Model Predictive Control

Leonard Mikša

December 6th, 2024

$$\begin{bmatrix} \frac{di_a(t)}{dt} \\ \frac{d\omega(t)}{dt} \end{bmatrix} = \begin{bmatrix} -\frac{R_a}{L_a} & -\frac{c}{L_a} \\ \frac{c}{J} & -\frac{b}{J} \end{bmatrix} \begin{bmatrix} i_a(t) \\ \omega(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{L_a} \\ 0 \end{bmatrix} u_a(t)$$

$$\mathbf{Q} = \begin{bmatrix} 10 & 0 \\ 0 & 10000 \end{bmatrix}$$

$$\mathbf{R} = 1$$

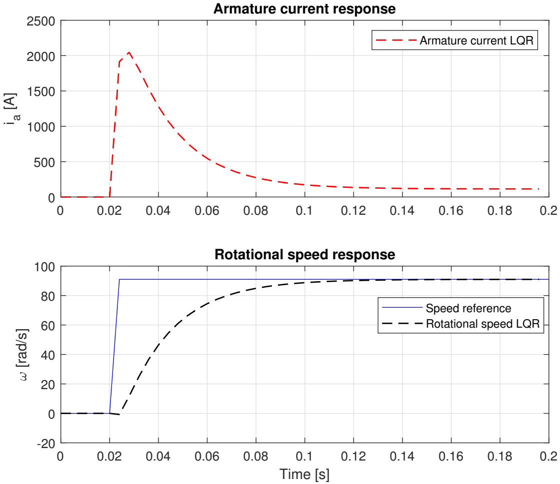
- LQR doesn't inherently eliminate steady-state error $\longrightarrow$ pregain needed

```
%Built-in LQR solver:
Ksystem = lqr(A, B, Q, R)
system = ss((A-B*Ksystem), B, C, D);
Kdc = dcgain(system)
Kr = 1/Kdc
system = ss((A-B*Ksystem), B*Kr, C, D);

[y, t, xlqr] = lsim(system, ref, t);     %Forced response
```

Linear-quadratic regulator
Results

- forced response at 0.02s from 0 to rated speed value



```matlab
steps = 50;                  %Number of steps
sim_time = 0.2;              %Simulation time
Ts = sim_time / steps;       %Sampling time

%Discretize the state-space model
[Ad, Bd] = c2d(A, B, Ts);
```

Model predictive control

MATLAB code - MPC parameters

```
%% MPC Parameters
prediction_horizon = 10;
control_horizon = 3;
Q = [10 0;
        0 10000];          %State weighting matrix
R = 1;                     %Penalizes control effort
p = prediction_horizon;
m = control_horizon;
```

Model predictive control

MATLAB code - Initial conditions and simulation setup

```
%% Initial Conditions
x0 = [0; 0];                       %Initial states
x = x0;                            %Current states
u_prev = 0;                        %Previous input

%% Simulation setup
x_trajectory = zeros(2, steps); %States for plotting
u_trajectory = zeros(1, steps); %Inputs for plotting
```

- $\mathbf{\Phi}$ and $\mathbf{\Gamma}$ $\longrightarrow$ system dynamics and control influence
- can be intuitively linked with natural and forced responses
- recursive expression of future states:

$$x_{k+1} = Ax_k + Bu_k$$

$$x_{k+2} = A^2x_k + ABu_k + Bu_{k+1}$$

$$x_{k+3} = A^3x_k + A^2Bu_k + ABu_{k+1} + Bu_{k+2}$$

$$\mathbf{\Phi} = \begin{bmatrix} A \\ A^2 \\ A^3 \\ \vdots \\ A^p \end{bmatrix}, \qquad \mathbf{\Gamma} = \begin{bmatrix} B & 0 & 0 & \cdots & 0 \\ AB & B & 0 & \cdots & 0 \\ A^2B & AB & B & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A^{p-1}B & A^{p-2}B & A^{p-3}B & \cdots & B \end{bmatrix}$$

- $\mathbf{\Phi} \longrightarrow (p \cdot n_x) \times n_x \qquad \mathbf{\Gamma} \longrightarrow (p \cdot n_x) \times (m \cdot n_u)$

Model predictive control

MATLAB code - Prediction matrices

```
%% Optimization setup
%Quadratic Cost Matrices
Q_bar = kron(eye(p), Q);     %Block diagonal Q
R_bar = kron(eye(m), R);     %Block diagonal R
H = blkdiag(Q_bar, R_bar);   %Quadratic cost matrix

%Prediction matrices
Phi = zeros(2*p, 2);         %State prediction matrix
Gamma = zeros(2*p, m);       %Control influence matrix

%Filling the prediction matrices
for i = 1:p
    Phi(2*i - 1:2*i, :) = Ad^i;
    for j = 1:min(i, m)
        Gamma(2*i - 1:2*i, j) = Ad^(i - j) * Bd;
    end
end
```

Model Predictive Control

MATLAB code - Input constraints

- fixed constraints
- states not included yet

```
%Constraints
U_max = 10000;             %Maximum input
U_min = -10000;            %Minimum input
delta_U_max = 200;         %Maximum change in input

b_u = [U_max*ones(p, 1); delta_U_max*ones(m, 1)];
b_l = [U_min*ones(p, 1); -delta_U_max*ones(m, 1)];

predicted_states = zeros(2, p, steps);%3D array
control_inputs = zeros(m, steps);     %Storing u
u_opt = zeros(m, 1);                  %Storing u_opt
```

- initially *turned off* due to algorithm validation
- potential problem $\longrightarrow$ ΔU_{max} doesn't work

- **H** and *f* 'transformed' to affect states via $\mathbf{Q_{bar}}$

```
%% MPC Loop
for k = 1:steps

  x_pred = Phi * x;        %Predicted states (20x1)
  predicted_states(:, :, k) = reshape(x_pred, [2, p]);%
      Store predicted states (2x10x50)

  %Define quadratic cost matrix H
  H = Gamma' * Q_bar * Gamma + R_bar;       %Size = [m, m]

  %Define linear cost vector f
  f = (Gamma' * Q_bar * (Phi * x - ref))';  %Size = [m, 1]

  u_opt = quadprog(H, f,[],[],[],[], b_l, b_u,[],[]);
```

Model Predictive Control

MATLAB code - MPC loop

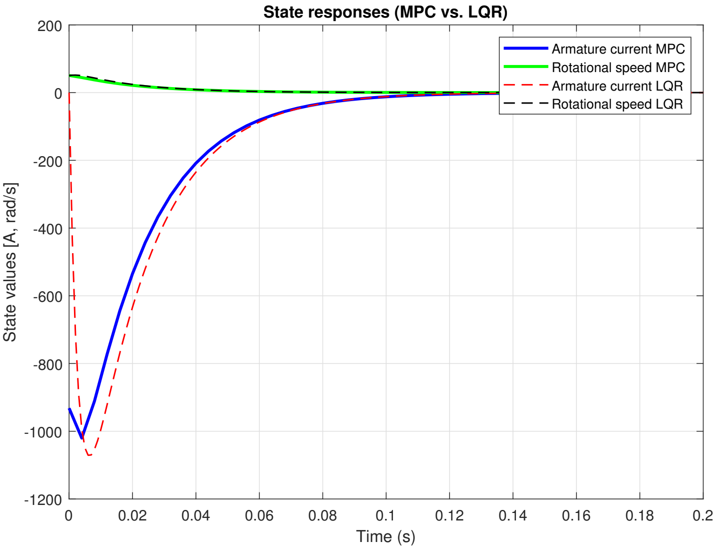
```
%% MPC Loop - continued
 control_inputs(:, k) = u_opt(1:m);  %Store control
     inputs over m
 u = u_opt(1); %Extract the first control input
 x = Ad * x + Bd * u;

 %Save trajectories
 x_trajectory(:, k) = x;
 u_trajectory(k) = u;

 u_prev = u; %Update previous input
end
```

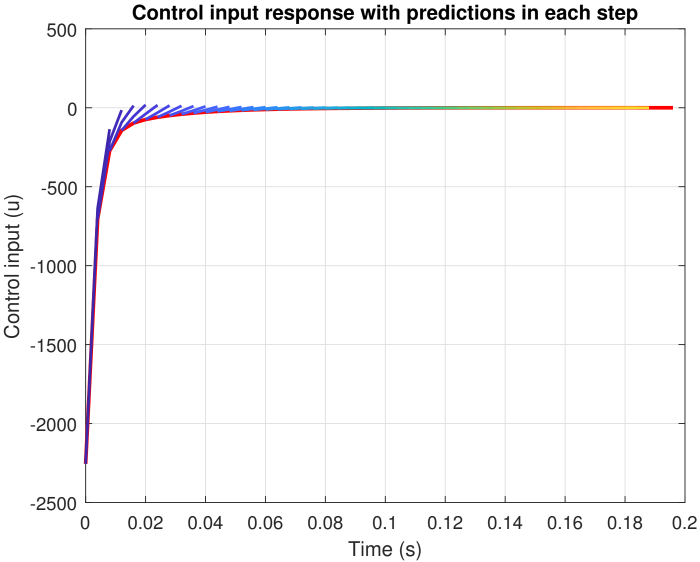
Model Predictive Control

Input constraints results - constraints off

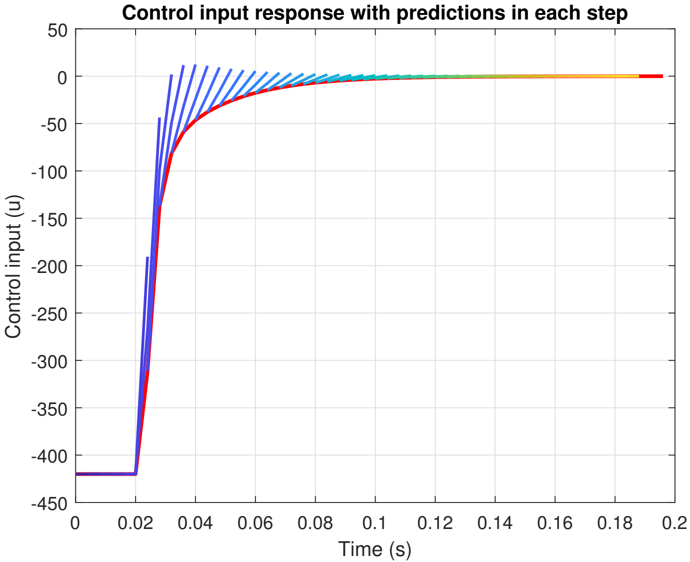


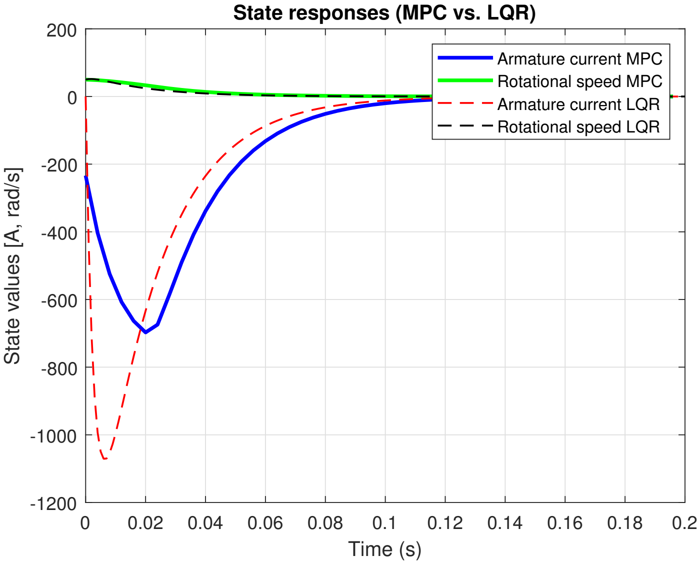
Model Predictive Control

Input constraints results - constraints off



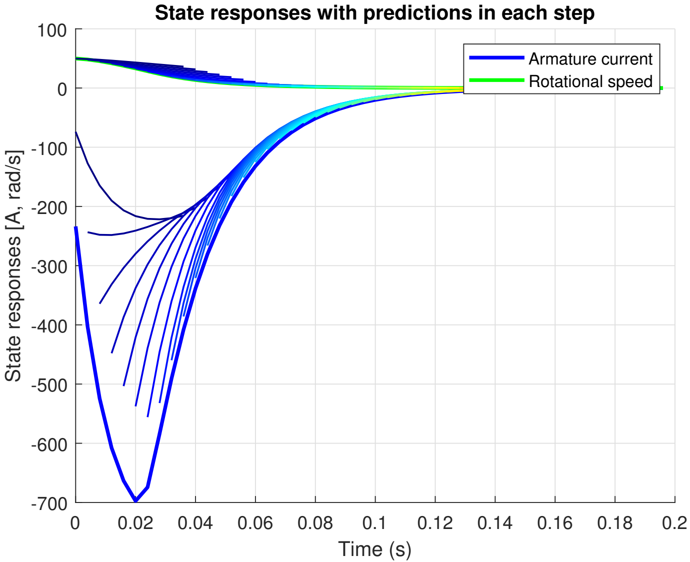
Input constraints results - constraints on




State responses (MPC vs. LQR)
Armature current MPC
Rotational speed MPC
Armature current LQR
Rotational speed LQR
State values [A, rad/s]
Time (s)
200
0
-200
-400
-600
-800
-1000
-1200
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2

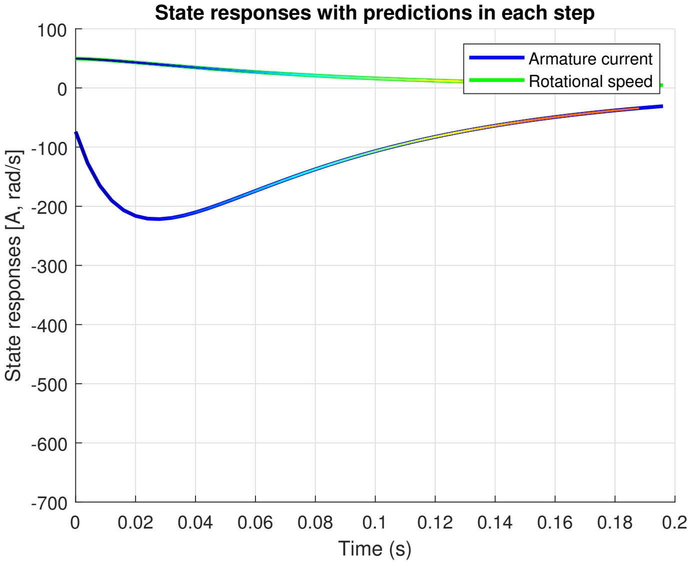
Model Predictive Control

Input constraints results - constraints on



Model Predictive Control

Input constraints results - constraints ± 0



Model Predictive Control

State constraints

- linear constraints
- both inputs and states

```
%Constraints
U_max = 10000;             %Maximum input [V]
U_min = -10000;            %Minimum input [V]
delta_U_max = 2;           %Maximum change in input [V]
x_max = [10000; 500];      %Maximum state [A], [rad/s]
x_min = [-10000; -500];    %Minimum state [A], [rad/s]
```

- initially *turned off* due to algorithm validation
- forced response at 0.02s from 0 to rated speed value 91 $\frac{\mathrm{rad}}{\mathrm{s}}$

- **Γ** included in x_pred
- step input as a speed reference at 0.025s

```
%% MPC Loop
for k = 1:steps
 current_time = k * Ts;

 if current_time >= 0.025
  ref = 91 * ones(2 * p, 1);
 else
  ref = zeros(2 * p, 1);
 end

 x_pred = Phi * x + Gamma * u_opt;
 predicted_states(:, :, k) = reshape(x_pred, [2, p]);
```

Model Predictive Control

MATLAB code - MPC loop modifications

- bounds changing dynamically (within the MPC loop)

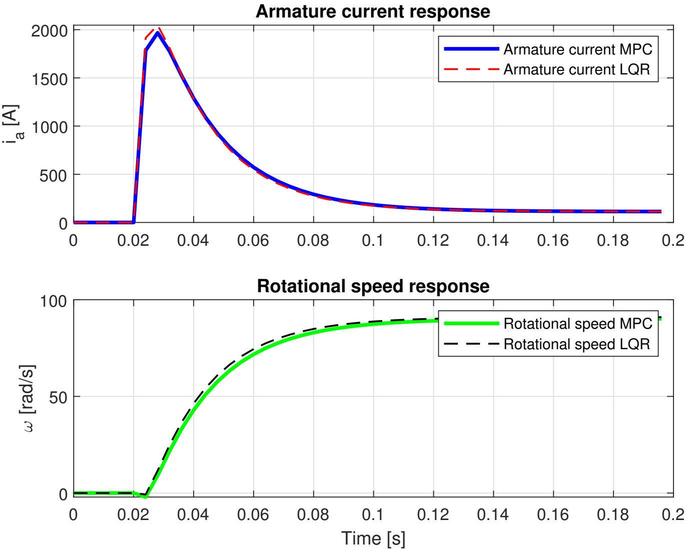
```
%Adjust bounds for states (xmax - xpred --> predicting possible constraint violation)
b_x_upper = repmat(x_max, p, 1) - Phi * x;
b_x_lower = repmat(x_min, p, 1) - Phi * x;

%Combine constraints
F_total = [Gamma; -Gamma; eye(m); -eye(m)]; %Input constraints
F_x = eye(2 * p); % State constraints (multiplied by Gamma in the next step)

F_total = [F_x * Gamma; -F_x * Gamma; eye(m); -eye(m)];
b_total = [b_x_upper; -b_x_lower; U_max * ones(m, 1); -U_min * ones(m, 1)];

u_opt = quadprog(H, f, F_total, b_total, [], [], [], [], []);
```

State constraints results - constraints off



Run MATLAB simulation for more results

References

MATLAB Documentation
https://www.mathworks.com/help.

L. Wang
Model Predictive Control System Design and Implementation Using MATLAB®
Springer, 2009.

Internet
Video lectures, AI, forums