TTK4190 Guidance and Control of Vehicles

Assignment 3, Part 5

Written Fall 2020 By Magnus Dyre-Moe, Patrick Nitschke and Siawash Naqibi

1 Kalman Filter Design

1.1 Problem a)

We extracted our model from chapter 13 by looking at the design of the heading autopilot. Our model only consists of three states; heading, heading rate and rudder bias. Thus we only needed the system equations for these states only. Our states are heading, heading rate and rudder bias.

Our states are given as such and the measurement is only the heading.

$$x = \begin{bmatrix} \psi \\ r \\ b \end{bmatrix} \quad y = \psi \tag{1}$$

The system differential equations are given as such.

$$\begin{aligned} \dot{\psi} &= r \\ \dot{r} &= -\frac{1}{T}r - \frac{K}{T}b + \frac{K}{T}u + \omega_r \\ \dot{b} &= \omega_b \end{aligned} \tag{2}$$

The Kalman filter matrices are illustrated below.

$$\begin{gathered} A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -\frac{1}{T} & \frac{K}{T} \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ \frac{K}{T} \\ 0 \end{bmatrix} \\ C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 0 \end{bmatrix} \quad E = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \end{gathered} \tag{3}$$

1.2 Problem b)

The first order discrete time Kalman filter matrices are given by these formulas. The time step has the symbol h.

$$\begin{aligned} A_d &= I_3 + Ah \\ B_d &= Bh \\ C_d &= C \\ D_d &= D \\ E_d &= Eh \end{aligned} \tag{4}$$

1.3 Problem c)

The observability of the system can be calculated by checking the determinant of the given matrix.

$$O = \begin{bmatrix} C \\ CA \\ CA^2 \end{bmatrix} \tag{5}$$

Using the **obsv(A,C)** function in MATLAB along with rank allows us to find whether the system is observable. As the function returns 3, which matches dim(A), we know that the system is observable.

2 Implementation

2.1 Problem a)

Adding the Gaussian distributed noise to the yaw and yaw rate states produced a ψ_{meas} and r_{meas} as seen in Figure 1.

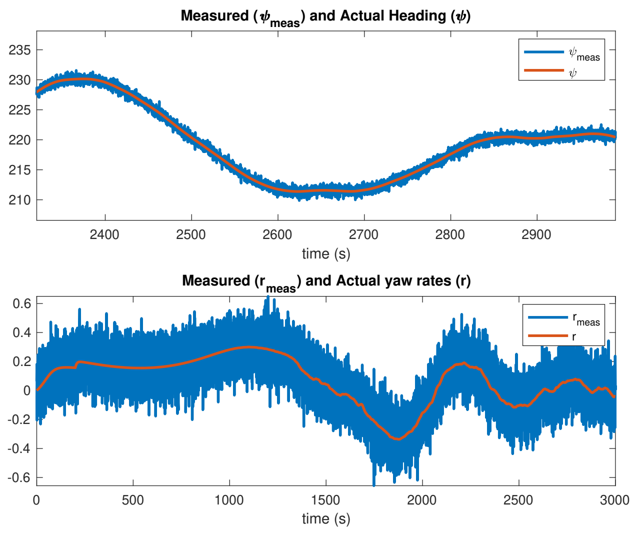


Figure 1: Noisy yaw and yaw rate measurements versus true states.

2.2 Problem b)

Using the formulas in Fossen chapters 13.4.1, we implemented the Kalman filter in our code. This is seen in lines 180 to 236 in the appendix, starting with the Nomoto model and ending with the algorithm.

By plotting the Kalman estimates against the true states we obtained Figure 2. This was done

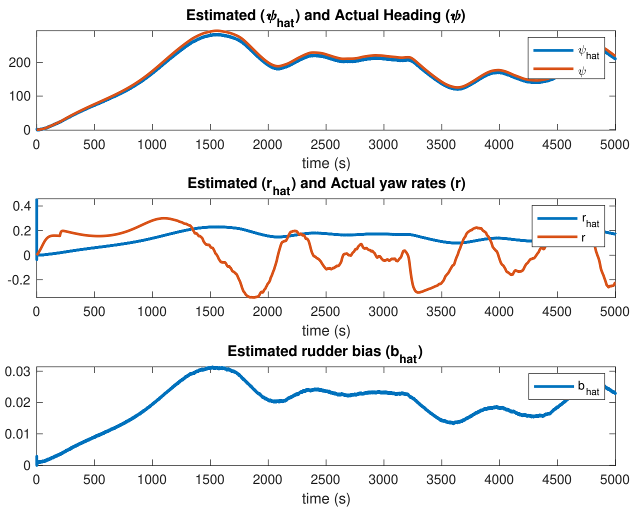


Figure 2: Kalman estimates against the true states.

using our tuned values for **R**, **Q** and **P**(0):

$$\mathbf{R} = deg2rad(0.5)^2, \quad \mathbf{Q} = \begin{bmatrix} deg2rad(20)^2 & 0 \\ 0 & deg2rad(14.7)^2 \end{bmatrix}, \quad \mathbf{P}(0) = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \tag{6}$$

These values were tuned through different methods:

P(0) somewhat represents our "uncertainty" on our current state, and since we knew our start position, we could essentially put this to 0. Values of 0.1 were used as these as was relatively low, but not 0. (However, having it at 0 doesn't change anything it seems.)

R represents the measurement noise covariance. Since we are directly adding Gaussian noise to states, the standard deviation of the noise could be directly inserted into R.

Q represents our process noise covariance. With the ground truth values present, we tuned the Q matrix by starting low, and gradually increasing its values. This means we started with an overconfident filter that had significant deviation from our ground truth, but as we increased the process noise began to line up nicely.

In the end, we see that the estimated and true Kalman values correspond nicely. However, the Kalman filter struggles to follow the yaw rate without any measurement updates.

2.3 Problem c)

Using the measurements directly as feedback into the heading autopilot resulted in the following plots:

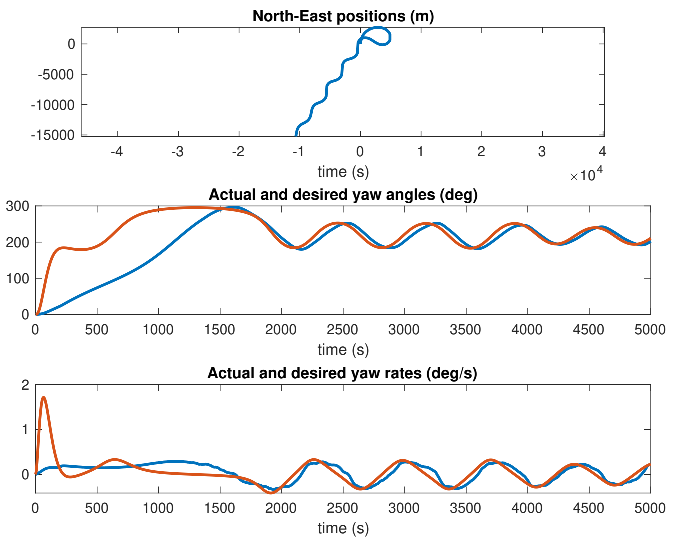


Figure 3: Ship position, yaw and yaw rates, along with commanded yaw and yaw rates.

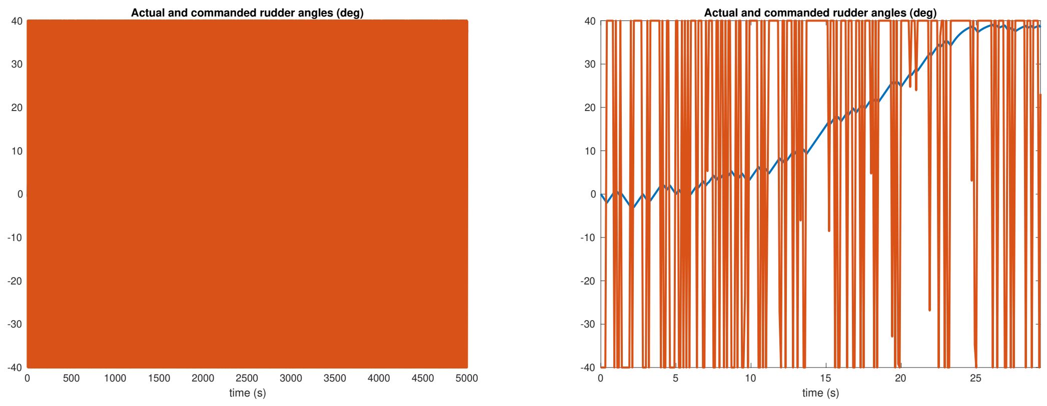


Figure 4: The actual and commanded rudder angles, with the right figure being a zoom of the left.

What we observe is an underdamped system where the noisy measurements causes oscillations in the ship behaviour. This is caused by the ship rudder oscillating very quickly as well, seen in figure 4. This results in a difficulty to follow a desired path seen in the oscillatory upper plot in figure 3.

2.4 Problem d)

Using Kalman state estimates as feedback gave the following results:

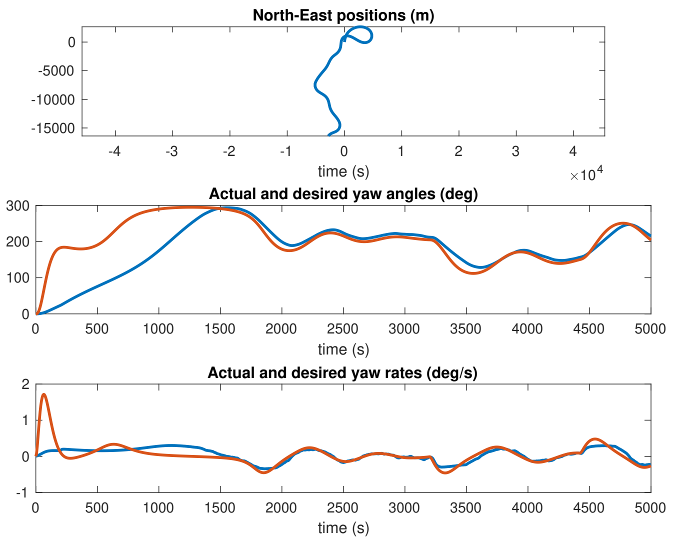
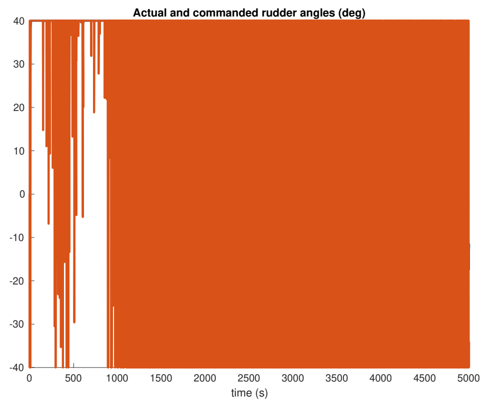


Figure 5: Ship position, yaw and yaw rates, along with commanded yaw and yaw rates.



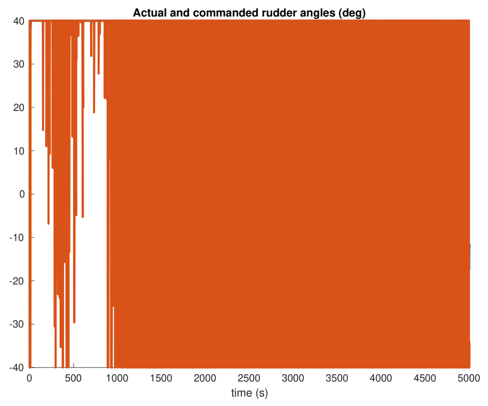


Figure 6: The actual and commanded rudder angles, with the right figure being a zoom of the left.

From what we see in figure 5, the desired path of the boat is followed much closer albeit some small oscillations. We see the heading autopilot still responds to quite a bit of noise, reflected in the very rapid switching of the rudder. However, this of smaller magnitude than with a pure measurement feedback in problem 2c). Ultimately, we would say that this is an impressive performance of the path-following system.

3 Navigation Systems

3.1 Problem a)

A wave filter in model-based navigation system works as filter on high frequent wave motions. Generally, waves comes in different frequencies, with slow, damped waves and fast, sharp waves. Navigating on the sharp, high frequency waves will be computationally complex and wear-and-tear on the physical system, such as the rudder. Hence, by using a filter we only capture the slow, damped waves. We can navigate through these waves without applying to much stress on the physical system.

A wave filter has the response of a low pass filter. It is realized in practice by a low pass filter in series with a notch filter.

3.2 Problem b)

With model-based navigation systems we use estimated states as inputs to the system model. With inertial navigation (INS) we use measurements as inputs. These inputs comes from an IMU and is ofter measured through accelerometer and ARS (gyroscope) and maybe a magnetometer. Because measurements from the IMU drifts it is crucial to have correction measurements, often from GNSS. For model-based navigation systems these correction measurements are not as necessary, but still used for a lot of physical systems.

A Matlab code

A.1 project.m

```
% Project in TTK4190 Guidance and Control of Vehicles
%
% Author:           Magnus Dyre-Moe, Patrick Nitschke and Siawash Naqibi
% Study program:    Cybernetics and Robotics

clear;
clc;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% USER INPUTS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
h  = 0.1;    % sampling time [s]
Ns = 10000;  % no. of samples

psi_ref = 10 * pi/180;  % desired yaw angle (rad)
U_d = 7;                % desired cruise speed (m/s)

% ship parameters
m = 17.0677e6;          % mass (kg)
Iz = 2.1732e10;         % yaw moment of inertia about CO (kg m^3)
xg = -3.7;              % CG x-ccordinate (m)
L = 161;                % length (m)
B = 21.8;               % beam (m)
T = 8.9;                % draft (m)
%KT = 0.7;               % propeller coefficient (-)
Dia = 3.3;              % propeller diameter (m)
rho = 1025;             % density of water (kg/m^3)
visc = 1e-6;            % kinematic viscousity at 20 degrees (m/s^2)
eps = 0.001;            % a small number added to ensure that the denominator of ...
    Cf is well defined at u=0
k = 0.1;                % form factor giving a viscous correction
t_thr = 0.05;           % thrust deduction number


% rudder limitations
Δ_max  = 40 * pi/180;        % max rudder angle      (rad)
DΔ_max = 5  * pi/180;        % max rudder derivative (rad/s)

% added mass matrix about CO
Xudot = -8.9830e5;
Yvdot = -5.1996e6;
Yrdot =  9.3677e5;
Nvdot =  Yrdot;
Nrdot = -2.4283e10;
MA = -[ Xudot 0    0
        0 Yvdot Yrdot
        0 Nvdot Nrdot ];

% rigid-body mass matrix
MRB = [ m 0    0
        0 m    m*xg
        0 m*xg Iz ];

Minv = inv(MRB + MA); % Added mass is included to give the total inertia

% ocean current in NED
Vc = 1;                             % current speed (m/s)
betaVc = deg2rad(45);               % current direction (rad)

% wind expressed in NED
Vw = 10;                   % wind speed (m/s)
betaVw = deg2rad(135);     % wind direction (rad)
rho_a = 1.247;             % air density at 10 deg celsius
cy = 0.95;                 % wind coefficient in sway
cn = 0.15;                 % wind coefficient in yaw
```

```
A_Lw = 10 * L;              % projected lateral area

% linear damping matrix (only valid for zero speed)
T1 = 20; T2 = 20; T6 = 10;

Xu = -(m - Xudot) / T1;
Yv = -(m - Yvdot) / T2;
Nr = -(Iz - Nrdot)/ T6;
D = diag([-Xu -Yv -Nr]); % zero speed linear damping


% rudder coefficients (Section 9.5)
b = 2;
AR = 8;
CB = 0.8;

lambda = b^2 / AR;
tR = 0.45 - 0.28*CB;
CN = 6.13*lambda / (lambda + 2.25);
aH = 0.75;
xH = -0.4 * L;
xR = -0.5 * L;

X_Δ2 = 0.5 * (1 - tR) * rho * AR * CN;
Y_Δ = 0.25 * (1 + aH) * rho * AR * CN;
N_Δ = 0.25 * (xR + aH*xH) * rho * AR * CN;

% input matrix
Bu = @(u_r,Δ) [ (1-t_thr)  -u_r^2 * X_Δ2 * Δ
                        0      -u_r^2 * Y_Δ
                        0      -u_r^2 * N_Δ            ];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Heading Controller
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Linearlized coriolis matrices
CRBstar = [ 0 0 0
        0 0 m*U_d
        0 0 m*xg*U_d];
CRBstar = CRBstar(2:3,2:3); % reduced to sway and yaw

CAstar = [   0   0                    0
        0   0               -Xudot*U_d
        0   (Xudot-Yvdot)*U_d   -Yrdot*U_d];
CAstar = CAstar(2:3,2:3); % reduced to sway and yaw

% Reduced D matrix
D_reduced = D(2:3,2:3);

% Reduced M
Minv_reduced = Minv(2:3,2:3); % 2 by 2


% linearized sway-yaw model (see (7.15)-(7.19) in Fossen (2021)) used
% for controller design. The code below should be modified.
N_lin = CRBstar + CAstar + D_reduced;
b_lin = [-2*U_d*Y_Δ -2*U_d*N_Δ]';

% initial states
eta = [0 0 0]';
nu  = [0.1 0 0]';
Δ = 0;
n = 0;
z = 0;
xd = [0; 0; 0];

% Tranfer function from Δ to r
[num,den] = ss2tf(-Minv_reduced * N_lin, Minv_reduced * b_lin, [0 1], 0);
```

```
root = roots(den);
T1 = -1/root(1);
T2 = -1/root(2);
T3 = num(2)/num(3);
T_nomoto = T1 + T2 - T3;
K_nomoto = num(3)/(root(1)*root(2));


% rudder control law
wb = 0.06;
zeta = 1;
wn = 1 / sqrt( 1 - 2*zeta^2 + sqrt( 4*zeta^4 - 4*zeta^2 + 2) ) * wb;

m_nomoto = T_nomoto / K_nomoto;
d = 1 / K_nomoto;
Kp = wn^2 * m_nomoto;
Kd = ( 2 * zeta * wn * T_nomoto - 1 ) / K_nomoto;
Ki = wn^3 / 10 * m_nomoto;
w_ref = 0.03;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PART 3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Propeller coefficients
num_blades = 4;          % number of propeller blades
AEAO = 0.65;             % area of blade
PD = 1.5;                % pitch/diameter ratio
Ja = 0;                  % bollard pull
[KT, KQ] = wageningen(Ja, PD, AEAO, num_blades); %Propeller coefficients

Qm = 0;
t_T = 0.05;

% LOS guidance law initalization. Part 4
way_points = load('WP.mat', '-mat');
way_points = way_points.WP
start_point = 1;
end_point = 2;
lookaheaddistance = 4 * L;
epsilon = 3 * L;

y_int = 0;

% Part 5: Kalman filter
A_kont = [0 1 0;
         0 -(1/T_nomoto) -(K_nomoto/T_nomoto);
         0 0 0];

B_kont = [0; K_nomoto/T_nomoto; 0];

E_kont = [0 0;
          1 0;
          0 1];

Cd = [1 0 0];
Dd = 0;

[Ad, Bd] = c2d(A_kont, B_kont, h);
[Ad, Ed] = c2d(A_kont, E_kont, h);

rank_obsv = rank(obsv(Ad, Cd));

angle_noise = normrnd(0, deg2rad(0.5), 1, (Ns+1)*8);
angle_rate_noise = normrnd(0, deg2rad(0.1), 1, (Ns+1)*8);

x0 = [0; 0; 0]; %yaw, yaw angle, rudder bias initialization
P0 = diag([0.0, 0.0, 0.0]);

```

```
Qd = diag([deg2rad(20)^2, deg2rad(14.7)^2]); % model disturbance
Rd = deg2rad(0.5)^2; % measurement noise

x = x0; x_pred = x0;
P_pred = P0;
Δ = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MAIN LOOP
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
simdata = zeros(Ns+1,22);                % table of simulation data

for i=1:(5*Ns+1)
    t = (i-1) * h;                       % time (s)

    % Kalman loop

    %Kalman gain K[k]
    K = P_pred * Cd' * inv( Cd * P_pred * Cd' + Rd );
    IKC = eye(3) - K * Cd;

    %Control input
    u = Δ; %nu(3) + angle_rate_noise(i);
    psi_meas = ( eta(3) + angle_noise(i) );
    r_meas = ( nu(3) + angle_rate_noise(i) );

    %Corrector: x_hat[k] and P_hat[k]
    x_hat = x_pred + K * ( psi_meas - Cd * x_pred );
    P_hat = IKC * P_pred * IKC' + K * Rd * K';

    %Predictor: x_pred[k+1] and P_pred[k+1]
    x_pred = Ad * x_hat + Bd * u;
    P_pred = Ad * P_hat * Ad' + Ed * Qd * Ed';

    %Ship simulator
    psi_est = x_hat(1);
    r_est = x_hat(2);
    rudder_bias_est = x_hat(3);

    if ( (way_points(1,end_point) - eta(1))^2 + (way_points(2,end_point) - eta(2))...
        ^2 < epsilon^2 )
        display('yolo')
        start_point = start_point + 1;
        end_point = end_point + 1;
    end

    % Reference model
    [psi_ref, y_int_dot] = integral_los_guidancelaw(eta(1), eta(2), way_points(:,...
        start_point), way_points(:,end_point), lookaheaddistance, y_int);
    %psi_ref = los_guidancelaw(eta(1), eta(2), way_points(:,start_point), ...
        way_points(:,end_point), lookaheaddistance);
    Ad = [0              1                   0;
          0              0                   1;
          -w_ref^3 -(2*zeta+1)*w_ref^2 -(2*zeta+1)*w_ref];

    Bd = [0; 0; w_ref^3];

    xd_dot = Ad * xd + Bd * psi_ref;

    % Rotation from body to NED
    R = Rzyx(0,0,eta(3));

    % current (should be added here)
    nu_r = nu - [Vc*cos(betaVc - eta(3)), Vc*sin(betaVc - eta(3)), 0]';
    u_c = Vc*cos(betaVc - eta(3));

    % wind (should be added here)
    if t > 200
        u_rw = nu(1) - Vw * cos(betaVw - eta(3));
        v_rw = nu(2) - Vw * sin(betaVw - eta(3));
        V_rw = sqrt(u_rw^2 + v_rw^2);
        gamma_w = -atan2(v_rw, u_rw);
        Cy = cy * sin( gamma_w );
        Cn = cn * sin( 2*gamma_w );
        Ywind = 0.5 * rho_a * V_rw^2 * Cy * A_Lw; % expression for wind moment in sway should be added.
        Nwind = 0.5 * rho_a * V_rw^2 * Cn * A_Lw * L; % expression for wind moment in yaw should be added.
    else
        Ywind = 0;
        Nwind = 0;
    end
    tau_env = [0 Ywind Nwind]';

    % state-dependent time-varying matrices
    CRB = m * nu(3) * [ 0 -1 -xg
                        1  0  0
                        xg 0  0  ];

    % coriolis due to added mass
    CA = [  0    0    Yvdot * nu_r(2) + Yrdot * nu_r(3)
            0    0    -Xudot * nu_r(1)
          -Yvdot * nu_r(2) - Yrdot * nu_r(3)    Xudot * nu_r(1)    0];
    N = CRB + CA + D;

    % nonlinear surge damping
    Rn = L/visc * abs(nu_r(1));
    Cf = 0.075 / ( (log(Rn) - 2)^2 + eps);
    Xns = -0.5 * rho * (B*L) * (1 + k) * Cf * abs(nu_r(1)) * nu_r(1);

    % cross-flow drag
    Ycf = 0;
    Ncf = 0;
    dx = L/10;
    Cd_2D = Hoerner(B,T);
    for xL = -L/2:dx:L/2
        vr = nu_r(2);
        r = nu_r(3);
        Ucf = abs(vr + xL * r) * (vr + xL * r);
        Ycf = Ycf - 0.5 * rho * T * Cd_2D * Ucf * dx;
        Ncf = Ncf - 0.5 * rho * T * Cd_2D * xL * Ucf * dx;
    end
    d = -[Xns Ycf Ncf]';

    % reference models
    psi_d= xd(1);
    r_d = xd(2);
    u_d = U_d;

    % thrust
    thr = rho * Dia^4 * KT * abs(n) * n;    % thrust command (N) Equation 9.7

    % control law
    z_dot = psi_est - xd(1);
    Δ_c = - ( Kp * (psi_est - xd(1)) + Kd * (r_est - xd(2)) + Ki * z );  % rudder angle command (rad)

    % ship dynamics
    u = [ thr Δ ]';
    tau = Bu(nu_r(1),Δ) * u;
    nu_dot = Minv * (tau_env + tau - N * nu_r - d);
    eta_dot = R * nu;

    % Rudder saturation and dynamics (Sections 9.5.2)

    if abs(Δ_c) ≥ Δ_max
        z = z - (h / Ki) * (sign(Δ_c)*Δ_max - Δ_c);
        Δ_c = sign(Δ_c)*Δ_max;
    end

    Δ_dot = Δ_c - Δ;
    if abs(Δ_dot) ≥ DΔ_max
        Δ_dot = sign(Δ_dot)*DΔ_max;
    end

    % propeller dynamics
    Im = 100000; Tm = 10; Km = 0.6;         % propulsion parameters
    n_c = 10;                               % propeller speed (rps)

    T_prop = rho * Dia^4 * KT * abs(n) * n;
    Q_prop = rho * Dia^5 * KQ * abs(n) * n;
    T_d = (U_d - u_c)*Xu / (t_T - 1);
    n_d = sign(T_d) * sqrt( abs(T_d) / (rho*Dia^4*KT) );
    Q_d = rho * Dia^5 * KQ * abs(n_d) * n_d;
    Y = (1 / Km) * Q_d;
    Qm_dot = (1 / Tm) * (-Qm + Y*Km);
    Qf = 0;

    n_dot = (1/Im) * (Qm - Q_prop - Qf);              % should be changed in Part 3

    % Stored in simdata
    sideslip_angle = asin( nu_r(2) / sqrt(nu_r(1)^2 + nu_r(2)^2) );
    crab_angle = atan( nu(2) / nu(1) );
    course = eta(3) + crab_angle;

    % store simulation data in a table (for testing)
    simdata(i,:) = [t n_c Δ_c n Δ eta' nu' u_d psi_d r_d sideslip_angle crab_angle...
         course psi_meas psi_est r_meas r_est rudder_bias_est];

    % Euler integration
    eta = euler2(eta_dot,eta,h);
    nu  = euler2(nu_dot,nu,h);
    Δ = euler2(Δ_dot,Δ,h);
    n  = euler2(n_dot,n,h);
    z = euler2(z_dot,z,h);
    xd = euler2(xd_dot,xd,h);
    Qm = euler2(Qm_dot,Qm,h);
    y_int = euler2(y_int_dot,y_int,h);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PLOTS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
t       = simdata(:,1);                 % s
n_c     = 60 * simdata(:,2);            % rpm
Δ_c = (180/pi) * simdata(:,3);      % deg
n       = 60 * simdata(:,4);            % rpm
Δ   = (180/pi) * simdata(:,5);      % deg
x       = simdata(:,6);                 % m
y       = simdata(:,7);                 % m
psi     = (180/pi) * simdata(:,8);      % deg
u       = simdata(:,9);                 % m/s
v       = simdata(:,10);                % m/s
r       = (180/pi) * simdata(:,11);     % deg/s
u_d     = simdata(:,12);                % m/s
psi_d   = (180/pi) * simdata(:,13);     % deg
r_d     = (180/pi) * simdata(:,14);     % deg/s
sideslip = simdata(:,15);
crab    = simdata(:,16);
course = (180/pi) * simdata(:,17);
psi_meas = (180/pi) * simdata(:,18);
psi_est = (180/pi) * simdata(:,19);
r_meas = (180/pi) * simdata(:,20);
r_est = (180/pi) * simdata(:,21);
rudder_bias_est = (180/pi) * simdata(:,22);


figure(1)
figure(gcf)
subplot(311)
```

```
plot(y,x,'linewidth',2); axis('equal')
title('North-East positions (m)'); xlabel('time (s)');
subplot(312)
plot(t,psi,t,psi_d,'linewidth',2);
title('Actual and desired yaw angles (deg)'); xlabel('time (s)');
subplot(313)
plot(t,r,t,r_d,'linewidth',2);
title('Actual and desired yaw rates (deg/s)'); xlabel('time (s)');

%{
figure(2)
figure(gcf)
subplot(311)
plot(t,u,t,u_d,'linewidth',2);
title('Actual and desired surge velocities (m/s)'); xlabel('time (s)');
subplot(312)
plot(t,n,t,n_c,'linewidth',2);
title('Actual and commanded propeller speed (rpm)'); xlabel('time (s)');
subplot(313)
plot(t,∆,t,∆_c,'linewidth',2);
title('Actual and commanded rudder angles (deg)'); xlabel('time (s)');

figure(3)
figure(gcf)
subplot(211)
plot(t,u,'linewidth',2);
title('Actual surge velocity (m/s)'); xlabel('time (s)');
subplot(212)
plot(t,v,'linewidth',2);
title('Actual sway velocity (m/s)'); xlabel('time (s)');

figure(4)
figure(gcf)
subplot(211)
plot(t, sideslip, t, crab, 'linewidth', 2)
title('Sideslip angle (\beta) and Crab angle (\beta_c)'); xlabel('time (s)')
legend('Sideslip', 'Crab')
subplot(212)
plot(t, psi, t, psi_d, t, course, 'linewidth',2)
title('Heading (\psi), Desired course (\chi_d) and course (\chi)'); xlabel('time (...
    s)')
legend('Heading', 'Desired course', 'Course')

%}

figure(5) % measured against true states
figure(gcf)
subplot(2,1,1)
plot(t, psi_meas, t, psi, 'linewidth', 2)
title('Measured (\psi_{meas}) and Actual Heading (\psi)'); xlabel('time (s)')
legend('\psi_{meas}', '\psi')
subplot(2,1,2)
plot(t, r_meas, t, r, 'linewidth', 2)
title('Measured (r_{meas}) and Actual yaw rates (r)'); xlabel('time (s)')
legend('r_{meas}', 'r')

figure(6) % kalman estimates against true states
figure(gcf)
subplot(3,1,1)
plot(t, psi_est, t, psi, 'linewidth', 2)
title('Estimated (\psi_{hat}) and Actual Heading (\psi)'); xlabel('time (s)')
legend('\psi_{hat}', '\psi')
subplot(3,1,2)
plot(t, r_est, t, r, 'linewidth', 2)
title('Estimated (r_{hat}) and Actual yaw rates (r)'); xlabel('time (s)')
legend('r_{hat}', 'r')
subplot(3,1,3)
plot(t, rudder_bias_est, 'linewidth', 2)
title('Estimated rudder bias (b_{hat})'); xlabel('time (s)')
legend('b_{hat}')
```

```

figure(7)
figure(gcf)
plot(t,Δ,t,Δ_c,'linewidth',2);
title('Actual and commanded rudder angles (deg)'); xlabel('time (s)');

% meas_psi against eta(3)
```

References