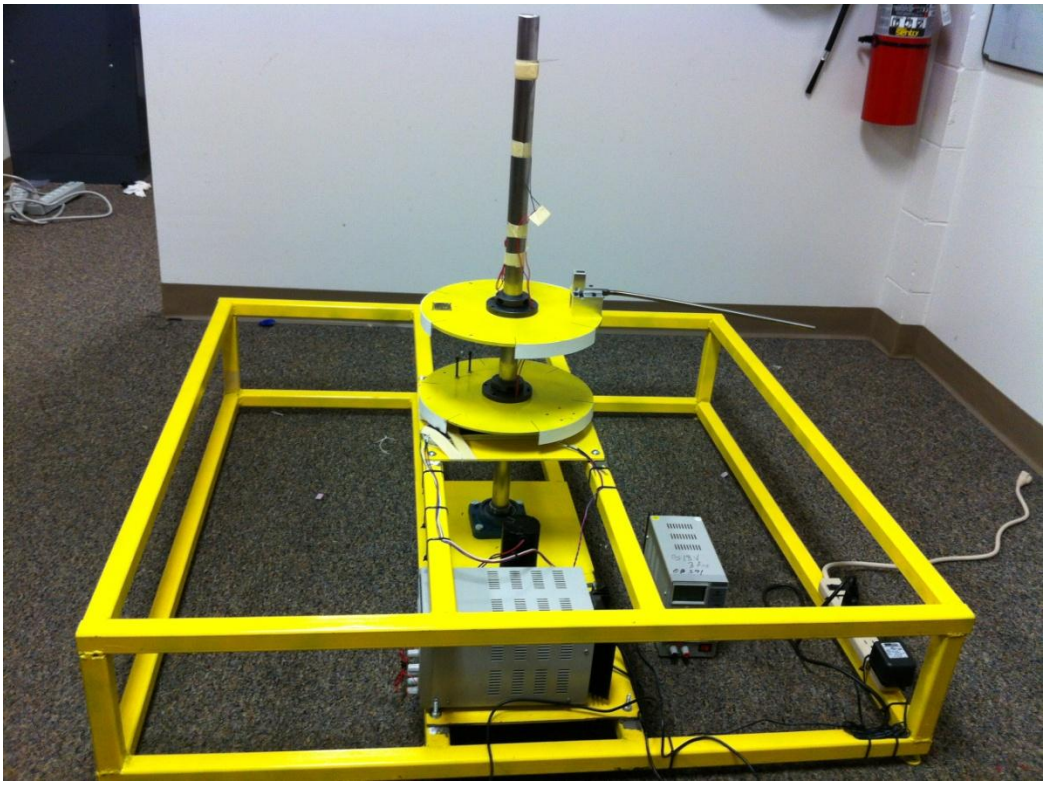

Spin-Test Stand Manual

Michael Arsenault

Dr. Aroh Barjatya

09/18/2012

Contents

Overview

IMPORTANT: Read this entire manual before operating or modifying the spin-test stand. Failure to do so could result in damage to the test stand, or potential injury from spinning/liberated components.

This manual is written to explain the assembly, disassembly and operation of the spin-test stand shown on the cover of this manual. The manual also contains possible modifications that could be made to the test stand, circuit diagrams, and the code on the microprocessor of the circuit board and in the operating .m (MATLAB) file.

The spin-test stand is designed for running spin tests on small instruments and components that will undergo high rates of spin in their operational environment (up to approximately 7Hz, or 420RPM). The test stand includes a brush system on the rotating disks to allow power to be supplied to the instrument or component during testing.

Figures 1 through 5 show the completed test stand and its various components.

Figure 1 and Figure 2 show the completely assembled test stand with both testing platforms (disks) in place on the shaft.

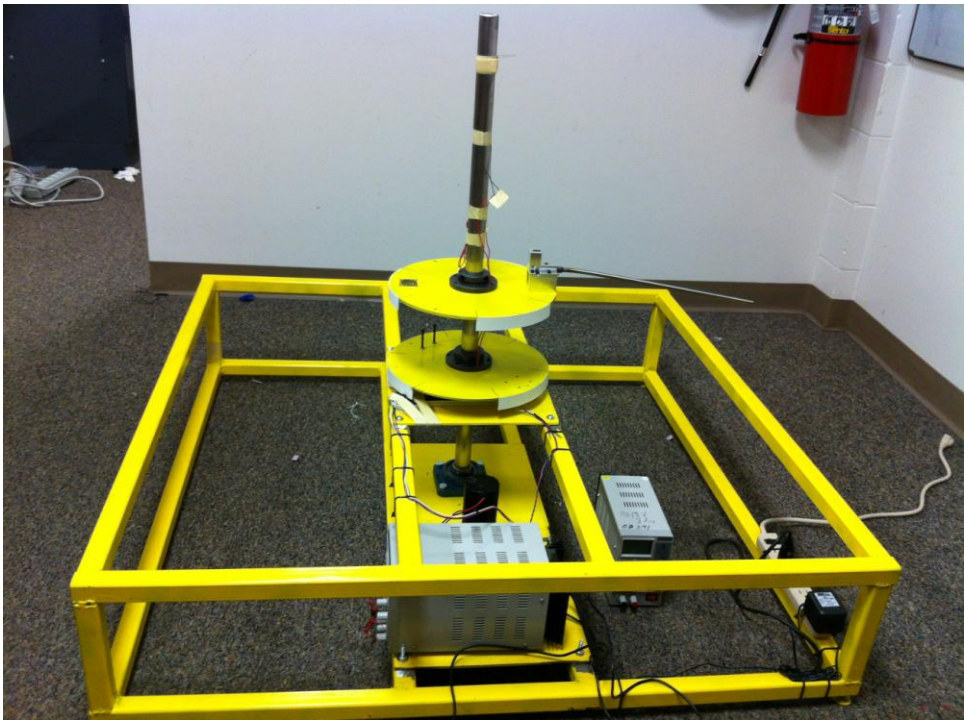

Figure 1 – Completely assembled test stand

Figure 2 – Close-up of shaft and testing platforms on assembled test stand

Figure 3 shows the circuit board and motor controller used to operate the test stand. Notice that the motor controller is run off of a separate power supply and only receives inputs from the microcontroller on the circuit board.

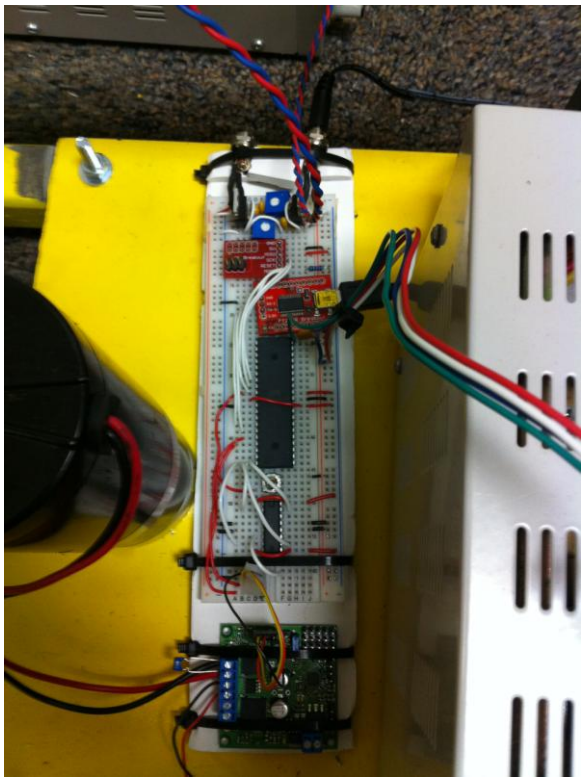

Figure 3 – Circuit board and motor controller of test stand

Figure 4 shows a close-up of the optical sensor used to measure the RPM of the test stand. The optical sensor works via an infrared LED and an infrared sensor. If the gate is open (nothing blocking the path of the IR light), the sensor puts out a high voltage (approx. 5V); if the gate is closed (the path of the IR light is blocked), the sensor puts out a low voltage (less than 2V). The path of the IR light is blocked via cardboard strips glued to the circumference of the two rotating disks. Each strip is 1/6 of the circumference of the disk. As the disks spin, the IR path is broken by the cardboard strips and the RPM of the disk is calculated based on the frequency that the gate opens and closes.

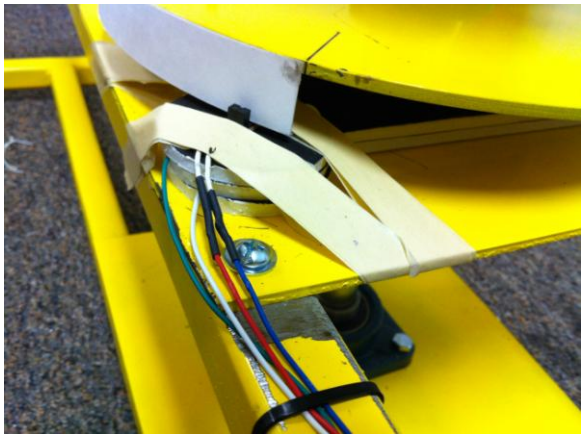

Figure 4 – Four-wire optical sensor setup

Figure 5 shows the powerbar (attached to the frame via zip-ties) that is used to power both the circuit board, as well as the separate power supply connected to the motor controller. By leaving the motor controller's power supply in the *ON* position, the entire test stand can be powered on and off via the switch on the powerbar.

Figure 5 – Powerbar setup

Disassembly

1) Remove the top testing platform (disk)

   The top and bottom disks are both removed from the shaft by loosening the collar from the shaft, shown in Figure 6, and then sliding the disk off of the shaft.

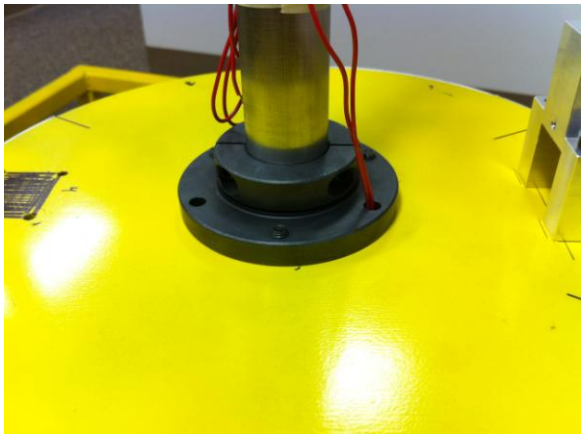

Figure 6 – Testing platform (disk)

   a. Loosen the two bolts in the collar that tighten it to the shaft, as shown in Figure 7.

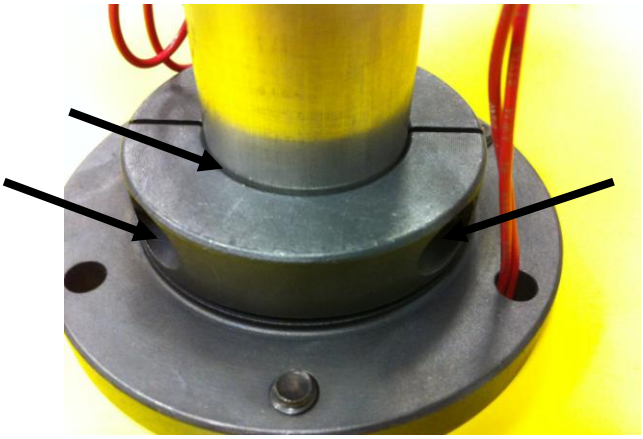

Figure 7 – Close-up of collar on disk

b. Carefully and slowly slide the disk up and off of the shaft. **Be careful not to grab the white paper strips when pulling the disk off of the shaft.**

Figure 8 – Shaft with top testing platform (disk) removed

c. Place the disk on a flat surface, resting upside down on the collar and not on the white paper strips. Figure 9 shows the proper way to set the disk down.

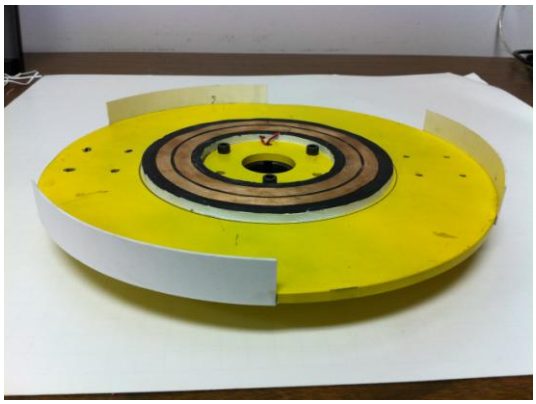

Figure 9 – Testing platform set down on collar

IMPORTANT: Before attempting to slide either disk off of the shaft, ensure that the shaft is completely smooth and clean. Any nicks, bumps or debris on the shaft must first be removed via sanding by hand with sand paper and then wiping the shaft with a rag and lab-grade alcohol.

It is equally important to ensure that there is no debris in the joint between the collar and the shaft prior to removing the disk, as shown in Figure 7. Even a small piece of metal or sand can become wedged between the two and permanently stop the disk from sliding.

DO NOT HIT THE DISKS OR THE SHAFT WITH A HAMMER. Hitting the shaft with a hammer will cause it to mushroom or deform and render the disks permanently stuck. Hitting the disks could cause severe damage to both the shaft and collar and throw the disk out of balance.

2) Remove the bottom testing platform (disk)

The bottom disk can be removed identically to the top disk, as described in Step #1.

Figure 10 – Shaft with both testing platforms (disks) removed

3) Remove electrical brushes assembly

The brushes assembly, shown in Figure 11, is held in place by a pressure fit and is not glued or bolted in place.

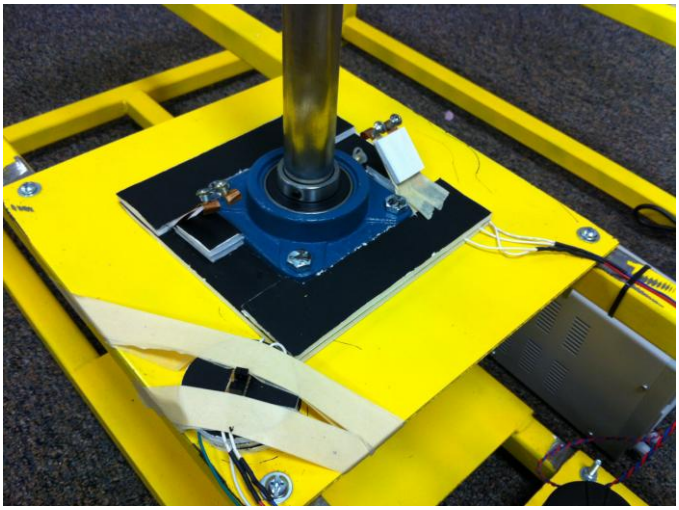

Figure 11 – Brush system

a. Carefully lift one side of the brushes assembly and remove it from the top flange bearing, as shown in Figure 12.

   Note: The wires leading to the brushes assembly may be zip-tied to the frame. If this is the case, carefully cut and discard the zip ties to free the wires.

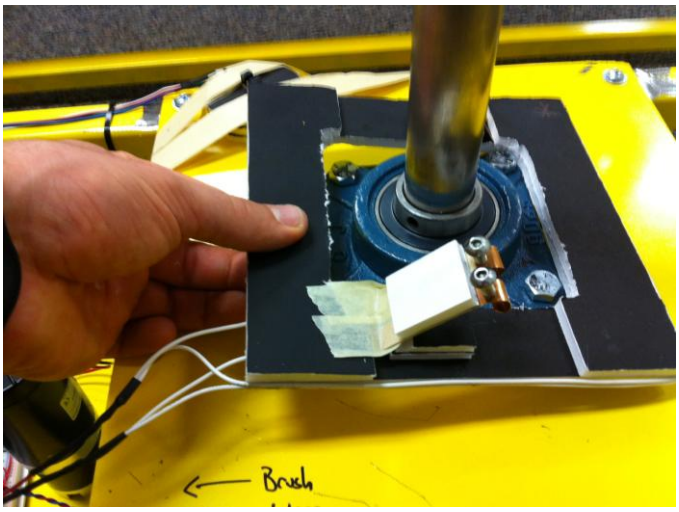

Figure 12 – Removal of brushes from test stand

4) Remove circuit board and power supply

   a. Unscrew the connectors on the motor controller, shown in Figure 13, and remove the wires leading to the electric motor and to the power supply. Ensure that the 0.1uF capacitor across the motor wires is not lost.

   b. Unplug the USB connection and remove the four wires attached to the optical sensor (not shown).

   c. Unplug the power supply to the circuit board (not shown). At this point the circuit board and motor controller are separate from the stand and can be removed and set aside.

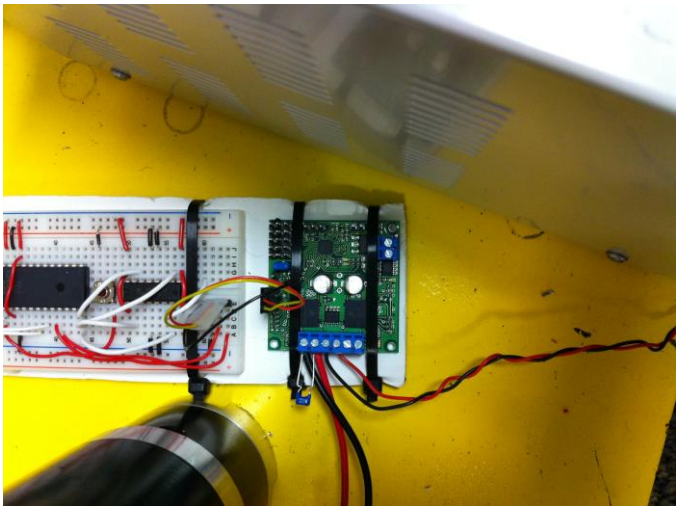

Figure 13 – Electrical connections to motor controller

d. Unplug and remove the motor's power supply (shown in Figure 14).

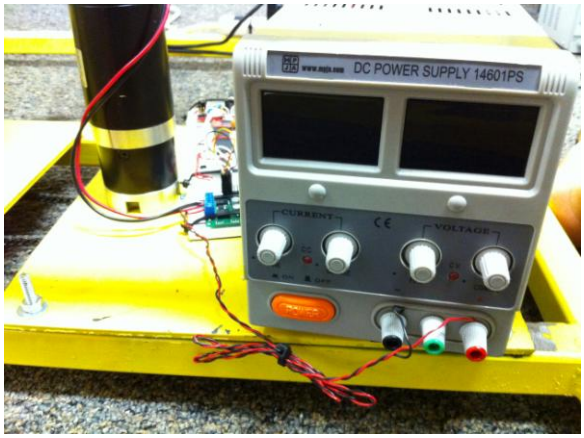

Figure 14 – Power supply for motor / motor controller

5) Remove top flange bearing

The top flange bearing, shown in Figure 15, is attached to the frame via four bolts, as shown in Figure 16. The collar of the flange bearing is tightened to the shaft via two locking screws, as can be seen in Figure 17.

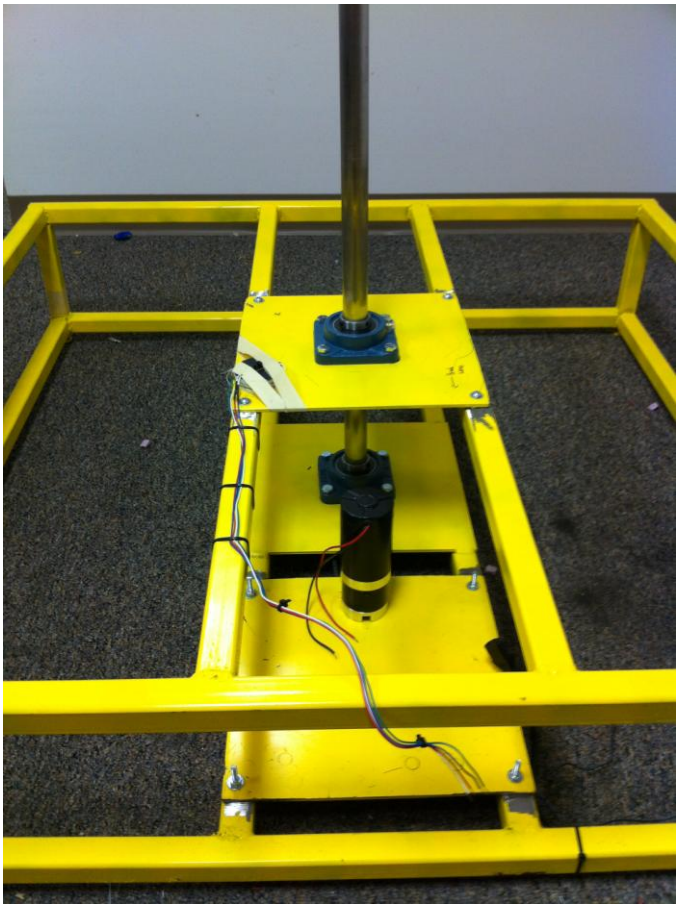

Figure 15 – Test stand with shaft, optical sensor and motor

Figure 16 – Bolts for top flange bearing

a. Remove the four bolts holding the flange bearing onto the frame, shown in Figure 16.

   Take care to notice the markings on the bearing and the test stand used to properly align the bearing on the plate. These markings will be particularly useful when assembling the test stand.

b. Loosen (but to not remove) the two lock screws in the collar of the flange bearing, as can be seen in Figure 17.

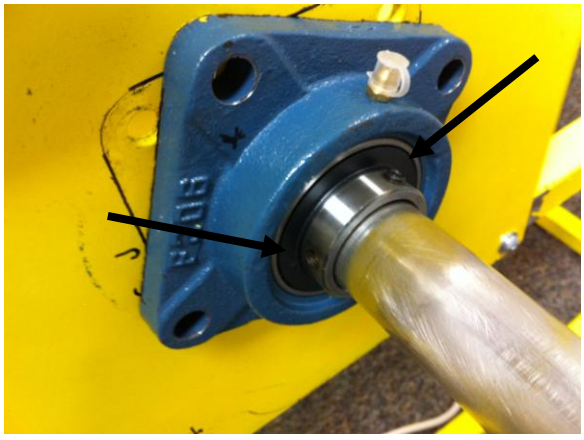

Figure 17 – Top flange bearing on shaft

c. Carefully slide the bearing up and off of the shaft in the same fashion as the two testing platforms (disks).

IMPORTANT: The shaft must be completely smooth and the joint between the collar and the shaft must be free of all debris prior to removing the bearing. Refer to Step #1 of the disassembly.

6) Disassemble chain and sprocket assembly

The chain and sprocket assembly, shown in Figure 18, is located on the bottom of the test stand and can be accessed by leaning the test stand on its side.

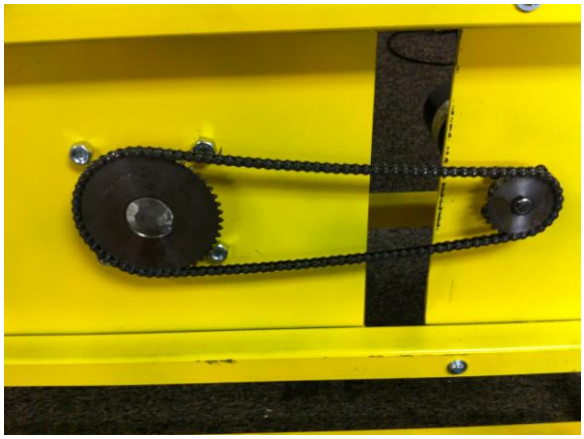

Figure 18 – Chain and sprocket system from motor to sharf

a. Remove the retaining ring from the motor sprocket, shown in Figure 19.

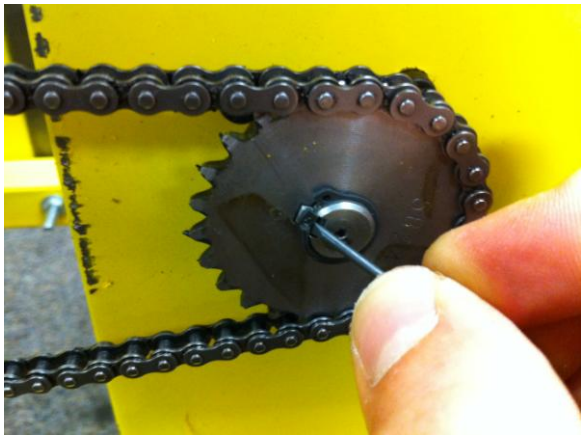

Figure 19 – Removal of retaining ring from motor-side sprocket

b. Pull the motor sprocket off of the motor shaft, shown in Figure 20.

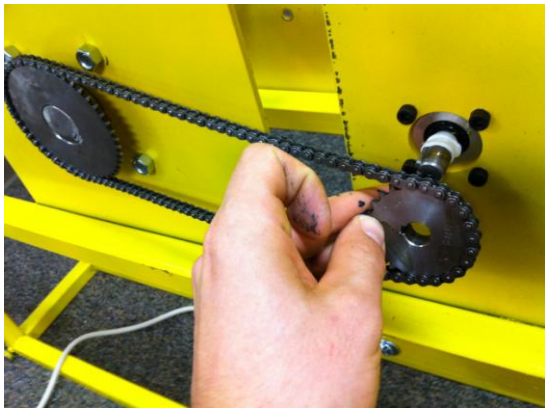

Figure 20 – Removal of motor-side sprocket

c. Remove the chain and motor sprocket and set them aside.

Note: The motor shaft of the original design is wrapped with Teflon tape to hold the sprocket tightly against the retaining ring and to prevent the sprocket from moving on the shaft. Refer to Figure 20.

7) Remove shaft from stand/bottom flange bearing

The bottom flange bearing does not need to be removed from the stand, and in fact **should not be**. This is to ensure proper alignment of the shaft through the holes in the plates in the stand and to ensure proper tightness of the chain between the two sprockets.

a. Loosen (but do not remove) the two lock screws in the collar of the flange bearing, shown in Figure 21.

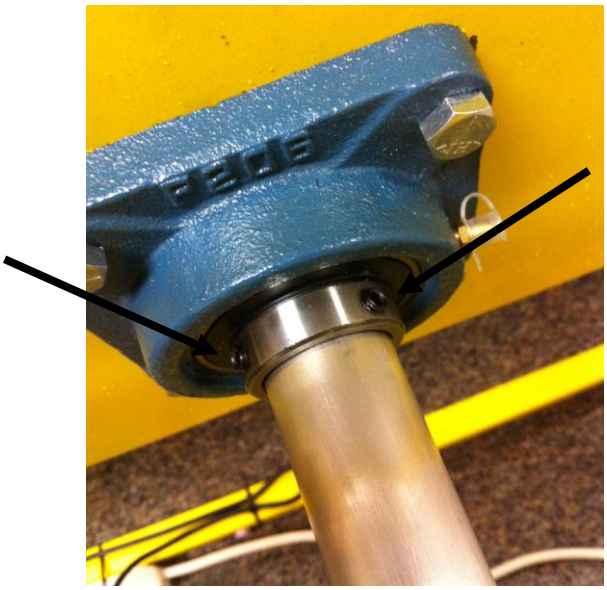

Figure 21 – Bottom flange bearing

b. Carefully slide the shaft out of the bottom of the test stand, as shown in Figure 22.

Figure 22 – Removal of shaft and sprocket

c. Gently and carefully set the shaft and sprocket aside.

Figure 23 – Shaft and sprocket

IMPORTANT: Take care to ensure that the shaft is not damaged while removing it from the test stand, as well as when setting it aside.

The shaft must be completely smooth and the joint between the collar and the shaft must be free of all debris prior to removing the shaft. Refer to Step #1 of the disassembly.

8) Separate shaft and sprocket

The sprocket on the shaft is held in place by a pressure fit, two lock screws and a key. The key prevents the sprocket from spinning on the shaft while the pressure fit and lock screws hold it in place on the shaft.

Note: The sprocket-end of the shaft has been mushroomed using a hammer, and so the sprocket must be removed by sliding it the entire length of the shaft and pulling it off the same end of the shaft that the two disks and the top flange bearing were removed from.

a. Loosen (but do not remove) the two lock screws in the sprocket collar, shown in Figure 24.

Figure 24 – Shaft sprocket

b. Orient the shaft vertically and gently tap near the center circumference of the sprocket with a hammer until the sprocket slides freely on the shaft, as shown in Figure 25.

Figure 25 – Removal of sprocket from shaft

c. Slide the sprocket off of the shaft and carefully set both aside. Be careful not to lose the key, shown in Figure 25.

IMPORTANT: The shaft must be completely smooth and the joint between the collar and the shaft must be free of all debris prior to removing the sprocket. Refer to Step #1 of the disassembly.

Assembly

1) Join shaft and sprocket

   a. Place the key in the keyway of the shaft and slide the sprocket up the shaft, starting at the opposite end of the shaft, as shown in Figure 26.

Figure 26 – Shaft sprocket with key in keyway

   b. Continue to slide the sprocket into place, ensuring that the key is not shifted up past the end of the shaft.

   c. To finish the placement of the sprocket, suspend the shaft by the sprocket, as shown in Figure 27, and gently tap the center of the shaft with a hammer until it is flush with the sprocket surface.

Figure 27 – Shaft with sprocket in place

d. Tighten the two lock screws in the collar of the sprocket to hold it in place, as shown in Figure 28.

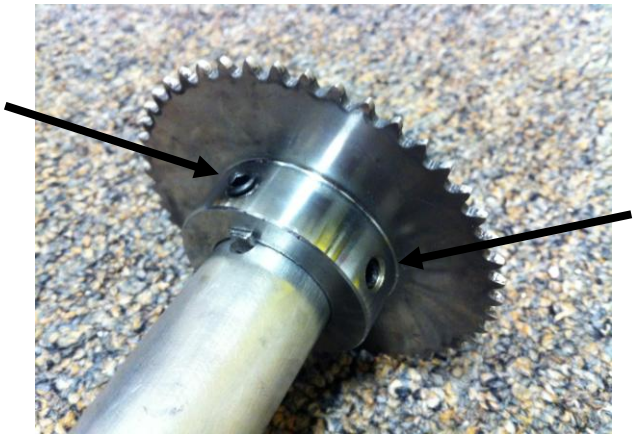

Figure 28 – Locking screw in shaft sprocket

IMPORTANT: The shaft must be completely smooth and the joint between the collar and the shaft must be free of all debris prior to installing the sprocket. Refer to Step #1 of the disassembly.

2) Insert shaft into test stand/bottom flange bearing

It is important that the sprocket of the shaft line-up with the sprocket on the motor to ensure the chain remains in place during operation. To aid in the placement of the shaft, the motor sprocket must first be installed.

a. If the retaining ring is in place on the motor shaft, as shown in Figure 29, it must first be removed to allow the motor sprocket to be installed.

Figure 29 – Electric motor shaft with key and retaining ring

b. Slide the sprocket on the shaft of the motor and install the retaining ring to hold it in place, as shown in Figure 30.

Figure 30 – Installed motor sprocket

c. Insert the large shaft into the bottom flange bearing and carefully slide it in until the two sprockets are flush with one another, as shown in Figure 31. A ruler or other straight edge can help to make sure the two sprockets are flush.

Note: Ensure that the two lock screws in the collar of the bottom flange bearing are sufficiently backed-off as to not interfere with inserting the shaft.

Figure 31 – Shaft installed with both sprockets aligned

d. Tighten the two lock screws in the collar of the bottom flange bearing, shown in Figure 32, to hold the shaft in place.

Figure 32 – Locking screws on bottom flange bearing

IMPORTANT: The shaft must be completely smooth and the joint between the collar and the shaft must be free of all debris prior to installing the shaft. Refer to Step #1 of the disassembly.

3) Install chain and sprocket assembly
If the motor sprocket is still installed on the motor shaft, it first needs to be removed before the chain can be installed. Remove the motor sprocket by taking off the retaining ring (refer to Step #2) and then pulling the sprocket off.

a. Loop the chain around both sprockets and pull tight using the motor sprocket, as shown in Figure 33.

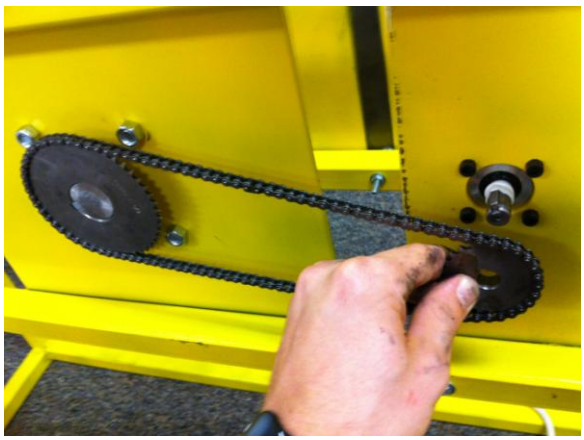

Figure 33 – Installation of chain and motor sprocket

b. Slip the motor sprocket onto the motor shaft and install the retaining ring, as shown in Figure 34. Make sure that the key is in place on the motor shaft and not protruding past the face of the sprocket, or the retaining ring will not fit properly.

Note: The motor shaft of the original design is wrapped with Teflon tape to hold the sprocket tightly against the retaining ring and to prevent the sprocket from moving on the shaft. Refer to Figure 29.

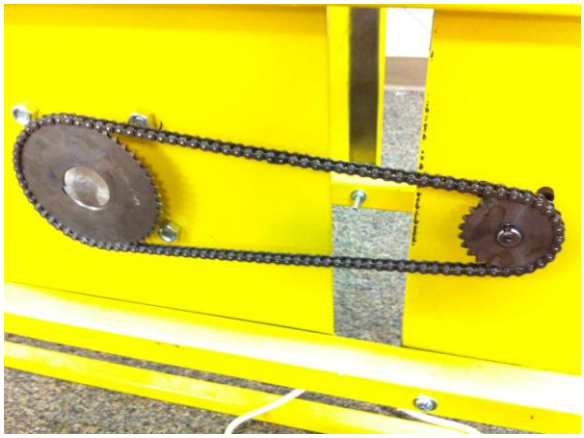

Figure 34 – Complete chain and sprocket assembly

4) Install top flange bearing

The remaining steps of the test stand assembly are more easily carried out if the stand is tipped back over so it is standing upright on the ground.

a. Ensure the lock screws in the collar of the flange bearing are adequately backed-off so as to not impede the shaft in any way.

b. Carefully slide the bearing down the shaft until it is flush with the top plate of the test stand.

IMPORTANT: Ensure that the top flange bearing is oriented so that the markings on the bearing (Figure 35) line up with the markings on the top plate (Figure 36). Failing to align these markings may cause the shaft to be off-center and off-balance, as well as to wear the bearings out more quickly.

Figure 35 – Prepared top flange bearing

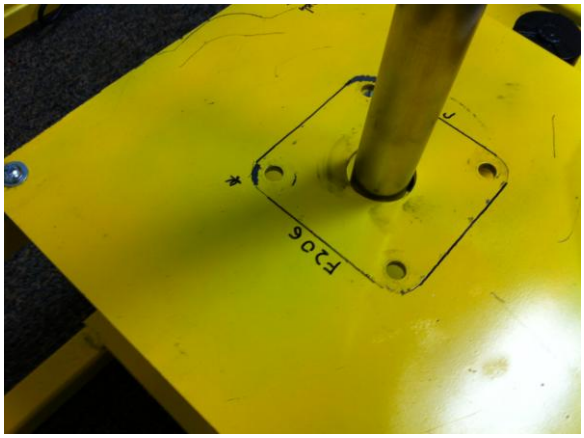

Figure 36 – Alignment markings for top flange bearing

c. Bolt the flange bearing in place and tighten the two lock screws on the collar, as shown in Figure 37.

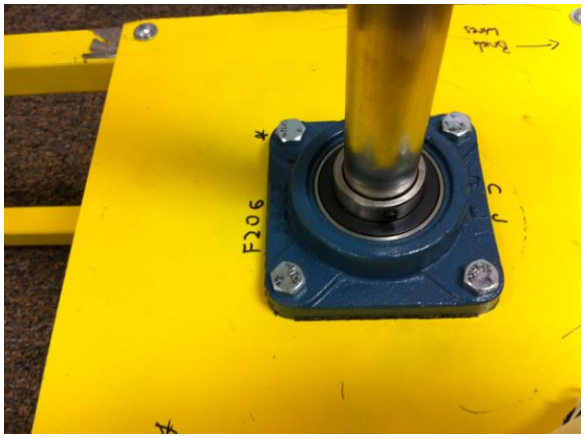

Figure 37 – Top flange bearing bolted in place

IMPORTANT: The shaft must be completely smooth and the joint between the collar and the shaft must be free of all debris prior to installing the top flange bearing. Refer to Step #1 of the disassembly.

5) Install brushes assembly

   a. Orient the brushes assembly so that the wires coming out of it are aligned with the markings on the top plate ("Brush Wires").

   b. Carefully slip the brushes assembly onto the flange bearing so that the assembly rests flush on the top plate of the test stand, as shown in Figure 38.

      Note: The brushes assembly is held in place only by pressure. It is not advised to permanently attach it to the stand.

   c. Secure the loose wires to the test stand frame using zip ties, as shown in Figure 38.

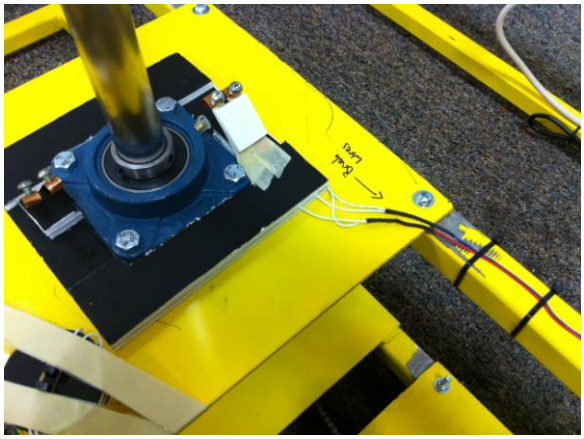

Figure 38 – Brush system in place around top flange bearing

6) Install power supply and circuit board

The motor that spins the shaft is operated via a Pololu TReX Jr motor controller, which is controlled via the microcontroller on the circuit board. The motor runs at a voltage in the range of 24V-30V, and at a load-dependent current that can spike at multiple amps. For this reason, the motor (and motor controller) are run on their own power supply, separate from that of the circuit board.

   a. Install the power supply for the motor and motor controller in the empty space next to the motor, as shown in Figure 39.

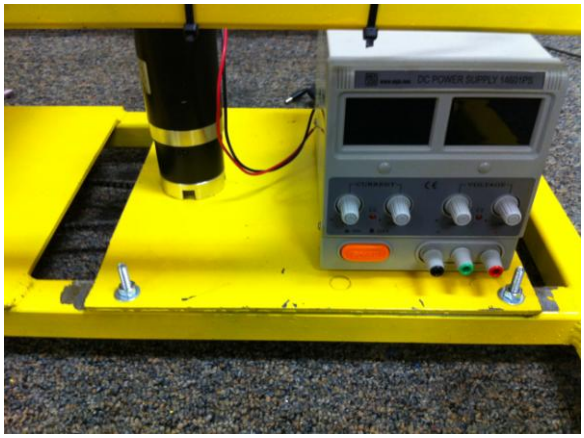

Figure 39 – Power supply for motor / motor controller

   b. Place the circuit board (and attached motor controller) in the space between the motor and the separate power supply.

c. Connect the motor controller to both the motor and the power supply, as shown in Figure 40 and Figure 41.

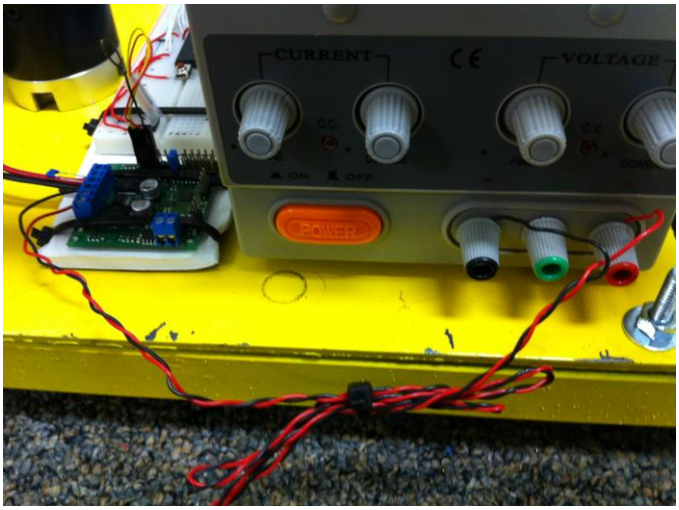

Figure 40 – Electrical connection from power supply to motor controller

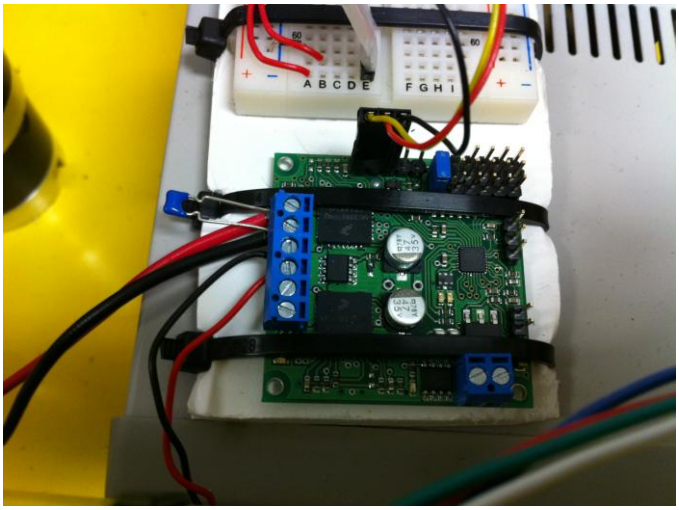

Figure 41 – Electrical connection for motor controller

Note that the motor and power supply must be connected in the same order as is shown in Figure 40 and Figure 41. If facing the six blue connection-points on the motor controller, they are filled as follows (from left to right):

1) Motor positive wire (thick red)
2) Motor ground (thick black)
3) Power supply ground (thin black)
4) Power supply positive (thin red)
5) EMPTY
6) EMPTY

Also note that the two motor wires are connected via a small capacitor (0.1uF). The capacitor is not essential to the operation of the test stand, but helps to smooth the spinning and noise of the motor.

Note: More detailed wiring diagrams can be found in the “Circuit Diagrams” section of this manual.

d. Connect the optical sensor to the circuit board.

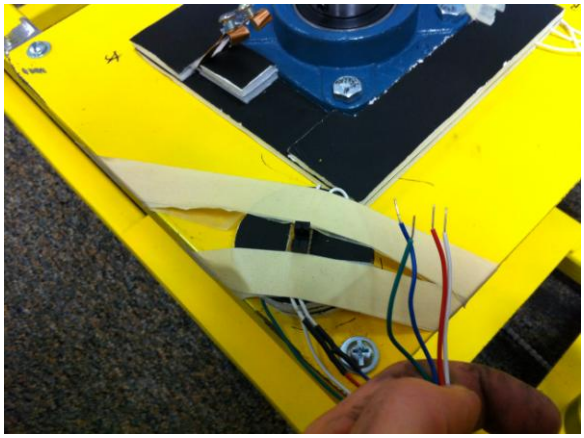

Figure 42 – Connecting wires for optical sensor

Four wires come out of the optical sensor and are soldered to wires colored blue, green, red and white, as shown in Figure 42. The four wires connect to the circuit board, as shown in Figure 43, in the following configuration:

- Blue wire to ground (inner rail)
- Red and white wires to positive (+5V) rail
- Green wire to pin A0 of the microcontroller
- Pin A0 of the microcontroller grounded via a 20pF capacitor

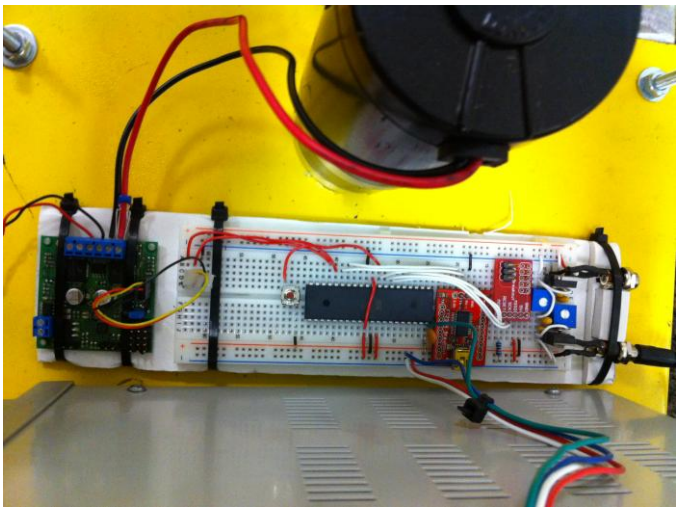

Figure 43 – Circuit board with optical sensor connections

Note that pin D6 MUST be grounded via a 20pF capacitor. The capacitor simulates the input capacitance of the oscilloscope used to calibrate the sensor. Without the capacitor, the microcontroller will read incorrect values from the sensor.

e. Connect the circuit board to its power supply and to the operating computer via a USB cable.

7) Install bottom testing platform (disk)

a. Ensure the lock screws in the collar of the disk are sufficiently backed-off so that the collar will easily slide over the shaft.

b. With the paper strips on the disk facing down, slowly and carefully slip the disk down the shaft until it is close to its final position, as shown in Figure 44.

Figure 44 – Test stand with shaft and bottom testing platform (disk)

c. Carefully position the disk on the shaft so that the paper strips are approximately 1/8" to 1/4" from the bottom of the optical sensor, as shown in Figure 45.

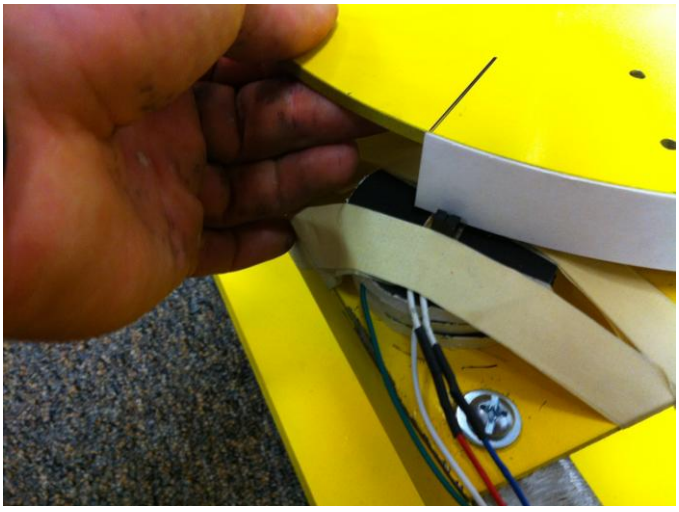

Figure 45 – Clearance for optical sensor

d. While still holding the disk (so that it does not fall down), tighten the two locking screws on the collar, shown in Figure 46, to secure the disk to the collar. Alternate between the two screws until both are tight and the disk is secured in place.

Note: Once the screws in the collar have been tightened, slowly rotate the disk to ensure that the paper strips do not impact the optical sensor at any point.

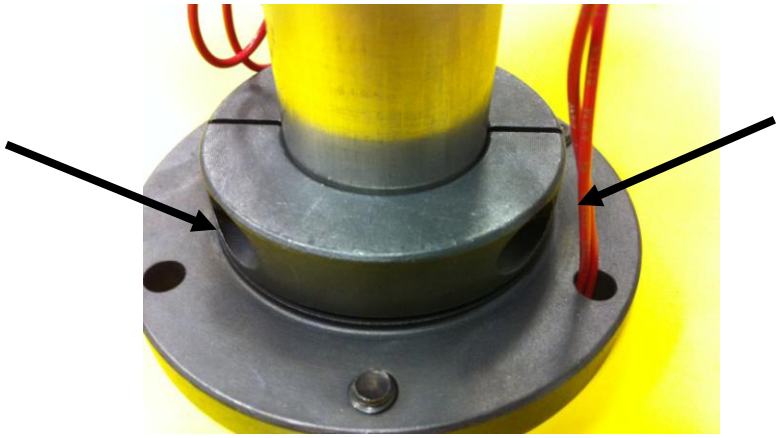

Figure 46 – Close-up of collar on disk

IMPORTANT: The shaft must be completely smooth and the joint between the collar and the shaft must be free of all debris prior to installing the disk. Refer to Step #1 of the disassembly.

8) Install top testing platform (disk)

The top disk can be fastened to the shaft in either orientation (paper strips up or down). The installation of the top disk is identical to that of the bottom disk, as described in Step #7.

IMPORTANT: The shaft must be completely smooth and the joint between the collar and the shaft must be free of all debris prior to installing the disk. Refer to Step #1 of the disassembly.

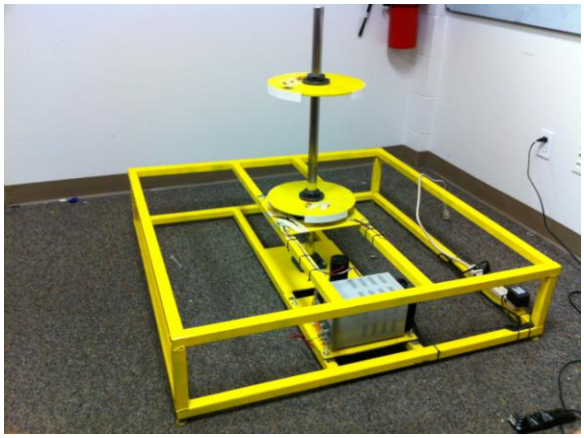

Figure 47 – Completed test stand assembly

9) Connect brushes to power supply (if required)

The two wires leading to the brushes assembly, shown in Figure 48, should be attached to a power supply of their own and not share a power supply with the circuit board. This will help prevent any malfunction of the circuitry if/when a high load is drawn through the brushes. The power supply for the brushes can be controlled manually or via the micro controller and user interface, as described in the "Operation" section of the manual.

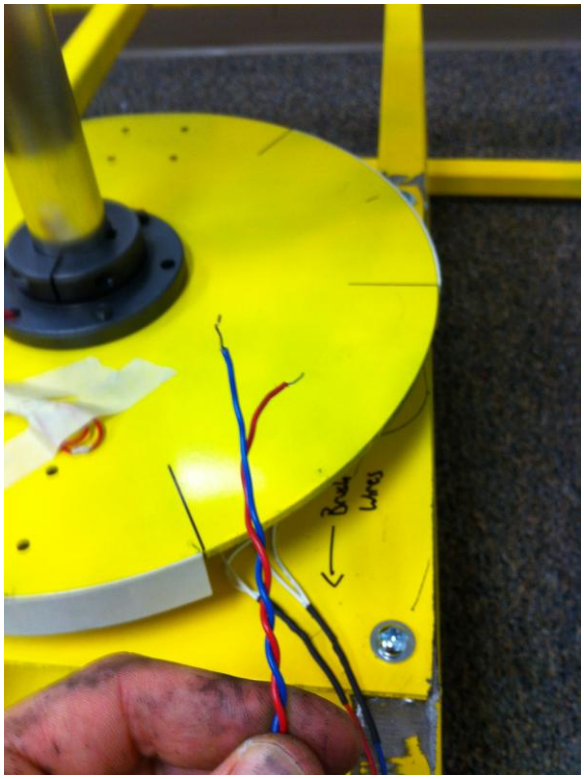

Figure 48 – Connection to brushed

Operation

The test stand is controlled via a user interface coded in MATLAB, as shown in Figure 49.

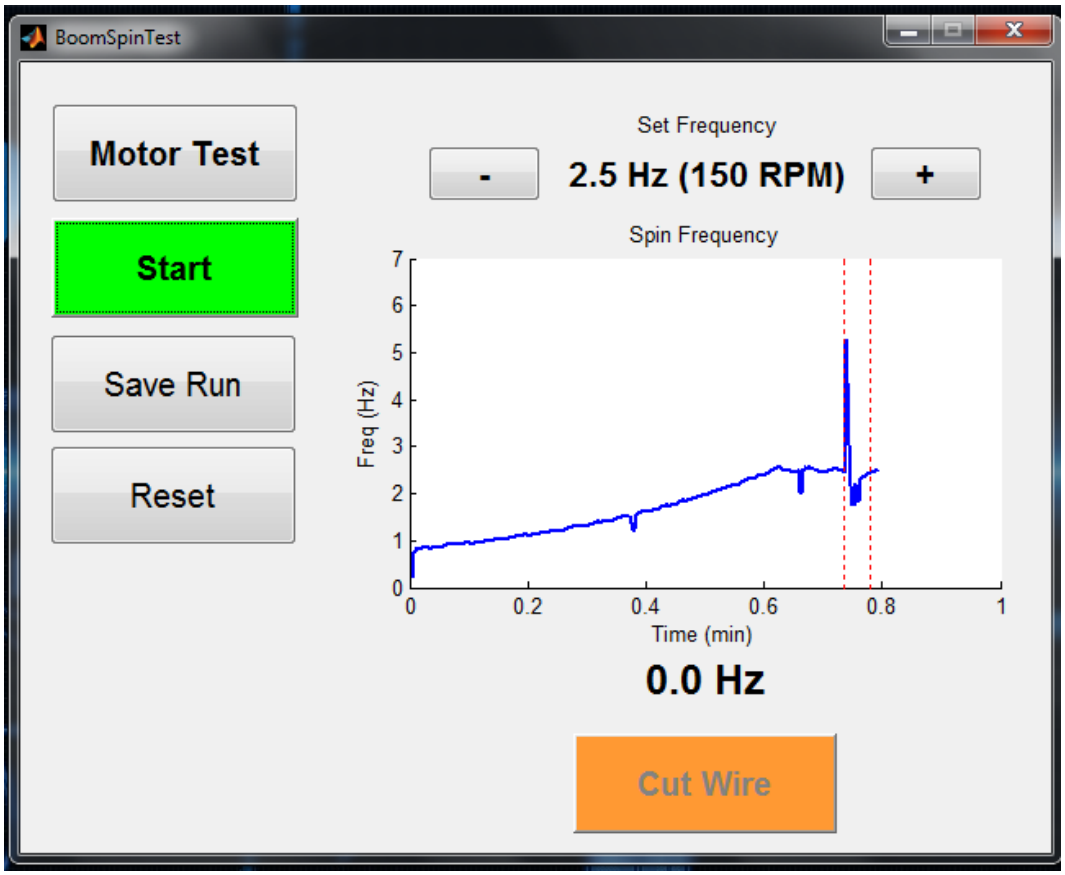

Figure 49 – MATLAB user interface for operating the test stand

Important: Before running the control program for the test stand, it is important to ensure the following:

1) **The test stand is fully (and properly) assembled according to the assembly instructions in this manual**
2) **The circuit board is connected to the control computer via a USB cable**
3) **The test stand is powered OFF (which is most easily accomplished via the switch on the power bar)**

The operating program must be run by first running MATLAB, opening BoomSpinTest.m and then running the program from the MATLAB window. Running the program via the .fig file will not initialize the program properly.

Note: Before running BoomSpinTest.m, it is important to ensure that the correct PORT on your computer is referenced in the code. The original code references "COM3" (found in the BoomSpinTest_OpeningFcn function). The COM number may need to be changed, depending on the operating computer.

When the program is running, a window similar to that in Figure 49 should display on the screen. Once the program is properly running without error, it is safe to power on the test stand (both the circuit board and the power supply for the motor controller). The most common runtime error when starting the program will be a PORT/COM error. Ensure that the computer is properly connected to the circuit board with a USB cable and that the correct PORT is referenced in the code.

The power supply for the motor controller should be set within the range of 24V-30V and no higher; multiple amps of current may be drawn during operation, and so the allowed current may need to be adjusted/increased after initial tests.

Once the program is open and the test stand is completely powered *ON*, use the "Motor Test" feature of the program to make sure the test stand and program are properly working.

To test the motor:

1) Click and hold the "Motor Test" button located in the upper left-hand side of the user interface

   If the test stand is set up correctly and the, operating program is connected to the microcontroller, the test stand should begin to slowly spin **clockwise** at a constant speed

2) Release the "Motor Test" button and the test stand should slow to a stop

   Note: If the test stand spins counter clockwise instead of clockwise, the wires connecting the power supply to the motor controller are likely switched with one another OR the wires leading to the motor from the motor controller are switched with one another. Refer to Step #6 in the Assembly section of this manual.

To run a spin test:

1) Set the desired rotation speed (in RPM) using the plus (+) and minus (-) buttons at the top of the user interface

2) Click the "Start" button and allow the test stand to accelerate to the preselected speed. The speed of rotation of the test stand versus time is plotted in the plot; the real-time speed can be seen displayed below the plot.

3) Perform the necessary actions while spinning.

4) Press the "Stop" button and allow the test stand to slow to a stop.

5) If needed, press "Save" to save the run (a text document of the points for the plot, saved as a unique number in the same folder that the BoomSpinTest.m file is located).

6) Press "Reset" to clear the previous run and reset the user interface.

7) Repeat steps 1-6 as desired.

   Note: While the test stand is running (after pressing the "Start" button), the "Cut Wire" button will become active and allow the user the option of performing a computer-controlled action. Pressing the "Cut Wire" button while the test is being performed will turn pin D6 of the microcontroller on for three seconds and then turn it off again. This three-second period is meant for a mechanism to be activated to deploy any moving components that are being tested. It is suggested that pin D6 only supply the trigger for the event, and the power for the event come from a power supply separate from that of the circuit board. Before using the "Cut Wire" button, examine the entire BoomSpinTest.m code to understand how it works.

Circuit Diagrams

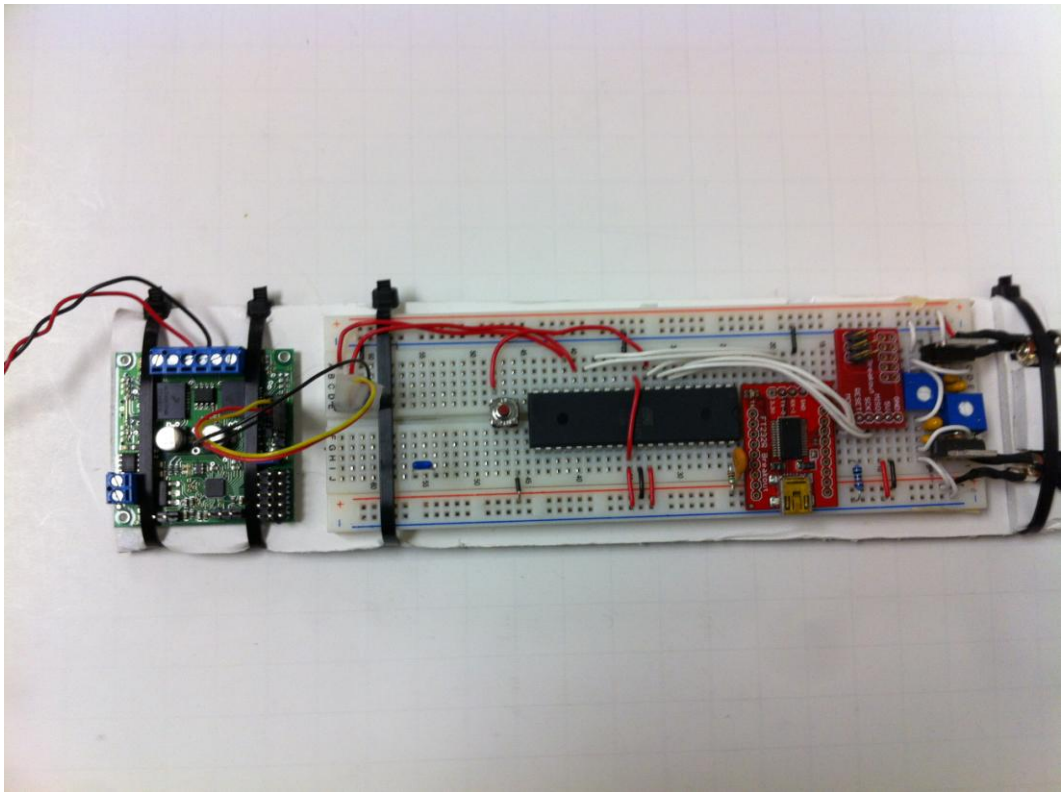

Figure 50 – Circuit board and motor controller unit

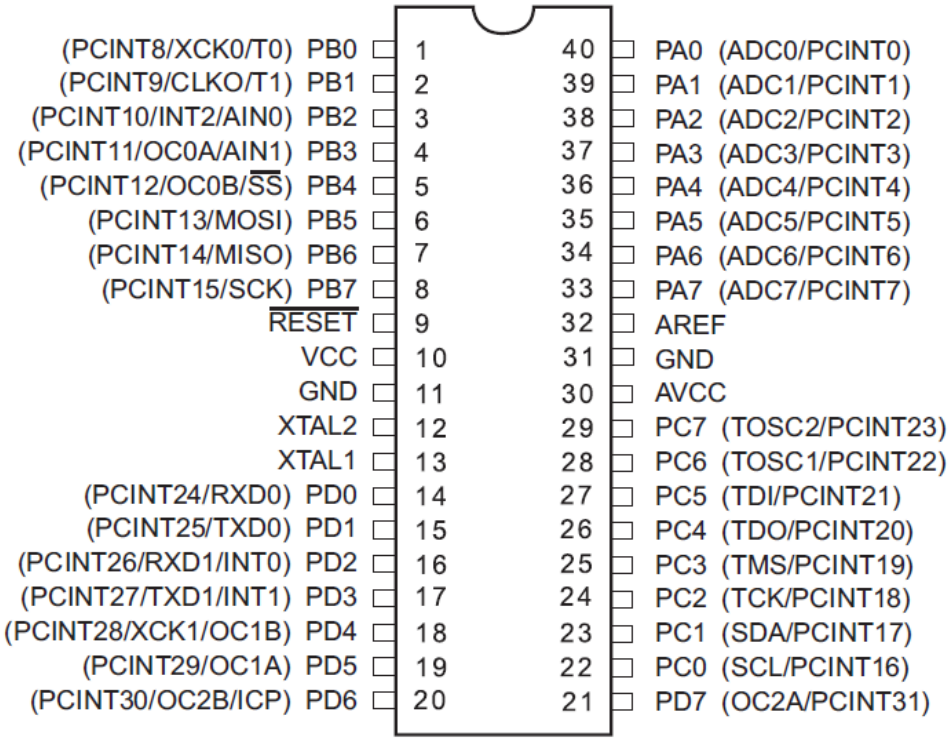

Figure 51 – ATmega 644PA microcontroller pin map

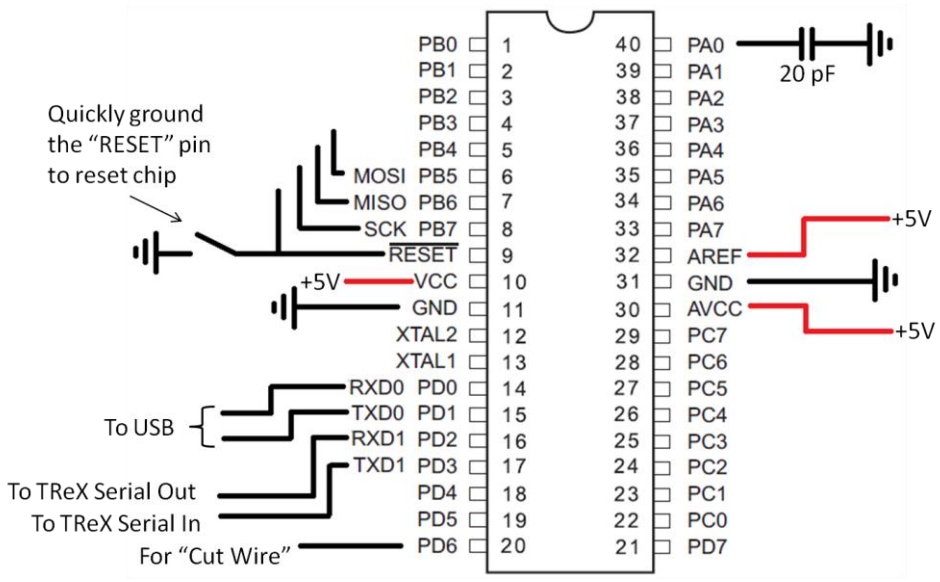

Figure 52 – ATmega 644PA microcontroller pin connections

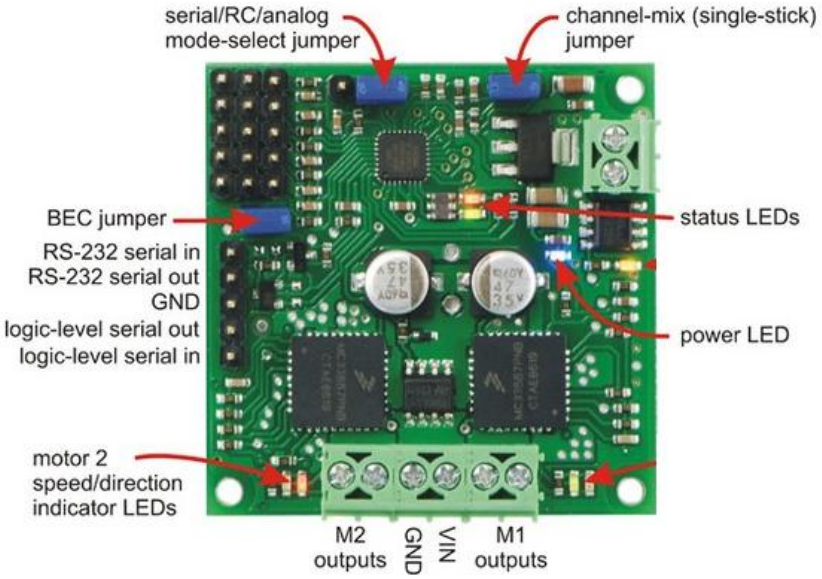

Figure 53 – Pololu TReX Jr motor controller

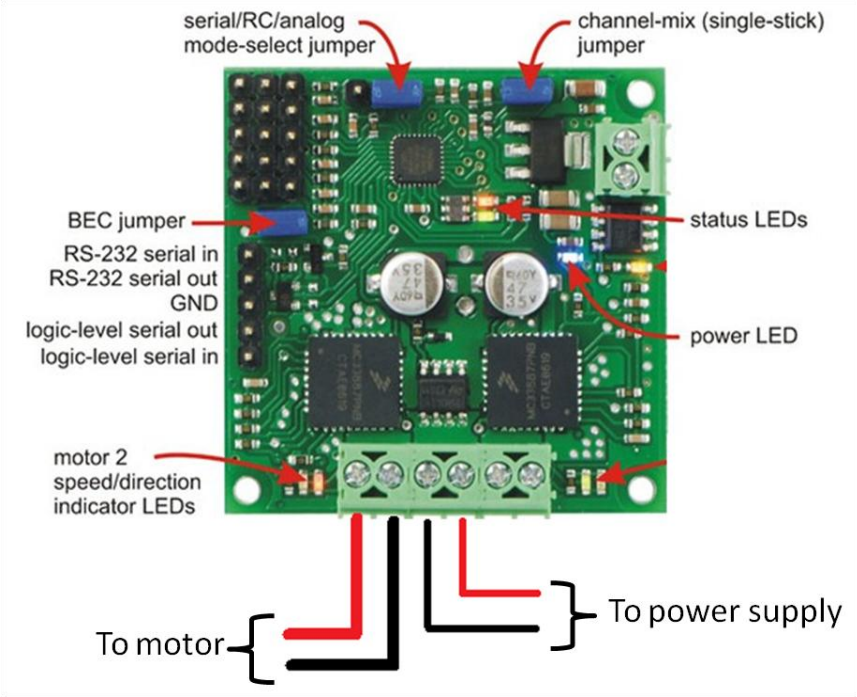

Figure 54 – Pololu TReX Jr motor controller connections

Making Modifications

IMPORTANT: Before making any modifications to the test stand, hardware or software, read this entire manual, cover to cover, including the code section.

Possible modifications might include:

- Developing a more stable control algorithm for maintaining a constant speed when the test stand is running
- Creating a purpose-built circuit board for the test stand, instead of using a breadboard
- Modifying the code on the microprocessor / in the MATLAB file to improve operation of the test stand
- Thorough de-bugging of all software
- Developing a smaller, purpose-made power supply for the motor controller instead of using the large generic one
- Customization of the test stand to accommodate a wider variety of test objects
- Improve the brushes assembly (something more permanent and less thrown together)
- Improve the method of determining RPM (removing fragile cardboard strips from disks, removing optical sensor)

Software (Code)

MATLAB Code

```
% Originally written by Michael Arsenault
% May 2012
% Embry-Riddle Aeronautical University
% Dr. Aroh Barjatya

function varargout = BoomSpinTest(varargin)
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @BoomSpinTest_OpeningFcn, ...
                   'gui_OutputFcn',  @BoomSpinTest_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before BoomSpinTest is made visible.
function BoomSpinTest_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;
guidata(hObject, handles);

clc;

set(gcf,'CloseRequestFcn',@my_closefcn);

priorPorts = instrfind; % finds any existing Serial Ports in MATLAB
delete(priorPorts); % and deletes them

% NOTE: The port on your computer may be different than COM3.
global port % MUST be global so that it can be closed by other functions!
port = serial('COM3');
fopen(port);

set(handles.motorTest, 'Enable', 'inactive');
set(handles.rpmUp, 'Enable', 'inactive');
set(handles.rpmDown, 'Enable', 'inactive');
set(handles.motorTest, 'ButtonDownFcn', {@motorTest_ButtonDownFcn, handles});
set(handles.rpmUp, 'ButtonDownFcn', {@rpmUp_ButtonDownFcn, handles});
set(handles.rpmDown, 'ButtonDownFcn', {@rpmDown_ButtonDownFcn, handles});

global rpmSet % Value that the user sets the RPM to (desired RPM)
global running % If running, 1.  If not running, 0.
global motorTesting % If doing a motor test, 1.  If not testing, 0.
global rpmHolder % Hold the value of the previous rpmSet.  May not be needed?
global time % Array of times to use for plotting
global timeMax % timeMax is the x-axis range.  Doubles when the current timeMax is reached.
global rpmVals % Array of RPM values
global ran % If the test stand has been run and not reset (may still be, may be stopped).
global cutting % If currently cutting
global cutTime % The start and end times for cutting. Always an even number of values.

rpmSet = 2.5;
rpmHolder = rpmSet;
running = 0;
ran = 0;
timeMax = 1; % One minute
motorTesting = 0;
time = [];
rpmVals = [];
cutting = 0;
cutTime = [];
```

```
% Set up RPM plot
axes(handles.rpmPlot);

    title('Spin Frequency');
    xlabel('Time (min)');
    xlim([0 1]);
    ylabel('Freq (Hz)');
    ylim([0 7]);

% --- Outputs from this function are returned to the command line.
function varargout = BoomSpinTest_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in startStop.
function startStop_Callback(hObject, eventdata, handles)

global running
global ran
global time
global rpmVals
global timeMax
global rpmSet
global cutting
global cutTime
global port

if(running == 0) % If not already running

    running = 1;
    % Aesthetics
    set(hObject, 'BackgroundColor', 'red');
    set(hObject, 'String', 'Stop');
    set(handles.cutWire, 'Enable', 'on');
    set(handles.motorTest, 'Enable', 'off');
    set(handles.save, 'Enable', 'off');
    set(handles.reset, 'Enable', 'off');

    if(length(time) > 0) % If a previous run has been done
        i = length(time) + 1;
    else
        i = 1;
    end

    speed = 75; %Start at 75 - near the lower range of possible speeds for the motor
    fwrite(port, speed); % Send command to the micro controller
    pause(3.5); % Pause and wait for the motor to spin up
    a = tic; % Reference point for times.

    while(running)

        data = fscanf(port); % Data is a tab-delimited string of info sent from the micro controller to
matlab

        if(~isempty(data)) % If data not empty, parse it
           vars = textscan(data, '%s%d\n\r', 'delimiter', '\t');
           [str1, num1] = deal(vars{:});
        else
           fprintf('Empty\n');
        end

        % The RPM value sent by the micro controller is for 1/3 of a
        % rotation and is in milliseconds.  The total RPM (in Hz) is equal to the
        % following...
        RPM = 1/((double(num1)*3)/1000);
        rpmVals(i) = RPM;

        t = toc(a); % Current time of the previously calculated RPM

        pause(.01); % Pause MUST be included, otherwise interupts not possible (cannot hit "stop"
button)

        if(i == 1)
            time(i) = 0;
```

```
        else
            time(i) = t/60; % Convert the time to minutes
        end

        % A simplified control of the speed.  If too slow, speed up.  If
        % too fast, slow down.  Not perfect, but works well enough.
        if((RPM < rpmSet) && (speed <= 127))
            speed = speed + 0.05;
            fwrite(port, speed);
        elseif((RPM > rpmSet) && (speed > 69)) % 69 is about as slow as it can go
            speed = speed - 0.05;
            fwrite(port, speed);
        end

        % When the "Cut" button it pressed, turn on pin D6 for 2.5 seconds
        if(cutting == 1) % Cutting = 1 when "Cut" button pressed
            if(mod(length(cutTime),2) == 0)
                cutTime(length(cutTime)+1) = time(i);
                fwrite(port, 128); % Pin D6 on
                set(handles.cutWire, 'Enable', 'off');
            end

            if(time(i)-cutTime(length(cutTime)) >= 2.5/60) % 2.5 seconds
                fwrite(port, 129); % Pin D6 off
                cutTime(length(cutTime)+1) = time(i);
                cutting = 0;
                set(handles.cutWire, 'Enable', 'on');
            end
        end

        % Plot the RPM and Cutting vs. Time
        axes(handles.rpmPlot);

            cla
            hold all
            p1 = plot(time, rpmVals, 'b');
            set(p1, 'LineWidth', 2);
            if(length(cutTime)>0)
                for j=1:length(cutTime)
                    p2 = plot([cutTime(j) cutTime(j)], [0 10], ':r');
                    set(p2, 'LineWidth', 1);
                end
            end

            title('Spin Frequency');
            xlabel('Time (min)');
            ylabel('Freq (Hz)');

            if(time(i) >= timeMax)
                timeMax = 2*timeMax;
            end

            xlim([0 timeMax]);
            ylim([0 7]);%set yaxis range

        % Print the current RPM value in Hz
        if(running == 1)
            rpmString = sprintf('%.1f Hz', rpmVals(i));
        else
            rpmString = sprintf('0.0 Hz');
            fwrite(port, 0);
        end
        set(handles.realRPM, 'String', rpmString);

        i = i+1;
    end

    ran = 1;
else % If currently running
    running = 0;

    fwrite(port, 0); % Set motor speed to 0 (not spinning)
    % Aesthetics
    set(hObject, 'BackgroundColor', 'green');
    set(hObject, 'String', 'Start');
    set(handles.cutWire, 'Enable', 'off');
```

```
    set(handles.motorTest, 'Enable', 'inactive');
    set(handles.save, 'Enable', 'on');
    set(handles.reset, 'Enable', 'on');
end

% --- Executes on button press in save.
% Saves the most recent run to a text file (can be imported into excel and
% plotted)
function save_Callback(hObject, eventdata, handles)

global running
global ran
global time
global rpmVals
global cutTime

if(running == 0 && ran == 1)
    set(hObject, 'Enable', 'off');

    t = num2str(floor(now*10000)); % Completely unique string based on current time (date and time)
    file = strcat(t, '.txt'); % File to be written

    fileID = fopen(file,'w');
    % Report the cut times
    if(length(cutTime)>0)
        for j=1:2:length(cutTime)
            fprintf(fileID, 'Cut from %.5f to %.5f\r\n', cutTime(j), cutTime(j+1));
        end
    end

    % Print the RPM (Hz) and Times (sec)
    fprintf(fileID, 'Time\t\tFreq\r\n');
    for i=1:length(time)
        fprintf(fileID, '%.5f\t\t%.1f\r\n', time(i), rpmVals(i));
    end
    fclose(fileID);

end

% --- Executes on button press in reset.
% Resets everything back to original values (empties arrays, clears plot,
% etc)
function reset_Callback(hObject, eventdata, handles)

global running
global rpmSet
global time
global rpmVals
global timeMax
global rpmHolder
global cutTime

if(running == 0)
    set(handles.save, 'Enable', 'off');

    rpmSet = 2.5;
    rpmHolder = 2.5;
    time = [];
    timeMax = 1;
    rpmVals = [];
    cutTime = [];

    rpmString = sprintf('%.1f Hz (%.0f RPM)', rpmSet, rpmSet*60);
    set(handles.rpmSetValue,'String',rpmString)

    cla(handles.rpmPlot);
    axes(handles.rpmPlot);
        xlim([0 timeMax]);
        ylim([0 7]);%set yaxis range
end

% --- Executes on button press in cutWire.
function cutWire_Callback(hObject, eventdata, handles)
global cutting
cutting = 1;
```

```
% --- Executes on button press in motorTest.
function motorTest_Callback(hObject, eventdata, handles)
% ButtonDownFcn allows the user to hold the button down and not have to
% click it repeatedly
% Holding the "Motor Test" button will spin the motor at a speed of 74,
% near the lower end of possible speeds.  This is not for conducting tests,
% but to ensure the motor is working properly and the test is properly set
% up on the stand
function motorTest_ButtonDownFcn(hObject, eventdata, handles)

global running
global motorTesting
global port

if(running == 0)
    set(hObject, 'Value', 1);
    if(motorTesting == 0)
        motorTesting = 1;
        fwrite(port, 74)
    end
end

% --- Executes on button press in rpmDown.
function rpmDown_Callback(hObject, eventdata, handles)
% ButtonDownFcn allows the user to hold the button down and not have to
% click it repeatedly
% Set the RPM to a lower value
function rpmDown_ButtonDownFcn(hObject, eventdata, handles)

global rpmSet

set(hObject, 'Value', 1);

while(get(hObject, 'Value') == 1)

    rpmSet = rpmSet - 0.1;

    if(rpmSet<0)
        rpmSet = 0; % Cannot have a negative RPM
    end

    % Print the new RPM value
    rpmString = sprintf('%.1f Hz (%.0f RPM)', rpmSet, rpmSet*60);
    set(handles.rpmSetValue,'String',rpmString);

    pause(0.1);
end

% --- Executes on button press in rpmUp.
function rpmUp_Callback(hObject, eventdata, handles)
% ButtonDownFcn allows the user to hold the button down and not have to
% click it repeatedly
% Set the RPM to a higher value
function rpmUp_ButtonDownFcn(hObject, eventdata, handles)

global rpmSet

set(hObject, 'Value', 1);

while(get(hObject, 'Value') == 1)

    rpmSet = rpmSet + 0.1;

    if(rpmSet>7)
        rpmSet = 7; % Max rpm of 7 (the original motor can only do 6.9 max)
    end

    % Print the new RPM value
    rpmString = sprintf('%.1f Hz (%.0f RPM)', rpmSet, rpmSet*60);
    set(handles.rpmSetValue,'String', rpmString);

    pause(0.1);
end

% If any of the "button down" buttons is released
function figure1_WindowButtonUpFcn(hObject, eventdata, handles)
```

```
global motorTesting
global rpmSet
global rpmHolder
global port

if(motorTesting == 1) % If the motor was testing, stop it
    motorTesting = 0;
    fwrite(port, uint8(0))
    set(handles.motorTest, 'Value', 0); % Appearance of the button to unpressed
else
    set(handles.rpmUp, 'Value', 0); % Appearance of the button to unpressed
    set(handles.rpmDown, 'Value', 0); % Appearance of the button to unpressed

    if(rpmSet ~= rpmHolder)
        rpmHolder = rpmSet;
    end
end

% This function is called when closing the GUI.  The motor is stopped
% (speed of 0) and the port is closed.
function my_closefcn(hObject, eventdata, handles)
global port
fprintf('Goodbye\n');
fwrite(port, 0)
fclose(port)
close force

% --- Executes on mouse press over figure background.
% Does nothing.  Cannot delete or errors with the compiling.
function figure1_ButtonDownFcn(hObject, eventdata, handles)
```

Microcontroller Code

Note: This code is written for an ATmega 644PA Micro Controller

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <math.h>
#include <avr/interrupt.h>

#define UART_BAUD1 9600  // Baud for talking to computer software
#define UART_BAUD2 19200 // Baud for talking to motor controller (Pololu, TReX Jr)


//Define Baud Rate as described in AVR manual
#define UART_UBBR_VALUE1 (((F_CPU / (UART_BAUD1 * 8UL))) - 1) // Baud for talking to computer, 9600
#define UART_UBBR_VALUE2 (((F_CPU / (UART_BAUD2 * 8UL))) - 1) // Baud for talking to motor controller,
19,200 (Pololu, TReX Jr)

// Function Prototypes
void uart_init();
void adc_init();
void timer_init();
void write_uart1(unsigned char c[]);
void write_uart2(unsigned char c);
unsigned char receive();


int ADCflag = 0;

int main()
{
        int i = 0; // Counter variable
        int j = 0; // Counter variable
        int vals[4] = {0,0,0,0}; // Store the 4 most recent time intervals for the rotation
        int sum = 0; // Sum of the four values in vals[]
        int ave = 0; // Average of the four values in vals[]
        unsigned char buffer[15];

        adc_init();
        timer_init();
        uart_init(); // Initializes BOTH uarts (for talking to computer at 9600 and motor controller at
19,200)

        UCSR0B |= (1 << RXCIE0); // For received interupt - NEED THIS! IMPORTANT
        sei(); // Enable global interrupts

        // For the LED or wire-cutter.  Set the pin to output and then make sure it's off.
        DDRD |= (1<<DDD6);
        PORTD |= (1<<PORTD6);
        PORTD &= ~(1<<PORTD6);

        sei(); // Enable global interrupts

        while(1) // Loop waiting for interupts and sending the RPM of the stand to the matlab program
        {
                ADCSRA |= (1<<ADSC); // Start ADC

                if(ADCflag == 1)
                {
                        ADCSRA &= ~(1<<ADEN); // Disable ADC
                        ADCflag = 2;

                        vals[i] = TCNT1/4;

                        sum = 0;
                        for(j=0; j<4; j++)
                        {
                                sum = sum+vals[j];
                        }

                        ave = sum/4;

                        sprintf(buffer,"RPMstring\t%d\n\r", ave); // TCNT1 is the count of timer1
```

```
                        TCNT1=0; // Reset the timer back to zero
                        write_uart1(buffer);

                        ADCSRA |= (1<<ADEN); // Enable ADC
                        ADCSRA |= (1<<ADSC); // Start ADC

                        i++;
                        if(i == 4)
                        {
                                i = 0;
                        }
                }
        }
}



void uart_init()
{
// For RX and TX 0, to talk to the computer at 9600
        //Set Baud rate
        UBRR0H = UART_UBBR_VALUE1 >> 8;
        UBRR0L = UART_UBBR_VALUE1;
        UCSR0A |= (1<<U2X0);
        //Frame format: 8 data bits, no parity, 1 stop bit
        UCSR0C |= (1<<UCSZ01)|(1<<UCSZ00);
        //Enable Transmit and Receive
        UCSR0B |= (1<< RXEN0)|(1<<TXEN0);

// For RX and TX 1, to talk to the motor controller (Pololu, TReX Jr) at 19,200
        //Set Baud rate
        UBRR1H = UART_UBBR_VALUE2 >> 8;
        UBRR1L = UART_UBBR_VALUE2;
        UCSR1A |= (1<<U2X0);
        //Frame format: 8 data bits, no parity, 1 stop bit
        UCSR1C |= (1<<UCSZ11)|(1<<UCSZ10);
        //Enable Transmit and Receive
        UCSR1B |= (1<< RXEN1)|(1<<TXEN1);
}

void adc_init()
{
        ADCSRA |= (1<<ADATE); // Set ADC auto trigger enable bit
        ADMUX &= 0xe0;
        ADMUX |= 0; // Set ADCMUX to choose the MUX pin where the IR sensor is wired to
        ADCSRA |= (1<<ADPS1) | (1<<ADPS0); // Set ADC prescaler select bits for CLK/8
        ADMUX |= (1<<REFS0); // Set ADC reference to AVCC
        ADMUX |= (1<<ADLAR); // Set ADLAR pin to left-adjust for CLK/8
        ADCSRA |= (1<<ADIE); // Set ADC interupt bit
        ADCSRA |= (1<<ADEN); // Enable ADC
}

void timer_init()
{
        TCCR1B |= (1<<WGM12); // CTC with OCR1A TOP
        OCR1A = 2000*8; // Timer1 should reach TOP every 67s
        TIMSK1 |= (1<<OCIE1A); // Output compare interrupt
        TCCR1B |= (1<<CS10) | (1<<CS12); // CLK/1024
}

void write_uart1(unsigned char c[])
{
        do
        {
                while((UCSR0A&(1<<UDRE0)) == 0); // Wait for Transmit Buffer to Empty
                UDR0 = *c; // Transmit the character
                c++; // Increment the pointer to point to the next character
        }while(*c != '\0');
}

void write_uart2(unsigned char c) // Sending commands to the motor controller (Pololu, TReX Jr)
{
                while((UCSR1A&(1<<UDRE1)) == 0); // Wait for Transmit Buffer to Empty
                UDR1 = c; // Transmit the character
}
```

```
// Timer Interupt
ISR(TIMER1_COMPA_vect) // this interrupt is fired when timer 1 equals OCR1A
{
        write_uart1("RPMstring\t0\n\r"); // Write the string to the UART
}

// ADC Interupt
ISR(ADC_vect) // When each ADC completes, this routine is executed
{
        int tripCount = 240;

        if(ADCH >= tripCount && ADCflag == 0)
        {
                ADCflag = 1;
        }
        else if(ADCH >= tripCount && ADCflag == 1)
        {
                ADCflag = 2;
        }
        else if(ADCH <= tripCount && ADCflag == 2)
        {
                ADCflag = 0;
        }
}

// Receive Interupt
ISR(USART0_RX_vect)
{
        char c = 0;
        char k = 0;

        k = receive();

        // If the received bite is between 0 and 127 inclusive, send that same value to motor
controller (speed)
        if(k <= 127)
        {
                c = 0xCA;
                write_uart2(c);
                c = k;
                write_uart2(c);
        }

        // If the received bite is 128, turn on pin D6. If 129, turn if off.
        else
        {
                if(k == 128)
                {
                        PORTD |= (1<<PORTD6); // Turn on
                }
                else if(k == 129)
                {
                        PORTD &= ~(1<<PORTD6); // Turn off
                }
        }
}

unsigned char receive() // Return the received character
{
        while ((UCSR0A & (1<<RXC0)) == 0);
        return UDR0;
}
```