessity for theoretical convergence proof.

- effectively imposed by long prediction horizon
- more important when using very short horizons

Natural to use (constraint admissible) invariant set of $u_t = -L_{\mathrm{LQ}} x_t$

- a polyhedral set, represented by a finite number of linear inequalities

Terminal constraint computation done off-line:

- can spend significant computing on finding and simplifying invariant set
- can also consider robustly invariant sets (not covered in this course)

Controller tuning: observer gain selection

Convenient to use a Kalman filter

- defined by disturbance and measurement noise covariance matrices

If observer is used with pole placement,

- make observer bandwidth (significantly) faster than closed-loop
- avoid overshoot and oscillations in observer dynamics

Controller tuning: disturbance models and integral action

Uniform approach to integral action and disturbance compensation:

- modeling disturbances as outputs of linear systems
- estimating the disturbance model states using an observer
- compensating for these in the MPC design

(in this framework, constant disturbance models result in integral action)

Modeling errors may lead to biased state estimates

- may need to measure all critical states (the ones subject to constraints)

Outline

- Getting the model
- Tuning the controller
- Implementation platforms and solver technology

Solver technology

We have claimed that convex optimization is a mature technology.

- essentially true, as long as boundaries are not stretched

However,

- many control applications have fast dynamics,
- and need implementation on resource-constrained embedded platforms

so solver technology selection is still a key issue in many implementations.

First choice is between *explicit* and *embedded* QP-solvers.

Explicit MPC

We have already shown that optimal solution to batch-LQ is linear feedback

$$u_t = -L_{\text{LQ}} x_t$$

Follows since $U^\star = -P_{\text{LQ}}^{-1} q_{\text{LQ}}$ and q_{LQ} depends linearly on x_t.

When adding state and control constraints, we need to solve

$$\begin{array}{ll} \text{minimize} & U^T P_{\text{LQ}} U + 2 q_{\text{LQ}}^T U + r_{\text{LQ}} \\ \text{subject to} & AU \leq b \end{array}$$

where q_{LQ} depends linearly on x_t and $b = b_0 + b_1 x_t$.

Can show that optimal solution is

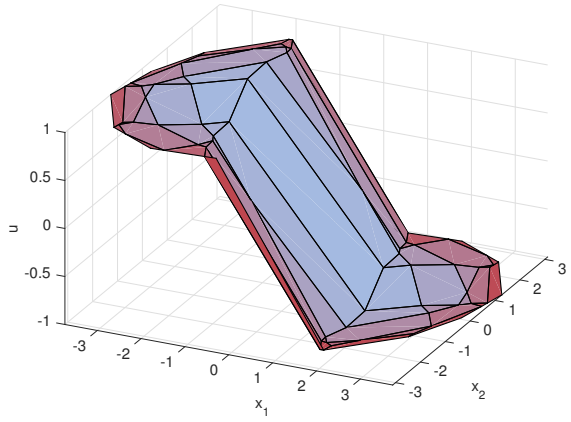
$$u_t = -L_i x_t + l_i \qquad \text{for } x_t \in X_i$$

where $X_i \subseteq \mathbb{R}^n$ are polyhedral subset of the state space.

Can compute X_i, L_i and l_i off-line $\Rightarrow$ explicit (piecewise linear) control law.

Explicit MPC example

Consider simple system from Lecture 8



Explicit MPC example

Number of regions, computation time increases with horizon length.

N	1	3	5	7	9	11
no. regions	5	27	63	115	183	265

And this is for a two-state system!

For the inverted pendulum design in Lecture 6, $N = 5$ gives 32834 regions.

Much more details on presentation day!

Embedded QP solvers

If memory not an issue, interior-point methods (IPM) are state-of-the art.

- several numerically reliable and fast commercial solvers available

Solution times does not only depend on size of decision vectors

- also structure (especially sparsity) of matrices important
- sometimes faster to solve QP where predicted states are *not* eliminated (even though it has $(N-1)n$ additional decision variables)

Best to try both formulations and evaluate memory and CPU consumption.

Some QP solvers have been developed specifically for embedded systems

- very encouraging results, but still not the same solver maturity as IPM
- solvers for nonlinear and hybrid MPC are also evolving quickly!

How fast and large can we go?

Commercial IPM solvers handle millions of variables without problems.

- if sample times on minute-scale and use standard PC for computations,

On embedded platforms used e.g. in automotive industry

- memory is slow and limited;
- can deal with $4-10$ states and sampling times around $20-100$ ms.

On custom hardware (FPGA, parallel processing)

- problems up to 100kHz range have been reported

When we hit the computational limits

Several approaches to reduce computational load

- simplify model (reduce model order) - many techniques available
- increase sample time (a little), reduce horizon (a little)
- reduce freedom in control (so-called move blocking)
- simplify terminal set (typically at the expense of making it smaller)
- model for solver efficiency
  - use only simple bound constraints on decision variables (x, u and s)
  - scale model to improve numerical conditioning

Summary

- Getting the model
- Tuning the controller
- Implementation platforms and solver technology

Project presentations

I would like to talk to all groups:

- who will coordinate the work?
- how will you organize your work?
- when do you plan to meet for the first time?