Distributed Model Predictive Control

Oliver Gäfvert

August 29, 2014

Outline

Model Predictive Control

Given a discrete linear time-invariant state space model

$$\begin{aligned} x(k+t+1|k) &= Ax(k+t|k) + Bu(k+t|k) \\ y(k+t|k) &= Cx(k+t|k) \end{aligned}$$

we can predict future states of the system as a function of the input signals, $u(k+t|k), t \in \mathbb{N}$.

Define A Cost Function

By defining a prediction horizon $N \in \mathbb{Z}$ and a cost function

$$J(\mathbf{u}) = x(k+N|k)^T Q_f x(k+N|k) +$$

$$\sum_{i=0}^{N-1} x(k+i|k)^T Q x(k+i|k) + u(k+i|k)^T R u(k+i|k)$$

where

$$\mathbf{u} = \begin{bmatrix} u(k|k) \\ u(k+1|k) \\ \vdots \\ u(k+N-1|k) \end{bmatrix}$$

we can find the "cheapest" next input signal to control the system by minimizing $J(\mathbf{u})$.

Receding Horizon

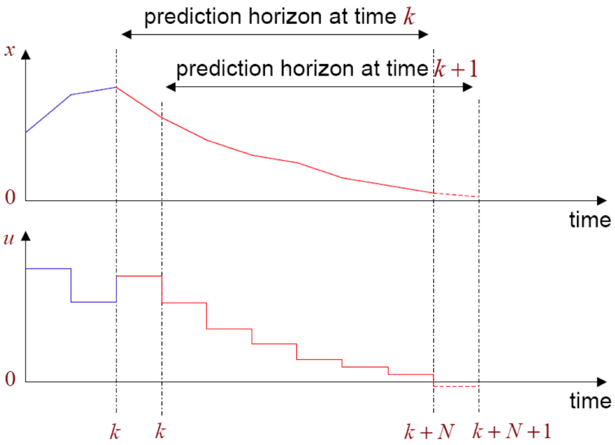


Figure: Image from www.eng.ox.ac.uk/ conmrc/ mpc

Forming a Quadratic Programming Problem

By collecting the terms for $x(k+i|k), i \in [0, N]$ we can express $J(\mathbf{u})$ in the form

$$J(\mathbf{u}) = \mathbf{u}^T H \mathbf{u} + 2x(k|k)^T F^T \mathbf{u}$$

and hence we get the quadratic programming problem

$$\begin{array}{ll} \underset{\mathbf{u}}{\text{minimize}} & \mathbf{u}^T H \mathbf{u} + 2x(k|k)^T F^T \mathbf{u} \\ \text{subject to} & u_{min} \leq u(k+i|k) \leq u_{max}, \ \text{for } i \in [0, N-1] \\ & y_{min} \leq y(k+i|k) \leq y_{max}, \ \text{for } i \in [0, N-1] \end{array}$$

where we can incorporate constraints on the input and output signals.

Distributed Model Predictive Control

Consider a finite collection $\mathcal{T} \subset \mathbb{N}$ of indices describing discrete time-invariant linear systems such that system $i \in \mathcal{T}$ is described by

$$x_i(k+t+1|k) = A_i x_i(k+t|k) + B_i u_i(k+t|k) \quad (1)$$

$$y_i(k+t|k) = C_i x_i(k+t|k)$$

where $x_i \in \mathbb{R}^{p_i}$, $u_i \in \mathbb{R}^{q_i}$, $y_i \in \mathbb{R}^{n_i}$, and system i is coupled to system $j \in \mathcal{T}$ with the constraints

$$E_{ij} y_i(k+t|k) = E_{ji} y_j(k+t|k)$$

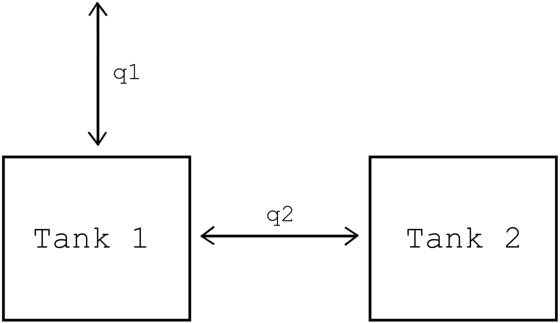
for some $E_{ij} \in \mathbb{R}^{m_{ij} \times n_i}$, $E_{ji} \in \mathbb{R}^{m_{ij} \times n_j}$.

The Local Quadratic Programming Problems

By formulating the MPC problem for each system $i \in \mathcal{T}$ we get the quadratic programming problem

$$\begin{aligned} \underset{\mathbf{u}_i}{\text{minimize}} \quad & J(\mathbf{u}_i) \\ \text{subject to} \quad & u_{min} \leq u(k+t|k) \leq u_{max}, \ \text{for } t \in [0, N-1] \\ & y_{min} \leq y(k+t|k) \leq y_{max}, \ \text{for } t \in [0, N-1] \\ & E_i y_i(k+t|k) = z_i(k+t|k), \ \text{for } t \in [0, N-1] \end{aligned}$$

Example



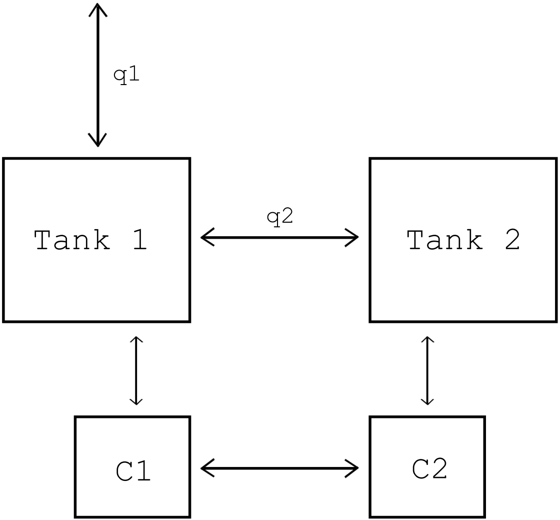
Example

Then we have the global problem

$$x(k+t+1|k) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1(k+t|k) \\ x_2(k+t|k) \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1(k+t|k) \\ u_2(k+t|k) \end{bmatrix}$$

$$y(k+t|k) = x(k+t|k)$$

Example cont.



Example cont.

Can be decomposed into the two subsystems

$$x_1(k+t+1|k) = x_1(k+t|k) + \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} u_1(k+t|k) \\ u_2(k+t|k) \end{bmatrix}$$

$$y_1(k+t|k) = x_1(k+t|k)$$

and

$$x_2(k+t+1|k) = x_2(k+t|k) + u_2(k+t|k)$$

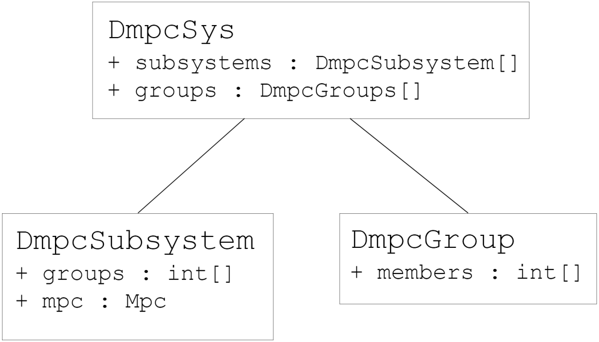
$$y_2(k+t|k) = x_2(k+t|k)$$

where we have the constraint

$$u_2^{(1)}(k+t|k) = u_2^{(2)}(k+t|k)$$

between the two subsystems.

Distributed System Representation



`DmpcSubsystem` describes an MPC controller within a distributed system and `DmpcGroup` represents an abstract connection between two subsystems.

Example

```
sys = DmpcSys(10, 10);
LTI1 = ss(1, [1 -1], 1, [], 1);
params.Q = 1; params.R = 0.1*eye(2);
params.umax = [10; 10];
params.umin = [-10; -10;];
params.ymax = 10; params.ymin = -10;
params.x_0 = -5;
[sys, id1] = sys.addSubsystem(LTI1, params);
```

then add subsystem 2 in a similar way...

Example cont.

```
sys = sys.connect(id1, 2, 'input', id2, 1,
'input');
sys.print();
```

Now we have a representation of our distributed system in a `DmpcSys` object.

Solution Methods

The solution methods implemented in the toolbox are

- Nesterov's Accelerated Gradient Method [2]
- a Multi-step Gradient Method [1]
- the Alternating Direction Method of Multipliers (ADMM) [3]

Dual Decomposition

In the dual formulation of the local quadratic programming problem for each subsystem we get

$$\underset{\lambda_i}{\text{maximize}} \quad \underset{\mathbf{u}_i \in \mathcal{U}_i}{\text{minimize}} \quad J(\mathbf{u}_i) + \lambda_i^T (\tilde{E}_i \mathbf{y}_i - \mathbf{z}_i)$$

The Gradient Ascent Method

$\forall i \in \mathcal{T}$ in parallell

repeat

$\mathbf{u}_i^+ = \arg\min_{\mathbf{u}_i \in \mathcal{U}_i} J(\mathbf{u}_i) + \lambda_i^T(\tilde{E}_i \mathbf{y}_i - \mathbf{z}_i)$

send $\tilde{E}_{ij}\mathbf{y}_i^+$ to all neighbors $j \in \mathcal{N}_i$ and receive $\mathbf{z}_i^+$

$\lambda_i^+ = \lambda_i + \alpha \nabla d(\lambda_i)$

until *convergence*;

Algorithm 1: Gradient ascent algorithm for dual formulation of distributed MPC.

Nesterov's Accelerated Gradient Method

Initialize $\alpha = \frac{\sqrt{5}-1}{2}$, $\gamma = \lambda = 0$
$\forall i \in \mathcal{T}$ in parallell
repeat

$$\mathbf{u}_i^+ = \arg\min_{\mathbf{u}_i \in \mathcal{U}_i} J(\mathbf{u}_i) + \lambda_i^T(\tilde{E}_i \mathbf{y}_i - \mathbf{z}_i)$$

send $\tilde{E}_{ij}\mathbf{y}_i^+$ to all neighbors $j \in \mathcal{N}_i$ and receive $\mathbf{z}_i^+$

$$\gamma_i^+ = \lambda_i + \frac{1}{L}(\tilde{E}_i \mathbf{y}_i - \mathbf{z}_i)$$

$$\alpha^+ = \frac{\alpha}{2}(\sqrt{\alpha^2 + 4} - \alpha)$$

$$\beta = \frac{\alpha(1-\alpha)}{\alpha^2+\alpha^+}$$

$$\lambda_i^+ = \gamma_i + \beta(\gamma_i^+ - \gamma_i)$$

until *convergence*;
Algorithm 2: Accelerated gradient ascent algorithm for dual formulation of distributed MPC.

Multi-step Gradient Method

$\forall i \in \mathcal{T}$ in parallell

repeat

$\mathbf{u}_i^+ = \arg \min_{\mathbf{u}_i \in \mathcal{U}_i} J(\mathbf{u}_i) + \lambda_i^T(\tilde{E}_i \mathbf{y}_i - \mathbf{z}_i)$

send $\tilde{E}_{ij}\mathbf{y}_i^+$ to all neighbors $j \in \mathcal{N}_i$ and receive $\mathbf{z}_i^+$

$\lambda_i^+ = \lambda_i + \alpha \nabla d(\lambda_i) + \beta(\lambda_i - \lambda_i^-)$

until *convergence*;

Algorithm 3: Multi-step gradient method for dual formulation of distributed MPC.

The Alternating Direction Method of Multipliers (ADMM)

$\forall i \in \mathcal{T}$ in parallell

repeat

$$\mathbf{u}_i^+ = \arg\min_{\mathbf{u}_i \in \mathcal{U}_i} J(\mathbf{u}_i) + \lambda_i^T(\tilde{E}_i \mathbf{y}_i - \mathbf{z}_i) + \frac{\rho}{2}||E_i \mathbf{y}_i - \mathbf{z}_i||_2^2$$

send $E_{ij}\mathbf{y}_i^+ + \lambda_i/\rho$ to all $j \in \mathcal{N}_i$ and receive $E_{ji}\mathbf{y}_j^+ + \lambda_j/\rho$

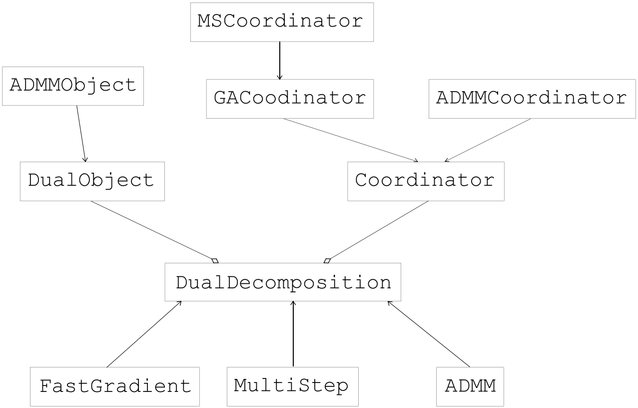
$$\mathbf{z}_i^+ = \frac{\sum_{j\in\mathcal{N}_i} E_{ij}^T(E_{ij}\mathbf{y}_i^+ + \lambda_i/\rho + E_{ji}\mathbf{y}_j^+ + \lambda_j/\rho)}{2}$$

$$\lambda_i^+ = \lambda_i + \rho(E_i\mathbf{y}_i^+ - \mathbf{z}_i^+)$$

until *convergence*;

Algorithm 4: ADMM for dual formulation of distributed MPC.

Solution Method Representation



Example

```
maxIter = 500; tol = 0.001;
n_simulations = 10; iter = [];

sol = FastGradient(sys, maxIter, tol);
sol = sol.sim(n_simulations);
iter = [iter ; sol.histIter];
```

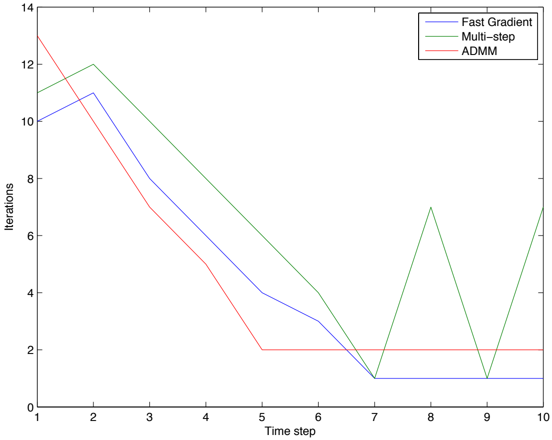
Example

```
sol = MultiStep(sys, maxIter, tol);
sol = sol.sim(n_simulations);
iter = [iter ; sol.histIter];

sol = ADMM(sys, maxIter, tol);
sol = sol.sim(n_simulations);
iter = [iter ; sol.histIter];
```

Example

```
plot(1:n_simulations, iter)
legend('Fast Gradient', 'Multi-step', 'ADMM')
```



References

- Euhanna Ghadimi, Imam Shames, Mikael Johansson.
  Multi-step gradient methods for networked optimization.
  IEEE, 2013.
- Y. Nesterov.
  Introductory lectures on convex optimization.
  Springer, 2004.
- Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato and Jonathan Eckstein.
  Distributed optimization and statistical learning via the alternating direction method of multipliers.
  Foundations and Trends in Machine Learning Vol. 3 No. 1, 2010.