Exercise Journal: Isak Skeie

Looking at the Simulink code, the first function block is formating the described problem so that QPSOlve is able to fix it. The formulation block is connected to the QPSolver block, with its outputs leading to a second custom defined funtion, that extracts the current values (inputs, outputs, states) for each iteration. The states, inputs and outputs are then plotted with an array function.

The simulation is run with a step change on the input force on the cart of the pendulum, to demonstrate the control of the system.

QP solver formulation

Based on the example in the lecture notes. The equations for the inverted pendulum are given in continous form. These are transformed to discrete form, with a timestep of 0.1. seconds.

The number of uknowns is then found. The numbe of uknowns is based on the states in the system, system input, system output, and error from reference point. These are then multiplied by horizon, giving us in this case 1080 unkown variables.

After formatting the H matrix. The constraints are formulated based on the state-, measurement- and error equation. Bounds are for this problem not specified.

Result extraction

The QP solver outputs a single array for alle the unknowns. This is sent to a function block that sorts out the first iteration of input signal, output signal, and states from the Z array.

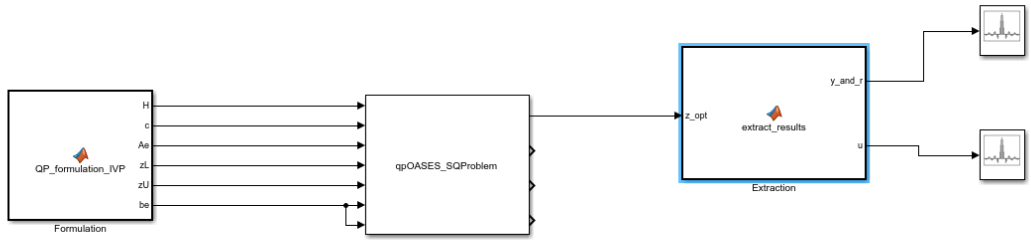


Figure 1: SImulink block code

```
function [H,c,Ae,zL,zU,be] = QP_formulation_IVP()
%discrete time model
A = [1.0745,0.1025,0,0;
     1.5079,1.0745,0,0;
     -0.0248,-0.0008,1,0.1;
     -0.5026, -0.0248,0,1];

B = [-0.0025;-0.0512;0.0025;0.0504];
C = [1,0,0,0;0,0,1,0];
N = 120; % prediction horizon length

% set up the references for the whole prediction horizon length
r = [zeros(1,N); % first row for first output (pendulum angle)
 zeros(1,N/2) ones(1,N/2)]; % second row for second output (cart
position)

%initial values of the states should be known or estimated
x0 = [0; 0; 0; 0];
%size of matrices
nx = 4; ny = 2; nu = 1;

%size of the unknow vector z
nz = N*(nx + nu + 2*ny);

%weighting matrices
Q=diag([1.0 1.0]); %tuning weight for error: there are 2
P=1e-3; %tuning weight for inputs: there is 1

% build matrices
H11 = kron(eye(N),P);
H22 = zeros(N*nx,N*nx);
H33 = kron(eye(N),Q);
H44 = zeros(N*ny,N*ny);
H_mat = blkdiag(H11,H22,H33,H44);

%qpOASES does not accept matrices, but only vectors
%we have to change H matrix to vector by stacking elements column wise
H = H_mat(:);
c = zeros(nz,1);

%constraints (from process model)
% from eq 3.33: state equation
Ae1u = -kron(eye(N),B);
Ae1x = eye(N*nx)-kron(diag(ones(N-abs(-1),1),-1),A);
Ae1e = zeros(N*nx,N*ny);
Ae1y = zeros(N*nx,N*ny);
be1 = [A*x0;zeros((N-1)*nx,1)];

%from eq 3.34: measurement equation
Ae2u = zeros(N*ny,N*nu);
Ae2x = -kron(eye(N),C);
Ae2e = zeros(N*ny,N*ny);
Ae2y = eye(N*ny);
be2 = zeros(N*ny,1);

%from eq 3.35: error equation
Ae3u = zeros(N*ny,N*nu);
Ae3x = zeros(N*ny,N*nx);
Ae3e = eye(N*ny);
```

```
Ae3y = eye(N*ny);

% since be3 contains the reference vector in specific order, we have to
use
% the function "reshape" to put the reference values in the right order
be3 = reshape(r,N*ny,1);
Ae_mat=[Ae1u Ae1x Ae1e Ae1y;
 Ae2u Ae2x Ae2e Ae2y;
 Ae3u Ae3x Ae3e Ae3y];

%qpOASES does not accept matrices, but only vectors
%we have to change Ae matrix to vector by stacking elements column wise
Ae = Ae_mat(:); %stacking column wise
% make the standard be vector
be=[be1;be2;be3];

%bounds (not specified so assume between -inf to +inf)
zL=(-Inf*ones(nz,1));
zU=(Inf*ones(nz,1));
H_mat = blkdiag(H11,H22,H33,H44);

%qpOASES does not accept matrices, but only vectors
%we have to change H matrix to vector by stacking elements column wise
H = H_mat(:);
Ae_mat=[Ae1u Ae1x Ae1e Ae1y;
 Ae2u Ae2x Ae2e Ae2y;
 Ae3u Ae3x Ae3e Ae3y];

%qpOASES does not accept matrices, but only vectors
%we have to change Ae matrix to vector by stacking elements column wise
Ae = Ae_mat(:); %stacking column wise
```

Table 1: Code for formatting the simple pendulum problem

```
function [y_and_r,u] = extract_results(z_opt)

    N = 120; %prediction horizon length
    nx = 4; nu = 1; ny = 2; %no. of states, inputs and outputs

    %extract results
    Ua = z_opt(1+N*(0) :N*(nu),1); %control inputs
    Xa = z_opt(1+N*(nu) :N*(nu+nx),:); %states
    Ea = z_opt(1+N*(nu+nx) :N*(nu+nx+ny),:); %error in tracking
    Ya = z_opt(1+N*(nu+nx+ny) :N*(nu+nx+ny+ny),:); %outputs
    %we use the reshape function to rearrange the data
    y_temp = reshape(Ya,ny,N);%arranged outputs(rows as signals, columns
as data)
    %the array plot in Simulink accepts signals in a format such that the
columns
    %represent the different signals and the rows represent the data for
each  signal. So we need to transpose it.
    %The first column is for output y1 (angle), second column for output
y2
    %(cart postion)
    y = y_temp';
    %Let us also plot the reference signals along with the output signals
for
    %better viewing.
```

```
    r = [zeros(N,1),[zeros(N/2,1);ones(N/2,1)]];

    %Put it as the third column (for r1) and fourth column (for r2).
    y_and_r = [y,r(:,1),r(:,2)];
    u_temp = reshape(Ua,nu,N); %arranged input(row as signals, columns as
data)
    u = u_temp';
```

Table 2: Code for extracting the current states

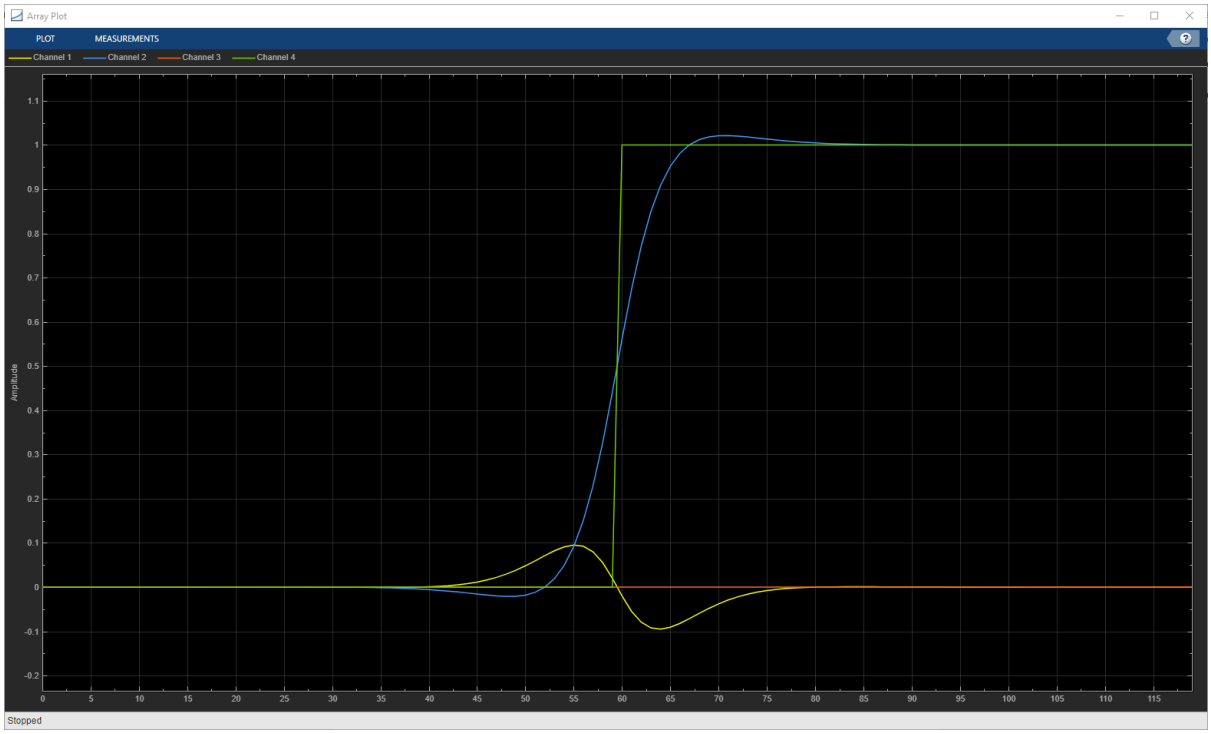
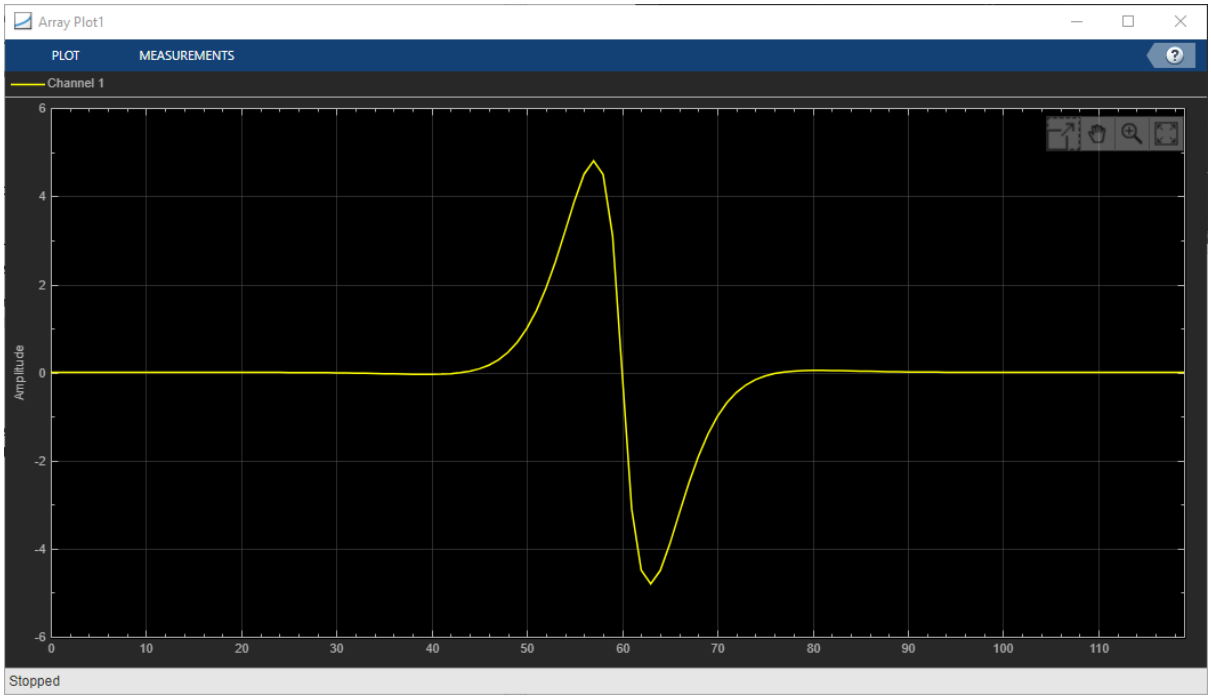


Figure 3: Plotted results from inputs, outputs and states for the simulation



Figur 1: Higher resolution plot of the input to the system