

MATLAB kode fra Optimalisering og regulering

0.1 Optimization - helicopter controllers

Optimal Control of Pitch/Travel without Feedback

```
1 % TTK4135 Optimization and Control - Helicopter lab
2 % Chapter 1, Optimal Control of Pitch/Travel without feedback
3
4 init;
5 dt = 0.25; % sampling time
6 sek_forst = 5;
7
8 % Continuous system model. x=[lambda r p p_dot]'
9 A = [ 0 1 0 0 ;
10      0 0 -K_2 0 ;
11      0 0 0 1 ;
12      0 0 -K_1*K_pp -K_1*K_pd ];
13 B = [0 0 0 K_1*K_pp]';
14
15 % Number of states and inputs
16 mx = size(A,2); % Number of states (number of ...
17      columns in A)
18 mu = size(B,2); % Number of inputs (number of ...
19      columns in B)
20
21 % Discrete system model
22 A1 = eye(mx) + dt*A;
23 B1 = dt*B;
24
25 % Initial values
26 x1_0 = pi ; % Lambda
27 x2_0 = 0; % r
28 x3_0 = 0; % p
29 x4_0 = 0; % p_dot
30 x0 = [x1_0 x2_0 x3_0 x4_0]'; % Initial values
```

```

29
30 % Time horizon and initialization
31
32 N = 100; % Time horizon for states
33 M = N; % Time horizon for inputs
34 z = zeros(N*mx+M*mu,1); % Initialize z for the whole ...
    horizon
35 z0 = z; % Initial value for optimization
36
37 % Bounds
38
39 ul = -30*pi/180; % Lower bound on control -- u1
40 uu = 30*pi/180; % Upper bound on control -- u1
41 xl = -Inf*ones(mx,1); % Lower bound on states (no bound)
42 xu = Inf*ones(mx,1); % Upper bound on states (no bound)
43 xl(3) = ul; % Lower bound on state x3
44 xu(3) = uu; % Upper bound on state x3
45
46 % Generate constraints on measurements and inputs
47
48 [vlb,vub] = genbegr2(N,M,xl,xu,ul,uu);
49 vlb(N*mx+M*mu) = 0; % We want the last input to be ...
    zero
50 vub(N*mx+M*mu) = 0; % We want the last input to be ...
    zero
51
52 % Generate the matrix Q and the vector c (objective function weights ...
    in the QP problem)
53 Q1 = zeros(mx,mx);
54 Q1(1,1) = 1; % Weight on state x1
55 P1 = 1; % Input weight(0.1, 1 and 10)
56 Q = genq2(Q1,P1,N,M,mu); % Generate Q
57
58 % Generate system matrixes for linear model
59 Aeq = gena2(A1,B1,N,mx,mu); % Generate A
60 beq = [A1*x0; zeros((N-1)*mx,1)]; % Generate b
61
62 % Solve Qp problem with linear model
63 tic
64 [z,lambda] = quadprog(Q,c,[],[],Aeq,beq,vlb,vub,z0);
65 t1=toc;
66
67 % Calculate objective value
68 phil = 0.0;
69 PhiOut = zeros(N*mx+M*mu,1);
70 for i=1:N*mx+M*mu
71     phil=phil+Q(i,i)*z(i)*z(i);
72     PhiOut(i) = phil;
73 end
74
75 % Extract control inputs and states
76 u = [z(N*mx+1:N*mx+M*mu); z(N*mx+M*mu)]; % Control input from solution
77

```

```

78 x1 = [x0(1);z(1:mx:N*mx)]; % State x1 from solution
79 x2 = [x0(2);z(2:mx:N*mx)]; % State x2 from solution
80 x3 = [x0(3);z(3:mx:N*mx)]; % State x3 from solution
81 x4 = [x0(4);z(4:mx:N*mx)]; % State x4 from solution
82
83 Antall = 5/dt;
84 Nuller = zeros(Antall,1);
85 Enere = ones(Antall,1);
86
87 u = [Nuller; u; Nuller];
88 x1 = [pi*Enere; x1; Nuller];
89 x2 = [Nuller; x2; Nuller];
90 x3 = [Nuller; x3; Nuller];
91 x4 = [Nuller; x4; Nuller];
92
93 %save bane
94 t = 0:dt:dt*(length(u)-1); % real time
95 pitch_setpoint=[t' u];
96
97 figure(2)
98 subplot(511)
99 stairs(t,u),grid
100 ylabel('u')
101 subplot(512)
102 plot(t,x1,'m',t,x1,'mo'),grid
103 ylabel('lambda')
104 subplot(513)
105 plot(t,x2,'m',t,x2', 'mo'),grid
106 ylabel('r')
107 subplot(514)
108 plot(t,x3,'m',t,x3,'mo'),grid
109 ylabel('p')
110 subplot(515)
111 plot(t,x4,'m',t,x4', 'mo'),grid
112 xlabel('tid (s)'),ylabel('pdot')

```

Optimal Control of Pitch/Travel with Feedback (LQ)

```

1 % TTK4135 Optimization and Control - Helicopter lab
2 %
3 % Chapter 2, Optimal Control of Pitch/Travel with Feedback (LQ)
4
5 init;
6 dt = 0.25; % sampling time
7 sek_forst = 5;
8
9 % Continuous model
10 A = [ 0 1 0 0 ;
11       0 0 -K_2 0 ;
12       0 0 0 1 ;
13       0 0 -K_1*K_pp -K_1*K_pd ];
14 B = [0 0 0 K_1*K_pp]';
15
16 % Number of states and inputs
17 mx = size(A,2); % Number of states (number of ...
18      columns in A)
19 mu = size(B,2); % Number of inputs (number of ...
20      columns in B)
21
22 % Discrete system model
23 A1 = eye(mx) + dt*A;
24 B1 = dt*B;
25
26 % Initial values
27 x1_0 = pi ; % Lambda
28 x2_0 = 0; % r
29 x3_0 = 0; % p
30 x4_0 = 0; % p_dot
31 x0 = [x1_0 x2_0 x3_0 x4_0]'; % Initial values
32
33 % Time horizon and initialization
34 N = 100; % Time horizon for states
35 M = N; % Time horizon for inputs
36 z = zeros(N*mx+M*mu,1); % Initialize z for the whole ...
37      horizon
38 z0 = z; % Initial value for optimization
39
40 % Bounds
41 ul = -30*pi/180; % Lower bound on control -- u1
42 uu = 30*pi/180; % Upper bound on control -- u1
43 xl = -Inf*ones(mx,1); % Lower bound on states (no bound)
44 xu = Inf*ones(mx,1); % Upper bound on states (no bound)
45 xl(3) = ul; % Lower bound on state x3
46 xu(3) = uu; % Upper bound on state x3
47
48 % Generate constraints on measurements and inputs
49
50 [v1b,vub] = genbegr2(N,M,xl,xu,ul,uu);

```

```

48 vlb(N*mx+M*mu) = 0; % We want the last input to be ...
    zero
49 vub(N*mx+M*mu) = 0; % We want the last input to be ...
    zero
50
51 % Generate the matrix Q and the vector c (objecitve function weights ...
    in the QP problem)
52
53 Q1 = zeros(mx,mx);
54 Q1(1,1) = 1; % Weight on state x1
55 P1 = 0.1; % Weight on input
56 Q = 2*genq2(Q1,P1,N,M,mu); % Generate Q
57
58 % LQR tuning
59 Q_lqr = zeros(mx,mx);
60 Q_lqr(1,1)=1;
61 Q_lqr(2,2)=0;
62 Q_lqr(3,3)=1;
63 Q_lqr(4,4)=0;
64 P_lqr=1;
65 [K,P,E] = dlqr(A1,B1,2*Q_lqr,2*P_lqr);
66
67 % Generate system matrixes for linear model
68 Aeq =gena2(A1,B1,N,mx,mu); % Generate A
69 beq = [A1*x0; zeros((N-1)*mx,1)]; % Generate b
70
71 % Solve Qp problem with linear model
72 tic
73 [z,lambda] = quadprog(Q,c,[],[],Aeq,beq,vlb,vub,z0);
74 t1=toc;
75
76
77 % Calculate objective value
78 phil = 0.0;
79 PhiOut = zeros(N*mx+M*mu,1);
80 for i=1:N*mx+M*mu
81     phil=phil+Q(i,i)*z(i)*z(i);
82     PhiOut(i) = phil;
83 end
84
85 % Extract control inputs and states
86 u = [z(N*mx+1:N*mx+M*mu); z(N*mx+M*mu)]; % Control input from solution
87
88 x1 = [x0(1); z(1:mx:N*mx)]; % State x1 from solution
89 x2 = [x0(2); z(2:mx:N*mx)]; % State x2 from solution
90 x3 = [x0(3); z(3:mx:N*mx)]; % State x3 from solution
91 x4 = [x0(4); z(4:mx:N*mx)]; % State x4 from solution
92
93 Antall = 5/dt;
94 Nuller = zeros(Antall,1);
95 Enere = ones(Antall,1);
96
97 u = [Nuller; u; Nuller];

```

```

98 x1 = [pi*Enere; x1; Nuller];
99 x2 = [Nuller; x2; Nuller];
100 x3 = [Nuller; x3; Nuller];
101 x4 = [Nuller; x4; Nuller];
102
103 x = [x1 x2 x3 x4]';
104
105 %saving trajectory
106 t = 0:dt:dt*(length(u)-1);           % real time
107 pitch_setpoint = [t' u];
108 x_optimal = [t' x'];

```

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

```

1  % TTK4135 Optimization and Control - Helicopter lab
2  %
3  % Chapter 3, Optimal Control of Pitch/Travel and Elevation with and ...
   without Feedback
4
5  init;
6  dt = 0.25;                                % sampling time
7  sek_forst = 5;
8
9  % Continuous model
10
11 A = [ 0      1      0      0      0      0 ...
        ;
12       0      0     -K_2      0      0      0 ...
        ;
13       0      0      0      1      0      0 ...
        ;
14       0      0     -K_1*K_pp  -K_1*K_pd  0      0 ...
        ;
15       0      0      0      0      0      1 ...
        ;
16       0      0      0      0      0     -K_3*K_ep  -K_3*K_ed] ...
        ;
17 B = [ 0      0      0      K_1*K_pp  0      0 ...
        ;
18       0      0      0      0      0      0     K_3*K_ep]' ...
        ;
19
20 % Number of states and inputs
21
22 mx = size(A,2);                            % Number of states (number of ...
        columns in A)
23 mu = size(B,2);                            % Number of inputs (number of ...
        columns in B)
24
25 % Discrete system model
26 A1 = eye(mx) + dt*A;
27 B1 = dt*B;
28
29 % Initial values
30 x1_0 = pi;                                % Lambda
31 x2_0 = 0;                                  % r
32 x3_0 = 0;                                  % p
33 x4_0 = 0;                                  % p_dot
34 x5_0 = 0;                                  % e
35 x6_0 = 0;                                  % e_dot
36 x0 = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0]'; % Initial values
37

```

```

38 % Time horizon and initialization
39
40 N = 55; % Time horizon for states
41 M = N; % Time horizon for inputs
42 z = zeros(N*mx+M*mu,1); % Initialize z for the whole ...
    horizon
43 z0 = z; % Initial value for optimization
44
45 % Bounds
46 ul = [-30*pi/180 -30*pi/180]'; % Lower bound on control -- u1
47 uu = [30*pi/180 30*pi/180]'; % Upper bound on control -- u1
48
49 xl = -Inf*ones(mx,1); % Lower bound on states (no bound)
50 xu = Inf*ones(mx,1); % Upper bound on states (no bound)
51 xl(3) = ul(1); % Lower bound on state x3
52 xu(3) = uu(1); % Upper bound on state x3
53
54 % Generate constraints on measurements and inputs
55
56 [vlb,vub] = genBegr2(N,M,xl,xu,ul,uu);
57 vlb(N*mx+M*mu) = 0; % We want the last input to be ...
    zero
58 vub(N*mx+M*mu) = 0; % We want the last input to be ...
    zero
59
60 % Generate the matrix Q and the vector c (objective function weights ...
    in the QP problem)
61
62 Q1 = zeros(mx,mx);
63 Q1(1,1) = 10; % Weight on state x1
64 P1 = 0.1*eye(2); % Weight on input
65 Q = 2*genq2(Q1,P1,N,M,mu); % Generate Q
66 FUN= @(z) z'*Q*z;
67
68 % LQR
69 Q_lqr = zeros(mx,mx);
70 Q_lqr(1,1)=1;
71 Q_lqr(2,2)=0;
72 Q_lqr(3,3)=1;
73 Q_lqr(4,4)=0;
74 Q_lqr(5,5)=1;
75 Q_lqr(6,6)=1;
76 r_lqr = 1*eye(2);
77 [K,P,E] = dlqr(A1,B1,2*Q_lqr,2*r_lqr);
78
79 % Generate system matrixes for linear model
80 Aeq =gena2(A1,B1,N,mx,mu); % Generate A
81 beq = [A1*x0; zeros((N-1)*mx,1)]; % Generate b
82
83 % Solve Qp problem with linear model
84 options = optimset('disp','iter','Algorithm','active-set');
85 tic
86 [z,lambda] = fmincon(FUN,z0,[],[],Aeq,beq,vlb,vub,@el,options);

```



```

87
88 t1=toc;
89
90 % Calculate objective value
91 phil = 0.0;
92 PhiOut = zeros(N*mx+M*mu,1);
93 for i=1:N*mx+M*mu
94     phil=phil+Q(i,i)*z(i)*z(i);
95     PhiOut(i) = phil;
96 end
97
98 % Extract control inputs and states
99
100 u1 = [z(mx*N+1:mu:end);z(N*mx+M*mu)] ; % Control input from solution
101 u2 = [z(mx*N+mu:mu:end);z(N*mx+M*mu)] ;
102
103 x1 = [x0(1);z(1:mx:N*mx)] ; % State x1 from solution
104 x2 = [x0(2);z(2:mx:N*mx)] ; % State x2 from solution
105 x3 = [x0(3);z(3:mx:N*mx)] ; % State x3 from solution
106 x4 = [x0(4);z(4:mx:N*mx)] ; % State x4 from solution
107 x5 = [x0(5);z(5:mx:N*mx)] ; % State x5 from solution
108 x6 = [x0(6);z(6:mx:N*mx)] ; % State x6 from solution
109
110 Antall = 5/dt;
111 Nuller = zeros(Antall,1);
112 Enere = ones(Antall,1);
113
114 u1 = [Nuller; u1; Nuller];
115 u2 = [Nuller; u2; Nuller];
116 x1 = [pi*Enere; x1; Nuller];
117 x2 = [Nuller; x2; Nuller];
118 x3 = [Nuller; x3; Nuller];
119 x4 = [Nuller; x4; Nuller];
120 x5 = [Nuller; x5; Nuller];
121 x6 = [Nuller; x6; Nuller];
122
123 x = [x1 x2 x3 x4 x5 x6]';
124
125 %saving trajectory
126 t = 0:dt:dt*(length(u1)-1); % real time
127 setpoint = [t' u1 u2];
128 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

```
1
2 function [z,A]=jaccsd(fun,x,u,t)
3 % JACCSd Jacobian through complex step differentiation
4 % [z J] = jaccsd(@f,x, u, t)
5 % z = f(x, u, t)
6 % J = df/dx(x, u, t)
7 %
8 z=fun(x,u,t);
9 n=numel(x);
10 m=numel(z);
11 A=zeros(m,n);
12 h=n*eps;
13 for k=1:n
14     x1=x;
15     x1(k)=x1(k)+h*1i; %for bedre speed la jeg til 1-tall foran i.
16     A(:,k)=imag(fun(x1,u,t))/h;
17 end
```

```
1 clear all;
2
3 h=0.01; %samplingtime
4 N=1000; %iterations
5 sigma=0.001; %sigma^2
6
7 %initial values
8 x=[0 0 0 0 0 0 0 0 0]';
9 xd=[0.1 0.3 0.2 0 0 0]'; %Desired Euler Angles
10 I=diag([0.0108 0.0113 0.0048]); %inertia matrix
11 b_g=[0.012;0.017;0.014]; %The constant Bias
12 x(7:9) = b_g; %adding bias to the states
```

```

13
14 %Initialization Controller from Task 2.1
15 Kp=diag([0.07 0.07 0.07]);
16 Kd=diag([0.1 0.1 0.1]);
17 e=[0;0;0];
18
19 %Initialization of Kalman filter
20 y=[0;0;0;0;0;0];
21 x_bar=[0 0 0 0 0 0 0 0 0]';
22 p_bar= (x-x_bar)*(x-x_bar)' + 1e-9*eye(9); %Ensure pos def.
23 I_k=eye(9);
24 x_hat=[0 0 0 0 0 0 0 0 0]';
25 R=[1e-09*eye(3) zeros(3); zeros(3) sigma*eye(3)]; %6x6
26 H=[eye(3) zeros(3) zeros(3); zeros(3) eye(3) eye(3)]; %6x9
27 Q=10^-9*eye(9); %9x9 With small diag to avoid singularities and ensure ...
    keep pos def.
28
29 %Function for the system 9x1
30 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
31
32 %Used for data storing and plot
33 true=zeros(9,N);
34 est=zeros(9,N);
35 time=zeros(1,N);
36 %Loop
37 for i=1:N+1,
38     t = (i-1)*h;
39
40     %noise
41     noise(:,i)=sqrt(0.001).*randn(3,1); %Computing noise
42
43
44 %f(x_hat,tau_hat)+EW 9x1
45 tau=-(Kp*e+Kd*x_hat(4:6)); %input with e=(x_hat_euler)-(x_d_euler)
46
47 %measurement
48 v = [zeros(3,1);
49     noise(:,i)]; %Noise
50 y = H*x + v;
51
52
53 %%Kalman gain
54 K=p_bar*H'*inv(H*p_bar*H'+R);
55
56 %State estimate update
57 x_hat=x_bar+K*(y-H*x_bar); %xbar+K(y-ybar). Ybar contains zero bias ...
    at the first iteration.
58
59 P_hat=(I_k-K*H)*p_bar*(I_k-K*H)'+K*R*K';
60

```

```

61
62     %linearization of f(x_hat,tau_hat) inserting x_hat for [phi theta . ...
        . .], df/dx
63     [fev, df_xhat] = jaccsd(f,x_hat, tau, t);
64
65     %State estimate propagation and error covariance propagation ...
        (predictor k+1)
66     x_bar=x_hat+h*fev; %x_hat + h*x_dothat
67
68     %discrete-time matrix PHI
69     PHI=I_k+h*df_xhat; %(11.52)
70     p_bar=PHI*P_hat*PHI'+Q; %with no process noise, we do not include ...
        gamma.
71
72     %From Task 2.1
73     e=(x_hat(1:3)-xd(1:3)); %error
74
75
76     %Euler integration(k+1)
77     x=x+h*f(x,tau,t);
78     xest=x_hat;
79
80
81
82     %data storing
83     time(i)=i*h; %to update time table
84     true(:,i)=x;
85     est(:,i)=xest;
86     yvector(:,i)= y; %used only for checking if measurement y is ...
        correct by plotting it
87
88 end
89
90
91
92 %plotting actual and estimated angular rates
93
94
95 %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.

```
1
2 % TTK4190 - Guidance and Control
3 % Assignment 3
4 % 746072 - Bjorn-Olav H. Eiksen
5 % 746029 - Marius Hjertaker
6 % 705009 - Sveinung Ohrem
7 % 745621 - Ole Maurice Rabanal
8 %
9 % This is the code for task 1.4 of the assignment
10
11 clear; clc;
12 dispstat('','init') % Simulation progress. ...
    Initialization. Does not print anything.
13 dispstat('run_task_1_4','keepthis'); % Output first
14
15 %Integration values
16 sim_time=3000; %Simulation end time
17 dt=0.1; %Steplength
18 L=sim_time/dt; %Simulation steps
19
20 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% System ...
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
21
22 K=-0.0598; %Transfer function gain
23 T=115.5667; %Transfer function time constant
24
25 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% User parameters ...
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
26
27 x_0=[6.63,0,0,0,0,0]'; %x = [ u v r x y psi]'
28 omega_b=0.06; %Bandwith of controller
29 zeta=1; %Damping coefficient
30
31 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% References ...
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
32
33 t=0:dt:(sim_time); %Time steps
34 psi_d=-0.3*sin(0.008*t); %Desired heading(yaw)
35 r_d=-0.3*0.008*cos(0.008*t); %Desired yaw rate
36 r_d_dot=0.3*0.008*0.008*sin(0.008*t); %Desired yaw accceleration
37
38
39 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Controller parameters ...
```

```

40      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
41      omega_n=omega_b/(sqrt(1-2*zeta^2+...
42          sqrt(4*zeta^4-4*zeta^2+2)));          %Natural frequency
43
44      Kp=(T/K)*omega_n^2;                      %Proportional gain
45      Kd=(T/K)*2*zeta*omega_n-1/T;             %Derivative gain
46      Ki=(omega_n/10)*Kp;                      %Integrator gain
47
48      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%      Data storage      ...
49      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
50      time=zeros(1,L+1);                      %Time
51      x=zeros(6,L+1);                         %States x=[ u v r x y psi]'
52      tau=zeros(2,L);                         %Inputs to ms fartoystyring tau=[Δ_c n_c]'
53      e=zeros(2,L);                          %Error
54
55      x(:,1)=x_0;                             %Set first step equal to initial values
56      e_int=zeros(1,L+1);                     %Error integrated
57
58      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%      Control loop      ...
59      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
60      for i=1:L
61          if (mod(i,100))==0
62              dispstat(sprintf('Progress %02d%%',round((i/L)*100)));      % ...
63              %Simulation progress in %
64          end
65          time(i+1)=time(i)+dt;                %Time
66
67          %Control algorithm
68          e(:,i)=[x(6,i)-psi_d(i);x(3,i)-r_d(i)];      %Error
69          e_int(i+1)=e_int(i)+dt*e(1,i);              %Error integrated
70
71          Δ_ff=(1/K)*r_d(i)+(T/K)*r_d_dot(i);          %Feed forward input
72          Δ_PID=-Kp*e(1,i)-Kd*e(2,i)-Ki*e_int(i+1);    %PID control input
73
74          tau(:,i)=[Δ_ff+Δ_PID ; 7.3];                %Input with constant shaft ...
75          %velocity
76          x_dot=msfartoystyring(x(:,i),tau(:,i),1);    %Send to ms ...
77          %fartoystyring
78
79          %Integrate
80          x(:,i+1)=x(:,i)+dt*x_dot;                  %Update states
81      end
82
83      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%      Plot figures      ...
84      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

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

```

```

4 % 746029 - Marius Hjertaker
5 % 705009 - Sveinung Ohrem
6 % 745621 - Ole Maurice Rabanal
7 %
8 % This is the code for task 1.8 of the assignment
9
10 clear; clc;
11 close all
12 dispstat('','init') % Simulation progress. ...
    Initialization. Does not print anything.
13 dispstat('run_task_1_8','keepthis'); % Output first
14
15 %Integration values
16 sim_time=10000; %Simulation end time
17 dt=1; %Steplength
18 L=sim_time/dt; %Simulation steps
19
20 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% System ...
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
21 %Heading
22 K=-0.0598;
23 T=115.5667;
24 %Surge
25 Ksurge=0.9820;
26 Tsurge=450.8367;
27 time=0:dt:sim_time;
28
29 %User parameters
30 x_0=[4,0,0,0,0,0]'; %x = [ u v r x y psi]'
31 omega_b=0.06; %Bandwith for heading controller
32 omega_bs=0.015; %Bandwith for surge controller
33 zeta=1; %Damping of system
34 n_c_max=85*2*pi/60; %Max shaft velocity[rad/s]
35
36 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% References ...
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
37
38 %%Choose these for time varying reference in yaw
39 % psi_d=-0.3*sin(0.008*time(1:L));
40 % r_d=-0.3*0.008*cos(0.008*time(1:L));
41 % r_d_dot=0.3*0.008*0.008*sin(0.008*time(1:L));
42 %
43 % psi_d=[psi_d(1:(L/2)) ones(1,L/2).*psi_d(end)];
44 % r_d=[r_d(1:(L/2)) zeros(1,L/2)];
45 % r_d_dot=[r_d_dot(1:(L/2)) zeros(1,L/2)];
46
47 %%Choose these for constant yaw and yaw rate:
48 psi_d=zeros(1,L);
49 r_d=zeros(1,L);
50 r_d_dot=zeros(1,L);
51
52 %%Choose between step input or time varying input for u:
53 u_r=[4.*ones(1,700/dt) ...

```

```

54     7.*ones(1,L-700/dt)]; %Step
55 %u_r=4+abs(sin(0.001*time(1:L))); %Time varying
56
57 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Controller parameters ...
58 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
59 %Heading controller
60 omega_n=omega_b/(sqrt(1-2*zeta^2+... %Natural frequency
61     sqrt(4*zeta^4-4*zeta^2+2))); %Kp
62 Kp=(T/K)*omega_n^2; %Kd
63 Kd=(T/K)*2*zeta*omega_n; %Ki
64 Ki=(T/K)*((omega_n^3)/10);
65
66 %Surge controller
67 omega_n_surge=omega_bs/(sqrt(1-2*... %Natural frequency
68     zeta^2+sqrt(4*zeta^4-4*zeta^2+2))); %Kp
69 Kp_surge=5.7388; %Ti
70 Ti_surge=Tsurg+50;
71 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Data storage ...
72 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
73 x=zeros(6,L+1); %x = [ u v r x y psi]'
74 tau=zeros(2,L); %tau=[Δ_c n_c]'
75 e=zeros(3,L); %e=[psi,r,u]'
76 e_int=zeros(3,L+1); %Integrated error
77
78 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Reference prefilter ...
79 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
80 omega_ref=0.05; %Natural frequency for ...
81 reference model
82 zeta_ref=1; %Damping in prefilter
83 prefilter=ss([0 1;-omega_ref^2 ... %Making state space for prefilter
84     -2*zeta_ref*omega_ref],[0;...
85     omega_ref^2],eye(2),zeros(2,1));
86 [ref,-]=lsim(prefilter,u_r,time... %Simulate to get reference model
87     (1:end-1),[u_r(1);0]); %Filtered desired value
88 u_d=ref(:,1)'; %Derivative of filtered ...
89 u_d_dot=ref(:,2)';
90 desired value
91 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
92 x(:,1)=x_0; %Set x_0
93 for i=1:L
94     if (mod(i,100))==0
95         dispstat(sprintf('Progress %02d%%',round((i/L)*100))); % ...
96         Simulation progress in %
97     end
98
99 %Control algorithm
100 e(:,i)=[x(6,i)-psi_d(i);... %Error
101     x(3,i)-r_d(i);x(1,i)-u_d(i)]; %Error integrated
102 e_int(:,i+1)=e_int(:,i)+dt*e(:,i);

```



```

101 %Heading
102 Δ_ff=(1/K)*r_d(i)+...
103     (T/K)*r_d_dot(i); %Reference feed forward for ...
104     heading
105 Δ_PID=-Kp*e(1,i)-Kd*...
106     e(2,i)-Ki*e_int(1,i+1); %PID controller for heading
107
108 %Surge
109 n_c_ff=(1/Ksurge)*u_d(i)+...
110     (Tsurge/Ksurge)*u_d_dot(i); %Reference feed forward for surge
111 n_c=-Kp_surge*e(3,i)-(Kp_surge...
112     /Ti_surge)*e_int(3,i+1); %PI controller for surge
113
114 %Anti Windup
115 if abs(n_c+n_c_ff)>n_c_max
116     Δ_I=-Ki*e_int(3,i+1)-(-Ki*e_int(3,i));
117     if sign(Δ_I)≠sign(n_c+n_c_ff)
118         e_int(3,i+1)=e_int(3,i); %Locks the integrator ...
119         if the
120             n_c=-Kp_surge*e(3,i)-... %Shaft speed reaches ...
121             max value
122             (Kp_surge/Ti_surge)...
123             *e_int(3,i+1);
124     end
125 end
126
127 %Set tau and send it to MS Fartoystyring
128 tau(:,i)=[Δ_ff+Δ_PID... %Sets tau
129     ;n_c_ff+n_c];
130 x_dot=msfartoystyring(x(:,i),... %Send to MS Fartoystyring
131     tau(:,i),1);
132
133 %Integrate
134 x(:,i+1)=x(:,i)+dt*x_dot; %Integrates with forward Euler
135 end

```

```

1 % TTK4190 - Guidance and Control
2 % Assignment 3
3 % 746072 - Bjorn-Olav H. Eiksen
4 % 746029 - Marius Hjertaker
5 % 705009 - Sveinung Ohrem
6 % 745621 - Ole Maurice Rabanal
7 %
8 % This is the code for task 2.3 of the assignment
9
10 clear; clc;
11 dispstat('','init') % ...
12     Simulation progress. Initialization. Does not print anything.
13 dispstat('run_task_2_3. Path-tracking, with direct assignment',...
14     'keepthis'); % Output first
15 no=menu('Choose steering method','Enclosure-Based Steering',...

```

```

15     'Lookahead-Based Steering','Exit'); % ...
        Steering method menu
16 no2=menu('Choose WP changing method','Circle of acceptance',...
17     'Circle of acceptance ignoring cross-track error','Exit'); % WP ...
        changing menu
18 if no==3 || no2==3 % Exit?
19     break
20 end
21 %Integration values
22 sim_time=2500; %Simulation end time
23 dt=1; %Steplength
24 L=sim_time/dt; %Simulation steps
25
26 %System
27 K=-0.0598; % Heading model gain
28 T=115.5667; % Heading model time constant
29 Ksurge=0.9820; % Surge model gain
30 Tsurge=450.8367; % Surge model time constant
31
32 %User parameters
33 x_0=[6.63,0,0,0,0,0]'; % Initial condition, x = [ u v r x y psi]'
34 n_c_max=85*2*pi/60; % Max shaft velocity [rad/s]
35 Δ_max = 25*pi/180; % max rudder angle [rad]
36
37 %Controller parameters %%%%%%%%%%
38 %Heading controller
39 zeta=1; % Damping coefficient
40 omega_b=0.03; % Bandwidth frequency
41 omega_n=omega_b/(sqrt(1-2*zeta^2+sqrt(4*zeta^4-4*zeta^2+2))); % ...
    Natural frequency
42 Kp=(T/K)*omega_n^2; % Heading controller proportional gain
43 Kd=(T/K)*2*zeta*omega_n-1/T; % Heading controller derivative gain
44 Ki=(omega_n/10)*Kp; % Heading controller integral gain
45
46 %Surge controller
47 Kp_surge=2.7388; % Surge controller proportional gain
48 Ti_surge=Tsurge+50; % Surge controller integral time
49
50 %%%%%%%%%% Data storage %%%%%%%%%%
51 time=zeros(1,L+1); % Time vector
52 x=zeros(6,L+1); % State vector, x = [ u v r x y psi]'
53 tau=zeros(2,L); % Input vector, tau = [Δ n_c]'
54 e=zeros(3,L); % Control error vector, e = [psi_r r u]'
55 e_int=zeros(3,L); % Integrated control error e_int = [psi_r_int ...
    r_int u_int]'
56 LOS=zeros(2,L); % LOS point for Enclosed-Based Steering, LOS = ...
    [xlos ylos]'
57 psi_r=zeros(1,L); % Requested heading
58 psi_d=zeros(3,L); % Filtered heading references psi_d = [psi_d ...
    psi_d_dot psi_d_ddot]'
59 u_r=zeros(1,L); % Requested surge speed
60 u_d=zeros(2,L); % Filtered surge references u_d = [u_d u_d_dot]'
61

```

```

62 %%%%%%%%%%% Prefilters %%%%%%%%%%%
63 %Heading
64 omega_ref=0.15; % Natural frequency for reference model
65 zeta_ref=1; % Damping coefficient
66 A_h=[0 1 0;0 0 1;-omega_ref^3 -(2*zeta_ref+1)*omega_ref^2 ...
67 -(2*zeta_ref+1)*omega_ref]; % Continuous A matrix
68 B_h=[0;0;omega_ref^3]; % Continuous B matrix
69 psi_d_0 = [x_0(6) x_0(3) 0]'; % Initial state of filtered heading ...
    reference
70
71 %Surge
72 omega_ref=0.05; % Natural frequency for reference model
73 zeta_ref=1; % Damping coefficient
74 A_s=[0 1;-omega_ref^2 -2*zeta_ref*omega_ref]; % Continuous A matrix
75 B_s=[0;omega_ref^2]; % Continuous B matrix
76 u_d_0 = [x_0(1) 0]'; % Initial state of filtered surge references
77
78
79 x(:,1)=x_0; % Set initial state
80 load('WP.mat'); % Load waypoints
81 k = 2; % Next waypoint selector
82 R_LOS=4*305; % LOS radius for Enclosed-Based steering
83 Δ_lookahead=3*305; % Lookahead distance for Lookahead-Based Steering
84 R = 2*305; % Acceptance radius
85
86
87 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]'
88 CurrentWP=zeros(1,L); % Current WP log
89 for i=1:L
90     if (mod(i,100))==0
91         dispstat(sprintf('Progress %02d%%',round((i/L)*100))); % ...
            Simulation progress in %
92     end
93     time(i+1)=time(i)+dt; % Update time vector
94
95     %%%%%%%%% Waypoint selector %%%%%%%%%
96     if k < size(WP,2) % More waypoints available?
97         if no2==1 % Circle of acceptance selected
98             if ((WP(1,k)-x(4,i))^2 + (WP(2,k)-x(5,i))^2) ≤ R^2
99                 k=k+1; % Select next waypoint
100             end
101         elseif no2==2 % Circle of acceptance ignoring cross-track ...
            error selected
102             if NextWP(WP(:,k-1:k),x(4:5,i),R)
103                 k=k+1; % Select next waypoint
104             end
105         else
106             error('Invalid choice for WP changing method')
107         end
108     end
109     CurrentWP(i)=k; % Log current waypoint

```

```

110
111 %%%%%%%%% Guidance system %%%%%%%%%
112 if no==1 % Enclosed-Based steering
113     LOS(:,i) = LOSp(WP(:,k-1:k),x(4:5,i),R_LOS); % ...
114     Calculate LOS point
115     Chi_d = atan2((LOS(2,i) - x(5,i)), (LOS(1,i) - x(4,i))); % ...
116     Calculate desired course
117 elseif no==2 % Lookahead-Based steering
118     Chi_d = Lookahead(WP(:,k-1:k),x(4:5,i),Δ_lookahead); % ...
119     Calculate desired course
120 else
121     error('Invalid choice for steering method')
122 end
123 U_d = 6.63; % Desired craft speed, constant
124
125 %Mapping
126 psi_r(i) = Chi_d; % Requested heading
127 u_r(i) = U_d; % Requested surge speed
128
129 %Prefilters
130 if i==1 % First iteration, use initial state as psi_d(i-1)
131     psi_d(:,i)=(eye(3)+dt.*A_h)*psi_d_0+dt*B_h*psi_r(i); % ...
132     Heading prefilter
133     u_d(:,i)=(eye(2)+dt.*A_s)*u_d_0+dt*B_s*u_r(i); % ...
134     Surge prefilter
135 else
136     psi_d(:,i)=(eye(3)+dt.*A_h)*psi_d(:,i-1)+dt*B_h*psi_r(i); % ...
137     Heading prefilter
138     u_d(:,i)=(eye(2)+dt.*A_s)*u_d(:,i-1)+dt*B_s*u_r(i); % ...
139     Surge prefilter
140 end
141
142 %%%%%%%%% Control algorithm %%%%%%%%%
143 e(:,i)=[x(6,i)-psi_d(1,i);x(3,i)-psi_d(2,i);x(1,i)-u_d(1,i)]; % ...
144 Control Error
145 if i==1 % First iteration, use zero as e_int(i-1)
146     e_int(:,i)=dt*e(:,i); % Integrate error
147 else
148     e_int(:,i)=e_int(:,i-1)+dt*e(:,i); % Integrate error
149 end
150
151 %Heading controller
152 Δ_ff=(1/K)*psi_d(2,i)+(T/K)*psi_d(3,i); % Heading reference ...
153 feed forward
154 Δ_PID=-Kp*e(1,i)-Kd*e(2,i)-Ki*e_int(1,i); % Heading PID controller
155
156 %Surge controller
157 n_c_ff=(1/Ksurge)*u_d(1,i)+(Tsurge/Ksurge)*u_d(2,i); % ...
158 Surge reference feed forward
159 n_c_PI=-Kp_surge*e(3,i)-(Kp_surge/Ti_surge)*e_int(3,i); % ...
160 Surge PI controller
161
162 %Integrator Anti Wind-up

```

```

152 %Heading
153 if abs(Δ_PID+Δ_ff)>Δ_max && i>1 % Rudder saturated, and not ...
    first iteration
154 AntiWU(1,i)=-1; % Log anti WU
155 Δ_I=-Ki*e_int(1,i)-(-Ki*e_int(1,i-1)); % Integrator direction
156 if sign(Δ_I)==sign(Δ_PID+Δ_ff) % Integrator direction is the ...
    same as the propellor speed (Integrates further into ...
    saturation)
157 e_int(1,i)=e_int(1,i-1); % Reset integrator ...
    to previous value
158 Δ_PID=-Kp*e(1,i)-Kd*e(2,i)-Ki*e_int(1,i); % Calculate ...
    heading PID controller output again
159 AntiWU(1,i)=1; % Log anti WU
160 end
161 end
162 %Surge
163 if abs(n_c_PI+n_c_ff)>n_c_max && i>1 % Shaft speed saturated, ...
    and not first iteration
164 AntiWU(2,i)=-1; % Log anti WU
165 Δ_I=-(Kp_surge/Ti_surge)*e_int(3,i)-(-Kp_surge/Ti_surge)...
    *e_int(3,i-1); % Integrator direction
166 if sign(Δ_I)==sign(n_c_PI+n_c_ff) % Integrator direction is ...
    the same as the propellor speed (Integrates further into ...
    saturation)
167 e_int(3,i)=e_int(3,i-1); % Reset integrator to ...
    previous value
168 n_c_PI=-Kp_surge*e(3,i)-(Kp_surge/Ti_surge)...
    *e_int(3,i); % Calculate surge PI controller output ...
    again
169 AntiWU(2,i)=1; % Log anti WU
170 end
171 end
172 end
173 end
174
175 %%%%%%%%% MS Fartoystyring %%%%%%%%%
176 tau(:,i)=[Δ_ff+Δ_PID;n_c_ff+n_c_PI]; % Define input
177 if tau(2,i)<0 % Ensure non-negative propellor speed, ...
    msfartoystyring crashes if n_c < 0
178 tau(2,i)=0;
179 warning(['Desired propellor speed was negative at iteration ' ...
    num2str(i) '. Modified to zero.'])
180 end
181 x_dot=msfartoystyring(x(:,i),tau(:,i),1); % Simulate ship with ...
    current
182
183 %Integrate
184 x(:,i+1)=x(:,i)+dt*x_dot;
185 end
186
187 pathplotter(x(4,:), x(5,:), x(6,:), dt, 50, 0, sim_time, 0, WP)
188
189 %%%%%%%%% Optional plots %%%%%%%%%

```

```

1 % TTK4190 - Guidance and Control
2 % Assignment 3
3 % 746072 - Bjorn-Olav H. Eiksen
4 % 746029 - Marius Hjertaker
5 % 705009 - Sveinung Ohrem
6 % 745621 - Ole Maurice Rabanal
7 %
8 % This is the code for task 2.6 of the assignment
9
10 clear; clc;
11 dispstat('', 'init') % ...
    Simulation progress. Initialization. Does not print anything.
12 dispstat('run_task_2_6. Path-tracking, with transformed assignment', ...
13     'keepthis'); % Output first
14 no=menu('Choose steering method', 'Enclosure-Based Steering', ...
15     'Lookahead-Based Steering', 'Exit'); % ...
    Steering method menu
16 no2=menu('Choose WP changing method', 'Circle of acceptance', ...
17     'Circle of acceptance ignoring cross-track error', 'Exit'); % WP ...
    changing menu
18 if no==3 || no2==3 % Exit?
19     break
20 end
21 %Integration values
22 sim_time=2700; %Simulation end time
23 dt=1; %Steplength
24 L=sim_time/dt; %Simulation steps
25
26 %System
27 K=-0.0598; % Heading model gain
28 T=115.5667; % Heading model time constant
29 Ksurge=0.9820; % Surge model gain
30 Tsurge=450.8367; % Surge model time constant
31
32 %User parameters
33 x_0=[6.63,0,0,0,0,0]'; % Initial condition, x = [ u v r x y psi]'
34 n_c_max=85*2*pi/60; % Max shaft velocity [rad/s]
35 Δ_max = 25*pi/180; % max rudder angle [rad]
36
37 %Controller parameters %%%%%%%%%%
38 %Heading controller
39 zeta=1; % Damping coefficient
40 omega_b=0.03; % Bandwidth frequency
41 omega_n=omega_b/(sqrt(1-2*zeta^2+sqrt(4*zeta^4-4*zeta^2+2))); % ...
    Natural frequency
42 Kp=(T/K)*omega_n^2; % Heading controller proportional gain
43 Kd=(T/K)*2*zeta*omega_n-1/T; % Heading controller derivative gain
44 Ki=(omega_n/10)*Kp; % Heading controller integral gain
45
46 %Surge controller
47 Kp_surge=2.7388; % Surge controller proportional gain
48 Ti_surge=Tsurge+50; % Surge controller integral time
49

```

```

50 %%%%%%%%%%%%% Data storage %%%%%%%%%%%%%
51 time=zeros(1,L+1); % Time vector
52 x=zeros(6,L+1); % State vector, x = [ u v r x y psi]'
53 tau=zeros(2,L); % Input vector, tau = [ $\Delta$  n_c]'
54 e=zeros(3,L); % Control error vector, e = [psi_r r u]'
55 e_int=zeros(3,L); % Integrated control error e_int = [psi_int ...
    r_int u_int]'
56 LOS=zeros(2,L); % LOS point for Enclosed-Based Steering, LOS = ...
    [xlos ylos]'
57 psi_r=zeros(1,L); % Requested heading
58 psi_d=zeros(3,L); % Filtered heading references psi_d = [psi_d ...
    psi_d_dot psi_d_ddot]'
59 u_r=zeros(1,L); % Requested surge speed
60 u_d=zeros(2,L); % Filtered surge references u_d = [u_d u_d_dot]'
61
62 %%%%%%%%%%%%% Prefilters %%%%%%%%%%%%%
63 %Heading
64 omega_ref=0.15; % Natural frequency for reference model
65 zeta_ref=1; % Damping coefficient
66 A_h=[0 1 0; 0 0 1; -omega_ref^3 -(2*zeta_ref+1)*omega_ref^2 ...
    -(2*zeta_ref+1)*omega_ref]; % Continuous A matrix
67 B_h=[0; 0; omega_ref^3]; % Continuous B matrix
68 psi_d_0 = [x_0(6) x_0(3) 0]'; % Initial state of filtered heading ...
    reference
69
70
71 %Surge
72 omega_ref=0.05; % Natural frequency for reference model
73 zeta_ref=1; % Damping coefficient
74 A_s=[0 1; -omega_ref^2 -2*zeta_ref*omega_ref]; % Continuous A matrix
75 B_s=[0; omega_ref^2]; % Continuous B matrix
76 u_d_0 = [x_0(1) 0]'; % Initial state of filtered surge references
77
78
79 x(:,1)=x_0; % Set initial state
80 load('WP.mat'); % Load waypoints
81 k = 2; % Next waypoint selector
82 R_LOS=4*305; % LOS radius for Enclosed-Based steering
83  $\Delta$ _lookahead=3*305; % Lookahead distance for Lookahead-Based Steering
84 R = 2*305; % Acceptance radius
85
86
87 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]'
88 CurrentWP=zeros(1,L); % Current WP log
89 for i=1:L
90     if (mod(i,100))==0
91         dispstat(sprintf('Progress %02d%%',round((i/L)*100))); % ...
            Simulation progress in %
92     end
93     time(i+1)=time(i)+dt; % Update time vector
94
95 %%%%%%%%%%%%% Waypoint selector %%%%%%%%%%%%%

```

```

96     if k < size(WP,2)      % More waypoints available?
97         if no2==1         % Circle of acceptance selected
98             if ((WP(1,k)-x(4,i))^2 + (WP(2,k)-x(5,i))^2) ≤ R^2)
99                 k=k+1;    % Select next waypoint
100             end
101         elseif no2==2      % Circle of acceptance ignoring cross-track ...
102             error selected
103             if NextWP(WP(:,k-1:k),x(4:5,i),R)
104                 k=k+1;    % Select next waypoint
105             end
106         else
107             error('Invalid choice for WP changing method')
108         end
109     CurrentWP(i)=k; % Log current waypoint
110
111     %%%%%%%%% Guidance system %%%%%%%%%
112     if no==1              % Enclosure-Based steering
113         LOS(:,i) = LOSp(WP(:,k-1:k),x(4:5,i),R_LOS);           % ...
114         % Calculate LOS point
115         Chi_d = atan2((LOS(2,i) - x(5,i)), (LOS(1,i) - x(4,i))); % ...
116         % Calculate desired course
117     elseif no==2          % Lookahead-Based steering
118         Chi_d = Lookahead(WP(:,k-1:k),x(4:5,i),Δ_lookahead);   % ...
119         % Calculate desired course
120     else
121         error('Invalid choice for steering method')
122     end
123     U_d = 6.63;           % Desired craft speed, constant
124
125     %Mapping
126     u=x(1,i);
127     v=x(2,i);
128     U=sqrt(u^2+v^2);
129     beta=asin(v/U);
130     psi_r(i) = Chi_d-beta; % Requested heading
131
132     u_r(i) = sqrt(U_d^2-v^2); % Requested surge speed
133
134     %Prefilters
135     if i==1 % First iteration, use initial state as psi_d(i-1)
136         psi_d(:,i)=(eye(3)+dt.*A_h)*psi_d_0+dt*B_h*psi_r(i); % ...
137         % Heading prefilter
138         u_d(:,i)=(eye(2)+dt.*A_s)*u_d_0+dt*B_s*u_r(i); % ...
139         % Surge prefilter
140     else
141         psi_d(:,i)=(eye(3)+dt.*A_h)*psi_d(:,i-1)+dt*B_h*psi_r(i); % ...
142         % Heading prefilter
143         u_d(:,i)=(eye(2)+dt.*A_s)*u_d(:,i-1)+dt*B_s*u_r(i); % ...
144         % Surge prefilter
145     end
146
147     %%%%%%%%% Control algorithm %%%%%%%%%

```



```

141 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
142 if i==1 % First iteration, use zero as e_int(i-1)
143     e_int(:,i)=dt*e(:,i); % Integrate error
144 else
145     e_int(:,i)=e_int(:,i-1)+dt*e(:,i); % Integrate error
146 end
147
148 %Heading controller
149 Δ_ff=(1/K)*psi_d(2,i)+(T/K)*psi_d(3,i); % Heading reference ...
      feed forward
150 Δ_PID=-Kp*e(1,i)-Kd*e(2,i)-Ki*e_int(1,i); % Heading PID controller
151
152 %Surge controller
153 n_c_ff=(1/Ksurge)*u_d(1,i)+(Tsurge/Ksurge)*u_d(2,i); % ...
      Surge reference feed forward
154 n_c_PI=-Kp_surge*e(3,i)-(Kp_surge/Ti_surge)*e_int(3,i); % ...
      Surge PI controller
155
156 %Integrator Anti Wind-up
157 %Heading
158 if abs(Δ_PID+Δ_ff)>Δ_max && i>1 % Rudder saturated, and not ...
      first iteration
159     AntiWU(1,i)=-1; % Log anti WU
160     Δ_I=-Ki*e_int(1,i)-(-Ki*e_int(1,i-1)); % Integrator direction
161     if sign(Δ_I)==sign(Δ_PID+Δ_ff) % Integrator direction is the ...
          same as the propellor speed (Integrates further into ...
          saturation)
162         e_int(1,i)=e_int(1,i-1); % Reset integrator ...
          to previous value
163         Δ_PID=-Kp*e(1,i)-Kd*e(2,i)-Ki*e_int(1,i); % Calculate ...
          heading PID controller output again
164         AntiWU(1,i)=1; % Log anti WU
165     end
166 end
167 %Surge
168 if abs(n_c_PI+n_c_ff)>n_c_max && i>1 % Shaft speed ...
      saturated, and not first iteration
169     AntiWU(2,i)=-1; % Log anti WU
170     Δ_I=-(Kp_surge/Ti_surge)*e_int(3,i)-(-Kp_surge/Ti_surge)*...
          e_int(3,i-1); % Integrator direction
171     if sign(Δ_I)==sign(n_c_PI+n_c_ff) % Integrator direction is ...
          the same as the propellor speed (Integrates further into ...
          saturation)
172         e_int(3,i)=e_int(3,i-1); % Reset integrator to ...
          previous value
173         n_c_PI=-Kp_surge*e(3,i)-(Kp_surge/Ti_surge)*...
          e_int(3,i); % Calculate surge PI controller output ...
          again
174         AntiWU(2,i)=1; % Log anti WU
175     end
176 end
177 end
178 end
179

```

```

180     %%%%%%%%% MS Fartoystyring %%%%%%%%%
181     tau(:,i)=[Δ_ff+Δ_PID;n_c_ff+n_c_PI];      % Define input
182     if tau(2,i)<0      % Ensure non-negative propellor speed, ...
        msfartoystyring crashes if n_c < 0
183         tau(2,i)=0;
184         warning(['Desired propellor speed was negative at iteration ' ...
            num2str(i) ' . Modified to zero.'])
185     end
186     x_dot=msfartoystyring(x(:,i),tau(:,i),1);  % Simulate ship with ...
        current
187
188     %Integrate
189     x(:,i+1)=x(:,i)+dt*x_dot;
190 end
191
192 pathplotter(x(4,:), x(5,:), x(6,:), dt, 50, 0, sim_time, 0, WP)

```

```

1  % TTK4190 - Guidance and Control
2  % Assignment 3
3  % 105334 - Bjorn-Olav H. Eiksen
4  % 746029 - Marius Hjertaker
5  % 705009 - Sveinung Ohrem
6  % 745621 - Ole Maurice Rabanal
7  %
8  % This is the code for task 2.7 of the assignment
9
10 clear; clc;
11 dispstat('','init') % ...
    Simulation progress. Initialization. Does not print anything.
12 dispstat('run_task_2_7. Path-tracking','keepthis'); % Output first
13
14 %Integration values
15 sim_time=12000; % Simulation end time
16 dt=1; % Steplength
17 L=sim_time/dt; % Simulation steps
18
19 %System
20 K=-0.0261; % Heading model gain, modified from -0.0598
21 T=261.348; % Heading model time constant, modified from 115.5667
22 Ksurge=0.9820; % Surge model gain
23 Tsurge=450.8367; % Surge model time constant
24
25 %User parameters
26 load('WP.mat'); % Load waypoints
27 x_0=[6.63,0,0,0,0,0,0]'; % Initial condition, x = [ u v r x y psi]'
28 n_c_max=85*2*pi/60; % Max shaft velocity [rad/s]
29 Δ_max = 25*pi/180; % max rudder angle [rad]
30 s_d=2*305; % Desired lateral distance
31 e_d=200; % Desired cross-track error, e_d=0 causes the ...
    boat to line up with the target
32 U_a_max=4; % Maximum approach speed
33 U_target=3; % Target speed

```

```

34 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]'
35
36 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
37 %Heading controller
38 zeta=1; % Damping coefficient
39 omega_b=0.06; % Bandwidth frequency
40 omega_n=omega_b/(sqrt(1-2*zeta^2+sqrt(4*zeta^4-4*zeta^2+2))); % ...
    Natural frequency
41 Kp=(T/K)*omega_n^2; % Heading controller proportional gain
42 Kd=(T/K)*2*zeta*omega_n-1/T; % Heading controller derivative gain
43 Ki=(omega_n/10)*Kp; % Heading controller integral gain
44
45 %Surge controller
46 Kp_surge=2.7388; % Surge controller proportional gain
47 Ti_surge=Tsurg+50; % Surge controller integral time
48
49 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
50 time=zeros(1,L+1); % Time vector
51 x=zeros(6,L+1); % State vector, x = [u v r x y psi]'
52 tau=zeros(2,L); % Input vector, tau = [Δ n_c]'
53 e=zeros(3,L); % Control error vector, e = [psi_r-r u]'
54 e_int=zeros(3,L); % Integrated control error e_int = [psi_r-int ...
    r-int u-int]'
55 psi_r=zeros(1,L); % Requested heading
56 psi_d=zeros(3,L); % Filtered heading references psi_d = [psi_d ...
    psi_d_dot psi_d_ddot]'
57 u_r=zeros(1,L); % Requested surge speed
58 u_d=zeros(2,L); % Filtered surge references u_d = [u_d u_d_dot]'
59 target=zeros(3,L+1); % Target position target = [x_t y_t psi_t]'
60 track=zeros(2,L); % Tracking distance track = [s e]'
61
62 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
63 %Heading
64 omega_ref=0.15; % Natural frequency for reference model
65 zeta_ref=1; % Damping coefficient
66 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
67 B_h=[0;0;omega_ref^3]; % Continous B matrix
68 psi_d_0 = [x_0(6) x_0(3) 0]'; % Initial state of filtered heading ...
    reference
69
70
71 %Surge
72 omega_ref=0.05; % Natural frequency for reference model
73 zeta_ref=1; % Damping coefficient
74 A_s=[0 1;-omega_ref^2 -2*zeta_ref*omega_ref]; % Continous A matrix
75 B_s=[0;omega_ref^2]; % Continous B matrix
76 u_d_0 = [x_0(1) 0]'; % Initial state of filtered surge references
77
78
79 x(:,1)=x_0; % Set initial state
80 target(:,1)=target_0; % Initial position target
81 Δ_lookahead=8*305; % Lookahead distance for Lookahead-Based Steering

```

```

82  $\Delta_s=2400$ ; % Speed tuning parameter
83
84 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]'
85 for i=1:L
86     if mod(i,100)==0
87         dispstat(sprintf('Progress %02d%%',round((i/L)*100))); % ...
            Simulation progress in %
88     end
89     time(i+1)=time(i)+dt; % Update time vector
90
91     %%%%%%%%% Guidance system %%%%%%%%%
92     Chi_t=target(3,i); % Target ...
        heading
93     R_p=[cos(Chi_t) -sin(Chi_t) ; sin(Chi_t) cos(Chi_t)]; % ...
        Transformation matrix from ned to path-fixed frame
94     track(:,i)=[s_d e_d]'+R_p*[x(4,i)-target(1,i) ; ...
        x(5,i)-target(2,i)]; % Tracking error
95
96     Chi_r = atan(-track(2,i)/ $\Delta_{lookahead}$ ); % ...
        Lookahead LOS steering law
97     Chi_d = Chi_r + target(3,i); ...
        % Desired course
98     U_d = U_target-U_a_max*(track(1,i)/sqrt(track(1,i)^2+ $\Delta_s^2$ )); % ...
        Desired ship speed
99
100    %Mapping
101    u=x(1,i); % Surge speed
102    v=x(2,i); % Sway speed
103    U=sqrt(u^2+v^2); % Course speed
104    beta=asin(v/U); % Sideslip angle
105
106    psi_r(i) = Chi_d-beta; % Requested heading
107    u_r(i) = sqrt(U_d^2-v^2); % Requested surge speed
108
109    %Prefilters
110    if i==1 % First iteration, use initial state as psi_d(i-1)
111        psi_d(:,i)=(eye(3)+dt.*A_h)*psi_d_0+dt*B_h*psi_r(i); % ...
            Heading prefilter
112        u_d(:,i)=(eye(2)+dt.*A_s)*u_d_0+dt*B_s*u_r(i); % ...
            Surge prefilter
113    else
114        psi_d(:,i)=(eye(3)+dt.*A_h)*psi_d(:,i-1)+dt*B_h*psi_r(i); % ...
            Heading prefilter
115        u_d(:,i)=(eye(2)+dt.*A_s)*u_d(:,i-1)+dt*B_s*u_r(i); % ...
            Surge prefilter
116    end
117
118    %%%%%%%%% Control algorithm %%%%%%%%%
119    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
120    if i==1 % First iteration, use zero as e_int(i-1)

```

```

121     e_int(:,i)=dt*e(:,i); % Integrate error
122 else
123     e_int(:,i)=e_int(:,i-1)+dt*e(:,i); % Integrate error
124 end
125
126 %Heading controller
127 Δ_ff=(1/K)*psi_d(2,i)+(T/K)*psi_d(3,i); % Heading reference ...
    feed forward
128 Δ_PID=-Kp*e(1,i)-Kd*e(2,i)-Ki*e_int(1,i); % Heading PID controller
129
130 %Surge controller
131 n_c_ff=(1/Ksurge)*u_d(1,i)+(Tsurge/Ksurge)*u_d(2,i); % ...
    Surge reference feed forward
132 n_c_PI=-Kp_surge*e(3,i)-(Kp_surge/Ti_surge)*e_int(3,i); % ...
    Surge PI controller
133
134 %Integrator Anti Wind-up
135 %Heading
136 if abs(Δ_PID+Δ_ff)>Δ_max && i>1 % Rudder saturated, and not ...
    first iteration
137     AntiWU(1,i)=-1; % Log anti WU
138     Δ_I=-Ki*e_int(1,i)-(-Ki*e_int(1,i-1)); % Integrator direction
139     if sign(Δ_I)==sign(Δ_PID+Δ_ff) % Integrator direction is the ...
        same as the propellor speed (Integrates further into ...
        saturation)
140         e_int(1,i)=e_int(1,i-1); % Reset integrator ...
            to previous value
141         Δ_PID=-Kp*e(1,i)-Kd*e(2,i)-Ki*e_int(1,i); % Calculate ...
            heading PID controller output again
142         AntiWU(1,i)=1; % Log anti WU
143     end
144 end
145 %Surge
146 if abs(n_c_PI+n_c_ff)>n_c_max && i>1 % Shaft speed ...
    saturated, and not first iteration
147     AntiWU(2,i)=-1; % Log anti WU
148     Δ_I=-(Kp_surge/Ti_surge)*e_int(3,i)-(-Kp_surge/Ti_surge)...
        *e_int(3,i-1); % Integrator direction
149     if sign(Δ_I)==sign(n_c_PI+n_c_ff) % Integrator direction is ...
        the same as the propellor speed (Integrates further into ...
        saturation)
150         e_int(3,i)=e_int(3,i-1); % Reset integrator to ...
            previous value
151         n_c_PI=-Kp_surge*e(3,i)-(Kp_surge/Ti_surge)...
            *e_int(3,i); % Calculate surge PI controller output ...
            again
152         AntiWU(2,i)=1; % Log anti WU
153     end
154 end
155
156 end
157
158 %%%%%%%%% MS Fartostyring %%%%%%%%%
159 tau(:,i)=[Δ_ff+Δ_PID;n_c_ff+n_c_PI]; % Define input

```

```

160     if tau(2,i)<0          % Ensure non-negative propellor speed, ...
        msfartoystyring crashes if n_c < 0
161         tau(2,i)=0;
162         warning(['Desired propellor speed was negative at iteration ' ...
            num2str(i) ' . Modified to zero.'])
163     end
164     x_dot=msfartoystyring(x(:,i),tau(:,i),0);    % Simulate ship with ...
        current
165
166     %Integrate
167     x(:,i+1)=x(:,i)+dt*x_dot;
168     target(:,i+1) = target(:,i) + dt*U_target*[cos(target(3,i)) ...
        sin(target(3,i)) 0]';
169 end
170
171 pathplotter(x(4,:), x(5,:), x(6,:), dt, 50, 0, sim_time, 1, WP)

```