Arduino Step Response Experiment

Mini Project for SEED Lab

Alex Curtis

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

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

Wait for the Arduino

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

```
readline(s);
```

Tell the Arduino to start the experiment

```
disp('Starting Counting Event in Arduino')
writeline(s,"S"); % sends the start command to the Arduino
```

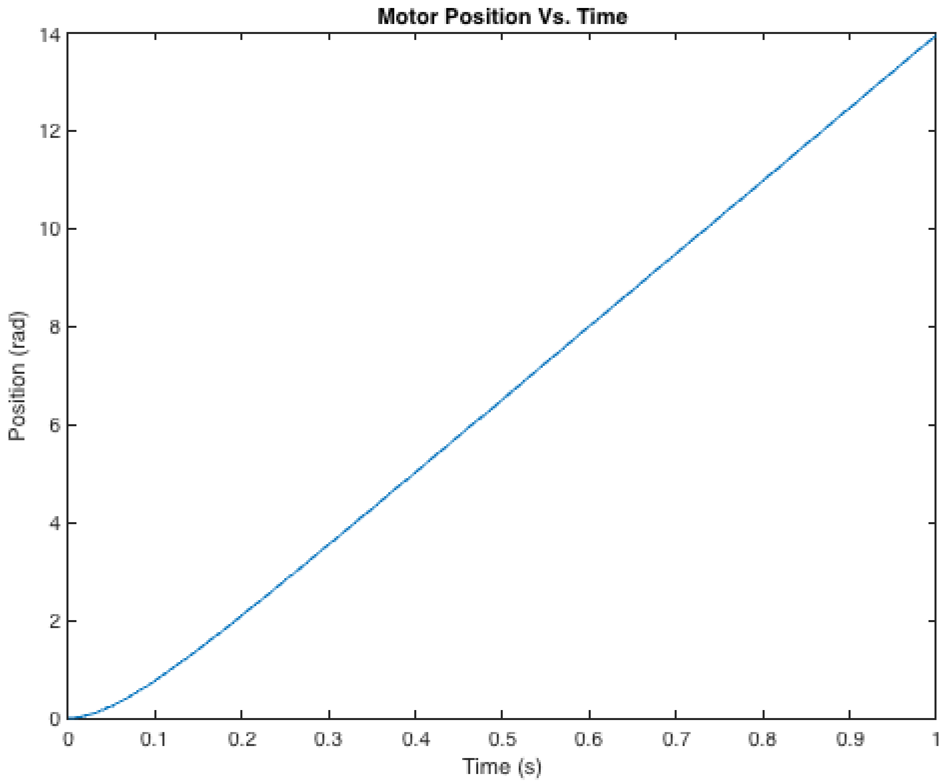
Read data from Arduino

```
k = 1; % array index

% Read the first line of output from the Arduino
stringArray(k) = readline(s);
```

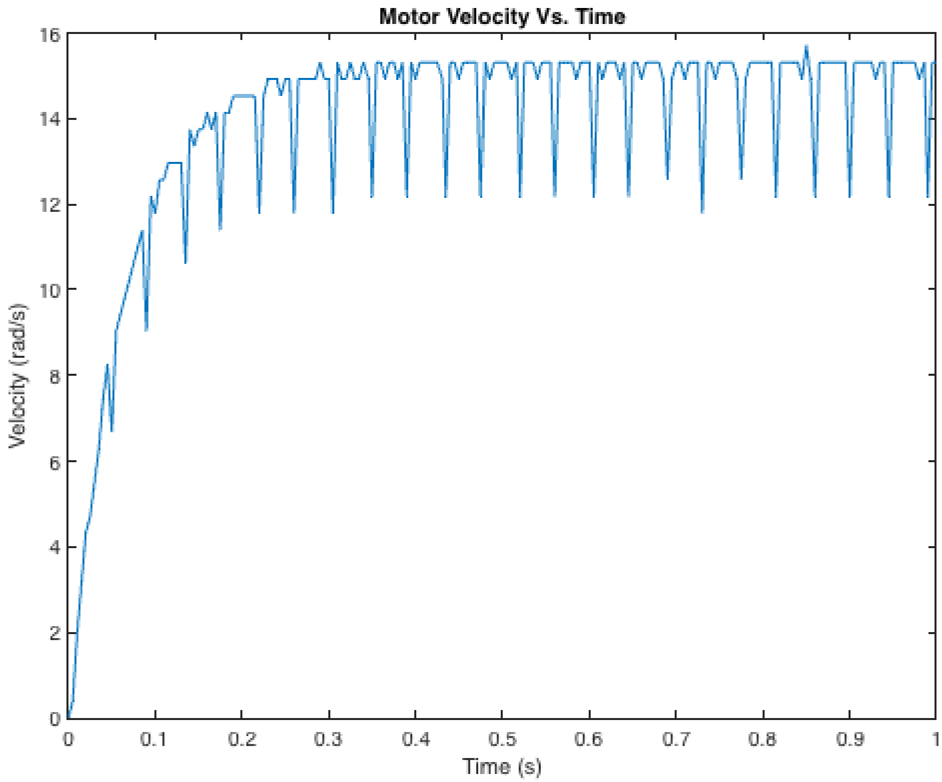
```
% Read new data until the Arduino signals it's done
while (~strncmp(newData,'Finished',8))
    k = k + 1; % k++

    % save the line of text to stringArray(k)
    stringArray(k) = newData;

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

clear s; % Disconect from the Arduino

% Transpose stringArray
stringArray = stringArray';

% change string data to cell array using tab delimiter
dataArray = str2double(split(stringArray,'\t'));

% 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'})

% 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)')
```



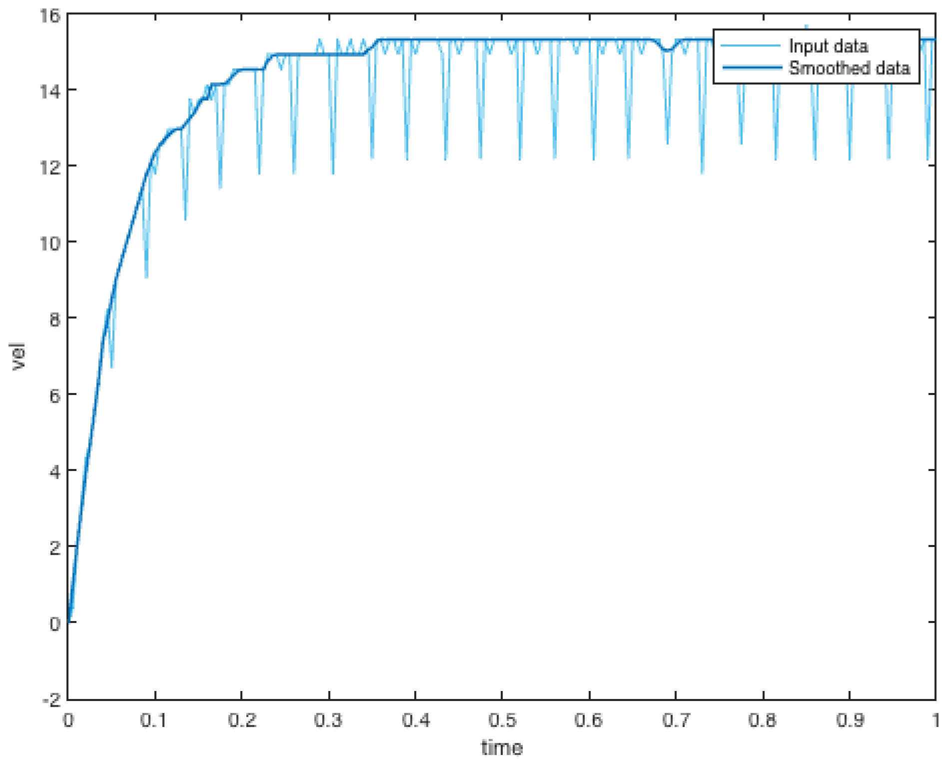

```
plot(dataTable.time,dataTable.vel)
title('Motor Velocity Vs. Time')
xlabel('Time (s)')
ylabel('Velocity (rad/s)')
```



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

Since the motor command is a number out of 255, I need to divide the motor command by 255.

To find the unit step response, I need to scale the velocity output by the scaled motor command.

```
newT.velScaled = newT.vel/(mean(newT.motor)/255);
```

Plot the processed data (motor unit step response)

```
plot(newT.time,newT.vel)
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)

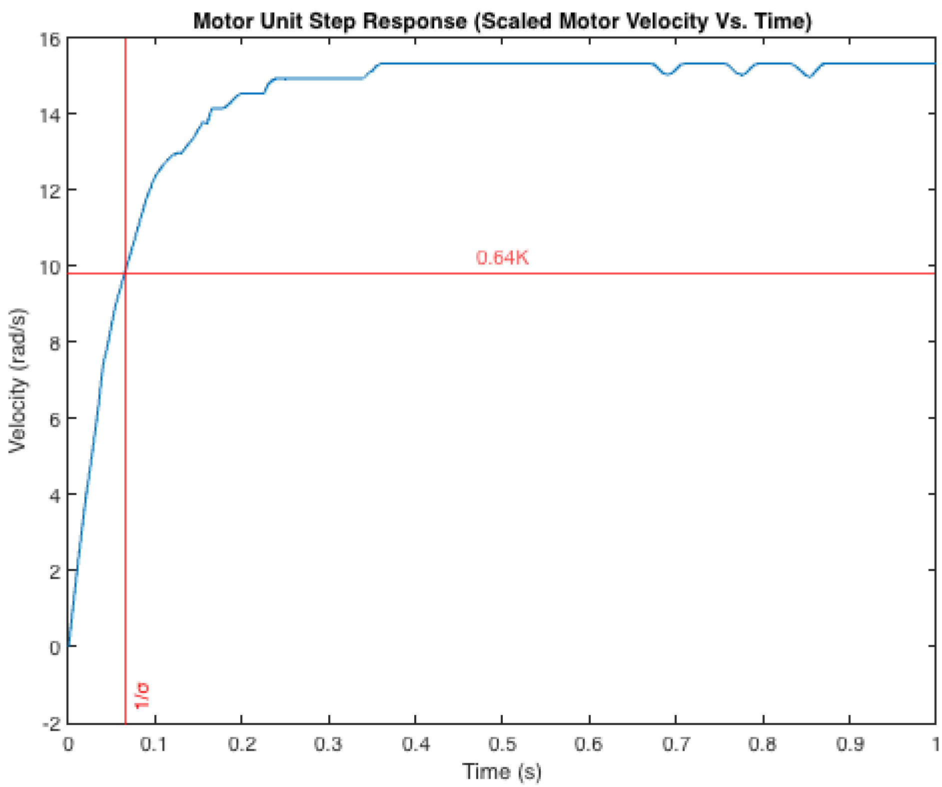
```
K = newT.vel(newT.time > 0.4);
K = mean(K)
```

```
K = 15.2848
```

find 0.64 * K and the time constant to find sigma

```
magAtTimeConstant = newT.vel(newT.vel <= 0.64*K+0.1 & newT.vel >= 0.64*K-0.1);
timeConstant = newT.time(newT.vel == 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 = 15.3846
```

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

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

```
velTF =

    235.2
  ---------
  s + 15.38

Continuous-time transfer function.
```

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

```
posTF =

      235.2
  -------------
  s^2 + 15.38 s
```

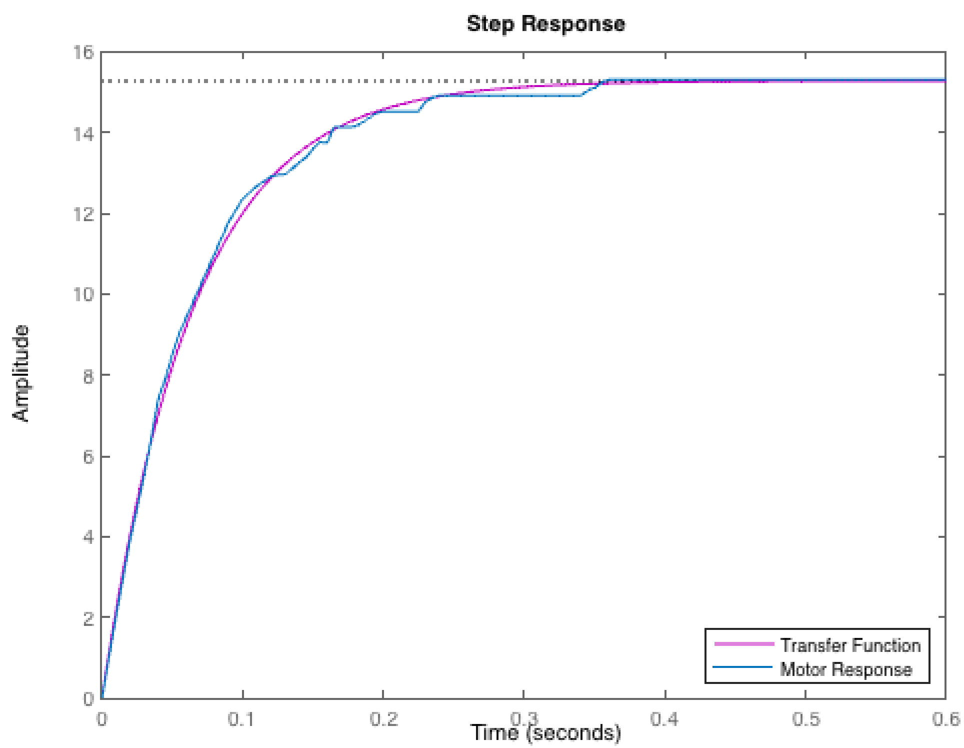
```
Continuous-time transfer function.
```

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

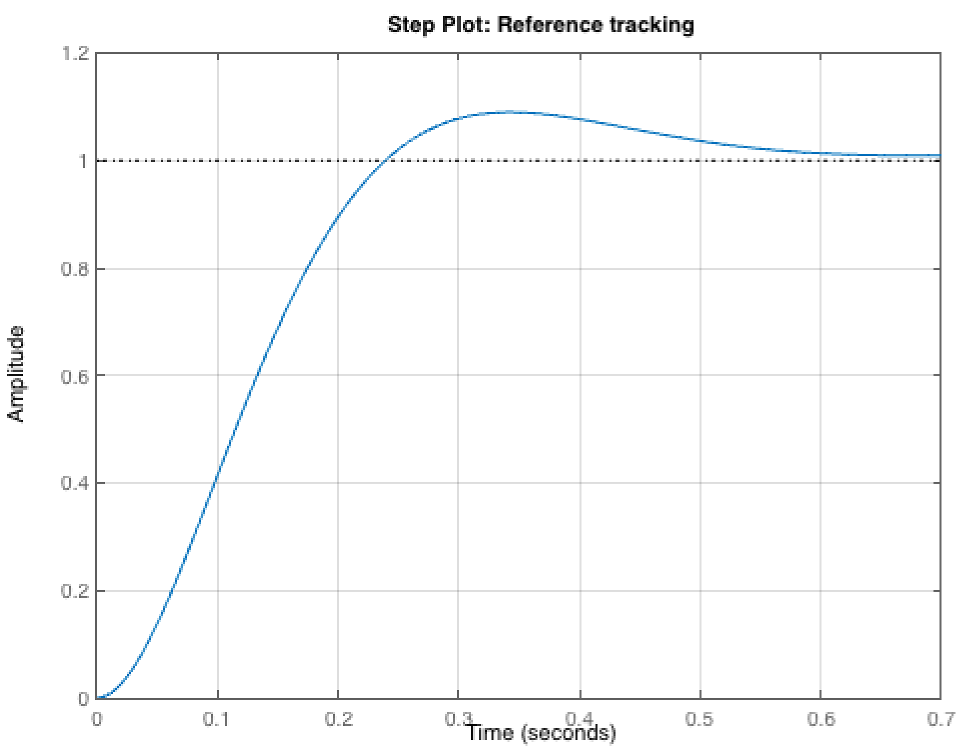
```
step(velTF,'m')
hold on
plot(newT.time,newT.vel)
legend('Transfer Function','Motor Response','Location','southeast')
hold off
```



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
wc2 = 2/0.242;

% Convert Transient Behavior to Phase Margin
% Phase Margin is equivalent to the Transient Behavior multiplied by 100
PM2 = 100*0.729;

% Define options for pidtune command
opts2 = pidtuneOptions('PhaseMargin',PM2);

% PID tuning algorithm for linear plant model
[C,pidInfo] = pidtune(posTF,'PI',wc2,opts2);

% Clear Temporary Variables
clear wc2 PM2 opts2

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

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




```
% Clear Temporary Variables
clear Response
```

```
Kp = C.Kp
```

```
Kp = 0.6137
```

```
Ki = C.Ki
```

```
Ki = 0.0888
```

Closed loop step response

```
% step(feedback(C*posTF,1),'m')
% hold on
% plot(newT.time,newT.vel)
% legend('Controlled TF','Motor Response','Location','southeast')
% hold off
```