Arduino Step Response Experiment

Mini Project for SEED Lab

Alex Curtis

This script runs the step response experiment I created in stepResponse.ino, interperets the data, makes some beautiful plots, determines an accurate transfer function for the motor, and tunes a PI controller to desired specifications.

Upload the stepResponse code to the Arduino before running.

Clear previous workspace data

```
clear dataTable dataArray
```

Automatically find the port the Arduino is connected to

```
allPorts = serialportlist;  % list of all available ports
pat = "/dev/cu.usbmodem";   % text to look for

% logical array of matching ports
matchingPorts = startsWith(allPorts, pat);

if any(matchingPorts)                   % If any matches were found,
    numMatches = sum(matchingPorts);    % Count the number of matches.
    if size(numMatches) > 1             % If multiple matching ports were found,
        error("Multiple ports found")   % throw an error.
    else                                % If only one match was found,
        port = allPorts(matchingPorts); % that string is the arduino port.
    end
else                                    % If no ports were found,
    error("Port not found")             % throw an error.
end
```

Create a serialport object to talk to the Arduino

```
ard = serialport(port, 115200);   % creates a device object (port, baud)
configureTerminator(ard, "CR/LF") % sets the terminator
```

Wait for the Arduino

When the Arduino is ready, it sends "Ready!"

```
readline(ard);
```

Tell the Arduino to start the step experiment

```
disp('Starting Counting Event in Arduino')
```

```
Starting Counting Event in Arduino
```

```
writeline(ard,"S"); % sends the start command to the Arduino
```

Read data from Arduino

```
k = 0; % array index

% Read the first line of output from the Arduino
newData = readline(ard);

% Read new data until the Arduino signals it's done
while (~strncmp(newData,'Finished',8))
    k = k + 1; % k++

    % save the line of text to stringArray(k)
    stringArray(k,:) = split(newData)';

    % Read a new line of data from Arduino
    newData = readline(ard);
end

clear ard; % Disconect from the Arduino
```

```
%cell2mat(stringArray)
% Transpose stringArray
%stringArray = stringArray';

% change string data to cell array using tab delimiter
dataArray = str2double(stringArray);

% Make time start at 0 and convert from ms to s
dataArray(:,1) = (dataArray(:,1)-1000)/1000;

% Create a table to store the data
dataTable = array2table(dataArray,"VariableNames",{'time', 'motor', 'vel', 'pos'})
```

dataTable = 201×4 table

	time	motor	vel	pos
1	0	255	0	0
2	0.0050	255	0	0
3	0.0100	255	1.5708	0.0079
4	0.0150	255	2.7489	0.0216
5	0.0200	255	3.9270	0.0412
6	0.0250	255	4.7124	0.0648
7	0.0300	255	4.7124	0.0884
8	0.0350	255	5.8905	0.1178
9	0.0400	255	6.6759	0.1512
10	0.0450	255	7.0686	0.1865

	time	motor	vel	pos
11	0.0500	255	8.2467	0.2278
12	0.0550	255	8.6394	0.2710
13	0.0600	255	9.4248	0.3181
14	0.0650	255	9.4248	0.3652
15	0.0700	255	9.8175	0.4143
16	0.0750	255	8.2467	0.4555
17	0.0800	255	10.6029	0.5085
18	0.0850	255	10.9956	0.5635
19	0.0900	255	10.9956	0.6185
20	0.0950	255	11.3883	0.6754
21	0.1000	255	11.7810	0.7343
22	0.1050	255	12.1737	0.7952
23	0.1100	255	12.1737	0.8561
24	0.1150	255	9.8175	0.9052
25	0.1200	255	12.5664	0.9680
26	0.1250	255	12.9591	1.0328
27	0.1300	255	12.5664	1.0956
28	0.1350	255	13.3518	1.1624
29	0.1400	255	12.9591	1.2272
30	0.1450	255	13.3518	1.2939
31	0.1500	255	13.3518	1.3607
32	0.1550	255	13.3518	1.4275
33	0.1600	255	10.6029	1.4805
34	0.1650	255	13.7445	1.5492
35	0.1700	255	13.7445	1.6179
36	0.1750	255	13.7445	1.6866
37	0.1800	255	13.7445	1.7554
38	0.1850	255	13.7445	1.8241
39	0.1900	255	13.7445	1.8928
40	0.1950	255	14.1372	1.9635
41	0.2000	255	11.3883	2.0204
42	0.2050	255	14.1372	2.0911
43	0.2100	255	13.7445	2.1598

	time	motor	vel	pos
44	0.2150	255	14.5299	2.2325
45	0.2200	255	14.1372	2.3032
46	0.2250	255	14.1372	2.3739
47	0.2300	255	14.5299	2.4465
48	0.2350	255	14.1372	2.5172
49	0.2400	255	14.5299	2.5899
50	0.2450	255	11.3883	2.6468
51	0.2500	255	14.5299	2.7194
52	0.2550	255	14.1372	2.7901
53	0.2600	255	14.5299	2.8628
54	0.2650	255	14.5299	2.9354
55	0.2700	255	14.5299	3.0081
56	0.2750	255	14.5299	3.0807
57	0.2800	255	14.5299	3.1534
58	0.2850	255	11.7810	3.2123
59	0.2900	255	14.5299	3.2849
60	0.2950	255	14.5299	3.3576
61	0.3000	255	14.5299	3.4302
62	0.3050	255	14.5299	3.5029
63	0.3100	255	14.5299	3.5755
64	0.3150	255	14.5299	3.6482
65	0.3200	255	14.5299	3.7208
66	0.3250	255	14.5299	3.7935
67	0.3300	255	11.7810	3.8524
68	0.3350	255	14.5299	3.9250
69	0.3400	255	14.5299	3.9977
70	0.3450	255	14.9226	4.0723
71	0.3500	255	14.5299	4.1449
72	0.3550	255	14.5299	4.2176
73	0.3600	255	14.5299	4.2902
74	0.3650	255	14.9226	4.3649
75	0.3700	255	11.3883	4.4218
76	0.3750	255	14.9226	4.4964

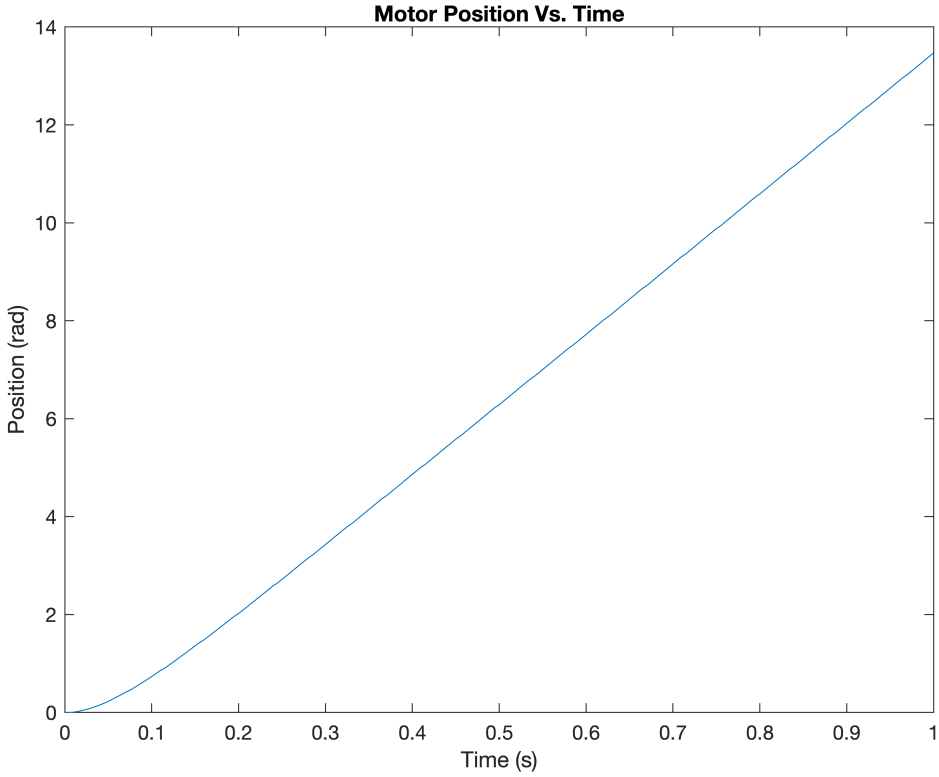
	time	motor	vel	pos
77	0.3800	255	14.5299	4.5691
78	0.3850	255	14.5299	4.6417
79	0.3900	255	14.9226	4.7163
80	0.3950	255	14.5299	4.7890
81	0.4000	255	14.9226	4.8636
82	0.4050	255	14.5299	4.9362
83	0.4100	255	14.5299	5.0089
84	0.4150	255	11.7810	5.0678
85	0.4200	255	14.9226	5.1424
86	0.4250	255	14.5299	5.2150
87	0.4300	255	14.5299	5.2877
88	0.4350	255	14.9226	5.3623
89	0.4400	255	14.5299	5.4350
90	0.4450	255	14.5299	5.5076
91	0.4500	255	14.9226	5.5822
92	0.4560	255	14.5299	5.6549
93	0.4600	255	11.7810	5.7138
94	0.4650	255	14.5299	5.7864
95	0.4700	255	14.9226	5.8610
96	0.4750	255	14.5299	5.9337
97	0.4800	255	14.5299	6.0063
98	0.4850	255	14.9226	6.0809
99	0.4900	255	14.5299	6.1536
100	0.4950	255	14.5299	6.2262

⋮

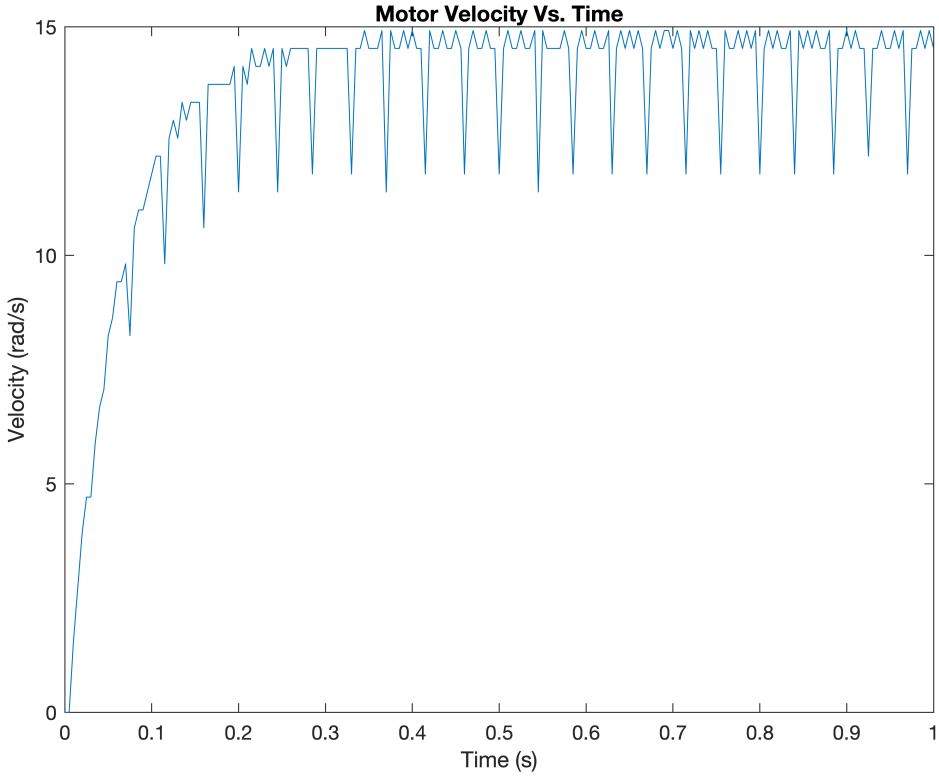
```
% Save data to a .mat file
save stepData.mat dataTable
```

Plot the data (raw motor response)

```
plot(dataTable.time,dataTable.pos)
title('Motor Position Vs. Time')
xlabel('Time (s)')
ylabel('Position (rad)')
```



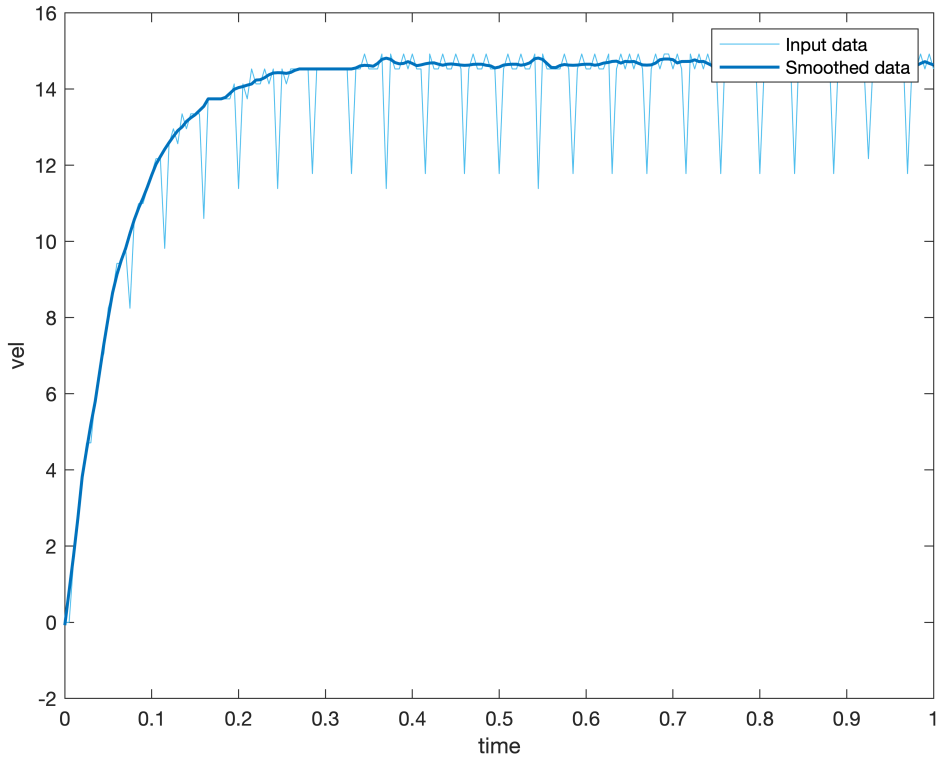

```
plot(dataTable.time,dataTable.vel)
title('Motor Velocity Vs. Time')
xlabel('Time (s)')
ylabel('Velocity (rad/s)')
```



Process data

Smoothing the velocity data gets rid of any noise that might mess up my resulting transfer function.

```
% Smooth input data
newT = smoothdata(dataTable,"rlowess","SmoothingFactor",0.2,...
    "DataVariables","vel","SamplePoints",dataTable.time);

% Display results
clf
plot(dataTable.time,dataTable.vel,"Color",[77 190 238]/255,...
    "DisplayName","Input data")
hold on
plot(dataTable.time,newT.vel,"Color",[0 114 189]/255,"LineWidth",1.5,...
    "DisplayName","Smoothed data")
hold off
legend
ylabel("vel")
xlabel("time")
```



Max analogWrite command = 255

max motor.setM1Speed() command = 400

Since the motor command is a number out of 400, I need to divide the velocity by 400 to get a unit step response.

THIS HAS NOTHING TO DO WITH THE VOLTAGE ON THE BATTERY.

```
newT.velScaled = newT.vel/400;
```

Plot the processed data (motor unit step response)

```
plot(newT.time,newT.velScaled)
xlabel('Time (s)')
ylabel('Velocity (rad/s)')
title('Motor Unit Step Response (Scaled Motor Velocity Vs. Time)')
hold on
```

Determine Transfer Function of Motor Command to Velocity

Find the steady state velocity of the motor (K)

```
K = newT.velScaled(newT.time > 0.4);
K = mean(K)
```

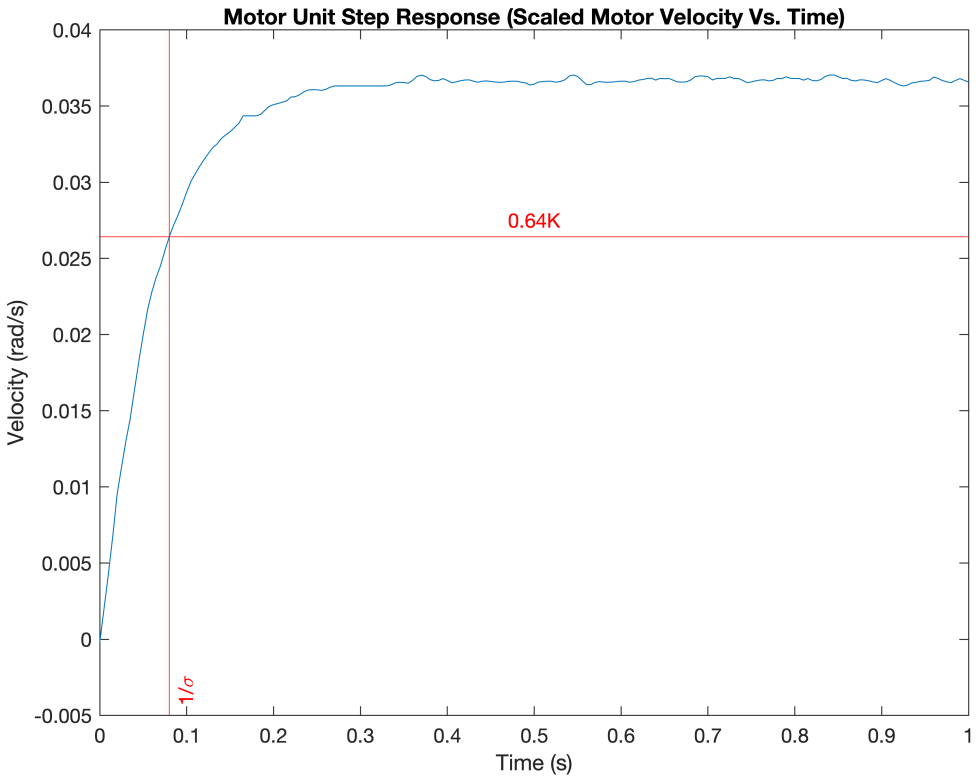
```
K = 0.0367
```

find 0.64 * K and the time constant to find sigma

```
magAtTimeConstant = max(newT.velScaled(newT.velScaled <= 0.64*K+0.1*K & newT.velScaled
```

```
magAtTimeConstant = 0.0264
```

```
timeConstant = newT.time(newT.velScaled == max(magAtTimeConstant));

yline(magAtTimeConstant,'Label','0.64K','Color','r',LabelHorizontalAlignment='center')
xline(timeConstant,'Label','1/\sigma','Color','r','LabelVerticalAlignment','bottom')
hold off
```




```
sigma = 1./timeConstant;
sigma = mean(sigma)
```

```
sigma = 12.5000
```

With K and sigma, create the velocity and position transfer functions

```
s = tf('s');
velTF = K*(sigma)/(s+sigma)
```

```
velTF =

   0.4586
  --------
  s + 12.5

Continuous-time transfer function.
```

```
posTF = velTF*(1/s)
```

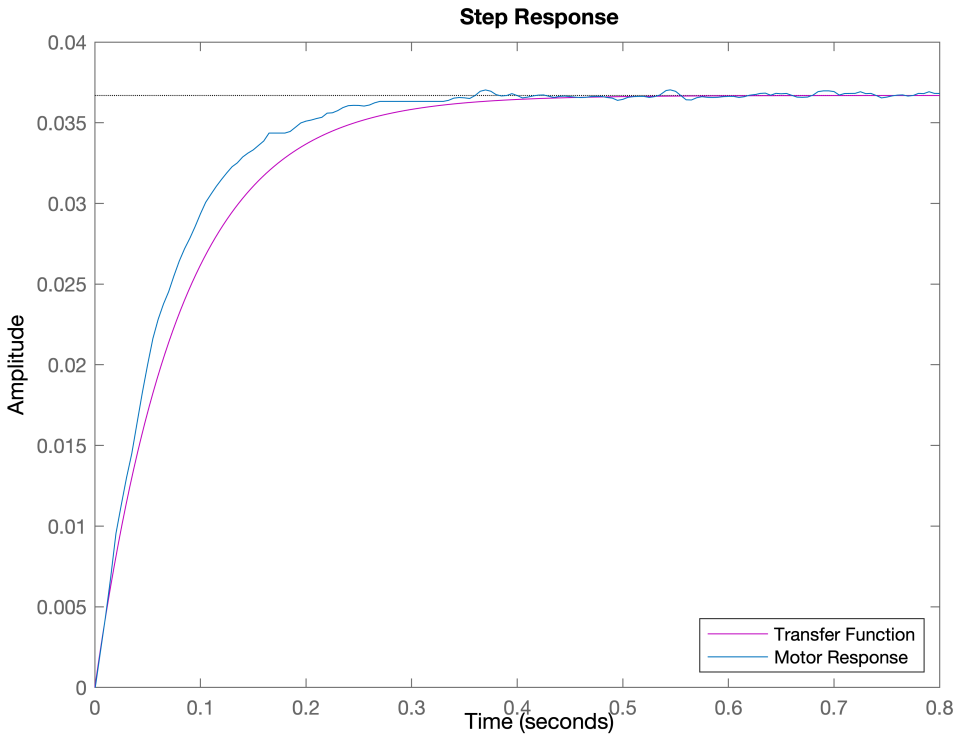
```
posTF =

     0.4586
  ------------
  s^2 + 12.5 s

Continuous-time transfer function.
```

Turn the generated transfer functions into coefficient arrays for use in Simulink.

```
posNum = posTF.Numerator;       % Find the numerator coefficients
posNum = posNum{1};             % Convert the numerator to a the right form
posDen = posTF.Denominator;     % Find the denominator coefficients
posDen = posDen{1};             % Convert the denominator to a the right form

velNum = velTF.Numerator;       % Find the numerator coefficients
velNum = velNum{1};             % Convert the numerator to a the right form
velDen = velTF.Denominator;     % Find the denominator coefficients
velDen = velDen{1};             % Convert the denominator to a the right form
```

Compare the Transfer Function with Motor Response

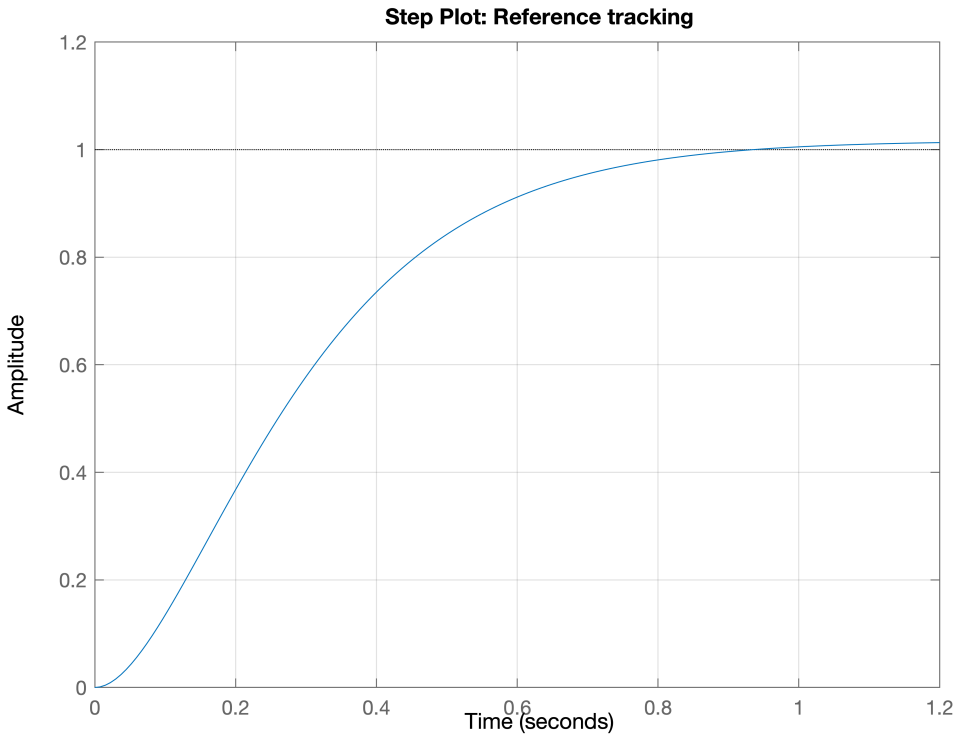
```
step(velTF,'m')
hold on
plot(newT.time,newT.velScaled)
legend('Transfer Function','Motor Response','Location','southeast')
hold off
```



Design a PI Controller to Regulate Position

Design a PI controller that achieves closed loop step response specifications of a rise time of 1 second and an overshoot of less than or equal to 12%, with zero steady state error.

```
% Convert Response Time to Bandwidth
% Bandwidth is equivalent to 2 divided by the Response Time
wc = 2/0.637509;

% Convert Transient Behavior to Phase Margin
% Phase Margin is equivalent to the Transient Behavior multiplied by 100
PM = 100*0.729;

% Define options for pidtune command
opts = pidtuneOptions('PhaseMargin',PM);

% PID tuning algorithm for linear plant model
[C,pidInfo] = pidtune(posTF,'PI',wc,opts);

% Clear Temporary Variables
clear wc PM opts

% Get desired loop response
Response = getPIDLoopResponse(C,posTF,'closed-loop');

% Plot the result
stepplot(Response)
title('Step Plot: Reference tracking')
grid on
```




```
% Display system response characteristics
disp(stepinfo(Response))
```

```
      RiseTime: 0.4962
  TransientTime: 0.7954
   SettlingTime: 0.7954
    SettlingMin: 0.9031
    SettlingMax: 1.0160
      Overshoot: 1.6042
     Undershoot: 0
           Peak: 1.0160
       PeakTime: 1.7171
```

```
% Clear Temporary Variables
clear Response
KP = C.Kp, KI = C.Ki
```

```
KP = 88.1417
KI = 4.8396
```

Discussion of Results

I'm writing this on Monday, 2/28, at 9pm. I have spent hours upon hours of my own time figuring out how to correctly design and implement a controller, and I finally got it working at around 8 pm today. I have put a ridiculous amount of time into creating this matlab script and implementing it on the Arduino, and despite recieving incorrect advice from professors, I managed to design and implement a controller that works exactly as it's intended to. I may not have created a simulink file implementing my controller on the Arduino, but I didn't

have any reason to. If I wasn't the only person working on this subsystem and wasn't taking five other classes, I would genuinely have loved to make a closed loop response experiment to show how great my controller is, but for now, I'm proud of what I've accomplished and thrilled that it's finally working.