MAE 3260 Final Group Work: Exploring a System of Interest

Report

Title: BALE the Boat

Topic of Interest: Autonomous Sailboats

Abstract: We plan to model an autonomous sailboat set to travel a fixed trajectory and speed, whose disturbance, caused by either wind, is represented by an angle deterring the boat from its course. We chose this project to understand the importance of feedback control in a system that, realistically, is too complex to model given its own physical limitations and environmental factors. Our simplified model allows us to define a PD controller and create MATLAB scripts to visualize our developed state space and actuator model for implementing line-following sailing conditions.

Students/Roles:

Student	Task/Role	Portfolio
Emily Gamez-Garcia	MATLAB Simulation	https://cornell-mae-ug.github.io/fa25-portfolio-emilygamez-garcia/
Alison Chavez	Model Definition and Control Block Diagram	TBD
Lilly Varon	Define State Space	https://cornell-mae-ug.github.io/fa25-portfolio-lgv22/projects/systems-project/
Bella Pensabene	MATLAB Animation	https://cornell-mae-ug.github.io/fa25-portfolio-isabellapensabene/projects/systems-system-design/

List of MAE 3260 concepts or skills used in this group work:

- Models:
  - ODEs
  - State space
  - Block diagrams
- Active control:
  - Feedback control law
  - PD controllers

Defining the Model

For our final group work, our team decided to investigate the dynamics and controls of an autonomous sailboat. Specifically, we aim to implement a model that enables autonomous line following by controlling the rudder angle of the sailboat through a rudder actuator. We considered the hydrodynamic and aerodynamic forces acting on the rudder, hull, and sail of the boat. Based on the 3-DOF model described in [1, 2], we defined our model in Figure 1 which focuses on rotational motion.

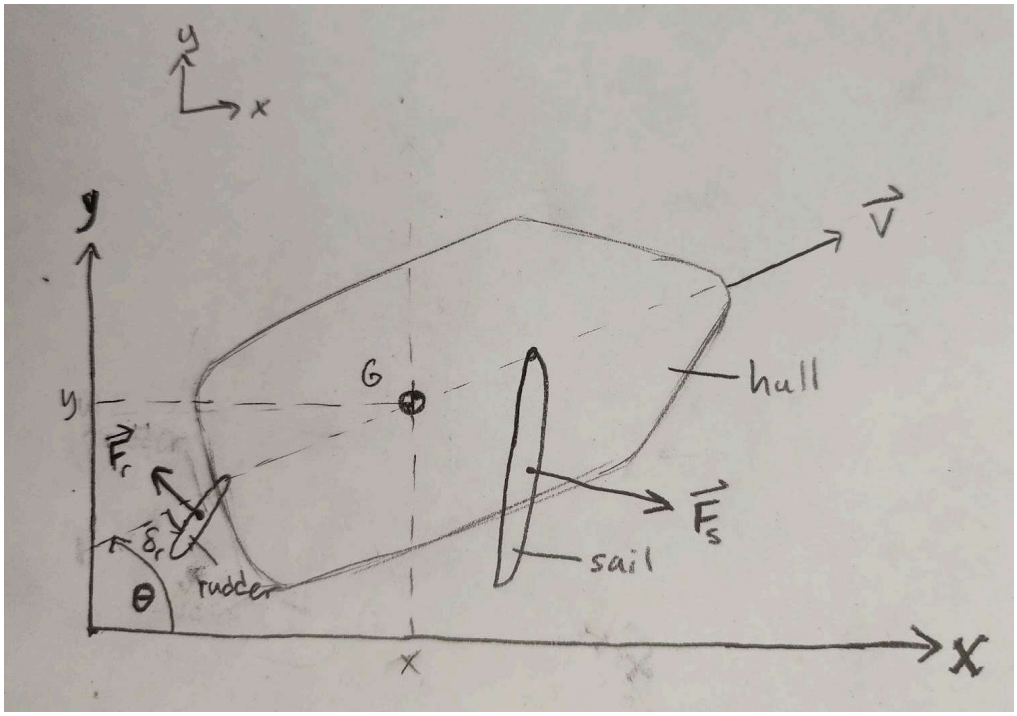

Figure 1.

We defined the center of mass as the sum of the center of masses of the fixed rudder, hull, and sail masses. This is shown as G in the model. The position of the sailboat is defined as the x and y coordinates of the center of mass. The two forces we modeled as acting on our sailboat are *Fs*, the wind force on the sail, and *Fr*, the water force acting on the rudder. The heading angle of the boat from the x-axis is defined as θ. The forward speed of the sailboat is *v*, which acts along the line intersecting G and the angle θ. The portion of the sailboat that is actuator controlled is the rudder angle δ_r, which affects the heading of the boat. Not pictured in Figure 1 but essential to the dynamics of the system is the yaw rate, ω, which is the derivative of change in θ.

In order to simplify the dynamics of our system, we assumed negligible sway in the boat. We also assumed constant roll and pitch of the sailboat, such that solely yaw remains dynamic. In addition, for the sake of simplicity, we neglect controlling the sail angle so that we only define a singular control law.

State Space Model

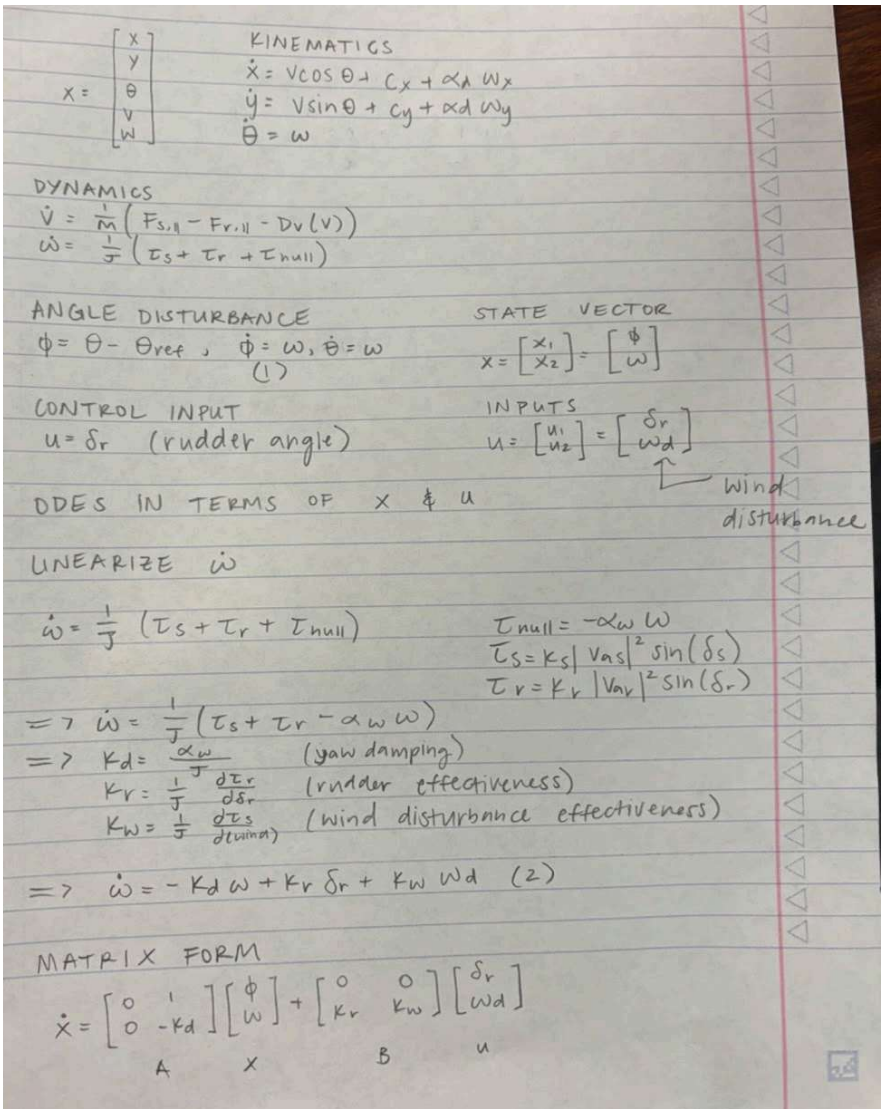

Figure 2.

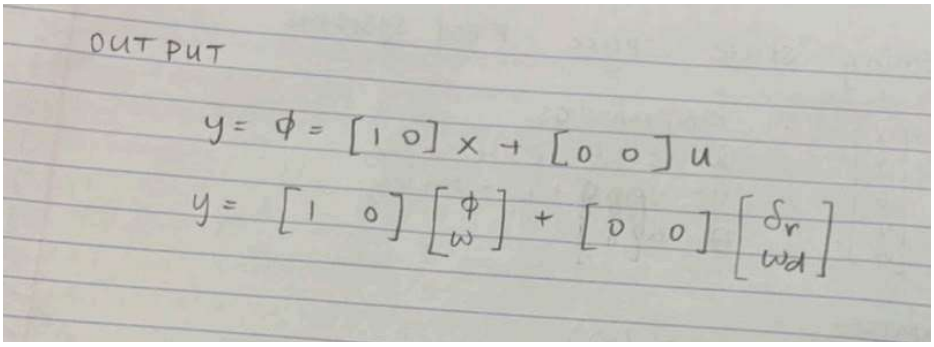

Figure 3.

To build the control model for the sailboat, I focused on the rotational dynamics that determine its heading. The full 3-DOF model describes motion in $x,\ y,\ \theta,\ v,\ \omega$ [1], [2], but only the heading angle θ and yaw rate ω matter for line following. Since we assume constant forward speed, negligible sway, and fixed sail trim, the system reduces cleanly to a two-state representation. I first defined the angle disturbance as:

$$\phi = \theta - \theta_{ref}$$

Which measures deviation from the desired trajectory. Because the kinematics give $\theta \cdot = \omega$, this immediately produces the first state equation:

$$\phi \cdot = \omega$$

Next, I simplified the yaw dynamics. The full model expresses yaw acceleration as:

$$\omega \cdot = \frac{1}{J}(\tau_s + \tau_r - \alpha_\omega \omega)$$

where the sail and rudder torques depend on apparent wind and water velocities. By linearizing these torques around nominal operating conditions, I obtained effective constants: k_d for yaw damping, k_r for rudder effectiveness, and k_w for wind disturbance. This yields:

$$\omega \cdot = - k_d \omega + k_r \delta_r + k_\omega w_d$$

Combining the two equations gives the state vector:

$$x = [\phi \quad \omega]^T, u = \begin{bmatrix} \delta_r & w_d \end{bmatrix}^T$$

And the final state space form:

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & -k_d \end{bmatrix} x + \begin{bmatrix} 0 & 0 \\ k_r & k_w \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x$$

This reduced model captures the essential heading dynamics, separating control input from wind disturbance and providing the foundation for the PD controller used in our simulation.

Control Block Diagram

From the defined state space model of the sailboat, we determined a PD controller for the sail rudder based on the model discussed in class. The reference input is the heading input, $r = \theta_{ref}$, of the boat and the control input is the rudder angle δ_r. The output is the actual heading of the boat, θ, and the derivative term is the yaw rate, ω. Thus, the control law for the sail rudder is: $\delta_r = K_p(r - \theta) - K_D\omega$. The block diagram for the system is shown below as Figure 4.

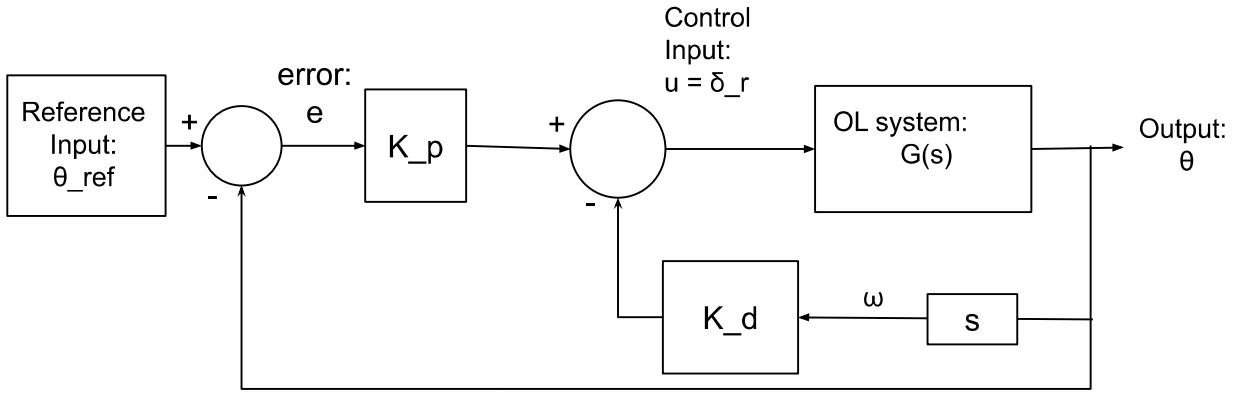

Figure 4.

To determine K_p and K_D, we utilize the results from the state space model, where the characteristic polynomial of the closed loop system becomes $s^2 + (k_d + k_r K_D) + k_r K_p = 0$. Wind disturbance is not included in the polynomial since we consider the $k_w w_d$ term as disturbance that will indirectly be mitigated by the other gains. This also allows us to implement a PD controller design compared to a PID controller, since wind disturbance affects the rudder angle to much lesser degree than the the other variables, as can be seen by our dynamics. Equating the above polynomial to the general form $s^2 + 2\zeta\omega_n s + \omega_n^2 = 0$ yields the formulas in the table below, with ω_n being the natural frequency of the system and ζ being the damping ratio.

Gain	**Formula**
K_p	$K_p = (\omega_n^2) / k_r$
K_d	$K_d = (2\zeta\omega_n - k_d) / k_r$

Matlab Script and Simulation

To simulate the sailboat, we compiled our state space and PD controller models to generate a simple graph of the heading angle, θ, over time and against our chosen θ_{ref}. To further simplify the model, we noted that changes in the rudder angle δ_r directly affect the heading angle θ such that $\delta_r = \ddot{\theta}$. In other words, the are no physical limitations involved in the boat receiving direction from the rudder. Additionally, our state space model $X = [\phi\ \omega]^T$ can be rearranged to represent $X = [\theta\ \omega]^T$ by adjusting the angle error, $e = (\theta_{ref} - \theta)$. Running our simulation integrates our state space model to $\dot{X} = [\omega\ \delta_r]^T$, giving us our $\theta(t)$ plot shown in Figure 5 with its main script and ODE script in Figures 6 and 7.

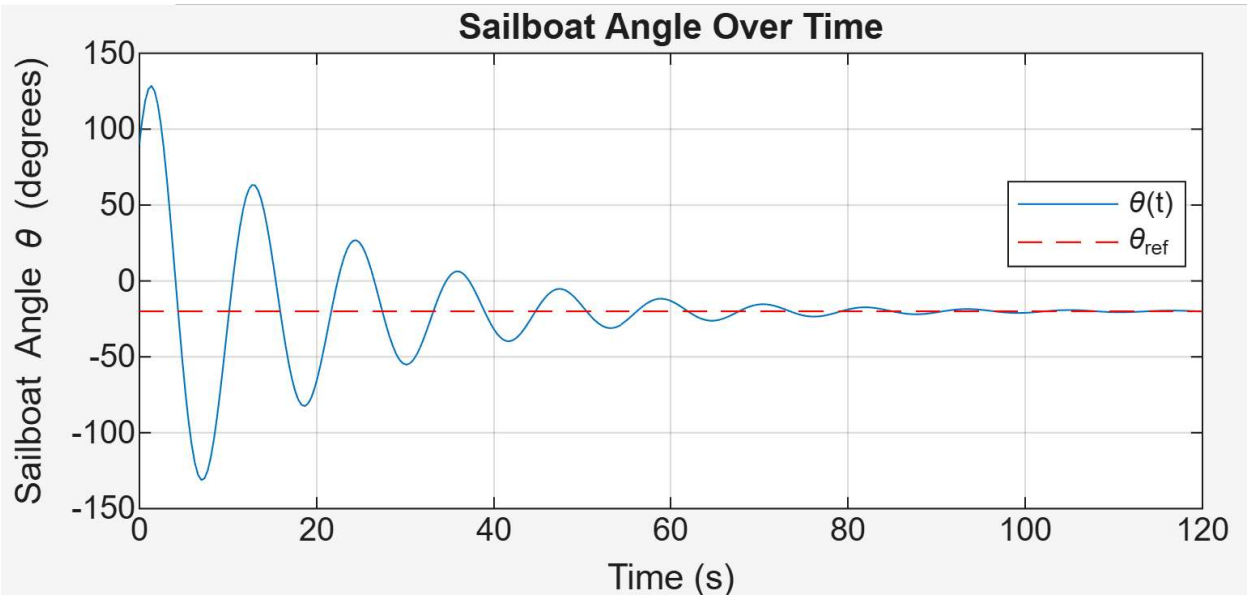

Figure 5.

```
%Controller Gains
Kpc = 0.3; %wn^2/kr;
Kdc = 0.1; %(2*zeta*wn-kd)/kr;

theta_ref = deg2rad(-20); %In radians to use ode45
tspan = [0, 120]; %Set to random time

%Set initial state space conditions
% Recall x = [phi(angle error); omega(yaw rate)], rearrange angle error for theta so x = [theta; omega]
theta_ic = deg2rad(90); %Set to desired angle
omega_ic = 1; %Set to desired yaw
x0 = [theta_ic; omega_ic];

%Call ODE
[t, x] = ode45(@(t, x) sailboatode(t, x, Kpc, Kdc, theta_ref), tspan, x0);

%Get all values for theta and omega from the two x matrix columns
theta = x(:,1);
omega = x(:,2);

%Convert from rad to deg to graph
theta_deg = rad2deg(theta);
omega_deg = rad2deg(omega);
theta_ref_deg = rad2deg(theta_ref)*ones(size(t)); %Create ref angle vector

%Make graph
figure;
plot(t, theta_deg); hold on;
plot(t, theta_ref_deg, 'r--');
legend('\theta(t)', '\theta_{ref}', 'Location','best');
xlabel('Time (s)');
ylabel('Sailboat Angle \theta (degrees)');
title('Sailboat Angle Over Time');
grid on;
```

Figure 6.

```
function dx = sailboatode(t, x, Kpc, Kdc, theta_ref )

%Assign x matrix columns
theta = x(1);
omega = x(2);

%PD controller input (rudder angle)
e = theta_ref - theta;
delta_r = Kpc*e - Kdc*omega;

%Set derivatives
d_theta = omega;
d_omega = delta_r;
dx = [d_theta; d_omega];
end
```

Figure 7.

The figures above prove that our PD controller is effective in steering the sailboat to any desired angle, from any orientation.

Matlab Animation

I created an animation of the sailboat which shows the sailboat's movement and heading relative to the X-axis. The variables available to control include the controller gains, Kpc and Kdc, the initial heading of the boat, initial angular velocity of the boat, the reference input heading, and a constant velocity with which the boat will be traveling. After all of the initial inputs, the script calls the ODE to retrieve the heading and rotational velocity values for the boat throughout the time range [0s, 120s]. Then, it draws the shape of the boat and its sail and makes them into patch objects so they can be more easily moved around and rotated throughout the animation. The animation loop uses the theta values obtained from the ODE to construct a rotation matrix, which is used to rotate the boat and sail. The new positions of these objects are also updated based on the speed and direction of travel. The title is updated to display the time elapsed since the beginning of the boat's travel and its heading/angle of travel relative to the x-axis. Below is a video of the animation with the following initial conditions, reference inputs, and controller gains.

Link to video of animation

Code:

```
clear; close all; clc;
%% Inputs to the system
% Controller gains
Kpc = 0.3;
Kdc = 0.1;
% Initial conditions
theta_ic = deg2rad(90);
omega_ic = 1;
x0 = [theta_ic; omega_ic];
% Reference angle/desired heading
theta_ref = deg2rad(-20);
tspan = [0, 120];
% Boat Speed
speed = 0.3;
%% Use ODE to get roataitonal dynamics
[t, x] = ode45(@(t, x) sailboatode(t, x, Kpc, Kdc, theta_ref), tspan, x0);
theta = x(:,1);    % heading angle (rad)
%%  Add forward motion at constant speed
U = speed;   % constant speed, adjust as desired
dt = diff(t);
dt = [dt; dt(end)];   % match array size
% Integrate x,y position
dx = cumsum(U * cos(theta) .* dt);
dy = cumsum(U * sin(theta) .* dt);
%% Animation Setup
figure('Color','w');
axis equal; hold on; grid on;
xlabel('X'); ylabel('Y');
title('Sailboat Orientation & Motion Animation');
axis([-5 5 -5 5]);
%%  Sailboat Hull & Sail Shape
hull = [-0.6 -0.4;
```

```
         0.6 -0.4;
         1.1  0;
         0.6  0.4;
        -0.6  0.4 ];
sail = [ 0  0.45;
        1.2  0;
        0 -0.45 ];
% Patch objects
hull_patch = patch(hull(:,1), hull(:,2), [0 0.4 1], 'EdgeColor','k');
sail_patch = patch(sail(:,1), sail(:,2), [1 1 1]*0.9, 'EdgeColor','k');
%% Animation Loop
for k = 1:length(t)
   angle = theta(k);
   % Rotation matrix
   Rmat = [cos(angle) -sin(angle);
           sin(angle)  cos(angle)];
   % Rotate hull & sail
   rot_hull = (Rmat * hull')';
   rot_sail = (Rmat * sail')';
   % Translate according to px, py
   rot_hull(:,1) = rot_hull(:,1) + dx(k);
   rot_hull(:,2) = rot_hull(:,2) + dy(k);
   rot_sail(:,1) = rot_sail(:,1) + dx(k);
   rot_sail(:,2) = rot_sail(:,2) + dy(k);
   % Update hull and sail patches
   set(hull_patch, 'XData', rot_hull(:,1), 'YData', rot_hull(:,2));
   set(sail_patch, 'XData', rot_sail(:,1), 'YData', rot_sail(:,2));
   % Update title
   heading_deg = rad2deg(theta(k));
   title(sprintf('Sailboat Motion   t = %.1f s   |   Heading = %.1f°', t(k),
heading_deg));
   % Adjust view to follow boat
   axis([dx(k)-3 dx(k)+3  dy(k)-3 dy(k)+3]);
   drawnow;
end
```

References

[1] B. Clement, “Control algorithms for a sailboat robot with a sea experiment,” IFAC Proceedings Volumes, vol. 46, Osaka, Japan, Sep 2013

[2] Saoud, Hadi & Hua, Minh-Duc & Plumet, Frederic & Amar, Faïz. (2013). Modeling and Control Design of a Robotic Sailboat. 95-110. 10.1007/978-3-319-02276-5_8.

[3] “Autonomous Sailboat ODE” “Using these sources, construct an ODE for an autonomous sailboat traveling with an input speed and trajectory (represented as an angle from true north) and disturbances from wind and water represented as point vectors” prompt. ChatGPT, OpenAI, 1 December 2025, chat.openai.com/chat.