4

4.3

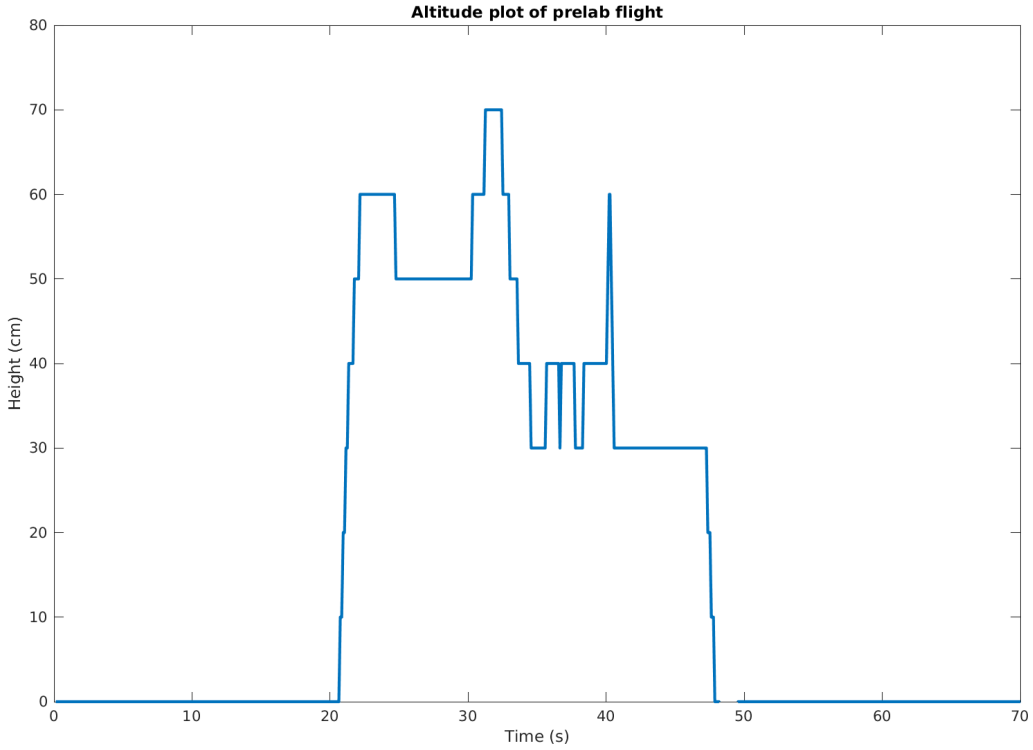

Figure 1: Altitude plot.

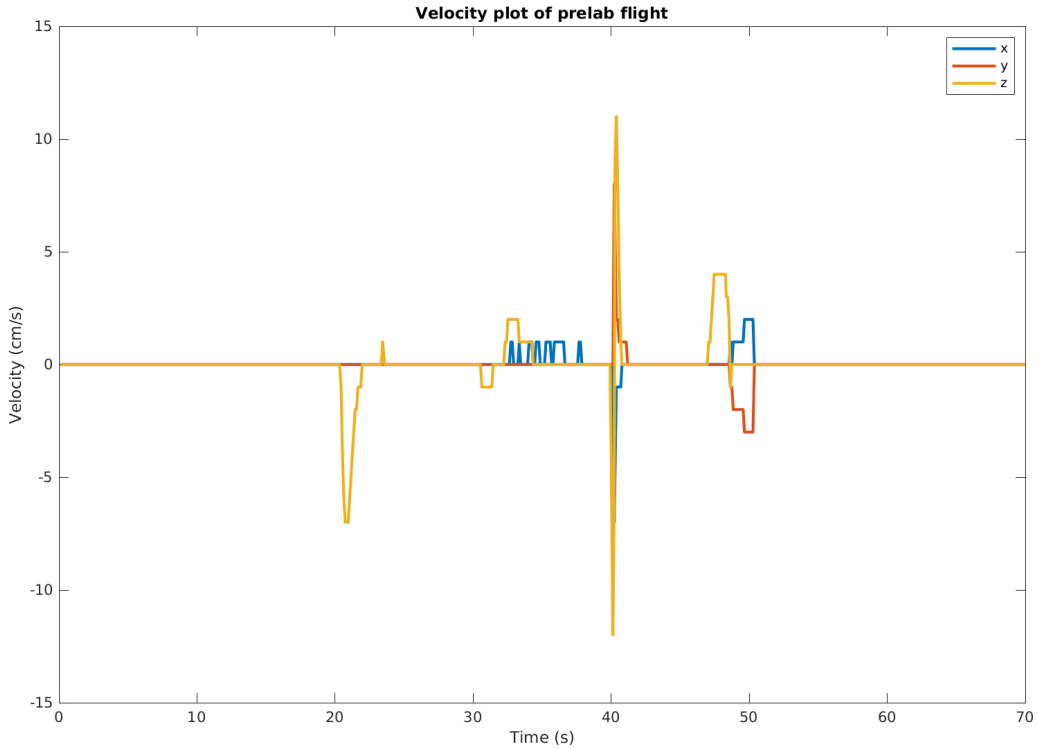

Figure 2: Velocity plot.

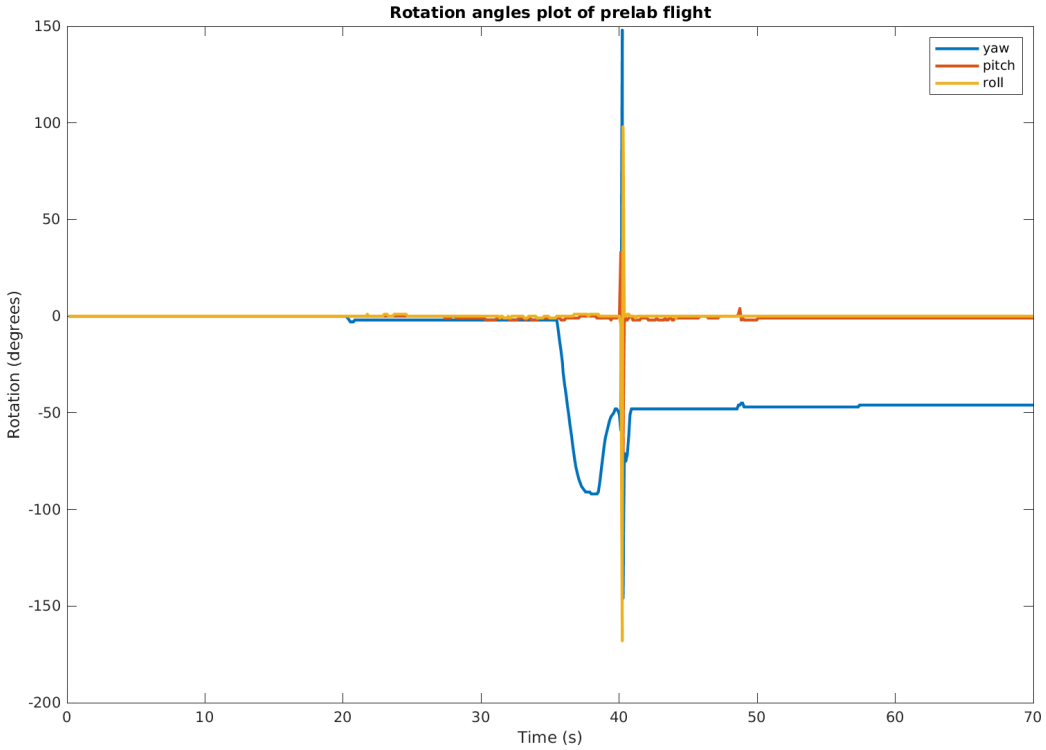

Figure 3: Rotation angle plot.

5

5.1 System Identification

We chose the frequencies 0.5, 0.75, 1.0, 2.0, and 4.0 rad/s.

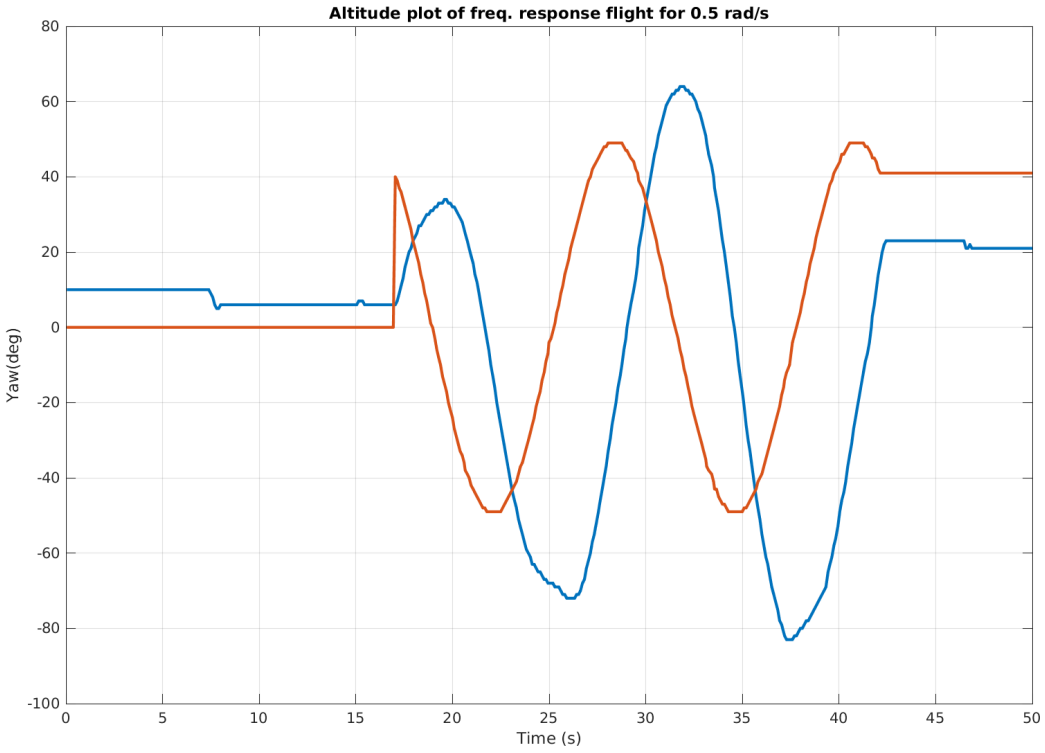

Figure 4: $\omega = 0.5$ rad/s

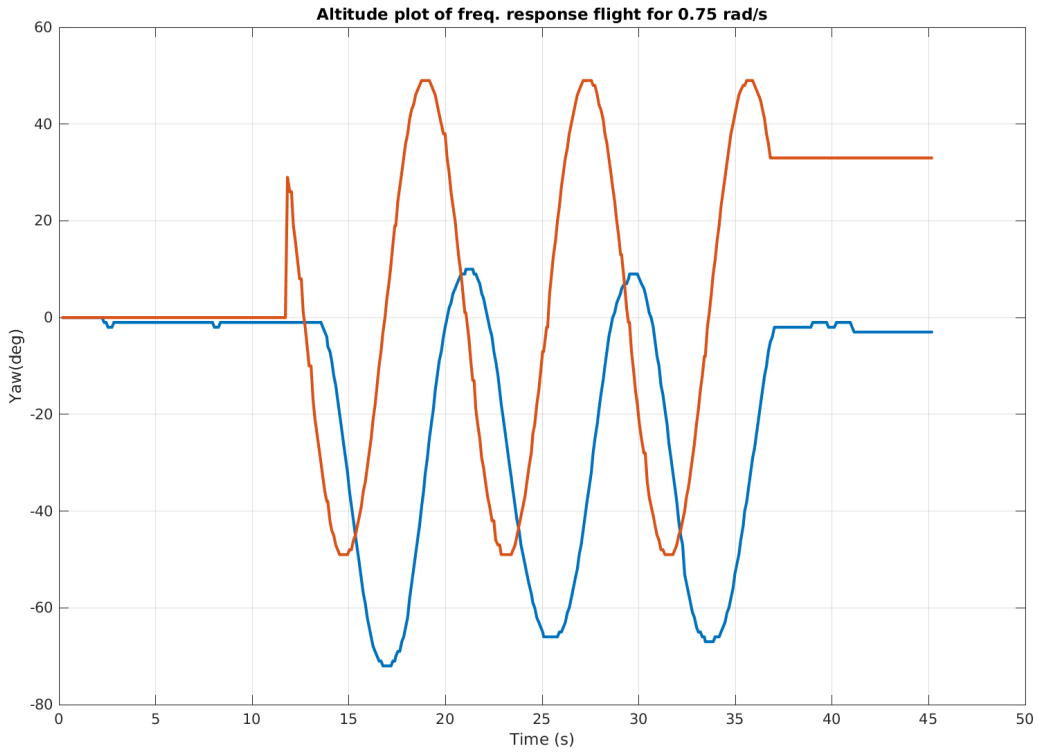

Figure 5: $\omega = 0.75$ rad/s

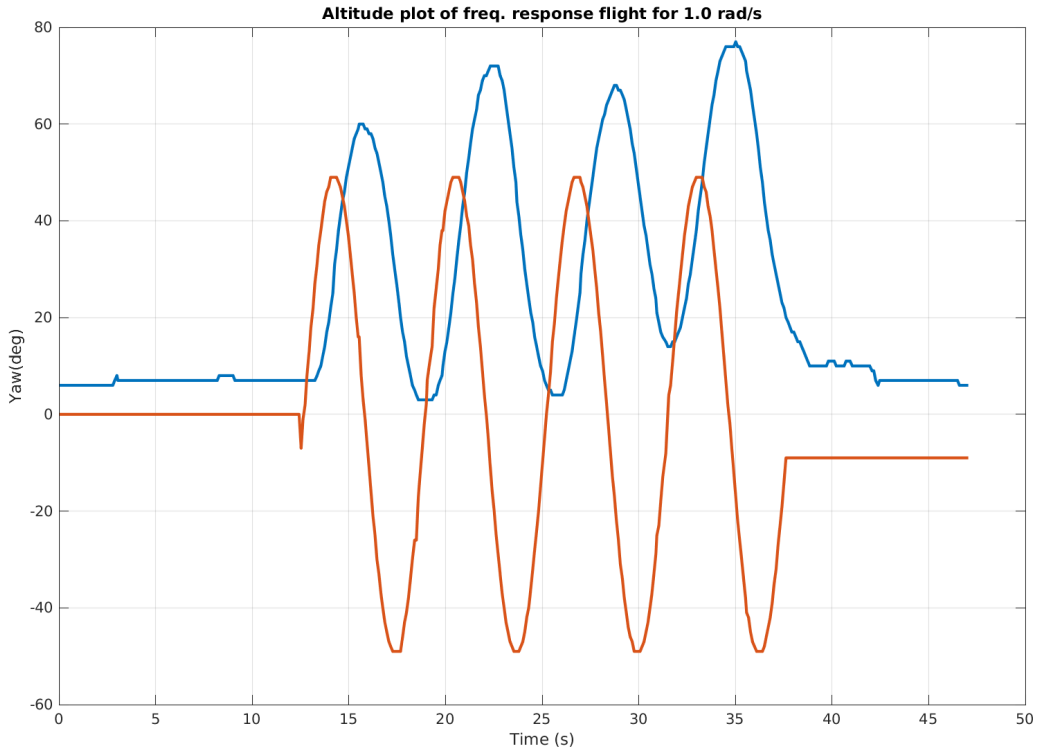

Figure 6: $\omega = 1.0$ rad/s

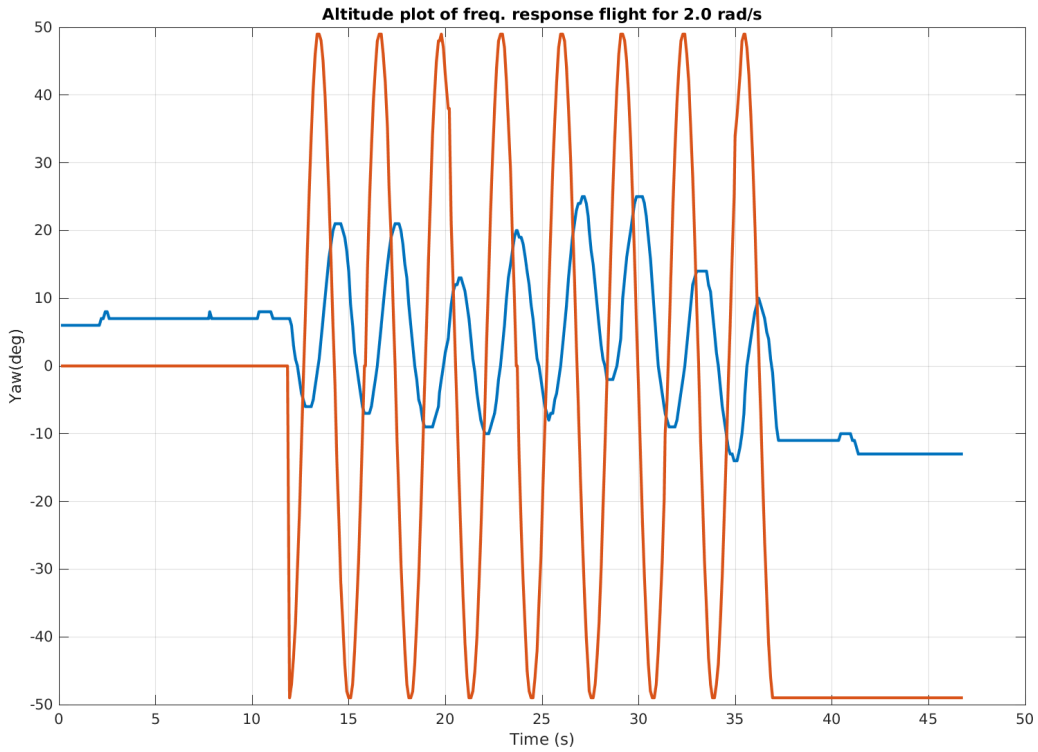

Figure 7: $\omega = 2.0$ rad/s

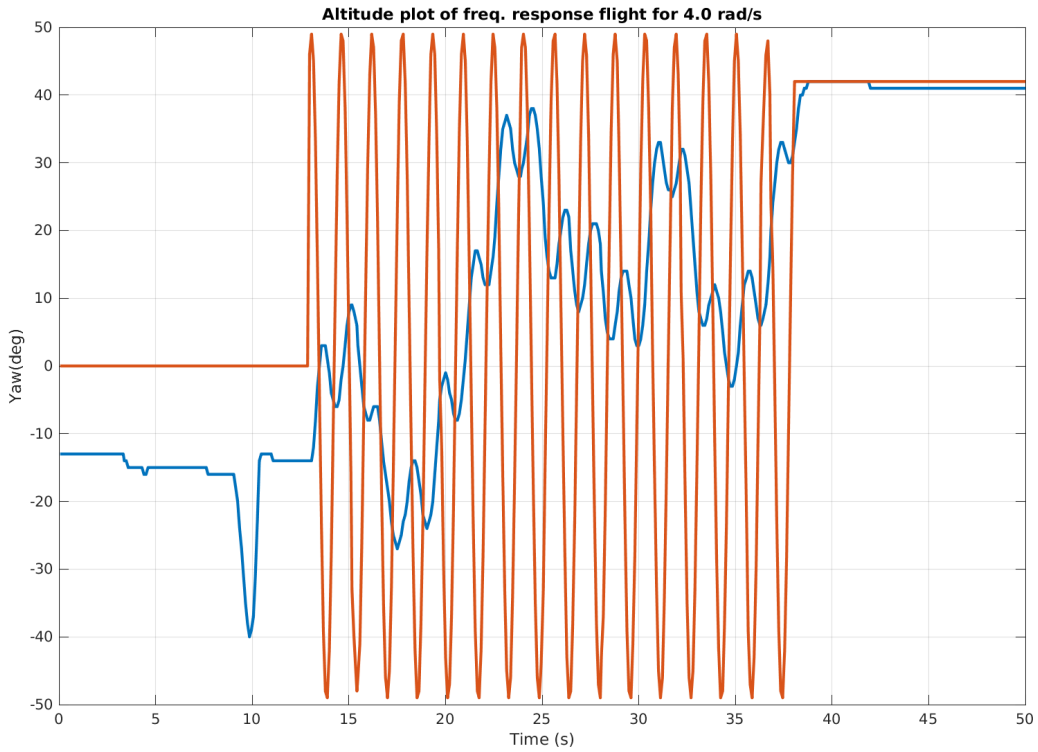

Figure 8: $\omega = 4.0$ rad/s

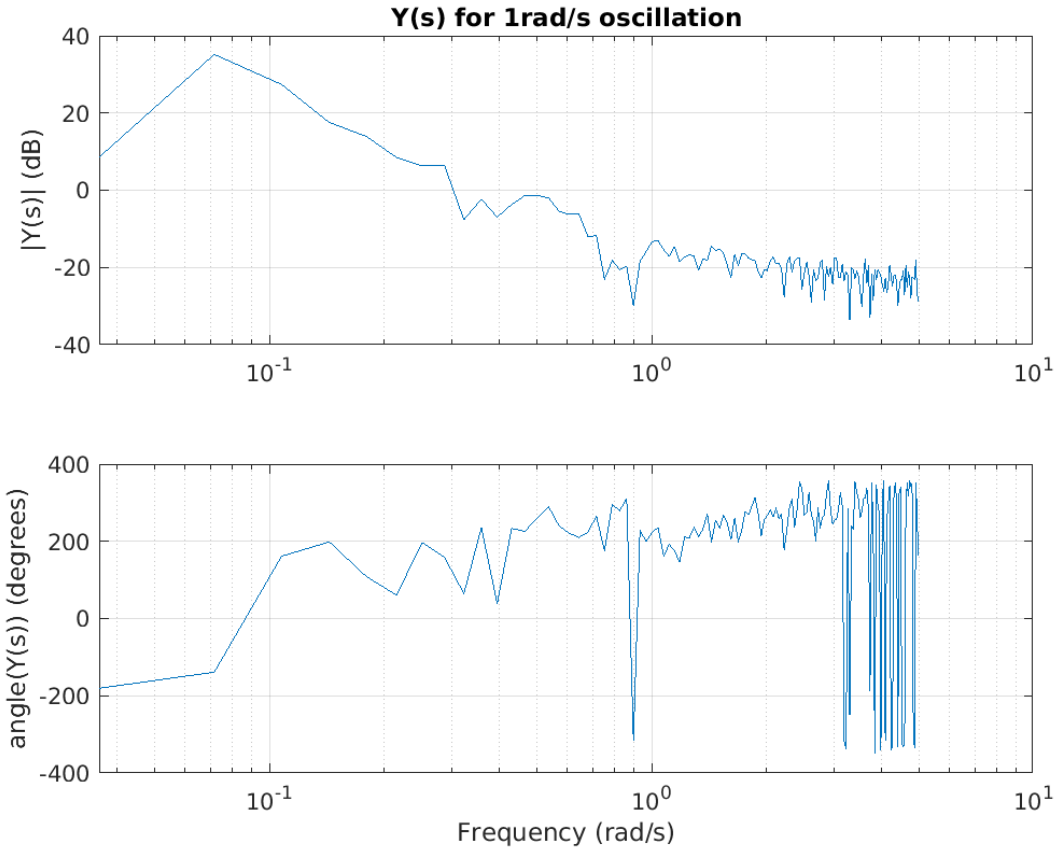

Figure 9: FFT for $\omega = 0.5$ rad/s

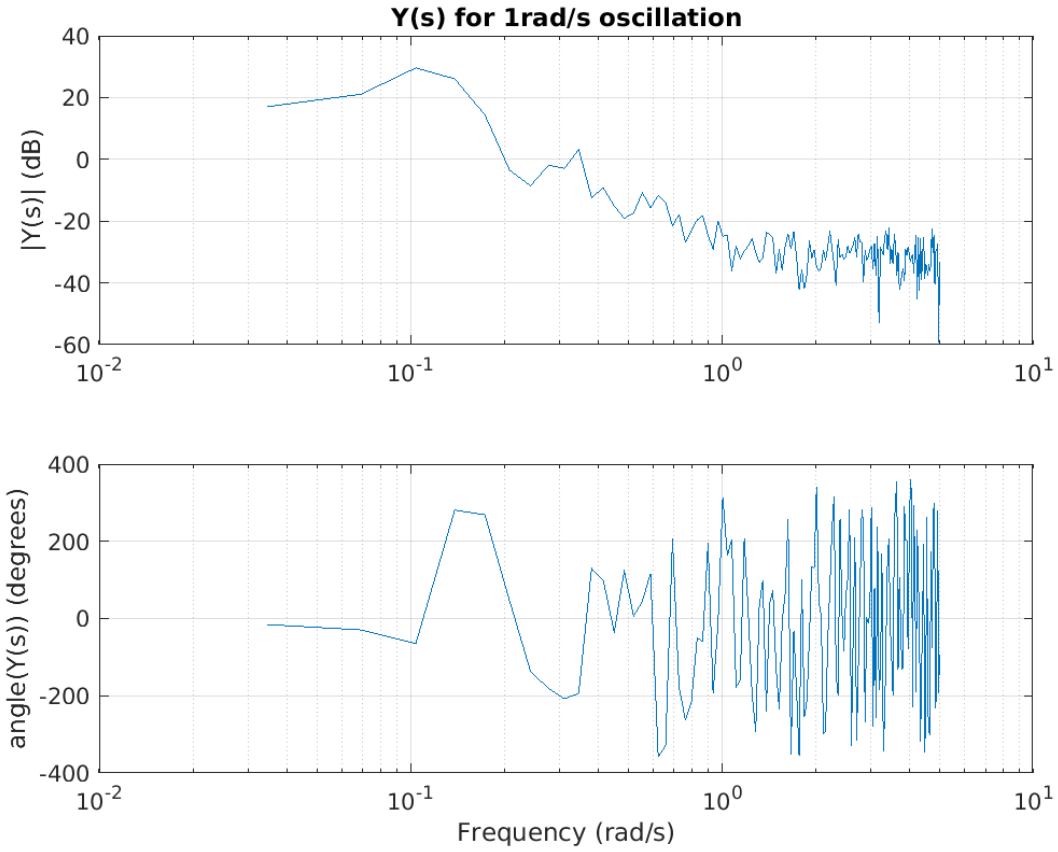

Figure 10: FFT for $\omega = 0.75$ rad/s

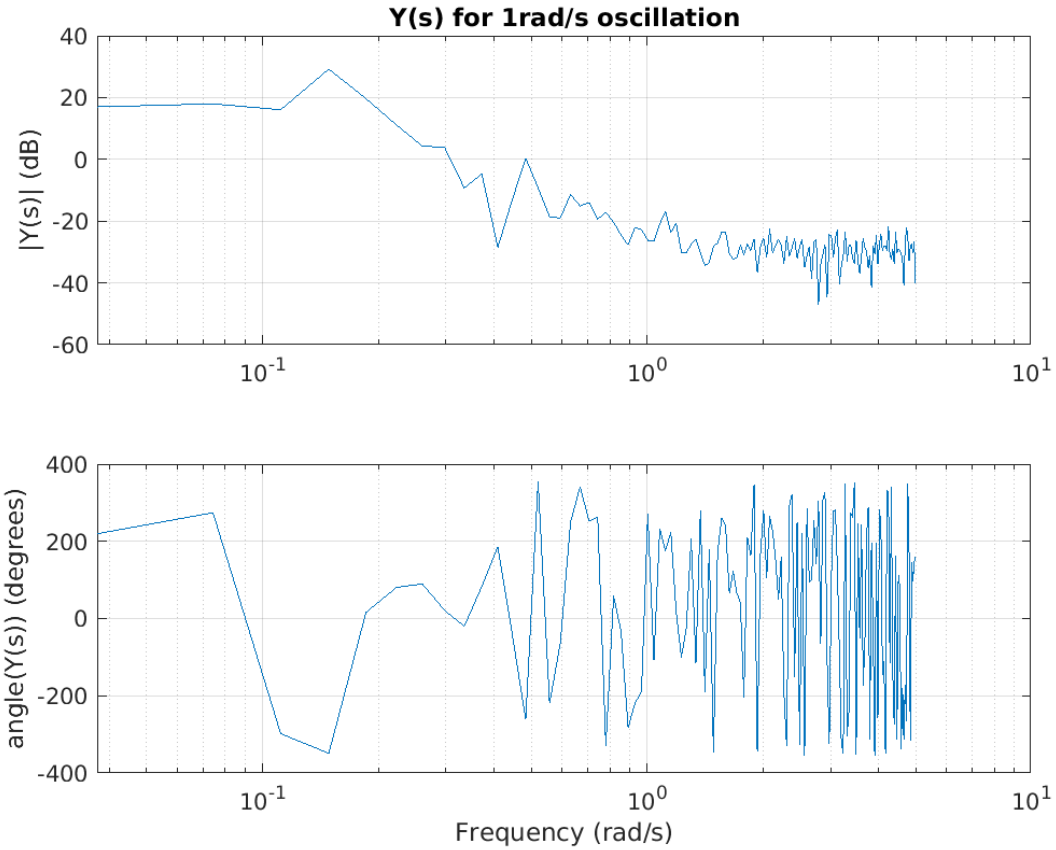

Figure 11: FFT for $\omega = 1.0$ rad/s

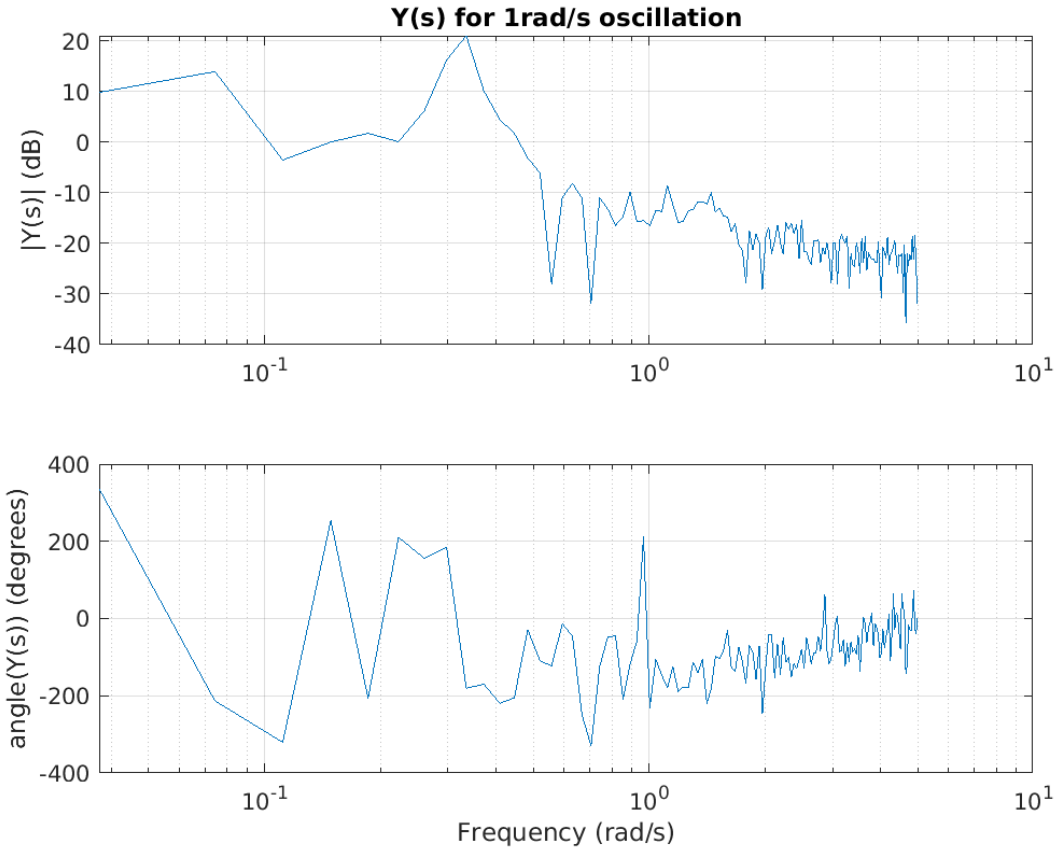

Figure 12: FFT for $\omega = 2.0$ rad/s

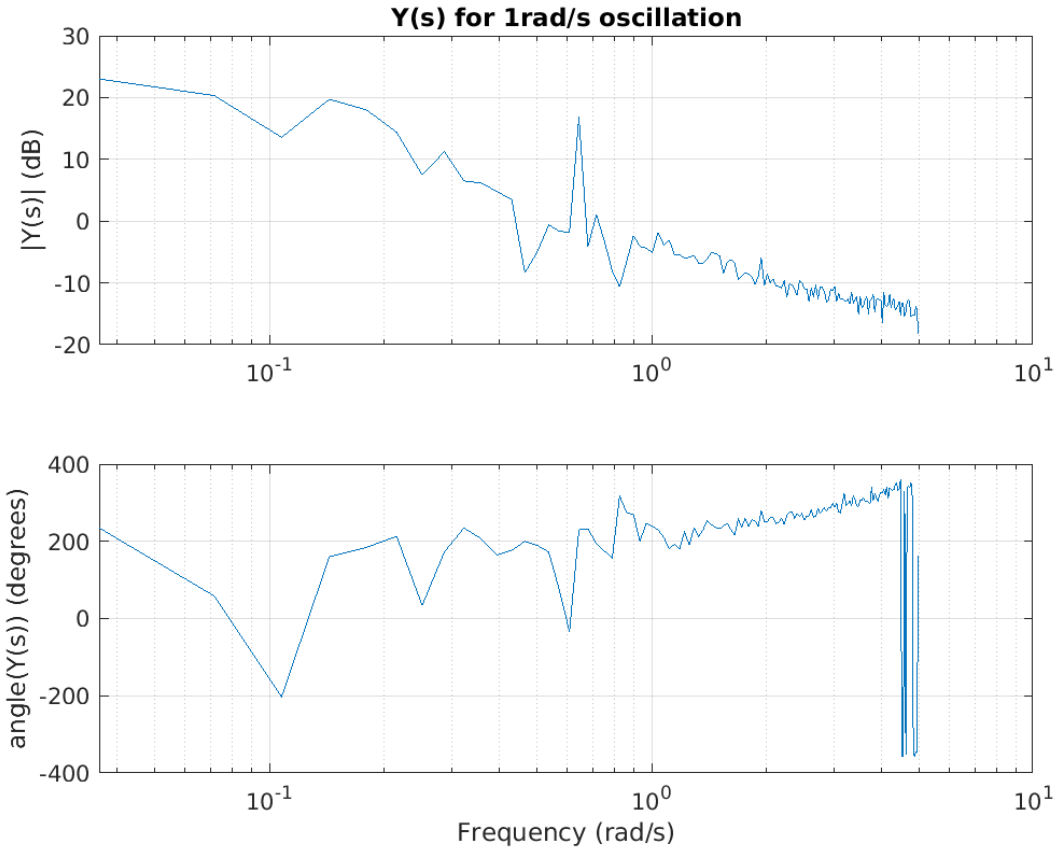

Figure 13: FFT for $\omega = 4.0$ rad/s

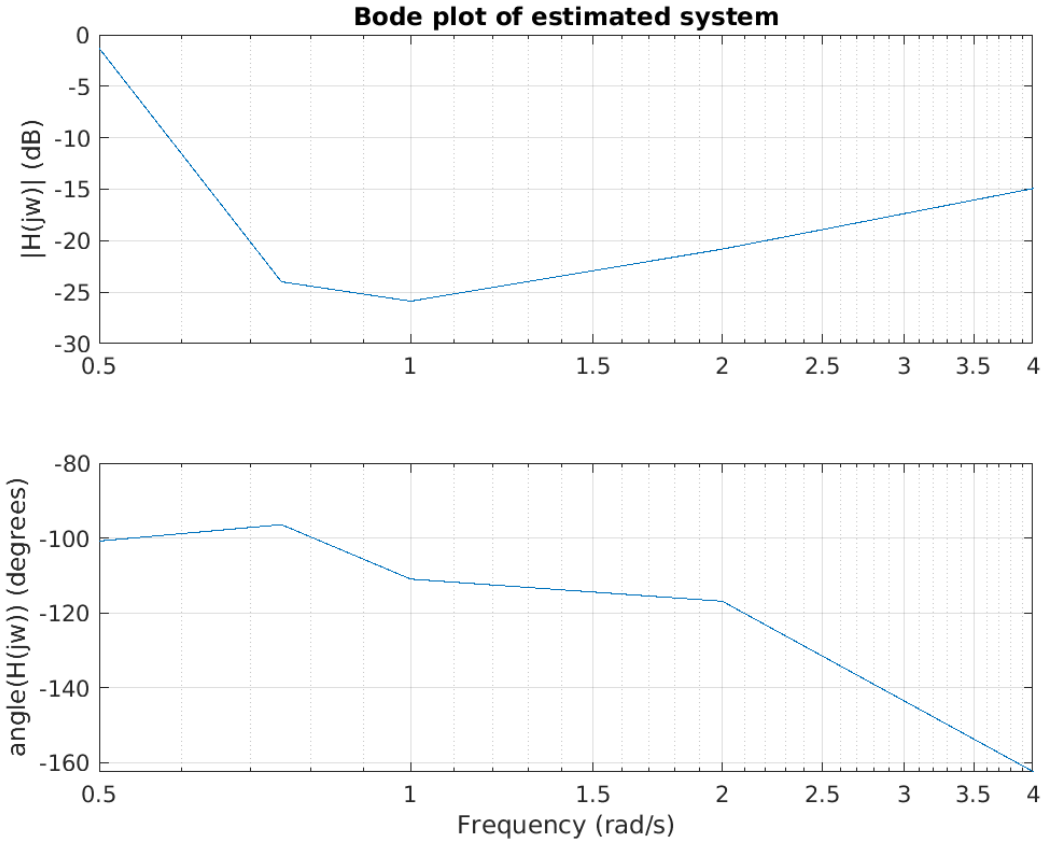

Figure 14: Bode plot derived from gains and phases. See code for how it was generated. There are 4 poles and 5 zeros, and there exists a delay in the system.

Code for plotting and calcuation:

```
close all;


files = {'test_0.5.txt', 'test_0.75.txt', 'test_1.0.txt', 'test_2.0.txt',
    ↪ 'test_4.0.txt'};
names = {'0.5', '0.75', '1.0', '2.0', '4.0'};
in_freqs = [0.5 0.75 1.0 2.0 4.0];

% Calculated by inspecting time differences between peaks in the graphs.
time_shifts = -[(31.875 - 28.356) (21.203 - 18.959) (22.417 - 20.48)
    ↪ (14.377 - 13.357) (23.166 - 22.457)];

% times to clip the input data at
time_start = [15 10 12 11 11];
time_end = [43 39 39 38 39];

% to be computed below
gains = [0 0 0 0 0];
% phase shifts in radians
shifts = in_freqs .* time_shifts;
```

```
% change to 1 for time plots
draw_time_plots = 0;

for file_idx=1:length(files)
    %disp("data/" + files(file_idx));
    data = testDataDecoder("data/" + files(file_idx));

    if (draw_time_plots)
        f = figure('Renderer', 'painters', 'Position', [10 10 900 600]);
        figure(f);
        plot(data.time, data.yaw, 'LineWidth', 2);
        hold on;
        plot(data.time, data.control_yaw, 'LineWidth', 2);
        title("Altitude plot of freq. response flight for "+
            ↪ names(file_idx) + " rad/s");
        xlabel("Time (s)");
        ylabel("Yaw(deg)");
        xlim([0 50]);
        grid;
        % uncomment to save
        saveas(f, "../figures/q51_time"+names(file_idx)+".png");
        hold off;
    end

    tstart = find(data.time > time_start(file_idx), 1);
    tend = find(data.time > time_end(file_idx), 1);
    if (mod(tend - tstart, 2) == 1)
        tend = tend + 1;
    end

    Y_data = data.yaw(tstart:tend);
    U_data = data.control_yaw(tstart:tend);
    t_data = data.time(tstart:tend);

    freqs = [0:0.01:10];

    Y = fft(Y_data);
    L = length(Y_data);

    % calc/plot magnitude of output
    P2 = abs(Y/L);
    P1 = P2(1:L/2+1);
    P1(2:end-1) = 2 *P1(2:end-1);
    f = figure;
    figure(f);

    subplot(2, 1, 1);
    semilogx(10*(0:(L/2))/L, mag2db(P1));
    title("Y(s) for " + string(1) + "rad/s oscillation");
    %xlabel("Frequency (rad/s)");
    grid;
    ylabel("|Y(s)| (dB)");
```

```
    % plot the phase...i think
    Ph2 = angle(Y/L);
    Ph1 = Ph2(1:L/2+1);
    Ph1(2:end-1) = 2 *Ph1(2:end-1);

    subplot(2, 1, 2);
    semilogx(10*(0:(L/2))/L, Ph1 / pi * 180);
    grid;
    %title("|Y(s)| for " + string(1) + "rad/s oscillation");
    xlabel("Frequency (rad/s)");
    ylabel("angle(Y(s)) (degrees)");

    saveas(f, "../figures/q51_ft_"+names(file_idx)+".png");

    % get output Y for corresponding in frequency
    gains(file_idx) = interp1(10*(0:(L/2))/L, mag2db(P1),
        ↪ in_freqs(file_idx));

end

f = figure;
figure(f);


subplot(2, 1, 1);
semilogx(in_freqs, gains);
title("Bode plot of estimated system");
ylabel("|H(jw)| (dB)");
grid;

subplot(2, 1, 2);
semilogx(in_freqs, pi / 180 * (in_freqs .* time_shifts));
ylabel("angle(H(jw)) (degrees)");
xlabel("Frequency (rad/s)");
grid;
saveas(f, "../figures/q51_bode.png");

return;
```

5.2 Proportional control

We use $k_P = 3$ for our proportional controller. The gain margin is roughly 12 dB and the phase margin is -40 deg, and we got this value from the estimation and some tuning.

Here's the code:

```
# Controller Variables
kp = 3

# Control stores
integratedError = 0.0
errorStore = 0.0

# Useful Reference Signal Variables
period = 20
amplitude = 180

##### ... skip starter code ... #####

# Compute Reference Signal (Triangle wave)
reference = (2 * amplitude) / np.pi * np.arcsin(np.sin(2*np.pi * ptime /
    ↪ period))

# Compute Error and Control Input
error = reference - yaw
control_YA = kp * error
```

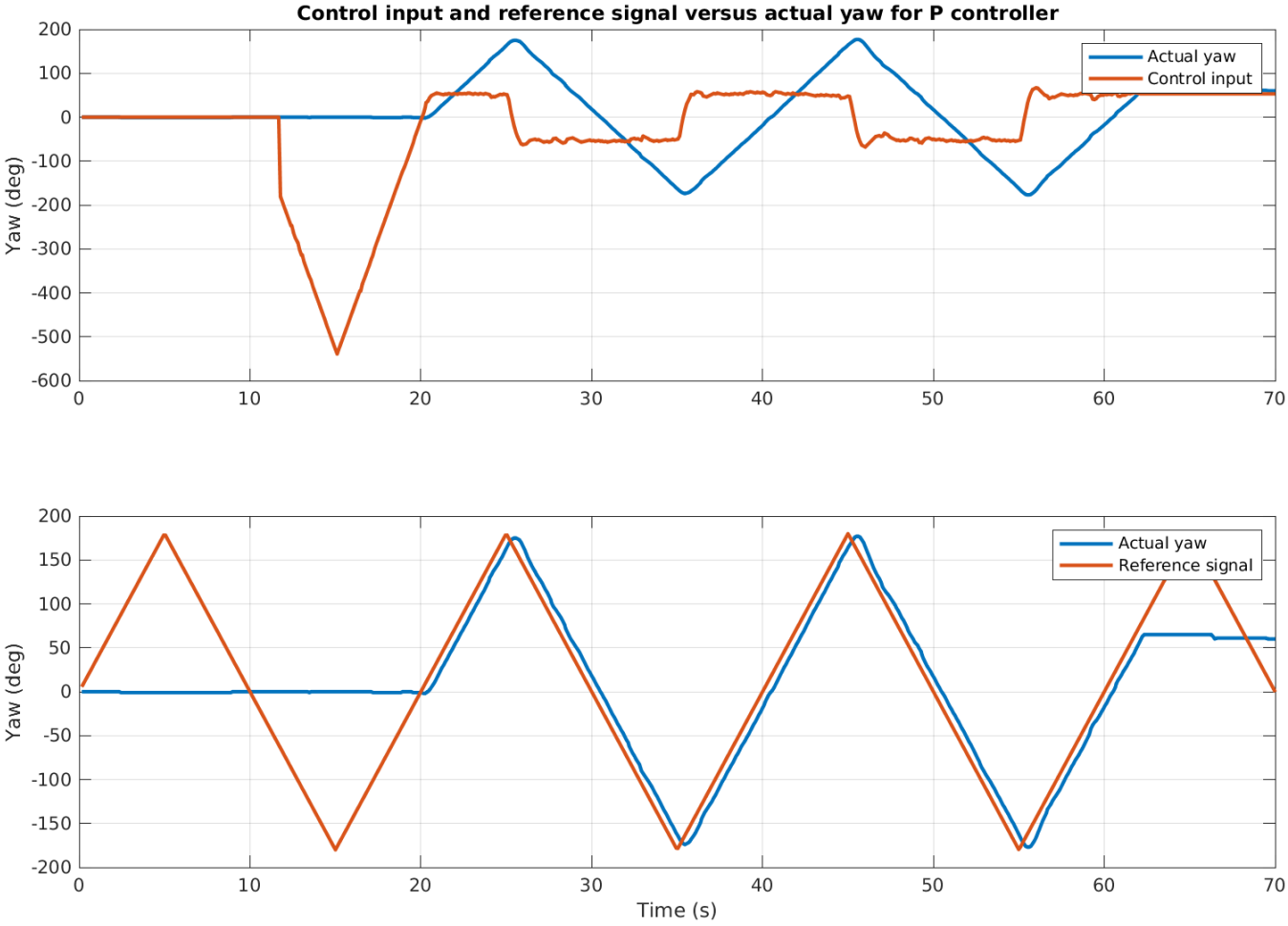

Figure 15: Here's the plot. It actually tracks pretty well, although there is a small steady state error.

Here's plotting code:

```
%data = testDataDecoder("data/test_0.5.txt");
data = testDataDecoder("data/q52_p_control.txt");

f = figure('Renderer', 'painters', 'Position', [10 10 900 600]);
figure(f);

subplot(2, 1, 1);

plot(data.time, data.yaw, 'LineWidth', 2, 'DisplayName', 'Actual yaw');
hold on;
plot(data.time, data.control_yaw, 'LineWidth', 2, 'DisplayName', 'Control
    ↪ input');
title("Control input and reference signal versus actual yaw for P
    ↪ controller");
ylabel("Yaw (deg)");
xlim([0, 70]);
grid;
legend;
hold off;

subplot(2, 1, 2);
plot(data.time, data.yaw, 'LineWidth', 2, 'DisplayName', 'Actual yaw');
hold on;

plot(data.time, 360 / pi * asin(sin(2 * pi * data.time/20)), 'LineWidth',
    ↪ 2, 'DisplayName', 'Reference signal');

xlabel("Time (s)");
ylabel("Yaw (deg)");
xlim([0, 70]);
grid;
legend;
hold off;

saveas(f, "../figures/q52.png");

return;
```

5.3 Better Triangular Wave Tracking

5.3.1 Designing a controller

We used a PI controller with $k_I = 0.5$ and an anti-windup decay term of 0.7. Here's the code:

```
# Controller Variables
kp = 3.0
ki = 0.05
kw = 0.7 # decay term to avoid integral windup

###### skip starter code ######
# Compute Reference Signal (Triangle wave)
reference = (2 * amplitude) / np.pi * np.arcsin(np.sin(2*np.pi * ptime /
    ↪ period))
```

```
# Compute Error and Control Input
error = reference - yaw
integratedError = kw * integratedError + (ptime - lastTime) * error
control_YA = kp * error + ki * integratedError
```

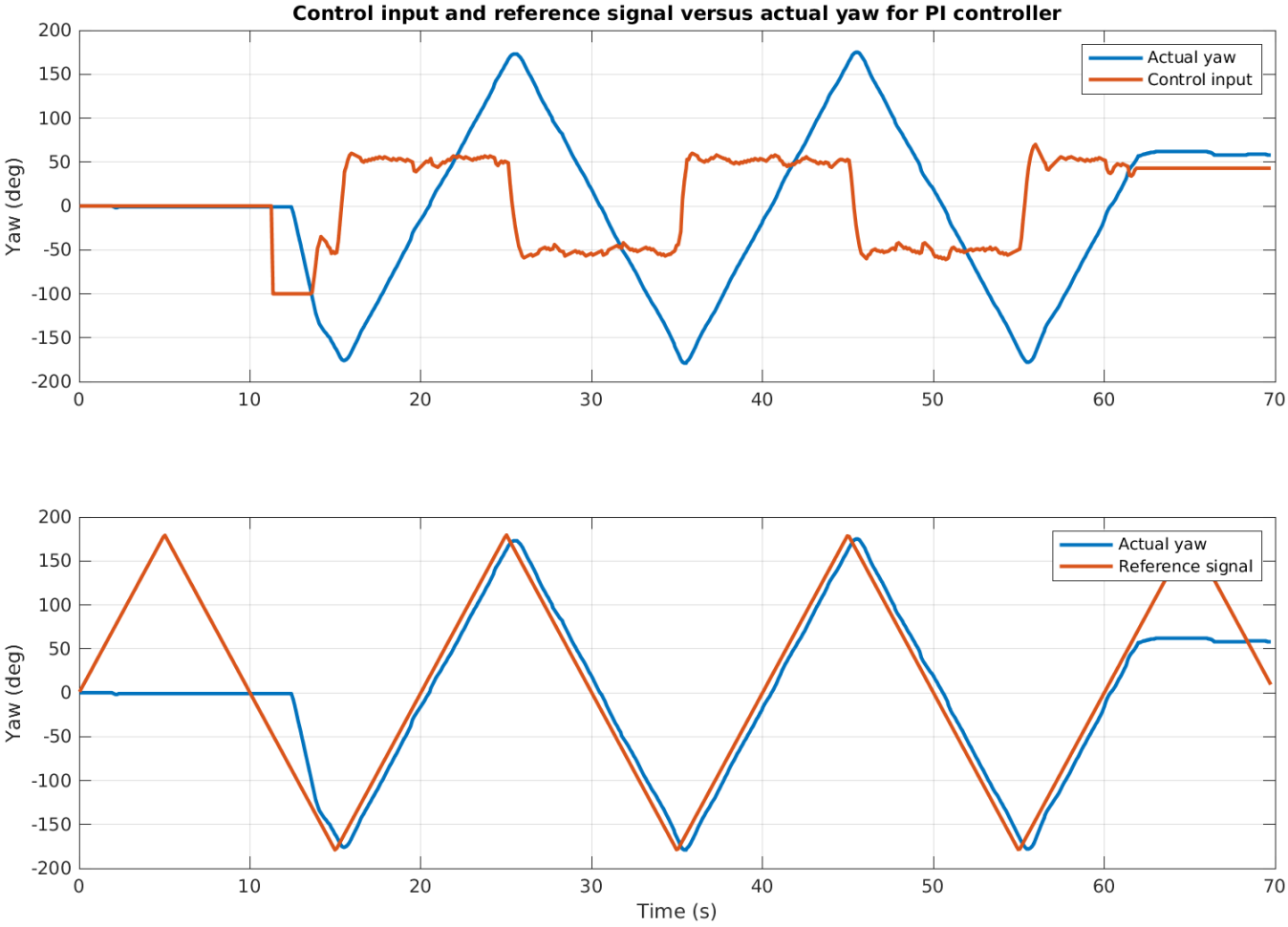

Figure 16: Here's the plot. It still tracks well, but we were unable to remove the (very small) SSE entirely.

5.3.2 Comparison to DJI's controller

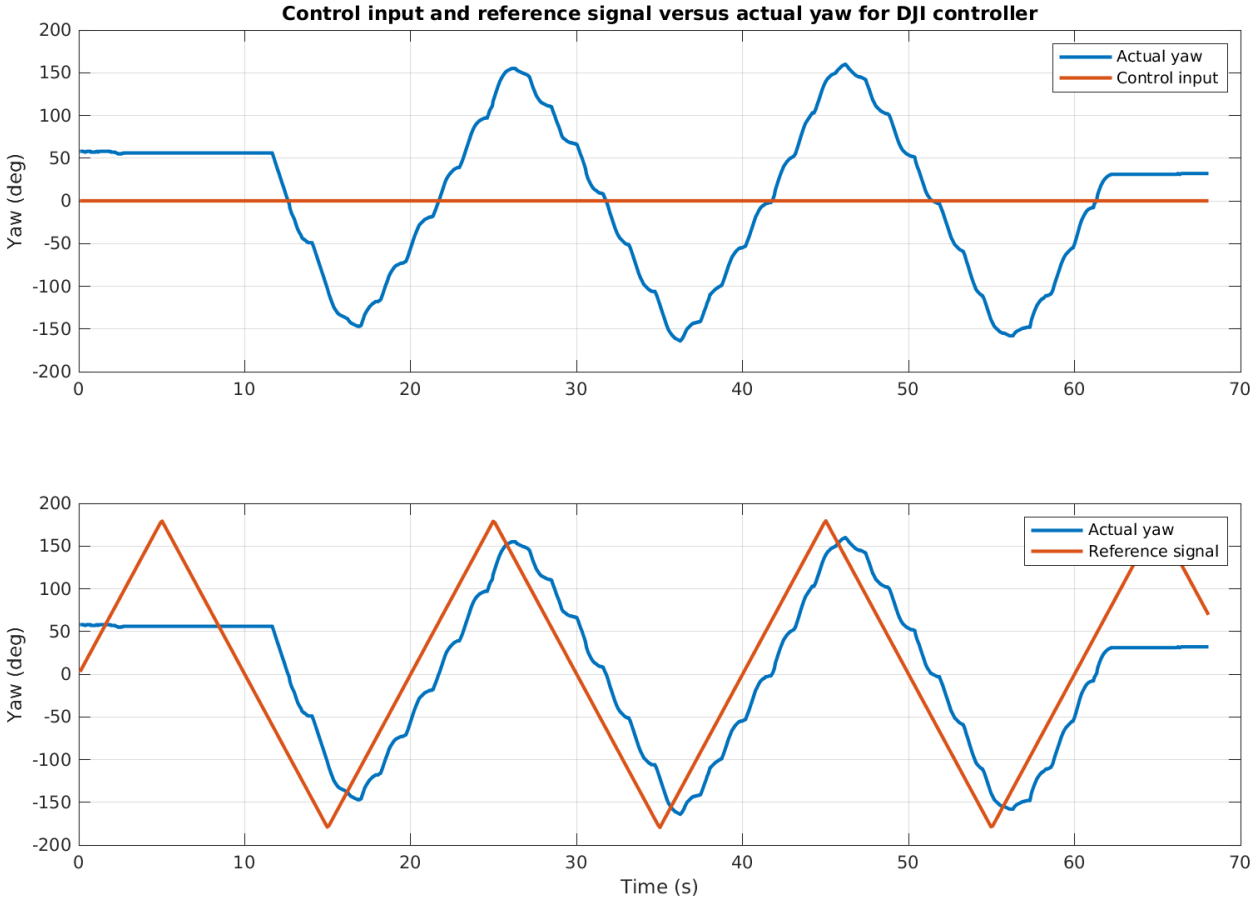

Figure 17: Here's the plot. In DJI's defense, it seems rather dubious that the commands used to attempt to follow the function were intended to be used for this sort of thing, as it seems to be constantly updating a yaw position setpoint every 10 Hz instead of being fully smooth.

5.4 Lab 5a comparisons

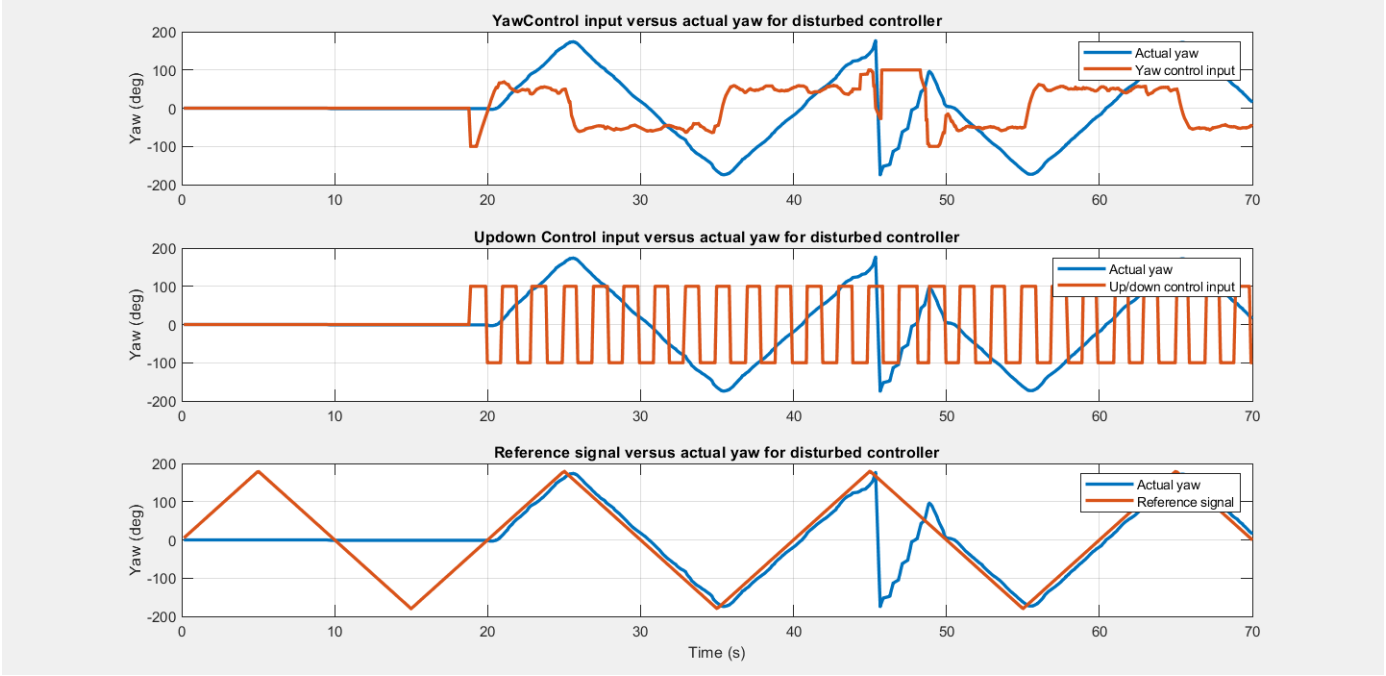

Figure 18: Our system was decent, but not as good as the one in 5.3.1. There is a sudden drop at around 45 seconds, which is likely because the controller and drone are not able to handle one of the sudden disturbance we applied to the system. (Or it could be lighting issues in the environment or other hardware-related issue). Aside from that, our controller is actually pretty good.