EE155/255 Homework #3 Solutions Fall 2017-18

Question 1 (Total 40 pts):

We use the following motor model to create our motor simulation function.

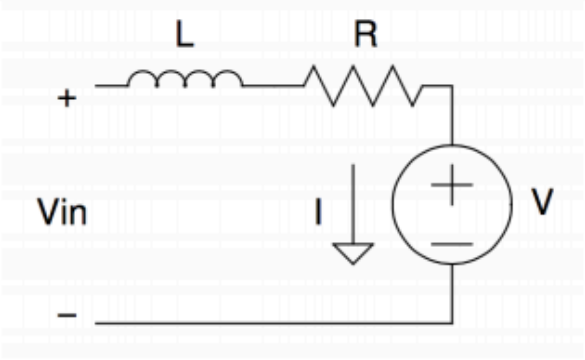


Figure 1: Motor model

A brushed DC motor can be modeled as shown in Figure 1. The inductance and resistance are properties of the motors coils. The current I through the motor is proportional to the torque exerted by the shaft, and the voltage V is proportional to the rotational speed ω.

Definitions:

R	Resistance in ohms
L	Inductance in henries
V	Back-EMF (voltage produced by rotation) in volts
I	Current through the motor
K_e	Back-EMF constant, $\frac{V}{rad/s}$
K_T	Torque constant, $\frac{N \cdot m}{A}$
T	Torque, $N \cdot m$
J	Load inertia, $\frac{kg \cdot m^2}{rad^2}$

Using the units shown above, $K_e = K_T$. Motors are often specified with a speed constant K_V in $\frac{RPM}{V}$.

Motor behavior:

$$V = K_e \omega$$
$$T = K_T I$$
$$\dot{\omega} = \frac{T}{J}$$
$$V_{in} = K_e \omega + IR$$

With these equations, it is quite simple to model our motor over a small time-step in a function.
Every time step, we will update our back-EMF voltage, update our change in current, and update our change in velocity. Then we will send this information to the main loop from where this function is called. We run this over many time-steps and then plot our results!

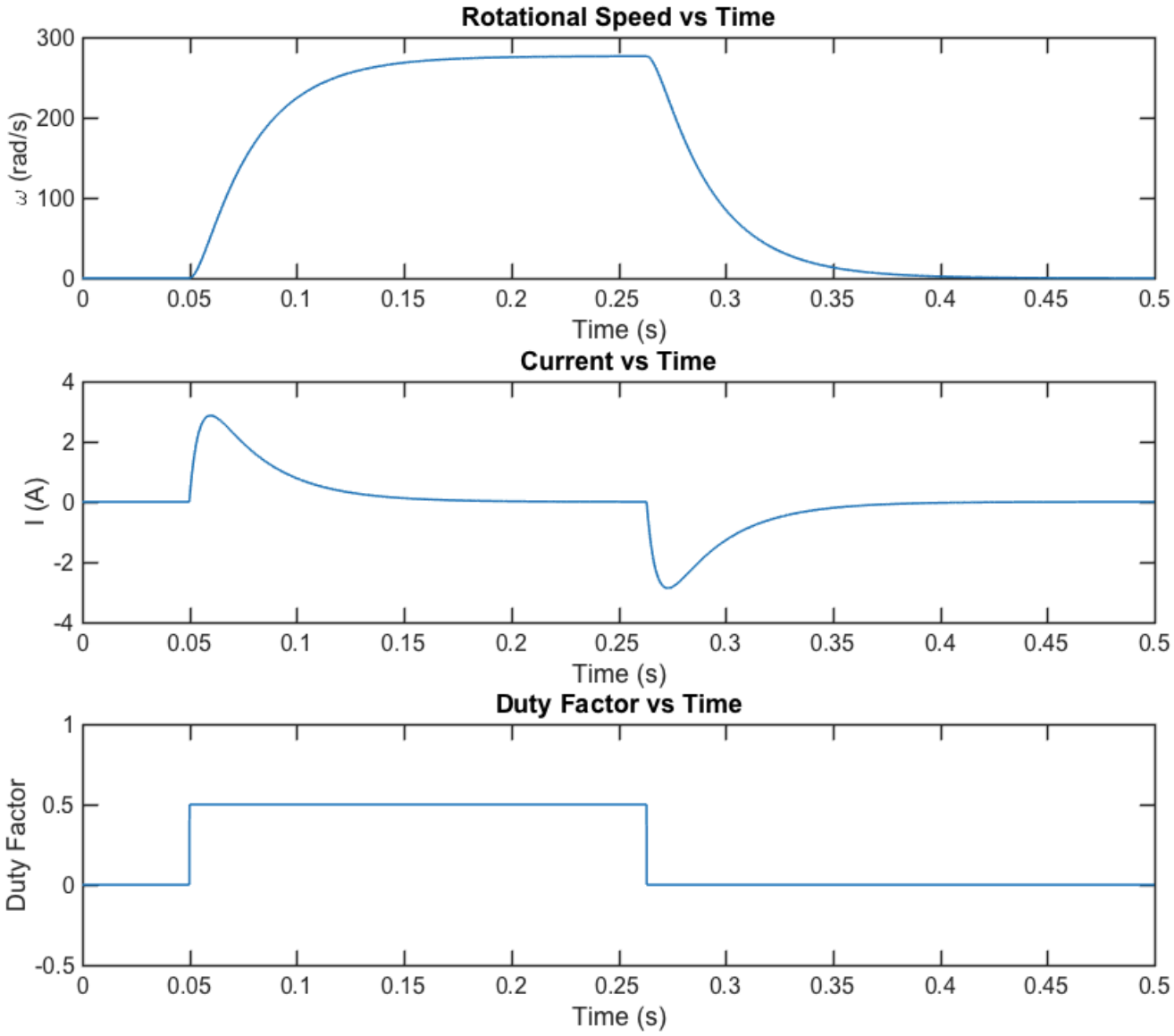
Motor simulation function (10 pts):

```
function [ omega, I ] = hw3_motor_sim(t_sim, duty, Vss, I, omega)
    Km = 220; % RPM/V
    R = 3.401; % Winding resistance in Ohms
    L = 13.58e-3; % Winding inductance in Henries
    J = 1.754e-5; % Load inertia
    Km = Km * 2 * pi / 60;
    Ke = 1./Km;
    Kt = 1./Km;
    dt = 1e-6;
    n_step = t_sim/dt;
    Vin = duty*Vss;
    for n=1:n_step
        Ve = Ke*omega;
        dI = dt*(Vin-(I*R+Ve))/L;
        I = I + dI;
        d_omega = dt*Kt*I/J;
        omega = omega + d_omega;
    end
end
```

Main loop function (15 pts):

```
t_step = 1e-4;
t_max = 0.5;
n_step = t_max/t_step;
t = 0;
error_nback = 0;
T_arr = linspace(0,t_step*n_step,n_step);
I_arr = zeros(1,n_step);
Omega_arr = zeros(1,n_step);
df = 0;
Vss_arr = ones(1,n_step);
Vss_arr = Vss_arr*24;
I = 0;
omega = 0;
Df = zeros(1,n_step);

for n=1:n_step
    if (t > t_max/10)
        df = 0.5;
    end
    if (t > t_max/1.9)
        df = 0;
    end
    [omega, I] = hw3_motor_sim( t_step, df, Vss_arr(n), I, omega );
    derivative, Errors(1), Vss_arr(n) );
    Df(n) = df;
    I_arr(n) = I;
    Omega_arr(n) = omega;
    t = t + t_step;
end

figure

subplot(311);
plot(T_arr, Omega_arr) %
ylabel('\omega (rad/s)') %
xlabel('Time (s)');
title('Rotational Speed vs Time');

subplot(312);
plot(T_arr, I_arr) %
ylabel('I (A)') %
xlabel('Time (s)');
title('Current vs Time');

subplot(313);
plot(T_arr, Df) %
ylabel('Duty Factor') %
xlabel('Time (s)');
title('Duty Factor vs Time');
axis([0 t_max -0.5 1]);
```

Plots for question 1 (15pts):



As you can see the current ramps up pretty quickly!

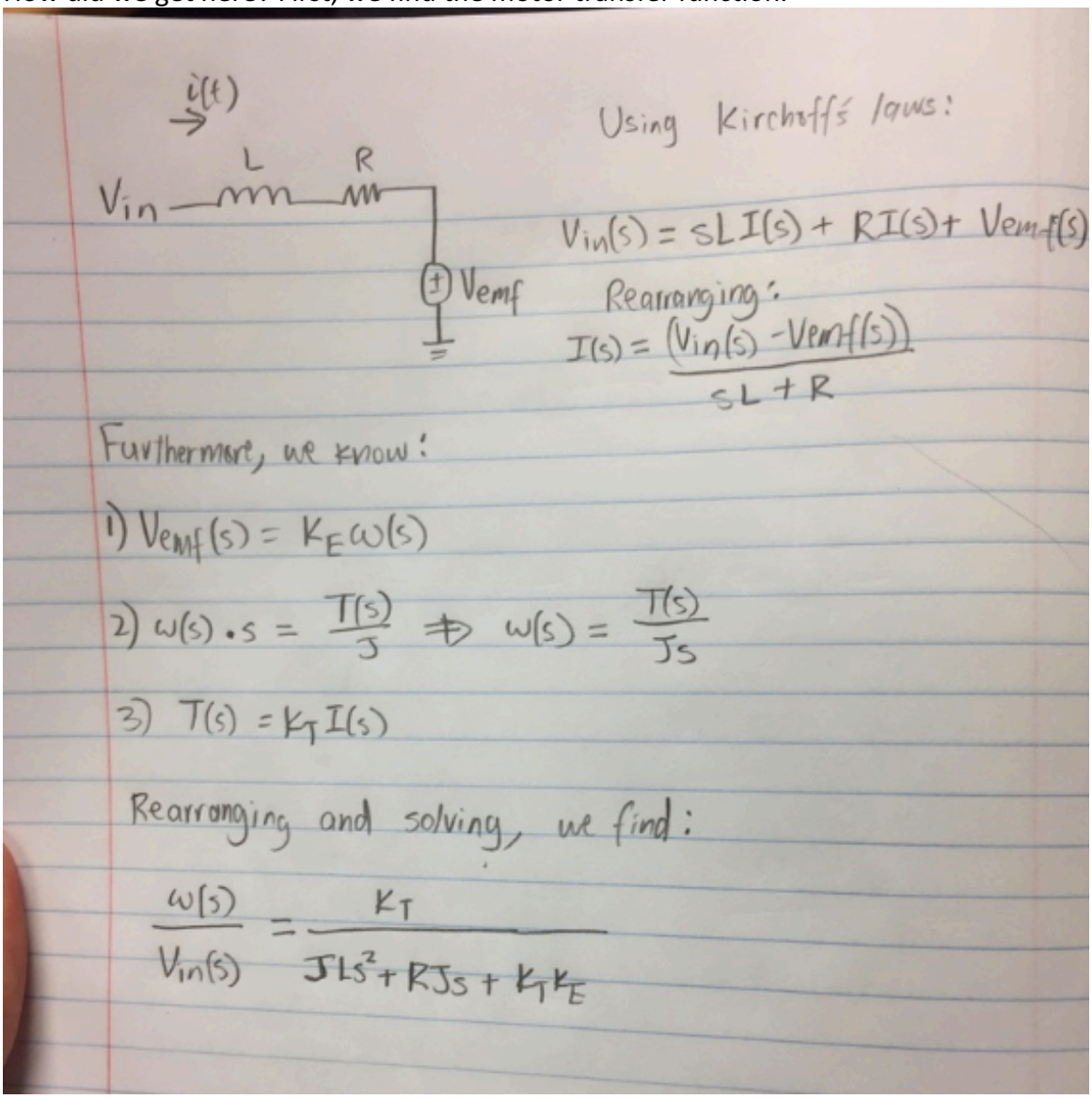
<u>Make sure speeds and currents match exactly.</u>

-5 pts for not showing speed vs time
-5 pts for not showing current vs time
-5 pts for not showing duty factor vs time
-5 pts for incorrect Km calculation
-5 pts for limiting current

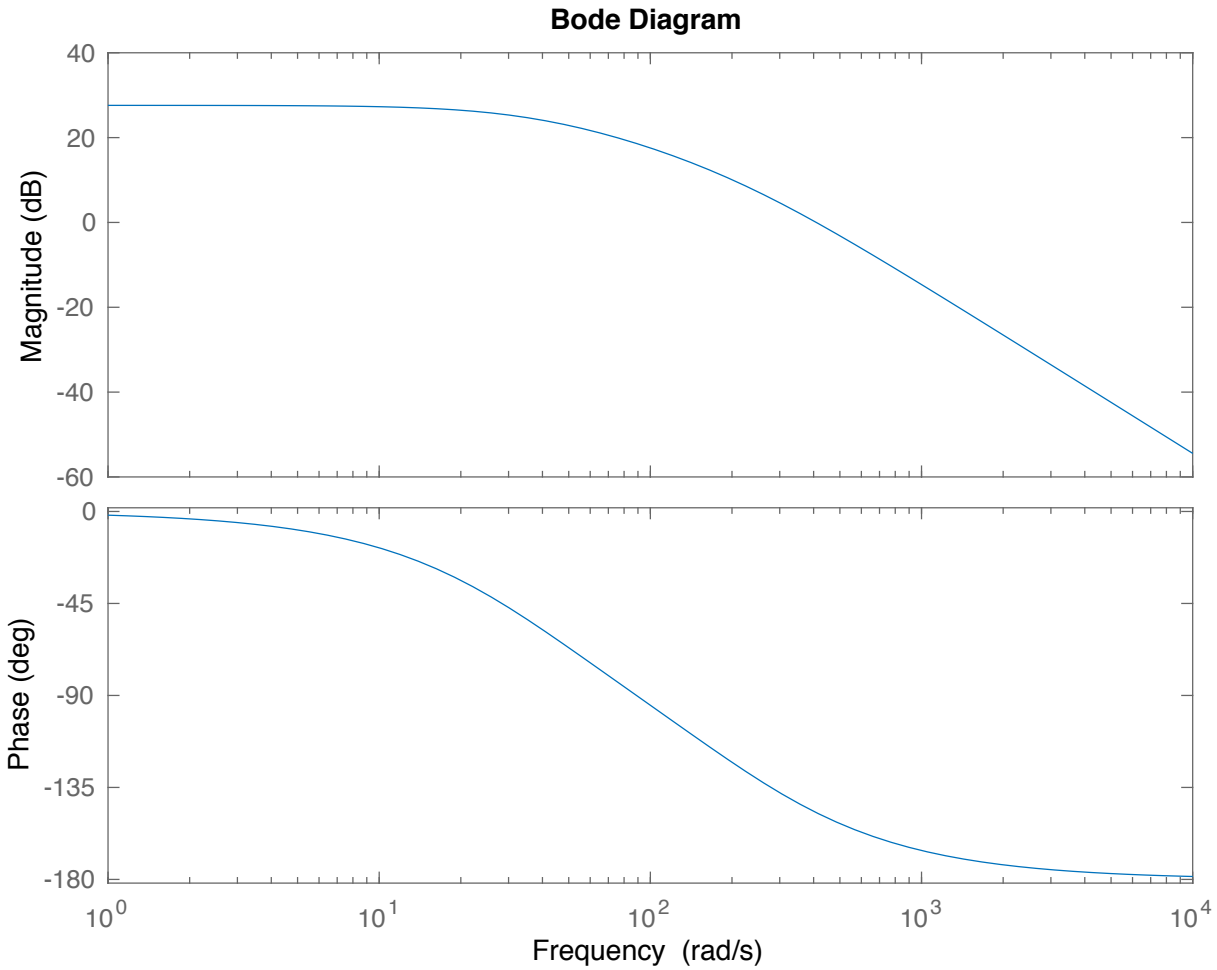
Question 2 (Total 60 pts):

For simplicity, we will limit the current into the motor to have a maximum of 1 Amp, so we can really test our control system. We use a PD controller to control the duty factor of the motor. With a given velocity setpoint, current velocity, and maximum allowable current into the motor, the PD controller determines the necessary duty factor to most easily and safely get to the desired speed. We use a PID controller because a simple P controller will never reach the desired velocity, and a "D" term to prevent ringing and unnecessary oscillations in current and duty factor. Given the motor parameters, we create a motor model and find KP = 10 and KD = 0.1 match our needs for this question.

How did we get here? First, we find the motor transfer function.

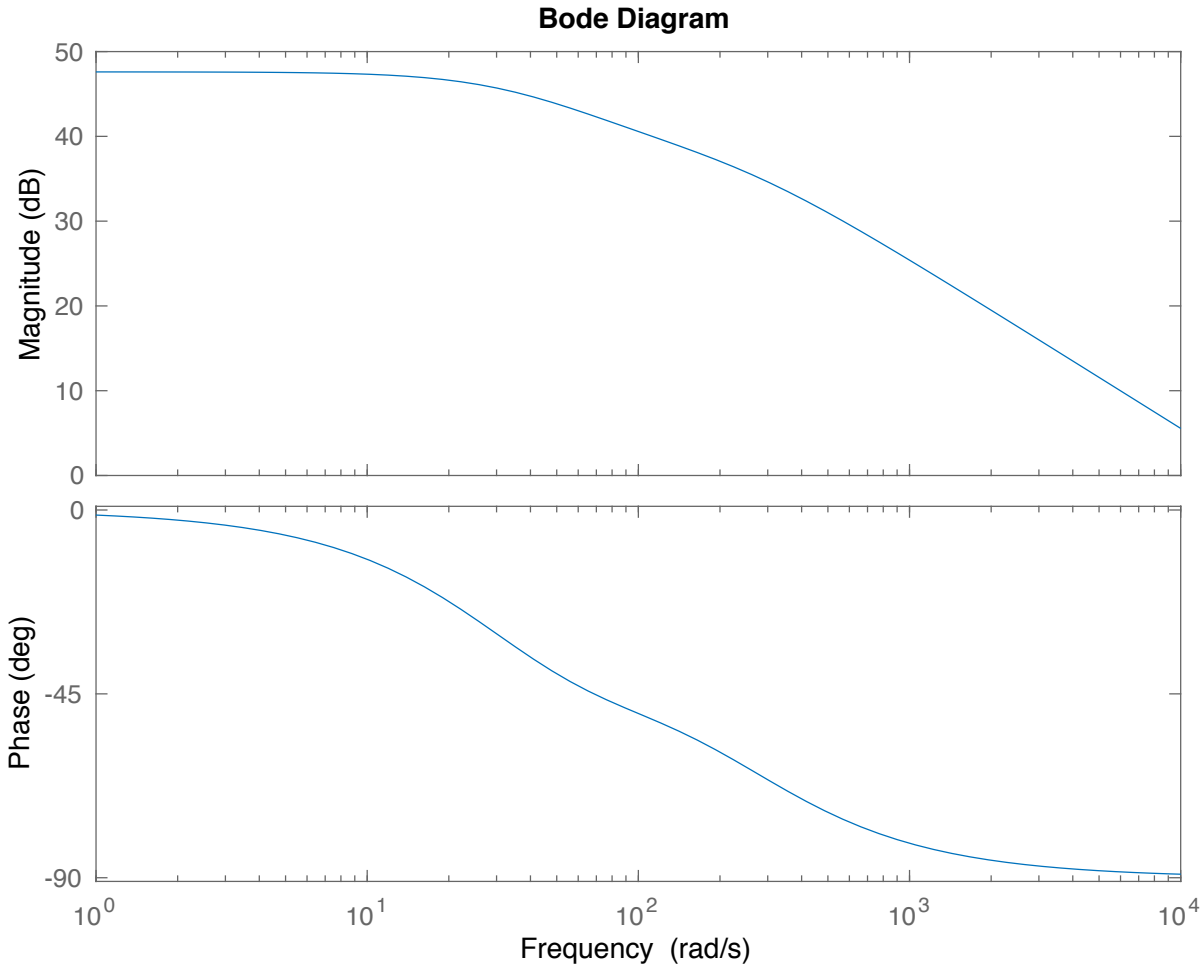


Let's take a look at the bode plot of this accounting for the fact that Vin = 24*Duty:

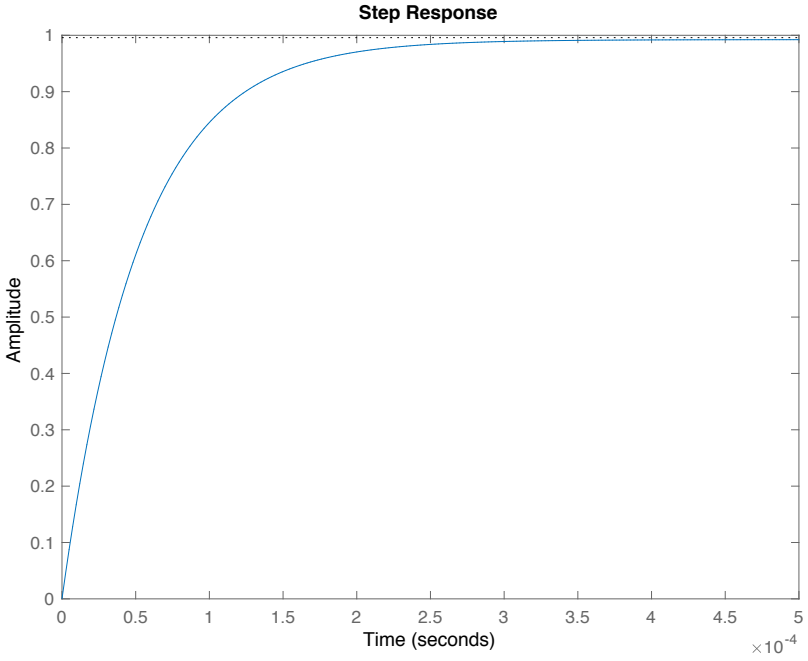


What this shows is that if you add a lot of gain, you will move the unity-gain point up to where you have no phase margin. Hence, it will ring with large Kp (anything more than about 0.1) unless you add derivative gain.

Now let us add in the controller function to the plant. To stabilize this we want to put in a zero at about 100 rad/s. To do so we simply set Kp/Kd = 100. The proportional gain is chosen large enough so that the residual error will be close to zero without integral gain. A value of 10 was picked which with the 24V supply gives a gain of 240, with a residual error of 1/241 (small enough that we won't need an integral term). The open loop bode plot of such a plant + controller system is shown below:

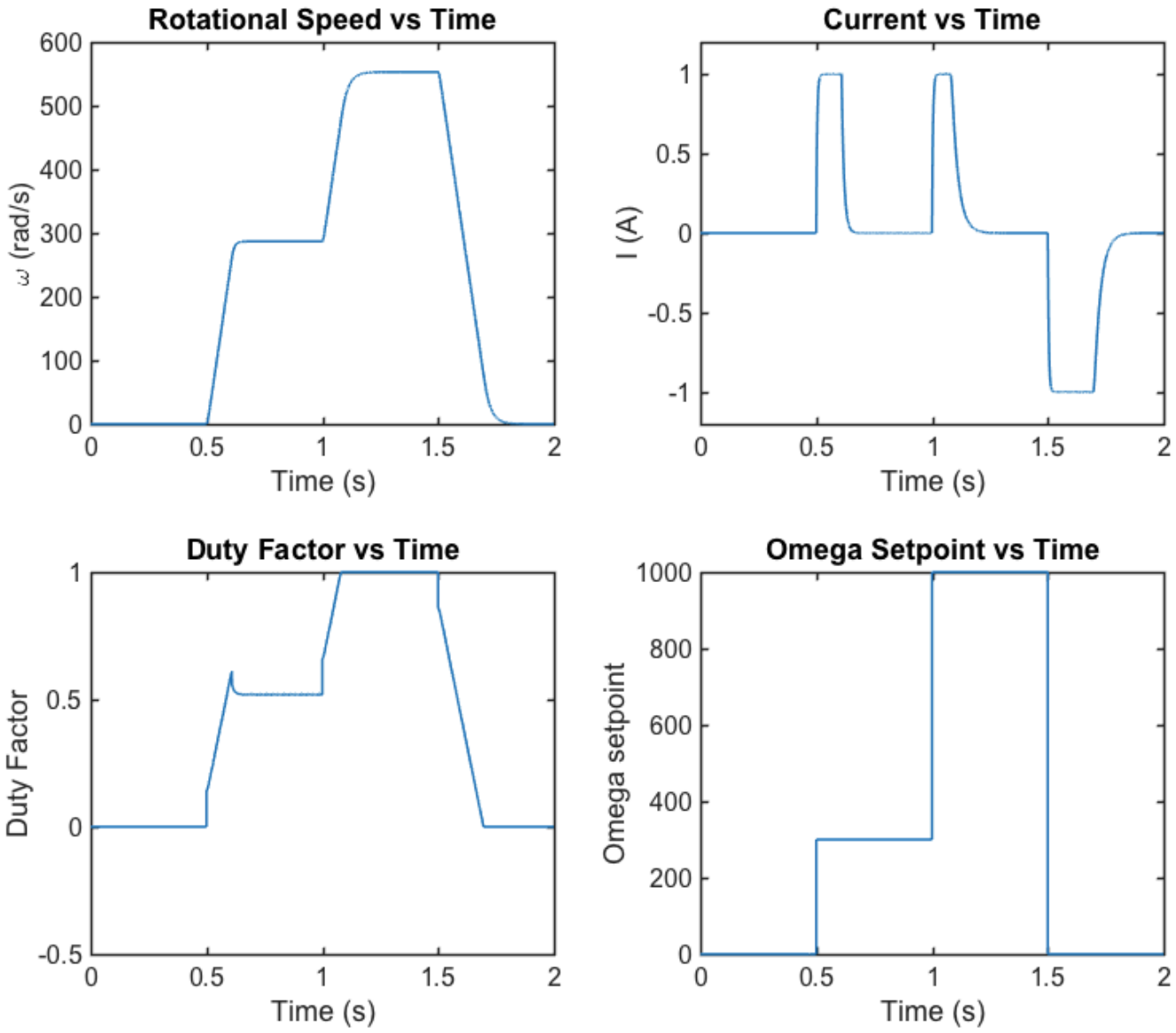


This plot shows that even with the unity gain frequency so far out, we will have a lot of phase marign, thus guaranteeing stability. Closing the loop gives a step response of:



As you can see we have some error but it is negligible.

Putting this result into a Matlab script give us the following result:



As you can see initially we have 0 current and 0 velocity. At time = 0.5s we ramp up the setpoint velocity to 300 rad/s. The current shoots up to a maximum of 1A and stays there until the duty factor reaches around 0.5, then the current exponentially decays, and our velocity reaches 300 rad/s. Our rate of change of velocity is limited by our arbitrary 1A current limit. Then at time = 1s, we ramp up the setpoint velocity to 1000 rad/s. The current shoots up to a maximum of 1A and stays there until the duty factor reaches 1, then the current exponentially decays, and our velocity reaches ~550 rad/s. Our rate of change of velocity is limited by our arbitrary 1A current limit. Notice how we do not reach our desired 1000 rad/s velocity. This is because the motor physically cannot increase its velocity more than ~550 rad/s at a duty factor of 1.0. We thus have shown that we can safely operate the motor at its maximum velocity and the controller will not break even if the setpoint is much higher than the maximum velocity. Furthermore, we can see that at time = 1.5s we give the controller a 0 velocity setpoint, and the duty factor and velocity ramp down to 0. The current is also bounded

to -1A during this downfall, thus proving our motor can be safely operated at its intended current limits. The steepness or rate of change of velocity can be increased by allowing the motor to have a higher maximum current.

-5 pts for ringing/oscillation in speed
-5 pts for not bounding duty factor 0-1
-5 pts for breaking arbitrary current limit
-5 pts for overshoot in speed
-10 pts for pure D control
-10 pts for pure I control
-10 pts for leveling off at a speed higher than 550 rad/s
-5 pts for not showing setpoint vs time
-5 pts for not showing duty factor vs time
-5 pts for not showing speed vs time
-5 pts for not showing current vs time
Minimum for this problem is 0/60.

Controller Function (15pts):

```
function [ pwm, error, integral, derivative, pwm_max ] = EE255HW3_motor_controller( dt, omega, I, set_omega, integral, derivative, errorlast, Vss )
    Kp = 10;
    Kd = 0.1;
    Ki = 0;
    I_max = 1;
    R = 3.401;
    Km = 220 * 2 * pi / 60;
    Ke = 1./Km;
    error = set_omega - omega;
    pwm = (Kp * error) + (Ki * integral) + (Kd * derivative);
    % calculate max and min pwm that will keep us below current limit
    Vin_max = I_max*R + Ke*omega;
    Vin_min = -I_max*R + Ke*omega;
    df_max = min(Vss,Vin_max)/Vss;
    df_min = max(0,Vin_min)/Vss;
    pwm_max = (df_max)*255;
    pwm_min = (df_min)*255;
    if (pwm > min(pwm_max,255))
         pwm = min(pwm_max,255);
    elseif (pwm < max(pwm_min,0))
         pwm = max(pwm_min,0);
    else
         integral = integral + (error * dt);
    end
    pwm = round(pwm);

    derivative = (error - errorlast)/dt;
end
```

Main function (30 pts):

```
t_step = 1e-4;
t_max = 2;
n_step = t_max/t_step;
t = 0;
error_nback = 0;
T_arr = linspace(0,t_step*n_step,n_step);
I_arr = zeros(1,n_step);
Omega_arr = zeros(1,n_step);
df = 0;
Vss_arr = ones(1,n_step);
Vss_arr = Vss_arr*24;
I = 0;
omega = 0;
set_omega = zeros(1,n_step);
set_omega(1:int16(n_step/4)) = 0;
set_omega(int16(n_step/4):int16(2*n_step/4)) = 300;
set_omega(int16(2*n_step/4):int16(3*n_step/4)) = 1000;
set_omega(3*int16(n_step/4):end) = 0;
error = set_omega(1) - omega;
integral = 0;
derivative = 0;
Errors = zeros(1,n_step);
Df = zeros(1,n_step);
for n=1:n_step
    Errors(n) = error;
    [omega, I] = hw3_motor_sim( t_step, df, Vss_arr(n), I, omega );
    if n <= error_nback
        [ pwm, error, integral, derivative, pwm_max ] = EE255HW3_motor_controller( t_step, omega, I, set_omega(n), integral,
            derivative, Errors(1), Vss_arr(n) );
    else
        [ pwm, error, integral, derivative, pwm_max ] = EE255HW3_motor_controller( t_step, omega, I, set_omega(n), integral,
            derivative, Errors(n-error_nback), Vss_arr(n));
    end
    Df(n) = df;
    df = pwm/255;
    I_arr(n) = I;
    Omega_arr(n) = omega;
    t = t + t_step;
end
figure
subplot(221);
plot(T_arr, Omega_arr) %
ylabel('\omega (rad/s)') %
xlabel('Time (s)');
title('Rotational Speed vs Time');
subplot(222);
plot(T_arr, I_arr) %
ylabel('I (A)') %
xlabel('Time (s)');
title('Current vs Time');
axis([0 t_max -1.2 1.2]);
subplot(223);
plot(T_arr, Df) %
ylabel('Duty Factor') %
xlabel('Time (s)');
title('Duty Factor vs Time');
axis([0 t_max -0.5 1]);
subplot(224);
plot(T_arr, set_omega);
ylabel('Omega setpoint');
xlabel('Time (s)');
title('Omega Setpoint vs Time');
```