



# USER MANUAL

## Twister and Twister Attachment

### Abstract

This document contains information regarding the construction and design of the Twister and the Twister Attachment. Furthermore, this document has potential solutions to problems that the user may encounter with when using or modify the device.

Sally M. Mei  
Austin Steiner  
Jacquelin Remaley  
Robert Regent

# Table of Contents

<b>Background</b> .....	<b>2</b>
<b>Quick Start Guide</b> .....	<b>3</b>
<b>Twister</b> .....	<b>4</b>
Frame Assembly .....	5
Torque Arm .....	8
Harness .....	12
Motor .....	14
Roller/Feet .....	17
<b>Lateral Attachment</b> .....	<b>19</b>
Attachment Bar .....	20
Attachment Harness .....	24
BNC Connector and String Potentiometer .....	27
Electromagnets .....	29
Weight Housing .....	30
<b>Electrical/Software</b> .....	<b>32</b>
GUI .....	33
Electromagnet Code .....	39
Motor Code .....	40
Power Supply .....	41
Torque Sensor and Voltage Regulator Circuit .....	44
Solid State Relay .....	45
<b>Appendix</b> .....	<b>46</b>
Arduino Motor Code .....	47
Visual Studio User Manuel C# Code .....	54

## Background

Dr. Victor Gurfinkel is a retired neurologist who was a professor at Oregon Health and Science University. Dr. Gurfinkel and Paul Cordo developed the Twister device to measure human muscle tone in torsion. When Dr. Gurfinkel retired, the Twister was then donated to Dr. Rajal Cohen at the University of Idaho. The Twister is used to measure torsional resistance and muscular responses in upright subjects during the twisting of the human body. The device can be configured to study various aspects of tonic control across the neck, trunk, and/or hips. To test the stiffness of the subject, the Twister has a rotating base plate that the subject stands on. As the body slowly twists, the torque sensors attached to the harness that the subject wears will record the data.

Postural habits can greatly affect a person's life. Studies have shown that patients with certain postural intentions had corresponding postural alignment traits. A person's state of mind can influence how they act, move, and react to known and unknown lateral forces against them. A hunched and arched posture can influence other muscles in the body and can increase the chances of injuries. A person with relaxed mood will typically have a relaxed posture, as will a person with a high stress levels will usually have a rigid posture. This can be applied generally to any person, regardless of age. The leading accidental causes of death for people aged 60 and above are falls. Senior citizens that do survive a fall have a 90% chance of a fracture of the hip. 40% of people with falling related injuries will be in chronic pain and will be dependent on medical assistance (). Risk factors for falls include ergonomic factors as in falling off a ladder, tripping over a curb, or slipping. Intrinsic factors like age and disease also contribute to an increased risk of falling. Postural instructions, lighten up, relax, and pull, also affect mobility. When a person knows that a force is going to be exerted towards them, their muscles will preemptively tense up creating a reaction that is completely different than an unknown force contacting a person. Comparing data related to mental responsiveness to physical responsiveness will help researchers figure out how and why people fall and the cause of negative postural habits.

Given how dangerous it can be for people when they fall, posture is the key to avoid such situations. Therefore, the Twister will be used to collect data to better improve posture in humans. In addition to the Twister, an attachment will be added to the device that will measure the lateral (side to side) rigidity of the human body.

**Please refer to Quick Start Guide file.**

---

# TWISTER

---

# Frame Assembly

Austin Steiner

## Background

The frame of the twister was already built at the beginning of the project. It is constructed from 1.5" eighth wall square tubing and can be divided into three separate welded sections; the base, top and vertical bars. Each of these sections can be unbolted from the rest for ease of transport.

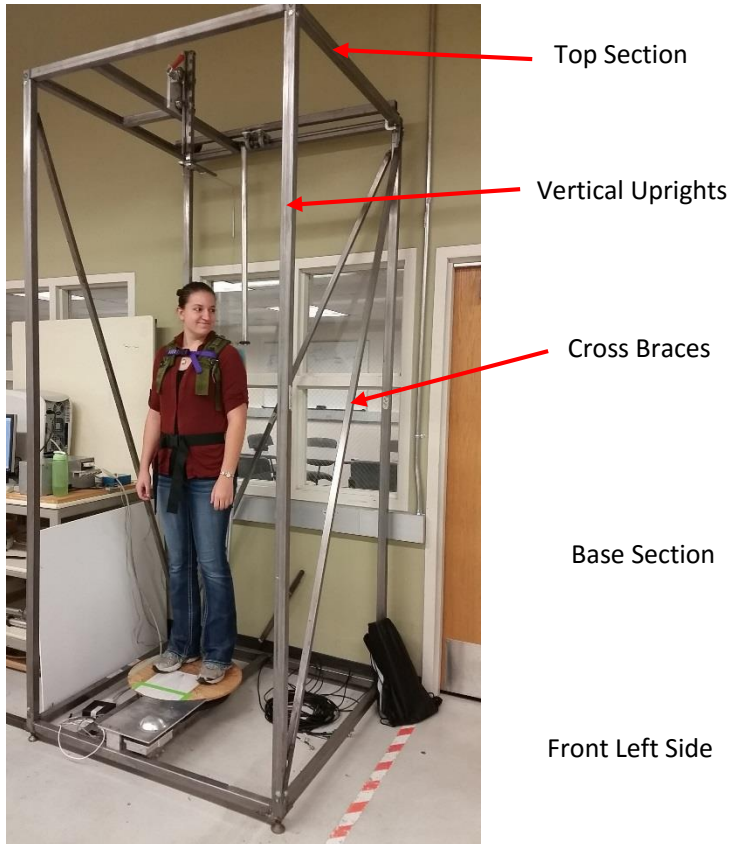


Figure 1. Old Twister Frame Construction



Figure 2. New Frame Construction

## Design

The original frame was too tall to fit in the Mind in Movement laboratory and thus had to be shortened to 8ft. A total of nine inches was cut off of the top of the vertical bars and all of the bolt holes were re-drilled. Also, because the frame was shortened, the cross bracing that was in place to stiffen the frame required shortening. The last design feature to overcome was the diagonal bars that were on the sides of the frame. Because of the new attachment bars that were added there bars could no longer extend all the way from corner to corner. This was rectified by simply shortening the bars so that they triangulated the top corner instead of the entire side.

## Procedure

Assembly of the twister from the ground up can be done by one person but it is much easier and safer with two.

1. Place the base of the twister on flat level ground and ensure that the support feet are extended.



Figure 3. Feet and Rollers

2. Each of the vertical uprights are labeled from the point of view of standing inside the twister. They are also labeled at the top and bottom so that there is no confusion as to which way they should be installed.
3. There are two bolt holes in the top and bottom of every tube and these line up with the holes in the top and base. Everything is bolted together with short  $\frac{1}{4}$ -20 bolts that have a  $\frac{7}{16}$  hex head along with a washer. There are two different lengths used. The shortest ones are for holes that don't also support a cross brace and the long one are for holes that do. The exception is the bolts on the top of the front right bar which uses a small bracket along with short  $\frac{1}{4}$ -20 bolts with no washers and allen head cap screws.



Figure 4. Bolt Locations



Figure 5. Bolt Sizes

4. After the uprights are installed then the top can be lifted into place and bolted on in the same manner.

Top right corner assembly. Allen bolts go on the front side. Short bolts are used without washers on the side.

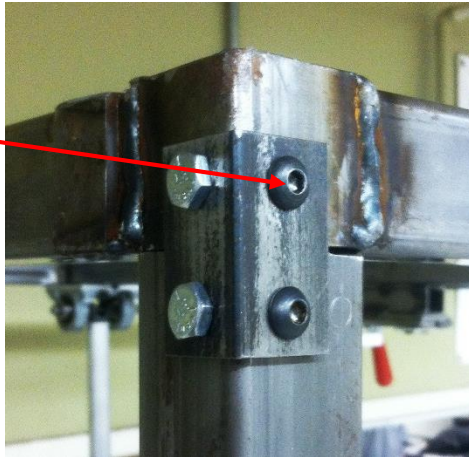


Figure 4. Upper Right Mounting

5. Next the cross braces on the sides and back can be installed. The long one goes on the back and the two short ones are interchangeable from side to side.
6. The last part that is installed is the fixture arm in the back of the top which is held on by two  $\frac{1}{2}$ " bolts with  $\frac{9}{16}$ " heads.



Figure 5. Body Fixture Mount

### Improvements

This system works very well as-is but the frame could benefit from additional cross bracing to stiffen it up even further. It also could use some more fine-tuning on the bolt holes so that the entire assembly lines up and goes together with less tweaking.



# Adjustable Torque Arm

Austin Steiner

## Background

The job of the torque arm on the twister is to transmit the torque from the harness up to the torque sensor mounted on the top of the frame. The old system utilized a series of pivoting plates that allowed movement in three planes without movement in torsion. While this was a very rigid and effective design it did have a few drawbacks. First the plates were held together with spring steel that caused a resistive force when expanded or contracted. This meant that the subject was not able to move freely which could potentially skew the test results. The second drawback was that it required a large amount of vertical space to mount and work effectively. Because the twister was to be moved into a space with a maximum height of 8ft this was not a feasible method to transfer torque.



## Design

The new design greatly simplifies the mechanical aspect of the torque arm and provides an almost resistance free movement while still rigidly transferring torque. Beginning from the top, a U-joint is used to provide translation along the x and y axis. Next a series of telescoping tubes are used to provide the initial height adjustment. These tubes are easily interchangeable for different heights and mounting locations of the harness. Finally, at the bottom of the arm, a linear rail is mounted to allow around four inches of vertical travel even while the adjustable tubes are fixed. An attachment bracket is also used to couple the torque arm to whichever harness is in use.

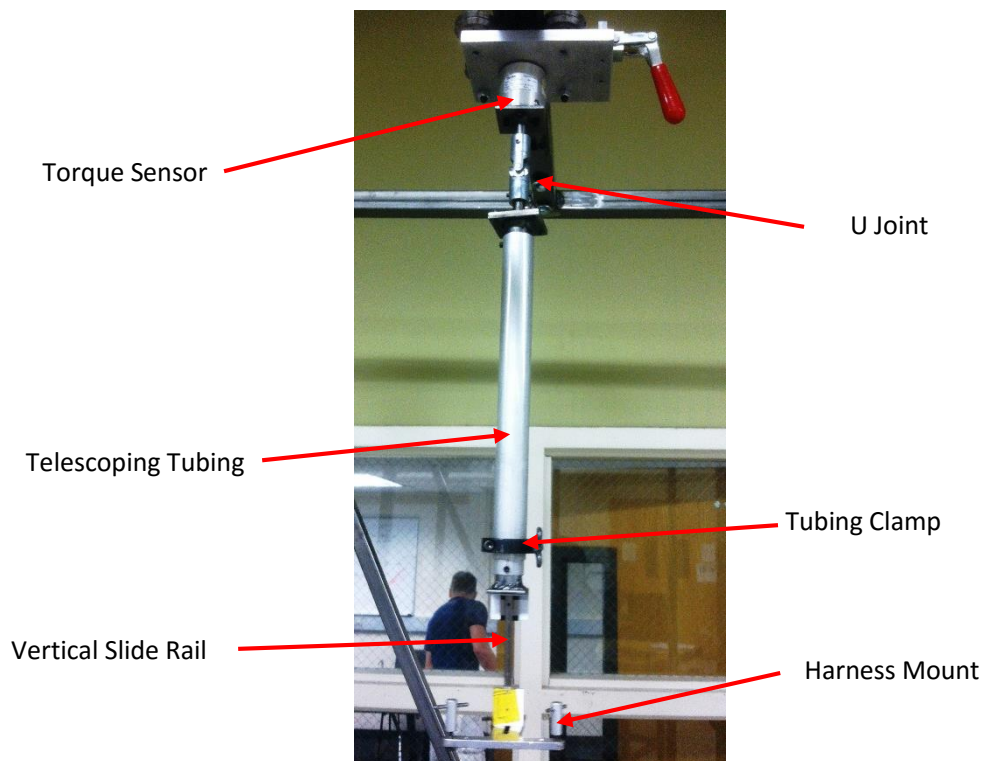


Figure 1. Torque Arm

## Procedure

The proper method to use the torque arm can be broken down into a few simple steps.

1. Choose the telescoping tubing that provides the appropriate length.
  - Depending on the height of the test subject and/or the position of the harness being used it is necessary to change the length of the torque arm. This is done by removing the two screws seen to the right. Once removed the two tubes can be removed from the arm and replaced with the proper length.



Figure 2. Tube Screw #1



Figure 3. Tube Screw #2

- Tube size pairs are available from 3-6", 6-12" and 12-24". Any one of these tubes can be used with all the others and there is also a 36" piece for extra-long reach.



Figure 3. Telescoping Tubing Sizes

## 2. Adjust the Telescoping Tubes

- Once the proper tubes are selected then they must be adjusted to the best position prior to testing. The optimal positioning of the tubes places the linear slide rail at the middle of its travel so that the subject can move up or down without restriction.
- Adjustment is done by loosening the wing nut on the clamp until the tubes can be easily slid in and out of one another. When the chosen point is reached the clamp is re-tightened and the tubes are then fixed in place.

## **Improvements**

While the torque arm assembly functions fairly well in its current form it could benefit from a few improvements.

1. The u-joint that is being used right now is not of a high enough quality. Because of this there is a slight amount of play when subjected to a torsional force which could potentially throw off the torque readings during the test. At the very least this slop will cause a delay in the torque reading which is not desirable. To fix this a high precision needle bearing joint is needed. These are substantially more expensive but in order to achieve the desired rigidity and freedom of movement a bearing type u-joint is the only option.
2. A second u-joint is needed at the bottom of the torque arm as well to maintain proper geometry when the test subject leans in any direction. In the current design with one joint the radial movement pattern will cause the subject to tilt the farther they get from center as illustrated below. However with a second joint in the system this is completely corrected and the test subject can move in any direction without an additional forces being placed on them by the torque arm.

## Harnesses

Jacquelin Remaley

### Background

There are four harnesses for use with the Twister. They should not be constricting. You should be able to pull gently on the attaching metal on the back side of the harnesses without them shifting on the subject's body. The subject should put on the harnesses before entering the Twister.

### Pelvic Harness

- This rests above directly above the hips with the opening on the front of the person. For comfort, this harness should be slipped onto the subject's torso level with the belly button, then slid down until it rests on the hips. The buckle is then fastened and tightened so the harness is snug.



### Twister Chest Harness

- This is worn on the chest for use with the Twister's rotating plate. It is worn like a backpack with the straps snug.



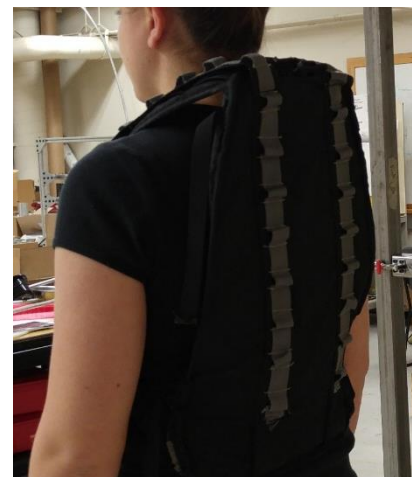
## Head Harness

- This is worn similarly to an adjustable hard hat or welding helmet. It has a knob on the front side which can be depressed and spun. While holding it down, it will spin and increase or decrease the circumference that will fit around the head. This harness should begin as large as possible. Then, either with assistance or alone, gently turn the knob while the back rests against the head until the foam is gently pressing on the forehead.



## Axial Force Chest Harness

- This is similar to the Twister chest harness, except it is designed for use with the axial force attachments. It is worn in the same fashion as a regular backpack with the straps tight enough so the harness is snug on the subject. The loops on the front and back are for attaching the cables that travel over the pulleys. The cables have carabiners to attach to loops at the same level on the front and back of the straps.



## Motor System Overview

Robert Regent

### Background

“Twister” is powered by a worm gear drive. The worm gear drive and the turn table each have pulley attached to them. The pulleys are 1.21” in diameter and have a pitch of 0.8”. An MXL timing-belt transfers the power produced by the motor and worm gear to the turn table. Pictured to the right is a picture of the “Twister” Motor Drive System. Gearbox, motor, and motor tensioner are pictured in the center of the turntable base. The turn table base is composed of a top piece and bottom piece with roller bearings sitting in between them, to allow for smooth rotations.



Figure 1 Top View of Motor System with Turntable detached

### Design

The motor and gearbox assembly is shown in Figure 3. The assembly is composed of the driving motor, the motor mount, gearbox, and pulley. Inside the gearbox is a worm gear, which translates the power from the motor to the main pulley.

A closer view of the drive components is pictured to the right. The blue arrow shows where the motor is in this assembly. The motor powering “Twister” is a Maxon 30 watt brushless flat motor. The datasheet can be seen on the project Wiki page. The motor turns the worm gear in the gearbox which turns the pulley, shown to the right above the purple arrow. The other end of the timing belt is attached the pulley on the turn table.

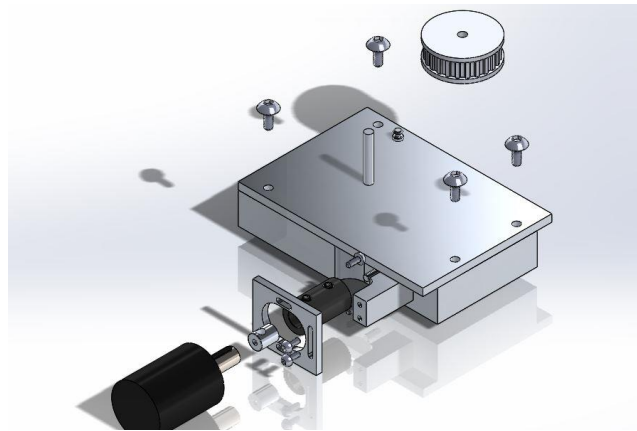


Figure 2 Gearbox Assembly

The belt is kept in tension by the belt tensioner, shown with the red arrow in Figure 4. The belt tensioner uses a right hand wound torsion spring to rotate it into the belt, keeping the belt in tension. Holes were drilled into the gearbox, which is where the bottom end of the spring is placed. The top end of the spring is placed in a 0.0625" diameter hole located on the bottom the spring tensioner body. The two pulleys have a 1:1 gear ratio, which is taken account for in the motor code.

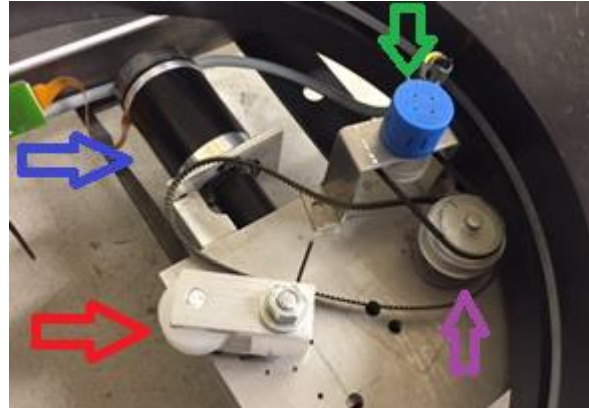


Figure 3 Close up view of drive system

The original “Twister” used a photodiode located under the turntable base to keep track of the position of the turntable. The new system uses a Bourns 10-turn 5k ohm precision potentiometer to keep track of the angle of the turntable during rotation cycles, and is pictured in Figure 4 with the green arrow above it. The potentiometer is sitting upside-down in a mount, and has a small toothless white gear attached to it. The gear is made of ABS plastic and has a small belt attaching it to the main motor pulley. Testing showed that the potentiometer did not have a high enough resolution, and thus, gave inaccurate positional data. For future use of “Twister”, it is recommended that the 10-turn potentiometer be changed to a single turn potentiometer, as the turntable never rotates more than +/- 60°.

All passive components in the Motor Assembly are replaceable and the aim of the system was to make it as modular as possible to help with future repairs and replacements. All part numbers are listed in this report.

## Procedure

To disassemble to motor assembly, first unscrew the four fasteners on the motor plate. These fasteners secure the wooden top as well as the metal top, which holds the pulley. When taking the metal top off, lift gently, as there is a belt attached to it securing it to the gear box pulley. The turntable assembly is detached next. The two piece, black turn table assembly is secured to the “Twister” base by hex nuts on the bottom piece of the turntable. Rotate the top piece of the turn table until a hex bolt can be seen through the hole on the top piece of the turn table. Unscrew the 4 hex bolts and lift the turn table assembly from the base. Once the two top plates and the turntable assembly are off the assembly, and assuming that “Twister” components are disassembled from the base, flip the base over. Next,

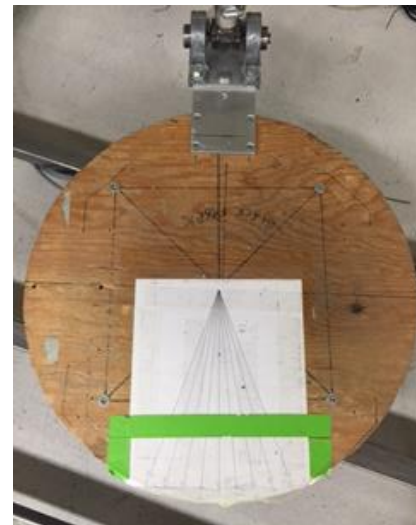


Figure 4 Top view of Motor Assembly

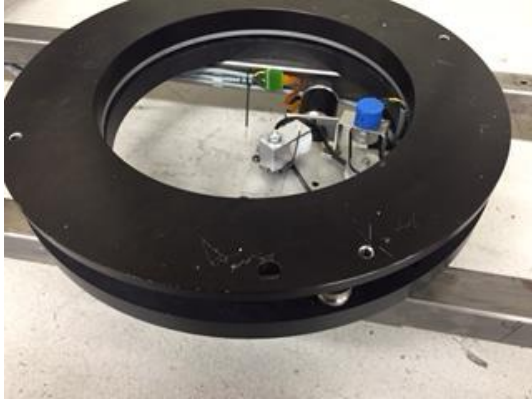


Figure 5 Turntable assembly

disassembly the four supporting bars, which secure the gear box in place. Once these steps are completed, full access to the motor components is available.

To reassemble the motor assembly, first secure the gear box to the “Twister” base through the four gear box supports. Next, flip the base over and secure the turn table assembly to the base. Attach the timing belt to the pulley attached to the gear box and set and secure the metal plate and wood plate to the turn table assembly. While doing this, remember to put the timing belt around the pulley attached to the metal plate. The final step is to reach under the assembly

and put the timing belt in line with both pulleys. This can be difficult, to test the connection between pulleys, run “Twister” at the lowest turntable speed available.

### **Improvements**

The potentiometer used for turntable position is a 10-turn precision potentiometer. Because the turntable only rotates  $\pm 60^\circ$ , the resolution is very low and cannot supply accurate positional feedback. For future modifications, it is suggested to either replace the potentiometer with a two-turn or single-turn potentiometer, or replace the potentiometer with a digital rotary encoder or optical encoding system.



## Swivel Feet and Rollers

Sally Mei

### Background

As part of the design specifications, the Twister needs to be easily mobile inside the lab, yet the design must provide the stability that is needed while testing is being conducted; i.e. it can't be sliding around. The Twister originally came with swivel feet at each of the four corners on the bottom of the base; as shown in *Figure 1*. Though, the swivel feet worked fine with providing the stability, it didn't allow for free movement when the Twister needs to be moved. Therefore, to accommodate for the new design specification, rollers were added and the swivel feet were replaced with newer feet; as shown in *Figure 2*.



Figure 6



Figure 2

### Design

The original feet shown in the *Figure 1* still worked, but one of the feet had a different thread size (it had a different diameter shaft). So, it was decided that the feet should get an upgrade. As partially seen in *Figure 2*, all the feet are now the same size and the Twister now has rollers for ease of mobility. There was another design that the team considered, and that was to just replace the feet. The feet for this design would look a little different, it would have a higher plastic cap, (*Figure 3*). We would rely on the plastic cap to provide the mobility on the carpet floor. However, this design was chosen over the one because we wanted it to have the ease of mobility on all surfaces.

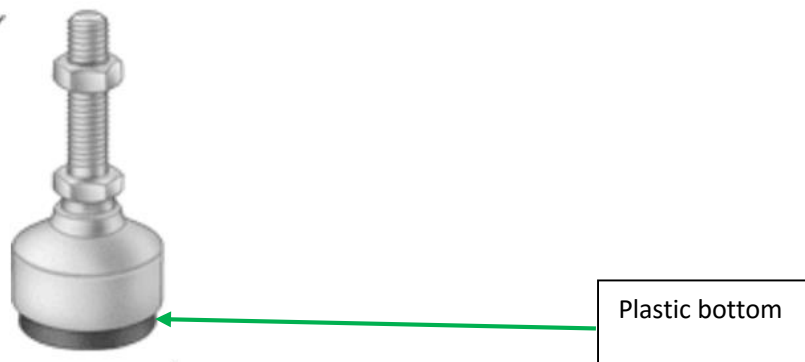


Figure 3

## Procedure

If the Twister are on the rollers:

1. The swivel feet can be lowered using a wrench by screwing the nut indicated shown in *Figure 4*.
2. The feet only need to be low enough to disengage the rollers from contacting the ground; this can be seen in the *Figure 4* as well.
3. Do step 2 on all four corners.

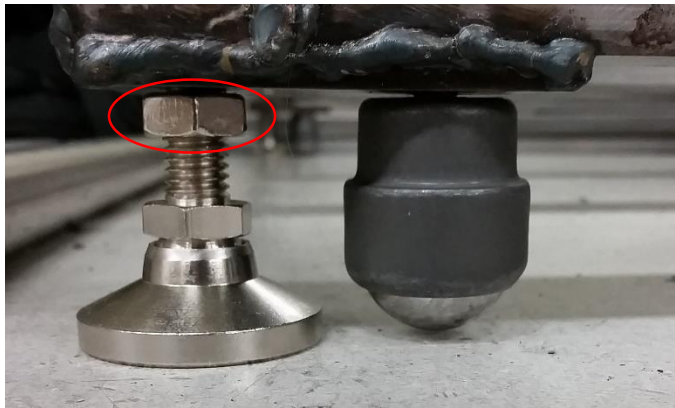


Figure 4

If the Twister are on the feet:

1. To engage the rollers, the nut shown in *Figure 4* would have to be screwed counter clockwise using a wrench.
2. The nut only needs a few turns, enough to engage the rollers.
3. Do step 2 on all four corners.

## Improvements

1. The rollers and the feet can switch positions shown in the figure below. This will provide a more even support when the Twister is being moved around. In theory, if this change was implemented, there shouldn't be an effect on the balance or stability when the feet are engaged.

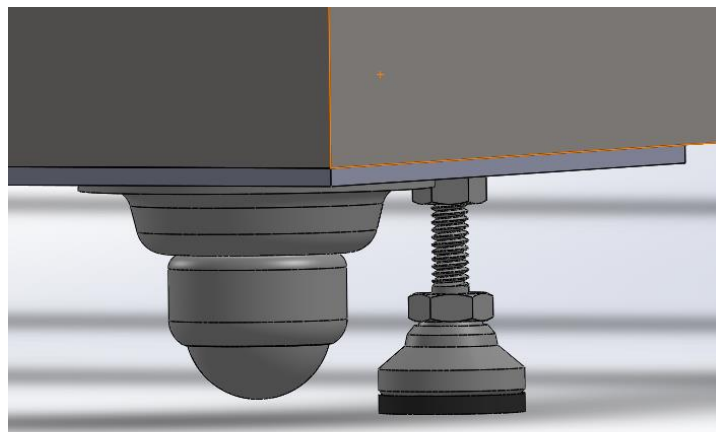


Figure 5

---

# LATERAL ATTACHMENT

---

# Attachment Bar

Sally Mei

## Background

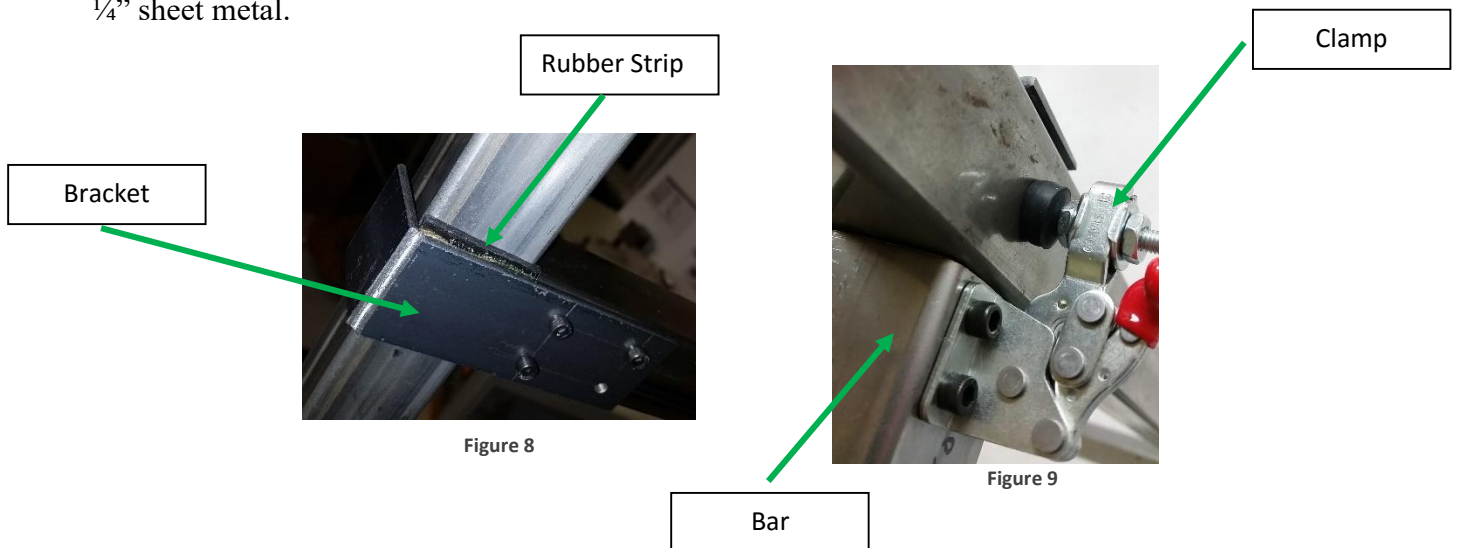
The Twister is a device that measures torsional stiffness in a human body. As part of a new test for the side to side lateral movement, a new assembly must be added. The Attachment bar is one of the main parts of the lateral force test on the Twister. (Please see Attachment Assembly for full assembly layout). The Attachment bar serves as the adjustable bar that shelves the pulley and BNC housing; it also provides the support needed for the weights to be hanging off the pulleys. The bar needs to be able to support a minimum of 20 [lbs]. Calculations have been done to calculate the amount of force the bar can hold, which is approximately 50 [lbs], with a safety factor of 3.



Figure 7

## Design

The design requirements for this bar was that it needs to be able to support a minimum weight of 20 [lbs], it to be easily adjustable, and compact. The assembly has four different parts, the clamp, the bar, the bracket, and the rubber strip. The clamp and rubber strip was purchased form McMaster-Carr. The bar is a piece of 1"x1" steel tube with 1/4" walls and the bracket is a piece of 1/4" sheet metal.



There were three different designs that the team came up with. This design was one of them, the other was to have a bar that has two sleeves that runs up and down the corners and use a thumb screw to hold the bar in place. The last design is similar to the thumb screw and sleeve design, but this design requires holes to be drilled on the corner bars or the Twister and use a pin to hold the bar in place. The two alternative design are shown in the two figures below.

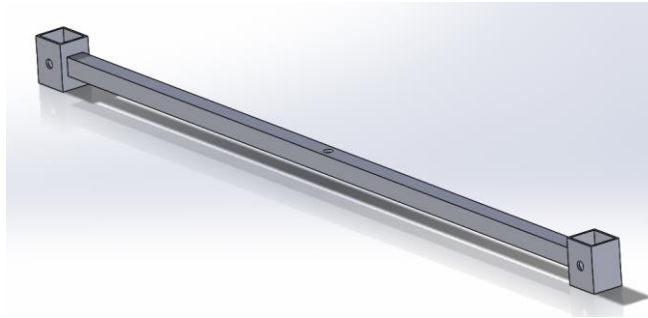


Figure 4



Figure 5

The reason the team went with the chosen design is so that that Attachment bar won't cause any damage to the structural strength of the Twister frame.

### Procedure

The Attachment bar is easily removable and adjustable. To remove/adjust simply pop the red lever to release. Do this on both sides of the bar and you can remove or adjust the bar up and down.



Figure 6

The figure below shows that there is a ruler on the side the bars to help assist adjusting the Attachment bar. The ruler starts at the point where it is parallel to the base plate (what the person is standing on). The ruler goes in increments of one foot, until it reaches to the bottom of the cross bars.

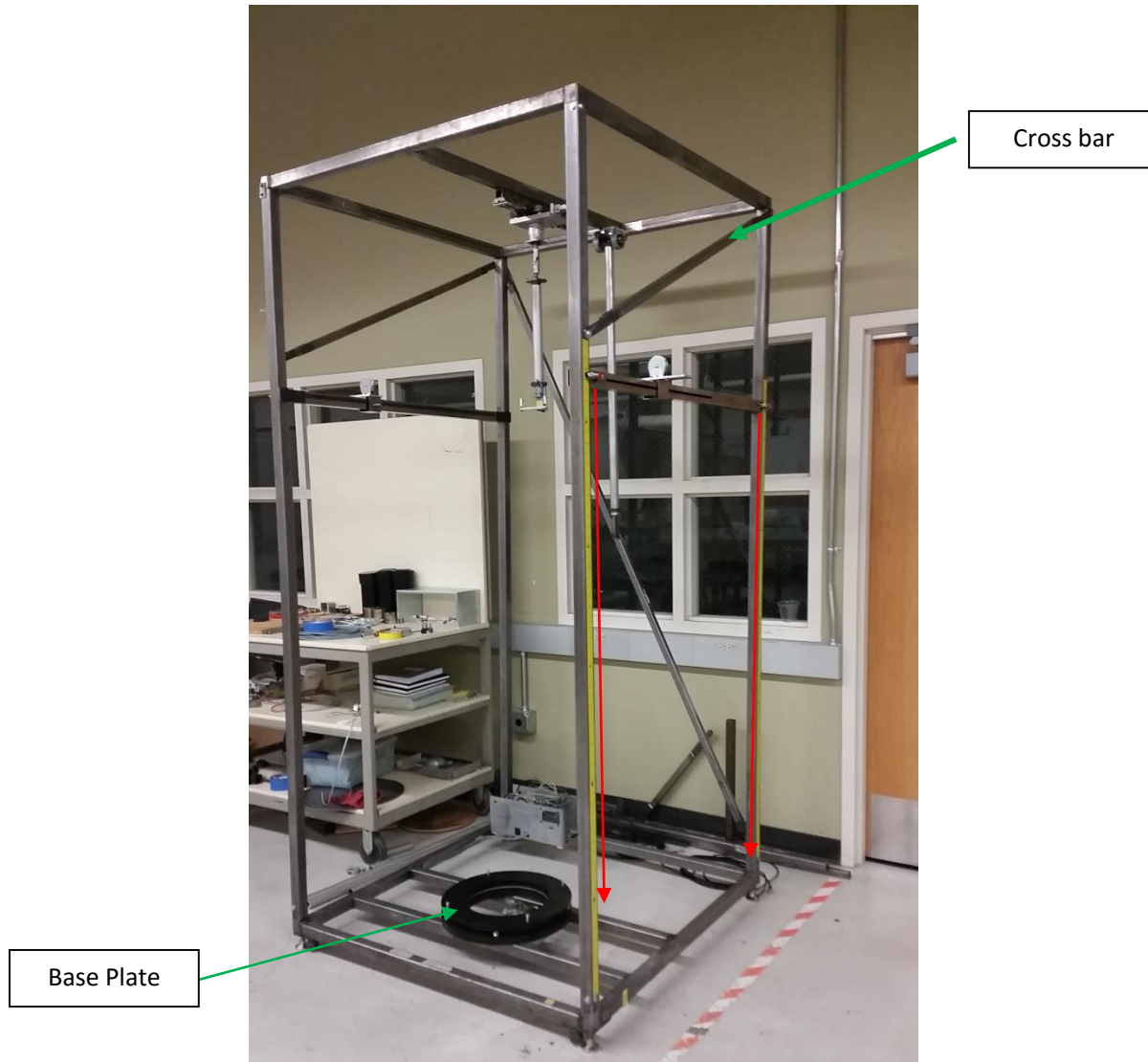


Figure 7

### Improvements

1. The Attachment bar design works pretty well overall, one of the adjustment/improvements that should be done to the design is to find a different method in attaching the rubber strip to the bracket. Currently the rubber has an adhesive back and that is sandwiched in between the bracket and the rubber strip itself. Though it is convenient and simple, the adhesive is actually acting as a lubricant, allowing mild slipping to occur; this can be seen in *Figure 3*.
2. Another improvement that this bar needs is possibly a different set of toggle clamps. The current toggle clamps work, but they are a little hard to get off for a level clamp.

Perhaps a slightly larger clamp to provide a larger surface area to grip onto or a larger lever for the current clamp.

# Attachment Harness and Pulley System

Sally Mei

## Background

As part of the new lateral force test attachment for the Twister, the harness provides the anchor point for the weights to be attached to the test subject. (Please see Attachment Assembly for full assembly layout). The design specifications for this part: the vertical adjustability for the weights to be attached to, the tightness of the harness, and the strength needed to support the weights. The vertical adjustability is needed because the lateral force test may require the weights to attach to the harness at various locations. The tightness of the harness is referencing, how “snug” the harness is on the test subject. The strength is referring to the material that the harness is made out of. It needs to be able to support the weights that will be hanging off of it. The other portion of this lateral force, is the pulley system. The pulley system is comprised of the cables, clips, bracket, and pulley. The pulley system, which will have the electromagnets holding on the weights will be attached to the harness via the cable and pulley.

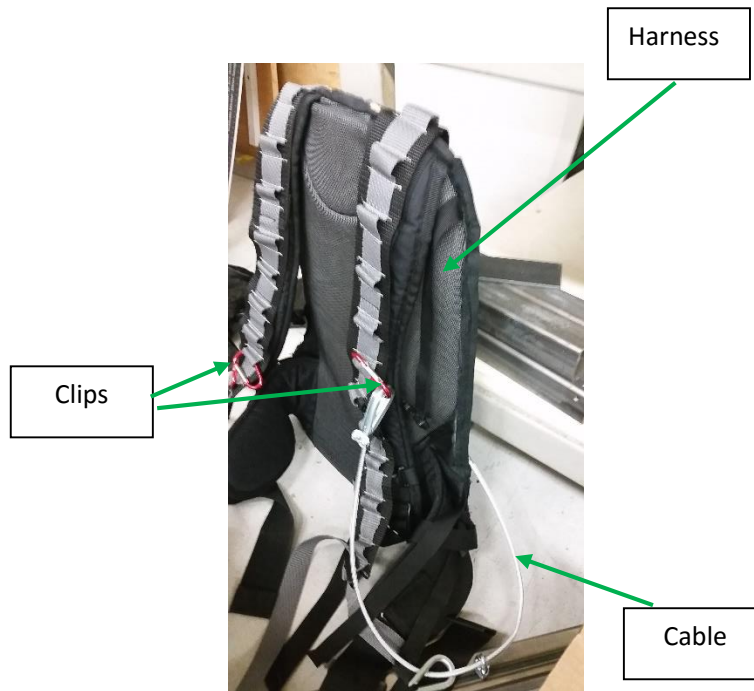


Figure 10



Figure 11



## Design

As listed above, the Attachment harness has several requirements. First it needs to have the vertical adjustability for the weights to attach to. To implement this design requirement, we decided that little carabiner clips will serve as the anchor for the weights, while the grey mesh as seen in *Figure 1* will be the anchor location. The next requirement is that the harness needs to fit “snuggly” on the person, such that it mimics as if the weights are being directly attached to the person. However, the harness must also be comfortable to wear, so this it doesn’t distract the test subject, and possibly screw the test results. So we went with the body of a hiking backpack, with the black and grey mesh sewn onto the backpack. Lastly, the harness must have the strength to hold up the weights, i.e. the weights don’t tear the harness apart. Since the body of the backpack is used for hiking, the body should have the strength to withstand 20 [lbs] of weight. The black and grey mesh have a tensile strength higher than 20 [lbs] as well. Finally, the harness was sewn together by a leather shoe repair shop, meaning the thread that is used to sew the harness together is also very durable. Therefore the harness shouldn’t fail under the maximum force that a person may be subjected to.

The pulley system is there to help guide the cables and provide shelving for the BNC housing. It wasn’t ideal to attach the pulley directly to the Attachment bar itself, because the pulley would then be fixed in place, and it won’t allow for any adjustment. Therefore to make it a more feasible for future testing, we made the pulley adjustable along the bar.

## Procedure

There is the harness, a short cable with two carabineers, a longer cable that is looped with the shorter cable, and has eye bolt at one end. First the harness must be secure on the person. Next, clip the shorter cable with the carabineers onto the harness in the front and in the back, as shown in the figures below, do this on both sides. Be sure the clips are parallel with each other in terms of vertical height. Next be sure the longer cable with the eye bolt goes over the pulley and into the slot as shown in the *Figure 4*. (The eye bolt is used to hold onto the electromagnets).



Figure 12

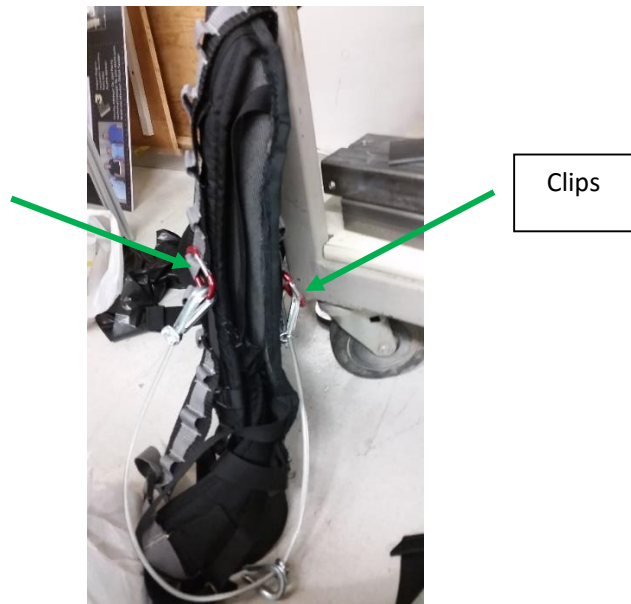


Figure 13

## **Improvements**

1. The brackets that the pulley and BNC housing sits on can be redesigned such that it is a better fit for the rope. For example, the slot can be a little bigger and the bracket should be centered, so that the part is symmetric.
2. The rope itself can be change to a more flexible material, either a smaller gauge wire or a rope.

# BNC Connector and String Potentiometer Housing

Sally Mei

## Background

The lateral force attachment needs a string potentiometer to record the displacement (torque) data and send it to Motion Monitor. To buy a genuine string potentiometer, it cost about \$200 dollars. Which is isn't feasible, because we need two string potentiometers on each bar. In addition to the sting potentiometer, we need a BNC connector such that it can be hooked up to the data acquisition box: Motion Monitor. Since buying a sting potentiometer was expensive, we found a website that provides the STL files, and sells a kit to make a string potentiometer. WE bought a kit and 3D printed a case to test the "homemade" string potentiometer. It worked really well, so we decided to design a string potentiometer case and add a cover to the back side to shield the BNC connector as shown in *Figure 2*.



Figure 14

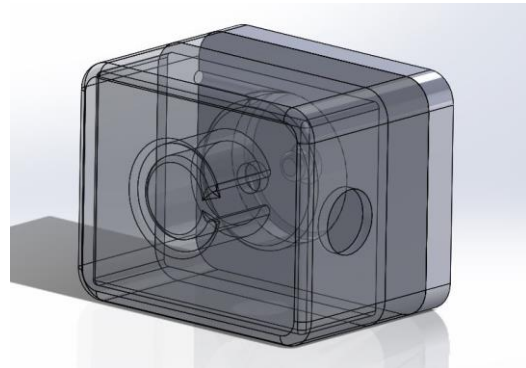


Figure 2

## Design

Most of the design was already done, because it is base off of a working design. The geometry changed so that it can sit on the bracket, have a BNC connector connected to the potentiometer, and for ease of assembly. The housing has four 3D printed pieces, string, potentiometer, and a BNC connector. As shown in the figure below the first piece is to shield the BNC connector and the potentiometer. The second piece is for the spool and string. The third piece is the cover that goes over the spool and string.

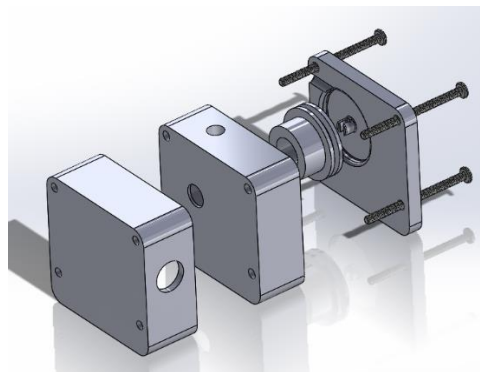


Figure 3

## **Procedure**

The housing shouldn't be disassembled unless necessary. The BNC connector needs to hook up with the BNC connector that is on Motion Monitor, through a BNC cable. The other cable that leads into the BNC housing is the wiring for the potentiometer that goes to the 5V power supply circuit in the silver box. The string needs to be attached to the intersection of the rope that is attached to the harness rope, as shown in the figure below.

## **Improvements**

1. The housing can be redesigned such that it doesn't have to be this big. (currently it is approximately 2"x2"x2").
2. The extrusions on the inside of the housing can be re-measured to ensure the spool to spin smoothly.

## Electromagnets

Jacquelin Remaley

### Background

The electromagnets are used in axial force application. They hang roughly two feet off the ground on either side of the Twister from cables that travel over the pulley wheels and attach to the subject.

The magnets have simple on/off operation. They are connected to a 24V power supply via a solid state relay controlled by the Arduino Mega. When the correct instructions are received, the relay will either allow current to flow or shut it off from the magnets, turning them on or off, respectively.

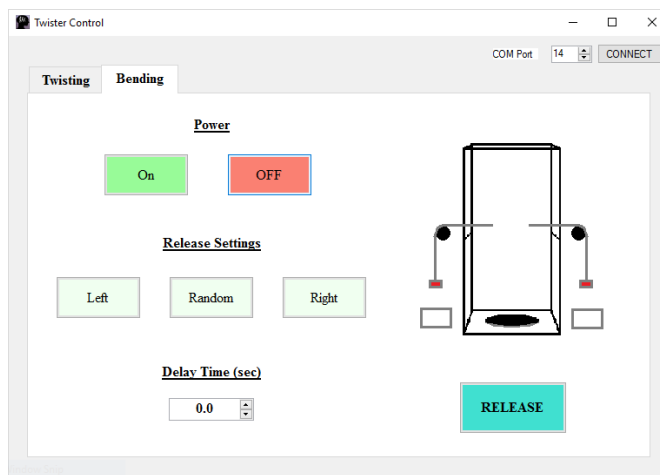


There is a built-in safety for the interface where if the user leaves the Bending tab, it shuts off power to the magnets, regardless of whether they were on or not. In addition, if the user attempts to release the weight before they are powered, an error message will appear with the text “Magnets not powered.”

The user interface includes a section for controlling the electromagnets. These instructions are also detailed in the section outlining general use of the interface.

### To apply axial force to the subject:

1. First, place the weight capsules with the desired weight disks below the magnets, making sure they are touching.
2. Ensure the COM port is connected. If you are unsure, connect again. The program will give an error if it cannot communicate with the Arduino.
3. Remove any sensitive electronics from the immediate vicinity of the magnets, as their electromagnetic field can damage them.
4. Turn the power to the magnets on by selecting the power ON button. Confirm the magnets are stuck to the weight canisters.
5. Select which direction the weight should pull by clicking on the appropriate button. The button should change color, showing it has been selected. The graphic image on the top right should update to give an illustration of what will happen when the magnets release. The random option will randomly select the left or right weight without informing the user.
6. Select the delay time desired.
7. Click release. The instructions should be sent to the Arduino and executed with the selected delay.



## Electromagnet Weights and Housing

Robert Regent

### Background

The weights used for the Lateral Perturbation Attachment cause a force to be applied to the test subject in the opposite direction than the weight which was released. The electromagnetic weight housing is composed of 3" ABS piping at a 7" length with flat end caps attached to each end of the pipe. The bottom end cap is glued to the pipe and is very secure. The top cap and the top of the pipe each have matching holes drilled through them and are secured together by a 6" 1/4x20 bolt. Matching nuts are applied to each end of the bolt to keep it in place. Two complete assemblies of the electromagnetic weights are included with "Twister".

### Procedure

To prep "Twister" for a lateral perturbation test, place the desired amount of weight in each housing. The weights from each side of the perturbation attachment should match one another to prevent injury to the test subject. The base weight with an empty housing is 1.5 lbs., with a maximum total weight allowable for each side of 15 lbs. Once the desired amount of weight is added to each side, secure the top cap to each canister. Make sure to line the holes on the top cap and canister up. Next, slide the 6" bolt through the canister and top cap and secure both sides. Once the weight housing is secured it is ready to be attached to the electromagnets.

### Design

The weight assemblies are attracted to the pulley system on "Twister" through electromagnets. On the top of each weights housing is a 1.5"x1.5"x0.125" piece of steel, which is used as a magnet to attract to the electromagnets. The electromagnets are secured to the pulley system, and when the user selects a side to produce a lateral force on the test subject, power is turned off to said electromagnet, and the weight assembly is release and falls to the ground. This causes a lateral force in the opposite direction of the weight which was released.



Figure 15 Electromagnetic Weights Assembly

The weight can be adjusted to match the desired weight test percentages. A percentage of 7% percent was used during testing, which would be 14 lbs. for a 200 lb. person. A maximum weight of 15 lbs. can be applied to each side of the test subject. The housing has a standalone weight of 1.5 lbs., and weight can be incremented in ½ lb. increments. Figure 2 shows all components included in the magnet assembly. The kit includes (1) 5 lb. weight, (4) 1 lb. weights, and (1) ½ lb. weight. The weights are made from 3” cylindrical steel stock and are themselves magnetic, but do not generate a large enough magnetic field to produce a solid connection with the electromagnets.



Figure 16 All included components of Electromagnet Housing

---

# ELECTRICAL/SOFTWARE

---



## Using the Graphical User Interface

Jacquelin Remaley

The Twister is controlled through a user interface designed specifically to send signals to the system through the computer's serial port. The executable can be placed anywhere on the computer that is accessible.

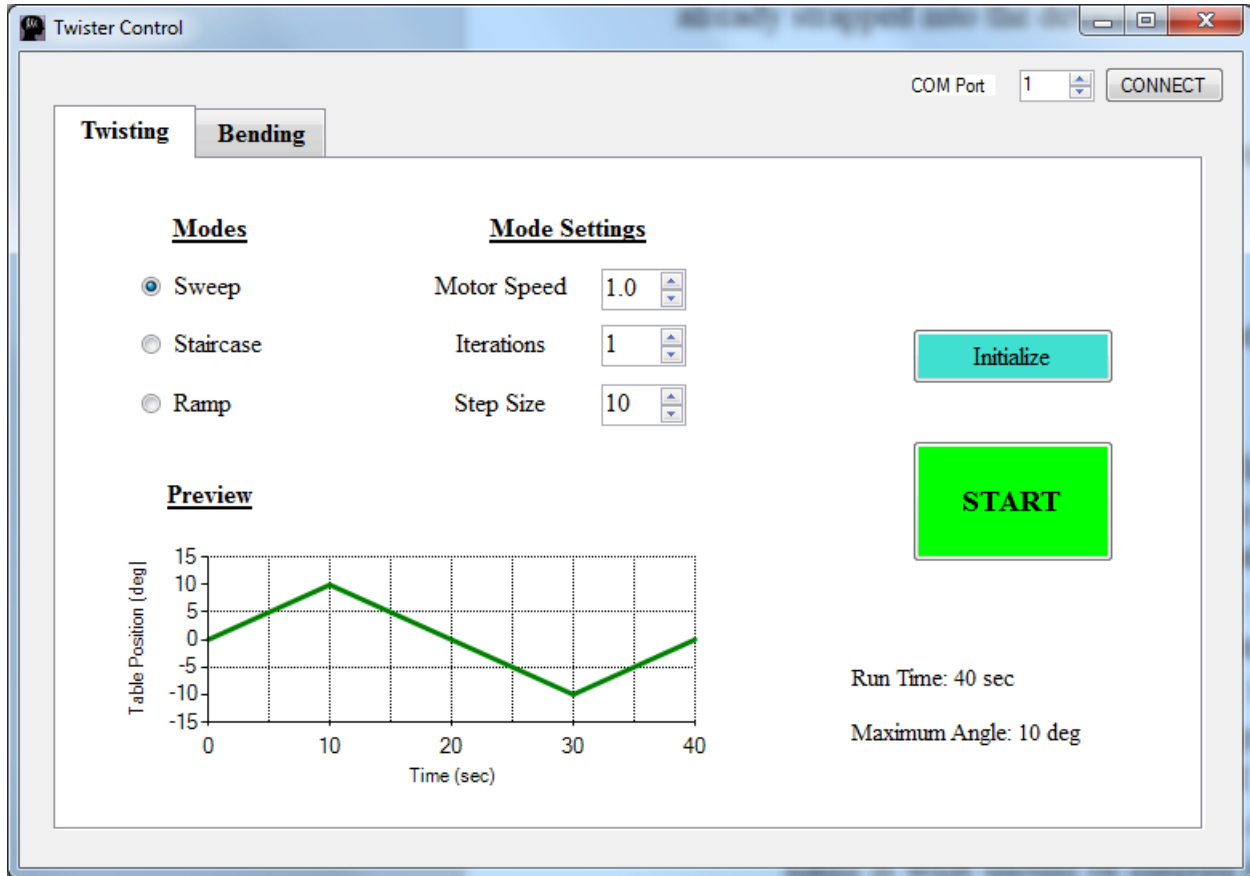
Using the interface is split into two categories: the original Twister and the Axial Force addition. This guide will describe the basic operation of the interface assuming the subject is ready to be (but not already) strapped into the device and the user has their test planned out.

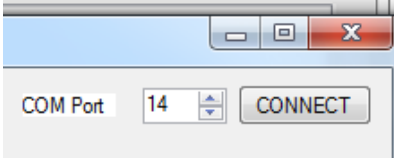
Some basic notes about the Interface:

- It opens to the Twisting function by default.
- There is a built-in safety which attempts communication with the Arduino any time either tab is selected to switch from twisting to bending. If the Arduino is not connected, an error box will appear. This has no immediate implications, but if testing is desired, you should connect the Arduino before continuing.
- The preview window shown on the Twisting function controls is a very ideal estimation. The actual path the motor takes will vary. Similarly, the estimated time and maximum angle shown are also ideal, and should not be taken into account for real measurements.

## Connecting to the Arduino

1. Open the Interface. It should open to the Twisting function by default and look like the image below.



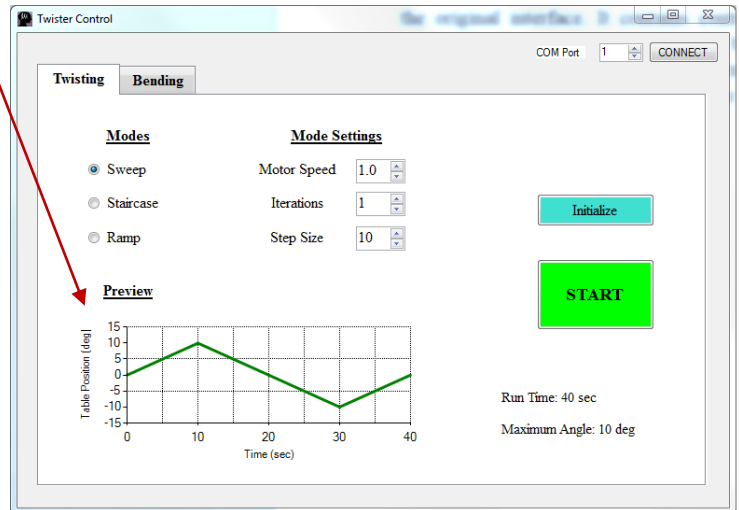
2. The controls for connecting to the Arduino are in the top right hand corner. Make sure the Arduino is connected to the computer (lights on the board should be blinking or steadily on) and use the numeric up/down selector to pick the COM port the Arduino is connected to.
- 
- The close-up image shows the 'COM Port' dropdown menu with the value '14' selected and the 'CONNECT' button next to it.
3. If you do not know which port the Arduino is connected to, open the start menu, in the search bar, type “Device Manager,” and press enter. In the Device Manager window that opens, expand the “Ports (COM & LPT)” section. If the Arduino is connected, the board should be listed under here with a name similar to “COM#.” The number at the end of this name is what should be entered into the Twister Interface.
  4. Click connect. If successful, a message box will appear with the text “Port Connected.” You can now send the Twister instructions.
  5. If unsuccessful, a message box with the text “Error! No serial port detected” will appear. If this is the case, try these troubleshooting steps:

- a. Double check the Arduino cable is connected.
  - b. Check which COM port the Arduino is connected to.
  - c. Try reconnecting the cable. Sometimes cycling the power to the Arduino board can help.
  - d. Try restarting the Twister Control executable.
6. Now that the Arduino is connected, you may proceed with testing. This process can be repeated at any time before executing a test.

## Using the Twisting portion of the Interface

The portion of the interface that opens by default is the Twisting section, which is very similar to the original interface. It contains controls for the motor speed, iterations of movement, and step size of movement, as well as the three original movement types. The speed, iterations, and step size may mean something different depending on the movement mode selected, so be sure to look at the preview window to ensure the predicted path is what is desired.

1. Make sure the COM port is connected. If it is not, refer to the instructions on the previous page.
2. Select the desired Mode. Examples of each mode are displayed in the Preview Window at the bottom. The run time and maximum angle are also displayed for convenience, but these are very ideal estimations and should not be considered realistic for use in measurements.
  - a. If selecting Sweep, click off of the sweep selection and back again. Sometimes the interface does not register it has been selected if starting a new session.



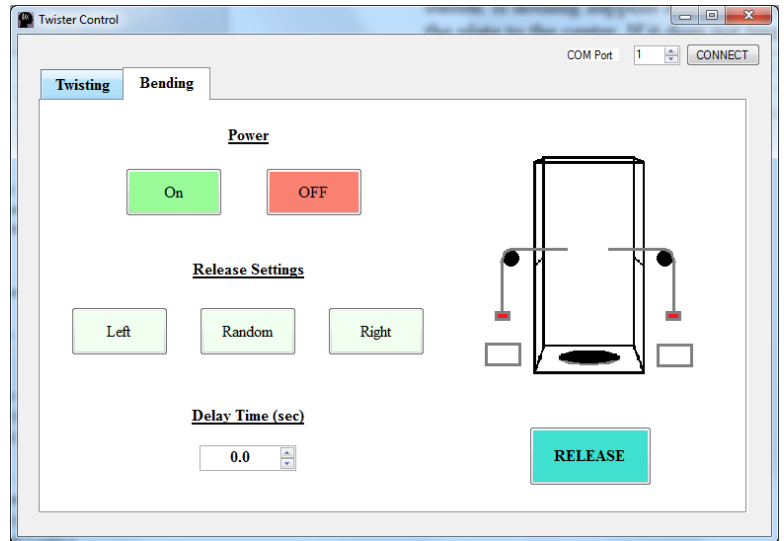
3. Also change the mode settings, which include the motor speed, iterations, and step size. These parameters mean different things for each mode, so be sure to observe the Preview Window to view what changes are made. A small description of each parameter will appear in the white space below the step size counter for convenience.
4. **Important: Before running the experiment, initialize the motor.** Click the Initialize button. If nothing happens immediately, click it again. This will instruct the motor to turn the plate to the center. If it does not turn after the second attempt, it is most likely already in the center.
5. If all the parameters are correct, you are ready to test. Strap the subject into the Twister, double check everything is as you wish, and click the START button. At this point, the Interface cannot communicate with the Twister until the motion is complete. If the motion must be interrupted, use the E-STOP mounted on the Twister frame.

## Using the Axial Force portion of the Interface

This is the addition to the old Twister control. It sends signals to the electromagnets which hold the weights used in the application of axial force.

1. Make sure the COM port is connected. If it is not, refer to the instructions outlining how to do so.
2. Place the appropriate weights into the canisters and secure the lids.

3. Place the canisters directly below the electromagnets. Allow the magnets to touch the metal on top of the canisters.
4. Remove any sensitive electronics (cell phones, cameras, etc.) from the immediate vicinity of the magnets. Anything further than a few feet should be fine.
5. Select the Power ON button. The magnets should now be attracted to the canisters and attached. Ensure this by gently pulling on the magnet.
6. Select which direction you would like the subject to be pulled. You may select Left, Right, or Random. Each is accompanied by a basic graphic showing what will physically happen when the magnets are released. If Random is selected, the user will not be informed which magnet will drop; the system decides that internally.
7. Select a delay time. There is a maximum of 10 and a minimum of zero seconds that the system can implement in increments of 0.1 seconds.
8. Click RELEASE. The system will wait the given delay time, then shut down the appropriate magnet, dropping the weight.
9. After the subject's movement has been recorded, click the Power OFF button to shut down the power to the electromagnets. Do not leave them on for an extended amount of time, or they may overheat. If you attempt to leave the Bending portion of the Interface, the magnets will shut off as a safety.



## Electromagnet Code Notes

Jacquelin Remaley

This section outlines the basic processes for the code that controls the Electromagnets used in the Axial Force application.

When instructions are sent to the Arduino, it is in the form of five numbers:

[A B C D E]

A: Electromagnets(1) or Motor(2)

EM: B: Power On(1)/Off(2)

C: Left(1) or Right(2) or Random(3) or Both on(4)

D: Delay time (seconds\*10) (serial communications don't like tenths)

Motor: B: Initialize(1) sweep(2) Staircase(3) ramp(4)

C: Speed

D: Frequency

E: Amplitude

As an example, if the numbers read [1,1,2,50], the system will power both magnets on and drop the right one after 5 seconds.

The code waits to receive the data string sent by the interface when the RELEASE or START buttons are selected. If the first number is a 1, it immediately jumps to the electromagnet portion and reads the second number. It decides based on the following numbers which magnet to drop and after how long.

If the random option was selected, the code will select a random number between 1 and 2 and continue the instructions from there.

# Motor Code

Austin Steiner

## Background

Because the old motor system was replaced it was also necessary to come up with a cleaner, more modern and more efficient way of controlling the turn table's path. The existing system utilized several feedback loops and many microcontrollers to synthesize and monitor outputs from two rotary encoders. All of this was very complex, bulky and was out of date. Through research it was discovered that almost all of the processing could be done by a single Arduino Mega rather than the mess that existed before.



Figure 1. Old Control Electronics



Figure 2. Arduino Mega 2560

## Design

Refer to the provided appendix for the full Arduino motor code.

## Improvements

Because none of the members on the team are computer software engineers there is definitely a large area for improvement in the coding. As-written, the code does what it is supposed to but there are more than likely many ways to accomplish the same tasks in a much simpler and more streamlined manner.

# Power Supply

Austin Steiner

## Background

When the computer system was upgraded the old motor control system was no longer able to be used. One component of the old control box was a 24 volt dc power supply along with its accompanying distribution boards. Without knowing what each of these boards did and the output specs it was impossible to reuse the existing technology even though it worked fine. Thus a new power supply was sourced.

## Design

The new system uses an Omron AC to DC conversion power block. This model is rated at 24 volts 100 watts continuous output and features an adjustable output for consistent voltage readings. All of the inputs and outputs are screw terminals which makes modular use much easier because cords can be removed at will.

## Procedure

To power the system a three wire wall plug lead is necessary. The positive is connected to the L terminal and the negative or neutral is connected to the N. An earth ground connection is provided for safety and connects to the ground pin. On the supply side there are three output posts for both positive and negative. However many connections can be attached to one screw block.

Refer to the provided user manual for more information on the operation and features of the power supply module.

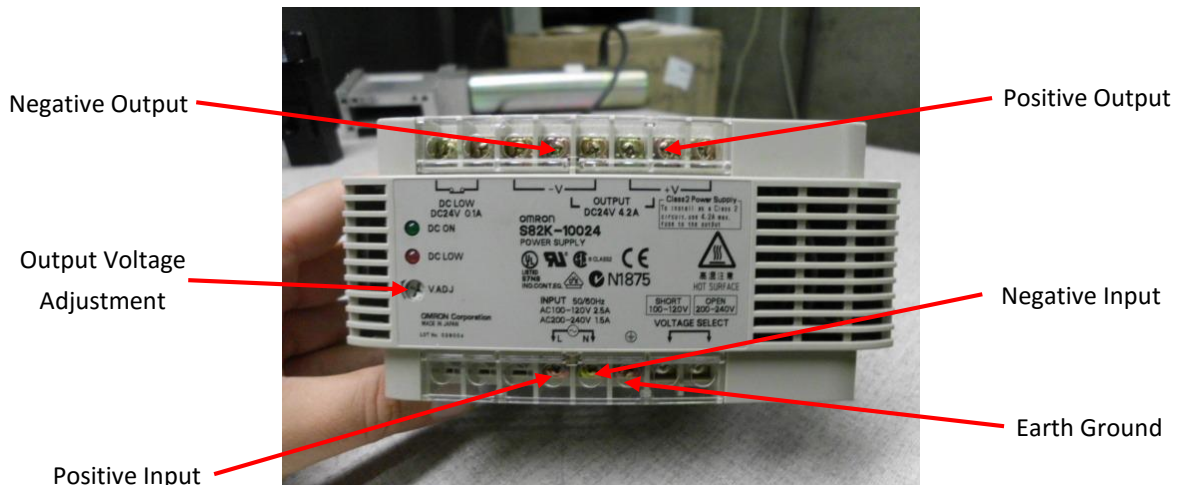


Figure 1. Power supply

## Improvements

Because this system is designed to work together from the manufacturer it does not need any more improvements.



## Torque Sensor and Voltage Regulator Circuit

Robert Regent

### Background

Torque sensors are used to obtain torsional data from the test subject during the turntable operation of “Twister”. Data is fed from “Twister” to an A/D converter, which creates a link from “Twister” to the Motion Monitor camera system. This creates the possibility of showing sensor data and movement data side by side in the Motion Monitor software. The torque sensors used in “Twister” are the [Futek TFF400](#). The sensor has +Excitation, -Excitation, +Signal, and -Signal pins to power and communicate with an output device. “Twister” uses a BNC connector to connect to the A/D converter, which posed a connectivity problem. A BNC connector has one ground pin and one signal pin, and two signal pins are outputting from the torque sensor. A circuit was created to merge the two torque sensor signal outputs into a single output. An operational amplifier, [Analog Devices AD622](#), was used to merge the torque sensor signals into one signal.

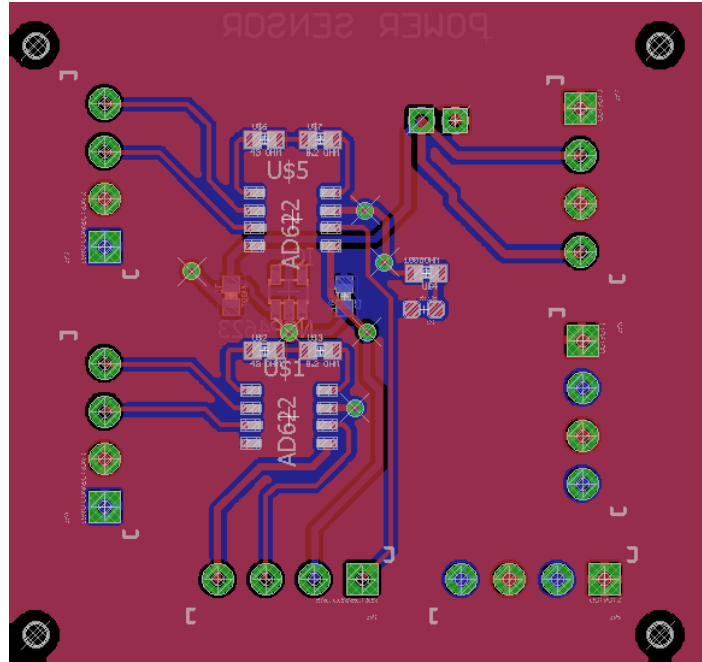
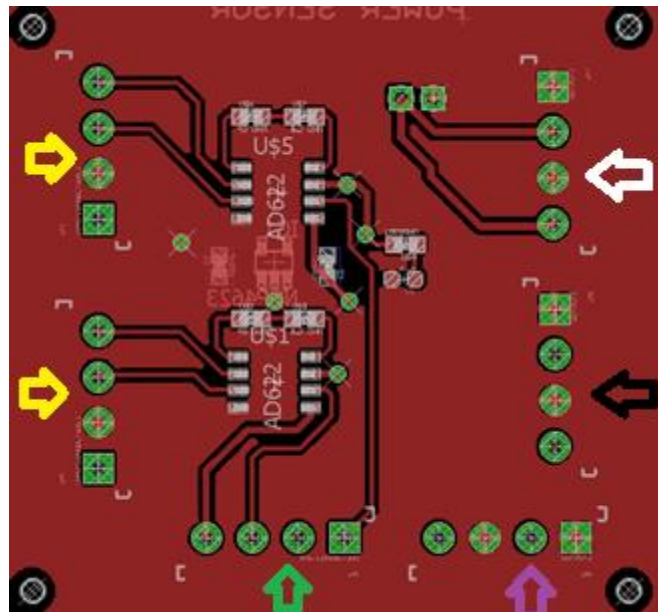


Figure 17 Completed Circuit Board Design

### Procedure

Figure 2 shows the top layer of the circuit board designed for the AD622 and NCP4623 chips. The top layer has an op-amp for each torque sensor. A gain value of 52.2 ohms was used to get the correct frequency signal to Motion Monitor. Shown with yellow arrows to the right are the headers for each torque sensor. The torque sensor has a color scheme for each pinout and the wires in the connecting cable will match the color scheme. The color scheme is shown on each torque sensor. Looking at the figure to the right and starting from the top, the header inputs are as follows:

1. +Excitation
2. -Excitation (ground)
3. +Signal
4. -Signal



The header signified by the green arrow is the BNC output header. Starting from the left and going right, the header outputs are as follows:

1. BNC GND1
2. BNC Signal1
3. BNC GND2
4. BNC Signal2

The purple and black arrows shows 5V and ground outputs for powering sensors and devices that run off of a 5V voltage. The white arrow shows a 24V and ground output for powering any devices needing 24 volts. The 2-pin header at the top of the board is the power connection for the circuit to the battery. The left pin is the 24V pin and the right pin is the ground pin. Either 24V headers can be used as an input to the circuit. Connections should all be made to said headers.

## Design

As stated in the Procedure section, the top layer of the circuit contains all headers. The goal of designing this board was to combine the torque sensor circuits with the voltage regulator circuit since one powers the other, while allowing for external connections for the rest of Twister's components. For any details about the top layer, please refer to the Procedure section.

Figure 3 shows the bottom of the circuit, which holds the voltage regulator. 10uF ceramic capacitors are used with the circuit on the VIN and VOUT pins. Once power is connected, a green light will shine from the top of the board. The voltage regulator has an SOT-23-5 footprint, and is rated at 150 mA. The op-amp has a footprint of SOIC-8. Please refer to the data sheets for specific information on the components used in this design.

All design files are included on the Wiki page for this project for future reference and modifications.

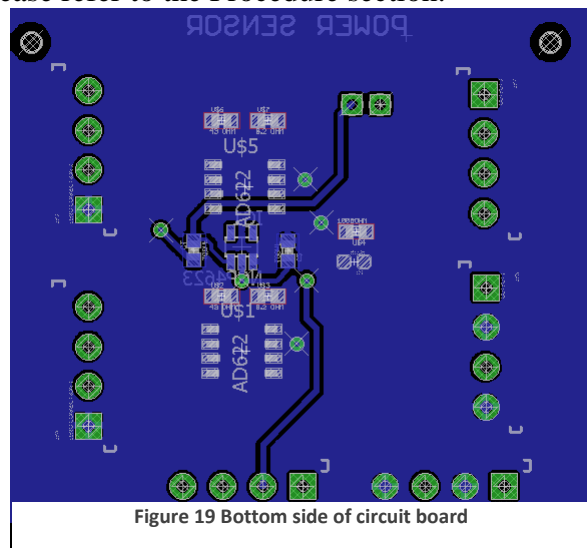


Figure 19 Bottom side of circuit board

## Improvements

The torque sensor and voltage regulator circuit had one main issue with it. During testing of the voltage regulator, a 9V battery was used to test the connection and voltage outputs from the headers. At the 5V output header, 5 volts was consistently streaming out of the header. A stress test was conducted and the circuit continued powering a 5V external device for 15 minutes, so an assumption was taken that since the voltage regulator was rated at 24 volts, there would not be an issue with the main power source. A stress test was later conducted with the 24V power source. The top layer of the circuit board contains a green LED light connected off the 5V output line. Once the 24V source was plugged in, the light illuminated for 35 seconds and suddenly shut off. The values of the capacitors on the bottom layer were taken off the datasheet for the voltage regulator, and all connections on all layers of the circuit board have been checked by a 3<sup>rd</sup> party and are correct, so it is unknown what the issue is. The issue may lay with the power source itself.

All 5V devices connected to the circuit pull a minimal amount of amperage from the source, so a temporary solution is using the 5V pin off the microcontroller to power the 5V devices. An external power source can also be used, but is not as convenient. The torque sensor portion of the circuit is operational and in working condition.

For the short term, the voltage regulator has been replaced with a non-isolated DC/DC converter, specifically, a [Murata OKI-78SR-5/1.5/1.5-W36-C](#). For long term use, the chipset used in the Murata converter will want to be broken out onto its own chip with the torque sensor operational amplifiers.

## Solid State Relay

Robert Regent

### Background

The electromagnets operate at 24V and receive a 5V signal from the microcontroller to turn on and off. This created a problem because the electromagnets could not run directly off the microcontroller. To alleviate this problem, a solid state relay was used as an intermediary between the 24V power supply, the microcontroller, and the electromagnets.

A solid state relay (SSR) is an electronic switching devices with the main purpose of acting as a mechanical relay to control high-voltage and low-voltage supplies/signals. The main advantages they have over mechanical relays is that they are much faster and that they can be triggered by much lower voltages.

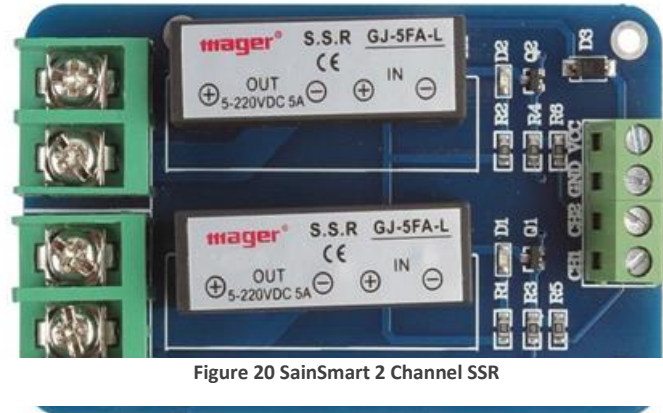


Figure 20 SainSmart 2 Channel SSR

### Procedure

The small header on the right of Figure 1 is labeled VCC, GND, CH2, CH1, from the top going down. The connections to these pins are as follows:

1. VCC → 5V connection to Arduino
2. GND → Ground connection to microcontroller
3. Ch1 → Digital pin on microcontroller that controls electromagnet 1
4. Ch2 → Digital pin on microcontroller that controls electromagnet 2

The large headers shown on the left side of Figure one are the power input and output headers for the electromagnets. From the top header going down, the connections are as follows:

1. Header 1 → Electromagnet 1 (+)
2. Header 2 → 24 V power input (+)
3. Header 3 → Electromagnet 2 (+)
4. Header 4 → 24V power input (+)

\*\*\*Connect ground pins from electromagnets to the ground directly on the power source.

---

# APPENDIX

---

Arduino Motor Code

Visual Studio User Interface C# Code

# Arduino Motor Code

```
// Build Date 4/11/16

// The 5 values (in order) that this code requires to
work are
// 1) Control (1 for EM 2 for Motor)
// 2) Function (1 for initialize 2 for ramp 3 for
Staircase 4 for Step)
// 3) Speed (in degrees/sec) (must be an integer
from 1-3 for now)
// 4) Number of repetitions (any integer)
// 5) Step size (integer from 0-60)

int intPin = 7;
int stopPin = 6;
int speedPin = 10;
int directionPin = 3;
int currentPin = 4;
int right = 24;
int left = 26;

void setup()
{
  // set each pin to output
  pinMode(intPin, OUTPUT);
  pinMode(stopPin, OUTPUT);
  pinMode(speedPin, OUTPUT);
  pinMode(directionPin, OUTPUT);
  pinMode(currentPin, OUTPUT);

  pinMode(right, OUTPUT);
  pinMode(left, OUTPUT);

  // begin serial operation
  Serial.begin(9600);

  // set all pin voltages to zero start value
  analogWrite(intPin, 0);
  digitalWrite(stopPin, LOW);
  digitalWrite(directionPin, LOW);

  analogWrite(currentPin, 0);
}
////////////////////////////////////
//Initialization Function
////////////////////////////////////

void InitFunction(int a) // a=speed

{
  analogWrite(intPin, 255);
  digitalWrite(stopPin, LOW);
  digitalWrite(directionPin, LOW);
  analogWrite(currentPin, 10);
  Serial.println("Initializing");

  // map to convert input degrees/sec to rpm
  int speedrpm = map(a, .15, 4.5, 125, 3750);
  // map to convert input rpm to pwm signal
  int speed = map(speedrpm, 125, 3750, 0, 255);

  // constrain values from 0 to 255 and apply
  speed = constrain(speed, 0, 255);
  analogWrite(speedPin, speed);
  Serial.println("Speed Sent");
  Serial.println(speed);

  // Begin direction control scheme
  int position = analogRead(A0);

  if (position < 530)
  {
    Serial.println("< C");
    for (position=analogRead(A0); position < 530;)
    {
      digitalWrite(directionPin,LOW);
      position=analogRead(A0);
    }
  }
}
```

```

else
{
  Serial.println("> C");
  for (position=analogRead(A0); position > 530;)
  {
    digitalWrite(directionPin,HIGH);
    position=analogRead(A0);
  }
}

//exists the function and goes back to the loop
return;
}

////////////////////////////////////
//RAMP FUNCTION
////////////////////////////////////

void RampFunction(int a,int b, int c) // a=speed
b=reps c=step size

{
  // This section of the code sets all of the initial
  conditions
  // for the chosen process and starts the table
  rotating clockwise

  analogWrite(intPin, 255);
  digitalWrite(stopPin, LOW);
  digitalWrite(directionPin, HIGH);
  analogWrite(currentPin, 10);
  Serial.println("Ramp Function Executed");

  // map to convert input degrees/sec to rpm
  int speedrpm = map(a, .15, 4.5, 125, 3750);
  // map to convert input rpm to pwm signal
  int speed = map(speedrpm, 125, 3750, 0, 255);

  // constrain values from 0 to 255 and apply
  speed = constrain(speed, 0, 255);
  analogWrite(speedPin, speed);
  Serial.println("Speed Sent");

  // Begin direction control scheme
  c=c*.277777;

  int direction = 1;

  for (int count=1; count <= b;)
  {
    int position = analogRead(A0);

    if (position>530+c)
    {
      int j=analogRead(A0);
      Serial.println(j);
      Serial.println("pos>c");

      if (direction==1)
      {
        digitalWrite(directionPin,LOW);
        Serial.println("direction=1 switch to ccw");
        direction=2;
        delay(1500);
      }

      else
      {
        digitalWrite(directionPin,HIGH);
        Serial.println("direction=2 switch to cw");
        direction=1;
        delay(1500);
      }
      count++;
    }

    else if (position<530-c)
    {
      int j=analogRead(A0);
      Serial.println(j);
      Serial.println("pos<c");

      if (direction==1)
      {

```

```

Serial.println("direction=1 switch to ccw");
digitalWrite(directionPin,LOW);
direction=2;
delay(1500);
}

else
{
Serial.println("direction=2 switch to cw");
digitalWrite(directionPin,HIGH);
direction=1;
delay(1500);
}
count++;
}
}
//exists the function and goes back to the loop
return;
}

////////////////////////////////////
//STAIRCASE FUNCTION
////////////////////////////////////

void StaircaseFunction(int a,int b, int c) // a=speed
b=reps c=step size

{
// This section of the code sets all of the initial
conditions
// for the chosen process and starts the table
rotating clockwise

analogWrite(intPin, 255);
digitalWrite(stopPin, LOW);
digitalWrite(directionPin, HIGH);
analogWrite(currentPin, 10);
Serial.println("Staircase Function Executed");

// map to convert input degrees/sec to rpm
int speedrpm = map(a, .15, 4.5, 125, 3750);
// map to convert input rpm to pwm signal

int speed = map(speedrpm, 125, 3750, 0, 255);

// constrain values from 0 to 255 and apply
speed = constrain(speed, 0, 255);
analogWrite(speedPin, speed);
Serial.println("Speed Sent");

// Begin direction control scheme
c=c*.277777;
int d=c;

for (int count=1; count <= b;)
{
int position = analogRead(A0);

if (position>(530-c))
{

}

else if (position==(530-c))
{
Serial.println("==530-c");
Serial.println(position);
Serial.println(530-c);
analogWrite(intPin, 0);
count++;
c=d+c;
delay(3000);
analogWrite(intPin, 255);
}

else
{
Serial.println("else");
analogWrite(intPin, 0);
count++;
c=d+c;
delay(3000);
analogWrite(intPin, 255);
}

}

digitalWrite(directionPin, LOW);

```



```

c=c-(2*d);
Serial.println("2nd phase");

for (int count=1; count <= 2*b;)
{
  int position = analogRead(A0);

  if (position<530-c)
  {

  }
  else if (position==530-c)
  {
    Serial.println("==530-c");
    Serial.println(position);
    Serial.println(530-c);
    analogWrite(intPin, 0);
    count++;
    c=c-d;
    delay(1000);
    analogWrite(intPin, 255);
  }
  else
  {
    Serial.println("else");
    Serial.println(position);
    Serial.println(530-c);
    analogWrite(intPin, 0);
    count++;
    c=c-d;
    delay(1000);
    analogWrite(intPin, 255);
  }
}

digitalWrite(directionPin, HIGH);
c=c+(2*d);
Serial.println("Third Phase");

for (int count=1; count <= b;)
{
  int position = analogRead(A0);

  if (position>530-c)
  {

  }
  else if (position==530-c)
  {
    Serial.println("==530-c");
    Serial.println(position);
    analogWrite(intPin, 0);
    count++;
    c=c+d;
    delay(2000);
    analogWrite(intPin, 255);
  }
  else
  {
    Serial.println("else");
    Serial.println(position);
    analogWrite(intPin, 0);
    count++;
    c=c+d;
    delay(2000);
    analogWrite(intPin, 255);
  }
}
analogWrite(intPin, 0);
//exists the function and goes back to the loop
return;
}

////////////////////////////////////
//STEP FUNCTION
////////////////////////////////////
void RotationFunction(int c) // c=step size
{
  Serial.println("Rotation Function");
  for (int count=1; count <= 4;)
  {
    int position = analogRead(A0);

    if (position>=530+c)

```

```

{
  Serial.println(">530+c");
  Serial.println(position);
  digitalWrite(directionPin,HIGH);
  count++;
  delay(2000);
}
else if (position<=530-c)
{
  Serial.println("<530-c");
  Serial.println(position);
  digitalWrite(directionPin,LOW);
  count++;
  delay(2000);
}

else
{

}

}
//exists the function and goes back to the loop
return;
}
void StepFunction(int a,int b,int c) // a=speed
b=reps c=step size

{
  // This section of the code sets all of the initial
  conditions
  // for the chosen process and starts the table
  rotating clockwise

  analogWrite(intPin, 255);
  digitalWrite(stopPin, LOW);
  digitalWrite(directionPin, HIGH);
  analogWrite(currentPin, 10);
  Serial.println("Step Function Executed");

  // map to convert input degrees/sec to rpm
  int speedrpm = map(a, .15, 4.5, 125, 3750);

  // map to convert input rpm to pwm signal
  int speed = map(speedrpm, 125, 3750, 0, 255);

  // constrain values from 0 to 255 and apply
  speed = constrain(speed, 0, 255);
  analogWrite(speedPin, speed);
  Serial.println("Speed Sent");

  // Begin direction control scheme
  int direction = 1;
  c=c*.27777;

  for (int count=1; count <= b;)
  {
    RotationFunction(c);
    c=2*c;
    count++;
    Serial.println("rep");
  }

  InitFunction(a);

  //exists the function and goes back to the loop
  return;
}
////////////////////////////////////
//MAIN LOOP
////////////////////////////////////

void loop()
{
  Serial.println("HELLO");
  Serial.println("Enter 6 Control Values Separated by
  Commas");
  Serial.println("1: Control Mode: 1=Electromagnet
  2=Motor");
  Serial.println("2: Function Type: 1=Ramp
  2=Staircase 3=Step");
  Serial.println("3: Speed (deg/sec) !Integer from 1-
  3!");
  Serial.println("4: Number of periods !Integer from
  1-50!");
}

```



```

RampFunction(a,b,c);
InitFunction(a);
analogWrite(intPin, 0);

}
else if (function == 3) //Use Staircase Function
{
  Serial.println("Staircase Function Executed");
  StaircaseFunction(a,b,c);
  analogWrite(intPin, 0);
}
else if (function == 4) //Use Step Function
{
  StepFunction(a,b,c);
  analogWrite(intPin, 0);
}
}
else
{
}
}
else if (control==0)
{
}
else
{
}
}
}

```

## Visual Studio User Interface C# Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
//using Graph = System.Windows.Forms.DataVisualization.Charting;

//latest edit date: 4/18/16
// 4/19/16 - added comments and (hopefully) fixed index offsets with motor mode

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        private SerialPort myport;

        // Creates an empty data array for filling with data.
        // The data that goes into each slot is described below.
        string[] data = new string[5];

        // data string to be sent to Arduino.
        // takes form [A B C D E]
        //      A: Electromagnets(1) or Motor(2)
        // EM:   B: Power On(1)/Off(2)
        //      C: Left(1) or Right(2) or Random(3) or Both on(4)
        //      D: Delay time (seconds*10)
        // Motor: B: Initialize(1) sweep(2) Staircase(3) ramp(4)
        //      C: Speed
        //      D: Frequency
        //      E: Amplitude

        public static void GetCOMS() // find com port names. Not currently used.
        {
            string[] ports = SerialPort.GetPortNames();
        }

        public Form1() //Initialize form
        {
            InitializeComponent();
            datainit();
            tabControl1.Selected += new TabControlEventHandler(tabControl1_Selected);

            sweepPreview();
        }

        //initialization
    }
}
```

```

private void init() //tries to connect to the selected port. Throws error if
failed.
{
    try
    {
        string COMport = Convert.ToString(COMPort.Value);

        myport = new SerialPort();
        myport.BaudRate = 9600;
        myport.PortName = "COM"+COMport;
        myport.Open();
        //MessageBox.Show(myport.PortName);
    }
    catch (Exception)
    {
        MessageBox.Show("Error! No serial connection detected.");
    }
}
public void datainit() //Runs at beginning of code. Sets defaults in case nothing
is changed.
{
    //Sets defaults for motor motion
    data[0] = "2";
    data[1] = "2";
    data[2] = Convert.ToString(Motor_Speed.Value);
    data[3] = Convert.ToString(Step_Frequency.Value);
    data[4] = Convert.ToString(Amplitude.Value);

    randomPanel.Visible = false;
    rightPanel.Visible = false;
    leftPanel.Visible = false;
    centerPanel.Visible = false;
    offPanel.Visible = true;
}

//Preview window things
public void sweepPreview() // Generates sweep path for motor path preview window
{
    decimal s = (Amplitude.Value);
    decimal n = Step_Frequency.Value;
    decimal m = Motor_Speed.Value;
    double[] x = new double[4 * Convert.ToInt32(n) + 2];
    double[] y = new double[4 * Convert.ToInt32(n) + 2];
    //MessageBox.Show(Convert.ToString(x.Length));
    decimal t_sweep = s / m;
    //double x;
    //double y;
    chart1.Series["Preview"].Points.Clear();

    for (int i = 0; i <= 4 * n + 1; i++)
    {
        x[i]= (i) * Convert.ToDouble(t_sweep);
    }
    for (int i = 1; i <= n; i++)
    {
        y[4 * (i) - 3] = Convert.ToDouble(s);
        y[4 * (i) - 1] = Convert.ToDouble(-s);
    }
}

```

```

        //x[0] = 0;
        //y[0] = 0;

        for (int i = 0; i < x.Length - 1; i++)
        {
            chart1.Series["Preview"].Points.AddXY(x[i], y[i]);
        }
        //chart1.ChartAreas[0].AxisX.Maximum = x[x.Length - 1] +
Convert.ToDouble(t_sweep);
        decimal xMax = 4*t_sweep*n;
        decimal yMax = s;

        runParam.Text = "Run Time: " + xMax.ToString("#.#") + " sec" +
"\r\n\r\nMaximum Angle: " + yMax.ToString("#.#") + " deg";
    }
    public void stairPreview() // Generates stair path for motor path preview window
    {
        decimal s = (Amplitude.Value);
        decimal n = Step_Frequency.Value;
        decimal m = Motor_Speed.Value;
        double[] b = new double[8 * Convert.ToInt32(n) + 2];
        double[] y = new double[8 * Convert.ToInt32(n) + 2];
        //MessageBox.Show(Convert.ToString(x.Length));
        decimal t_stair = s / m;
        //double x;
        //double y;
        chart1.Series["Preview"].Points.Clear();

        for (int i = 0; i <= 8 * n + 1; i++)
        {
            b[i] = (i) * Convert.ToDouble(t_stair);
        }

        double[] x = new double[b.Length - 1];
        Array.Copy(b, 0, x, 0, b.Length - 1);

        double[] z = new double[2 * Convert.ToInt32(n) + 1];

        for (int i = 0; i <= z.Length - 1; i++)
        {
            z[i] = Convert.ToDouble(s) * Math.Ceiling(i*0.5);
        }

        Array.Copy(z, 0, y, 0, 2 * Convert.ToInt32(n));
        Array.Copy(y, 0, y, 2 * Convert.ToInt32(n), 2 * Convert.ToInt32(n));

        for (int i = 2 * Convert.ToInt32(n); i <= 4 * Convert.ToInt32(n); i++)
        {
            y[i] = -y[i] + Convert.ToInt32(n*s);
        }

        for (int i = 4 * Convert.ToInt32(n); i <= 8 * Convert.ToInt32(n); i++)
        {
            y[i] = -y[i - 4 * Convert.ToInt32(n)];
        }
    }

```

```

    for (int i = 0; i < x.Length - 1; i++)
    {
        chart1.Series["Preview"].Points.AddXY(x[i], y[i]);
    }

    decimal xMax = ((8*n)-1) * t_stair;
    decimal yMax = n * s;

    runParam.Text = "Run Time: " + xMax.ToString("#.#") + " sec" +
    "\r\n\r\nMaximum Angle: " + yMax.ToString("#.#") + " deg";

    //MessageBox.Show(Convert.ToString(x.Length) + ", " +
    Convert.ToString(y.Length));
}
public void rampPreview() // Generates ramp path for motor path preview window
{
    decimal s = (Amplitude.Value);
    decimal n = Step_Frequency.Value;
    decimal m = Motor_Speed.Value;
    double[] x = new double[8 * Convert.ToInt32(n) + 3];
    double[] y = new double[8 * Convert.ToInt32(n) + 3];
    //MessageBox.Show(Convert.ToString(x.Length));
    decimal t_ramp = s / m;
    //double x;
    //double y;
    chart1.Series["Preview"].Points.Clear();

    for (int i = 0; i <= 7; i++)
    {
        x[i] = (i) * Convert.ToDouble(t_ramp);
    }
    for (int i = 1; i <= 2; i++)
    {
        y[4 * (i) - 3] = Convert.ToDouble(s);
        y[4 * (i) - 1] = Convert.ToDouble(-s);
    }

    if (n >= 2)
    {
        for (int i = 2; i <= n; i++)
        {
            //Array.Copy(x, 0, x, i * 8 - 8, 8);
            //MessageBox.Show(Convert.ToString(2^(-1)));
            Array.Copy(y, 0, y, i * 8 - 8, 8);

            for (int j = i*8-8; j <= i*8-1; j++)
            {
                //x[j - 0] = i * x[j - 0] + Convert.ToDouble(t_ramp * (4 * ((i -
                1) ^ 2) + (4 * (i - 1))));
                x[j] = (i) * x[j - (i - 1)*8] + Convert.ToDouble(t_ramp) * (4 *
                Math.Pow(i - 1, 2) + (4 * (i - 1)));
                y[j] = (i) * y[j];
            }
        }
    }
}

```



```

    }

    x[(8 * Convert.ToInt32(n))] = Convert.ToDouble(n) * x[0] +
Convert.ToDouble(t_ramp * (4 * n * n + 4 * n));
    y[(8 * Convert.ToInt32(n))] = 0;

    for (int i = 0; i < x.Length - 2; i++)
    {
        chart1.Series["Preview"].Points.AddXY(x[i], y[i]);
    }

    decimal xMax = Convert.ToDecimal(x[(8 * Convert.ToInt32(n))]);
//    decimal xMax = Convert.ToDecimal((4 * (n + 1) * (n + 1)) + (4 * (n + 1)) *
t_ramp);
    decimal yMax = n * s;

    runParam.Text = "Run Time: " + xMax.ToString("#.#") + " sec" +
"\r\n\r\nMaximum Angle: " + yMax.ToString("#.#") + " deg";

    //MessageBox.Show(Convert.ToString(x.Length) + ", " +
Convert.ToString(y.Length));
}

//start and stop
private void Start_Main_Click(object sender, EventArgs e) // sets data values and
sends data array to Arduino
{
    //myport.WriteLine("300,"+Motor_Speed.Value);

    if (tabControl1.SelectedTab == tabPage1) //motor
    {
        data[0] = "2";

        if (Sweep_Mode.Checked == true)
        {
            data[1] = "2"; //set mode to sweep
        }
        else if (Staircase_Mode.Checked == true)
        {
            data[1] = "3"; //set mode to staircase
        }
        else if (Ramp_Mode.Checked == true)
        {
            data[1] = "4"; //set mode to ramp
        }

        data[2] = Convert.ToString(Motor_Speed.Value * 10);
        data[3] = Convert.ToString(Step_Frequency.Value);
        data[4] = Convert.ToString(Amplitude.Value);
    }

    string toDisplay = string.Join(Environment.NewLine, data);
    MessageBox.Show(toDisplay);
    for (int i = 0; i < 5; i++)

```

```

    {
        //double[] datasend = [1;0;.4]; // This sends the data array piece by
piece, followed by a comma for arduino
        myport.WriteLine(data[i]); //Error!
        myport.WriteLine(",");
    }
    //Left_RS.BackColor = Color.Honeydew;
    //Right_RS.BackColor = Color.Honeydew;
    //Random_RS.BackColor = Color.Honeydew;

    //displays data array written to arduino
    //For debugging only: remove before final

}
private void Stop_Main_Click(object sender, EventArgs e) //not currently used.
{
    //myport.WriteLine("1,4"); //This code was me trying to send a quick value
for the meeting.
    //myport.Write(Convert.ToString(Motor_Speed.Value));

}

//Motor motion parameters
private void Amplitude_ValueChanged(object sender, EventArgs e) //Sets data
value for amplitude and generates preview
{
    data[4] = Convert.ToString(Amplitude.Value);
    SettingProps.Text = "Step Size. 0-60 by integers.";

    if (data[0] == "2" && data[1] == "2")
    {
        sweepPreview();
    }
    if (data[0] == "2" && data[1] == "3")
    {
        stairPreview();
    }
    if (data[0] == "2" && data[1] == "4")
    {
        rampPreview();
    }
}
private void Step_Frequency_ValueChanged(object sender, EventArgs e) //Sets data
value for iterations and generates preview
{
    data[4 - 1] = Convert.ToString(Step_Frequency.Value);
    SettingProps.Text = "Number of Iterations. 1-20 by integers.";

    if (data[0] == "2" && data[1] == "2")
    {
        sweepPreview();
    }
    if (data[0] == "2" && data[1] == "3")
    {
        stairPreview();
    }
    if (data[0] == "2" && data[1] == "4")
    {

```

```

        rampPreview();
    }
}
private void StairDuration_Mode_ValueChanged(object sender, EventArgs e)//Not
used anymore
{
    //data[6 - 1] = Convert.ToString(StairDuration_Mode.Value);
    //SettingProps.Text = "Maximum Pot value. Set to 1000 for test.";

    if (data[0] == "2" && data[1] == "2")
    {
        sweepPreview();
    }
    if (data[0] == "2" && data[1] == "3")
    {
        stairPreview();
    }
    if (data[0] == "2" && data[1] == "4")
    {
        rampPreview();
    }
}
private void Motor_Speed_ValueChanged(object sender, EventArgs e) //Sets data
value for motor speed and generates preview
{
    data[2] = Convert.ToString(Motor_Speed.Value*10); //Include a conversion in
Arduino code.
                                                    //Must be sent as
integer.
    SettingProps.Text = "Speed in deg/s. 0-4 by tenths.";

    if (data[0] == "2" && data[1] == "2")
    {
        sweepPreview();
    }
    if (data[0] == "2" && data[1] == "3")
    {
        stairPreview();
    }
    if (data[0] == "2" && data[1] == "4")
    {
        rampPreview();
    }
}

//Motor mode selection
private void Sweep_Mode_CheckedChanged(object sender, EventArgs e) //Sets
motor mode to sweep and generates preview
{
    data[1] = Convert.ToString(2);

    sweepPreview();
}
private void Staircase_Mode_CheckedChanged(object sender, EventArgs e)//Sets
motor mode to Stair and generates preview
{
    data[1] = Convert.ToString(3);
}

```

```

        stairPreview();
    }
    private void Ramp_Mode_CheckedChanged(object sender, EventArgs e) //Sets
motor mode to Ramp and generates preview
    {
        data[1] = Convert.ToString(4);

        rampPreview();
    }

    //Select motor or electromagnets
    private void tabControl1_Selected(object sender, EventArgs e) //Set current mode
as motor
    {
safety
        if (tabControl1.SelectedTab == tabPage2) //Turn off the electromagnets as a
        {
            data[0] = "1"; //EM
            data[1] = "2"; //Power off

            //Check if the com port is open, and write data to turn magnets off
            //myport.Open();

            init();

            if(myport.IsOpen == true)
            {
                for (int i = 0; i < 2; i++){

                    myport.WriteLine(data[i]); //write first two characters to arduino
                    myport.WriteLine(",");
                }

                //only change the data if the port is open. Avoids throwing error.
            }

            //Now set all the correct default values
            data[0] = "2"; //motor
            data[1] = "1"; //default to initialize
            data[2] = Convert.ToString(Motor_Speed.Value*10); //motor speed
            data[3] = Convert.ToString(Step_Frequency.Value); //iterations
            data[4] = Convert.ToString(Amplitude.Value); //amplitude
        }

        else if (tabControl1.SelectedTab == tabPage1)
        {
            init();

            data[1 - 1] = Convert.ToString(1);

            randomPanel.Visible = false;
            rightPanel.Visible = false;
            leftPanel.Visible = false;
            centerPanel.Visible = false;
            offPanel.Visible = true;
        }
    }
}

```

```

//old versions, empty
private void tabPage2_Click(object sender, EventArgs e)
{
}
private void tabPage1_Click(object sender, EventArgs e)
{
}

//electromagnet selection
private void Left_RS_Click(object sender, EventArgs e) //Selects Left option
{
    data[2] = Convert.ToString(1); //Left
    Left_RS.BackColor = Color.Aqua; //sets colors for ease of visibility
    Right_RS.BackColor = Color.Honeydew;
    Random_RS.BackColor = Color.Honeydew;

    //Changes panel visibility
    leftPanel.Visible = true;
    randomPanel.Visible = false;
    rightPanel.Visible = false;
    centerPanel.Visible = false;
    offPanel.Visible = false;
}
private void Right_RS_Click(object sender, EventArgs e) //Selects Right option
{
    data[2] = Convert.ToString(2); //Right
    Left_RS.BackColor = Color.Honeydew; //sets colors for ease of visibility
    Right_RS.BackColor = Color.Aqua;
    Random_RS.BackColor = Color.Honeydew;

    //Changes panel visibility
    rightPanel.Visible = true;
    randomPanel.Visible = false;
    leftPanel.Visible = false;
    centerPanel.Visible = false;
    offPanel.Visible = false;
}
private void Random_RS_Click(object sender, EventArgs e) //Selects Random option
{
    data[2] = Convert.ToString(3); //Random
    Left_RS.BackColor = Color.Honeydew; //sets colors for ease of visibility
    Right_RS.BackColor = Color.Honeydew;
    Random_RS.BackColor = Color.Aqua;

    //Changes panel visibility
    randomPanel.Visible = true;
    rightPanel.Visible = false;
    leftPanel.Visible = false;
    centerPanel.Visible = false;
    offPanel.Visible = false;
}

//magnet power
private void On_Power_Click(object sender, EventArgs e) //Turns on power to both
electromagnets
{
    data[0] = "1"; //EM
}

```

```

data[1] = "1"; //Power on
data[2] = "4"; //mode 4, both EM on

magnets
for (int i = 0; i < 3; i++) //write first 3 characters to arduino to turn on
{
    myport.WriteLine(data[i]);
    myport.WriteLine(",");
}

//Changes panel visibility
centerPanel.Visible = true;
randomPanel.Visible = false;
rightPanel.Visible = false;
leftPanel.Visible = false;
offPanel.Visible = false;

//debugging
//string toDisplay = string.Join(Environment.NewLine, data);
//MessageBox.Show(toDisplay);
}
off
private void Off_Power_Click(object sender, EventArgs e) //turns power to magnets
{
    data[0] = "1"; //EM
    data[1] = "2"; //Power off

    for (int i = 0; i < 2; i++)
    {
        myport.WriteLine(data[i]); //write first two characters to arduino
        myport.WriteLine(",");
    }
    //string toDisplay = string.Join(Environment.NewLine, data);
    //MessageBox.Show(toDisplay);

    randomPanel.Visible = false;
    rightPanel.Visible = false;
    leftPanel.Visible = false;
    centerPanel.Visible = false;
    offPanel.BringToFront();
    offPanel.Visible = true;
}
magnet
private void releaseButton_Click(object sender, EventArgs e) //Releases selected
{
    if (data[1] == "1") //If the magnets are powered
    {
        for (int i = 0; i < 5; i++) //write existing data
        {
            myport.WriteLine(data[i]);
            myport.WriteLine(",");
        }
        //reset button colors
        Left_RS.BackColor = Color.Honeydew;
        Right_RS.BackColor = Color.Honeydew;
        Random_RS.BackColor = Color.Honeydew;
    }
}

```

```

        else if (data[1] == "2") //If they're not powered, don't do anything!
        {
            MessageBox.Show("Magnets not powered. \r\nPlease power on before
continuing.");
        }

        //displays data array written to arduino
        //For debugging only: remove before final
        //string toDisplay = string.Join(Environment.NewLine, data);
        //MessageBox.Show(toDisplay);
    }

    //empty old things
    private void Stairsize_TextChanged(object sender, EventArgs e) //empty
    {
    }
    private void Motor_Ramp_TextChanged(object sender, EventArgs e)
    {
    }

    //COM Port selection
    private void COMPort_ValueChanged(object sender, EventArgs e) //changes port
value to selected
    {
        string COMport = Convert.ToString(COMPort.Value);
        //default must be set in numericUpDown properties
    }
    private void button1_Click(object sender, EventArgs e) //connects to arduino and
keeps connection open
    {
        init(); //runs connection function
        if (myport.IsOpen)
        {
            MessageBox.Show("Port Connected."); //Yay!
        }
        else
        {
            MessageBox.Show("No serial connection detected."); //whoops, try again
        }
    }

    //Magnet release time
    private void delayTime_ValueChanged(object sender, EventArgs e) //sets delay time
for magnet release
    {
        data[3] = Convert.ToString(delayTime.Value * 10); //Include a conversion in
Arduino code.
    }

    //Motor Initialize
    private void button2_Click(object sender, EventArgs e) // initialize motor; move
to 0 deg
    {
        data[0] = "2"; //set motor
        data[1] = "1"; //set mode to initialize
    }

```

```
data[2] = "20"; //set motor speed
//data[2] = Convert.ToString(Motor_Speed.Value*10); //set motor speed
//MessageBox.Show(Convert.ToString(data[1]));

for (int i = 0; i < 3; i++)
{
    myport.WriteLine(data[i]); //write first 3 characters to arduino
    myport.WriteLine(",");
}
}
}
```