Contents

Energy-Aware MPC Demo

This demo illustrates the usage of the EAMPC class on a 2-states, 1-input linear system subject to state and output noise. Several sensor nodes are used for disturbance rejection purposes. We assume that every node measures the full state vector, corrupted by an additive disturbance. At every time step, once every node got a measurement, an estimated state value is obtained by taking the mean value of the outputs. This estimation is transmitted to the controller in accordance with a threshold-based policy, where the estimated state is compared with a predicted value which has been precomputed by the controller and transmitted beforehand to the sensors. This policy is inteded to reduce the number of communications between controller and sensors, hence saving sensor nodes battery. In this example the energy-aware MPC is compared with a traditional MPC scheme, where the measurements are simply transmitted to the controller at every time step, and no predictions are computed and transmitted by the controller.

by D. Bernardini, 2010.

```
clear all
close all
```

Plant model

State-space matrices of the discrete-time LTI system

$$x(k+1) = Ax(k) + Bu(k)$$

```
Plant.A = [.21 -.39; -.39 .82];
Plant.B = [0 1]';
```

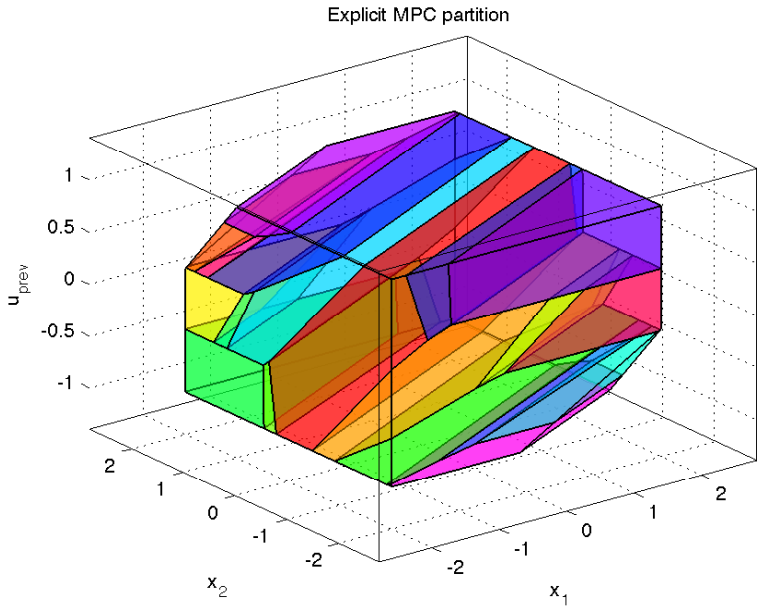
Network model

specify parameters for sensor nodes

```
Net.nodes = 3; % number of wireless sensor nodes
Net.th = .05*ones(2,1); % transmission thresholds
Net.Ne = 10; % estimation horizon
```

Controller design

```
% element-wise constraints on output, input, and input rate
Limits.ymin = [-2 -2]'; % min output
Limits.ymax = [2 2]'; % max output
Limits.umin = -1;  % min input
Limits.umax = 1; % max input
Limits.dumin = -.6; % min input rate
Limits.dumax = .6; % max input rate

% weights matrices for MPC objective function
Weights.Qu = .1; % input weight
Weights.Qy = eye(size(Plant.A,1)); % output weight
Weights.Qn = Weights.Qy; % terminal weight
Weights.rho = Inf; % positive weight for soft output constraints
% (if rho=Inf then hard constraints are imposed)

% other parameters
Params.pnorm = 2; % norm used in the objective function (1, 2 or Inf).
Params.N = 10; % prediction horizon
Params.Nc = Params.N; % control horizon

% build EAMPC object
EAobj = eampc(Plant,Net,Limits,Weights,Params);

% plot explicit MPC partition
Options = mpt_options;
Options.shade = .5;
Options.samecolors = 1;
figure
plot(EAobj.Ctrl.Pn,Options);
xlabel 'x_1'
ylabel 'x_2'
zlabel 'u_{prev}'
axis(1.4*[Limits.ymin(1) Limits.ymax(1) ...
    Limits.ymin(2) Limits.ymax(2) Limits.umin Limits.umax]);
title('Explicit MPC partition')
box on
```



Simulation: energy-aware MPC vs standard MPC

The energy-aware MPC controller is compared with a standard MPC control scheme, where at every time step measurements are transmitted to the controller regardless to the threshold logic, and no predictions are transmitted to the sensors.

The two controllers are simulated in closed-loop with the LTI system

$$x(k+1) = Ax(k) + Bu(k) + w(k)$$

where $|w(k)| \leq w_{max},\ \forall k$. Moreover, measurements are affected by an additive error, i.e.,

$$|y^i(k) = x(k) + v^i(k)|$$

for all $i = 1, 2, ...,$`Net.nodes`, where $|y^i|$ is the output measured by the `i`-th node, and $|v^i(k)| \leq v_{max}$, for all k, i.

```
% build EAMPC object for standard MPC
MPCobj = EAobj; % copy EAMPC object
MPCobj.Net.th = zeros(MPCobj.ny,1); % set thresholds to zero

% simulation length
Tsteps = 20;

% disturbance realizations
loadNoise = 1; % 0: pick random noise, 1: load noise data form Noise.mat
if loadNoise==0

    % additive disturbance bounds
    wmax = [.03 .03]';

    % output noise bounds for each output and each node
    vmax = kron([.06 .06]',ones(1,Net.nodes));

    % additive disturbance realizations
    W = diag(2*wmax)*rand(EAobj.ny,Tsteps)-wmax*ones(1,Tsteps);

    % output noise realizations
    V = zeros(EAobj.ny,EAobj.Net.nodes,Tsteps);
    for j=1:EAobj.Net.nodes
        V(:,j,:) =  diag(2*vmax(:,j))*rand(EAobj.ny,Tsteps) ...
            -kron(ones(1,Tsteps),vmax(:,j));
    end

    % initial state (randomly picked in the feasible state set)
    vert = extreme(projection(EAobj.Ctrl.Pfinal,1:EAobj.ny));
    comb = rand(size(vert,1),1);
    comb = comb/sum(comb);
    x0 = (comb'*vert)';
    save Noise wmax vmax W V x0
else
    load Noise.mat
end

% init energy-aware MPC simluation variables
X = x0; % state
Xestim = []; % estimated state
U = []; % input
Uprev = zeros(EAobj.nu,1); % previous input
J = []; % experimental cost function
EAobj = EAobj.init_sim(); % init simulation data

% init standard MPC simluation variables
```

```
XX = x0; % state
XXestim = []; % estimated state
UU = []; % input
UUprev = zeros(MPCobj.nu,1); % previous input
JJ = []; % experimental cost function
MPCobj = MPCobj.init_sim(); % init simulation data

% compute first sequence of output predictions and transmit them to sensors
% (we assume that the initial state is known)
EAobj = EAobj.send_predictions(X,Uprev);
MPCobj = MPCobj.send_predictions(XX,UUprev);
% note: since MPCobj.Net.th = 0, predictions are not actually transmitted here

% run simulation for k=1,2,...,Tsteps
for k=1:Tsteps

    % measurements acquisition
    [xestim,EAobj] = EAobj.get_measurements(X(:,end),V(:,:,k));
    Xestim(:,end+1) = xestim;
    [xestim,MPCobj] = MPCobj.get_measurements(XX(:,end),V(:,:,k));
    XXestim(:,end+1) = xestim;

    % compute the control law
    U(:,end+1) = EAobj.get_input(Xestim(:,end),Uprev);
    UU(:,end+1) = MPCobj.get_input(XXestim(:,end),UUprev);

    % if needed, compute new predictions
    EAobj = EAobj.send_predictions(Xestim(:,end),Uprev);
    MPCobj = MPCobj.send_predictions(XXestim(:,end),UUprev);

    % evaluate evperimental costs
    if EAobj.Params.pnorm == 2
        J(k) = X(:,end)'*Weights.Qy*X(:,end) + ...
            U(:,end)'*Weights.Qu*U(:,end);
    else
        J(k) = norm(Weights.Qy*X(:,end),EAobj.Params.pnorm) + ...
            norm(Weights.Qu*U(:,end),EAobj.Params.pnorm);
    end
    if MPCobj.Params.pnorm == 2
        JJ(k) = XX(:,end)'*Weights.Qy*XX(:,end) + ...
            UU(:,end)'*Weights.Qu*UU(:,end);
    else
        JJ(k) = norm(Weights.Qy*XX(:,end),MPCobj.Params.pnorm) + ...
            norm(Weights.Qu*UU(:,end),MPCobj.Params.pnorm);
    end
```

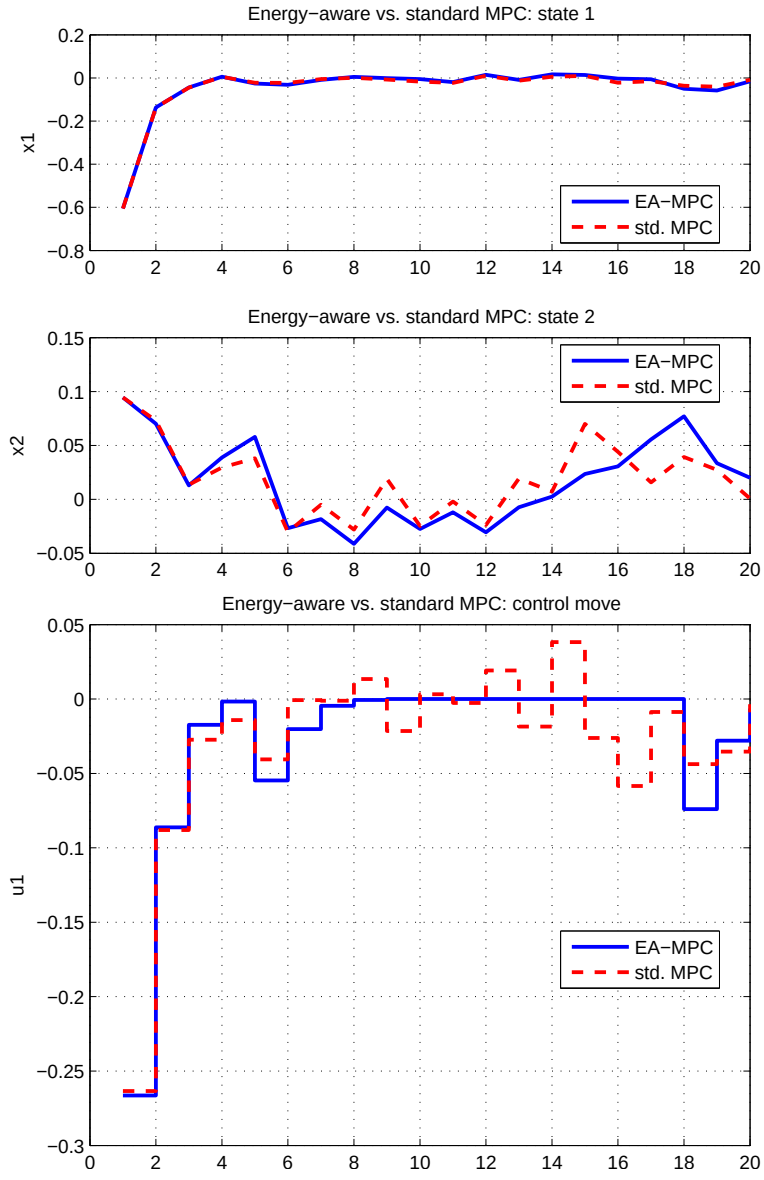
```
    % state evolution
X(:,end+1) = Plant.A*X(:,end) + Plant.B*U(:,end) +  W(:,k);
    XX(:,end+1) = Plant.A*XX(:,end) + Plant.B*UU(:,end) +  W(:,k);

    % update previous input
    Uprev = U(:,end);
    UUprev = UU(:,end);

end
```

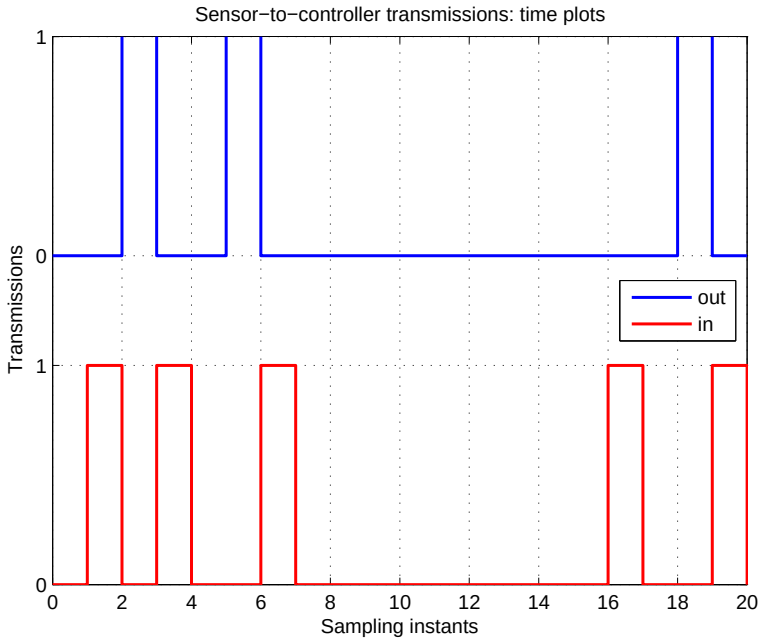
Results

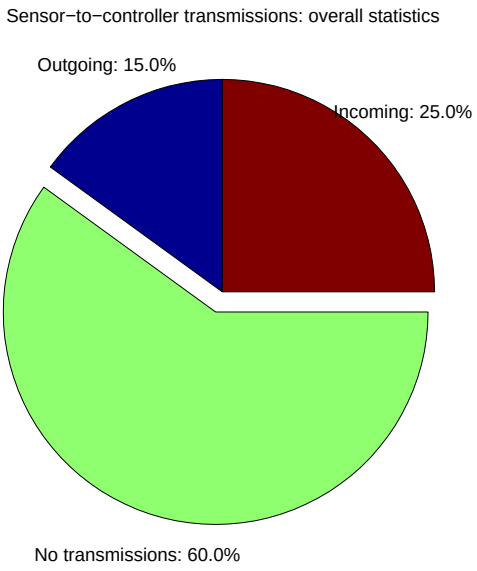
```
% plot state trajectory
figure;
subplot(2,1,1);
plot(1:Tsteps,X(1,1:Tsteps),'b',1:Tsteps,XX(1,1:Tsteps),'r--','LineWidth',2);
title('Energy-aware vs. standard MPC: state 1')
ylabel 'x1'
legend('EA-MPC','std. MPC','Location','Best')
grid on
subplot(2,1,2);
plot(1:Tsteps,X(2,1:Tsteps),'b',1:Tsteps,XX(2,1:Tsteps),'r--','LineWidth',2);
title('Energy-aware vs. standard MPC: state 2')
ylabel 'x2'
legend('EA-MPC','std. MPC','Location','Best')
grid on

% plot input trajectory
figure
stairs(1:Tsteps,U(1,1:Tsteps),'b','LineWidth',2)
hold on
stairs(1:Tsteps,UU(1,1:Tsteps),'r--','LineWidth',2)
title('Energy-aware vs. standard MPC: control move')
ylabel 'u1'
legend('EA-MPC','std. MPC','Location','Best')
grid on

% plot WSN activity
figure
stairs(0:Tsteps,[0 EAobj.Sim.tx(1:Tsteps)]+1.5,'b','LineWidth',1.5);
hold on
stairs(0:Tsteps,[0 EAobj.Sim.rx(1:Tsteps)],'r','LineWidth',1.5);
set(gca,'Ytick',[0 1 1.5 2.5],'YTickLabel',{'0' '1' '0' '1'});
title('Sensor-to-controller transmissions: time plots')
xlabel 'Sampling instants'
ylabel 'Transmissions'
```

```
legend('out','in','Location','Best')
grid on

% plot pie graph for transmission data
figure
tx_perc = 100*sum(EAobj.Sim.tx)/Tsteps;
rx_perc = 100*sum(EAobj.Sim.rx)/Tsteps;
free_perc = 100-tx_perc-rx_perc;
pie([tx_perc free_perc rx_perc],[0 1 0], ...
    {['Outgoing: ' num2str(tx_perc,'%3.1f') '%'], ...
    ['No transmissions: ' num2str(free_perc,'%3.1f') '%'], ...
    ['Incoming: ' num2str(rx_perc,'%3.1f') '%']});
title('Sensor-to-controller transmissions: overall statistics')

% print experimental costs
sumJ = sum(J);
sumJJ = sum(JJ);
fprintf('\n== Results for Energy-Aware MPC:\n');
fprintf('threshold TH = [%.2f %.2f]'', experimental cost J = %.3f.\n', ...
    EAobj.Net.th',sumJ);
fprintf('\n== Results for standard MPC:\n');
fprintf('threshold TH = [%.2f %.2f]'', experimental cost J = %.3f.\n', ...
    MPCobj.Net.th',sumJJ);
fprintf('\n== Energy-Aware vs standard MPC: transmission savings = %.2f%%\n', ...
    free_perc);
fprintf('\n== Energy-Aware vs standard MPC: performance ratio = %.2f%%\n', ...
    (sumJ/sumJJ-1)*100);
```

```
== Results for Energy-Aware MPC:
threshold TH = [0.05 0.05]', experimental cost J = 0.441.

== Results for standard MPC:
threshold TH = [0.00 0.00]', experimental cost J = 0.432.

== Energy-Aware vs standard MPC: transmission savings = 60.00%

== Energy-Aware vs standard MPC: performance ratio = 1.98%
```


Energy−aware vs. standard MPC: state 1
x1
EA−MPC
std. MPC
Energy−aware vs. standard MPC: state 2
x2
EA−MPC
std. MPC
Energy−aware vs. standard MPC: control move
u1
EA−MPC
std. MPC





Conclusions

In the proposed example, the energy-aware MPC scheme provided a reduction of 60% in the number of transmissions between the controller and the sensor nodes. This energy savings have been obtained with a small loss in the performance index (around 2%), with respect to a traditional MPC scheme.

Indeed, the threshold values `Net.th` need to be properly tuned, especially in function of the magnitude of the disturbances, in order to obtain a valuable reduction in the transmission rate and also achieve a satisfying performance.