Table of Contents


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%             Model Predictive Control - Exercise 5
%               EPFL - Spring semester 2017 -
%
%             Huber Lukas - Zgraggen Jannik
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear variables;
addpath(genpath('../tbxmanager'))
```

System initialization

```
A = [0.7115, -0.4345; 0.4345, 0.8853];
B = [0.2173; 0.0573];

C = [0, 1];

% Augmented system
B_d = zeros(2,1);
C_d = [1];

A_augm = [A,B_d;zeros(1,2),1];
B_augm = [B;0];
C_augm = [C, 1];


% Make sure that eigenvalues of (A+LC) are in unit circle
L = (place(A_augm',-C_augm',[0.5,0.6,0.7]))';

% Initial Estimation
x0_est = [3;0];
d0_est = [0];

%Initial Conditions - Real system
x0_r = [1;2];
d_r = 0.2;
```

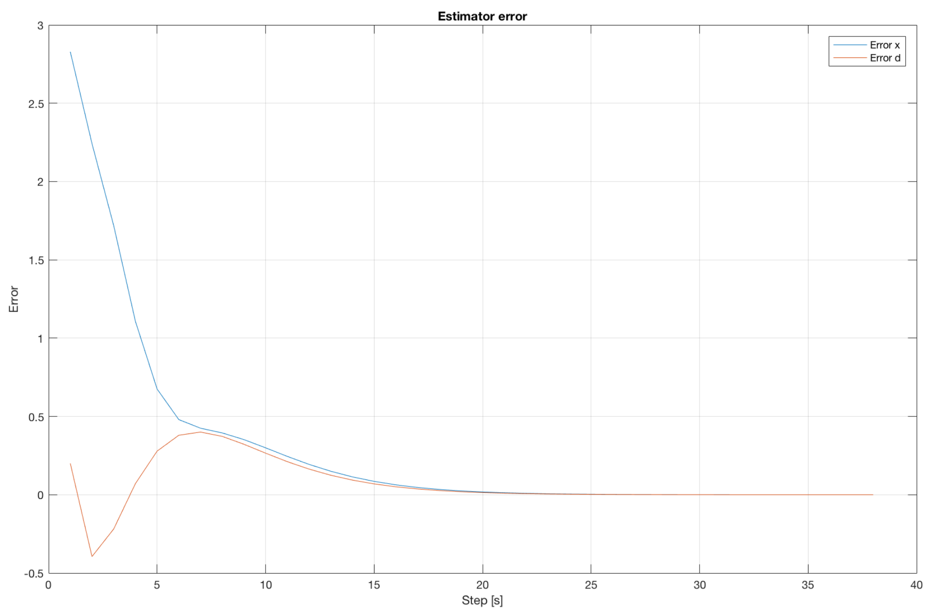
Exercise 1 - Observer Design

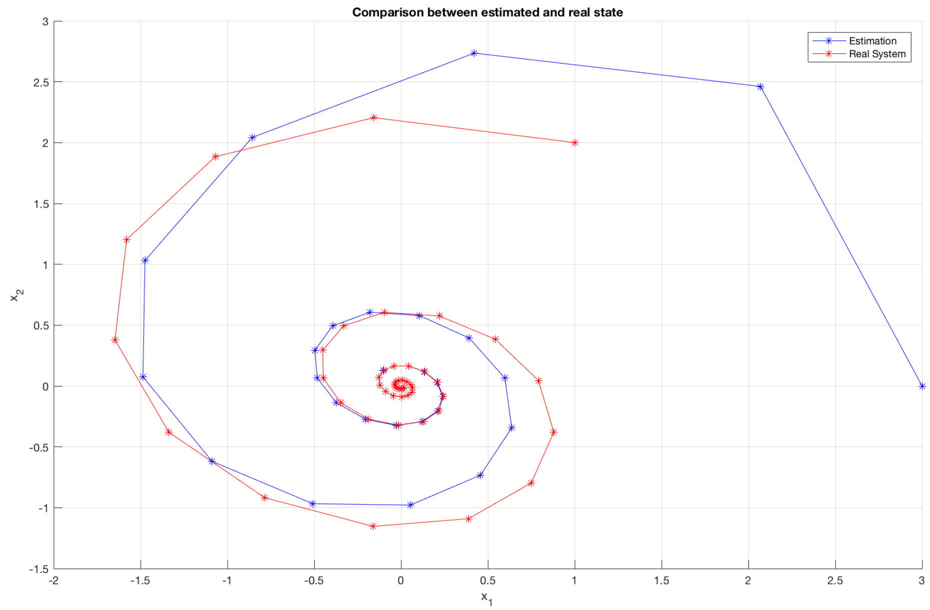
```
% Error between real system and estimation
```

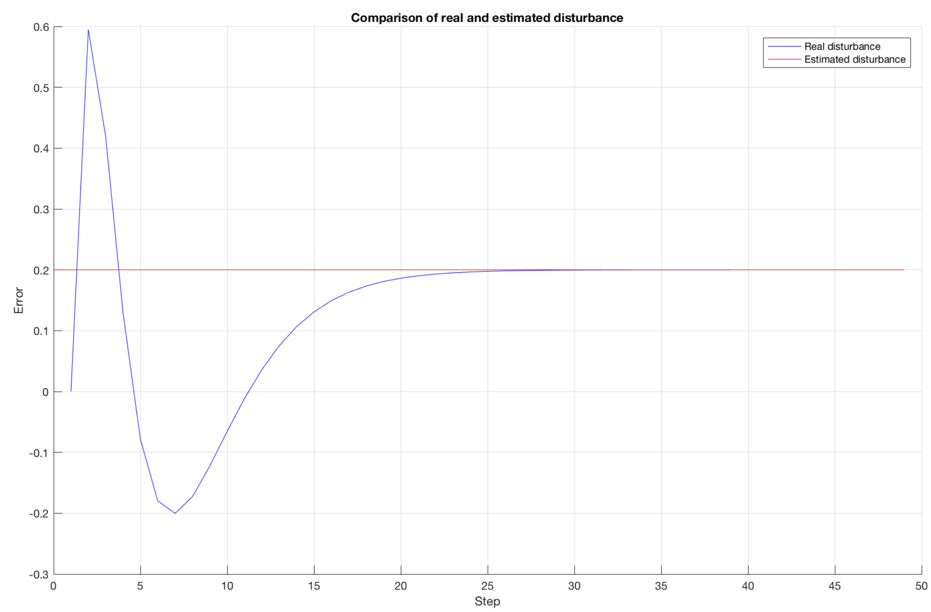
```
deltaX = [x0_r-x0_est];
deltaD = [d_r-d0_est];

obsError = [deltaX; deltaD];


% Rund the integral disturbance dynamics

MAXITER = 50; minTol = 1e-2;

for i = 2:MAXITER
    obsError(:,i) = (A_augm + L *C_augm)*obsError(:,i-1);
    if(norm(obsError(i)) < minTol)
        fprintf('Problem converged after iteration %d \n',i);
        break;
    end
end

% Plot the results

figure('Position',[0 0 1000 600]); grid on;
plot(sqrt(sum(obsError(1:2,:).^2,1))); hold on;
plot(obsError(3,:)); grid on;
legend('Error x', 'Error d')
xlabel('Step [s]'); ylabel('Error')
title('Estimator error');


% Estimation converges very nicely towards the real value. The error
 in x
% and d converges below the minimum Tolerance in less than 100
 iterations.

% Initialize vectores
xVal_est = [x0_est]; xVal_r = [x0_r];
dVal_est = [d0_est]; %dVal_r = [d_r];


% Define loop
MAXITER= MAXITER; minTol = 1e-2;

for i = 2:MAXITER
    u = 0; % No control or input
    y = C*xVal_r(:,i-1)+d_r;

    xVal_r(:,i) = A*xVal_r(:,i-1) + B*u;

    dh_hat2 = A_augm*[xVal_est(:,i-1); dVal_est(i-1)]+B_augm*u ...
            +  L*(C*xVal_est(:,i-1)+C_d*dVal_est(i-1)-y);
    xVal_est(:,i) = dh_hat2(1:2);
    dVal_est(i) = dh_hat2(3);

    if(and(norm(xVal_est(i)-xVal_r(i)) < minTol, ...
            norm(xVal_r(:,i)-xVal_r(:,i-1)) < minTol))
```

```
        fprintf('Problem converged after iteration %d \n',i);
        break;
    end
end


figure('Position',[0 0 1000 600]); hold on; grid on;
plot(xVal_est(1,:), xVal_est(2,:),'b-*')
plot(xVal_r(1,:),xVal_r(2,:), 'r-*')
xlabel('x_1'); ylabel('x_2')
legend('Estimation','Real System')
title('Comparison between estimated and real state');

figure('Position',[0 0 1000 600]); hold on; grid on;
plot(dVal_est,'b'); plot([0, length(dVal_est)], [d_r, d_r], 'r')
xlabel('Step'); ylabel('Error')
legend('Real disturbance', 'Estimated disturbance')
title('Comparison of real and estimated disturbance');
```

Problem converged after iteration 38
Problem converged after iteration 49







Exercise 2 & 3 - Controller Design

```
N =5; % Horizon length

% Define optimization variables
x = sdpvar(2,N,'full');
u = sdpvar(1,N,'full');

% Constraints
```

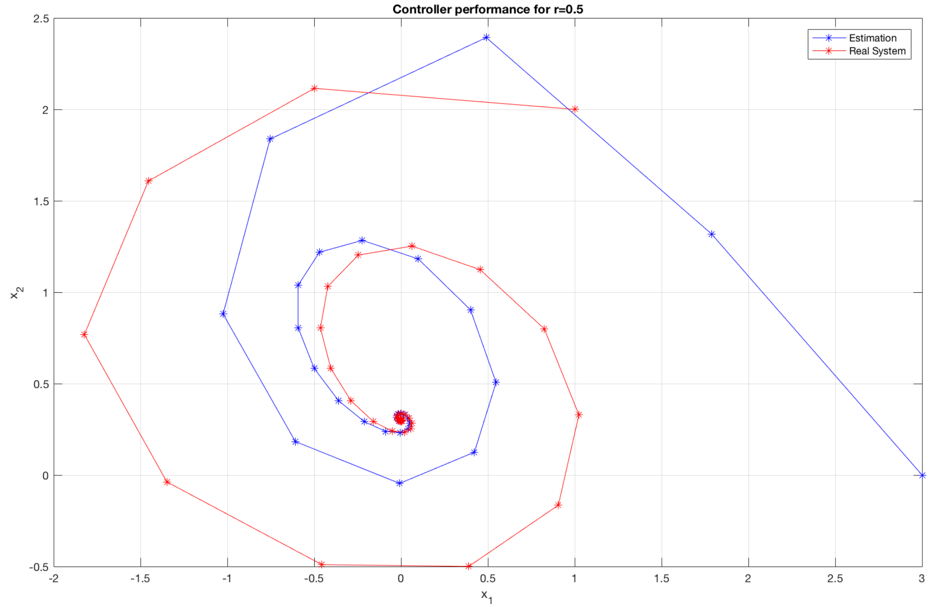
```
h = [3; 3];              %Input constraint
H = [1; -1];             %Input constraint

% Stage cost
%Weights Controller that is able to track constant output reference
Q = eye(size(A,1));
R = 1;
I=eye(2);


% Weight of final cost
P = dlyap(A,Q);

% Solver settings
opt = sdpsettings;
opt.solver = 'quadprog';
opt.quadprog.TolCon = 1e-16;


% Initial conditions
xi = x0_est; % try to controll estimation
xd_est = [x0_est; d0_est]
y_est = [C*xd_est(1:2,1)+d0_est];
r_val = [0.5, 1];

for r = r_val
    % Real conditions
    y_r = [0];
    u_r = [0];

    x_r = [x0_r];
    y_r = [C*xi+d_r];

    t = [0];

    tolX = 1e-8;
    % Can now compute the optimal control input using
    for i = 2:MAXITER

        % Exercise 2 - Optimize u^2 (Target tracking)
        x_s = sdpvar(2,1,'full');
        u_s = sdpvar(1,1,'full');
        obj_ss = u_s*R*u_s;
        con_ss =[I-A,-B;C,0]*[x_s;u_s] == [0;0;r-C_d*xd_est(3,i-1)]; %
 System dynamics
        con_ss = [con_ss, H*u_s <= h];                        % Input
 constraint
        solvesdp(con_ss, obj_ss, opt);
        x_s=double(x_s);
        u_s=double(u_s);

        % Define constraints and objective for MPC-controller
        con = [];
        obj = 0;
```

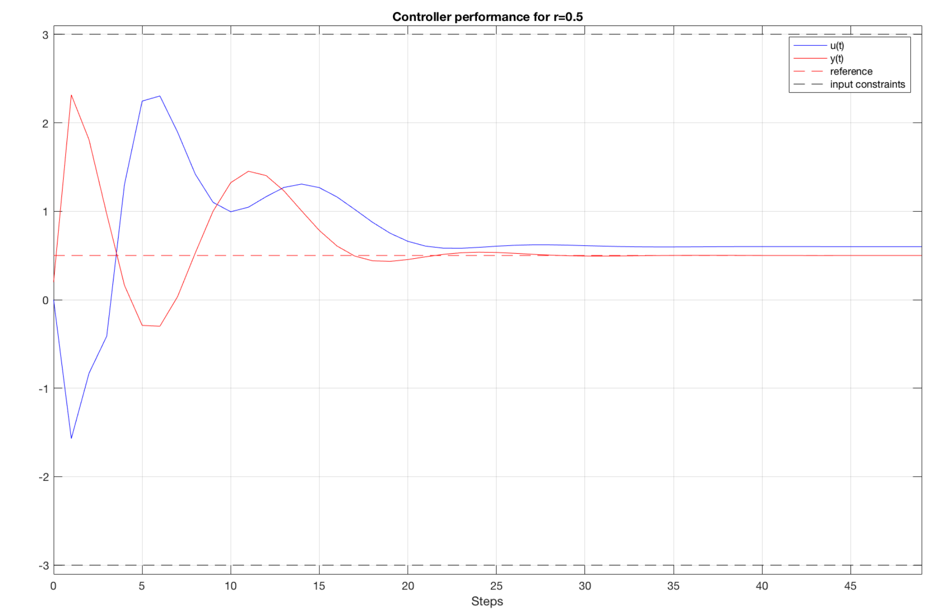
```
        %con = [con, x(:,1) == x0];
        for j = 1:N-1
            obj = obj + (x(:,j)-x_s)'*Q*(x(:,j)-x_s) + (u(:,j)-
u_s)'*R*(u(:,j)-u_s); % Cost function
            con = [con, x(:,j+1) == A*x(:,j) + B*u(:,j)]; % System
 dynamics
            con = [con, H*u(:,j) <= h];                   % Input
 constraints
        end
        obj = obj + (x(:,N)-x_s)'*P*(x(:,N)-x_s);      % Terminal
 weight
        ctrl = optimizer(con, obj, opt, x(:,1), u(:,1));
        [u_opt,infeasible] = ctrl{xi};

        if(infeasible); fprintf('Problem infeasible at i=%d
 \n',i); break; end;

        u_r(i) = u_opt;
        t(i) = i;

        % Real sytem
        x_r(:,i) = A*x_r(:,i-1) + B*u_opt;
        y_r(i) = C*x_r(:,i)+d_r;

        % Estimated system
        xd_est(:,i) = [A_augm*xd_est(:,i-1)+B_augm*u_opt ...
                      + L*(C*xd_est(1:2,i-1) + C_d*xd_est(3,i-1) -
 y_r(i-1))];

        xi = xd_est(1:2,i);

        if(norm(abs(y_r(i)-r)) < tolX);
            fprintf('System converged at after %d steps. \n',i);
            break
        end

    end

    % Plot results
    figure('Position',[0 0 1000 600]);
    plot(xd_est(1,:),xd_est(2,:),'b-*');
    grid on; hold on;
    plot(x_r(1,:),x_r(2,:),'r-*');
    legend('Estimation','Real System')
    xlabel('x_1'), ylabel('x_2')
    title(['Controller performance for r=',num2str(r)]);


    figure('Position',[0 0 1000 600]); grid on;
    plot(0:length(u_r)-1, u_r,'b'); hold on; grid on;
    plot(0:length(u_r)-1,y_r,'r');
    plot([0,length(u_r)-1],[r,r],'r--')
    plot([0,length(u_r)-1],[-3,-3],'k--')
    plot([0,length(u_r)-1],[3,3],'k--')
```

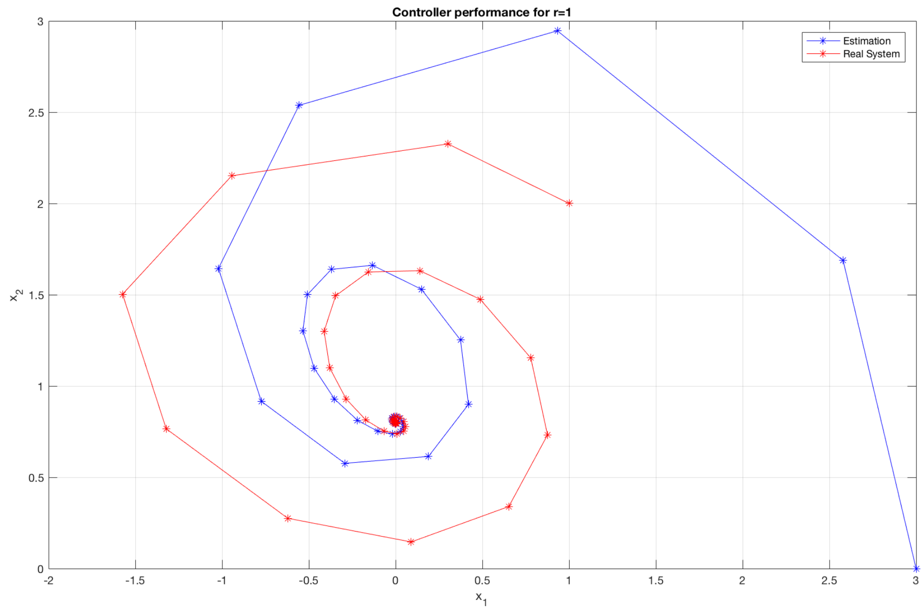
```
    xlim([0,length(u_r)-1])
    ylim([-3.1,3.1])
    title(['Controller performance for r=',num2str(r)]);

    legend('u(t)','y(t)','reference','input constraints')
    xlabel('Steps')


end
```

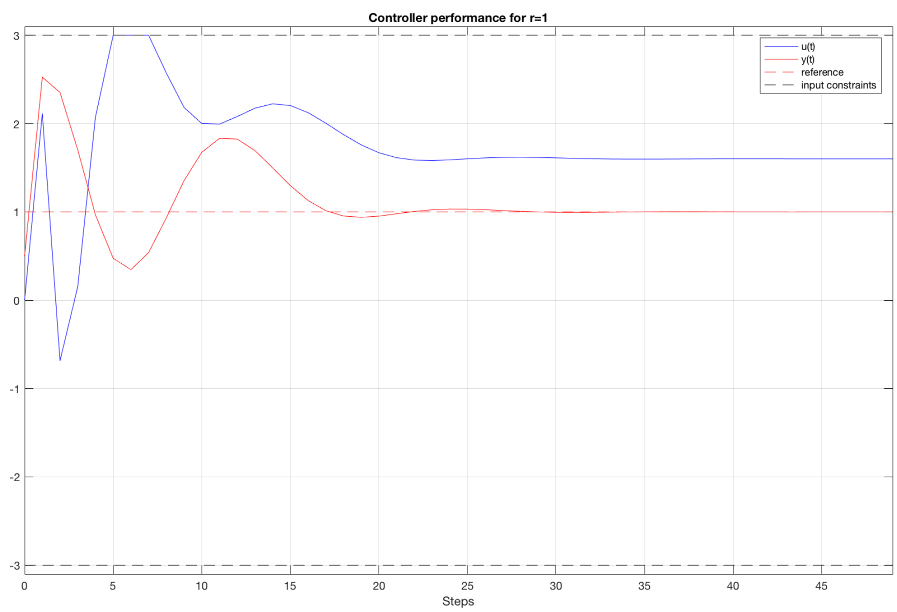
```
xd_est =

     3
     0
     0


Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-
decreasing in
feasible directions, to within the default value of the optimality
 tolerance,
and constraints are satisfied to within the selected value of the
 constraint tolerance.




Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-
decreasing in
feasible directions, to within the default value of the optimality
 tolerance,
and constraints are satisfied to within the selected value of the
 constraint tolerance.




Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-
decreasing in
feasible directions, to within the default value of the optimality
 tolerance,
and constraints are satisfied to within the selected value of the
 constraint tolerance.
```

```
feasible directions, to within the default value of the optimality
 tolerance,
and constraints are satisfied to within the selected value of the
 constraint tolerance.
```










```
fprintf('Programm terminated. \n')
```

Programm terminated.

Published with MATLAB® R2016b