MATLAB kode fra Optimalisering og regulering

0.1 Optimization - helicopter controllers

Optimal Control of Pitch/Travel without Feedback

```
% TTK4135 Optimization and Control - Helicopter lab
%  Chapter 1, Optimal Control of Pitch/Travel without feedback

init;
dt = 0.25;                      % sampling time
sek_forst = 5;

% Continuous system model. x=[lambda r p p_dot]'
A = [    0       1       0           0           ;
         0       0       -K_2        0           ;
         0       0       0           1           ;
         0       0       -K_1*K_pp   -K_1*K_pd   ];
B = [0 0 0 K_1*K_pp]';

% Number of states and inputs
mx = size(A,2);                         % Number of states (number of ...
    columns in A)
mu = size(B,2);                         % Number of inputs(number of ...
    columns in B)

% Discrete system model
A1 = eye(mx) + dt*A;
B1 = dt*B;

% Initial values
x1_0 = pi ;                              % Lambda
x2_0 = 0;                               % r
x3_0 = 0;                               % p
x4_0 = 0;                               % p_dot
x0   = [x1_0 x2_0 x3_0 x4_0]';          % Initial values
```

```

% Time horizon and initialization

N   = 100;                              % Time horizon for states
M   = N;                                % Time horizon for inputs
z   = zeros(N*mx+M*mu,1);               % Initialize z for the whole ... horizon
z0 = z;                                 % Initial value for optimization

% Bounds

ul       = -30*pi/180;                  % Lower bound on control -- u1
uu       = 30*pi/180;                   % 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*mx+M*mu)   = 0;                   % We want the last input to be ... zero
vub(N*mx+M*mu)   = 0;                   % We want the last input to be ... zero

% Generate the matrix Q and the vector c (objecitve function weights ... in the QP problem)
Q1 = zeros(mx,mx);
Q1(1,1) = 1;                             % Weight on state x1
P1 = 1;                                  % Input weight(0.1, 1 and 10)
Q = genq2(Q1,P1,N,M,mu);                 % Generate Q

% Generate system matrixes for linear model
Aeq =gena2(A1,B1,N,mx,mu);               % Generate A
beq = [A1*x0; zeros((N-1)*mx,1)];        % Generate b

% Solve Qp problem with linear model
tic
[z,lambda] = quadprog(Q,c,[],[],Aeq,beq,vlb,vub,z0);
t1=toc;

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

% Extract control inputs and states
u  = [z(N*mx+1:N*mx+M*mu);z(N*mx+M*mu)]; % 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

Antall = 5/dt;
Nuller = zeros(Antall,1);
Enere  = ones(Antall,1);

u   = [Nuller; u; Nuller];
x1  = [pi*Enere; x1; Nuller];
x2  = [Nuller; x2; Nuller];
x3  = [Nuller; x3; Nuller];
x4  = [Nuller; x4; Nuller];

%save bane
t = 0:dt:dt*(length(u)-1);             % real time
pitch_setpoint=[t' u];

figure(2)
subplot(511)
stairs(t,u),grid
ylabel('u')
subplot(512)
plot(t,x1,'m',t,x1,'mo'),grid
ylabel('lambda')
subplot(513)
plot(t,x2,'m',t,x2','mo'),grid
ylabel('r')
subplot(514)
plot(t,x3,'m',t,x3,'mo'),grid
ylabel('p')
subplot(515)
plot(t,x4,'m',t,x4','mo'),grid
xlabel('tid (s)'),ylabel('pdot')
```

Optimal Control of Pitch/Travel with Feedback (LQ)

```
% TTK4135 Optimization and Control - Helicopter lab
%
% Chapter 2, Optimal Control of Pitch/Travel with Feedback (LQ)

init;
dt = 0.25;                               % sampling time
sek_forst = 5;

% Continuous model
A = [   0       1       0           0           ;
        0       0       -K_2        0           ;
        0       0       0           1           ;
        0       0       -K_1*K_pp   -K_1*K_pd   ];
B = [0 0 0 K_1*K_pp]';

% Number of states and inputs
mx = size(A,2);                         % Number of states (number of columns in A)
mu = size(B,2);                         % Number of inputs(number of columns in B)

% Discrete system model
A1 = eye(mx) + dt*A;
B1 = dt*B;

% Initial values
x1_0 = pi ;                              % Lambda
x2_0 = 0;                                % r
x3_0 = 0;                                % p
x4_0 = 0;                                % p_dot
x0   = [x1_0 x2_0 x3_0 x4_0]';           % Initial values

% Time horizon and initialization
N  = 100;                                % Time horizon for states
M  = N;                                  % Time horizon for inputs
z  = zeros(N*mx+M*mu,1);                 % Initialize z for the whole horizon
z0 = z;                                  % Initial value for optimization

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

```matlab
vlb(N*mx+M*mu)  = 0;                    % We want the last input to be ...
    zero
vub(N*mx+M*mu)  = 0;                    % We want the last input to be ...
    zero

% Generate the matrix Q and the vector c (objecitve function weights ...
    in the QP problem)

Q1 = zeros(mx,mx);
Q1(1,1) = 1;                            % Weight on state x1
P1 = 0.1;                               % Weight on input
Q = 2*genq2(Q1,P1,N,M,mu);              % Generate Q

% LQR tuning
Q_lqr = zeros(mx,mx);
Q_lqr(1,1)=1;
Q_lqr(2,2)=0;
Q_lqr(3,3)=1;
Q_lqr(4,4)=0;
P_lqr=1;
[K,P,E] = dlqr(A1,B1,2*Q_lqr,2*P_lqr);

% Generate system matrixes for linear model
Aeq =gena2(A1,B1,N,mx,mu);              % Generate A
beq = [A1*x0; zeros((N-1)*mx,1)];       % Generate b

% Solve Qp problem with linear model
tic
[z,lambda] = quadprog(Q,c,[],[],Aeq,beq,vlb,vub,z0);
t1=toc;


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

% Extract control inputs and states
u  = [z(N*mx+1:N*mx+M*mu);z(N*mx+M*mu)];% 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

Antall = 5/dt;
Nuller = zeros(Antall,1);
Enere  = ones(Antall,1);

u   = [Nuller; u; Nuller];
```

```
x1  = [pi*Enere; x1; Nuller];
x2  = [Nuller; x2; Nuller];
x3  = [Nuller; x3; Nuller];
x4  = [Nuller; x4; Nuller];

x = [x1 x2 x3 x4]';

%saving trajectory
t = 0:dt:dt*(length(u)-1);                % real time
pitch_setpoint = [t' u];
x_optimal = [t' x'];
```

Optimal Control of Pitch/Travel and Elevation with and without Feedback

```
% TTK4135 Optimization and Control - Helicopter lab
%
% Chapter 3, Optimal Control of Pitch/Travel and Elevation with and ...
    without Feedback

init;
dt = 0.25;                              % sampling time
sek_forst = 5;

% Continuous model

A = [   0       1       0           0          0              0 ...
            ;
        0       0       -K_2        0          0              0 ...
                    ;
        0       0       0           1          0              0 ...
                    ;
        0       0       -K_1*K_pp   -K_1*K_pd  0              0 ...
                    ;
        0       0       0           0          0              1 ...
                    ;
        0       0       0           0          -K_3*K_ep      -K_3*K_ed] ...
           ;
B = [   0       0       0           K_1*K_pp   0              0 ...
            ;
        0       0       0           0          0              K_3*K_ep]' ...
           ;

% Number of states and inputs

mx = size(A,2);                        % Number of states (number of ...
    columns in A)
mu = size(B,2);                        % Number of inputs(number of ...
    columns in B)

% Discrete system model
A1 = eye(mx) + dt*A;
B1 = dt*B;

% Initial values
x1_0 = pi;                              % Lambda
x2_0 = 0;                               % r
x3_0 = 0;                               % p
x4_0 = 0;                               % p_dot
x5_0 = 0;                               % e
x6_0 = 0;                               % e_dot
x0   = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0]';% Initial values

```

```
% Time horizon and initialization

N   = 55;                              % Time horizon for states
M   = N;                               % Time horizon for inputs
z   = zeros(N*mx+M*mu,1);              % Initialize z for the whole ...
    horizon
z0 = z;                                % Initial value for optimization

% Bounds
ul       = [-30*pi/180 -30*pi/180]';   % Lower bound on control -- u1
uu       = [30*pi/180 30*pi/180]';     % 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(1);                      % Lower bound on state x3
xu(3)    = uu(1);                      % Upper bound on state x3

% Generate constraints on measurements and inputs

[vlb,vub]        = genBegr2(N,M,xl,xu,ul,uu);
vlb(N*mx+M*mu)   = 0;                  % We want the last input to be ...
    zero
vub(N*mx+M*mu)   = 0;                  % We want the last input to be ...
    zero

% Generate the matrix Q and the vector c (objecitve function weights ...
    in the QP problem)

Q1 = zeros(mx,mx);
Q1(1,1) = 10;                          % Weight on state x1
P1 = 0.1*eye(2);                       % Weight on input
Q =  2*genq2(Q1,P1,N,M,mu);            % Generate Q
FUN= @(z) z'*Q*z;

% LQR
Q_lqr = zeros(mx,mx);
Q_lqr(1,1)=1;
Q_lqr(2,2)=0;
Q_lqr(3,3)=1;
Q_lqr(4,4)=0;
Q_lqr(5,5)=1;
Q_lqr(6,6)=1;
r_lqr = 1*eye(2);
[K,P,E] = dlqr(A1,B1,2*q_lqr,2*r_lqr);

% Generate system matrixes for linear model
Aeq =gena2(A1,B1,N,mx,mu);             % Generate A
beq = [A1*x0; zeros((N-1)*mx,1)];      % Generate b

% Solve Qp problem with linear model
options = optimset('disp','iter','Algorithm','active-set');
tic
[z,lambda] = fmincon(FUN,z0,[],[],Aeq,beq,vlb,vub,@el,options);
```

```
t1=toc;

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

% Extract control inputs and states

u1  = [z(mx*N+1:mu:end);z(N*mx+M*mu)] ; % Control input from solution
u2  = [z(mx*N+mu:mu:end);z(N*mx+M*mu)];

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
x5 = [x0(5);z(5:mx:N*mx)];              % State x5 from solution
x6 = [x0(6);z(6:mx:N*mx)];              % State x6 from solution

Antall = 5/dt;
Nuller = zeros(Antall,1);
Enere  = ones(Antall,1);

u1   = [Nuller; u1; Nuller];
u2   = [Nuller; u2; Nuller];
x1  = [pi*Enere; x1; Nuller];
x2  = [Nuller; x2; Nuller];
x3  = [Nuller; x3; Nuller];
x4  = [Nuller; x4; Nuller];
x5  = [Nuller; x5; Nuller];
x6  = [Nuller; x6; Nuller];

x = [x1 x2 x3 x4 x5 x6]';

%saving trajectory
t = 0:dt:dt*(length(u1)-1);             % real time
setpoint = [t' u1 u2];
x_optimal = [t' x'];
```

MATLAB kode fra Fartøystyring

0.2 Guidance and Control of Vehicles (2014) / Fartøystyring

Satellite Angular Rates and Gyro Bias Estimation Using the Ekstended Kalman Filter

```

function [z,A]=jaccsd(fun,x,u,t)
% JACCSD Jacobian through complex step differentiation
% [z J] = jaccsd(@f,x, u, t)
% z = f(x, u, t)
% J = df/dx(x, u, t)
%
z=fun(x,u,t);
n=numel(x);
m=numel(z);
A=zeros(m,n);
h=n*eps;
for k=1:n
    x1=x;
    x1(k)=x1(k)+h*1i; %for bedre speed la jeg til 1-tall foran i.
    A(:,k)=imag(fun(x1,u,t))/h;
end
```

```
clear all;

h=0.01; %samplingtime
N=1000; %iterations
sigma=0.001; %sigma^2

%initial values
x=[0 0 0 0 0 0 0 0 0]';
xd=[0.1 0.3 0.2 0 0 0]'; %Desired Euler Angles
I=diag([0.0108 0.0113 0.0048]); %inertia matrix
b_g=[0.012;0.017;0.014]; %The constant Bias
x(7:9) = b_g; %adding bias to the states
```

```

%Initialization Controller from Task 2.1
Kp=diag([0.07 0.07 0.07]);
Kd=diag([0.1 0.1 0.1]);
e=[0;0;0];

%Initialization of Kalman filter
y=[0;0;0;0;0;0];
x_bar=[0 0 0 0 0 0 0 0 0]';
p_bar= (x-x_bar)*(x-x_bar)' + 1e-9*eye(9); %Ensure pos def.
I_k=eye(9);
x_hat=[0 0 0 0 0 0 0 0 0]';
R=[1e-09*eye(3) zeros(3); zeros(3) sigma*eye(3)]; %6x6
H=[eye(3) zeros(3) zeros(3);zeros(3) eye(3) eye(3)];%6x9
Q=10^-9*eye(9); %9x9 With small diag to avoid singularities and ensure ...
    keep pos def.

%Function for the system 9x1
f= @(x, tau, t) [[1 sin(x(1))*tan(x(2)) cos(x(1))*tan(x(2));0 ...
    cos(x(1)) -sin(x(1));0 sin(x(1))/cos(x(2)) ...
    cos(x(1))/cos(x(2))]*x(4:6) ; ...
    inv(I)*(tau-(Smtrx(x(4:6))*(I*x(4:6)))); zeros(3,1)]; %function

%Used for data storing and plot
true=zeros(9,N);
est=zeros(9,N);
time=zeros(1,N);
%Loop
for i=1:N+1,
   t = (i-1)*h;

   %noise
   noise(:,i)=sqrt(0.001).*randn(3,1); %Computing noise


 %f(x_hat,tau_hat)+EW 9x1
   tau=-(Kp*e+Kd*x_hat(4:6)); %input with e=(x_hat_euler)-(x_d_euler)

    %measurement
   v = [zeros(3,1);
        noise(:,i)]; %Noise
   y = H*x + v;


   %%Kalman gain
   K=p_bar*H'*inv(H*p_bar*H'+R);

   %State estimate update
   x_hat=x_bar+K*(y-H*x_bar); %xbar+K(y-ybar). Ybar contains zero bias ...
       at the first iteration.

   P_hat=(I_k-K*H)*p_bar*(I_k-K*H)'+K*R*K';

```

```

    %linearization of f(x_hat,tau_hat) inserting x_hat for [phi theta . ...
        .  .], df/dx
    [fev, df_xhat] = jaccsd(f,x_hat, tau, t);

    %State estimate propagation and error covariance propagation ...
        (predictor k+1)
    x_bar=x_hat+h*fev; %x_hat + h*x_dothat

    %discrete-time matrix PHI
    PHI=I_k+h*df_xhat; %(11.52)
    p_bar=PHI*P_hat*PHI'+Q; %with no process noise, we do not include ...
        gamma.

    %From Task 2.1
   e=(x_hat(1:3)-xd(1:3)); %error


    %Euler integration(k+1)
    x=x+h*f(x,tau,t);
    xest=x_hat;



    %data storing
    time(i)=i*h; %to update time table
    true(:,i)=x;
    est(:,i)=xest;
    yvector(:,i)= y; %used only for checking if measurement y is ...
        correct by ploting it

end



%plotting actual and estimated angular rates


%Plotting of actual and estimated bias
```

Autopilot design

Matlab kodene som følger er resultatet fra et formelt gruppearbeid som ble karaktersatt. Det er autopiloter for et skip som heter MS Fartøystyring. Skipet har begrenset bevegelse i North-East planet slik at autopiloter for fart og heading er det eneste som trengs. Autopilotene er deretter brukt til "path-following" og "path tracking". Kodene inkluderer implementering og simulering av skipet. Har ikke tatt med kode for skip-modell.

```

% TTK4190 - Guidance and Control
% Assignment 3
% 746072 - Bjorn-Olav H. Eiksen
% 746029 - Marius Hjertaker
% 705009 - Sveinung Ohrem
% 745621 - Ole Maurice Rabanal
%
% This is the code for task 1.4 of the assignment

clear; clc;
dispstat('','init')                     % Simulation progress. ...
    Initialization. Does not print anything.
dispstat('run_task_1_4','keepthis');    % Output first

%Integration values
sim_time=3000;      %Simulation end time
dt=0.1;             %Steplength
L=sim_time/dt;      %Simulation steps

%%%%%%%%%%%%%%%%%%%%%%%%%       System      ...
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

K=-0.0598;          %Transfer function gain
T=115.5667;         %Transfer function time constant

%%%%%%%%%%%%%%%%%%%%%%%%%     User parameters      ...
    %%%%%%%%%%%%%%%%%%%%%%%%

x_0=[6.63,0,0,0,0,0]';  %x = [ u v r x y psi]'
omega_b=0.06;           %Bandwith of controller
zeta=1;                 %Damping coefficient

%%%%%%%%%%%%%%%%%%%%%%%%%     References        ...
    %%%%%%%%%%%%%%%%%%%%%%%%%%%

t=0:dt:(sim_time);                      %Time steps
psi_d=-0.3*sin(0.008*t);                %Desired heading(yaw)
r_d=-0.3*0.008*cos(0.008*t);            %Desired yaw rate
r_d_dot=0.3*0.008*0.008*sin(0.008*t);   %Desired yaw accceleration


%%%%%%%%%%%%%%%%%%%%%%%%%    Controller parameters        ...
```

```
    %%%%%%%%%%%%%%%%%

omega_n=omega_b/(sqrt(1-2*zeta^2+...
    sqrt(4*zeta^4-4*zeta^2+2)));        %Natural frequency

Kp=(T/K)*omega_n^2;                     %Proportional gain
Kd=(T/K)*2*zeta*omega_n-1/T;            %Derivative gain
Ki=(omega_n/10)*Kp;                     %Integrator gain

%%%%%%%%%%%%%%%%%%%%%%%%%%       Data storage          ...
    %%%%%%%%%%%%%%%%%%%%

time=zeros(1,L+1);          %Time
x=zeros(6,L+1);             %States x=[ u v r x y psi]'
tau=zeros(2,L);             %Inputs to ms fartoystyring tau=[Δ_c  n_c]'
e=zeros(2,L);               %Error

x(:,1)=x_0;                 %Set first step equal to initial values
e_int=zeros(1,L+1);         %Error integrated

%%%%%%%%%%%%%%%%%%%%%%%%%      Control loop         ...
    %%%%%%%%%%%%%%%%%%%%%%%

for i=1:L
    if (mod(i,100))==0
        dispstat(sprintf('Progress %02d%%',round((i/L)*100)));     % ...
            Simulation progress in %
    end
    time(i+1)=time(i)+dt;       %Time

    %Control algorithm
    e(:,i)=[x(6,i)-psi_d(i);x(3,i)-r_d(i)];     %Error
    e_int(i+1)=e_int(i)+dt*e(1,i);              %Error integrated

    Δ_ff=(1/K)*r_d(i)+(T/K)*r_d_dot(i);         %Feed forward input
    Δ_PID=-Kp*e(1,i)-Kd*e(2,i)-Ki*e_int(i+1);   %PID control input

    tau(:,i)=[Δ_ff+Δ_PID ; 7.3];            %Input with constant shaft ...
        velocity
    x_dot=msfartoystyring(x(:,i),tau(:,i),1);       %Send to ms ...
        fartoystyring

    %Integrate
    x(:,i+1)=x(:,i)+dt*x_dot;                   %Update states
end

%%%%%%%%%%%%%%%%%%%%%%      Plot figures         ...
    %%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% TTK4190 - Guidance and Control
% Assignment 3
% 746072 - Bjorn-Olav H. Eiksen
```

```
% 746029 - Marius Hjertaker
% 705009 - Sveinung Ohrem
% 745621 - Ole Maurice Rabanal
%
% This is the code for task 1.8 of the assignment

clear; clc;
close all
dispstat('','init')                    % Simulation progress. ...
    Initialization. Does not print anything.
dispstat('run_task_1_8','keepthis');    % Output first

%Integration values
sim_time=10000;                        %Simulation end time
dt=1;                                  %Steplength
L=sim_time/dt;                         %Simulation steps

%%%%%%%%%%%%%%%%%%%%%%%%%       System       ...
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Heading
K=-0.0598;
T=115.5667;
%Surge
Ksurge=0.9820;
Tsurge=450.8367;
time=0:dt:sim_time;

%User parameters
x_0=[4,0,0,0,0,0]';                    %x = [ u v r x y psi]'
omega_b=0.06;                          %Bandwith for heading controller
omega_bs=0.015;                        %Bandwith for surge controller
zeta=1;                                %Damping of system
n_c_max=85*2*pi/60;                    %Max shaft velocity[rad/s]

%%%%%%%%%%%%%%%%%%%%%%%%%       References       ...
    %%%%%%%%%%%%%%%%%%%%%%%%%%

%%Choose these for time varying reference in yaw
%  psi_d=-0.3*sin(0.008*time(1:L));
%  r_d=-0.3*0.008*cos(0.008*time(1:L));
%  r_d_dot=0.3*0.008*0.008*sin(0.008*time(1:L));
%
% psi_d=[psi_d(1:(L/2)) ones(1,L/2).*psi_d(end)];
% r_d=[r_d(1:(L/2)) zeros(1,L/2)];
% r_d_dot=[r_d_dot(1:(L/2)) zeros(1,L/2)];

%%Choose these for constant yaw and yaw rate:
psi_d=zeros(1,L);
r_d=zeros(1,L);
r_d_dot=zeros(1,L);

%%Choose between step input or time varying input for u:
u_r=[4.*ones(1,700/dt) ...
```

```
    7.*ones(1,L-700/dt)];              %Step
%u_r=4+abs(sin(0.001*time(1:L)));      %Time varying

%%%%%%%%%%%%%%%%%%%%%%%     Controller parameters      ...
    %%%%%%%%%%%%%%%%%%%
%Heading controller
omega_n=omega_b/(sqrt(1-2*zeta^2+...
    sqrt(4*zeta^4-4*zeta^2+2)));        %Natural frequency
Kp=(T/K)*omega_n^2;                     %Kp
Kd=(T/K)*2*zeta*omega_n;                %Kd
Ki=(T/K)*((omega_n^3)/10);              %Ki

%Surge controller
omega_n_surge=omega_bs/(sqrt(1-2*...
zeta^2+sqrt(4*zeta^4-4*zeta^2+2)));     %Natural frequency
Kp_surge=5.7388;                             %Kp
Ti_surge=Tsurge+50;                     %Ti

%%%%%%%%%%%%%%%%%%%%%%%     Data storage                ...
    %%%%%%%%%%%%%%%%%%%

x=zeros(6,L+1);                         %x = [ u v r x y psi]'
tau=zeros(2,L);                         %tau=[Δ_c  n_c]'
e=zeros(3,L);                           %e=[psi,r,u]'
e_int=zeros(3,L+1);                     %Integrated error

%%%%%%%%%%%%%%%%%%%%%%%     Reference prefilter       ...
    %%%%%%%%%%%%%%%%%%%%
omega_ref=0.05;                         %Natural frequency for ...
    reference model
zeta_ref=1;                             %Damping in prefilter
prefilter=ss([0 1;-omega_ref^2 ...
    -2*zeta_ref*omega_ref],[0;...
    omega_ref^2],eye(2),zeros(2,1));    %Making state space for prefilter
[ref,¬]=lsim(prefilter,u_r,time...
    (1:end-1),[u_r(1);0]);              %Simulate to get reference model
u_d=ref(:,1)';                          %Filtered desired value
u_d_dot=ref(:,2)';                      %Derivative of filtered ...
    desired value

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x(:,1)=x_0;                             %Set x_0
for i=1:L
    if (mod(i,100))==0
        dispstat(sprintf('Progress %02d%%',round((i/L)*100)));     % ...
            Simulation progress in %
    end

    %Control algorithm
    e(:,i)=[x(6,i)-psi_d(i);...
        x(3,i)-r_d(i);x(1,i)-u_d(i)];   %Error
    e_int(:,i+1)=e_int(:,i)+dt*e(:,i);  %Error integrated

```

```
    %Heading
    Δ_ff=(1/K)*r_d(i)+...
        (T/K)*r_d_dot(i);             %Reference feed forward for ... heading
    Δ_PID=-Kp*e(1,i)-Kd*...
        e(2,i)-Ki*e_int(1,i+1);       %PID controller for heading

    %Surge
    n_c_ff=(1/Ksurge)*u_d(i)+...
        (Tsurge/Ksurge)*u_d_dot(i);   %Reference feed forward for surge
    n_c=-Kp_surge*e(3,i)-(Kp_surge...
        /Ti_surge)*e_int(3,i+1);      %PI controller for surge

    %Anti Windup
    if abs(n_c+n_c_ff)>n_c_max
        Δ_I=-Ki*e_int(3,i+1)-(-Ki*e_int(3,i));
        if sign(Δ_I)≠sign(n_c+n_c_ff)
            e_int(3,i+1)=e_int(3,i);          %Locks the integrator ... if the
                n_c=-Kp_surge*e(3,i)-...      %Shaft speed reaches ... max value
                (Kp_surge/Ti_surge)...
                *e_int(3,i+1);
        end
    end

    %Set tau and send it to MS Fartoystyring
    tau(:,i)=[Δ_ff+Δ_PID...     %Sets tau
        ;n_c_ff+n_c];
    x_dot=msfartoystyring(x(:,i),...    %Send to MS Fartoystyring
        tau(:,i),1);

    %Integrate
    x(:,i+1)=x(:,i)+dt*x_dot;         %Integrates with forward Euler
end
```

```
% TTK4190 - Guidance and Control
% Assignment 3
% 746072 - Bjorn-Olav H. Eiksen
% 746029 - Marius Hjertaker
% 705009 - Sveinung Ohrem
% 745621 - Ole Maurice Rabanal
%
% This is the code for task 2.3 of the assignment

clear; clc;
dispstat('','init')                                % ... Simulation progress. Initialization. Does not print anything.
dispstat('run_task_2_3. Path-tracking, with direct assignment',...
    'keepthis');                                   % Output first
no=menu('Choose steering method','Enclosure-Based Steering',...
```

```
    'Lookahead-Based Steering','Exit');                          % ...
        Steering method menu
no2=menu('Choose WP changing method','Circle of acceptance',...
    'Circle of acceptance ignoring cross-track error','Exit');  % WP ...
        changing menu
if no==3 || no2==3  % Exit?
    break
end
%Integration values
sim_time=2500;      %Simulation end time
dt=1;               %Steplength
L=sim_time/dt;      %Simulation steps

%System
K=-0.0598;          % Heading model gain
T=115.5667;         % Heading model time constant
Ksurge=0.9820;      % Surge model gain
Tsurge=450.8367;    % Surge model time constant

%User parameters
x_0=[6.63,0,0,0,0,0]';  % Initial condition, x = [ u v r x y psi]'
n_c_max=85*2*pi/60;     % Max shaft velocity [rad/s]
Δ_max  = 25*pi/180; % max rudder angle [rad]

%%%%%%%%%%%%%%% Controller parameters %%%%%%%%%%%%%%%%%%
%Heading controller
zeta=1;         % Damping coefficient
omega_b=0.03;   % Bandwidth frequency
omega_n=omega_b/(sqrt(1-2*zeta^2+sqrt(4*zeta^4-4*zeta^2+2)));   % ...
    Natural frequency
Kp=(T/K)*omega_n^2;             % Heading controller proportional gain
Kd=(T/K)*2*zeta*omega_n-1/T;    % Heading controller derivative gain
Ki=(omega_n/10)*Kp;             % Heading controller integral gain

%Surge controller
Kp_surge=2.7388;    % Surge controller proportional gain
Ti_surge=Tsurge+50; % Surge controller integral time

%%%%%%%%%%%%%% Data storage %%%%%%%%%%%%%%%%%%%%%%%%%%%%
time=zeros(1,L+1);  % Time vector
x=zeros(6,L+1);     % State vector, x = [ u v r x y psi]'
tau=zeros(2,L);     % Input vector, tau = [Δ n_c]'
e=zeros(3,L);       % Control error vector, e = [psi¬ r¬ u¬]'
e_int=zeros(3,L);   % Integrated control error e_int = [psi¬_int ...
    r¬_int u¬_int]'
LOS=zeros(2,L);     % LOS point for Enclosed-Based Steering, LOS = ...
    [xlos ylos]'
psi_r=zeros(1,L);   % Requested heading
psi_d=zeros(3,L);   % Filtered heading references psi_d = [psi_d ...
    psi_d_dot psi_d_ddot]'
u_r=zeros(1,L);     % Requested surge speed
u_d=zeros(2,L);     % Filtered surge references u_d = [u_d u_d_dot]'

```

```
%%%%%%%%%%%%%% Prefilters %%%%%%%%%%%%%%%%%
%Heading
omega_ref=0.15; % Natural frequency for reference model
zeta_ref=1;     % Damping coefficient
A_h=[0 1 0;0 0 1;-omega_ref^3 -(2*zeta_ref+1)*omega_ref^2 ...
    -(2*zeta_ref+1)*omega_ref]; % Continous A matrix
B_h=[0;0;omega_ref^3];          % Continous B matrix
psi_d_0 = [x_0(6) x_0(3) 0]';   % Initial state of filtered heading ...
    reference

%Surge
omega_ref=0.05; % Natural frequency for reference model
zeta_ref=1;     % Damping coefficient
A_s=[0 1;-omega_ref^2 -2*zeta_ref*omega_ref];   % Continous A matrix
B_s=[0;omega_ref^2];     % Continous B matrix
u_d_0 = [x_0(1) 0]';     % Initial state of filtered surge references


x(:,1)=x_0;              % Set initial state
load('WP.mat');          % Load waypoints
k = 2;                   % Next waypoint selector
R_LOS=4*305;             % LOS radius for Enclosed-Based steering
Δ_lookahead=3*305;   % Lookahead distance for Lookahead-Based Steering
R = 2*305;               % Acceptance radius


AntiWU=zeros(2,L); % Anti wind-up log for the surge controller: 0 -> ...
    No action, 1 -> integrator hold, -1 -> Saturation, but no action, ...
    [AntiWU_heading AntiWU_surge]'
CurrentWP=zeros(1,L);    % Current WP log
for i=1:L
    if (mod(i,100))==0
        dispstat(sprintf('Progress %02d%%',round((i/L)*100)));     % ...
            Simulation progress in %
    end
    time(i+1)=time(i)+dt;    % Update time vector

    %%%%%%%% Waypoint selector %%%%%%%%
    if k < size(WP,2)    % More waypoints available?
        if no2==1        % Circle of acceptance selected
            if (((WP(1,k)-x(4,i))^2 + (WP(2,k)-x(5,i))^2) ≤ R^2)
                k=k+1;  % Select next waypoint
            end
        elseif no2==2    % Circle of acceptance ignoring cross-track ...
            error selected
            if NextWP(WP(:,k-1:k),x(4:5,i),R)
                k=k+1;  % Select next waypoint
            end
        else
            error('Invalid choice for WP changing method')
        end
    end
    CurrentWP(i)=k; % Log current waypoint
```

```
    %%%%%%%% Guidance system %%%%%%%%
    if no==1        % Enclosed-Based steering
        LOS(:,i) = LOSp(WP(:,k-1:k),x(4:5,i),R_LOS);                 % ...
            Calculate LOS point
        Chi_d = atan2((LOS(2,i) - x(5,i)),(LOS(1,i) - x(4,i)));     % ...
            Calculate desired course
    elseif no==2    % Lookahead-Based steering
        Chi_d = Lookahead(WP(:,k-1:k),x(4:5,i),Δ_lookahead);    % ...
            Calculate desired course
    else
        error('Invalid choice for steering method')
    end
    U_d = 6.63;         % Desired craft speed, constant

    %Mapping
    psi_r(i) = Chi_d;   % Requested heading
    u_r(i) = U_d;       % Requested surge speed

    %Prefilters
    if i==1 % First iteration, use initial state as psi_d_(i-1)
        psi_d(:,i)=(eye(3)+dt.*A_h)*psi_d_0+dt*B_h*psi_r(i);         % ...
            Heading prefilter
        u_d(:,i)=(eye(2)+dt.*A_s)*u_d_0+dt*B_s*u_r(i);               % ...
            Surge prefilter
    else
        psi_d(:,i)=(eye(3)+dt.*A_h)*psi_d(:,i-1)+dt*B_h*psi_r(i);   % ...
            Heading prefilter
        u_d(:,i)=(eye(2)+dt.*A_s)*u_d(:,i-1)+dt*B_s*u_r(i);         % ...
            Surge prefilter
    end

    %%%%%%%% Control algorithm %%%%%%%%
    e(:,i)=[x(6,i)-psi_d(1,i);x(3,i)-psi_d(2,i);x(1,i)-u_d(1,i)];   % ...
        Control Error
    if i==1 % First iteration, use zero as e_int(i-1)
        e_int(:,i)=dt*e(:,i);                   % Integrate error
    else
        e_int(:,i)=e_int(:,i-1)+dt*e(:,i);  % Integrate error
    end

    %Heading controller
    Δ_ff=(1/K)*psi_d(2,i)+(T/K)*psi_d(3,i);     % Heading reference ...
        feed forward
    Δ_PID=-Kp*e(1,i)-Kd*e(2,i)-Ki*e_int(1,i);   % Heading PID controller

    %Surge controller
    n_c_ff=(1/Ksurge)*u_d(1,i)+(Tsurge/Ksurge)*u_d(2,i);        % ...
        Surge reference feed forward
    n_c_PI=-Kp_surge*e(3,i)-(Kp_surge/Ti_surge)*e_int(3,i);     % ...
        Surge PI controller

    %Integrator Anti Wind-up
```

```
    %Heading
    if abs(Δ_PID+Δ_ff)>Δ_max && i>1   % Rudder saturated, and not ...
        first iteration
        AntiWU(1,i)=-1;     % Log anti WU
        Δ_I=-Ki*e_int(1,i)-(-Ki*e_int(1,i-1));  % Integrator direction
        if sign(Δ_I)==sign(Δ_PID+Δ_ff)  % Integrator direction is the ...
            same as the propellor speed (Integrates further into ...
            saturation)
            e_int(1,i)=e_int(1,i-1);                % Reset integrator ...
                to previous value
            Δ_PID=-Kp*e(1,i)-Kd*e(2,i)-Ki*e_int(1,i);    % Calculate ...
                heading PID controller output again
            AntiWU(1,i)=1;  % Log anti WU
        end
    end
    %Surge
    if abs(n_c_PI+n_c_ff)>n_c_max && i>1    % Shaft speed saturated, ...
        and not first iteration
        AntiWU(2,i)=-1;     % Log anti WU
        Δ_I=-(Kp_surge/Ti_surge)*e_int(3,i)-(-(Kp_surge/Ti_surge)...
            *e_int(3,i-1)); % Integrator direction
        if sign(Δ_I)==sign(n_c_PI+n_c_ff)   % Integrator direction is ...
            the same as the propellor speed (Integrates further into ...
            saturation)
            e_int(3,i)=e_int(3,i-1);            % Reset integrator to ...
                previous value
            n_c_PI=-Kp_surge*e(3,i)-(Kp_surge/Ti_surge)...
                *e_int(3,i);    % Calculate surge PI controller output ...
                    again
            AntiWU(2,i)=1;  % Log anti WU
        end
    end

    %%%%%%%% MS Fartoystyring %%%%%%%%
    tau(:,i)=[Δ_ff+Δ_PID;n_c_ff+n_c_PI];   % Define input
    if tau(2,i)<0       % Ensure non-negative propellor speed, ...
        msfartoystyring crashes if n_c < 0
        tau(2,i)=0;
        warning(['Desired propellor speed was negative at iteration ' ...
            num2str(i) '. Modified to zero.'])
    end
    x_dot=msfartoystyring(x(:,i),tau(:,i),1);   % Simulate ship with ...
        current

    %Integrate
    x(:,i+1)=x(:,i)+dt*x_dot;
end

pathplotter(x(4,:), x(5,:), x(6,:), dt, 50, 0, sim_time, 0, WP)

%%%%%%%% Optional plots %%%%%%%%
```

```
% TTK4190 - Guidance and Control
% Assignment 3
% 746072 - Bjorn-Olav H. Eiksen
% 746029 - Marius Hjertaker
% 705009 - Sveinung Ohrem
% 745621 - Ole Maurice Rabanal
%
% This is the code for task 2.6 of the assignment

clear; clc;
dispstat('','init')                                       % ...
    Simulation progress. Initialization. Does not print anything.
dispstat('run_task_2_6. Path-tracking, with transformed assignment',...
    'keepthis');                                          % Output first
no=menu('Choose steering method','Enclosure-Based Steering',...
    'Lookahead-Based Steering','Exit');                       % ...
        Steering method menu
no2=menu('Choose WP changing method','Circle of acceptance',...
    'Circle of acceptance ignoring cross-track error','Exit');  % WP ...
        changing menu
if no==3 || no2==3  % Exit?
    break
end
%Integration values
sim_time=2700;      %Simulation end time
dt=1;               %Steplength
L=sim_time/dt;      %Simulation steps

%System
K=-0.0598;          % Heading model gain
T=115.5667;         % Heading model time constant
Ksurge=0.9820;      % Surge model gain
Tsurge=450.8367;    % Surge model time constant

%User parameters
x_0=[6.63,0,0,0,0,0]';  % Initial condition, x = [ u v r x y psi]'
n_c_max=85*2*pi/60;     % Max shaft velocity [rad/s]
Δ_max  = 25*pi/180; % max rudder angle [rad]

%%%%%%%%%%%%%%% Controller parameters %%%%%%%%%%%%%%%%%
%Heading controller
zeta=1;         % Damping coefficient
omega_b=0.03;   % Bandwidth frequency
omega_n=omega_b/(sqrt(1-2*zeta^2+sqrt(4*zeta^4-4*zeta^2+2)));   % ...
    Natural frequency
Kp=(T/K)*omega_n^2;             % Heading controller proportional gain
Kd=(T/K)*2*zeta*omega_n-1/T;    % Heading controller derivative gain
Ki=(omega_n/10)*Kp;             % Heading controller integral gain

%Surge controller
Kp_surge=2.7388;    % Surge controller proportional gain
Ti_surge=Tsurge+50; % Surge controller integral time

```

```
%%%%%%%%%%%%%% Data storage %%%%%%%%%%%%%%%%%%%%%%%%%%%%
time=zeros(1,L+1);  % Time vector
x=zeros(6,L+1);     % State vector, x = [ u v r x y psi]'
tau=zeros(2,L);     % Input vector, tau = [Δ n_c]'
e=zeros(3,L);       % Control error vector, e = [psi¬ r¬ u¬]'
e_int=zeros(3,L);   % Integrated control error e_int = [psi¬_int ...
    r¬_int u¬_int]'
LOS=zeros(2,L);     % LOS point for Enclosed-Based Steering, LOS = ...
    [xlos ylos]'
psi_r=zeros(1,L);   % Requested heading
psi_d=zeros(3,L);   % Filtered heading references psi_d = [psi_d ...
    psi_d_dot psi_d_ddot]'
u_r=zeros(1,L);     % Requested surge speed
u_d=zeros(2,L);     % Filtered surge references u_d = [u_d u_d_dot]'

%%%%%%%%%%%%%% Prefilters %%%%%%%%%%%%%%%%%%
%Heading
omega_ref=0.15; % Natural frequency for reference model
zeta_ref=1;     % Damping coefficient
A_h=[0 1 0;0 0 1;-omega_ref^3 -(2*zeta_ref+1)*omega_ref^2 ...
    -(2*zeta_ref+1)*omega_ref]; % Continous A matrix
B_h=[0;0;omega_ref^3];          % Continous B matrix
psi_d_0 = [x_0(6) x_0(3) 0]';   % Initial state of filtered heading ...
    reference

%Surge
omega_ref=0.05; % Natural frequency for reference model
zeta_ref=1;     % Damping coefficient
A_s=[0 1;-omega_ref^2 -2*zeta_ref*omega_ref];   % Continous A matrix
B_s=[0;omega_ref^2];     % Continous B matrix
u_d_0 = [x_0(1) 0]';     % Initial state of filtered surge references


x(:,1)=x_0;              % Set initial state
load('WP.mat');          % Load waypoints
k = 2;                   % Next waypoint selector
R_LOS=4*305;             % LOS radius for Enclosed-Based steering
Δ_lookahead=3*305;  % Lookahead distance for Lookahead-Based Steering
R = 2*305;               % Acceptance radius


AntiWU=zeros(2,L); % Anti wind-up log for the surge controller: 0 -> ...
    No action, 1 -> integrator hold, -1 -> Saturation, but no action, ...
    [AntiWU_heading AntiWU_surge]'
CurrentWP=zeros(1,L);    % Current WP log
for i=1:L
    if (mod(i,100))==0
        dispstat(sprintf('Progress %02d%%',round((i/L)*100)));     % ...
            Simulation progress in %
    end
    time(i+1)=time(i)+dt;    % Update time vector

    %%%%%%%% Waypoint selector %%%%%%%%
```

```
    if k < size(WP,2)    % More waypoints available?
        if no2==1        % Circle of acceptance selected
            if (((WP(1,k)-x(4,i))^2 + (WP(2,k)-x(5,i))^2) ≤ R^2)
                k=k+1;  % Select next waypoint
            end
        elseif no2==2    % Circle of acceptance ignoring cross-track ...
            error selected
            if NextWP(WP(:,k-1:k),x(4:5,i),R)
                k=k+1;  % Select next waypoint
            end
        else
            error('Invalid choice for WP changing method')
        end
    end
    CurrentWP(i)=k; % Log current waypoint

    %%%%%%%% Guidance system %%%%%%%%
    if no==1          % Enclosure-Based steering
        LOS(:,i) = LOSp(WP(:,k-1:k),x(4:5,i),R_LOS);                % ...
            Calculate LOS point
        Chi_d = atan2((LOS(2,i) - x(5,i)),(LOS(1,i) - x(4,i)));     % ...
            Calculate desired course
    elseif no==2     % Lookahead-Based steering
        Chi_d = Lookahead(WP(:,k-1:k),x(4:5,i),∆_lookahead);    % ...
            Calculate desired course
    else
        error('Invalid choice for steering method')
    end
    U_d = 6.63;          % Desired craft speed, constant

    %Mapping
    u=x(1,i);
    v=x(2,i);
    U=sqrt(u^2+v^2);
    beta=asin(v/U);
    psi_r(i) = Chi_d-beta;    % Requested heading

    u_r(i) = sqrt(U_d^2-v^2);        % Requested surge speed

    %Prefilters
    if i==1 % First iteration, use initial state as psi_d_(i-1)
        psi_d(:,i)=(eye(3)+dt.*A_h)*psi_d_0+dt*B_h*psi_r(i);        % ...
            Heading prefilter
        u_d(:,i)=(eye(2)+dt.*A_s)*u_d_0+dt*B_s*u_r(i);              % ...
            Surge prefilter
    else
        psi_d(:,i)=(eye(3)+dt.*A_h)*psi_d(:,i-1)+dt*B_h*psi_r(i);   % ...
            Heading prefilter
        u_d(:,i)=(eye(2)+dt.*A_s)*u_d(:,i-1)+dt*B_s*u_r(i);         % ...
            Surge prefilter
    end

    %%%%%%%% Control algorithm %%%%%%%%
```

```
    e(:,i)=[x(6,i)-psi_d(1,i);x(3,i)-psi_d(2,i);x(1,i)-u_d(1,i)];   % ...
        Control Error
    if i==1 % First iteration, use zero as e_int(i-1)
        e_int(:,i)=dt*e(:,i);                      % Integrate error
    else
        e_int(:,i)=e_int(:,i-1)+dt*e(:,i);  % Integrate error
    end

    %Heading controller
    Δ_ff=(1/K)*psi_d(2,i)+(T/K)*psi_d(3,i);      % Heading reference ...
        feed forward
    Δ_PID=-Kp*e(1,i)-Kd*e(2,i)-Ki*e_int(1,i);    % Heading PID controller

    %Surge controller
    n_c_ff=(1/Ksurge)*u_d(1,i)+(Tsurge/Ksurge)*u_d(2,i);         % ...
        Surge reference feed forward
    n_c_PI=-Kp_surge*e(3,i)-(Kp_surge/Ti_surge)*e_int(3,i);      % ...
        Surge PI controller

    %Integrator Anti Wind-up
    %Heading
    if abs(Δ_PID+Δ_ff)>Δ_max && i>1      % Rudder saturated, and not ...
        first iteration
        AntiWU(1,i)=-1;      % Log anti WU
        Δ_I=-Ki*e_int(1,i)-(-Ki*e_int(1,i-1));  % Integrator direction
        if sign(Δ_I)==sign(Δ_PID+Δ_ff)  % Integrator direction is the ...
            same as the propellor speed (Integrates further into ...
            saturation)
            e_int(1,i)=e_int(1,i-1);                 % Reset integrator ...
                to previous value
            Δ_PID=-Kp*e(1,i)-Kd*e(2,i)-Ki*e_int(1,i);    % Calculate ...
                heading PID controller output again
            AntiWU(1,i)=1;  % Log anti WU
        end
    end
    %Surge
    if abs(n_c_PI+n_c_ff)>n_c_max && i>1        % Shaft speed ...
        saturated, and not first iteration
        AntiWU(2,i)=-1;      % Log anti WU
        Δ_I=-(Kp_surge/Ti_surge)*e_int(3,i)-(-(Kp_surge/Ti_surge)...
            *e_int(3,i-1)); % Integrator direction
        if sign(Δ_I)==sign(n_c_PI+n_c_ff)    % Integrator direction is ...
            the same as the propellor speed (Integrates further into ...
            saturation)
            e_int(3,i)=e_int(3,i-1);             % Reset integrator to ...
                previous value
            n_c_PI=-Kp_surge*e(3,i)-(Kp_surge/Ti_surge)...
                *e_int(3,i);     % Calculate surge PI controller output ...
                    again
            AntiWU(2,i)=1;  % Log anti WU
        end
    end

```

```
    %%%%%%%% MS Fartoystyring %%%%%%%%
    tau(:,i)=[Δ_ff+Δ_PID;n_c_ff+n_c_PI];    % Define input
    if tau(2,i)<0      % Ensure non-negative propellor speed, ...
        msfartoystyring crashes if n_c < 0
        tau(2,i)=0;
        warning(['Desired propellor speed was negative at iteration ' ...
            num2str(i) '. Modified to zero.'])
    end
    x_dot=msfartoystyring(x(:,i),tau(:,i),1);    % Simulate ship with ...
        current

    %Integrate
    x(:,i+1)=x(:,i)+dt*x_dot;
end

pathplotter(x(4,:), x(5,:), x(6,:), dt, 50, 0, sim_time, 0, WP)
```

```
% TTK4190 - Guidance and Control
% Assignment 3
% 105334 - Bjorn-Olav H. Eiksen
% 746029 - Marius Hjertaker
% 705009 - Sveinung Ohrem
% 745621 - Ole Maurice Rabanal
%
% This is the code for task 2.7 of the assignment

clear; clc;
dispstat('','init')                                        % ...
    Simulation progress. Initialization. Does not print anything.
dispstat('run_task_2_7. Path-tracking','keepthis');        % Output first

%Integration values
sim_time=12000;     % Simulation end time
dt=1;               % Steplength
L=sim_time/dt;      % Simulation steps

%System
K=-0.0261;          % Heading model gain, modified from -0.0598
T=261.348;          % Heading model time constant, modified from 115.5667
Ksurge=0.9820;      % Surge model gain
Tsurge=450.8367;    % Surge model time constant

%User parameters
load('WP.mat');             % Load waypoints
x_0=[6.63,0,0,0,0,0]';  % Initial condition, x = [ u v r x y psi]'
n_c_max=85*2*pi/60;     % Max shaft velocity [rad/s]
Δ_max  = 25*pi/180; % max rudder angle [rad]
s_d=2*305;              % Desired lateral distance
e_d=200;                % Desired cross-track error, e_d=0 causes the ...
    boat to line up with the target
U_a_max=4;              % Maximum approach speed
U_target=3;             % Target speed
```

```
target_0=[WP(:,2)' atan2(WP(2,2)-WP(2,1),WP(1,2)-WP(1,1))]';    % ...
    Initial position target, target = [x_t y_t psi_t]'

%%%%%%%%%%%%%%% Controller parameters %%%%%%%%%%%%%%%%%
%Heading controller
zeta=1;         % Damping coefficient
omega_b=0.06;   % Bandwidth frequency
omega_n=omega_b/(sqrt(1-2*zeta^2+sqrt(4*zeta^4-4*zeta^2+2)));   % ...
    Natural frequency
Kp=(T/K)*omega_n^2;             % Heading controller proportional gain
Kd=(T/K)*2*zeta*omega_n-1/T;    % Heading controller derivative gain
Ki=(omega_n/10)*Kp;             % Heading controller integral gain

%Surge controller
Kp_surge=2.7388;    % Surge controller proportional gain
Ti_surge=Tsurge+50; % Surge controller integral time

%%%%%%%%%%%%%% Data storage %%%%%%%%%%%%%%%%%%%%%%%%%%%
time=zeros(1,L+1);  % Time vector
x=zeros(6,L+1);     % State vector, x = [ u v r x y psi]'
tau=zeros(2,L);     % Input vector, tau = [Δ n_c]'
e=zeros(3,L);       % Control error vector, e = [psi¬ r¬ u¬]'
e_int=zeros(3,L);   % Integrated control error e_int = [psi¬_int ...
    r¬_int u¬_int]'
psi_r=zeros(1,L);   % Requested heading
psi_d=zeros(3,L);   % Filtered heading references psi_d = [psi_d ...
    psi_d_dot psi_d_ddot]'
u_r=zeros(1,L);     % Requested surge speed
u_d=zeros(2,L);     % Filtered surge references u_d = [u_d u_d_dot]'
target=zeros(3,L+1);% Target position target = [x_t y_t psi_t]'
track=zeros(2,L);   % Tracking distance track = [s e]'

%%%%%%%%%%%%%% Prefilters %%%%%%%%%%%%%%%%%%
%Heading
omega_ref=0.15; % Natural frequency for reference model
zeta_ref=1;     % Damping coefficient
A_h=[0 1 0;0 0 1;-omega_ref^3 -(2*zeta_ref+1)*omega_ref^2 ...
    -(2*zeta_ref+1)*omega_ref]; % Continous A matrix
B_h=[0;0;omega_ref^3];          % Continous B matrix
psi_d_0 = [x_0(6) x_0(3) 0]';   % Initial state of filtered heading ...
    reference

%Surge
omega_ref=0.05; % Natural frequency for reference model
zeta_ref=1;     % Damping coefficient
A_s=[0 1;-omega_ref^2 -2*zeta_ref*omega_ref];   % Continous A matrix
B_s=[0;omega_ref^2];    % Continous B matrix
u_d_0 = [x_0(1) 0]';    % Initial state of filtered surge references


x(:,1)=x_0;             % Set initial state
target(:,1)=target_0;   % Initial position target
Δ_lookahead=8*305;  % Lookahead distance for Lookahead-Based Steering
```

```
Δ_s=2400;             % Speed tuning parameter

AntiWU=zeros(2,L); % Anti wind-up log for the surge controller: 0 -> ...
    No action, 1 -> integrator hold, -1 -> Saturation, but no action, ...
    [AntiWU_heading AntiWU_surge]'
for i=1:L
    if mod(i,100)==0
        dispstat(sprintf('Progress %02d%%',round((i/L)*100)));     % ...
            Simulation progress in %
    end
    time(i+1)=time(i)+dt;    % Update time vector

    %%%%%%%% Guidance system %%%%%%%%
    Chi_t=target(3,i);                                          % Target ...
        heading
    R_p=[cos(Chi_t) -sin(Chi_t) ; sin(Chi_t) cos(Chi_t)];    % ...
        Tansformation matrix from ned to path-fixed frame
    track(:,i)=[s_d e_d]'+R_p'*[x(4,i)-target(1,i) ; ...
        x(5,i)-target(2,i)];    % Tracking error

    Chi_r = atan(-track(2,i)/Δ_lookahead);                          % ...
        Lookahead LOS steering law
    Chi_d = Chi_r + target(3,i); ...
                                                % Desired course
    U_d = U_target-U_a_max*(track(1,i)/sqrt(track(1,i)^2+Δ_s^2));   % ...
        Desired ship speed

    %Mapping
    u=x(1,i);              % Surge speed
    v=x(2,i);              % Sway speed
    U=sqrt(u^2+v^2);       % Course speed
    beta=asin(v/U);        % Sideslip angle

    psi_r(i) = Chi_d-beta;       % Requested heading
    u_r(i) = sqrt(U_d^2-v^2);    % Requested surge speed

    %Prefilters
    if i==1 % First iteration, use initial state as psi_d_(i-1)
        psi_d(:,i)=(eye(3)+dt.*A_h)*psi_d_0+dt*B_h*psi_r(i);        % ...
            Heading prefilter
        u_d(:,i)=(eye(2)+dt.*A_s)*u_d_0+dt*B_s*u_r(i);              % ...
            Surge prefilter
    else
        psi_d(:,i)=(eye(3)+dt.*A_h)*psi_d(:,i-1)+dt*B_h*psi_r(i);   % ...
            Heading prefilter
        u_d(:,i)=(eye(2)+dt.*A_s)*u_d(:,i-1)+dt*B_s*u_r(i);         % ...
            Surge prefilter
    end

    %%%%%%%% Control algorithm %%%%%%%%
    e(:,i)=[x(6,i)-psi_d(1,i);x(3,i)-psi_d(2,i);x(1,i)-u_d(1,i)];   % ...
        Control Error
    if i==1 % First iteration, use zero as e_int(i-1)
```

```
        e_int(:,i)=dt*e(:,i);                % Integrate error
    else
        e_int(:,i)=e_int(:,i-1)+dt*e(:,i);   % Integrate error
    end

    %Heading controller
    Δ_ff=(1/K)*psi_d(2,i)+(T/K)*psi_d(3,i);      % Heading reference ...
        feed forward
    Δ_PID=-Kp*e(1,i)-Kd*e(2,i)-Ki*e_int(1,i);    % Heading PID controller

    %Surge controller
    n_c_ff=(1/Ksurge)*u_d(1,i)+(Tsurge/Ksurge)*u_d(2,i);        % ...
        Surge reference feed forward
    n_c_PI=-Kp_surge*e(3,i)-(Kp_surge/Ti_surge)*e_int(3,i);     % ...
        Surge PI controller

    %Integrator Anti Wind-up
    %Heading
    if abs(Δ_PID+Δ_ff)>Δ_max && i>1    % Rudder saturated, and not ...
        first iteration
        AntiWU(1,i)=-1;      % Log anti WU
        Δ_I=-Ki*e_int(1,i)-(-Ki*e_int(1,i-1));  % Integrator direction
        if sign(Δ_I)==sign(Δ_PID+Δ_ff)  % Integrator direction is the ...
            same as the propellor speed (Integrates further into ...
            saturation)
            e_int(1,i)=e_int(1,i-1);                 % Reset integrator ...
                to previous value
            Δ_PID=-Kp*e(1,i)-Kd*e(2,i)-Ki*e_int(1,i);    % Calculate ...
                heading PID controller output again
            AntiWU(1,i)=1;  % Log anti WU
        end
    end
    %Surge
    if abs(n_c_PI+n_c_ff)>n_c_max && i>1       % Shaft speed ...
        saturated, and not first iteration
        AntiWU(2,i)=-1;      % Log anti WU
        Δ_I=-(Kp_surge/Ti_surge)*e_int(3,i)-(-(Kp_surge/Ti_surge)...
            *e_int(3,i-1)); % Integrator direction
        if sign(Δ_I)==sign(n_c_PI+n_c_ff)    % Integrator direction is ...
            the same as the propellor speed (Integrates further into ...
            saturation)
            e_int(3,i)=e_int(3,i-1);             % Reset integrator to ...
                previous value
            n_c_PI=-Kp_surge*e(3,i)-(Kp_surge/Ti_surge)...
                *e_int(3,i);     % Calculate surge PI controller output ...
                    again
            AntiWU(2,i)=1;  % Log anti WU
        end
    end

    %%%%%%%% MS Fartoystyring %%%%%%%%
    tau(:,i)=[Δ_ff+Δ_PID;n_c_ff+n_c_PI];     % Define input
```

```
    if tau(2,i)<0      % Ensure non-negative propellor speed, ...
        msfartoystyring crashes if n_c < 0
        tau(2,i)=0;
        warning(['Desired propellor speed was negative at iteration ' ...
            num2str(i) '. Modified to zero.'])
    end
    x_dot=msfartoystyring(x(:,i),tau(:,i),0);   % Simulate ship with ...
        current

    %Integrate
    x(:,i+1)=x(:,i)+dt*x_dot;
    target(:,i+1) = target(:,i) + dt*U_target*[cos(target(3,i)) ...
        sin(target(3,i)) 0]';
end

pathplotter(x(4,:), x(5,:), x(6,:), dt, 50, 0, sim_time, 1, WP)
```