Lab 9: Basic DC Motor Feedback Control Justin Garcia

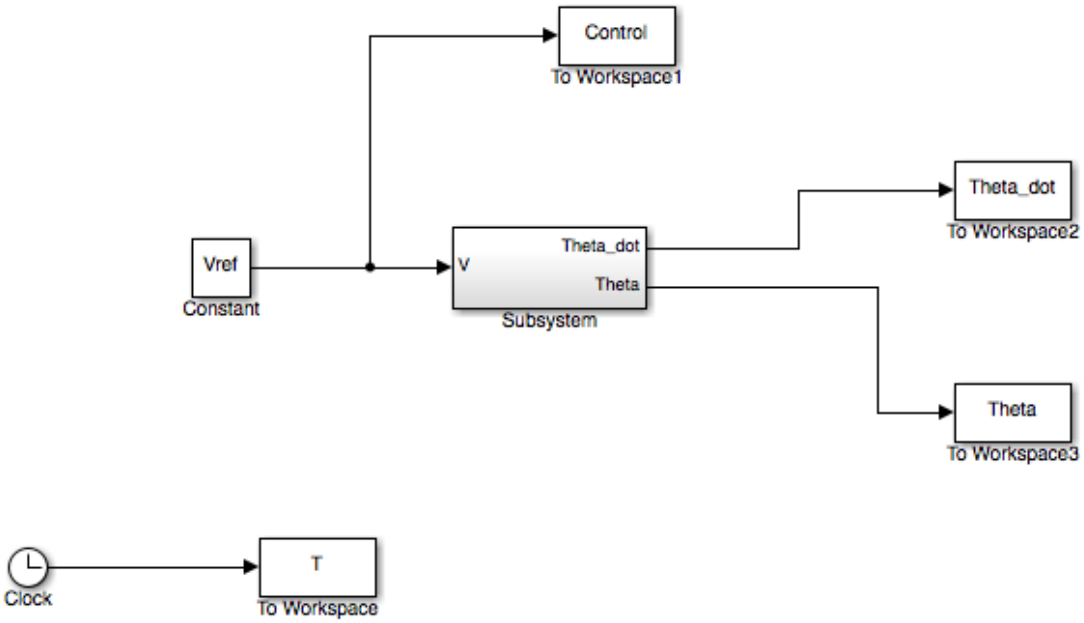
Table of Contents

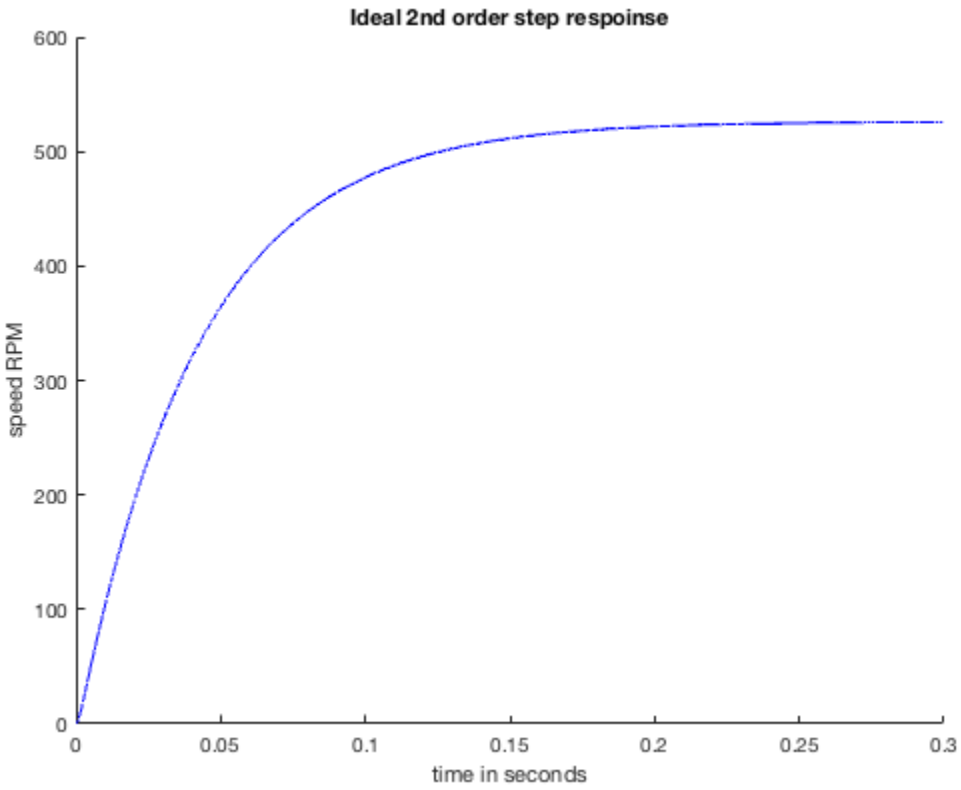
Part 1: Ideal 2nd order DC Motor Model 1
Part 2: Ideal PI Speed Control of DC Motor Model (Simulation) 3
Part Two Questions 4
Part Three P Controller 5
Conclusion 7

The Purpose of this las was to show the students how a DC motor works and behaves when implementing P controllers and PI controllers. Initially I had to run the motor and graph the velocity over time. By using different controllers i was able to determine which was better at stabilizing the position of the motor as well as determine which controller had less error. This lab helped me understand how controllers work and how the sampling time can affect the ability of the controller to respond well, specifically the lag between inputs can start to take over the controllers ability to work properly.

Part 1: Ideal 2nd order DC Motor Model

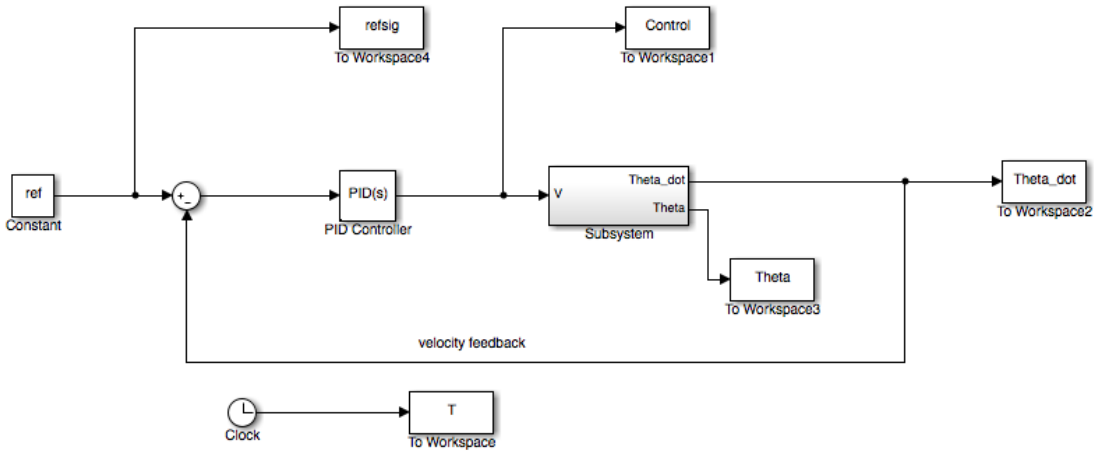
```
open_system('Lab9_Ideal_1.slx')
%open_system('Lab9_Ideal_1.slx/Subsystem')
% NXT motor parameter:
Rm = 5.262773292;       % ohms
Kb = 0.4952900056;      % Vs/rad
Kt = 0.3233728703;      % Nm/A
Bm = 0.0006001689451;   % Nms/rad (Viscous Friction)
Lm = 0.0047;            % H
Jm = 0.001321184025;    % kgm^2 (combined J)

% simulation parameters
sim_stop_time = 0.3;    % simulation length - seconds
sim_step_size = 0.0001; % how often log simulation data

% plot the step response;
Vref = 9;               % reference step in voltage

% run simulink diagram to solve system response
sim('Lab9_Ideal_1.slx')

%plot results
figure(1), hold on
h_step = plot(T, Theta_dot/(2*pi)*60, 'b');
xlabel('time in seconds')
ylabel('speed RPM')
title('Ideal 2nd order step respoinse')
hold off
```

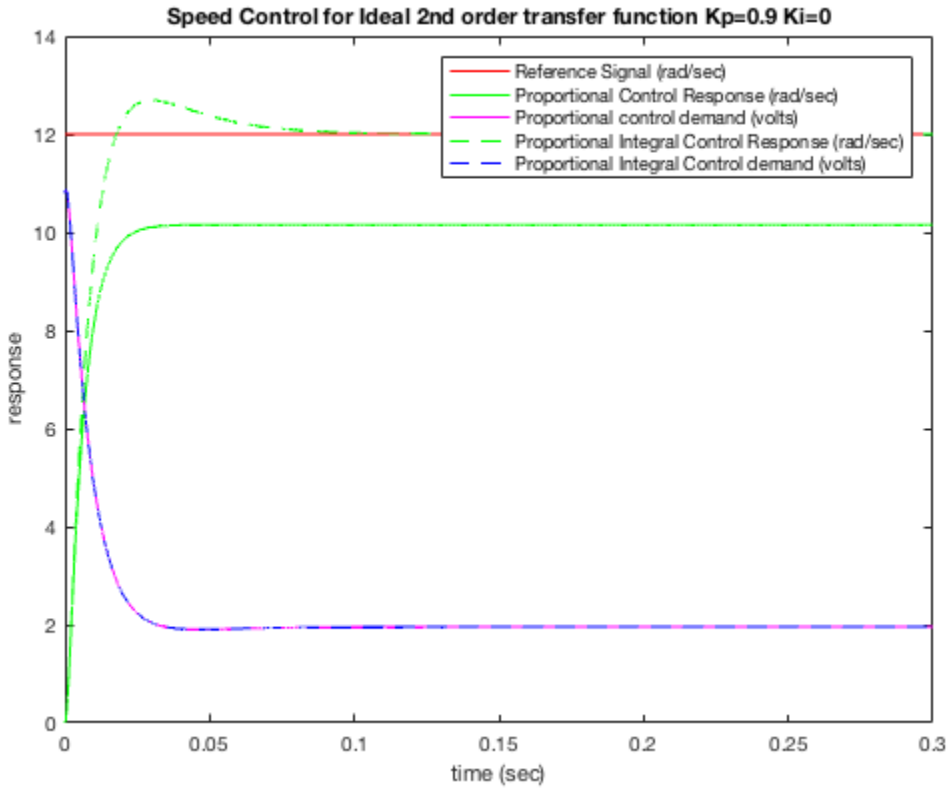

Control
To Workspace1
Theta_dot
To Workspace2
Vref
Constant
V
Theta_dot
Theta
Subsystem
Theta
To Workspace3
Clock
T
To Workspace


Ideal 2nd order step respoinse
speed RPM
time in seconds
0
100
200
300
400
500
600
0
0.05
0.1
0.15
0.2
0.25
0.3

Part 2: Ideal PI Speed Control of DC Motor Model (Simulation)

```
open_system('Lab9_Ideal_PID.slx')
%open_system('Lab9_Ideal_PID.slx/Subsystem')
% reference speed
ref = 12;               % rad/sec

% Run the simulation with the proportional settings
Kp = .9;
Ki = 0;
Kd = 0;                 % for units of rad/sec to Volts

sim('Lab9_Ideal_PID.slx')

%plot results
figure(2),
hKp = plot (T,Theta_dot, 'g'); hold on
hv_demand_Kp = plot(T, control, 'm');
hKp_ref = plot([0 T(end)], [ref ref], 'r');
plot(T, refsig, 'r');
xlabel('time (sec)')
ylabel('response')
title('Speed Control for Ideal 2nd order transfer function Kp=0.9
 Ki=0')

% Run the simulation with the proportional settings
Kp = .9;
Ki = 38;
Kd = 0;                 % for units of rad/sec to Volts

sim('Lab9_Ideal_PID.slx')

%plot results
figure(2),
hKi = plot (T,Theta_dot, 'g--'); hold on
hv_demand_Ki = plot(T, control, 'b--');

% create legend
legend([hKp_ref hKp hv_demand_Kp hKi hv_demand_Ki],...
    {'Reference Signal (rad/sec)',...
    'Proportional Control Response (rad/sec)',...
    'Proportional control demand (volts)',...
    'Proportional Integral Control Response (rad/sec)',...
    'Proportional Integral Control demand (volts)'})
```





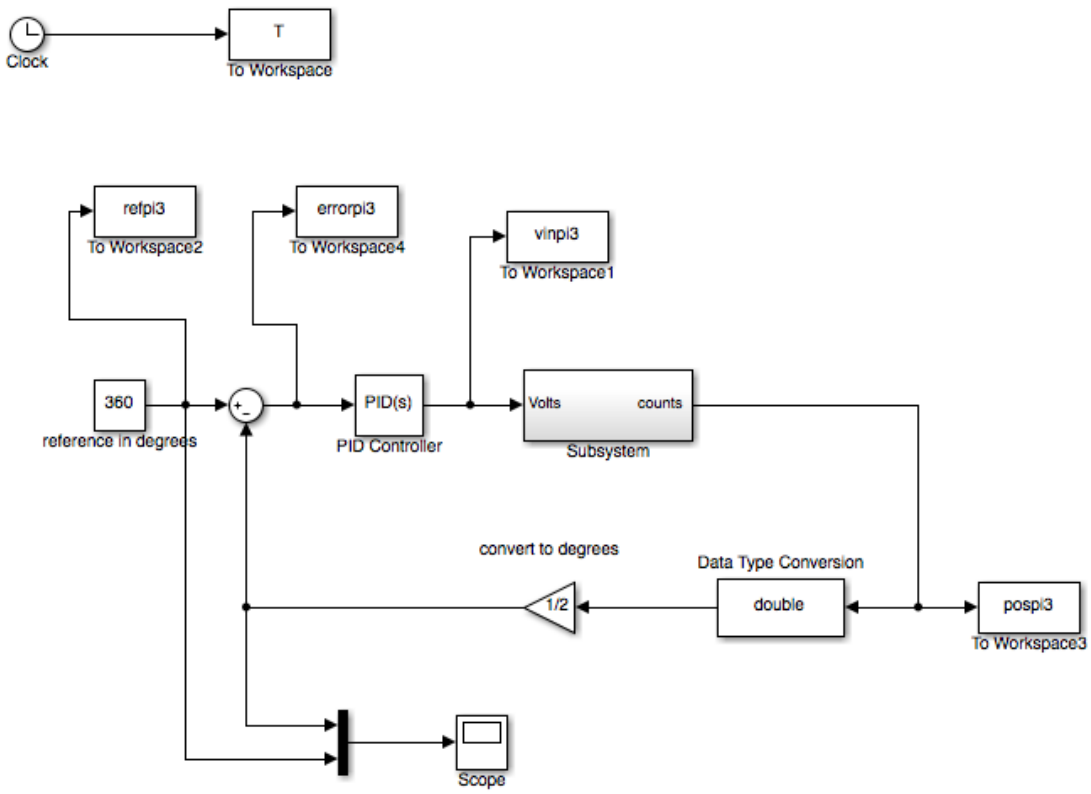
Part Two Questions

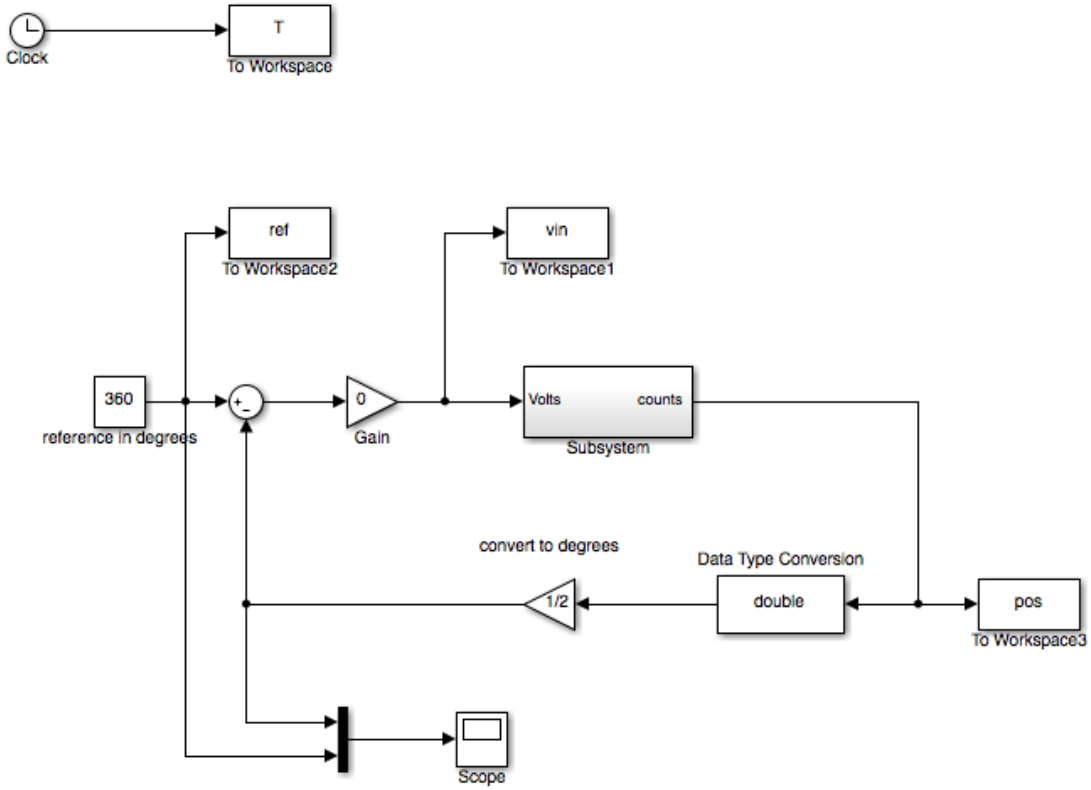
```
%What do you notice about the steady state error with just a
 proportional controller?
%We can see from the graph that the proportional controller does not
 allow
%the oputput to reach the reference signal.

%What is the maximum voltage the controller uses to obtain the
 response?
```

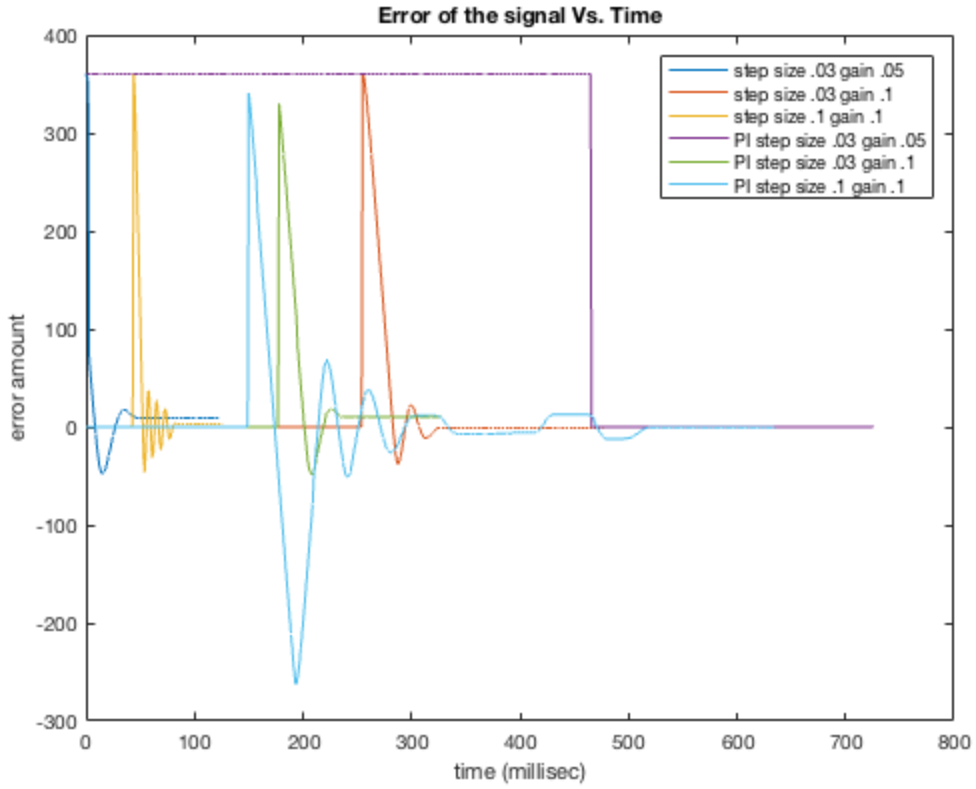
```
%Both controllers start by demanding just over ten volts from the
 system
%and quickly go down to two volts.

%How does the PI controller perform?
%The PI controller overshoots the reference signal but when the
 controller
%calms down the output matches the reference signal which is what is
%desirable when making a controller.
```

Part Three P Controller

```
open_system('Lab9_Experiment_P.slx')
open_system('Lab9_Experiment_Pi.slx')

figure(3)
first = plot(T_error,error);
hold on
second = plot(T_error2,error2);
third = plot(T_error3,error3);
fourth = plot(T_errorpi1,errorpi1);
fifth = plot(T_errorpi2,errorpi2);
sixth = plot(T_errorpi3,errorpi3);

xlabel('time (millisec)')
ylabel('error amount')
title('Error of the signal Vs. Time')

legend([first second third fourth fifth sixth],...
    {'step size .03 gain .05',...
    'step size .03 gain .1',...
    'step size .1 gain .1',...
    'PI step size .03 gain .05',...
    'PI step size .03 gain .1',...
    'PI step size .1 gain .1'})
```


Clock
T
To Workspace
refpi3
To Workspace2
errorpi3
To Workspace4
vinpi3
To Workspace1
360
reference in degrees
PID(s)
PID Controller
Volts
counts
Subsystem
convert to degrees
Data Type Conversion
1/2
double
pospi3
To Workspace3
Scope


Clock
T
To Workspace
ref
To Workspace2
vin
To Workspace1
360
reference in degrees
0
Gain
Volts
counts
Subsystem
convert to degrees
Data Type Conversion
1/2
double
pos
To Workspace3
Scope



Conclusion

```
%From the graph we can see that the PI controller do a much better job
 at
%bringing the error back to zero completely given enough time whereas
 the
%proportional controllers miss the zero error mark. It is hard to
 judge
%the controllers because when i took the data i didnt activate the
 change
%in refernce right away and at different times. Also for the last
 graph of
%step size .1 and gain .1 i moved the wheel after it reached steady
 state
%which you can see. However the increase in step time caused both
 types of
%controllers to take longer to reach a steady statea as the controller
%overcompensated and then took another reading, in a sense following
 the
%lag. The purely proportional controllers were much faster to react.
 One
%of my PI graphs is impossibly fast and is likely an error on my part
 to
%capture the correct data set.
```

Published with MATLAB® R2016b