MECH 6v29.002 – Model Predictive Control

L6 – MPC Formulation and Extensions

Outline

- Basic MPC Formulation
- Matlab Implementation
- Extensions
  - Constraints
  - Soft Constraints
  - Prediction Horizons
  - Reference Tracking
  - Preview
  - Rejection of Measured Disturbances
  - Rejection of Unmeasured Disturbances
  - Time-delays
    - Computational
    - Input

Alberto Bemporad. “Model Predictive Control Design: New Trends and Tools.” CDC, 2006.

Basic MPC Formulation

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N$$

$$s.t.$$

$$x_{k+1} = Ax_k + Bu_k, \; k \in \{0,1,\ldots,N-1\}$$

$$x_0 = x(0)$$

- Online Optimization
  - Formulate using Matlab/Yalmip
  - Yalmip does the heavy lifting in conversion from user-based form to optimization-based form
  - Yalmip then passes optimization problem to solver
    - Linear Program – linprog (Matlab)
    - Quadratic Program – quadprog (Matlab), GUROBI, others
    - Nonlinear Program – fmincon (Matlab), others

Basic MPC Formulation (cont.)

- Matlab/Yalmip Code

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N$$

$$s.t.$$

$$x_{k+1} = A x_k + B u_k, \; k \in \{0,1,\dots,N-1\}$$

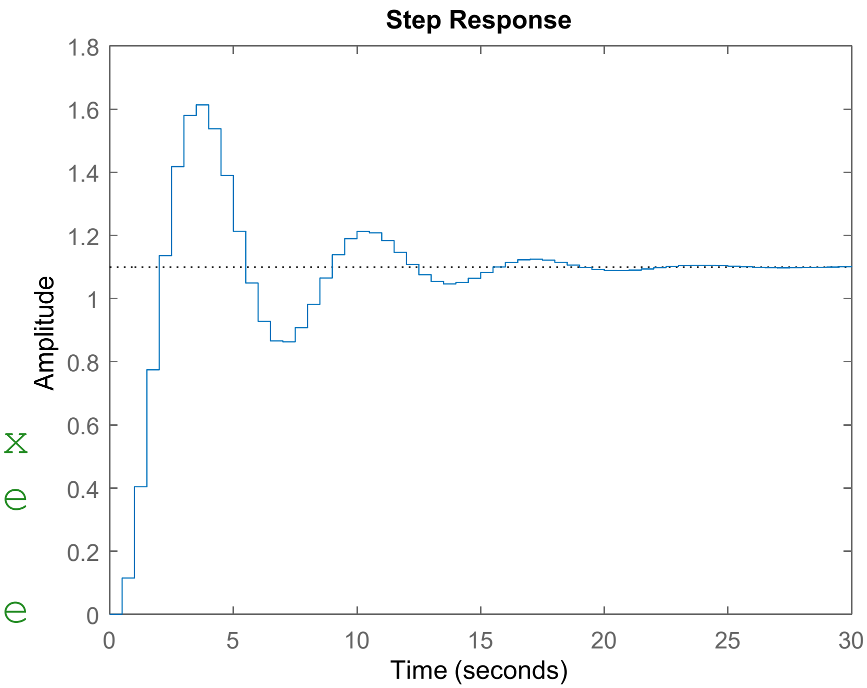
$$x_0 = x(0)$$

Example used throughout this lecture

$$x_{k+1} = \begin{bmatrix} 1.6 & -0.8 \\ 1 & 0 \end{bmatrix} x_k + \begin{bmatrix} 0.5 \\ 0 \end{bmatrix} u_k$$

$$y_k = \begin{bmatrix} 0.23 & 0.21 \end{bmatrix} x_k \qquad \Delta t = 0.5$$

```
%% Define System
n = 2;                  % Number of states
m = 1;                  % Number of inputs
A = [1.6 -0.8; 1 0];    % State matrix
B = [0.5; 0];           % Input matrix
C = [0.23 0.21];        % Output matrix
D = [0];                % Feedthrough matrix
dt = 0.5;               % Discrete step size
x0 = [0;0];             % Initial condition
step(ss(A,B,C,D,dt))    % Plot step response
```



Basic MPC Formulation (cont.)

- Matlab/Yalmip Code

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N$$

$$s.t.$$

$$x_{k+1} = Ax_k + Bu_k,\ k \in \{0,1,\ldots,N-1\}$$

$$x_0 = x(0)$$

```
%% Define Objective Function Parameters
Q = eye(n);   % State penalties
R = eye(m);   % Input penalties
P = Q;        % Terminal cost
N = 15;       % Prediction horizon

%% Controller Formulation in Yalmip

% Define decision variables
x_ = sdpvar(repmat(n,1,N+1), repmat(1,1,N+1));
u_ = sdpvar(repmat(m,1,N),   repmat(1,1,N));

% Initialize constraints and objectives
constraints = [];
objective = 0;

% State costs and constraints
for k = 1:N
    objective = objective + x_{k}'*Q*x_{k} + u_{k}'*R*u_{k};
    constraints = [constraints, x_{k+1} == A*x_{k} + B*u_{k}];
end
% Terminal Cost
objective = objective + x_{N+1}'*P*x_{N+1};
```

```
>> constraints
++++++++++++++++++++++++++++++++++++
|    ID|                  Constraint|
++++++++++++++++++++++++++++++++++++
|    #1|   Equality constraint 2x1|
|    #2|   Equality constraint 2x1|
|    #3|   Equality constraint 2x1|
|    #4|   Equality constraint 2x1|
|    #5|   Equality constraint 2x1|
|    #6|   Equality constraint 2x1|
|    #7|   Equality constraint 2x1|
|    #8|   Equality constraint 2x1|
|    #9|   Equality constraint 2x1|
|   #10|   Equality constraint 2x1|
|   #11|   Equality constraint 2x1|
|   #12|   Equality constraint 2x1|
|   #13|   Equality constraint 2x1|
|   #14|   Equality constraint 2x1|
|   #15|   Equality constraint 2x1|
++++++++++++++++++++++++++++++++++++
```

```
>> objective
Quadratic scalar (real, homogeneous, 47 variables)
```

Basic MPC Formulation (cont.)

- Matlab/Yalmip Code

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N$$

$$s.t.$$

$$x_{k+1} = Ax_k + Bu_k,\ k \in \{0,1,\ldots,N-1\}$$

$$x_0 = x(0)$$

```
% Specify Solver Settings
opts = sdpsettings('solver','gurobi');

% Specify Controller Inputs and Outputs
inputs = {x_{1}};
outputs = {[u_{1}]};

% Create controller
controller = optimizer(constraints,objective,opts,inputs,outputs);
```

```
>> controller
Optimizer object with 2 inputs (1 blocks) and 1 outputs (1 blocks). Solver: GUROBI-GUROBI
```

Basic MPC Formulation (cont.)

- Matlab/Yalmip Code

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N$$

$$s.t.$$

$$x_{k+1} = Ax_k + Bu_k,\ k \in \{0,1,\ldots,N-1\}$$

$$x_0 = x(0)$$

```
% Simulate Closed-loop System
% Define simulation length
N_sim = 20;
% Initialize state and input
trajectories
x0 = [10;10];
x_sim = [x0];
u_sim = [];
% Step through simulation
for i = 1:N_sim
    x0 = x_sim(:,i);
    inputs = {x0};
    u = controller{inputs};
    u_sim = [u_sim u];
    x_sim = [x_sim A*x0+B*u];
end
```

```
% Plot
figure;
subplot(2,1,1);hold on
stairs(0:N_sim,C*x_sim)
stairs([0 N_sim],[0 0],'--k')
xlabel('Time (s)')
ylabel('Output')
subplot(2,1,2);hold on
stairs(0:N_sim-1,u_sim)
stairs([0 N_sim],[0 0],'--k')
xlabel('Time (s)')
ylabel('Input')
```

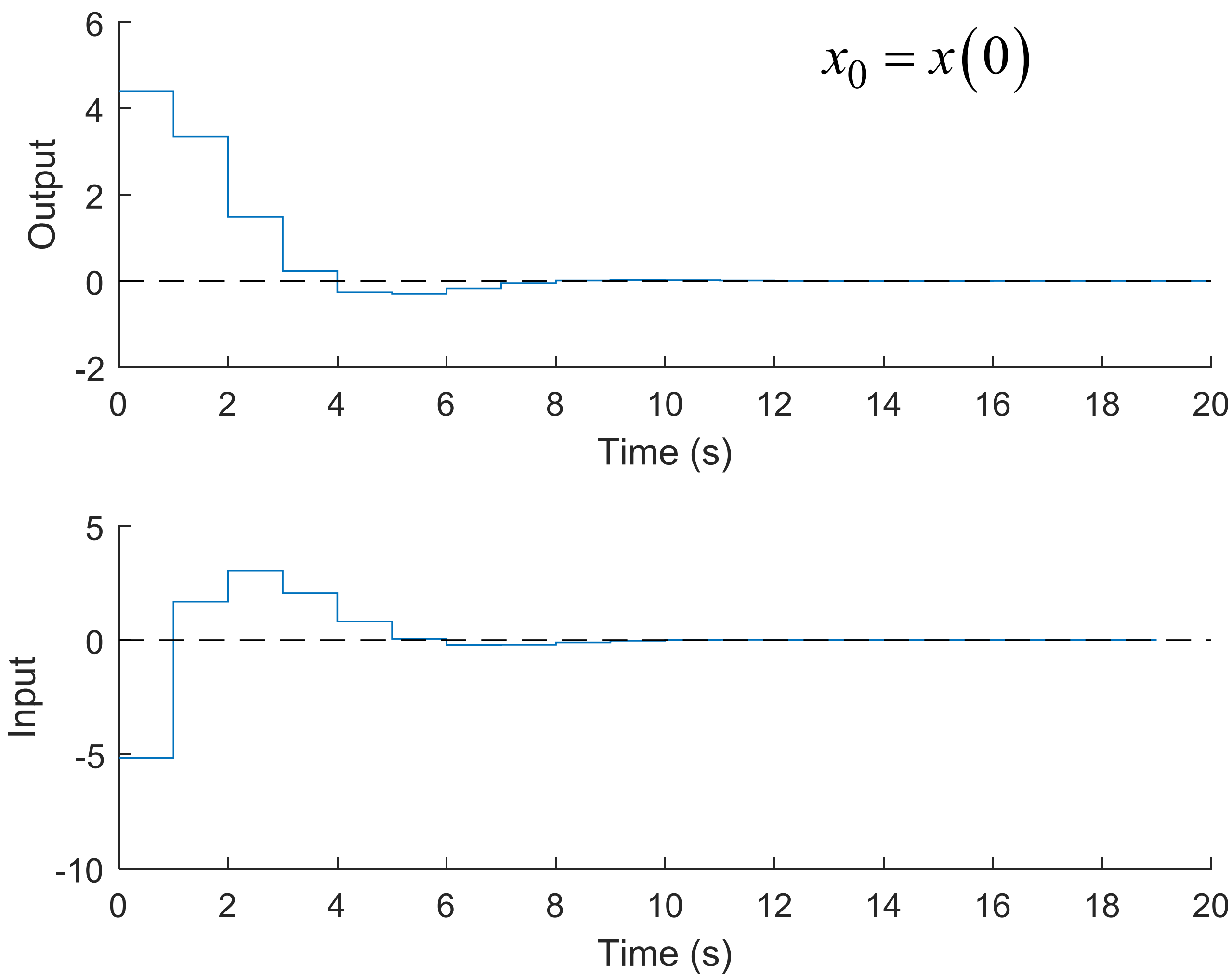
Basic MPC Formulation (cont.)

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N$$

$$s.t.$$

$$x_{k+1} = A x_k + B u_k, \; k \in \{0,1,\ldots,N-1\}$$

$$x_0 = x(0)$$



Yalmip Reformulation

- Yalmip allows you to formulate the MPC problem based on how you would write it
- Solvers typically need the optimization problem in a specific form

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N$$

$$s.t.$$

$$x_{k+1} = Ax_k + Bu_k,\ k \in \{0,1,\dots,N-1\}$$

$$x_0 = x(0)$$

```
% Create controller
controller = optimizer(constraints,objective,opts,inputs,outputs);
```

$$J_0^*(x_0) = \min_{U_0} U_0^T H U_0 + 2x_0^T F U_0 + x_0^T Y x_0$$

$$s.t.$$

$$GU_0 \le W + Sx_0$$

Constraints

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N$$

$$s.t.$$

$$x_{k+1} = Ax_k + Bu_k, \ k \in \{0,1,\ldots,N-1\}$$

$$u_{\min} \le u_k \le u_{\max}, \quad k \in \{0,1,\ldots,N-1\}$$

$$y_{\min} \le Cx_k \le y_{\max}, \ k \in \{1,2,\ldots,N\}$$

$$x_0 = x(0)$$

```
% State costs and constraints
for k = 1:N
    objective = objective + x_{k}'*Q*x_{k} + u_{k}'*R*u_{k};
    constraints = [constraints, x_{k+1} == A*x_{k} + B*u_{k}];
    constraints = [constraints, u_min <= u_{k} <= u_max];
    constraints = [constraints, y_min <= C*x_{k+1} <= y_max];
end
```

Soft Constraints

- Constraints create the potential of infeasibility
  - The set of solutions to the optimal control problem is empty
  - Solver will let you know, output NAN
- This cannot happen in practice, always need a solution
- Relax the constraints

Large penalty

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + \lambda s_k^2 + x_N^T P x_N$$

$W_{\min}, W_{\max} \in \mathbb{R}_+^p$

Relative softening

$$\begin{aligned}
&s.t.\\
&x_{k+1} = Ax_k + Bu_k, \ k \in \{0,1,\ldots,N-1\}\\
&u_{\min} \le u_k \le u_{\max}, \quad k \in \{0,1,\ldots,N-1\}\\
&y_{\min} - W_{\min} s_k \le Cx_k \le y_{\max} + W_{\max} s_k, \ k \in \{1,2,\ldots,N\}\\
&0 \le s_k, \quad k \in \{1,2,\ldots,N\}\\
&x_0 = x(0)
\end{aligned}$$

Positive slack

Prediction Horizons

- Every decision variable and every constraint adds complexity to the optimization problem
  - Typically not a problem for small systems
  - Big problem when using Explicit MPC
- Introduce multiple horizons
  - Prediction horizon, N
  - Input horizon, N_u $\quad N_u \leq N$
  - Constraint horizon, N_c $\quad N_c \leq N$

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N$$

$$\begin{aligned}
&s.t. \\
&x_{k+1} = Ax_k + Bu_k,\ k \in \{0,1,\ldots,N-1\} \\
&u_k = 0,\ k \in \{N_u,\ldots,N-1\} \\
&u_{\min} \leq u_k \leq u_{\max},\quad k \in \{0,1,\ldots,N_c\} \\
&y_{\min} \leq Cx_k \leq y_{\max},\ k \in \{1,2,\ldots,N_c\} \\
&x_0 = x(0)
\end{aligned}$$

Or, hold constant input

$$u_k = u_{N_u-1},\ k \in \{N_u,\ldots,N-1\}$$

Or, use LQR Solution

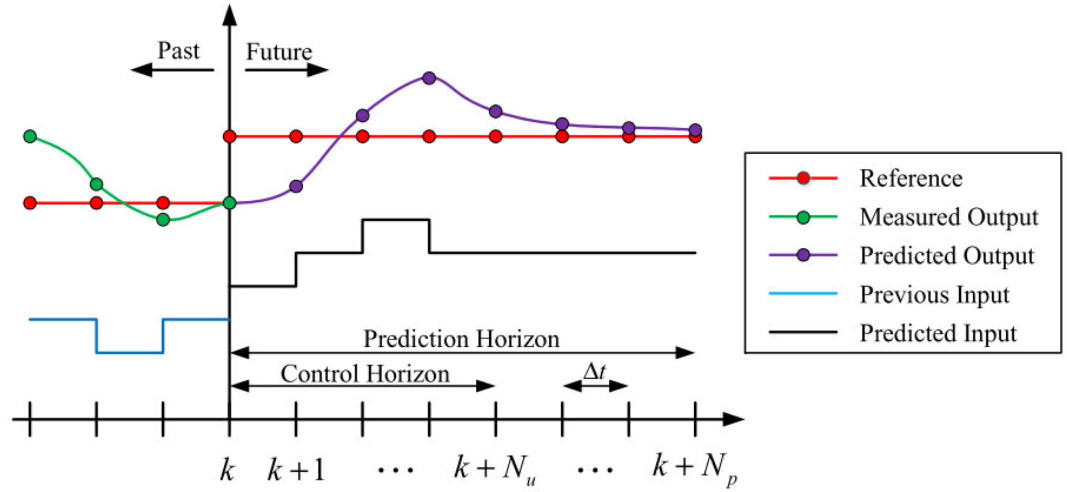
$$u_k = K_\infty x_k,\ k \in \{N_u,\ldots,N-1\}$$

Reference Tracking

- Currently, we are driving the system to the origin
- Frequently, we prefer to track an output reference

$$y_k = Cx_k \rightarrow r_k$$

Known reference



- Natural thing to do would be to modify cost function

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N \quad \Rightarrow \quad J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} (y_k - r_k)^T Q_y (y_k - r_k)$$

- However, this does not put any cost on the inputs.
    - Could lead to highly oscillatory or unstable behavior

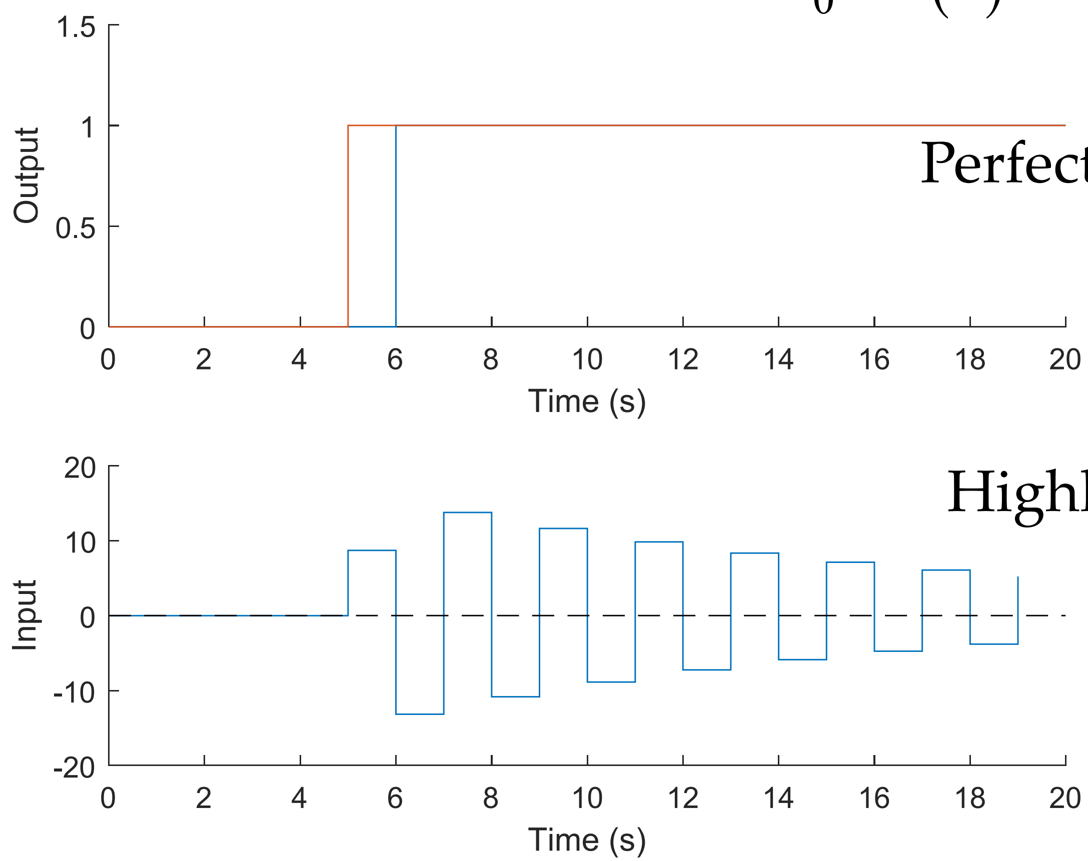
Reference Tracking (cont.)

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} (y_k - r_k)^T Q_y (y_k - r_k)$$

$$s.t.$$

$$x_{k+1} = Ax_k + Bu_k, \; k \in \{0,1,\ldots,N-1\}$$

$$x_0 = x(0)$$



Perfect tracking

Highly oscillatory input (undesirable)

Reference Tracking (cont.)

- Try adding back input costs

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} (y_k - r_k)^T Q_y (y_k - r_k) + u_k^T R u_k$$

- But we need to be careful about what we are asking the system to do
  - Currently, we want $y_k = Cx_k \to r_k$ and $u_k \to 0$
- Examine steady-state behavior

$$x_{k+1} = Ax_k + Bu_k \qquad x_{ss} = Ax_{ss} + Bu_{ss} \qquad \begin{bmatrix} A-I & B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_{ss} \\ u_{ss} \end{bmatrix} = \begin{bmatrix} 0 \\ y_{ss} \end{bmatrix}$$

$$y_k = Cx_k \qquad r_{ss} = Cx_{ss}$$

Can’t do both unless the reference is zero.

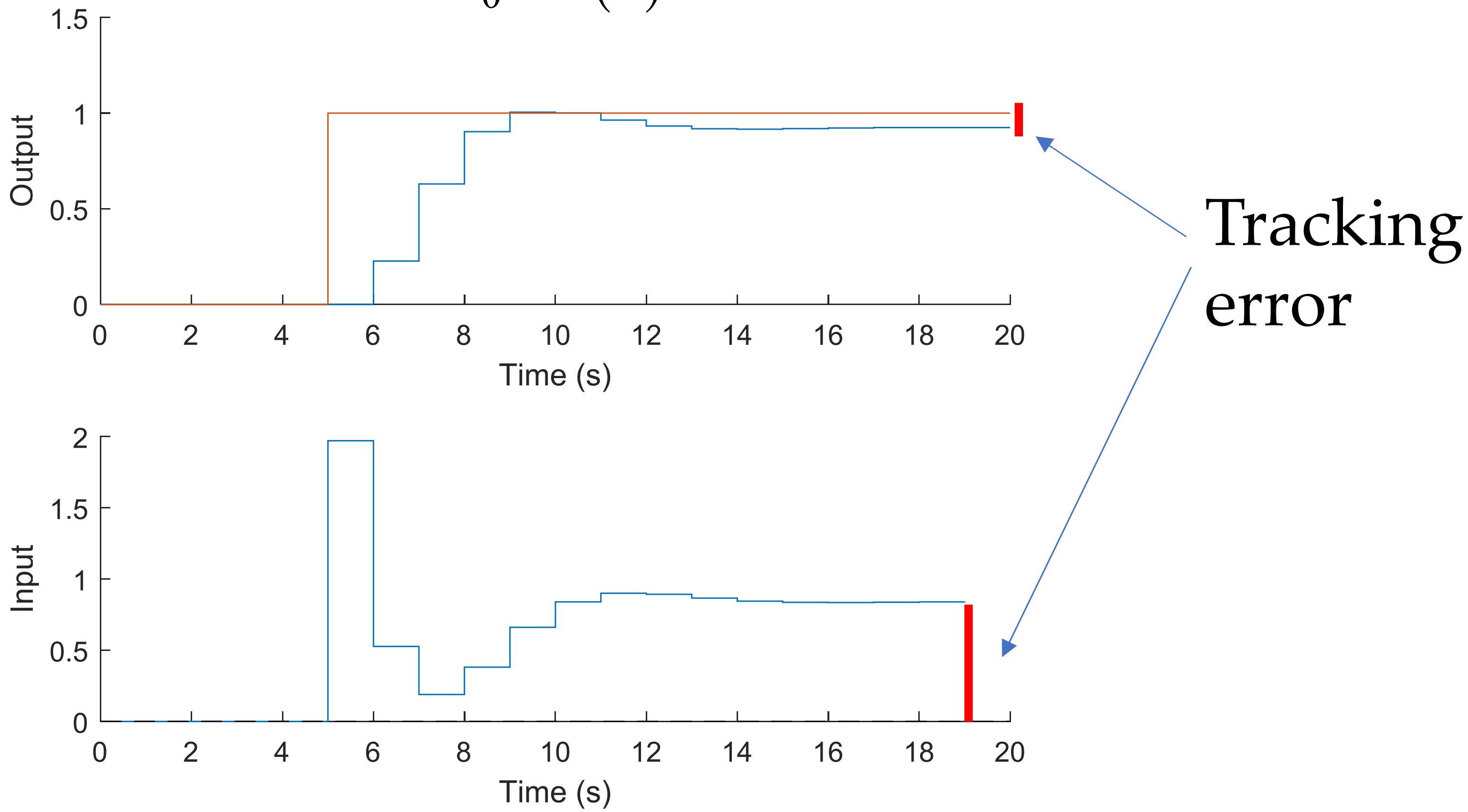
Reference Tracking (cont.)

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} (y_k - r_k)^T Q_y (y_k - r_k) + u_k^T R u_k$$

$$s.t.$$

$$x_{k+1} = Ax_k + Bu_k,\ k \in \{0,1,\ldots,N-1\}$$

$$x_0 = x(0)$$



Reference Tracking (cont.)

- Simple solution:
  - Penalize rate of change of input $\Delta u_k = u_k - u_{k-1}$

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} (y_k - r_k)^T Q_y (y_k - r_k) + \Delta u_k^T R \Delta u_k$$

  - Can add as new states to state space model $u_k = u_{k-1} + \Delta u_k$

$$\begin{bmatrix} x_{k+1} \\ u_k \end{bmatrix} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} + \begin{bmatrix} B \\ I \end{bmatrix} \Delta u_k$$

$$y_k = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix}$$

Reference Tracking (cont.)

- Or just change Yalmip formulation:

```
%% Reference Tracking
p = ; % Number of outputs
Qy = ;% Output tracking cost
% Define decision variables
x_ = sdpvar(repmat(n,1,N+1), repmat(1,1,N+1));
u_ = sdpvar(repmat(m,1,N),   repmat(1,1,N));
r_ = sdpvar(repmat(p,1,N),   repmat(1,1,N));
uOld_ = sdpvar(m,1); % Previous input

% Initialize constraints and objectives
constraints = [];
objective = 0;

% State costs and constraints
for k = 1:N
    if k == 1
        objective = objective + (C*x_{k}-r_{k})'*Qy*(C*x_{k}-r_{k}) + (u_{k}-uOld_)'*R*(u_{k}-uOld_);
    else
        objective = objective + (C*x_{k}-r_{k})'*Qy*(C*x_{k}-r_{k}) + (u_{k}-u_{k-1})'*R*(u_{k}-u_{k-1});
    end
    constraints = [constraints, x_{k+1} == A*x_{k} + B*u_{k}];
end
% Specify Solver Settings
opts = sdpsettings('solver','gurobi');

% Specify Controller Inputs and Outputs
inputs = {x_{1},[r_{:}],uOld_};
outputs = {[u_{1}]};

% Create controller
controller = optimizer(constraints,objective,opts,inputs,outputs);
```

Time-varying known reference signal

Tracking cost

Rate of input cost

Add references and previous input to controller inputs

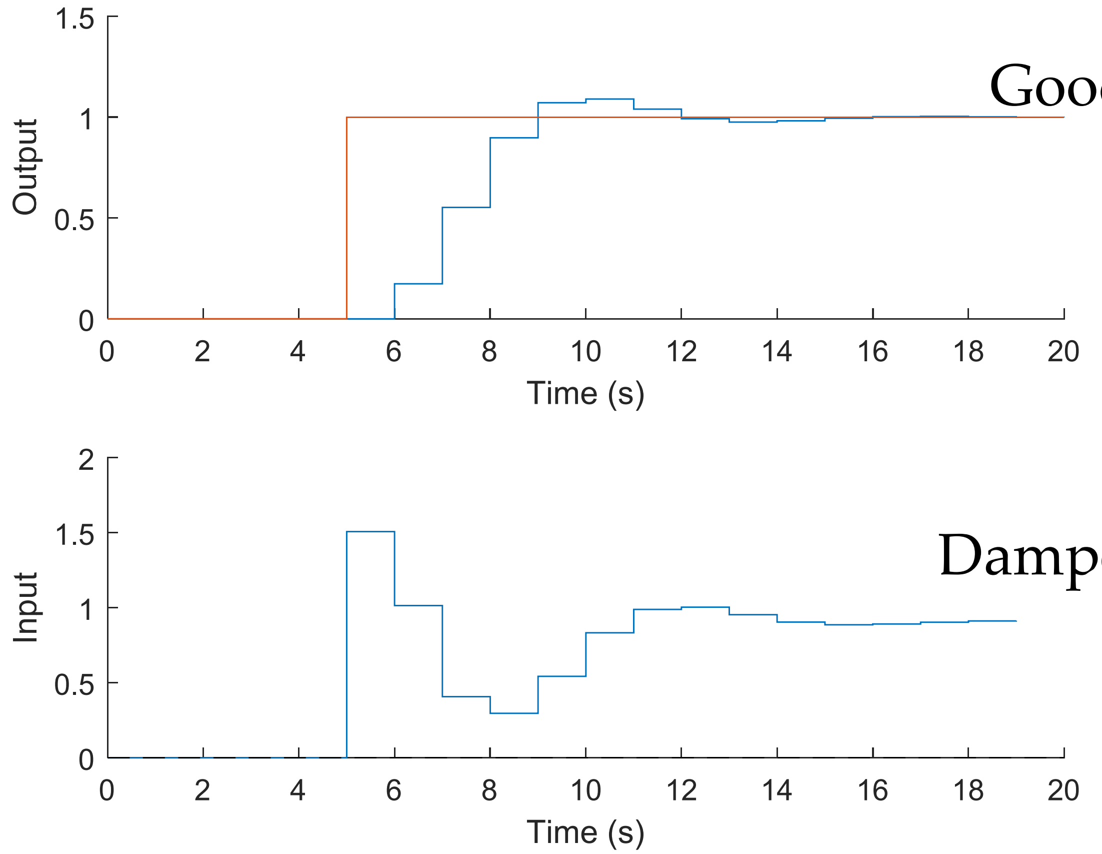
Reference Tracking (cont.)

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} (y_k - r_k)^T Q_y (y_k - r_k) + \Delta u_k^T R \Delta u_k$$

$$s.t.$$

$$x_{k+1} = Ax_k + Bu_k,\ k \in \{0,1,\dots,N-1\}$$

$$x_0 = x(0)$$

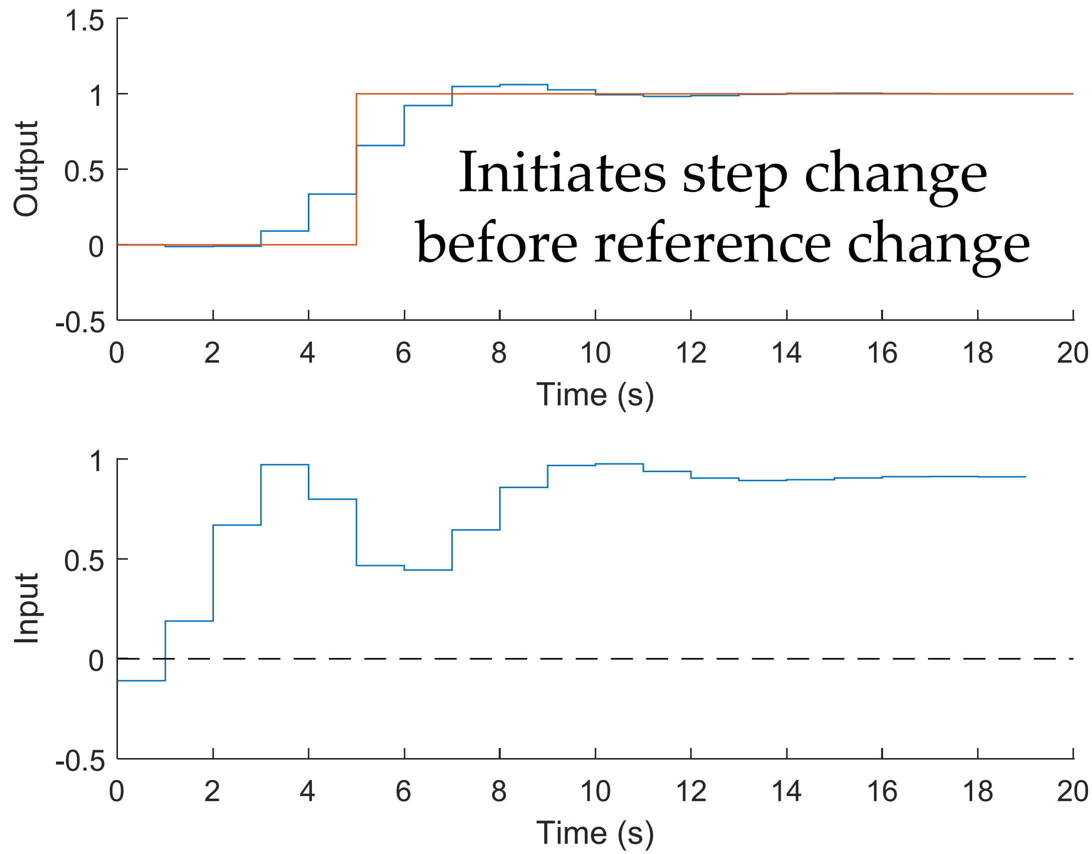


Good tracking

Damped input trajectory

Preview

- We just saw how known references can be provided to the controller
- They can either be constant or time-varying
- If a change in reference is known before it occurs, this can be used by the controller to act in what appears to be a feedforward or non-causal way



- This can be used for known disturbances in a very similar way

$$x_{k+1} = Ax_k + Bu_k + V\hat{d}_k$$
$$y_k = Cx_k$$

Unknown Disturbances

- Let there be an unknown disturbance d_k

$$x_{k+1} = Ax_k + Bu_k + Vd_k$$
$$y_k = Cx_k$$

- We can create an extended state vector and estimate the state and disturbance using standard methods for linear observer design (e.g. Kalman filter)

$$\tilde{x}_k = \begin{bmatrix} x_k \\ d_k \end{bmatrix}$$

- If reference tracking is the goal, we may not care about estimating the disturbance, all we want is $y_k = Cx_k \rightarrow r_k$
- Add "output integrators" $e_{k+1} = e_k + \left(y_k - r_k\right)$
  - Formulate MPC based on

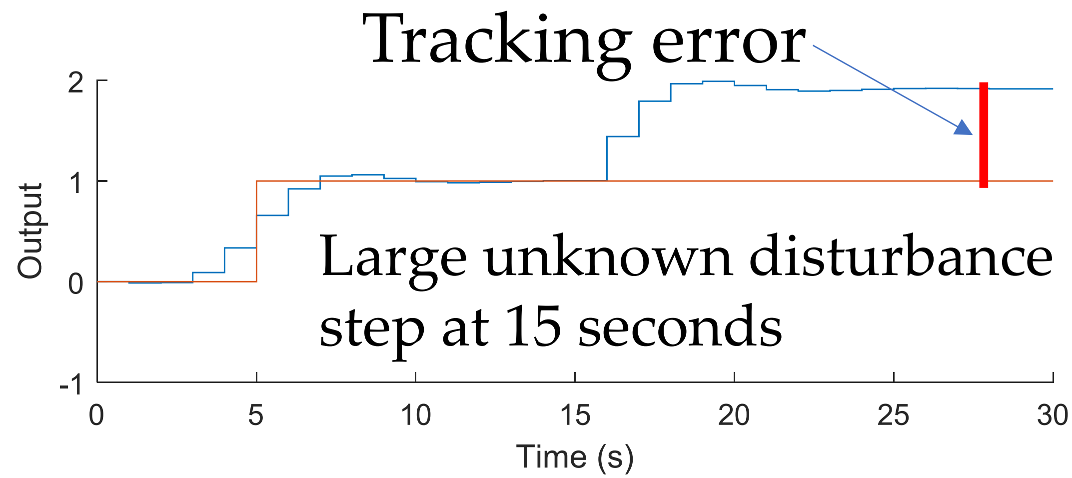
$$\begin{bmatrix} x_{k+1} \\ e_{k+1} \end{bmatrix} = \begin{bmatrix} A & 0 \\ C & I \end{bmatrix} \begin{bmatrix} x_k \\ e_k \end{bmatrix} + \begin{bmatrix} B & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} u_k \\ r_k \end{bmatrix}$$

$$y_k = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x_k \\ e_k \end{bmatrix}$$

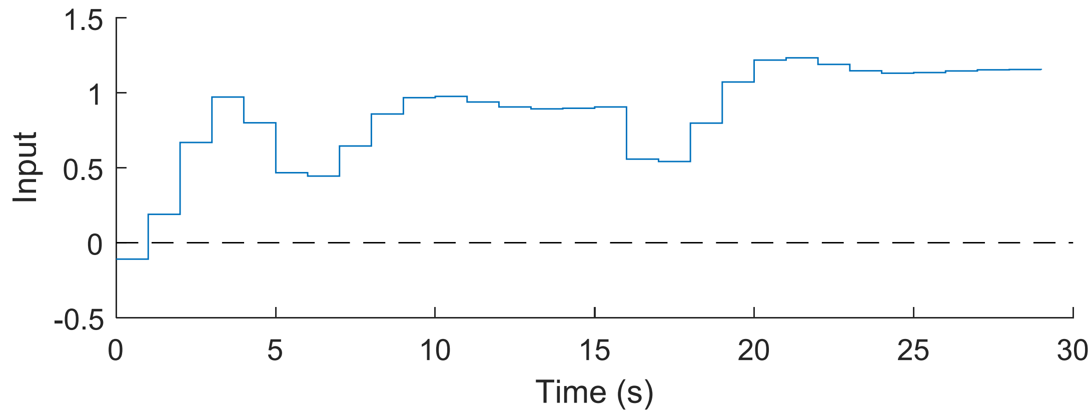
Unknown Disturbances (cont.)

- Unmodified case

```
% Unknown disturbance trajectory
d = zeros(1,N_sim);
d(:,16:end) = 1;
```




```
% Step through simulation
for i = 1:N_sim
    x0 = x_sim(:,i);
%      ref = repmat(r(:,i),1,N); % No
preview
    ref = r(:,i:i+N-1);       % With
preview
    inputs = {x0,ref,uOld};
    u = controller{inputs};
    u_sim = [u_sim u];
    x_sim = [x_sim A*x0+B*u+[1;1]*d(i)];
    uOld = u;
end
```



Unknown Disturbances (cont.)

```
p = 1; % Number of outputs
Qy = 10*eye(p); % Output tracking cost
Qe = 100*eye(p); % Output tracking cost
% Define decision variables
x_ = sdpvar(repmat(n,1,N+1), repmat(1,1,N+1));
e_ = sdpvar(repmat(p,1,N+1), repmat(1,1,N+1));
u_ = sdpvar(repmat(m,1,N),   repmat(1,1,N));
r_ = sdpvar(repmat(p,1,N),   repmat(1,1,N));
uOld_ = sdpvar(m,1); % Previous input



% State costs and constraints
for k = 1:N
    if k == 1
        objective = objective + (C*x_{k}-r_{k})'*Qy*(C*x_{k}-r_{k}) + e_{k}'*Qe*e_{k} +
(u_{k}-uOld_)'*R*(u_{k}-uOld_);
    else
        objective = objective + (C*x_{k}-r_{k})'*Qy*(C*x_{k}-r_{k}) + e_{k}'*Qy*e_{k} +
(u_{k}-u_{k-1})'*R*(u_{k}-u_{k-1});
    end
    constraints = [constraints, x_{k+1} == A*x_{k} + B*u_{k}];
    constraints = [constraints, e_{k+1} == C*x_{k} + e_{k} - r_{k}];
end
% Specify Solver Settings
opts = sdpsettings('solver','gurobi');

% Specify Controller Inputs and Outputs
inputs = {x_{1},e_{1},[r_{:}],uOld_};
outputs = {[u_{1}]};
```

Integral of tracking error cost

Integral of tracking error trajectory

Integral of tracking error dynamics

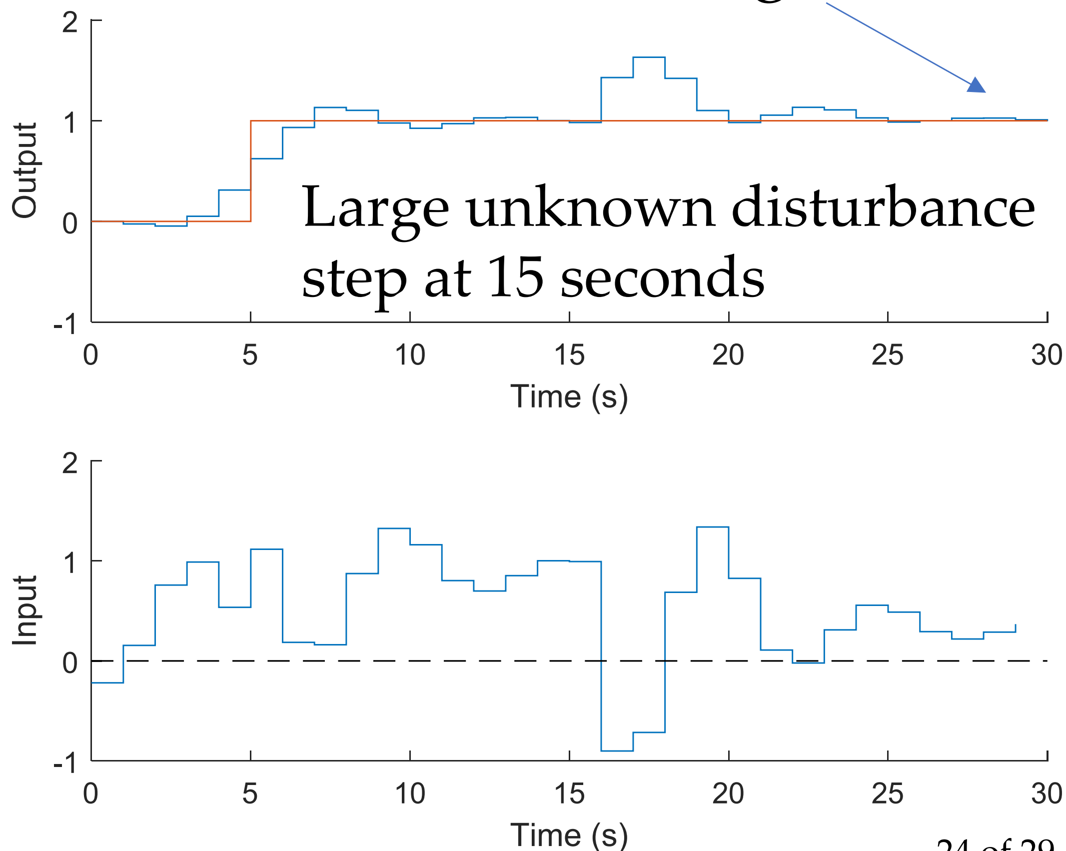
Input initial integral of tracking error

Unknown Disturbances (cont.)

```
% Initialize state and input trajectories
x0 = [0;0];
e0 = 0;
uOld = 0;
x_sim = [x0];
u_sim = [];
% Step through simulation
for i = 1:N_sim
    x0 = x_sim(:,i);
%     ref = repmat(r(:,i),1,N); % No preview
    ref = r(:,i:i+N-1);      % With preview
    e0 = e0 + C*x_sim(:,i) - r(:,i);
    inputs = {x0,e0,ref,uOld};
    u = controller{inputs};
    u_sim = [u_sim u];
    x_sim = [x_sim A*x0+B*u+[1;1]*d(i)];
    uOld = u;
end
```

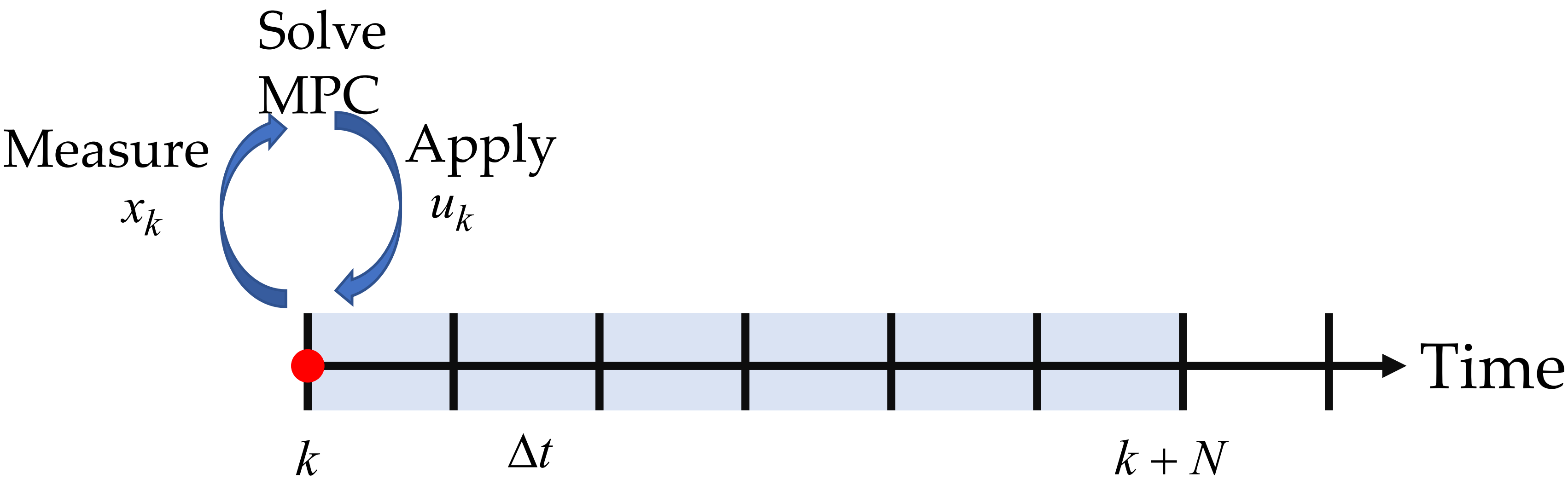
Integral of tracking error dynamics

Zero tracking error

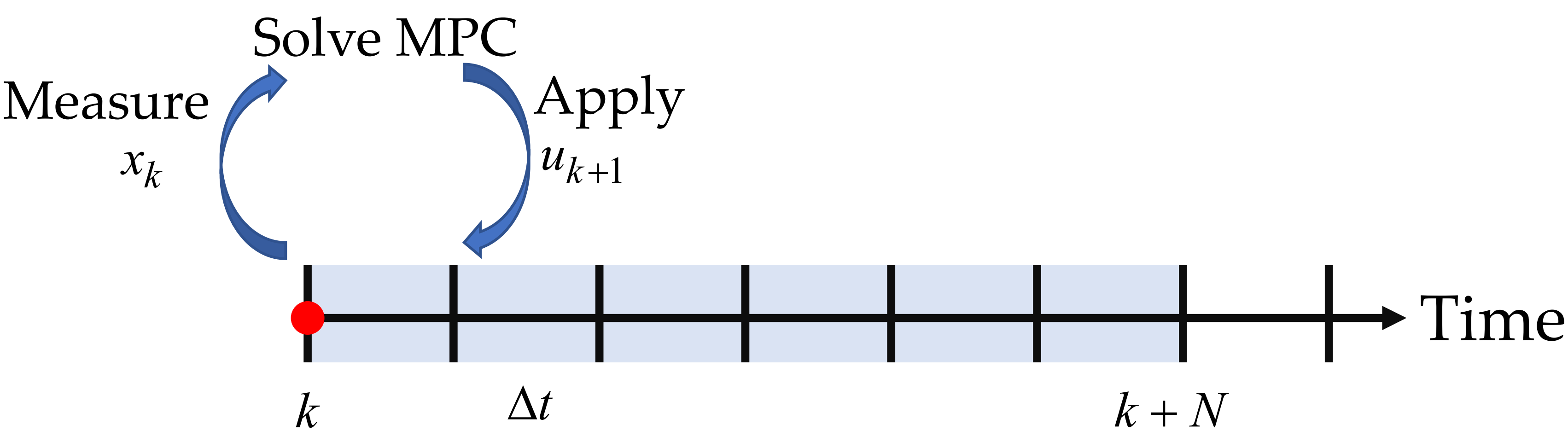


Time Delay – Computational

- In general, we assume instantaneous calculations



- However, in practice we solve optimization problem between discrete updates



Time Delay – Computational

- Modify MPC formulation

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N$$
$$s.t.$$
$$x_{k+1} = Ax_k + Bu_k,\ k \in \{0,1,\ldots,N-1\}$$
$$x_0 = x(0)$$

$$\Rightarrow$$

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N$$
$$s.t.$$
$$x_{k+1} = Ax_k + Bu_k,\ k \in \{0,1,\ldots,N-1\}$$
$$x_0 = x(0)$$
$$\color{red}{u_0 = u(0)}$$

- Initial input is fixed and provided as an input to the optimization problem
    - This input is what is effecting the system while the optimization problem is being solved
    - This input corresponds to the optimal solution at the previous time step $u(0) = u_{0|-1}^*$
- Output of the controller is the optimal input at the second time-step $u_{1|0}^*$

Time Delay – Input Delay

- Physical systems can have delays between when an input changes and when this change effects the states
- Often due to transport delay
- Let the input delay be $0 \le \tau$ seconds
- In discrete time, we can approximate this as a finite number of steps n_d

$$\tau \approx \Delta t \, n_d$$

- Input-delayed, discrete-time state-space model

$$x_{k+1} = Ax_k + Bu_{k-n_d}$$

$$y_k = Cx_k$$

- Can augment state-state space to account for this delay

$$x_{k+1} = Ax_k + Bx_k^1$$

$$x_{k+1}^1 = x_k^2$$

$$\vdots$$

$$x_{k+1}^{n_d-1} = x_{k+1}^{n_d}$$

$$x_{k+1}^{n_d} = u_k$$

Example: $n_d = 3$

$$\begin{bmatrix} x_{k+1} \\ x_{k+1}^1 \\ x_{k+1}^2 \\ x_{k+1}^3 \end{bmatrix} = \begin{bmatrix} A & B & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ x_k^1 \\ x_k^2 \\ x_k^3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ I \end{bmatrix} u_k$$

Time Delay – 1 step Input Delay

- Uncompensated

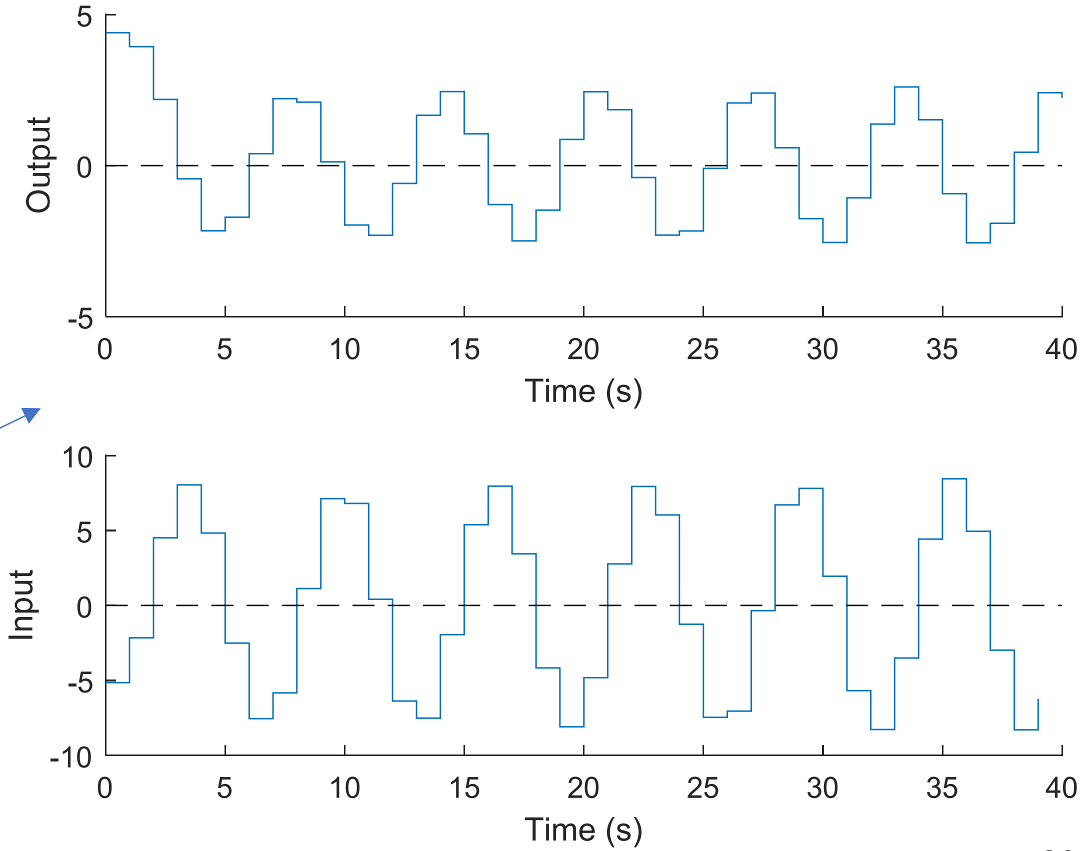
$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N$$

$$s.t.$$

$$x_{k+1} = Ax_k + Bu_k,\ k \in \{0,1,\ldots,N-1\}$$

$$x_0 = x(0)$$

```
% Simulate Closed-loop System
% Define simulation length
N_sim = 40;
% Initialize state and input trajectories
x0 = [10;10];
x_sim = [x0];
u_sim = [];
uOld = 0;
% Step through simulation
for i = 1:N_sim
    x0 = x_sim(:,i);
    inputs = {x0};
    u = controller{inputs};
    u_sim = [u_sim u];
    x_sim = [x_sim A*x0+B*uOld];
    uOld = u;
end
```



Marginally stable, any more time delay would result in a unstable CL system

Time Delay – 1 step Input Delay

- Compensated

$$J_0^*(x_0) = \min_{U_0} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N$$

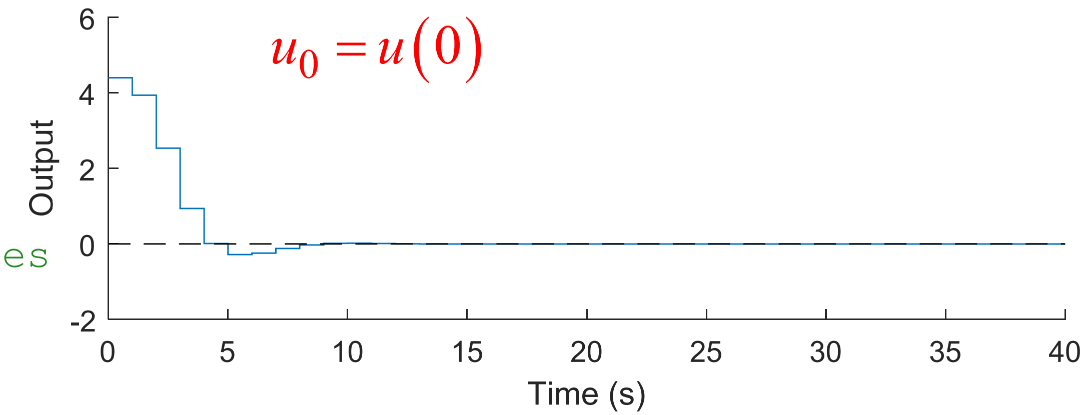
$$s.t.$$

$$x_{k+1} = Ax_k + Bu_k,\ k \in \{0,1,\ldots,N-1\}$$

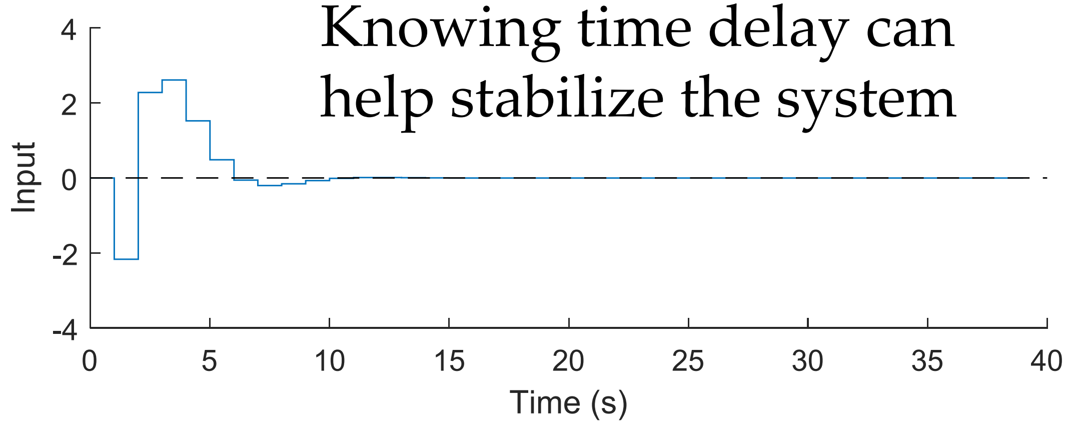
$$x_0 = x(0)$$

$$\color{red}{u_0 = u(0)}$$

```
% Specify Controller Inputs and Outputs
% Add first input as input to controller
inputs = {x_{1},u_{1}};
% Output the second input in the trajectory
outputs = {[u_{2}]};
```




```
% Initialize state and input trajectories
x0 = [10;10];
x_sim = [x0];
uOld = 0;
u_sim = [uOld];
% Step through simulation
for i = 1:N_sim
    x0 = x_sim(:,i);
    inputs = {x0,uOld};
    u = controller{inputs};
    u_sim = [u_sim u];
    x_sim = [x_sim A*x0+B*uOld];
    uOld = u;
end
```



Knowing time delay can help stabilize the system