Table of Contents

 1
Physical constants 1
Pitch closed loop syntesis 1
Elevation closed loop analysis 2
[---- 10.2.1 - Continuous time state space ----] 3
[---- 10.2.2 - Forward Euler Method ----] 3
[---- 10.2.3 - Optimal trajectory ----] 3
[---- 10.2.4 - Implement QP Simulink ----] 7
[---- Task 10.3.1 LQ controller ----] 7


```
% Initialization for the helicopter assignment in TTK4135.
% Run this file before you execute QuaRC -> Build.

% Updated spring 2017, Andreas L. Flåten

clear all;
close all;
clc;
```

Physical constants

```
m_h = 0.4; % Total mass of the motors.
m_g = 0.03; % Effective mass of the helicopter.
l_a = 0.65; % Distance from elevation axis to helicopter body
l_h = 0.17; % Distance from pitch axis to motor

% Moments of inertia
J_e = 2 * m_h * l_a *l_a;         % Moment of interia for elevation
J_p = 2 * ( m_h/2 * l_h * l_h);   % Moment of interia for pitch
J_t = 2 * m_h * l_a *l_a;         % Moment of interia for travel

% Identified voltage sum and difference
V_s_eq = 6.6; % Identified equilibrium voltage sum.
V_d_eq = 0.4; % Identified equilibrium voltage difference.

% Model parameters
K_p = m_g*9.81; % Force to lift the helicopter from the ground.
K_f = K_p/V_s_eq; % Force motor constant.
K_1 = l_h*K_f/J_p;
K_2 = K_p*l_a/J_t;
K_3 = K_f*l_a/J_e;
K_4 = K_p*l_a/J_e;
```

Pitch closed loop syntesis

Controller parameters

```
w_p = 1.8; % Pitch controller bandwidth.
d_p = 1.0; % Pitch controller rel. damping.
K_pp = w_p^2/K_1;
K_pd = 2*d_p*sqrt(K_pp/K_1);
Vd_ff = V_d_eq;

% Closed loop transfer functions
Vd_max = 10 - V_s_eq; % Maximum voltage difference
deg2rad = @(x) x*pi/180;
Rp_max = deg2rad(15); % Maximum reference step
s = tf('s');
G_p = K_1/(s^2);
C_p = K_pp + K_pd*s/(1+0.1*w_p*s);
L_p = G_p*C_p;
S_p = (1 + L_p)^(-1);

plot_pitch_response = 0;
if plot_pitch_response
    figure()
    step(S_p*Rp_max); hold on;
    step(C_p*S_p*Rp_max/Vd_max);
    legend('norm error', 'norm input')
    title('Pitch closed loop response')
end
```

Elevation closed loop analysis

Controller parameters

```
w_e = 0.5; % Elevation controller bandwidth.
d_e = 1.0; % Elevation controller rel. damping.
K_ep = w_e^2/K_3;
K_ed = 2*d_e*sqrt(K_ep/K_3);
K_ei = K_ep*0.1;
Vs_ff = V_s_eq;

% Closed loop transfer functions
Vs_max = 10 - V_s_eq; % Maximum voltage sum
Re_max = deg2rad(10); % Maximum elevation step
G_e = K_3/(s^2);
C_e = K_ep + K_ed*s/(1+0.1*w_e*s) + K_ei/s;
L_e = G_e*C_e;
S_e = (1 + L_e)^(-1);

plot_elev_response = 0;
if plot_elev_response
    figure()
    step(S_e*Re_max);
    hold on;
    step(C_e*S_e*Re_max/Vs_max);
    legend('norm error', 'norm input')
    title('Elevation closed loop response')
end
```

```
% *---------------- PART 10.2 -------------------------*
```

[---- 10.2.1 - Continuous time state space ----]

```
A_c = [0 1 0 0;
       0 0 -K_2 0;
       0 0 0 1;
       0 0 -K_1*K_pp -K_1*K_pd];
B_c = [0; 0; 0; K_1*K_pp];
```

[---- 10.2.2 - Forward Euler Method ----]

```
I = eye(4);
delta_t = 0.25;
A = I + delta_t*A_c;
B = delta_t*B_c;
```

[---- 10.2.3 - Optimal trajectory ----]

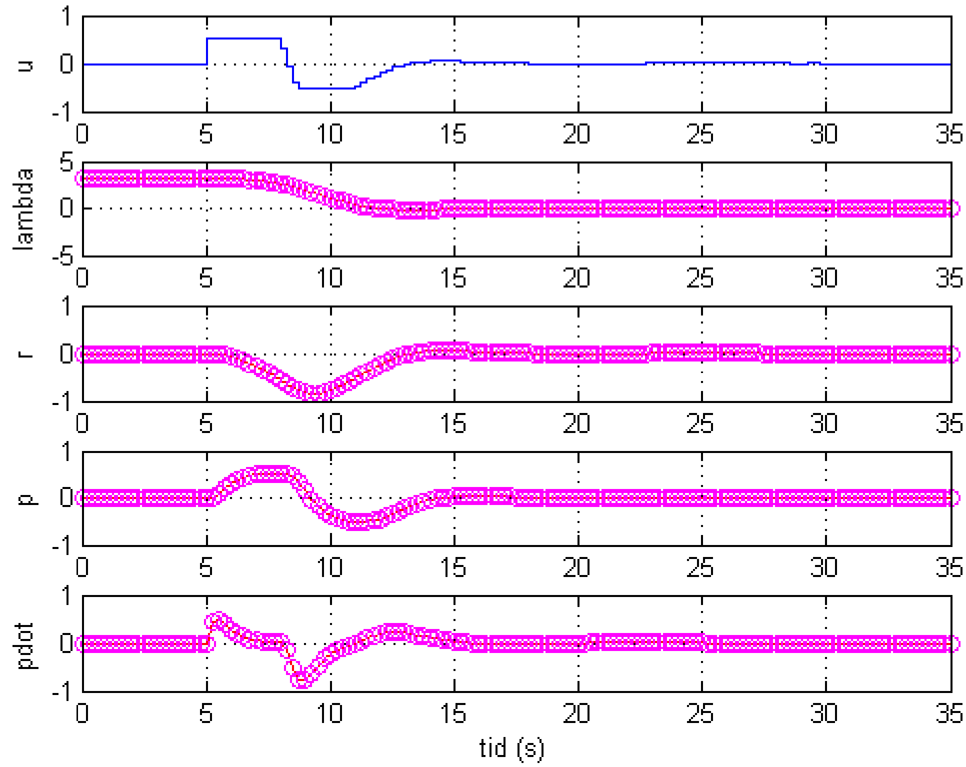
Inital values

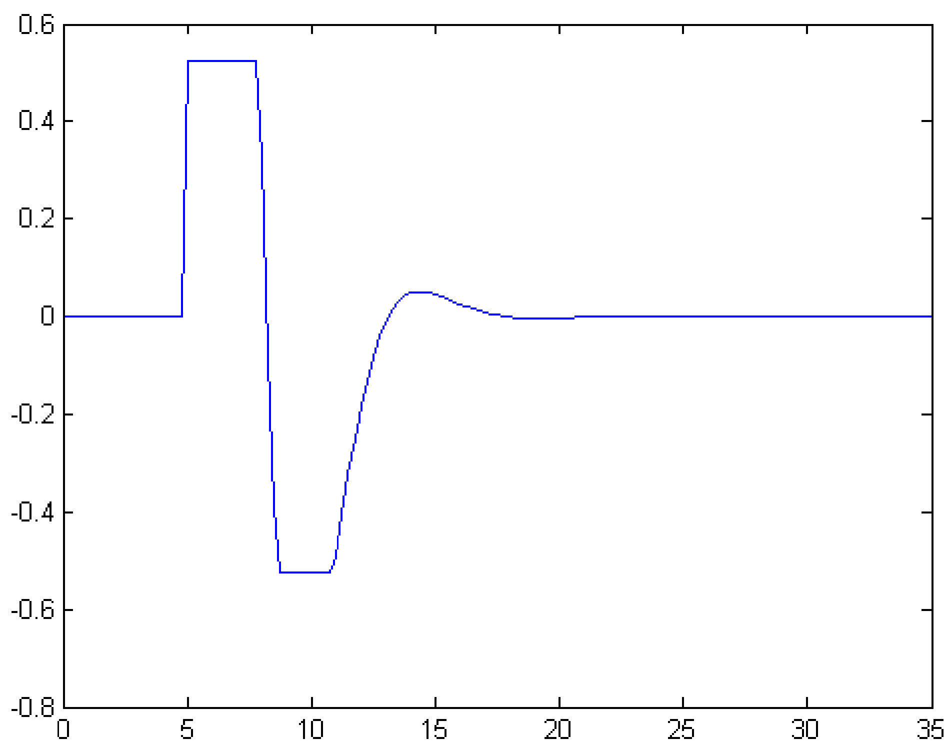
```
lambda_0 = pi;
lambda_f = 0;
x0 = [lambda_0 ;0 ;0 ;0];
xf = [lambda_f ;0 ;0 ;0];
Q1 = 2*diag([1 0 0 0]);
q0 = 1;
A1 = A;
B1 = B;
%k = 1:N;

% Number of states and inputs
mx = size(A1,2);                % Number of states (number of columns in A)
mu = size(B1,2);                % Number of inputs(number of columns in B)
N = 100;                        % Time horizon for states
M = N;                          % Time horizon for inputs
n = N*mx+M*mu;

% Generating A_eq, B_eq and Q
A_eq = gena2(A1, B1, N, mx, mu);
B_eq = zeros(size(A_eq,1),1);
B_eq(1:mx) = A1*x0;
Q = 2*genq2(Q1,q0,N,M,mu);

% Initialize z
c  = zeros(n,1);
z  = zeros(n,1);
z0 = z;

% Bounds
```

```
pk      = 30*pi/180;
ul      = -pk;                          % Lower bound on control -- u1
uu      = pk;                           % Upper bound on control -- u1

xl      = -Inf*ones(mx,1);              % Lower bound on states (no bound)
xu      = Inf*ones(mx,1);               % Upper bound on states (no bound)
xl(3)   = ul;                           % Lower bound on state x3
xu(3)   = uu;                           % Upper bound on state x3

% Generate constraints on measurements and inputs
[vlb,vub]   = genbegr2(N,M,xl,xu,ul,uu);
vlb(n)      = 0;                        % We want the last input to be zero
vub(n)      = 0;                        % We want the last input to be zero

% Using QP to find Z and lambda
tic
[z, lambda] = quadprog(Q,c,[],[],A_eq,B_eq,vlb,vub,x0);
t1 = toc;

% Calculate objective value
phi1 = 0.0;
PhiOut = zeros(n,1);
for i=1:n
  phi1=phi1+Q(i,i)*z(i)*z(i);
  PhiOut(i) = phi1;
end

% Extract control inputs and states
u  = [z(N*mx+1:n);z(n)]; % Control input from solution

x1 = [x0(1);z(1:mx:N*mx)];              % State x1 from solution
x2 = [x0(2);z(2:mx:N*mx)];              % State x2 from solution
x3 = [x0(3);z(3:mx:N*mx)];              % State x3 from solution
x4 = [x0(4);z(4:mx:N*mx)];              % State x4 from solution

num_variables = 5/delta_t;
zero_padding = zeros(num_variables,1);
unit_padding  = ones(num_variables,1);

u   = [zero_padding; u; zero_padding];
x1  = [pi*unit_padding; x1; zero_padding];
x2  = [zero_padding; x2; zero_padding];
x3  = [zero_padding; x3; zero_padding];
x4  = [zero_padding; x4; zero_padding];


% Plotting
t = 0:delta_t:delta_t*(length(u)-1);

figure();
hold on;
subplot(511)
stairs(t,u),grid
ylabel('u')
```

```
subplot(512)
plot(t,x1,'r',t,x1,'mo'),grid
ylabel('lambda')
subplot(513)
plot(t,x2,'r',t,x2','mo'),grid
ylabel('r')
subplot(514)
plot(t,x3,'r',t,x3,'mo'),grid
ylabel('p')
subplot(515)
plot(t,x4,'r',t,x4','mo'),grid
xlabel('tid (s)'),ylabel('pdot')



% Input imported to helicopter
calculated_input.time = t;
calculated_input.signals.values = u;
calculated_input.signals.dimensions = 1;
figure();
plot(calculated_input.time, calculated_input.signals.values)

% --------------------- PART 10.3 -----------------*
```

```
The interior-point-convex algorithm does not accept an initial point.
Ignoring X0.

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 function tolerance
and constraints are satisfied to within the default value of the constrain
```

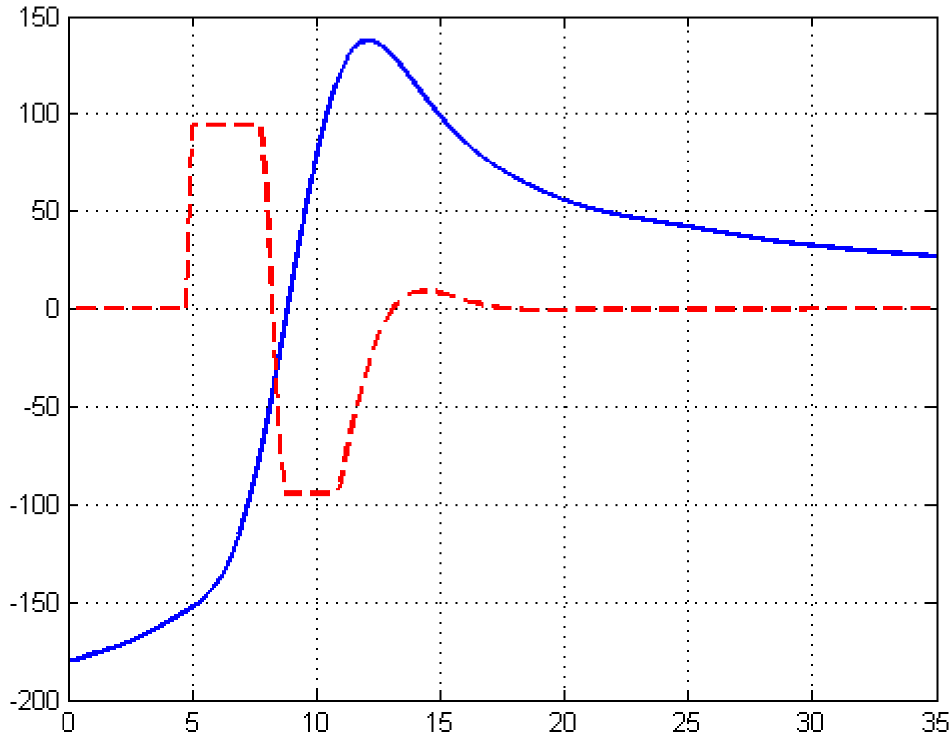

u
lambda
r
p
pdot
tid (s)



[---- 10.2.4 - Implement QP Simulink ----]

Plot travel

```
figure();
f = load('travel_10_2.mat');
plot(f.ans(1,:),f.ans(2,:), t,180*u,'r--', 'LineWidth',2); grid on;
```



[---- Task 10.3.1 LQ controller ----]

```
Q_lqr = diag([1 1 1 1]);
R_lqr = diag([1]);
K_lqr = dlqr(A,B,Q_lqr,R_lqr)
```

```
K_lqr =

   -0.6488   -2.4747    1.3122    0.4625
```

Published with MATLAB® R2014a