

```

% Senior Thesis
% Larry Loprete, 2019

% DESCRIPTION: This code interfaces between MATLAB and the JRK motor
% controller, allowing testing of the JRK controller's behavior to
% particular types of inputs that are standard in control theory, such
% as step, impulse, and low/high frequency sine waves.

% USE: Select one of the 4 available test inputs to send to your JRK using
% the TEST flag. Then, set other test parameters under "DEFINE UNIVERSAL
% TEST PARAMETERS." Next, scroll down to the appropriate test and edit the
% values under "VARIABLES TO SET." Plug in one JRK controller configured to
% Serial/USB communication into this computer and run the MATLAB script.

clear all; % Resets variable and arrays sizes to 0

% SELECT WHICH TEST TO PERFORM USING THIS FLAG
TEST = 1;
% TEST OPTIONS
% 0 = setpoint only
% 1 = step input
% 2 = impulse input
% 3 = sinusoidal input

% DEFINE UNIVERSAL TEST PARAMETERS/FLAGS
DURATION = 10;      % in sec
PLOT_DATA = 0;      % 0 = OFF, 1 = ON

% - - - - -

% Clears MATLAB command prompt, closes windows, initializes index value for
% data collection and prints a newline
clc;
close all;
j = 1; % index for data collection (if used)
fprintf("\n");

% Testing command line interface to ensure it is working (status = 0)
command = 'cd';
[status, cmdout] = system(command);
if (status == 1)
    disp('Error: Invalid command/terminal prompt is not working');
end % If no error, we are good to go!

% Let's start by clearing all latched errors and starting the motors
command = 'jrk2cmd --clear-errors';
[status, cmdout] = system(command);
command = 'jrk2cmd --run';
[status, cmdout] = system(command);
if (status == 1)
    disp('Error: JRK software is not properly installed');
end
fprintf("\n");

```



```

% Let's also start the clock (for DURATION) now that the motors are on
tic;

% A switch statement to determine which test to run
switch TEST

% ----- %
%                               0 - SETPOINT ONLY                               %
% ----- %

    case 0

        % VARIABLES TO SET
        setpoint = 1000; % target value, 0-4095

        % Prints test name
        disp('SETPOINT ONLY TEST');

        % Error checking
        if (setpoint < 0 || setpoint >= 4096)
            disp('Error: Setpoint must be between 0 and 4095.');
```

```

        else
            % Run command
            command = ['jrk2cmd --target' ' ' num2str(setpoint)];
            disp(['Going to setpoint: ' num2str(setpoint)]);
            [status, cmdout] = system(command);
            if (status == 1)
                disp(['Error: Invalid command: ' command]);
                disp('Consider checking if JRK software is installed properly');
            end
        end
    end

% ----- %
%                               1 - STEP INPUT                               %
% ----- %

    case 1

        % VARIABLES TO SET
        whenStepOccurs = 0.5; % a number 0 to 1, indicating at what
        % percentage of the overall time the step occurs
        initial = 2500; % value it starts at before the jump
        final = 500; % value after the jump

        % Prints test name
        disp('STEP INPUT TEST');
```

```

        % Sets initial position
        command = ['jrk2cmd --target' ' ' num2str(initial)];
        disp(['Initial setpoint: ' num2str(initial)]);
        [status, cmdout] = system(command);
        if (status == 1)
            disp(['Error: Invalid command: ' command]);
        end
    end
end

```



```

% Error checking on step time
if (whenStepOccurs < 0 || whenStepOccurs > 1)
    disp('Error: whenStepOccurs must be between 0 and 1');
else
    timeToStep = whenStepOccurs * DURATION; % calculate when to step

    % Prepare to send step command but wait until right time to send it
    command = ['jrk2cmd --target' ' ' num2str(final)];

    % Wait until it is time to step up/down
    while(toc < timeToStep)
        if PLOT_DATA == 1 % if collecting results
            output = sense();
            % Save function output in parallel arrays
            readings(j) = output(1);
            SPs(j) = output(2);
            j = j + 1;
        end
    end

    % Actually send step command
    [status, cmdout] = system(command);
    disp(['Final setpoint: ' num2str(final)])
    if (status == 1)
        disp(['Error: Invalid command: ' command]);
    end
end
end

```

```

% - - - - - %
%                               2 - IMPULSE INPUT                               %
% - - - - - %
case 2

```

```

% VARIABLES TO SET
whenImpulseOccurs = 0.5; % a number 0 to 1, indicating at what
% perecentage of the overall time the impulse occurs
initial = 2000; % value before and afetr impulse
impulse = 50; % value to jump to for impulse

% Prints test name
disp('IMPULSE INPUT TEST');

% Sets initial position
commandInit = ['jrk2cmd --target' ' ' num2str(initial)];
disp(['Initial setpoint: ' num2str(initial)]);
[status, cmdout] = system(commandInit);
if (status == 1)
    disp(['Error: Invalid command: ' commandInit]);
end
end

```



```

% Error checking on step time
if (whenImpulseOccurs < 0 || whenImpulseOccurs > 1)
    disp('Error: whenImpulseOccurs must be between 0 and 1');
else
    timeOfImpulse = whenImpulseOccurs * DURATION; % calculate when to step

    % Prepare to send impulse command but wait until right time to send it
    commandImp = ['jrk2cmd --target' ' ' num2str(impulse)];

    % Wait until it is time for impulse
    while(toc < timeOfImpulse)
        if PLOT_DATA == 1 % if collecting results
            output = sense();
            % Save function output in parallel arrays
            readings(j) = output(1);
            SPs(j) = output(2);
            j = j + 1;
        end
    end

    % Set impulse setpoint and then return to intial setpoint
    [status, cmdout] = system(commandImp);
    if PLOT_DATA == 1 % if collecting results
        output = sense();
        % Save function output in parallel arrays
        readings(j) = output(1);
        SPs(j) = output(2);
        j = j + 1;
    end
    [status, cmdout] = system(commandInit);
    disp(['Impulse setpoint: ' num2str(impulse)])
    if (status == 1)
        disp(['Error: Invalid command: ' commandImp]);
    end
end
end

```

```

% ----- %
%                               3 - SINUSOIDAL INPUT                               %
% ----- %

```

```

case 3

```

```

% VARIABLES TO SET
maxSetpoint = 4000; % largest s.p. of motor (top of sine curve)
minSetpoint = 300;  % smallest s.p. of motor (bottom of sine curve)
frequency = 0.2;    % number of periods per second
resolution = 100;   % number of setpoint commands sent per period
% TIP: Try a resolution of 2*DURATION/frequency as a starting point

% Prints test name
disp('SINUSOIDAL INPUT TEST');

% Checks input variables for errors
if (maxSetpoint > 4095 || maxSetpoint < 0)
    disp(['Error: MaxSetpoint out of range 0-4095: ' num2str(maxSetpoint)]);
end

```



```

elseif (minSetpoint > 4095 || minSetpoint < 0)
    disp(['Error: MinSetpoint out of range 0-4095: ' num2str(minSetpoint)]);
elseif (minSetpoint > maxSetpoint)
    disp('Error: MinSetpoint must be less than or equal to maxSetpoint');
else

    % Calculates scaling for the sine function
    amplitude = (maxSetpoint - minSetpoint) / 2.0;
    % Note: 2.0 prevents integer division (if it exists in MATLAB)
    vertShift = (maxSetpoint + minSetpoint) / 2.0;

    % Creates matrix of values and of commands, ready to send
    for i = 1:resolution
        spVals(i) = floor(sin(2*pi*(i-1)/resolution)*amplitude + vertShift);
        % floor rounds to nearest integer from minSP to maxSP
        command = ['jrk2cmd --target' ' ' num2str(spVals(i))];
        spCommands(i) = string(command);
    end

    % Display information about what test is being run:
    disp(['Setpoint range: ' num2str(minSetpoint) ' to ' num2str(maxSetpoint)]);
    disp(['Frequency: ' num2str(frequency) ' Hz']);
    disp(['Resolution: ' num2str(resolution) ' commands per period']);

    % Calculate how much time between each value calculated above
    period = 1.0/frequency;          % how much time per period
    i = 1;                          % which value are we on?

    tic; % resets clock since matrix generation took some time

    % Actually sends the comamnds to the JRK controller
    while (toc < DURATION) % until DURATION over

        % Send command
        [status, cmdout] = system(spCommands(i));

        % Collect data if collecting results
        if PLOT_DATA == 1
            output = sense();
            % Save function output in parallel arrays
            readings(j) = output(1);
            SPs(j) = output(2);
            j = j + 1;
        end

        % Get ready for next iteration of loop
        i = mod(floor(toc / period * resolution - 1), resolution) + 1;
        % Note: MATLAB mod of mod(-1, 100) = 99, which is ideal
    end
end

end

% - - - - -

% Points out error if user sets TEST flag value beyond allowed range
otherwise
    disp(['Error: TEST flag must be between 0 and 3, not ' num2str(TEST)]);
end

```



```

% Turn motor off after testing is concluded (and collect more results while
% waiting)
while(toc < DURATION)
    if PLOT_DATA == 1 % if collecting results
        output = sense();
        % Save function output in parallel arrays
        readings(j) = output(1);
        SPs(j) = output(2);
        j = j + 1;
    end
end
command = 'jrk2cmd --stop';
[status, cmdout] = system(command);

% Error checking in case stop command wasn't sent properly
fprintf("\n");
if (status == 1)
    disp(['Error: Invalid command: ' command]);
    disp('Warning! Motor may still be running.');
```

```

else
    disp('Motor stopped');
```

```

end

% Testing has ended. Let us go in peace.
disp(['Testing concluded ( ' num2str(toc) ' sec)']);

% ----- %
%                               DATA ACQUISITION/PLOTTING                               %
% ----- %

if PLOT_DATA == 1
    % To plot the data, we use the following code (plotting two parallel 1D
    % arrays called readings and SPs
    figure(); % new fig
    hold on; % allows multiple curves on one graph
    grid;

    % Plot the data
    t = linspace(0, DURATION, j - 1); % makes a timescale for x-axis
    plot(t, readings, 'r-');
    plot(t, SPs, 'b-');
    legend('Feedback', 'Target');
```

```

    % Set up axes
    ylabel('Setpoint or Feedback Value (0-4095)')
    xlabel('Time (s)');
    xlim([0 DURATION]);
    ylim([0 4096]);

    % Add a title depending on which test was performed
    switch TEST
        case 0
            title(['Response to Setpoint of ' num2str(setpoint)]);
        case 1
            title('Step Response');
```



```

        case 2
            title('Impulse Response');
        case 3
            title('Response to Sinusoidal Input');
        otherwise
            title('NO GRAPH FOR YOU!'); % :D
    end
end

% Helper function to collect data - returns current value of setpoint and
% potentiometer reading by scraping the output of the terminal command
% jrk2cmd -s for tokens "Target:" and "feedback:" The output returned
% (called s) is a size 2 vector containing the setpoint and potentiometer
% reading, in that order.
function s = sense()
% Send command to ask for current state and save response as textLeft
command = 'jrk2cmd -s';
[status, textLeft] = system(command);
if (status == 1)
    disp(['Error: Invalid command: ' command]);
end

% Now, look for "Target" value (setpoint)
token = "nothing yet";
while not(strcmp(token, 'Target:')) % scrape text until word "Target" found
    [token, textLeft] = strtok(textLeft);
end
% Note: I could have added error checking to ensure that reaching the end
% of the textLeft would have stopped the (otherwise infinite) loop -
% however I opted out of doing this since these extra checks would have
% decreased the runtime performance of this code. If you would like to add
% this check yourself, just throw an error if textLeft is empty

% Now that we found the word "Target" we can save its corresponding value
[token, textLeft] = strtok(textLeft);
s(2) = str2num(token);

% Now, look for word "feedback:" to find the scaled feedback value
while not(strcmp(token, 'feedback:')) % scrape text until found
    [token, textLeft] = strtok(textLeft);
end
% Now that we found the word "feedback" we can save its corresponding value
[token, textLeft] = strtok(textLeft);
s(1) = str2num(token);

end

```

STEP INPUT TEST

Initial setpoint: 2500

Final setpoint: 500

Motor stopped

Testing concluded (10.0087 sec)