TTK4190 Guidance and Control of Vehicles

Assignment 3, Part 4

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

1 LOS Guidance Law for Straight-Line Path Following

1.1 Problem a)

The proportional line-of-sight (LOS) guidance is given by equation 12.78 in Fossen

$$\chi_d = \pi_p - \tan^{-1}(K_p y_e) \tag{1}$$

Here, $\pi_p = \text{atan2}(\Delta y, \Delta x)$, where Δy and Δx are position difference between two way points in NED. K_p is given by $K_p = \frac{1}{\Delta}$, where Δ is the lookahead distance. y_e is the cross-track error, that is the distance between the desired path and position of the vessel in y-coordinate given body frame. The cross-track error can be found through **crosstrackWpn(x_2,y_2,x_1,y_1,x,y)**, where x_i and y_i are way point coordinates and x and y are coordinates of the vessel. Alternatively, the cross-track error is

$$y_e = -(x - x_1) \cdot sin(\pi_p) + (y - y_1) \cdot cos(\pi_p) \tag{2}$$

The way points are stored in a given file WP.mat. Retrieving the way points are done by

```
way_points = load('WP.mat', '-mat')
way_points = way_points.WP
```

Where way_points(:,i) gives x and y coordinates for way point i. In total there are 6 way points.

1.2 Problem b)

To incorporate the LOS guidance law with our code, we simply set the heading reference ψ_d to follow the desired course χ_d. With look-ahead-distance $= 4 \cdot L$, $R = 3 \cdot L$, where R is the circle of acceptance for way points, and $\omega_{ref} = 0.03$, we arrived at the responses in Figures 6 and 2.

1.3 Problem c)

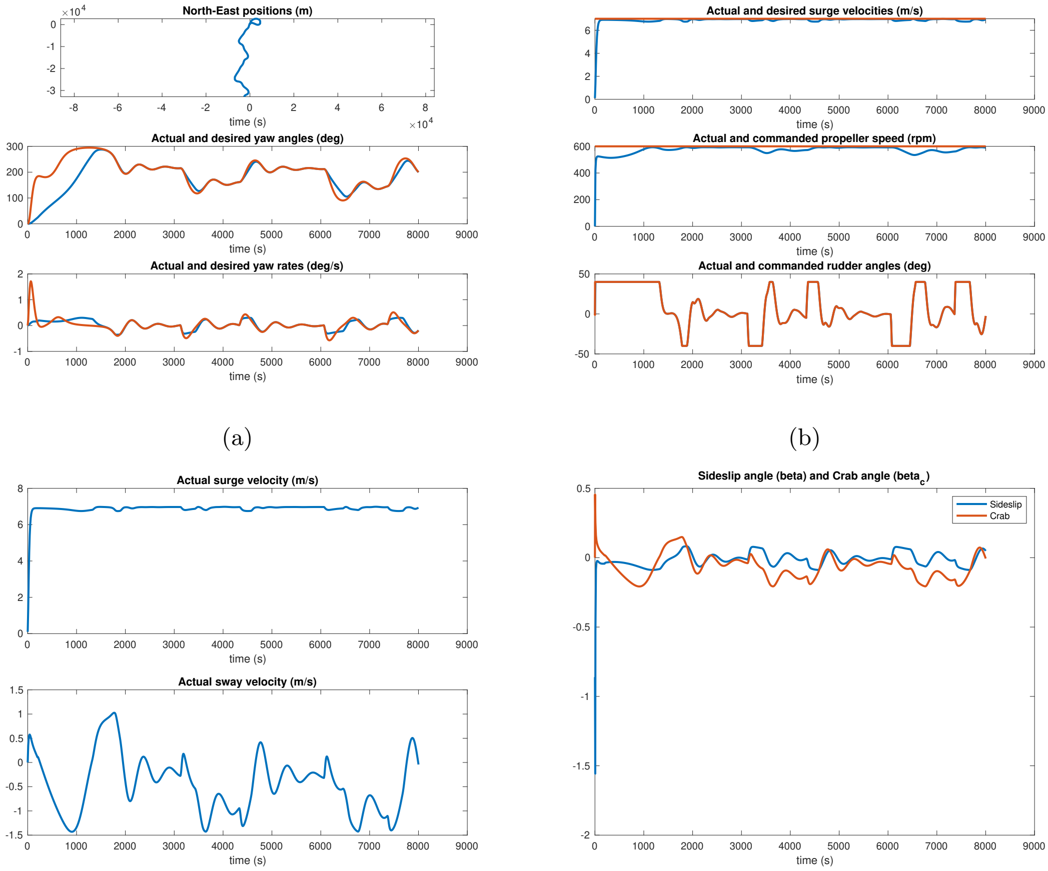
As observed in figure 6 and 2 the vessel manages to find desired path. This was while having added current and wind. This is as expected as the cross-track error is dependant on actual position. This means that a disturbance induced by currents and wind will be accounted for when finding the desired course angle.

However, worth noting, we had to account for integrator wind-up. Without an anti wind-up scheme the vessel kept looping, eventually heading in the opposite direction of the desired path. This was clearly visible from plots as the yaw angle increased linearly beyond the commanded yaw angle.

2 Crab Angle Compensation and Integral LOS

2.1 Problem a)

With the current off, we simulated the ship and observed the effects of crab and sideslip on the desired course and the actual course and heading. This is seen in Figure 3. We see that with no current, the crab angle matches the sideslip angle. This is as expected, knowing that the ship relative velocity is the same as its absolute velocity. No current indicate that sideslip and crab angle are the same, which can be observed in 4a. In turn, this means that course and heading are identical.



(a) (b)

(c) (d)

Figure 1: Ship performance with LOS guidance on straight line path following.

2.2 Problem b)

The simulation with current is seen in Figure 4.

As a result of the current, we observe some small differences in the course and heading, which match the difference in crab and sideslip. This follows our expectation of how a current induces a crab angle different from sideslip angle.

2.3 Problem c)

Transforming from χ_d to ψ_d is done by the following relationship

$$\chi = \psi + \beta_c \tag{3}$$

As output of the reference model we get χ_d. This is because χ_{ref} is the input to the reference model. Hence by compensation we get

$$\psi_d = \chi_d - \beta_c \tag{4}$$

In code this can be written as

```
chi_d = xd(1);
psi_d = chi_d - atan( nu(2) / nu(1) );
```

As observed in figure 5 course and heading are closer than in the previous figures. However, this can be even better with integrator LOS, which we will explore in section 2.4.

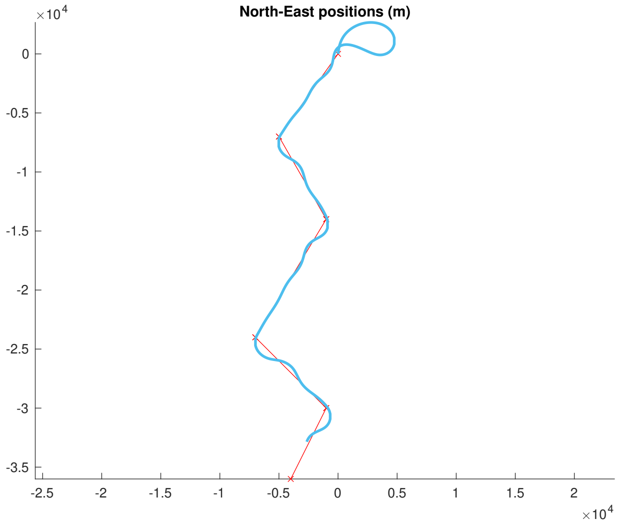
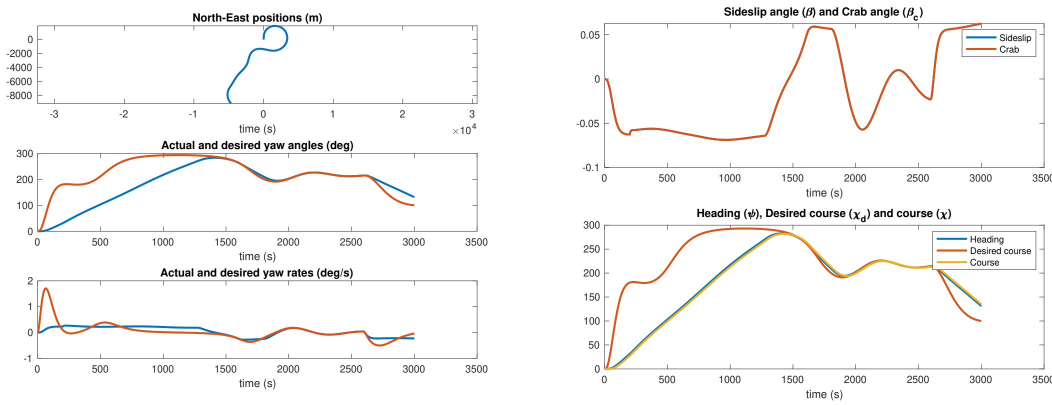


Figure 2: Comparison between ship and straight line paths.



(a) (b)

Figure 3: Simulated without current.

2.4 Problem d)

Implementing integral line-of-sight (ILOS) can be done through equation 12.108 and 12.109 in Fossen

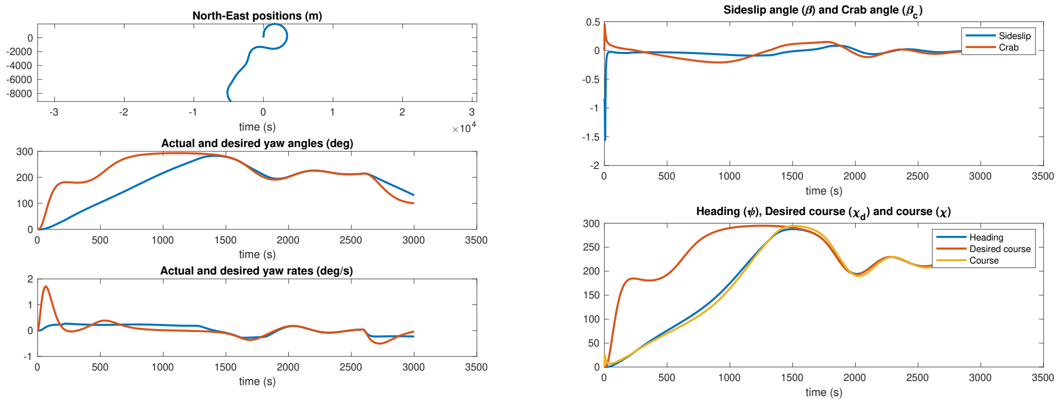
$$\begin{aligned} \psi_d &= \pi_p - tan^{-1}(K_p y_e + K_i y_{int}) \\ \dot{y}_{int} &= \frac{\Delta y_e}{\Delta^2 + (y_e + \kappa y_{int})^2} \end{aligned} \tag{5}$$

Here, Δ is the look-ahead-distance, y_{int} is an internal state, y_e is the cross-track error, K_i is the integral gain given as $K_i = \kappa K_p$. $\kappa > 0$ is a design parameter and K_p is the proportional gain, given as $K_p = 1/\Delta$.

In order to use the internal state in the ILOS we initialize y_{int} and use it as input in the guidance law. Outputs are the desired course and the internal state derivative, $\dot{y}_{int}$. Finally Euler integration is used to obtain y_{int} from $\dot{y}_{int}$.

Simlating with ILOS gave the following response:

Using direct crab compensation has the advantage of being easy, computationally. However it



(a) (b)

Figure 4: Simulated with current.

requires a measurement of the crab angle. It is also prone to modelling errors and measurement errors.

On the other hand, ILOS is more computationally heavy and may be prone to wind up, but comes with advantages. When using ILOS we do not need measurements on crab angle. Also, because of using the integral effect, we handle disturbances, such as measurement noise and unmodelled effects in the model. This means that the ILOS will outperform crab angle compensation when accounting from waves, whihch we have not done so far.

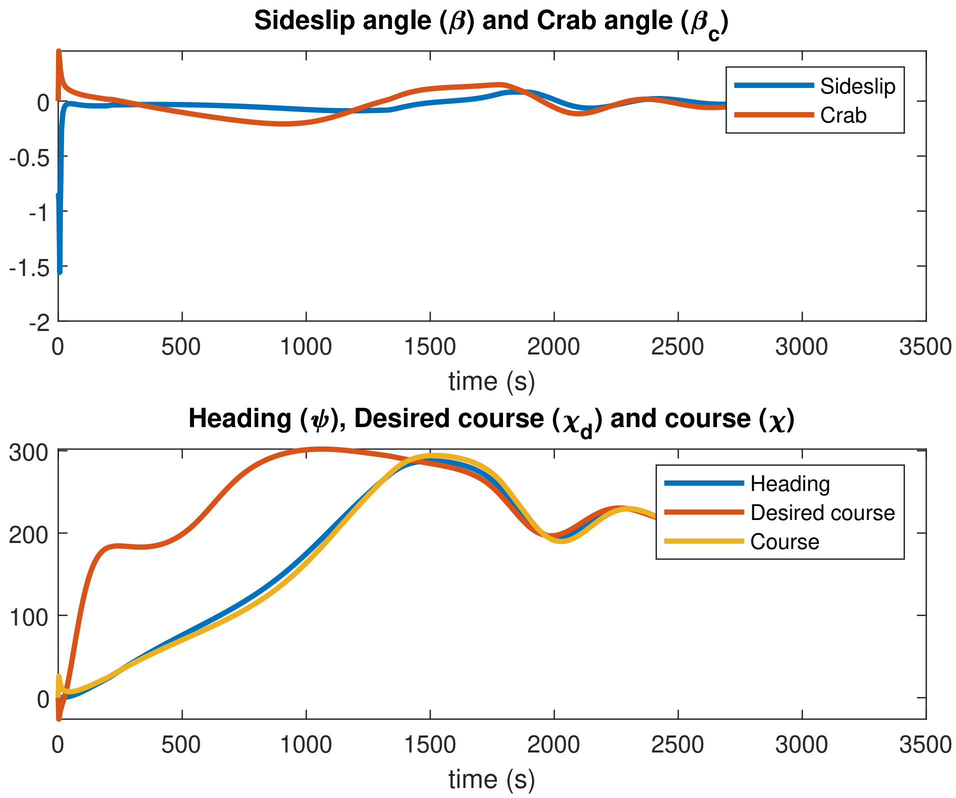


Figure 5: Course, heading and desired course, with sideslip and crab angle. Course angle was obtained through crab angle compensation.

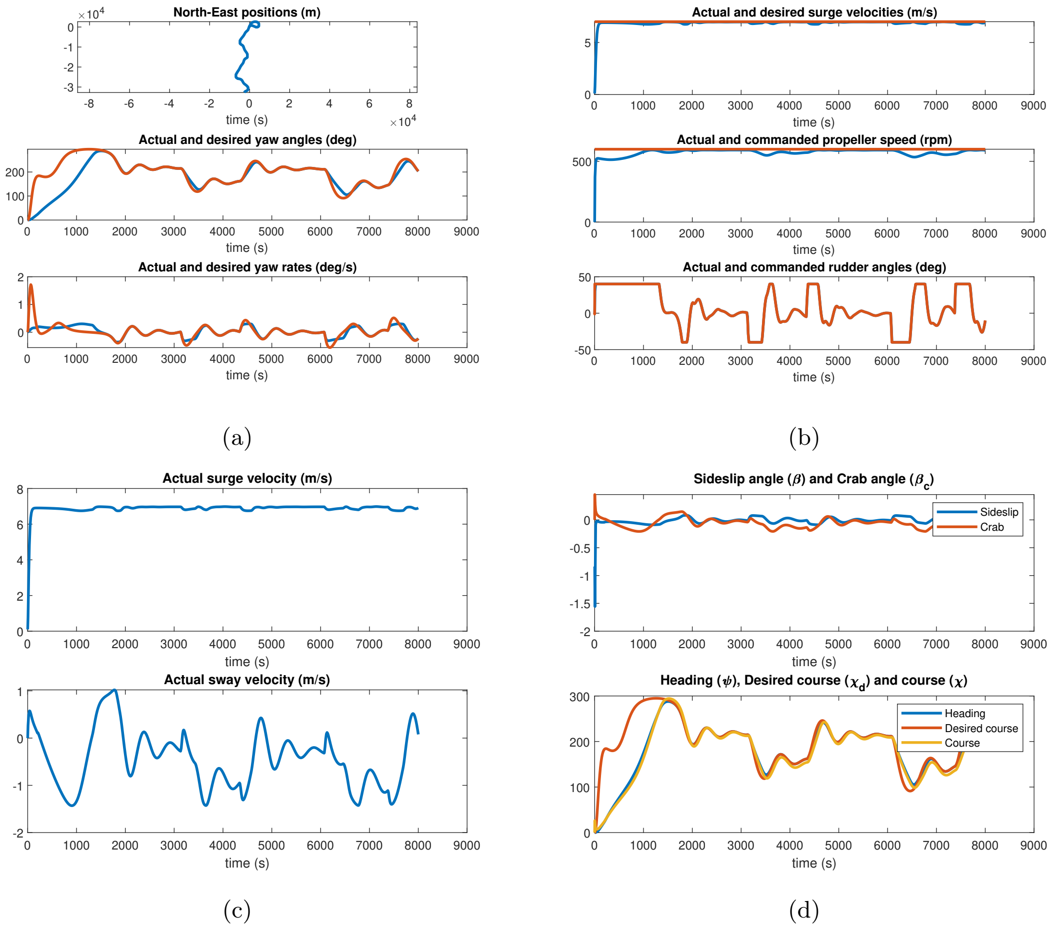


Figure 6: Ship performance with ILOS guidance on straight line path following.

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
```

```matlab
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);
```

```matlab
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
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;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MAIN LOOP
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
simdata = zeros(Ns+1,17);                 % table of simulation data

for i=1:((Ns+1)*8)
    t = (i-1) * h;                        % time (s)

    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 = eta(3) - xd(1);
    Δ_c = - ( Kp * (eta(3) - xd(1)) + Kd * (nu(3) - 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];

    % 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);

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,Δ,t,Δ_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')
```

A.2 los_guidancelaw.m

```
%% LOS guidance law

% By Magnus Dyre-Moe, Patrick Nitschke and Siawash Naqibi

%% Main

function chi_d = los_guidancelaw(x, y, start_point, end_point, Δ)
    % Input :   actual position - x and y
    %           start position for line segment
    %           end position for line segment
    %           lookaheaddistance
    % Returns : Desired course angle
    x_2 = end_point(1);
    y_2 = end_point(2);
    x_1 = start_point(1);
```

```
    y_1 = start_point(2);


    pi_p = atan2(y_2 - y_1, x_2 - x_1);
    Kp = 1 / Δ;

    % Cross-track error
    y_e = -(x-x_1) * sin(pi_p) + (y-y_1) * cos(pi_p);

    % Desired course
    chi_d = wrapTo2Pi( pi_p - atan(Kp * y_e) );
end
```

A.3 integral_los_guidancelaw.m

```
%% LOS guidance law

% By Magnus Dyre-Moe, Patrick Nitschke and Siawash Naqibi

%% Main

function [psi_d, y_int_dot] = integral_los_guidancelaw(x, y, start_point, end_point, Δ, y_int)
    % Input :   actual position - x and y
    %           start position for line segment
    %           end position for line segment
    %           lookaheaddistance
    % Returns : Desired course angle
    x_2 = end_point(1);
    y_2 = end_point(2);
    x_1 = start_point(1);
    y_1 = start_point(2);

    kappa = 0.2;
    Kp = 1 / Δ;
    Ki = kappa * Kp;
    pi_p = atan2(y_2 - y_1, x_2 - x_1);

    % Cross-track error
    y_e = -(x-x_1) * sin(pi_p) + (y-y_1) * cos(pi_p);

    % Internal state
    y_int_dot = Δ * y_e / ( Δ^2 + (y_e + kappa*y_int)^2 );

    % Desired heading angle
    psi_d = wrapTo2Pi( pi_p - atan(Kp * y_e + Ki * y_int) );
end
```

References