MAE 3260 Final Groupwork

Group 3

Title: RC Sailboat Path-Following
Subtitle: Implementing Non-Linear Control Systems for Environment w/ Disturbances

Outline:
Page 1: cover page
Pages 2-9: any format you want in these pages, just include a few section headings in this outline)
Page 10: References

Abstract: This project models an rc sailboat and implements a control algorithm to guide the path of the sailboat along a predetermined set of waypoints. We decided on this project both because of a shared interest in sailing and because we were curious to explore the implementation of a control model for a more complex, non-linear system. The scope of the project extends from the definition of various parameters involved in the control of the boat, the modelling of system and performance constraints, and then to control system implementation and animation in MATLAB.

Students/Roles:

Student	Task/Role	Portfolio
Jake Dick (jdd249)	Assisted with the development of the script and developed constraints for the boat behavior.	https://github.com/Cornell-MAE-UG/fall-2025-portfolio-jdd249/blob/main/_projects/2025-sailboat-control-algorithm.md
Julius Goldberg (jlg445)	Developed the wind inputs using the Dryden Turbulence model. Also wrote next steps and contributed to the control law.	https://github.com/Cornell-MAE-UG/fa25-portfolio-Julius-Goldberg/blob/main/_projects/2025-MAE3260.md
Will Hutter (wdh64)	Developed the script for the sailboat model.	https://cornell-mae-ug.github.io/spring-2025-portfolio-wdh64/
Emma Moore (ejm338)	Explained the broader context of the model and went step by step of the model components.	

MAE 3260 concepts used:

- ODEs
- TFs
- Block Diagrams
- Steady State Behavior
- Feedback Control Law

- Command Following

Context

A sailboat is controlled via rudder, the steering mechanism of the boat, and the sail, the power mechanism of the boat. Both controls utilize lift to move the boat. In general, the rudder controls the direction change and speed of direction change, and the sail angle is tuned based on wind direction and heading, with the goal of maximizing lift.

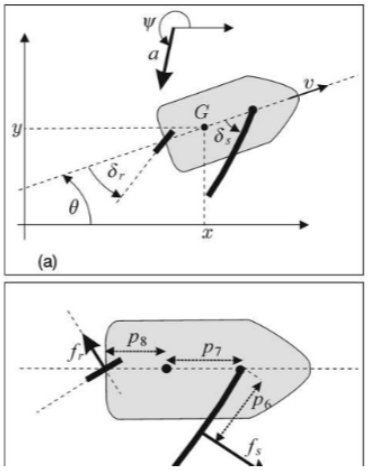
Sailboats experience effects from many different variables in real-life situations. Any variations in wind and water conditions will affect the sailboat's heading and speed. Examples of these include: wind direction, wind speed, wind turbulence, water turbulence, waves, current, etc. In the reference model, the main variables considered in the system are wind speed and direction. Wind behavior in real-life can be highly dynamic, so we explore modeling wind variations, including turbulence, which would cause disturbance.

Sailboats also have unique limitations and constraints compared to a gas or electric powered vehicle. Since the boat utilizes lift for propulsion, certain rudder and sail angles can induce significant drag and affect sailboat performance. For example, a heading that is too close to the wind will call for a corresponding sail angle that induces too much drag to allow for forward motion. Due to this, there is a range of headings that a sailboat cannot utilize. This is also true for rudder angles, where angles that are too high will induce significant drag.

System Variables and Parameters

Name	Description
(x, y, θ)	boat position and orientation
v	boat velocity
ω	rate of rotation
f_s	wind force on the sail
f_r	water force on the rudder
δ_v	angle between the sail and the longitudinal axis
a	wind speed
ψ	wind orientation
σ	sheet tension indicator: $\sigma \geq 0$ means no sheet tension, else the sail is efficient
δ_s	maximum angle between the sail and the boat axis
δ_r	angle between the rudder and the boat axis

Name	Description
$\alpha_d = 0.5$	drift coefficient
$\alpha_v = 100$	longitudinalfriction coefficient
$\alpha_\omega = 6000$	rotational friction coefficient
$\alpha_s = 1750$	lift coefficient for the sail
$\alpha_r = 250$	lift coefficient for the rudder
$p_6 = 2m$	geometric property (see figure 2)
$p_7 = 2m$	geometric property (see figure 2)
$p_8 = 4m$	geometric property (see figure 2)
$M = 200kg$	mass
$J = 1000kg.m^2$	inertia



The system variables and parameters we elected to use were taken directly from the Clement research paper. We chose to model a 200kg sailboat with the geometric parameters described above. While these geometric parameters aren't representative of every sailboat, the conclusions we draw from our control algorithm should be generalizable to boats of varying performance and scales.

Control Algorithm Constraints

Before implementing and modelling the control algorithm of the sailboat, we must set boundaries on the control algorithm based on practical limitations of sailboat control.

We defined values for δ_{s_max}, the maximum allowable angle between the sail and the boat axis, and δ_{r_max}, the maximum angle between the rudder and boat axis. Based on research of typical sailboat performance, we opted for a δ_{s_max} of 90 degrees. It is unlikely that this constraint ever bounds our performance unless we are sailing directly downwind, but it is an important practical consideration to prevent the sail from hitting the stays (guy wires supporting the mast). For the maximum angle between the rudder and the boat axis, δ_{r_max}, we selected a value of 30 degrees. Beyond this value, the rudder will hit diminishing returns of steering force and will induce significant drag that we do not account for in our model.

Another constraint we discussed is the max sail angle (i.e., how close to the wind we will allow our boat to sail). Typical values for this parameter range from 30 - 60 degrees depending on sailboat

characteristics, with performance sailboats achieving lower angles. We chose not to define a constraint for maximum sail angle because this can be determined by the other parameters of our system. For example, if our control algorithm commands a direction 10 degrees from the wind, the boat will be “in-irons,” and the model will implicitly prevent forward progress.

Wind Characterization

To account for the complexity of actual wind conditions, we decomposed the wind into two different layers, a constant base wind and then a turbulent layer. Our model requires inputting a base wind speed and direction, and then gives the option to overlay colored noise in addition with a 2-D Dryden Wind Turbulence Model. This model treats components of continuous gusts

This model takes in parameters for the wind, turbulence, and timespan and then outputs values for the longitudinal and cross-wind turbulence contribution.

Inputs		
Parameter	Definition	Rationale
seed	The grounding point for a random number generator	Repeatable random outputs for testing
N	The number of cycles to run the model for	To set the length of the simulation
dt	The time between cycles	To define the quality of the simulation
u_{base}	The speed of the base wind	The basis for the boat propulsion and turbulence intensity
TI	The ratio of the turbulence RMS speed to the mean wind speed	Rather than having a separate value for the velocity of the turbulence, we use the ratio to ground the value.
L_u	The length scale of the longitudinal turbulence	The length scale determines how fast fluctuations evolve. The longitudinal and cross-wind are separate because the two have different ranges of typical values and are encoded differently in the Dryden model.
L_v	The length scale of the cross-wind turbulence	-

Outputs		
Parameter	Definition	Rationale
u_{turb}	Longitudinal turbulence	Isolating the two turbulences from the base wind speed can help us separate the disturbance from the input.
v_{turb}	Cross-wind turbulence	-

It’s quite a lot to include for the purpose of this group work, but the Dryden Model is provided as a power spectral density function, but can be rewritten as an ODE by rewriting the PSD as a transfer function.

$$\Phi(\omega) = |G(j\omega)|^2$$

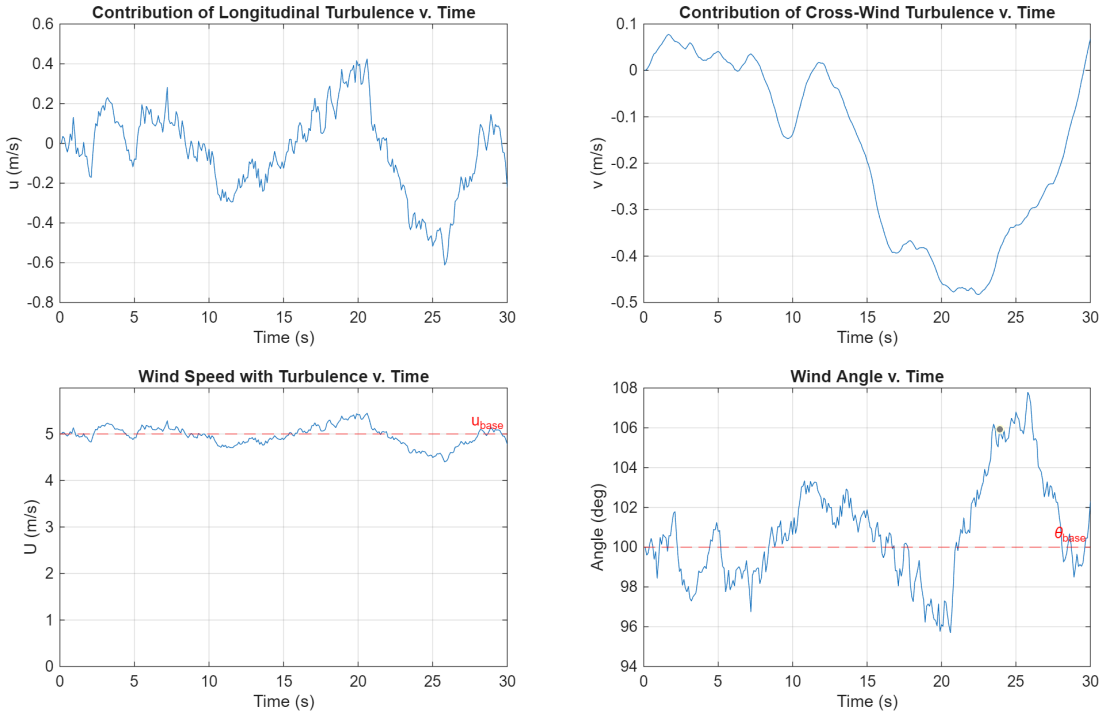
From the transfer functions, we can obtain the ODEs for the longitudinal and cross-wind ODEs:

$$\frac{du}{dt} = -\frac{u_{mean}}{L_u}(u) + \sigma_u\sqrt{\frac{2u_{mean}}{L_u}}\,w(t)$$

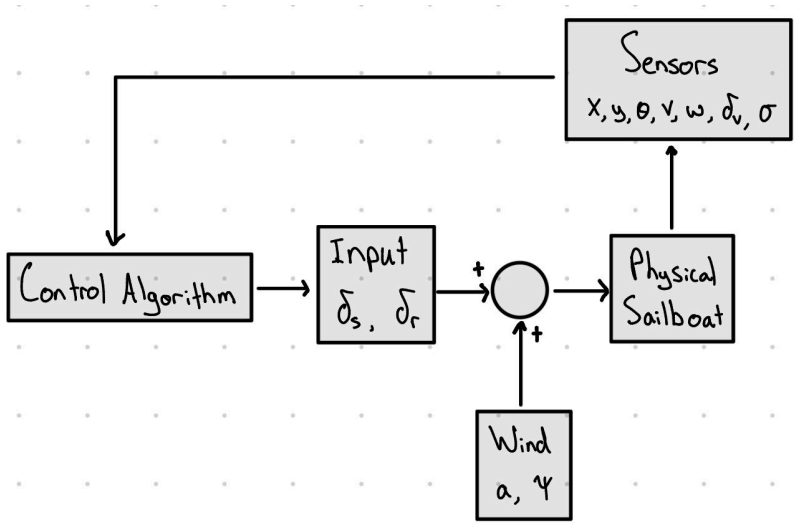
$$\frac{d^2v}{dt^2} + 2\left(\frac{u_{base}}{L_v}\right)\left(\frac{dv}{dt}\right) + \left(\frac{u_{base}}{L_v}\right)^2(v) = \sigma_v\sqrt{\frac{3u_{mean}}{L_v}}\,w(t)$$

These ODEs then get integrated over the periods that are defined by N and dt for the outputs.

As an example, given the inputs of N = 300, dt = 0.1 seconds, U_{base} = 5, $theta_{base}$ = 100 degrees, TI = 0.1, L_u = 100, L_v = 50, seed = 100000



Control Law

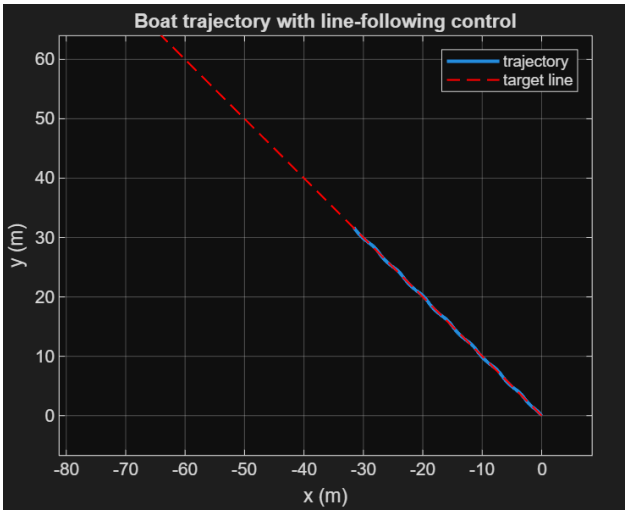


The block diagram drawn above describes the system at a high level. The control algorithm takes as input the sailboat position, velocities, and geometric information about the sail and outputs values for the desired sail position and rudder position. Combining these two variables with information about the wind and the geometric parameters of our sailboat, our model is able to accurately predict how the physical sailboat will respond. Several sensors on the sailboat are used to obtain information about its position, velocity, and sail geometry, which then feeds back into the control algorithm.

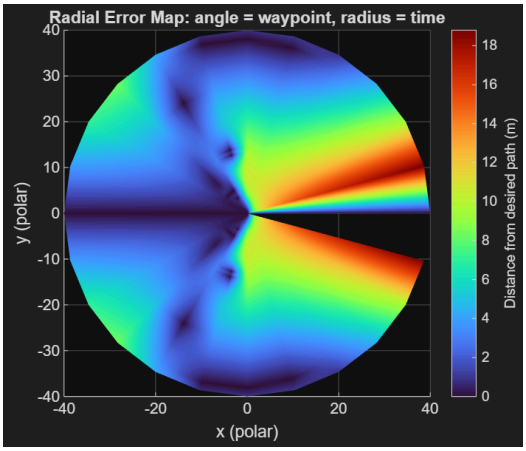
The control algorithm included a value lambda, a performance variable that can be modified based on the needs of the system. It defines how quickly the boat returns to the desired path when it gets pushed off course. The fastest path back to the line would be a line between the desired path and the vehicle perpendicular to the intended path but this is highly impractical. While it wasn't described in terms of proportional, derivative, or integral gain, we noticed that it drove the steady state to zero and so we are guessing that it was a modified PI or PID control.

Results

The matlab code has 5 functions. The core is sailboat_ode.m that defines the equations governing the parameters of the boat based on the inputs. Next, there is sailboat_control.m that defines the feedback control that calculates inputs for the sailboat model. Sailboat_sim.m defines the sailboat parameters and computes the time history of the sailboat_ode.m using inputs from sailboat_control. Animate_sailboat.m visualizes the trajectory of the boat, displaying wind direction, desired heading, actual heading, sail angle and rudder angle. Angle_bath.m creates graphs that are used to generate an error map as the sailboat attempts to travel on paths at different angles to the wind.



Here is a graph of the position of the sailboat attempting to follow a target line. You can see the boat follows the line closely, yet oscillates about the line as it overshoots.



Here is a radial graph showing error over time for courses with different angles relative to the wind. Assuming the wind is blowing from the right to the left, the sailboat is sent on many different courses relative to the wind. The radial component represents time, while the color represents distance from the target path. As you can see the boat had a hard time sailing straight upwind as it got blown off course. At courses plotted straight into the wind, the distance from the desired path is small, but that is because the boat is being blown straight backwards yet does not technically stray from the path.

Next Steps

Given the model that we have, it would be interesting to optimize the behavior of the boat for a certain set of performance parameters. For example, if we could define a situation such as a race versus an autonomous security vehicle, the guiding inputs would be vastly different. Changing the parameters such as lambda or even changing the overall physical abilities of the vehicle would yield interesting insights into the limitations of this model and the potential for different operational efficiencies.

Additionally, this model focused primarily on the heading of the sailboat due to limitations on time and resources. However, in future iterations it would be interesting to add additional waypoints to the model and figure out how to interpolate between them so that the splines are smooth and the boat follows the path as accurately as necessary.

Finally, adding in other autonomous features such as obstacle avoidance or detection and classification could make the model more usable in real-world settings such as security, marine biology study, or ocean debris classification.

MATLAB Code

```
function [t, X, p] = sailboat_sim
  clc; clear; close all;
   %--------- Parameters from the paper -----------------
  % Longitudinal drift (wind pushing hull directly)
  p.ad  = 0.05;      % was 0.5: MUCH less pure wind drift
  % Hydrodynamic / aerodynamic coefficients
  p.av  = 120;       % longitudinal drag (slightly higher to keep speed sane)
  p.aw  = 8000;      % yaw damping (slightly higher for directional stability)
  p.as  = 2200;      % sail force gain (more drive from sail)
  p.ar  = 600;       % rudder effectiveness (slightly more authority)
  p.p6  = 2;
  p.p7  = 2;
  p.p8  = 4;
  p.M   = 200;
  p.J   = 1000;
  % Wind
  p.a_wind = 8;      % you can also try 6 instead of 8 if you want it gentler
  p.psi    = 0;
  % Boat geometry (Miniji)
  p.L = 3.65;
  p.B = 0.86;
  % --- Line to follow (waypoints) ---
  p.a = [0; 0];
  r = 100;
  p.theta = deg2rad(135);
  p.b = [r*cos(p.theta); r*sin(p.theta)];
  % --- Rudder control parameters ---
  p.delta_r_max = deg2rad(60);
  p.lambda      = 0.4;
  % --- Sail control parameters ---
  p.beta = deg2rad(40);              % optimal sail angle at abeam wind
  p.q    = log(pi/(2*p.beta)) / log(2);
  p.delta_s_max = pi/2;              % global mechanical sheet limit
```

```
    % Desired line heading
  phi = atan2(p.b(2) - p.a(2), p.b(1) - p.a(1));   % here: ~pi/2
  %--------- Initial state [x y theta v omega] ----------
  % Start slightly below the first waypoint, on the line, pointing along it
  x0 = [ p.a(1);         ...  % x  = -10  (on the line)
        p.a(2);    ...  % y  =  5-10 = -5 (below waypoint A)
        phi;           ...  % theta aligned with the trajectory
        4;             ...  % moderate initial speed (m/s)
        0 ];               % no initial angular rate
  %--------- Integrate system ---------------------------
  tspan = [0 30];
  [t, X] = ode45(@(t,x) sailboat_ode(t, x, p), tspan, x0);
  %--------- Trajectory plot ----------------------------
  figure;
  plot(X(:,1), X(:,2), 'LineWidth', 2); hold on; grid on; axis equal;
  plot([p.a(1) p.b(1)], [p.a(2) p.b(2)], 'r--', 'LineWidth', 1);
  xlabel('x (m)'); ylabel('y (m)');
  title('Boat trajectory with line-following control');
  legend('trajectory','target line');
  %--------- State variable plots -----------------------
  figure;
  subplot(3,1,1)
  plot(t, X(:,4), 'LineWidth', 1.5); grid on
  ylabel('v (m/s)')
  subplot(3,1,2)
  plot(t, X(:,5), 'LineWidth', 1.5); grid on
  ylabel('\omega (rad/s)')
  %--------- Log & plot controls ------------------------
  delta_r_log = zeros(size(t));
  delta_s_log = zeros(size(t));
  for k = 1:numel(t)
     [delta_r_log(k), delta_s_log(k)] = sailboat_control(X(k,:).', p);
  end
  subplot(3,1,3)
  plot(t, rad2deg(delta_r_log), t, rad2deg(delta_s_log), 'LineWidth', 1.5)
  grid on; xlabel('time (s)')
  legend('\delta_r','\delta_s')
end


function dxdt = sailboat_ode(t, x, p)
  % Unpack state
  theta = x(3);
  v     = x(4);
  omega = x(5);
  % Environment
  a   = p.a_wind;
  psi = p.psi;
```

```
    % State-feedback control
    [delta_r_cmd, delta_s_cmd] = sailboat_control(x, p);
    % Clamp sheet
    ds = max(min(delta_s_cmd, p.delta_s_max), -p.delta_s_max);
    dr = delta_r_cmd;
    % Sheet tension indicator
    sigma = cos(theta - psi) + cos(ds);
    % Aerodynamic sail angle
    if sigma <= 0
        delta_v = pi - theta + psi;         % luffing (aligned with wind)
    else
        delta_v = sign(sin(theta - psi)) * ds;
    end
    % Forces
    f_s = p.as * a * sin(theta - psi + delta_v);
    f_r = p.ar * v * sin(dr);
    % Boat dynamics
    xdot     = v * cos(theta) + p.ad * a * cos(psi);
    ydot     = v * sin(theta) + p.ad * a * sin(psi);
    thetadot = omega;
    vdot = ( f_s * sin(delta_v) ...
           - f_r * sin(dr) ...
           - p.av * v^2 ) / p.M;
    omegadot = ( f_s * (p.p6 - p.p7 * cos(delta_v)) ...
               - p.p8 * f_r * cos(dr) ...
               - p.aw * omega ) / p.J;
    dxdt = [xdot; ydot; thetadot; vdot; omegadot];
end
function [delta_r_cmd, delta_s_cmd] = sailboat_control(x, p)
% SAILBOAT_CONTROL  Line-following rudder + sail trim controller
%
%   [delta_r_cmd, delta_s_cmd] = sailboat_control(x, p)
%
% x = [x; y; theta; v; omega]
% p fields used here:
%   a, b          : waypoints (2x1) defining the target line
%   delta_r_max   : max rudder deflection (rad)
%   lambda        : line-following compromise parameter (0–1)
%   psi           : wind direction (rad, inertial frame)
%   q             : sail-cardioid exponent (see paper)
%   delta_s_max   : mechanical sheet limit (rad)
    % ----------------- Unpack state -----------------
    pos   = x(1:2);    % boat position [x; y]
    theta = x(3);      % boat heading
    % ================================================
    % 1) RUDDER CONTROL – EXACTLY AS IN THE PAPER
    %    δ_r = δ_r_max ( λ sin γ + (1−λ) sign(e) )
    %    with γ = φ − θ
    % ================================================
```

```
    % Line geometry
    a = p.a(:);
    b = p.b(:);
    u = b - a;                      % direction vector of the line
    % Line heading φ
    phi = atan2(u(2), u(1));
    % Heading error γ = φ − θ  (wrapped to [-π, π])
    dphi  = phi - theta;
    gamma = atan2(sin(dphi), cos(dphi));
    % Signed cross-track error e = ((m−a) × u) / ||u||
    Rperp = [0 -1; 1 0];              % 90° CCW rotation
    e     = (pos - a).' * (Rperp * u) / max(norm(u), 1e-6);
    % Rudder law
    delta_r_cmd = p.delta_r_max * ...
        ( p.lambda * sin(gamma) + (1 - p.lambda) * sign(e) );
    % Hard saturation (just in case)
    delta_r_cmd = max(min(delta_r_cmd,  p.delta_r_max), ...
                  -p.delta_r_max);
    % ================================================
    % 2) SAIL CONTROL – CARDIOID LAW (Eq. 5)
    %    δ_s,max(ψ−θ) = (π/2)*((cos(ψ−θ)+1)/2)^q
    % ================================================
    % Relative wind angle (wind w.r.t. boat heading)
    rel = p.psi - theta;                % ψ − θ
    rel = atan2(sin(rel), cos(rel));       % wrap
    base       = (cos(rel) + 1) / 2;       % in [0,1]
    delta_s_max = (pi/2) * (base.^p.q);     % Eq. (5)
    % Commanded sheet angle (non-negative)
    delta_s_cmd = min(delta_s_max, p.delta_s_max);
end

function animate_sailboat
    % Run simulation and get parameters
    [t, X, p] = sailboat_sim;
    % Unpack geometry
    L = p.L;
    B = p.B;
    %----------------- Hull geometry (boat frame) -----------------
    hull_x = L/2 * [ 1,  0.2, -1,  0.2,  1 ];
    hull_y = B/2 * [ 0,  1.0,  0, -1.0,  0 ];
    hull_local = [hull_x; hull_y];
    mast_local  = [0.1*L; 0];
    L_sail      = 0.8*L;
    rudder_root_local = [-L/2; 0];
    L_rudder          = 0.4*L;
    % Zoom half-window around the boat
    zoom_range = 2.5 * L;
    % Desired heading = line direction φ
    a = p.a(:);
```

```
b = p.b(:);
phi = atan2(b(2)-a(2), b(1)-a(1));    % desired course heading
%----------------- Figure setup ------------------------------
figure; clf;
hold on; grid on; axis equal;
plot(X(:,1), X(:,2), 'Color', [0.8 0.8 0.8], 'LineWidth', 1);
xlabel('x (m)');
ylabel('y (m)');
title('Sailboat Animation with Heading Vectors');
%----------------- Graphics objects --------------------------
hull_patch = patch(nan, nan, [0.85 0.85 0.85], 'EdgeColor', 'k');
sail_line  = plot(nan, nan, 'r', 'LineWidth', 2);
rud_line   = plot(nan, nan, 'b', 'LineWidth', 2);
wind_arrow = quiver(nan, nan, nan, nan, 'Color', [0 0.7 0], ...
                    'AutoScale', 'off', 'LineWidth', 2);
sail_pivot_dot   = plot(nan, nan, 'ko', 'MarkerFaceColor', 'k', 'MarkerSize', 6);
rudder_pivot_dot = plot(nan, nan, 'ko', 'MarkerFaceColor', 'k', 'MarkerSize', 6);
% --- Desired heading (cyan, dashed) ---
desired_len = 1.2 * L;
desired_heading_vec = plot(nan, nan, '--', ...
                           'Color', [0 1 1], 'LineWidth', 2);
% --- Actual heading (magenta, solid) ---
actual_heading_vec = plot(nan, nan, '-', ...
                          'Color', [1 0 1], 'LineWidth', 2);
legend([sail_line, rud_line, wind_arrow, desired_heading_vec, actual_heading_vec], ...
   {'Sail','Rudder','Wind','Desired Heading','Actual Heading'}, ...
   'Location','best');
%----------------- Animation loop -----------------------------
N = numel(t);
skip = max(1, round(N/400));
wind_len = L;
for k = 1:skip:N
   pos   = X(k,1:2).';
   theta = X(k,3);
   R = [cos(theta) -sin(theta);
        sin(theta)  cos(theta)];
   % --- Controls (same as dynamics) ---
   xk = X(k,:).';
   [delta_r_cmd, delta_s_cmd] = sailboat_control(xk, p);
   ds = max(min(delta_s_cmd, p.delta_s_max), -p.delta_s_max);
   % --- Sail aerodynamic angle ---
   sigma = cos(theta - p.psi) + cos(ds);
   if sigma <= 0
      delta_v = pi - theta + p.psi;
   else
      delta_v = sign(sin(theta - p.psi)) * ds;
   end
   % Flip sail just for graphics
   delta_v_geom = delta_v + pi;
```

```
    %--------------- Hull ----------------------------------
    hull_world = pos + R * hull_local;
    set(hull_patch, 'XData', hull_world(1,:), 'YData', hull_world(2,:));
    %--------------- Sail ----------------------------------
    mast_world = pos + R * mast_local;
    sail_tip_world = pos + R * (mast_local + ...
                L_sail * [cos(delta_v_geom); sin(delta_v_geom)]);
    set(sail_line, 'XData', [mast_world(1) sail_tip_world(1)], ...
              'YData', [mast_world(2) sail_tip_world(2)]);
    set(sail_pivot_dot, 'XData', mast_world(1), 'YData', mast_world(2));
    %--------------- Rudder ----------------------------------
    rudder_root_world = pos + R * rudder_root_local;
    rudder_tip_world = pos + R * (rudder_root_local + ...
                    L_rudder * [cos(delta_r_cmd - pi); sin(delta_r_cmd - pi)]);
    set(rud_line,'XData',[rudder_root_world(1) rudder_tip_world(1)], ...
            'YData',[rudder_root_world(2) rudder_tip_world(2)]);
    set(rudder_pivot_dot,'XData',rudder_root_world(1), ...
                    'YData',rudder_root_world(2));
    %--------------- Wind ----------------------------------
    set(wind_arrow,'XData',pos(1),'YData',pos(2), ...
              'UData',wind_len*cos(p.psi), ...
              'VData',wind_len*sin(p.psi));
    %--------------- Desired heading line ----------------------
    x_des = pos(1) + desired_len * cos(phi);
    y_des = pos(2) + desired_len * sin(phi);
    set(desired_heading_vec, 'XData', [pos(1) x_des], ...
                    'YData', [pos(2) y_des]);
    %--------------- Actual heading line ----------------------
    x_act = pos(1) + desired_len * cos(theta);
    y_act = pos(2) + desired_len * sin(theta);
    set(actual_heading_vec, 'XData', [pos(1) x_act], ...
                    'YData', [pos(2) y_act]);
    %--------------- Dynamic camera ---------------------------
    cx = pos(1);
    cy = pos(2);
    axis([ cx - zoom_range, cx + zoom_range, ...
          cy - zoom_range, cy + zoom_range ]);
    drawnow;
  end
end

% run_sailboat_360.m
% Run sailboat simulations for trajectories in all directions
% from the origin (0,0), sweeping 360 degrees, and build a radial
% error map: angle = waypoint direction, radius = time, color = path error.
clear; clc; close all;
%% -------- Base parameters (tuned model) -----------------------------
% Longitudinal drift (wind pushing hull directly)
p.ad  = 0.05;      % much less pure wind drift than 0.5
```

```
% Hydrodynamic / aerodynamic coefficients
p.av  = 120;       % longitudinal drag
p.aw  = 8000;      % yaw damping
p.as  = 2200;      % sail force gain
p.ar  = 600;       % rudder effectiveness
p.p6  = 2;
p.p7  = 2;
p.p8  = 4;
p.M   = 200;
p.J   = 1000;
% Wind
p.a_wind = 8;      % wind speed
p.psi    = 0;      % wind direction (rad, along +x)
% Boat geometry (Miniji)
p.L = 3.65;
p.B = 0.86;
% Rudder control parameters
p.delta_r_max = deg2rad(45);   % max rudder angle
p.lambda      = 0.7;           % mixing between sin(gamma) and sign(e)
% Sail control parameters
p.beta        = deg2rad(40);                  % optimal sail angle at abeam wind
p.q           = log(pi/(2*p.beta)) / log(2);      % cardioid exponent
p.delta_s_max = pi/2;                         % mechanical sheet limit
% Save base parameters
p_base = p;
%% -------- Direction sweep setup ----------------------------------
N_dirs   = 24;                  % number of directions around 360°
R_line   = 80;                  % length of each target line (radius)
angles   = linspace(0, 2*pi, N_dirs+1);
angles   = angles(1:end-1);       % drop duplicate 2π = 0
tspan    = [0 40];              % integration time horizon
colors = jet(N_dirs);            % color map so each heading is distinct
%% -------- Trajectory simulations & plotting ------------------------
figure; hold on; grid on; axis equal;
xlabel('x (m)');
ylabel('y (m)');
title('Sailboat trajectories for 360° waypoint directions from origin');
for i = 1:N_dirs
  % Start from base parameters each time
  p = p_base;
  % Waypoints: A at origin, B on circle of radius R_line
  p.a = [0; 0];
  p.b = R_line * [cos(angles(i)); sin(angles(i))];
  % Desired line heading (phi)
  phi = atan2(p.b(2) - p.a(2), p.b(1) - p.a(1));
  % Initial state [x; y; theta; v; omega]
  % Start exactly at A, oriented along the line, small forward speed
  x0 = [ p.a(1);      ...
        p.a(2);      ...
```

```
        phi;        ...
        0.5;        ...   % small initial speed
        0 ];            % no initial yaw rate
  % Integrate dynamics
  [t, X] = ode45(@(t,x) sailboat_ode(t, x, p), tspan, x0);
  % Plot trajectory
  plot(X(:,1), X(:,2), 'Color', colors(i,:), 'LineWidth', 1.2);
  % Plot the target line segment
  plot([p.a(1) p.b(1)], [p.a(2) p.b(2)], '--', ...
      'Color', colors(i,:), 'LineWidth', 0.8);
end
% Draw origin and wind direction for reference
plot(0, 0, 'ko', 'MarkerFaceColor', 'k', 'MarkerSize', 6);
quiver(0, 0, 20*cos(p_base.psi), 20*sin(p_base.psi), ...
    'k', 'LineWidth', 2, 'MaxHeadSize', 2);
text(20*cos(p_base.psi), 20*sin(p_base.psi), '  Wind', 'FontSize', 10);
axis equal;
%% ====== Radial error data (angle = waypoint direction, radius = time) ======
Nt       = 400;                    % number of time samples
t_uniform = linspace(0, max(tspan), Nt);    % uniform time grid
% Error array: rows = angle index, cols = time index
E = zeros(N_dirs, Nt);
Rperp = [0 -1; 1 0];    % 90° CCW rotation matrix
for i = 1:N_dirs
  % Parameters for this direction
  p = p_base;
  p.a = [0; 0];
  p.b = R_line * [cos(angles(i)); sin(angles(i))];
  % Same initial condition logic
  phi = atan2(p.b(2) - p.a(2), p.b(1) - p.a(1));
  x0  = [ p.a(1); p.a(2); phi; 0.5; 0 ];
  % Run sim
  [t_i, X_i] = ode45(@(t,x) sailboat_ode(t, x, p), tspan, x0);
  % Interpolate positions onto uniform time
  x_i = interp1(t_i, X_i(:,1), t_uniform, 'linear', 'extrap');
  y_i = interp1(t_i, X_i(:,2), t_uniform, 'linear', 'extrap');
  % Line direction vector
  u = p.b - p.a;
  % Cross-track error at each time
  for k = 1:Nt
     pos = [x_i(k); y_i(k)];
     e   = (pos - p.a).' * (Rperp * u) / norm(u);   % signed distance
     E(i,k) = abs(e);   % store absolute distance from desired path
  end
end
% Avoid exactly r = 0 to dodge origin-degenerate cells
t_uniform(1) = 1e-3;
% Build polar grid: angles (theta) x time (radius)
[TH, RR] = meshgrid(angles, t_uniform);   % (Nt, N_dirs)
```

```
% Convert polar -> Cartesian for plotting
[Xp, Yp] = pol2cart(TH, RR);
% E is (N_dirs, Nt) -> transpose to (Nt, N_dirs) to match TH/RR
E_grid = E.';   % (Nt, N_dirs)
%% ====== Radial error plot: theta = angle, r = time, color = |error| ====
figure;
surf(Xp, Yp, zeros(size(E_grid)), E_grid, ...
    'EdgeColor', 'none', 'FaceColor', 'interp');
axis equal tight;
colormap turbo;
cb = colorbar;
cb.Label.String = 'Distance from desired path (m)';
title('Radial Error Map: angle = waypoint, radius = time');
xlabel('x (polar)');
ylabel('y (polar)');
view(2);   % top-down 2D view
```

References

B. Clement, “Control Algorithms for a Sailboat Robot with a Sea Experiment,” *Proc. 9th IFAC Conf. Control Applications in Marine Systems*, Osaka, Japan, Sept. 17–20, 2013, pp. 19–10, doi: 10.3182/20130918-4-JP-3022.00061.

P. Abichandani, D. Lobo, G. Ford, D. Bucci and M. Kam, "Wind Measurement and Simulation Techniques in Multi-Rotor Small Unmanned Aerial Vehicles," in IEEE Access, vol. 8, pp. 54910-54927, 2020, doi: 10.1109/ACCESS.2020.2977693.