EXPERIMENTAL SEARCH FOR AVALANCHES OF ENTANGLED DISLOCATIONSAS A SOURCE OF DISSIPATION AND MECHANICAL NOISE

A Thesis

Presented to

The Faculty of the Department of Physics and Astronomy

California State University, Los Angeles

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Physics

By

Morgan B. Shaner

May 2018

LIGO Document P1800114

The thesis of Morgan B. Shaner is approved.

Ed Rezayi, Committee Chair

Riccardo DeSalvo

William Taylor

Jose Rodriguez

Radi Jishi, Department Chair

California State University, Los Angeles

May 2018

ABSTRACT

Experimental Search for Avalanches of Entangled Dislocations

as a Source of Dissipation and Mechanical Noise

By

Morgan B. Shaner

Recent measurements using highly sensitive instruments have shown increased dissipation and the appearance of random low-frequency noise in metal flexures. These effects have been attributed to avalanches of entangled dislocations, a phenomenon believed to be controlled by self-organized criticality (SOC) statistics. This experiment is attempting to detect these subtle effects using a variation on the Kimball-Lovell 1927 rotating beam experiment that was used to measure the loss angle of materials above 1 Hz. A frequency range of 1 Hz and below was chosen to study the effects of dislocation entanglement. The loss angle of a piano wire has been measured with milli-radian precision and the feasibility of making future measurements of loss angle with micro-radian precision has been demonstrated at arbitrarily low rotation speeds. If dislocation avalanches are a source of the 1/f noise in these flexures (where f is the event frequency), it is expected that this experiment will be capable of detecting the expected deviations in loss angle.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Riccardo DeSalvo for his mentorship and guidance throughout the last two years. His work in the field of instrumentation is truly inspiring and has made him a wonderful knowledge base to bounce ideas off of.

Dr. Marina Mondin and Samavarti Gallardo have been very helpful and I would like to thank them for their work on the tracking program and data analysis to help automate as much of this experiment as we can for the future. I should also mention the work of other students including Nicole Araya, Hope Hamamoto, and Greta O'Dea. The work they contributed by helping diagnose problems and looking into possible solutions were very helpful in the evolution of the design of the experiment; which will continue to improve as I pass it to future students.

I would like to thank Caleb Schreibeis for hosting me at Montana Instruments over the summer of 2017 to complete a comparison study of maraging and vitrelloy 105 and for his technical suggestions when visiting the lab here at CSULA in March of 2018.

Upon my return, I met with William Johnson at Caltech and I would like to thank him for discussing failure modes in glassy materials with myself and Dr. Riccardo DeSalvo; his insight into the field helped us to plan for the future of this avenue of the project.

Thank you to Montana State University for allowing the use of their SEM in the ICAL Laboratory, CSULA machinist Daniel Roberto for his help in machining parts for the experiment, and Production Lapping for grinding the two remaining vitrelloy 105 blades to the same thickness for free.

Finally, I would like to thank my colleagues Bo Truong, Ronald Melendrez, Alexander Kass, Michael Goff, Seth Linker, and Lamar Glover for their support and much needed meal breaks while working on this experiment and writing this thesis.

This research was conducted within the LIGO scientific collaboration and has a the following DCC number: P1800114.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

This study is composed of two parts; the study of low-frequency mechanical noise from dislocation avalanches, if they exist, and of the use of glassy metal flexures as a possible dislocation-free alternative. Recent measurements using highly sensitive instruments have shown increased dissipation and the appearance of random low frequency noise in metal flexures [1]. This transition was observed to occur below 0.5 Hz, introducing low frequency noise into devices using metal flexures [2] [3]. These devices include, but are not limited to, seismic attenuators for Gravitational Wave Observatories, seismometers and perhaps instruments measuring the gravitational constant. The Virgo inverted pendulums experience a suspect random walk around their equilibrium point [4]. The 1/f noise (where f is the event frequency) may be appearing within the suspension systems of gravitational wave detectors, introducing noise into the extremely precise position measurements these devices make [5] [6]. Since Henry Cavendish devised the first experiment to measure the gravitational constant, G, in 1798, scientists have continued to utilize and improve upon his original design [7]. Although measurement precision has reached the 10^{-12} ppm level, the discrepancy between experiments is as large as ±500 ppm [1] [8] [9]. It is clear that there are uncontrolled parameters, perhaps the same one as in our experiments.

It is proposed that the cause of these instabilities are avalanches of dislocations within the crystal structure evolving in a self-organized critical (SOC) regime. SOC is a phenomenon in statistical mechanics in which loosely connected states self-correlate to produce emergent (non-predictable) events [8]. In this case, i.e. metal flexures, the states are dislocations in the crystalline structure, which can move and entangle. SOC tries to establish a currently unknown, fundamental relationship linking movements of these entangled dislocations at the atomic level to macroscopic mechanical properties. Dislocations are topological defects that extend from end to end of crystals and easily move through the material under the changing stress field. Because they cannot cross each other they entangle, as it happens in work hardening [10]. These entanglements can disentangle in slow avalanches and cause measurable structural changes [11]. This experiment is attempting to detect these subtle effects using a variation on the rotating beam Kimball-Lovell 1927 experiment that was used to measure the loss angle of materials above 1 Hz. Kimball and Lovell found that the loss angle of several materials was constant above 1 Hz [12]. This experiment aims to study frequencies below 1 Hz, where slow dislocation avalanches are expected to happen, to determine if there is a new dissipation regime. It is expected that if disentanglement happens, larger loss angle and sudden deviations of the equilibrium point of the flexure will appear during the rod's rotation at very low frequencies [2] [10].

Should the theory of SOC prove to be valid, a new non-crystalline material needs to be developed in order to remove the 1/f noise from high sensitivity instruments using metal flexures. Self-organized criticality is believed to be present in all polycrystalline metals; therefore, one solution is to use a glassy metal. Glassy metals are materials containing no dislocations, have twice the elastic limit of polycrystalline metals, are electrically conductive and have low mechanical losses [2]. Previous studies for a new material in measurements of G considered ceramic glassy materials (fused silica fibers), however, these fibers create a problem because they lack electrical conductivity, which is important in these devices. Thus, G. Cagnoli and his team tried coating them in metals, only to find large mechanical losses of the thinly deposited metals [8]. Creating glassy metal flexures may not only remove the differences in experimental measurements of G, but also influence the designs of scales, seismometers, gravimeters, and other inertial sensors that use metal flexures or suspension wires.

CHAPTER 2

Theory

The source of low frequency noise detected in the seismic attenuation systems used in the gravitational-wave observatories is thought to originate from dislocations in a metal flexure's crystalline structure at the microscopic level. These microscopic defects have macroscopic effects in the material through the following mechanism. Dislocations are highly prominent in all metals. Maraging and high carbon steels are to be explored first because studies have already been conducted on these materials and shown equilibrium point fluctuations below 0.5 Hz where the dislocations are able to entangle and create a complex structure. At a low enough frequency and at a critical point the entangled dislocations can disentangle causing avalanches which changes the elastic stress distribution within the material. On the macroscopic side this would appear as a shift in flexure's equilibrium point. In this section the role of self-organized criticality will be discussed in regard to explaining this process as well as the inspiration for a method to study these effects through the measurement of the material's loss angle at low frequencies.

2.1 Self-Organized Criticality

Some characteristics of self-organized critical (SOC) systems are that their evolution towards a critical state is internal and is reached without outside interference over a long period of time. Thus, macroscopic effects can only be studied on time scales much longer than the evolutionary process and their evolution is heavily dependent on the system's history [10]. Per Bak uses a sandpile to explain SOC in an elegant way:

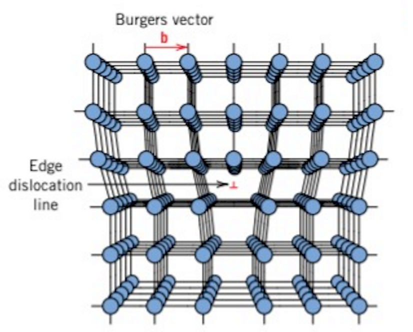
> *"The canonical example of SOC is a pile of sand. A sandpile exhibits punctuated equilibrium behavior, where periods of stasis are interrupted by intermittent sand slides. The sand slides, or avalanches, are caused by a domino effect, in which a single grain of sand pushes one or more other grains and causes them to topple. In turn, those grains, not gradual change, make the link between quantitative and qualitative behavior, and form the basis for emergent phenomena"* [10].

In this thesis SOC is used to explain dislocation behavior in order to connect the following observed macroscopic phenomena: "fluctuations of Young's Modulus, drastic changes in dissipation modes and hysteretic properties of the material, random walk of equilibrium point, spontaneous de-stabilization events leading to collapse, anomalous 1/f transfer fluctuations and 1/f mechanical noise" [2].

Dislocations are imperfections in a crystal's atomic structure and either consist of an extra partial plane of atoms (edge dislocation), a spiral distortion of normally parallel atom planes (screw dislocation), or (more commonly) a combination of the two types [14]. These dislocations are free to move through the crystal structure on timescales of micro-seconds which causes plasticity in metals. Their motion is called slip. The vectors

b and **l** (where **b** defines the strength of the slip a dislocation carries (also known as the Burger vector) and **l** is a unit vector tangent to the dislocation line that defines the orientation of the dislocation) form what is called a glide plane on which a dislocation is allowed to move as shown in **Figure 1** [14]. Climbing is the second mechanism of motion however, it is omitted because gliding is the dominant method of motion at low temperatures [14].

Dislocations are topological defects that cannot cross each other, thus leading to complex intertwined dislocation structures. The first method of blocking the motion of a dislocation is when a dislocation happens upon a point-like impurity which can manifest as a vacancy, self-interstitial, or substitutional impurity atom (in the case of maraging steel, these are most likely the added precipitates). At this point of contact, the dislocation can either become pinned, reflected or pass through. If pinned this creates a site, which cannot be avoided by the other dislocations because this dislocation locks a line that extends across the entire crystal, causing dislocation entanglement. Since dislocations can pass around point-like impurities, it is thought that the more common method of entanglement is through dislocations interacting with each other. As this complex structure evolves into self-critically it will reach a point of criticality in which the entangled system may avalanche causing a detectable macroscopic effect in the material. While an individual dislocation can respond in microseconds, the complex disentanglement involved in avalanching can be much slower. In maraging steel, characteristic times observed are on the order of seconds [2].



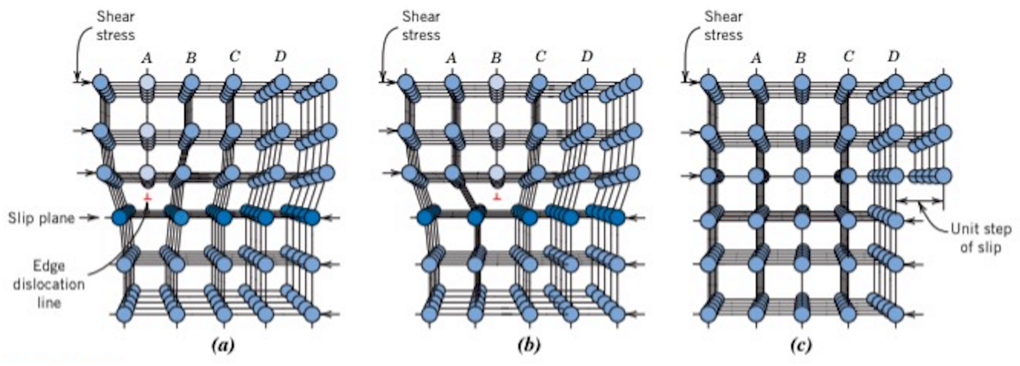


Figure 1. Crystal lattice structures illustrating an edge dislocation, an extra half-plane of atoms, (top) and the way an edge dislocation slips through the structure along the glide plane under a shear stress (bottom). The extra half-plane of atoms appears at site A in (a) and then travels one atomic distance to the right to site B as shown in (b), this movement continues until it reaches the edge where a step forms in (c) [14].

In this experiment we want to study these macroscopic effects on timescales long enough that the avalanche evolutionary process has time to complete. Thus, a low-frequency movement is necessary to give these dislocations time to disentangle and to entangle to track the macroscopic effects caused by the avalanches of these disentangling systems. If SOC theory proves correct in combining the outlined macroscopic effects (wreaking havoc on high precision measurements due to the microscopic dislocation motion) it would allow understanding of materials exhibiting SOC behavior.

2.2 Kimball and Lovell

In December of 1927, A. L. Kimball and D. E. Lovell published a paper entitled "Internal Friction in Solids" in which they studied the bend (sag) of a revolving shaft usually 1.27 cm in diameter and about 1 meter long [12]. They showed that what was then called the "loss angle" of a material, can be measured as the angle of deviation of the gravitational sag of the center of rotation from the vertical, as shown in **Figure 2**. The two scientists found that the internal friction forces were not behaving exactly like a viscus fluid. The dissipative forces instead appeared to be the same at all rotation speeds [12]. This experiment allows for the adjustment to the low frequencies needed to study the avalanches of entangled dislocations and for the study of how dislocation movements affect the internal friction of a solid.

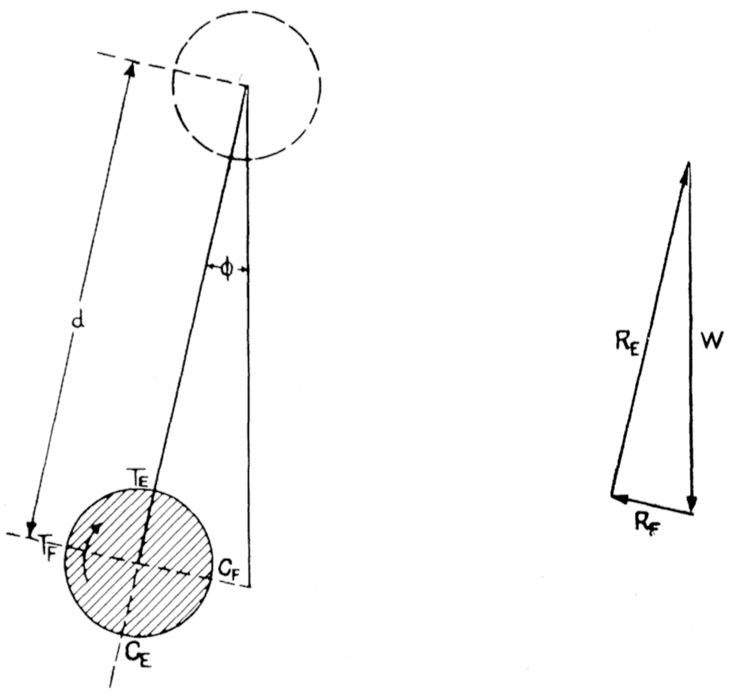


Figure 2. Lateral deflection of the rod's sag due to internal friction in the solid during rotation. Where the forces labeled in the diagram are as follows: frictional tension (T_F), frictional compression (C_F), elastic stresses (T_E and C_E), upward reaction (R_E) where R_E and R_F, are components of the force, which balances the weight (W) exerted on the end of the shaft. The other labels in the diagram are for the loss angle of the material (ie. angle of deflection) and d for the sag of the rod [12].

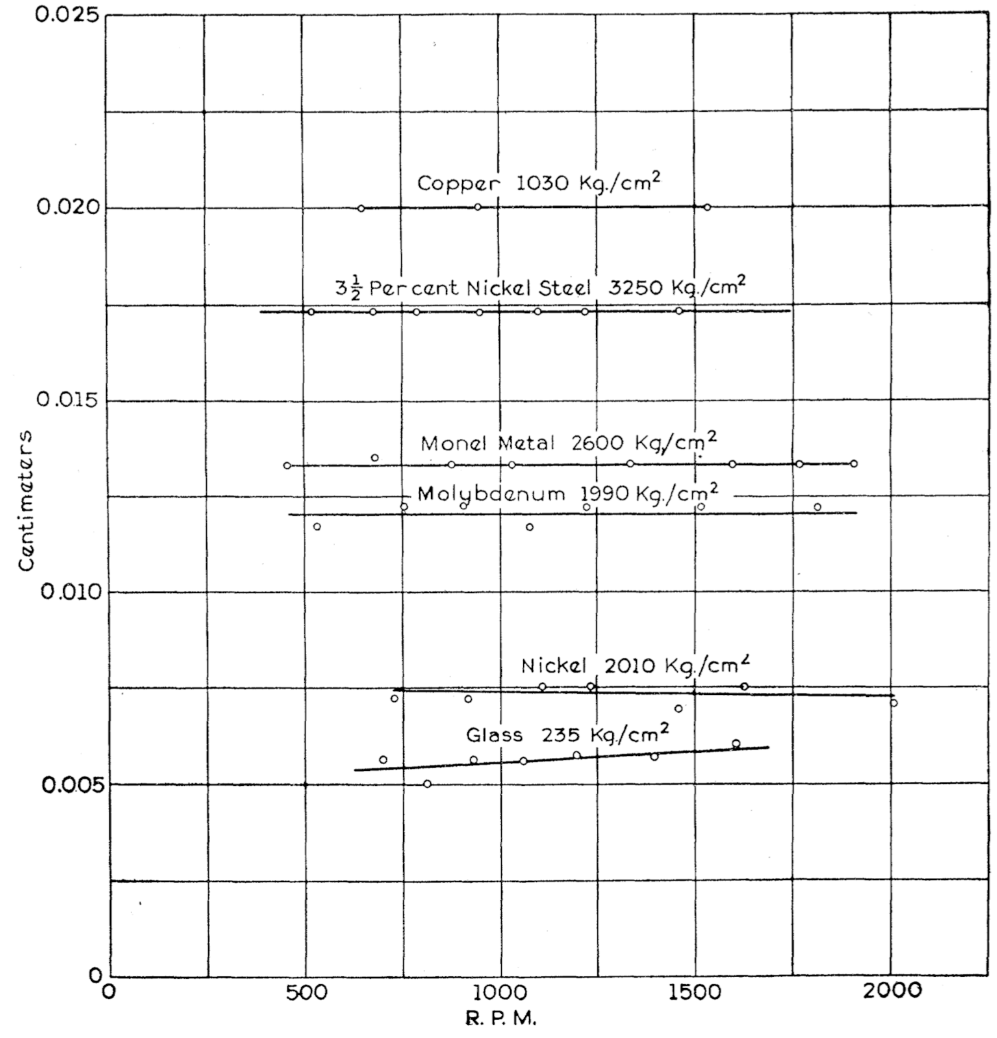


Figure 3. Plot of the sideways deflection of the end of the shaft against the speed of rotation for several different materials [12].

In **Figure 3** it is clear that for most materials the size of the deflection remains constant with rotational speed. If dislocations are a source of the dissipative forces previously discussed, it is expected that at low frequencies the deflection of the rod will increase. At low enough frequencies and a high enough precision in position measurement individual avalanches may even be visible with the modification of this experiment.

CHAPTER 3

Experiment

This experiment is attempting to detect the macroscopic effects on the equilibrium point of a flexure under gravitational stress caused by avalanches of entangled dislocations. The experiment is a variation on the rotating beam Kimball-Lovell 1927 experiment that was invented to measure the loss angle of materials. Kimball and Lovell were interested in dissipation above 1 Hz. The experiment has shown changes in dissipation at low frequency. The experiment, shown in **Figure 4**, is designed to work at frequencies below 1 Hz to study the dissipation behavior that is expected to occur below 0.5 Hz.

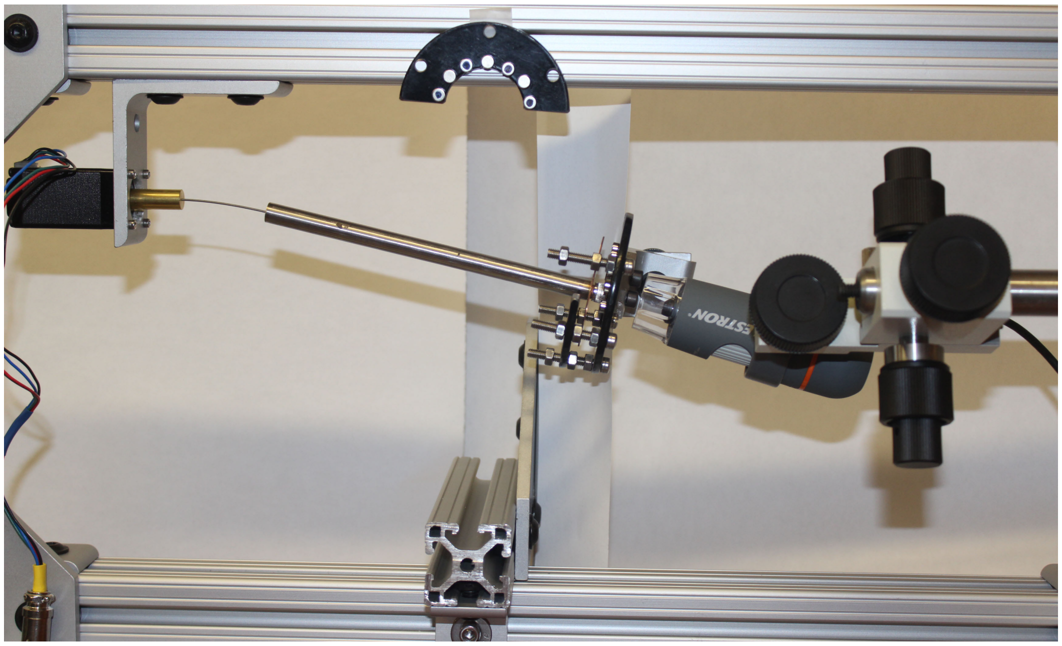


Figure 4. Image of self-organized criticality experiment (not fully assembled).

Starting on the far left is a stepper motor suspended from a rigid stainless steel frame, the flexure under test, in the form of a wire, is attached to the motor through a brass bushing and, at the other end, to a stainless steel tube with another brass bushing. The function of the tube is to enhance the torsional gravitational stress on the flexure, and to change any flexural deviation into large transversal motion that can easily be measured. At the end of the tube there is a third brass bushing supporting a high conductivity copper disk with a coaxial screw. The disk housed between two low carbon steel disks (not fully assembled in this image), each provided with 16 magnets glued in equal degree separation around the inner diameter of the disks with alternating polarity. These magnets provide strong magnetic field across the disk, thus forming an eddy current brake to damp all vibrations. The screw-cap supports an object used to track the movements of end of the rod. **Figure 14** shows the schematics of the setup, and the individual experiment components are discussed in detail in the following sections.

3.1 Frame

The rigid frame housing the experiment is composed of T-slot aluminum extrusion bars, with brackets and corners to enhance rigidity, as shown in **Figure 5**. The structure is also designed to be suspended from a geometric anti-spring (GAS) seismic isolation filter (see **Section 5.1.1**). All mountings for components are designed to be adjustable to allow for tuning range. They include:

- A stepper motor mount which is a simple corner bracket modified to accept the screws fastening to the front of the motor, see **Figure 6**.
- A microscope camera mount which allows for transverse motion in x and y, and vertical tilt of the camera to point perpendicular to the object being tracked at the end of the tube, see **Figure 8**.
- An eddy current damper mount, which extends from the frame using aluminum bars and corner brackets for transverse motion in x, y, and z and a rotation mount. Its function is to position the eddy current magnet holder around the copper disk, matching its angle, see **Figure 9**.

The frame can be tilted to change the stress level applied on the flexure. If turned 90^o it allows measurement of the flexure static bend, without gravitational torque stress applied.

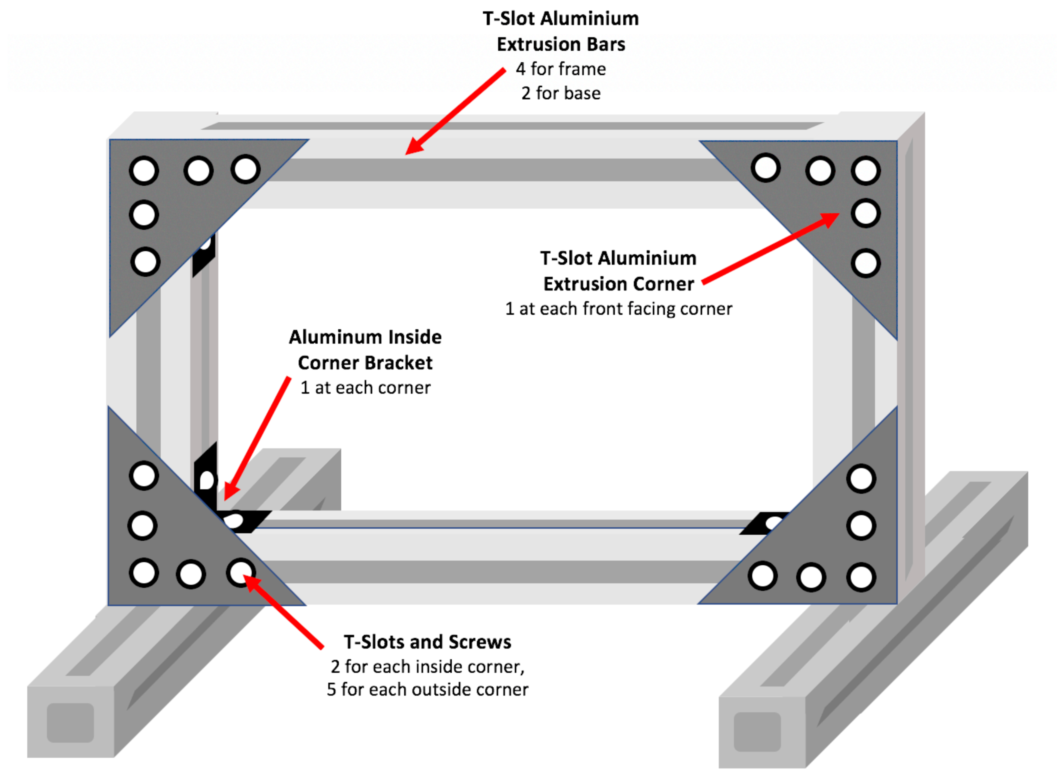

T-Slot Aluminium Extrusion Bars
4 for frame
2 for base
T-Slot Aluminium Extrusion Corner
1 at each front facing corner
Aluminum Inside Corner Bracket
1 at each corner
T-Slots and Screws
2 for each inside corner,
5 for each outside corner

Figure 5. Main frame without instrument mounts.

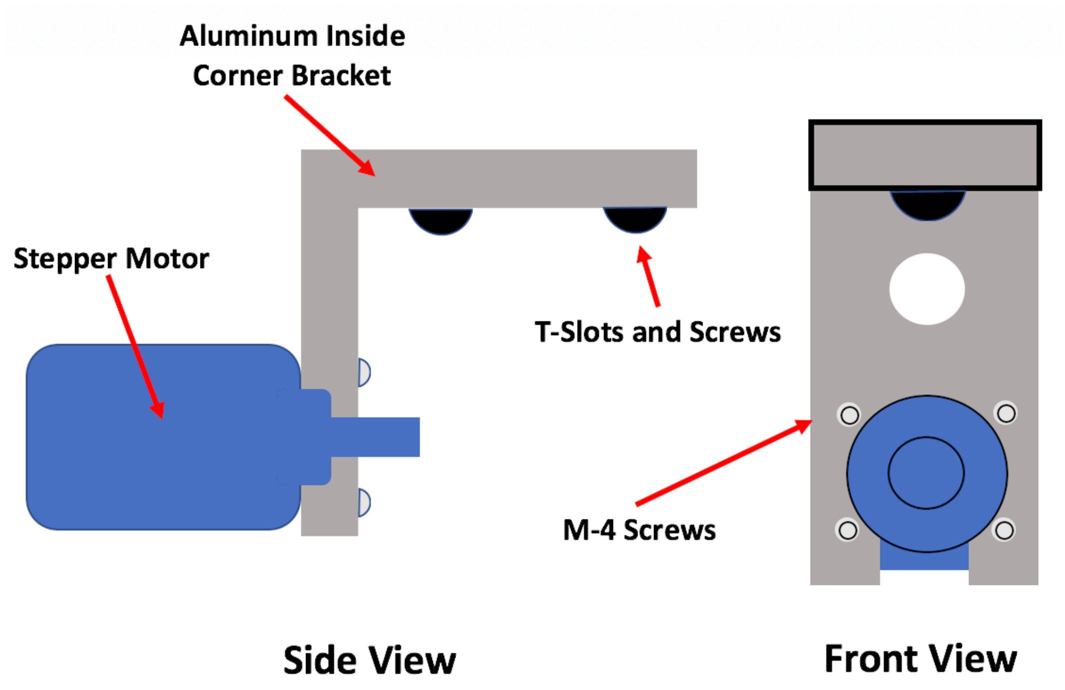


Figure 6. Motor mount drawing. Side view left and front view right.

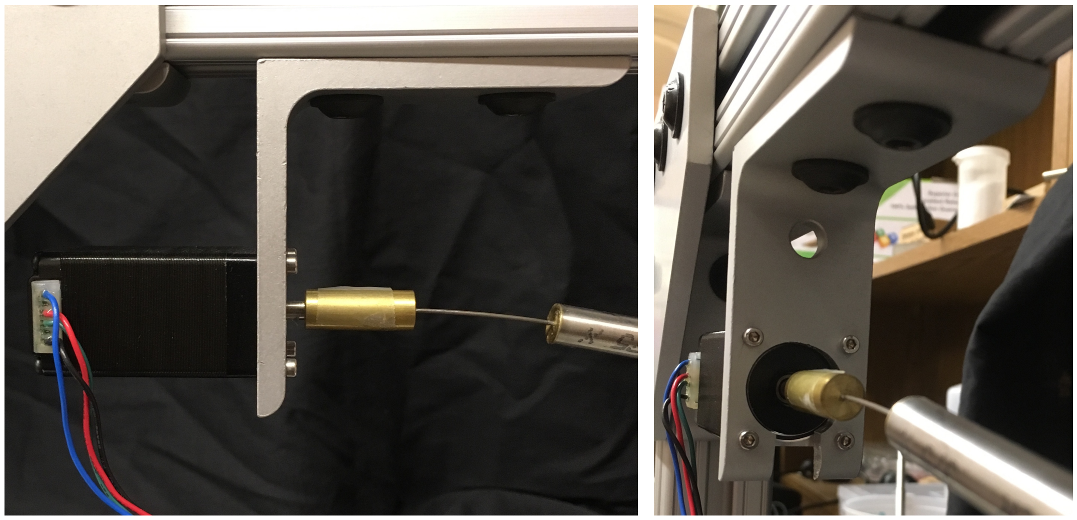


Figure 7. Images of the motor mount. Side view left and front view right.

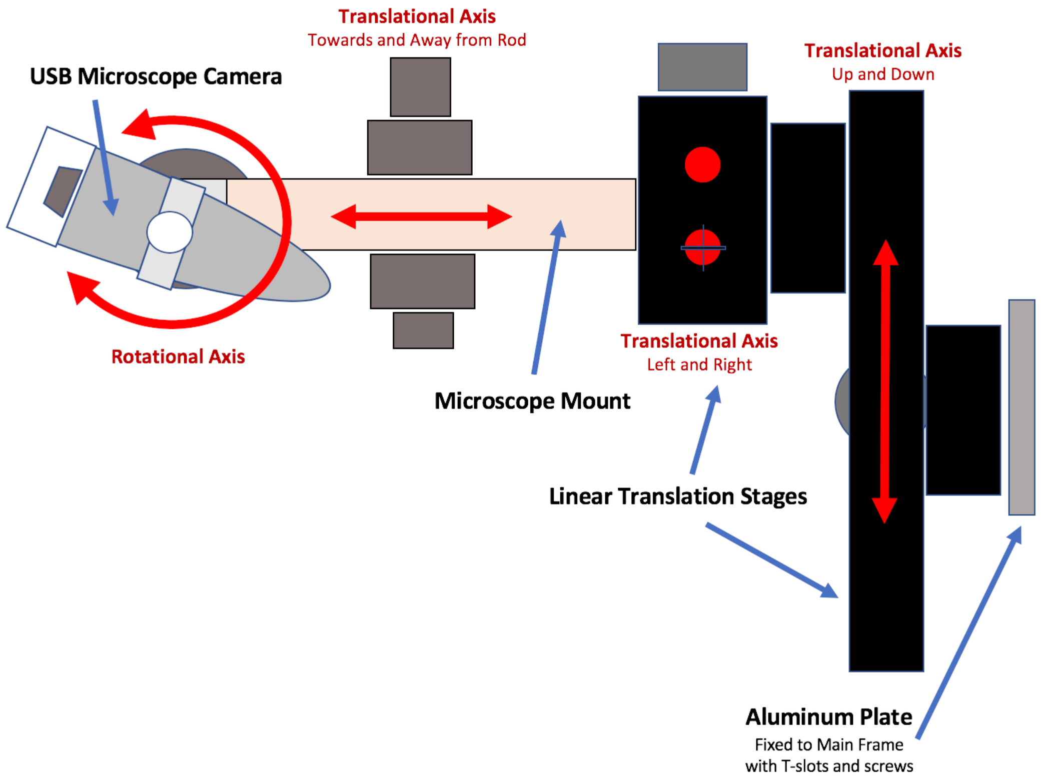


Figure 8. Side view of camera mount (Text in red are axis degrees of freedom).

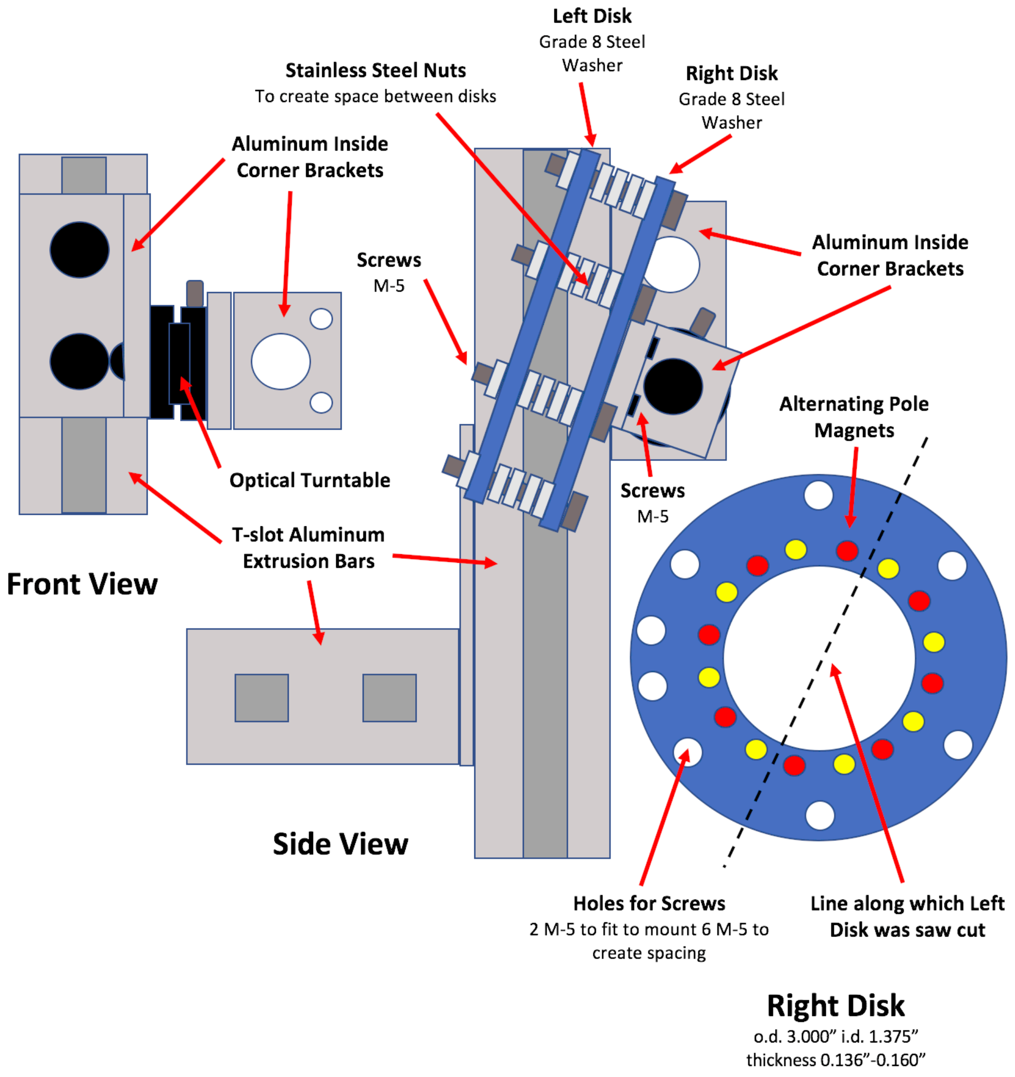


Figure 9. Eddy current brake mount. Left is a front view of the mount, center is the side view, and right is a detail of the right disk.

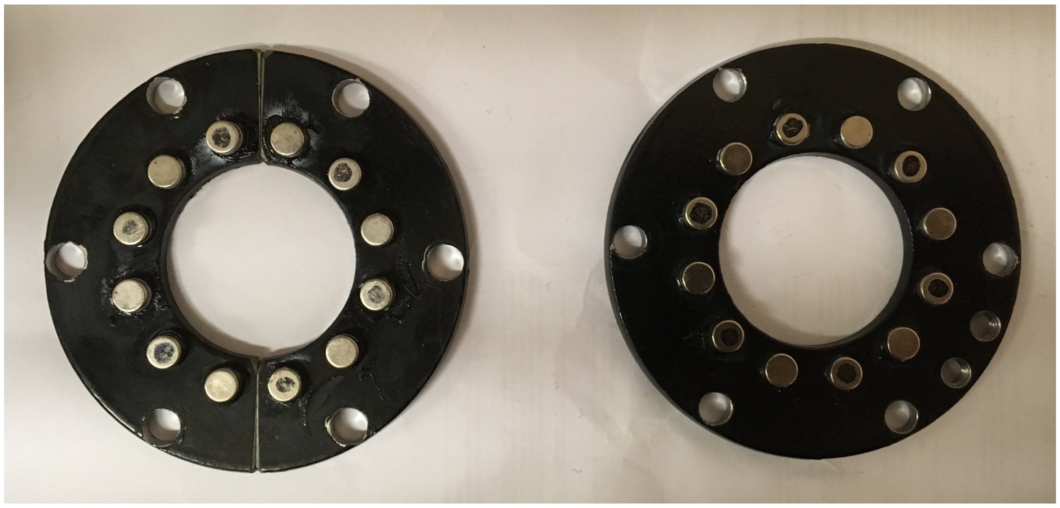
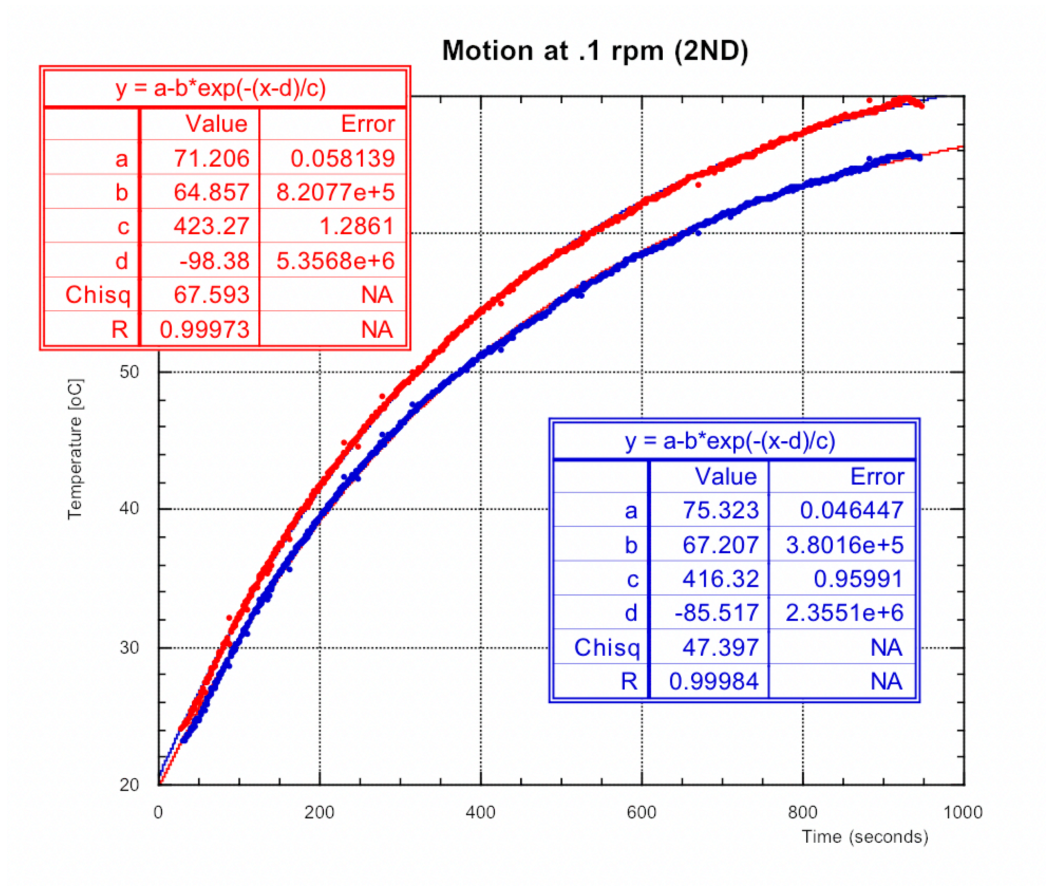


Figure 10. Image of the left and right disks. Black dots were applied with markers to indicate the North Pole of the magnet. One disk is cut in half for assembly reasons. The soft iron acts as a magnetic flux return confining the magnetic field to that which occurs between facing magnets.

3.2 Stepper Motor

The stepper motor used is a Trinamic MODEL QSH2818-51-07-012 (see **Figure 7**) and has 200 steps/turn. A high torque stepper motor was chosen because of its longer shaft, to reduce the effects of any residual play of its two pre-compressed roller bearings by a half. The motor driver MODEL TMC5130-EVAL allows for 256 microsteps/step. Stepper motors are known for their “stepping” vibrational noise at 200 times the rotational speed that may be a concern at the low r.p.m. chosen for this experiment. The operation in microsteps shifts the stepping noise to 51,200 times the chosen rotational frequency. Vibration frequencies from this noise, even when rotating as slow as 1 mHz, are well inside the frequency range effectively damped by the eddy current damper (see **Section 3.4**). It should be also noted that the stepping noise is principally a torque noise, which is orthogonal to the horizontal shift of the sag significant for this experiment. Since the motor is not subject to any significant torque load, it can be operated at low current. The motor is designed to run with a maximum current of 1.01 A. Tests were completed to ensure that rotation is not affected for currents as low as 0.08 A (see **Appendix A**). Operating the motor at 0.08 A, creates only 60 mW of power dissipation on the motor resistance of 9 Ω.



y = a-b*exp(-(x-d)/c)

	Value	Error
a	71.206	0.058139
b	64.857	8.2077e+5
c	423.27	1.2861
d	-98.38	5.3568e+6
Chisq	67.593	NA
R	0.99973	NA

y = a-b*exp(-(x-d)/c)

	Value	Error
a	75.323	0.046447
b	67.207	3.8016e+5
c	416.32	0.95991
d	-85.517	2.3551e+6
Chisq	47.397	NA
R	0.99984	NA

Figure 11. Plot of temperature data taken of the motor rotating at 0.1 r.p.m. fitted to find the maximum temperature of the motor and how long it takes the motor to thermalize. Plot and fit colors are reversed (i.e. Red data corresponds to blue fit parameters).

An investigation was done on the time constant of the heating caused by running the motor over large periods of time driven at different current inputs. It was found that regardless of the input current, the motor operated at 1.01 A, without cooling fins, thermalizes to a temperature of 71.21 ± 0.06 ^{o}C for the first thermometer and 75.32 ± 0.05 ^{o}C for the second, 50 ^{o}C above room temperature. Thermalization (at 5τ) can be considered to happen within 34.7 - 35.2 minutes. For this experiment, the motor was run at 1.01 A, however, in the actual experiment the motor will be run at 0.08 A, producing 0.16% of heating and a rise of 8 ^{o}C. Thus, data taken might show an initial drift in sag due to heating the motor, but then stabilize. A finned, aluminum heat radiator was designed to be attached to the motor and to increase its passive air cooling. In the future, the experiment will be placed inside an acoustic isolation box, which will also act as thermal isolation. The box will reduce the effect of external temperature fluctuations but will also bottle-up all heat produced by the motor. To avoid temperature run-off, a water-cooled radiator, separate from the motor, will be installed inside the box to sink the heat produced by the motor.

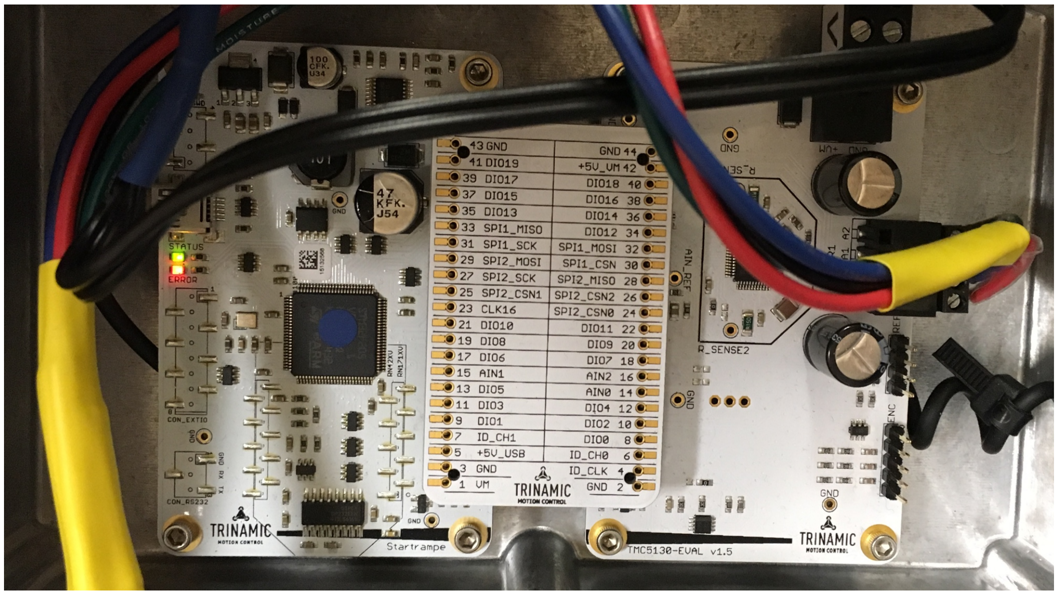
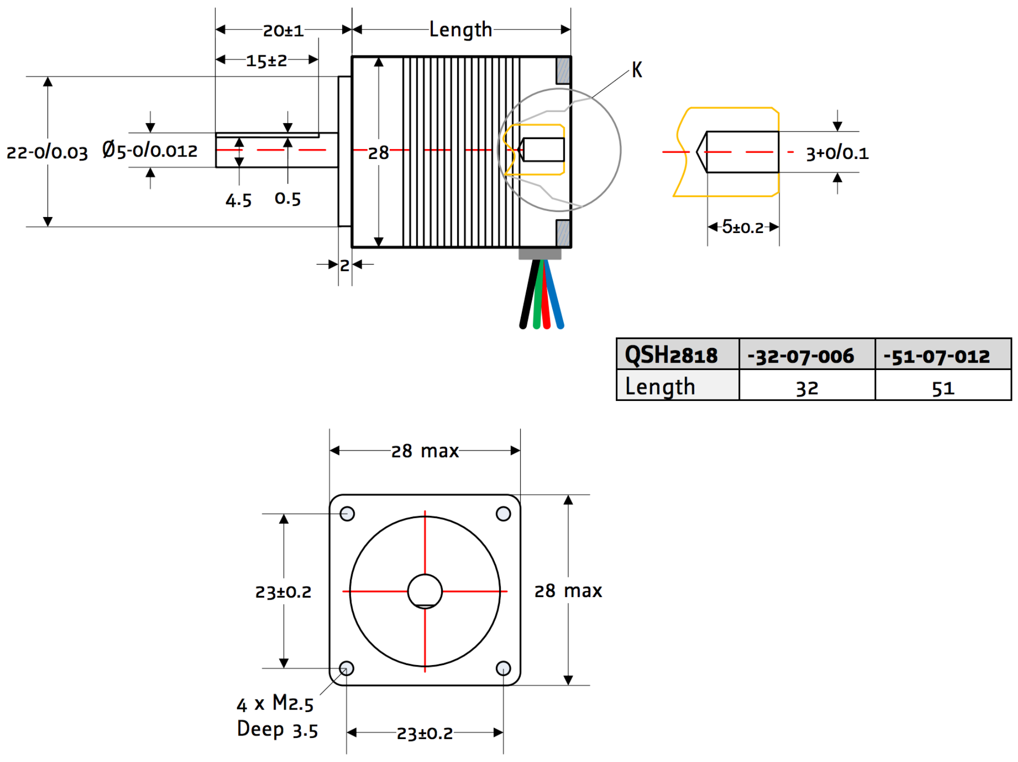


Figure 12. Image of the Trinamic stepper motor control board. The motor is attached on the right and the power supply on top the right corner. For manual see reference [15].



QSH2818	-32-07-006	-51-07-012
Length	32	51

Figure 13. Technical Drawing of the Stepper Motor with Dimensions in mm [16].

The set-up and use procedure for the control of the Trinamic motor is available in **Appendix A**. A larger and more rigid StepperOnline motor, Nema 17 MODEL 17HS24-2104S, was also procured in the unlikely case the Trinamic motor rigidity is shown to be problematic.

3.3 Rod Design

The rod design, see **Figure 14**, was created to obtain a large enough sag while imposing a well-defined torsional stress on the material, and allowing a easily customizable flexures to test a variety of materials[1]. These materials include high carbon steel, copper-beryllium, tungsten and maraging steel and were chosen because they exhibit excess losses [8]. The flexure stress level can be changed multiple ways:

- Changing the mass of the copper disk (which is a key component of the eddy current vibration damper)
- Changing the length of the tube
- Tilting the frame, thus changing the angle of the force applied by gravity
- Changing the flexure diameter

This allows for great flexibility in the experiment.

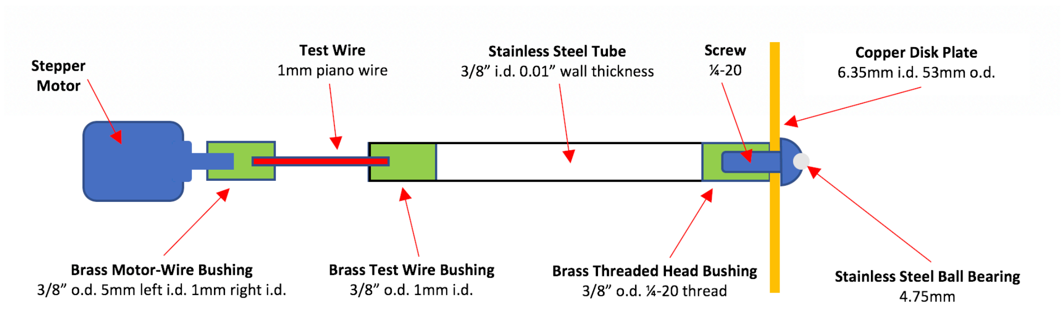


Figure 14. Current rod design under no gravitational stress.

Figure 14 shows the schematics of the flexure and rod design currently in use. In this design the parts are fit with close tolerances and glued together using a slow drying super glue to minimize spurious flexure between parts. The downside to gluing is that a new

[1] All parts were machined in bulk (6 each) by Daniel Roberto, a machinist who just retired from the California State University, Los Angeles Department of Engineering.

sample, including a new motor, must be created for each test material and configuration. Additionally, if not glued carefully we may end up measuring the loss angle of the glue instead of the loss angle of the material under test. For a more advanced version of the rod design see **Figure 30**.

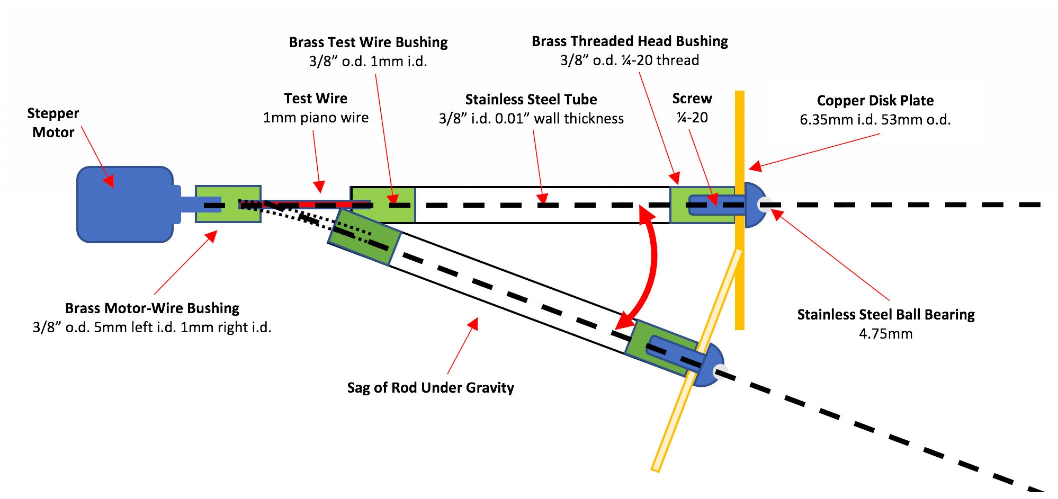


Figure 15. Current rod design showing how it sags under gravity.

The sketch in **Figure 15** and shows the rod bends under gravity. The dotted lines are the lines of symmetry for the rod with and without gravity applied.

3.4 Eddy Current Vibration Damper

Vibrations come from the surrounding environment, acoustic noise and from the motor itself. An eddy current disk damper was installed to remove these vibrations. A ring of magnets of alternating poles were glued around two disks, see **Figure 10**, in order to create a strong field crossing the disk in 16 locations. Two soft steel washers were used to channel the return field on the back of the magnets. One of the two washer was split in half for ease of installation around the tube, see **Figure 16**.

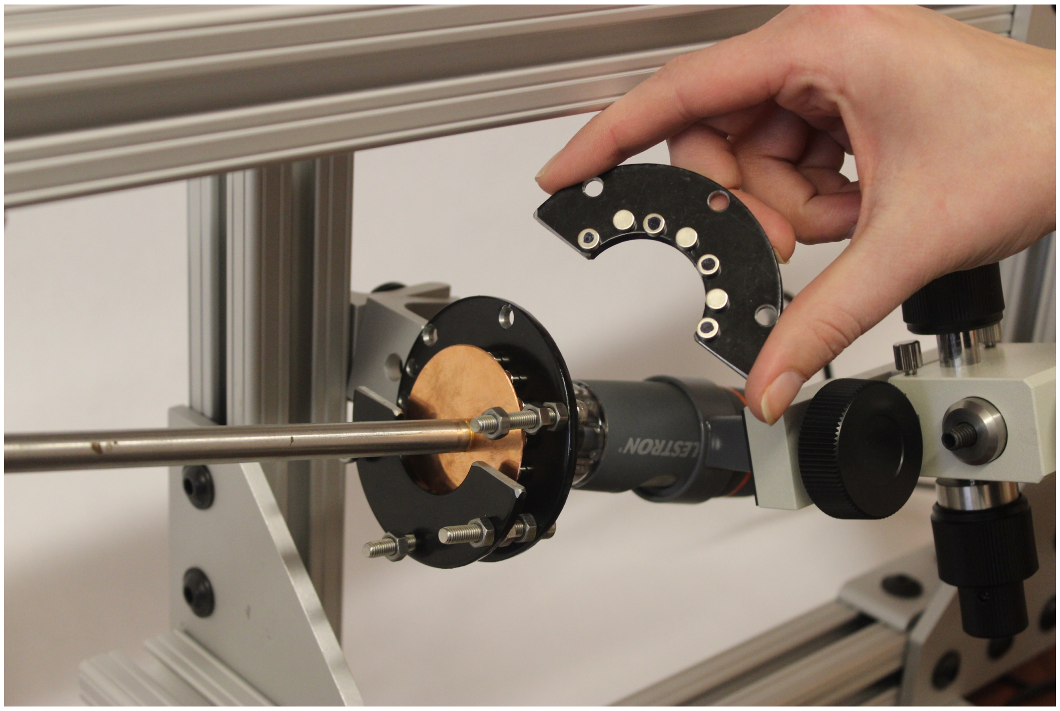


Figure 16. Eddy current damper (not fully assembled).

If the copper disk vibrates within the magnetic field, it induces current opposing the motion in the copper disk, viscously damping any vibration. A rough measurement of the magnetic field using a gaussmeter showing that the field remains constant across the separation an illustration of this effect is shown in **Figure 17** for one set of magnets across the disk. A test needs to be conducted to compare a copper disk to the aluminum disks of three different thicknesses to find the optimal material and disk separation for damping vibrations.

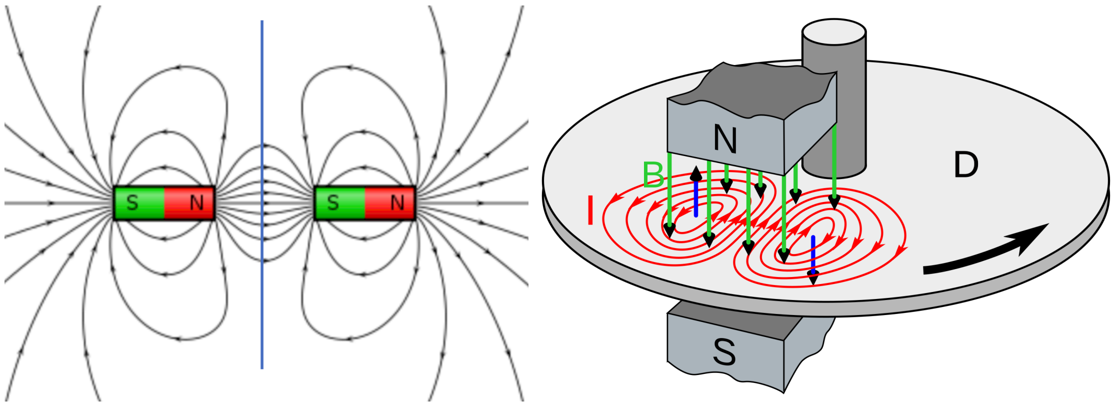
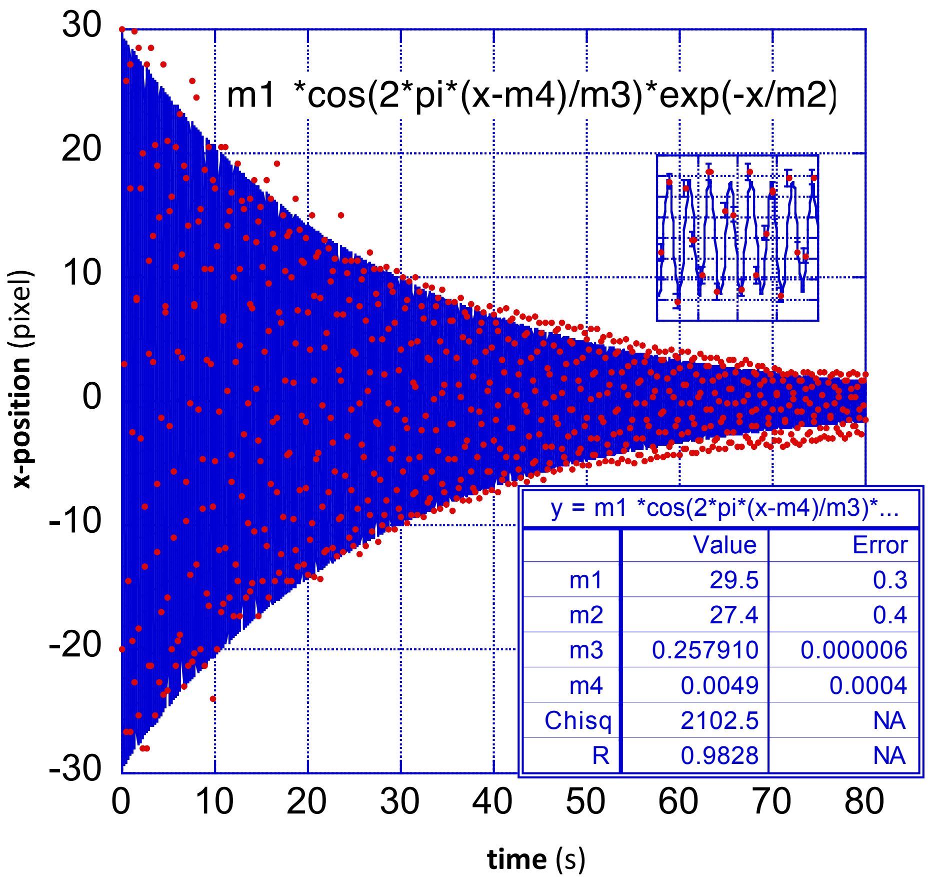


Figure 17. The image on the left illustrates the field lines from one magnet pair across the disk. The image on the right is a diagram showing how an eddy current disk brake works. The black arrow indicates that the disk is spinning clockwise, the green arrows indicate the magnetic field which induces circular current shown in red within the disk. These circular currents are called eddy currents. During rotation the induced currents are dragged with the motion and distorted by an amount proportional to the speed, and produce a force opposing the motion. From Lenz's law, the eddy currents create a changing magnetic field to oppose the field felt by the magnetic field applied across the disk [19] [20].

A functionality test was done to measure the effectiveness of the brake by inducing vibrations without the brake (**Figure 18**), with half of the brake (**Figure 19**), and with the entire brake (**Figure 20**).



y = m1 *cos(2*pi*(x-m4)/m3)*...		
	Value	Error
m1	29.5	0.3
m2	27.4	0.4
m3	0.257910	0.000006
m4	0.0049	0.0004
Chisq	2102.5	NA
R	0.9828	NA

Figure 18. No Damper. Red points are the data, the blue line is the fit to the data.

Clearly without damping any outside mechanical noise would excite large amplitude vibrations. The wire has a measured Q-Factor of 106.24 ± 0.2, obtained using the **Equation 3.1**.

$$Q = \frac{\tau\omega}{2\pi} \quad or \quad Q = \tau f \tag{3.1}$$

Where τ is the exponential decay time constant (the time it takes for the oscillation to damp to $1/e$ amplitude), and ω (or f) is the frequency of oscillation. The fit parameter m3 gives the period (T) of oscillation as 0.257910 ± 0.000006 s, using **Equation 3.2** this gives a frequency of 24.362 ± 0.004 Hz.

$$\omega = \frac{2\pi}{T} \quad or \quad f = \frac{1}{T} \tag{3.2}$$

The loss angle (ϕ) expected to be observed can be found using **Equation 3.3** to be 9.42 ± 0.02 milli-radians.

$$\phi = \tan^{-1}\frac{1}{Q} \tag{3.3}$$

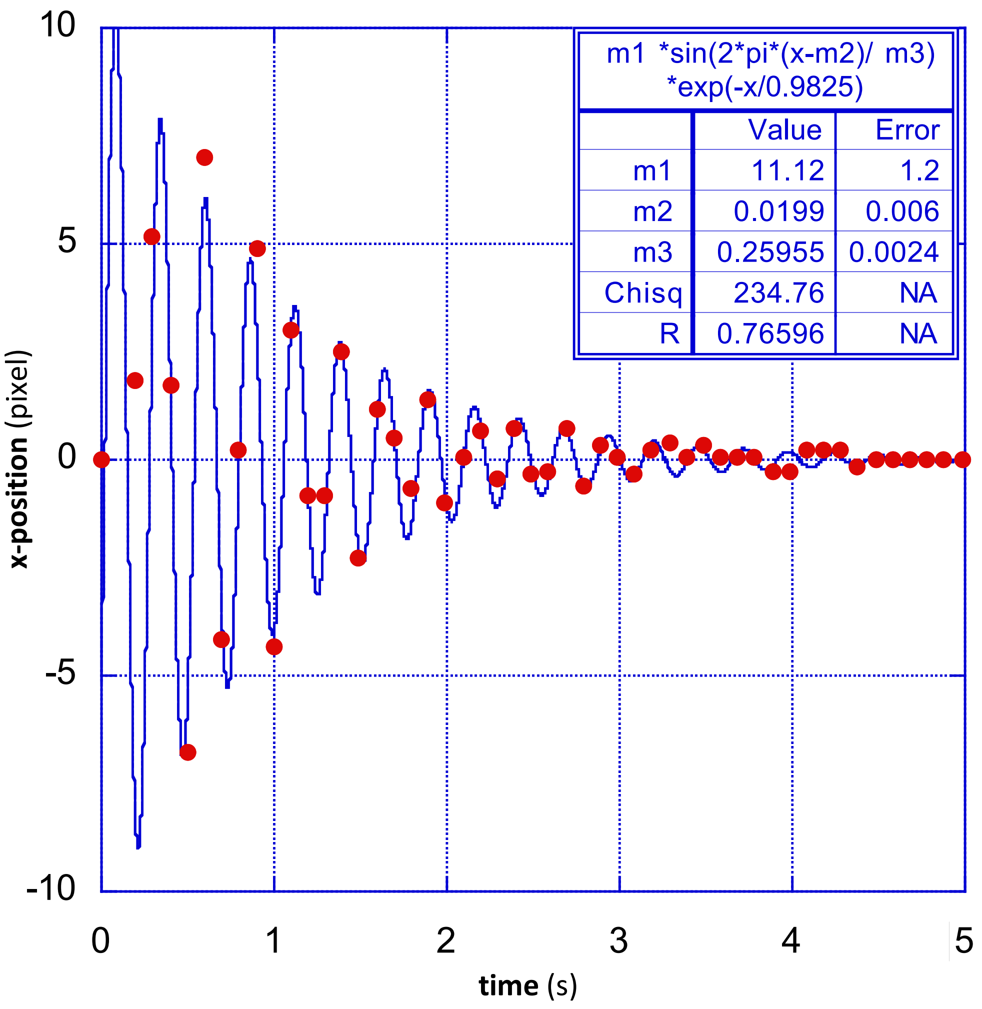
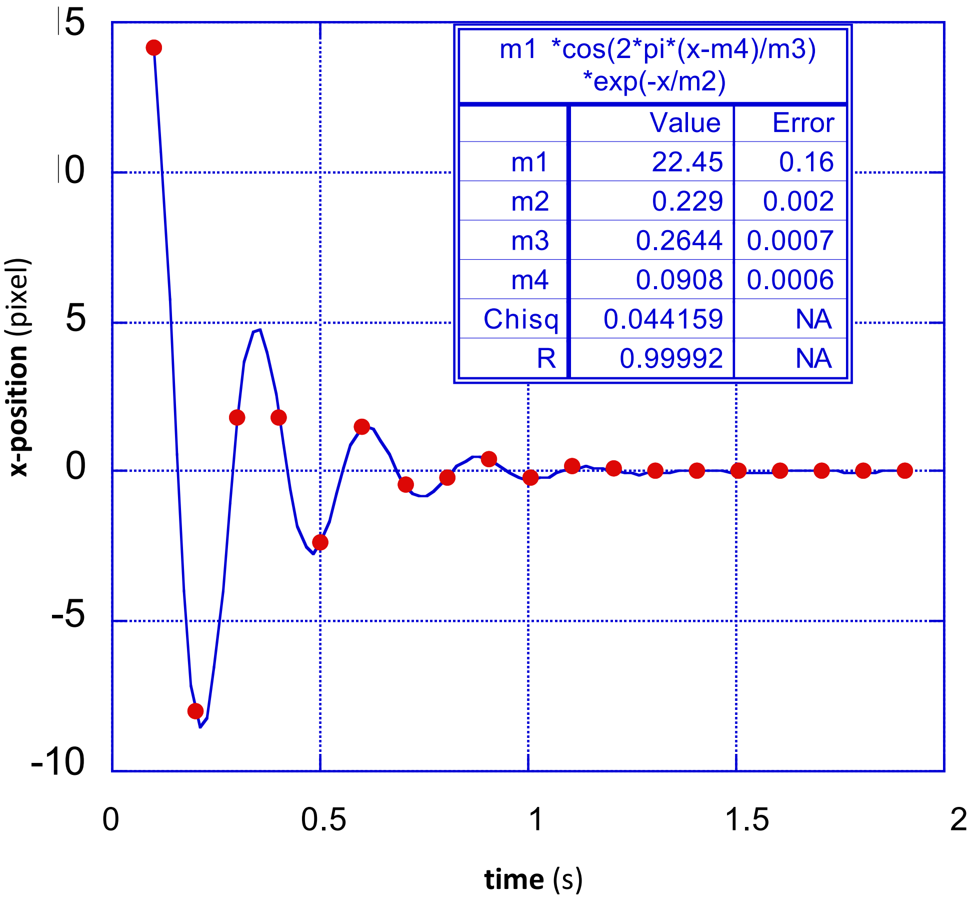


Figure 19. Half damper. Red points are the data, the blue line is the fit to the data.

With half of the damper installed a substantial amount of damping is induced.



m1 *cos(2*pi*(x-m4)/m3) *exp(-x/m2)	Value	Error
m1	22.45	0.16
m2	0.229	0.002
m3	0.2644	0.0007
m4	0.0908	0.0006
Chisq	0.044159	NA
R	0.99992	NA

Figure 20. Complete Damper. Red points are the data, the blue line is the fit to the data.

With the complete damper installed almost critical damping is achieved. The damper reduces the amplitude of resonances below the planned position resolution of tens of nanometers. In addition to the damper, the experiment will be housed in a Styrofoam box to remove the effects of air conditioning and suspended from a geometric anti-spring attenuation filter to remove seismic excitation, see **Section 5.1.1**. Another set of measurements will need to be taken of the magnetic field using a mount to ensure that the sensor remains stationary. It is interesting to note that in **Table 1** that the damper induces only a small frequency shift at the relatively high oscillation frequency of 24 Hz, indicating that its viscous damping has minimal effects on the sub-Hertz effects that are the object of this study.

Table 1

T and ω values for each test. Columns 2 and 3 are measured oscillation period and corresponding error, columns 4 and 5 are frequency and corresponding measurement error, column 6 is the frequency shift induced by the damper.

	T [s]	δT [s]	ω [Hz]	δω [Hz]	Δω [Hz]
No damper	0.257910	0.000006	24.362	0.004	----
Half damper	0.260	0.002	24	1	-0.36
Full damper	0.2644	0.0007	23.8	0.4	-0.60

The first resonant frequency of the flexure-tube-disk system (24 Hz), all higher frequency harmonics, and to the high frequency stepping noise will be damped, but the damper will not affect the static lateral changes of sag, nor its fluctuations, which if they exist, are expected to be well below 0.5 Hz.

3.5 Tracking and the Determination of the Gravitational Sag Changes

Neither the wire flexures, nor their fastenings to the motor and to the tube, are perfectly straight. Therefore, the end of the tube will move in a circle around a center of rotation that cannot be pre-determined. It is the sag of this center of rotation, and its lateral movements when rotating in different directions, which are relevant for this experiment. To do this we developed an orbit tracking method.

A first, low-resolution, experiment to track the position of the end of the rod was conducted using a 4.75 mm stainless steel ball which was glued to the hexagonal socket screw-cap fastening the damper disk to end of the rod. A program called "Tracker," available through open source educational software was used to track the motion of the ball as the stepper motor rotated the rod at a set frequency. The video of the rotation was taken using a USB microscope camera mounted to the frame angled to be perpendicular to the end surface. Once the video was taken, it was uploaded to the "Tracker" program and the reflection of the camera was used to create a template for the program to track. The program was then used to track the reflection through the video, in most cases the auto-tracking function was insufficient and manual tracking was necessary. This was due to the blur observed in videos taken at and above 0.5 Hz due to the limitation in frame rate and the exposure time for each image. The x-position, and y-position (in pixel units) were then transferred into a data file for fitting. The fitting method uses a polynomial (least squares) curve fitting method where it not only fits the set of data but uses a bootstrapping method to fit samples of the data to improve the fit. This technique allows the extraction of the center of both clockwise and counterclockwise rotations, which is then used to determine the loss angle of the material.

To convert the number of pixels was counted across the 4.75 mm diameter of the ball in one of the video frames. The size of a pixel was found to be 27.46 µm by 27.46 µm. An error of ±5.5 pixels in x and y from the tracking program, resulted in an error in a position of ±151 µm, see **Figure 21**.

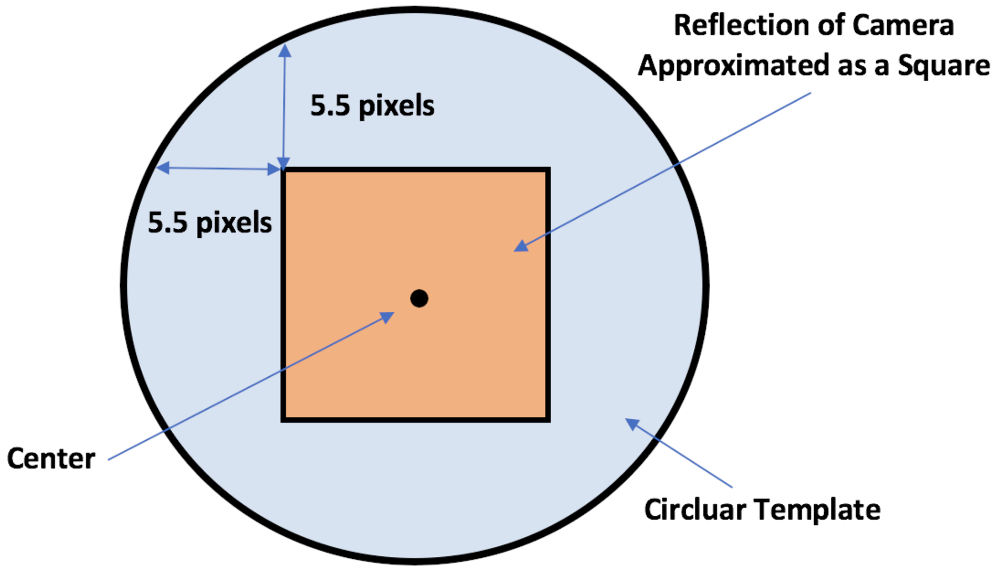


Figure 21. Determination of position error using "Tracker" program.

Once this conversion from pixels to µm was implemented, a Python code was used to complete the fitting process. The results from this method appear in the **Chapter 4** of this thesis. In the future, tracking will move away from this coarse method and a tracking software currently in development will be used to greatly improve the precision (see **Section 5.1.5**). A new camera needs to be purchased that will allow the user to take high-resolution videos without making the image acquisition too slow. At least 5 acquisitions/second are needed to obtain 10 images per orbit at 0.5 Hz. The current camera is too slow but has a very convenient feature; it has eight light emitting diodes (LEDs) around its charged coupled device (CCD), which is valuable to the eventual tracking of 49 μm glass beads. Not only does it allow for high contrast, but the reflection from each of the beads has a particular shape that allows for high precision tracking of the beads, as discussed in **Section 5.1.3**. Thus, when the new camera is purchased a ring of LEDs around the new CCD will need to be fabricated.

CHAPTER 4

Results

Using a 4.75 mm metal ball and a glued high carbon steel test wire, a signal was observed at a range of frequencies with loss angles appearing in **Table 2**. These values were found using the centers of the ellipses created during both clockwise and counterclockwise rotation by the sag of the rod under gravity.

$$\phi = \frac{\sqrt{\Delta x^2+\Delta y^2}}{2d} \tag{4.1}$$

Where d (33000 μm) is found by measuring the change in height of the end of the straight rod to the height of the end of the rod experiencing sag due to gravity, and both Δx and Δy mark the change in position from clockwise to counterclockwise rotation μm. The main data is in **Table 2** and plots of the raw data can be found in **Appendix B**. A control frequency of 1 Hz was used in order to correct for any drift in the sag due to the heating of the stepper motor. No such effect was seen in this data set, which could be due to the large error from the tracking program.

Table 2

Loss angle results at eight rotation speeds.

Frequency [Hz]		x [μm]	σx [μm]	y [μm]	σy [μm]	Δx [μm]	Δy [μm]	Magnitude [μm]	ϕ [radians]	Δϕ [radians]
0.05	clockwise	-705.0431204	17.38480948	2128.065546	16.17022022	-515.4567176	200.6335374	553.1269692	0.0084	0.0003
	counterclockwise	-1220.499838	21.66846895	2328.699083	14.78041749					
0.1	clockwise	-769.2187676	25.32793474	2162.614483	22.90867762	-580.4559033	193.2897986	611.79245	0.0093	0.0003
	counterclockwise	-1349.674671	32.3064157	2355.904281	20.64387386					
0.2	clockwise	-787.0012455	39.29380679	2210.593728	32.24326905	-497.9756569	124.4197428	513.2835739	0.0078	0.0002
	counterclockwise	-1284.976902	50.14229887	2335.013471	27.82479315					
0.3	clockwise	-757.0990577	59.75417122	2218.878636	40.11274887	-684.850828	109.7125948	693.5830953	0.0105	0.0003
	counterclockwise	-1441.949886	70.40102238	2328.591231	31.9400132					
0.4	clockwise	-785.0183955	73.91538813	2218.748387	50.77154644	-655.8207983	133.4556213	669.2617741	0.0101	0.0003
	counterclockwise	-1440.839194	84.71509025	2352.204008	36.92898495					
0.5	clockwise	-778.0979354	81.98026427	2193.572453	50.22247348	-679.7097049	157.7705196	697.7799222	0.0106	0.0003
	counterclockwise	-1457.80764	97.19698283	2351.342972	40.96974107					
0.75	clockwise	-675.856223	135.6713026	2176.615235	59.87064878	-795.1191666	97.25833473	801.0453625	0.0121	0.0004
	counterclockwise	-1470.97539	151.2326711	2273.87357	45.94170071					
1	clockwise	-800.5543475	54.47301304	2254.329266	23.91432305	-694.7015838	74.77637231	698.7143883	0.0106	0.0003
	counterclockwise	-1495.255931	62.9999566	2329.105638	20.06448893					

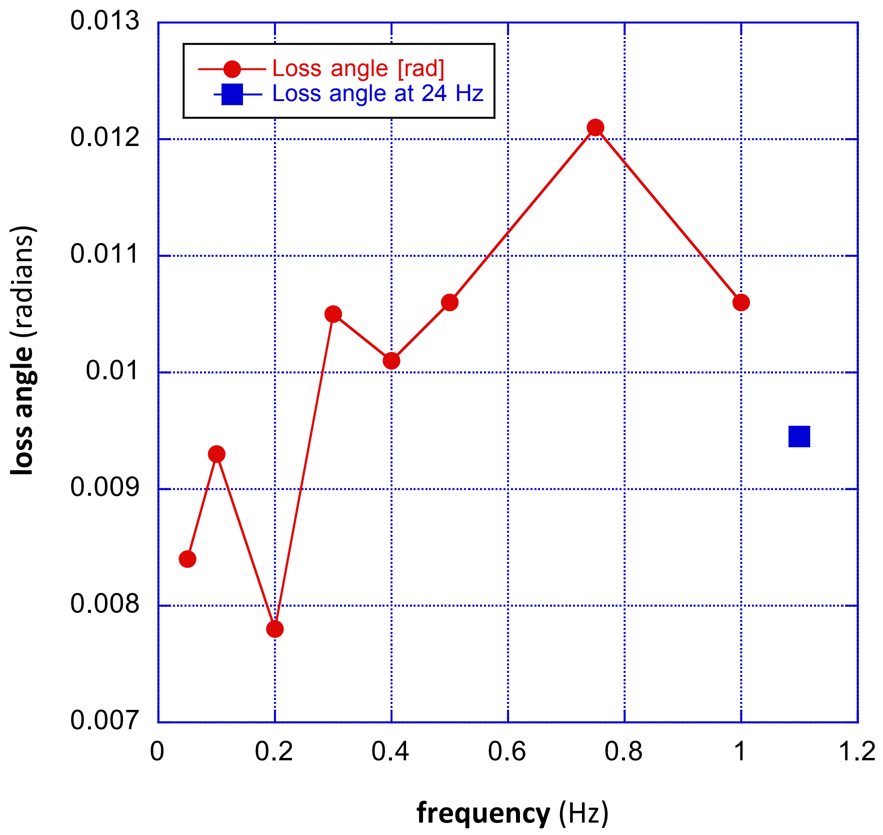


Figure 22. Plot of the measured loss angle against frequency of rotation. The expected loss angle measured at 24 Hz is artificially placed at 1.1 Hz to allow for convenience.

To obtain the data, data was taken in from the highest rotating frequency of 0.75 Hz to the lowest of 0.05 Hz, continuously alternating between 1Hz and the desired test frequency to reduce systematic drifts. Each data set contained one repetition of 5 rotations clockwise (motor position: 0 – 256,000), followed by 5 rotations counter-clockwise (motor position: 256,000 – 0); and were separated into clockwise and counter-clockwise rotation paths prior to fitting. The data was then run through a program in Python, which used a polynomial curve fit to find the center of the ellipse. It had error calculated from the bootstrapping method in the program that samples the data and then

calculates a center for each fit to obtain a sigma value. The difference in the x and y positions of the two centers were taken to find the loss angle using **Equation 4.1**.
The data analyzed in **Table 2** had a visible problem, as illustrated in **Figure 32**: it is possible that the copper disk was making contact with the eddy current brake mount during the measurement. Nevertheless, a first data analysis was attempted.

When testing the effectiveness of the eddy current damper, the loss angle of the piano wire was calculated from the 24 Hz ring-down data shown in **Figure 18** and was found to be 9.42 ± 0.02 milli-radians. This value was taken with the same rod used for this experiment and was thus used as a comparison value to ensure the results found using the rotating rod method agreed with the oscillation data. The measured loss angle was found to bounce around that value (see **Figure 22**). The 24 Hz value differed from the average of the results in **Table 2** (9.9 ± 0.3 milli-radians) by 5%. This is a superb agreement given the current state of the experiment. The postulated increase of loss angle at low frequency is not visible in this carbon steel sample. Please note that both measurements were performed with the glued sample, as a result we may have been measuring the losses in the glue.

With the improvements outlined in the following chapter; we can greatly improve the precision in the loss angle to 10's of micro-radians (nm precision in position) and remove many sources of noise currently appearing in the data.

CHAPTER 5

Future Improvements and Research Areas

This experiment is too complex to be completed in a single master thesis. From the beginning, this project was intended to be performed by a sequence of students. Therefore, this chapter is particularly important; it includes a number of improvements, which can be made to bring the experiment to a successful conclusion as well as a possible extension of the project, which entails looking into possible replacement materials for the outlined devices using metal flexures should the dislocation avalanche theory prove to be true.

5.1 Improvements to the SOC Experiment

This section contains improvements to be made including the suspension of the system from a GAS Filter, stabilizing the temperature of the experiment, improvements to the position measurement precision, a new design for the rod, and a new tracking program to automate the process. Orders of magnitude progress was made in improving the position measurement precision.

5.1.1 GAS Filter

As the precision is increased in the experiment, seismic noise will become more prominent in the data. Suspension from a geometric anti-spring (GAS) filter shown in **Figure 23** can be used to isolate the experiment from seismic noise. It does this through the use of maraging blade springs in a geometric anti-spring configuration, which behave like the three springs shown in **Figure 24**.

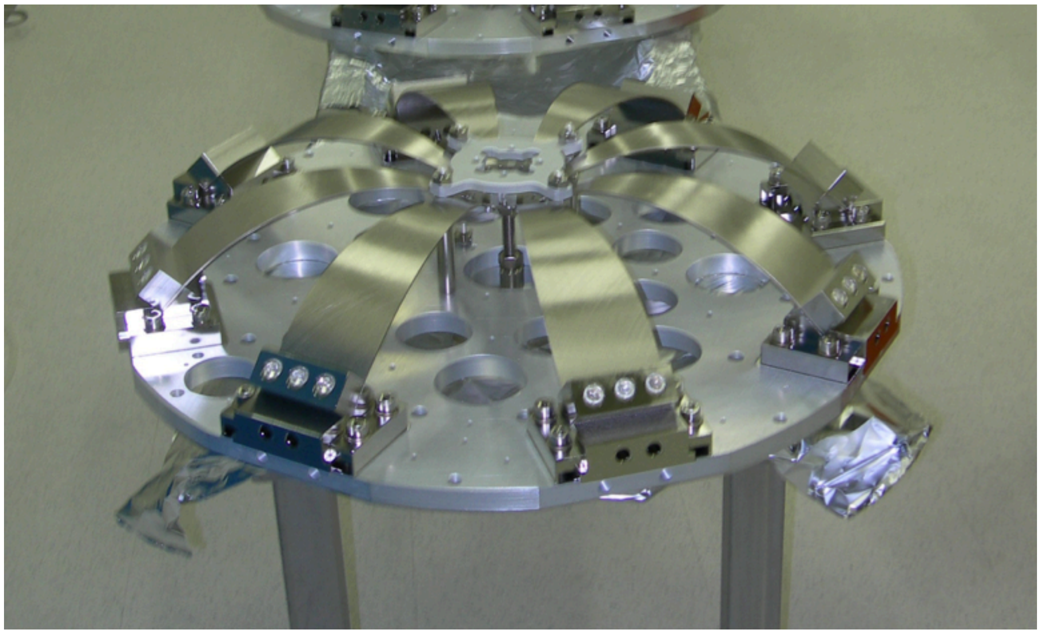


Figure 23. GAS filter used in LIGO [19].

The stiff vertical spring supports the weight with a large, negative elastic constant k_v. The two horizontal springs are pre-compressed against each other. At equilibrium the radial compressional forces F_H of the two horizontal springs cancel. When moving vertically away from the equilibrium point the vector sum of the two springs produce a vertically repulsive force proportional to the compressional force and the vertical displacement.

$$F = k_{anti}\, z \tag{5.1}$$

where $k_{anti} = 2F_H h/L$. The equation is formally equivalent to that of a spring, only with a positive sign, i.e. an antispring. An antispring of arbitrary strength is generated by changing the radial compression F_H. By tuning the antispring constant k_{anti} to match that of the vertical spring, $k_{eff} = k_{anti} - k_v \approx 0$, the effective stiffness of the GAS spring can be nulled while the suspended load remains unchanged. A spring of null elastic constant is unable to transmit perturbations in the z axis to its payload, hence the mechanical attenuation of vibration. A wire suspension isolates the payload from horizontal perturbations.

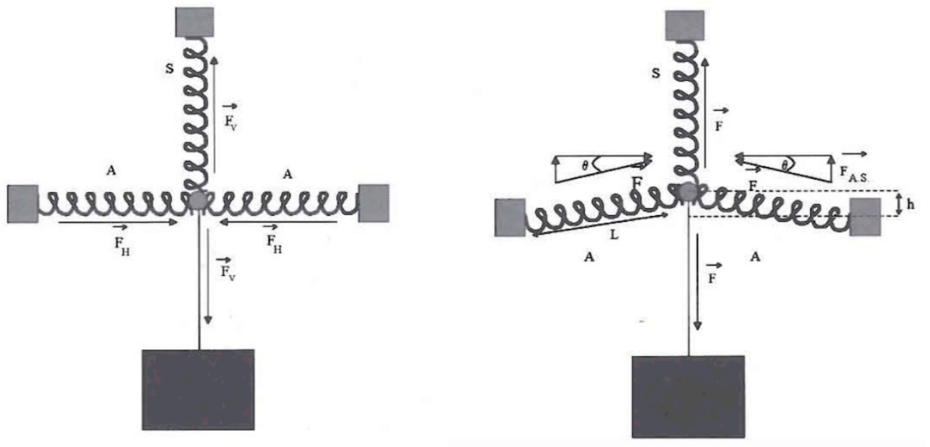


Figure 24. 2D vertical plane schematics of a geometric anti spring [9].

There is a GAS filter set-up available in the lab for this purpose.

5.1.2 Stabilizing the Experiment Temperature

The experiment needs to be housed in a thick Styrofoam box to shield it from ambient temperature fluctuations and from acoustic noise. A water-cooled heat sink will be installed to evacuate the heat radiated by the stepper motor and maintain the experiment at a given temperature. Both the heat sink and the motor will be equipped with fins to radiate and exchange the heat.

5.1.3 Position Measurement Precision

The steel ball does not provide sufficiently precise measurements of the tube front face. A much better position precision was demonstrated using a CCD microscope tracking the position of 49 μm diameter glass beads markers mounted on a black glass disk, mounted on the front face of the tube.

There is no need to mark the center of the rod, because it will “orbit” around a center of rotation determined by straightness defect s in the assembly, which cannot be pre-determined. Since only movements of the center of rotation are of concern, the orbit of any marker positioned on the front of the rod is suitable for identifying the center of rotation and its movements. Therefore, the beads were simply sprinkled on the surface of the black glass disk, which had been coated with a µm thin layer of wax. The disk was then heated to melt the wax to grab and hold the beads. The black glass disk is then simply glued on the cap of the screw at the end of the tube.

The choice of black glass is to increase contrast. The microscope has eight LEDs, see **Figure 25**, each producing a reflection on each bead. The reflections appear as a daisy shape, see **Figure 26**. The daisy of loose beads appears with a core due to the fraction of light entering and trapped in the bead; the wax produces an optical contact with the black glass that drains the trapped light and makes the center dot disappear. It is therefore easy to pick beads solidly attached to the glass and ignore loose ones. To produce position determination with nm resolution the image of a chosen daisy is digitized and analyzed for each successive microscope frame. A demonstration of the achievable resolution is illustrated in **Figure 26**, **Figure 27** and **Figure 28**. **Figure 26** shows a number of daisies. A slice of the digitized daisy, along the cut of **Figure 27**, was taken and fitted in **Figure 28** to achieve 38 nm precision.

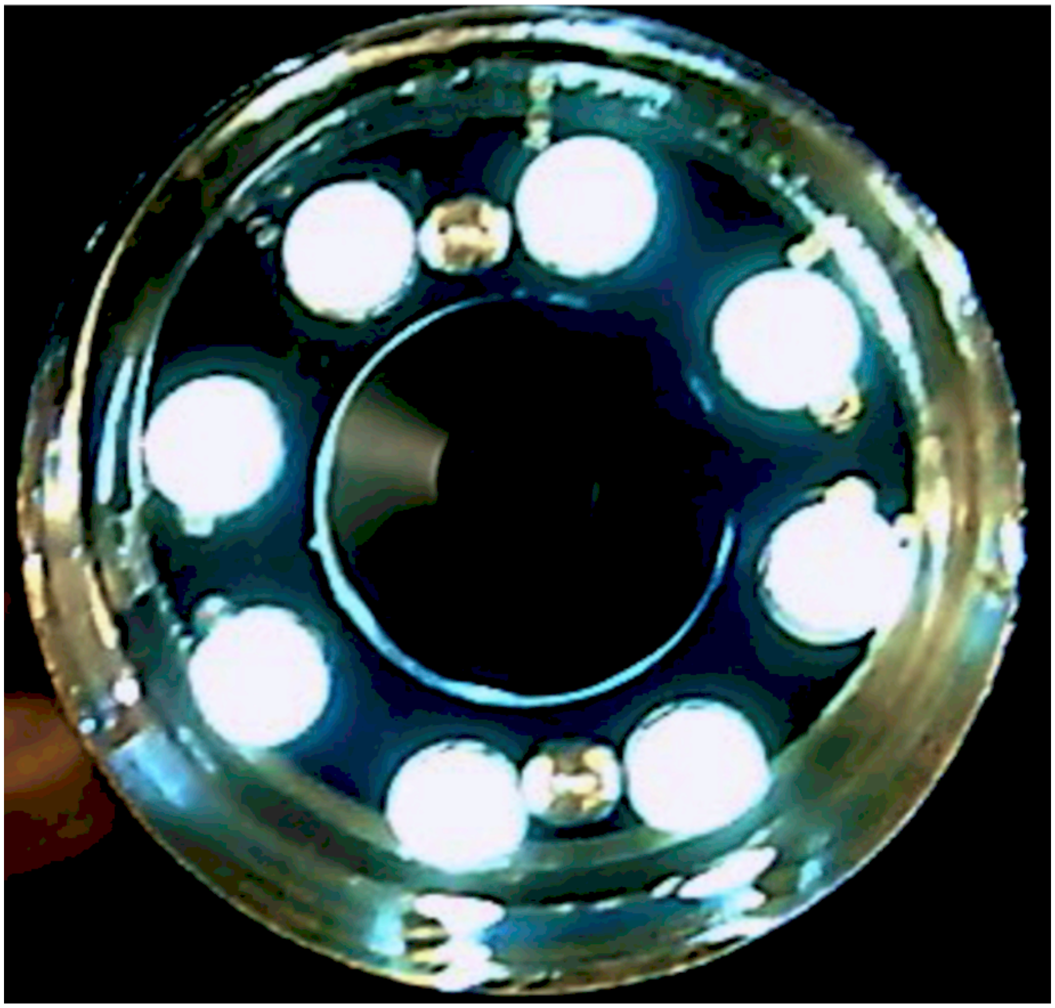


Figure 25. USB microscope camera's LEDs.

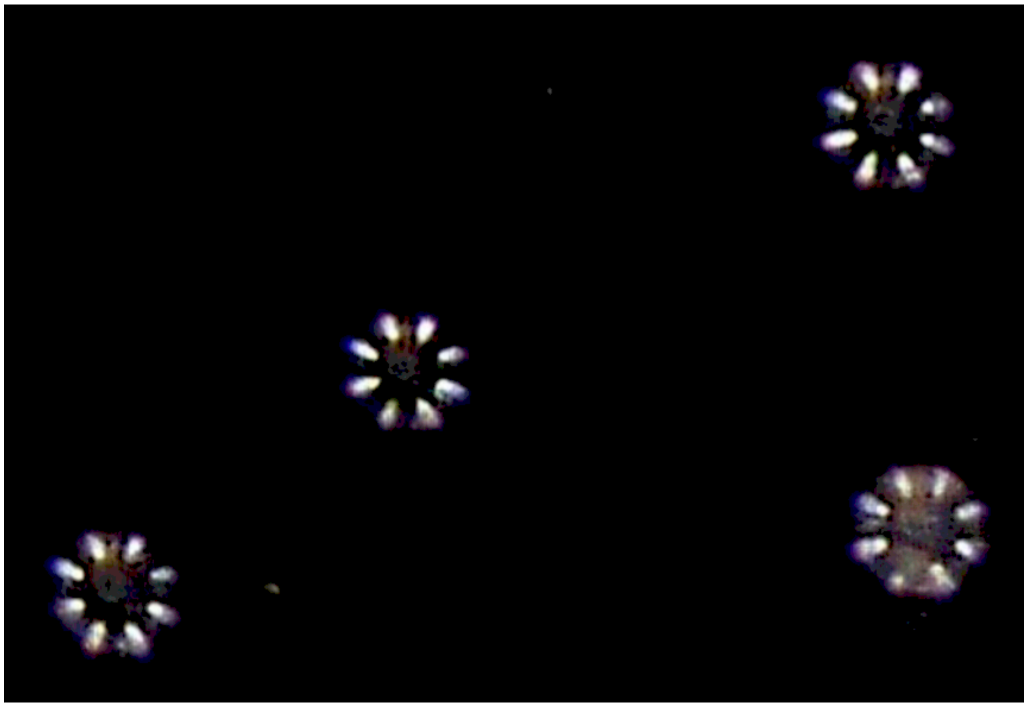


Figure 26. Reflections of the eight LED’s in the 49 μm glass beads.

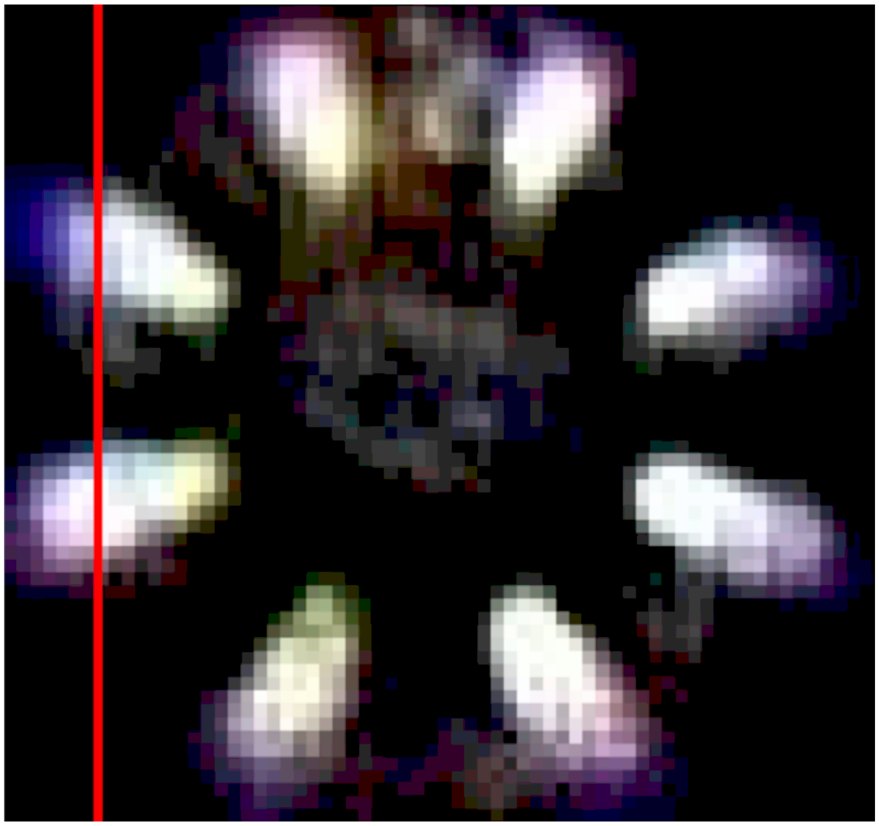
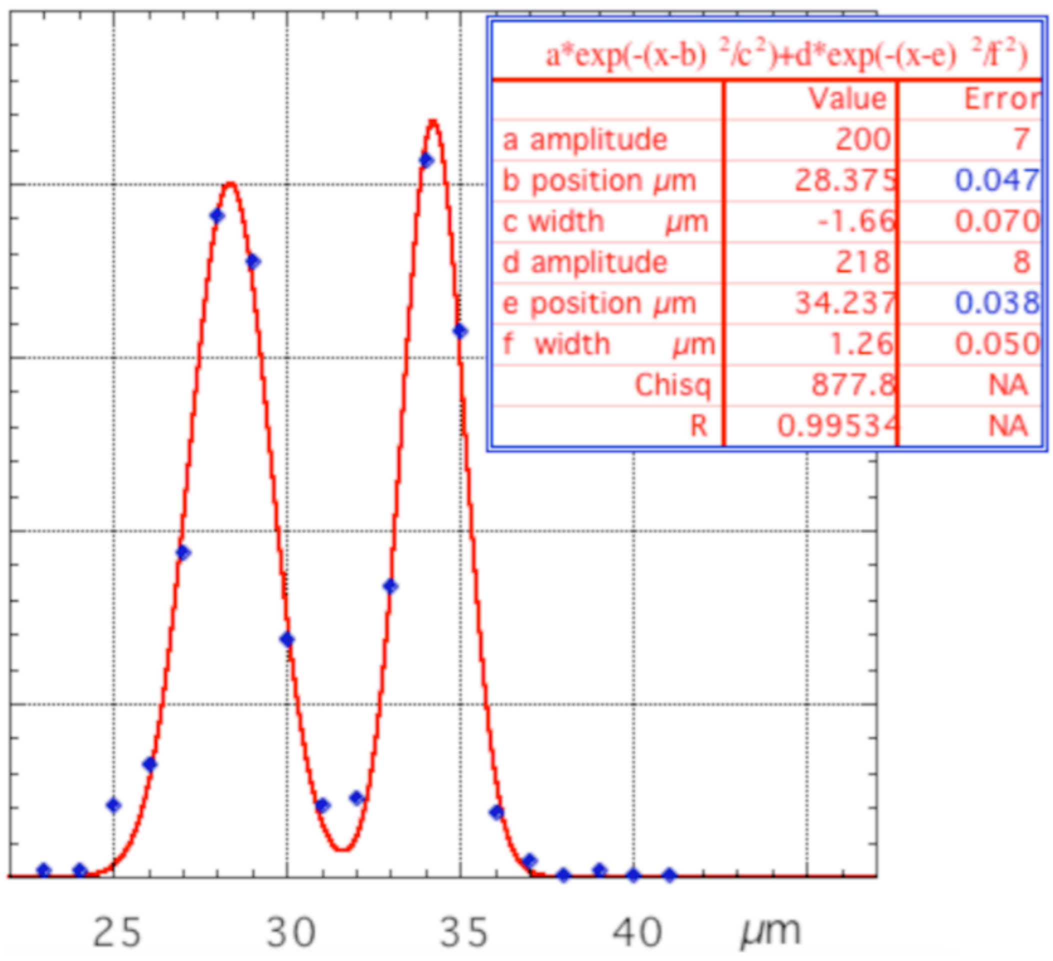


Figure 27. Zoom into a single bead with red line to show slice.



$a*\exp(-(x-b)^2/c^2)+d*\exp(-(x-e)^2/f^2)$	Value	Error
a amplitude	200	7
b position µm	28.375	0.047
c width µm	-1.66	0.070
d amplitude	218	8
e position µm	34.237	0.038
f width µm	1.26	0.050
Chisq	877.8	NA
R	0.99534	NA

Figure 28. Gaussian fit of the brightness across the slice, the fitting resolution of each Gaussian is indicated in blue.

To further enhance precision, the entire daisy in **Figure 27** can be fitted to a crown of 8 ellipsoidal Gaussians, see **Figure 29**. A daisy positioning precision better than 10 nm can be expected with a global daisy fit.

The USB microscope with its crown of eight LEDs coupled with 49 µm diameter beads proved capable of producing measurement precision in tens of nanometers. However, the USB microscope's acquisition rate is too slow in high definition mode and is not suitable for the measurement repetition rate required by tracking. A new camera, capable of faster high-definition video frame rate, and possibly better image quality, is necessary to implement high precision tracking with the 49 μm glass beads.

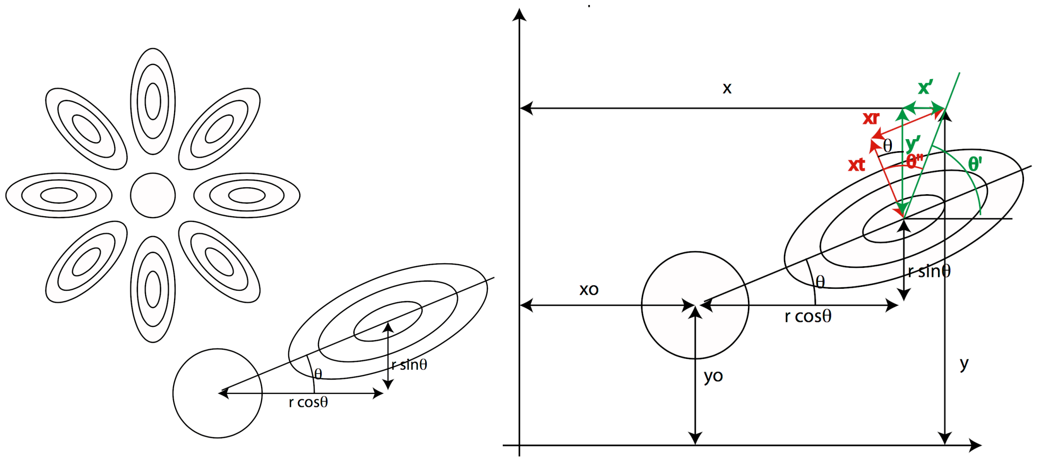


Figure 29. Illustration of fitting the eight ellipsoidal Gaussians of the daisy shape.

The aim of the experiment is to measure the loss angle of the flexure's material and its fluctuations, i.e. the movements of the gravitational sag of the end of the tube. What counts in determining the loss angle resolution is the transversal position resolution divided by the gravitational sag. A position resolution of 30 nm and a sag of 30 mm result in 1 micro-radian sensitivity, which should be compared with a typical metal loss angle of 100 micro-radian in a high-quality metal. The demonstrated single-daisy position resolution is sufficient to detect smaller than 1% fluctuations from the daisy circular orbit.

The demonstrated resolution is therefore adequate to detect the effects of sufficiently large dislocation avalanches, if the vibrations of the rod are reduced below the nm level. Ground vibrations are less than a μm above few Hz, the GAS filters easily attenuate to less than 1/1000 of ground vibration, and the Eddy current damper proved adequate to produce critical damping. Only internal movements of the frame-rod system are relevant. The main source of statistical and systematics error should be from acoustic couplings and thermal drifts, both mitigated by the thermal enclosure.

5.1.4 New Rod Design

A new rod attachment design was created with collet clamps that avoid the need of gluing and enable the use of the same rod to measure multiple flexures, thus avoiding using a new stepper motor and making a new rod for each material as required by the original design. This will speed-up and reduce costs for future data acquisition runs. The new rod design is shown in **Figure 30**.

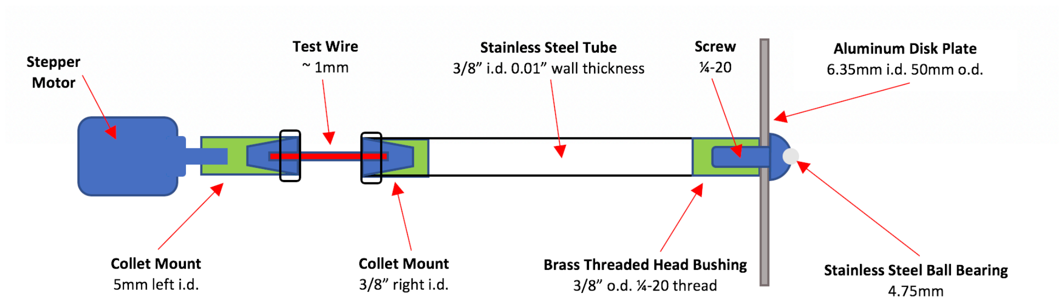

Stepper Motor
Test Wire ~ 1mm
Stainless Steel Tube 3/8" i.d. 0.01" wall thickness
Screw ¼-20
Aluminum Disk Plate 6.35mm i.d. 50mm o.d.
Collet Mount 5mm left i.d.
Collet Mount 3/8" right i.d.
Brass Threaded Head Bushing 3/8" o.d. ¼-20 thread
Stainless Steel Ball Bearing 4.75mm

Figure 30. Schematics of new rod design.

The clamping to the collet on the right side of the test wire connecting the wire to the tube is currently being machined to remove unnecessary mass.

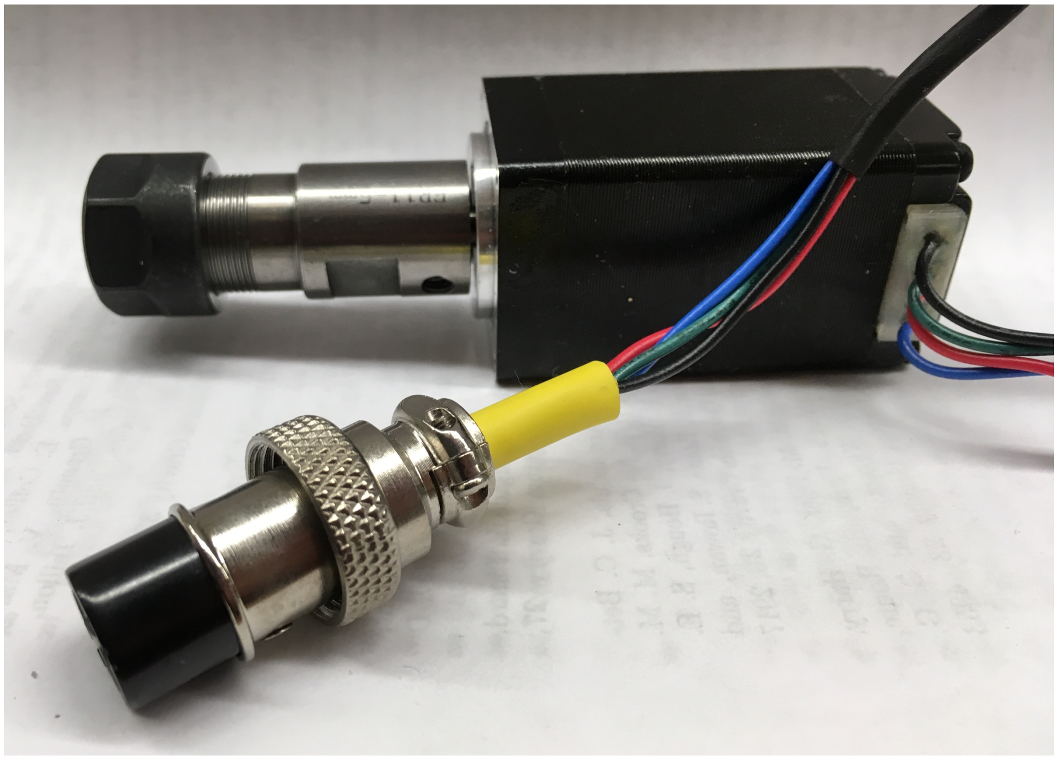


Figure 31. Image of the motor with a collet mount attached to the shaft.

5.1.5 Eddy Current Brake Modifications

The eddy current damper has been modified to allow for more freedom of transversal motion. Due to poor sample straightness, the copper disk made contact with the nuts separating the disks as data was taken. This contact is clearly seen in the raw rotation data shown on the left of **Figure 32**. One particular point of contact is indicated by a green ellipse encircling the region. On the right is a new set of raw data with the new configuration. In the new configuration a stainless steel tube used as a spacer to replace the nuts. With this improvement, the contact with the frame is removed. Another improvement to the eddy current brake is the testing of aluminum versus copper to determine the best material for the brake and then to cut shorter segments to decrease the spacing between the magnets to improve the damping.

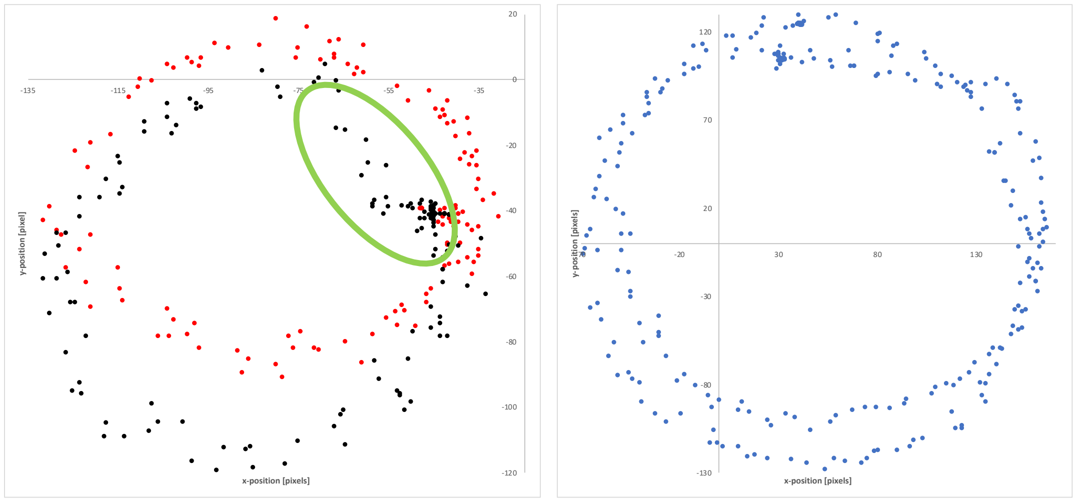


Figure 32. Raw 0.5 Hz data from the experiment (left) and new set of raw data (right).

5.1.6 Maraging Wire Design

Rather than simply cutting out a section of a wire in the case of our piano wire sample, a design was made for the maraging steel wires as shown in **Figure 33**.

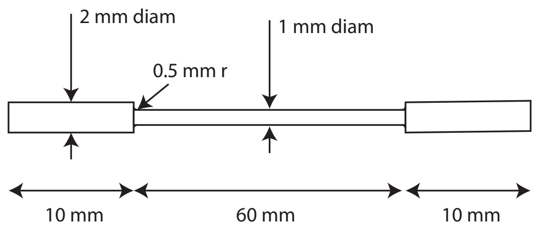


Figure 33. New maraging wire design.

5.1.7 Tracking program

A custom tracking program is in development. Presently it uses Matlab to take in a template of a daisy reflection in a glass bead, specified by the user, and track a chosen bead through successive images using correlation thresholds. Currently the program had a 74% success rate with a 94% correlation threshold using stop motion tracking of a single daisy before misreadings begin to appear. The capability of identifying a constellation of daisies to make sure that the same daisy is tracked is being added. After positive identification of the chosen daisy a new program would fit with the crown of eight ellipsoidal Gaussians get the position resolution. It is planned to move away from Matlab and toward a faster programming language. Machine learning could be incorporated to allow identification of daisies without inputting a template. For details on the method of testing the current program see **Appendix B**.

5.2 Glassy Metals as a Possible Replacement Material

Glassy metals have been considered to replace maraging steel in suspension blades for several reasons:

- Metals alloys in glassy form have double the elasticity range of the same alloy in polycrystalline form and can be loaded with twice the elastic potential energy.
- Glassy metals are particularly resistant to corrosion [21].
- Glassy metals have no crystalline structure and therefore no dislocations.

Therefore, they are free of all dislocation-mediated loss mechanisms, both the traditional Granato-Lueck and the anomalous mechanism investigated here. While glassy metals have a number of different drawbacks, they are obvious materials of interest for suspensions.

Work was completed at Montana Instruments over the summer of 2017 comparing the low-frequency performance of maraging and LM105 cantilever blade springs. As already stated, virtually all cantilever blade springs used in gravitational-wave detectors for seismic attenuation are made with maraging steel [8]. Geometric anti-spring filters made from this material and designed to reach arbitrarily low resonant frequencies have shown instabilities that impede their tuning below 0.5 Hz [8]. The instability and low frequency noise can be attributed to self-organized criticality effects within the material. Amorphous materials are free of dislocations and thus a hopeful candidate for removing the low frequency instability. The low frequency behavior was tested for the maraging steel blades alongside the vitreloy 105 blades to identify a possible replacement for the maraging steel blades.

5.2.1 The Goal

The research and development goal for this project was to test the low frequency behavior of commercially produced LM105 blades alongside maraging steel blades to determine if this material removes the low frequency instability noise in the GAS filter set-up. If fruitful, this research will provide a replacement material to produce a new generation of sensitive devices able to detect gravitational signals from more massive black-hole mergers.

A proposed application, should this research produce a positive result, is the implementation of glassy fibers in experiments measuring the gravitational constant. Metallic tungsten suspension wires are currently used because of their ability to ground all stray charges and null electrostatic forces; however, these wires are subject to unpredictable fluctuations with 1/f amplitude fluctuations due to avalanches within the suspension wires [5] [16]. The frequency of the events is strongly dependent on the velocity of the oscillation, its amplitude and the recent oscillatory history of the sample [8]. Self-organized criticality is predicted to change the slope of the flexure and therefore the equilibrium point of the oscillation, which would then be the likely culprit behind the experiment discrepancy of ±500 ppm [6].

5.2.2 Results

This test made over the summer was a failure as the commercially manufactured LM105 blades shattered. Work is being done with the company to determine if there is a way to manufacture the blades with the desired specifications.

Should glassy metals prove to remove the low-frequency instability in metal flexures, this material would be a commercially available option for future implementation.

The following sections are a discussion on the experimental set-up, why the blades may have broken, the analysis done on the maraging blades and a preliminary analysis of the LM105 blades before they shattered, and finally what may be expected in future trials.

5.2.3 Experimental Set-up

In order to compare the material to the maraging steel blades currently in use in many of the GAS filters in use, the LM105 vitreloy blades were manufactured to match the maraging steel blade shape and configuration. Apart from the material, the only difference between the two blade sets were the length and thickness of the blades imposed by the production process at the samples available at the commercial level. The final blades were 2.3 mm shorter than the maraging steel blades, still using the same set-up as shown in **Figure 34**.

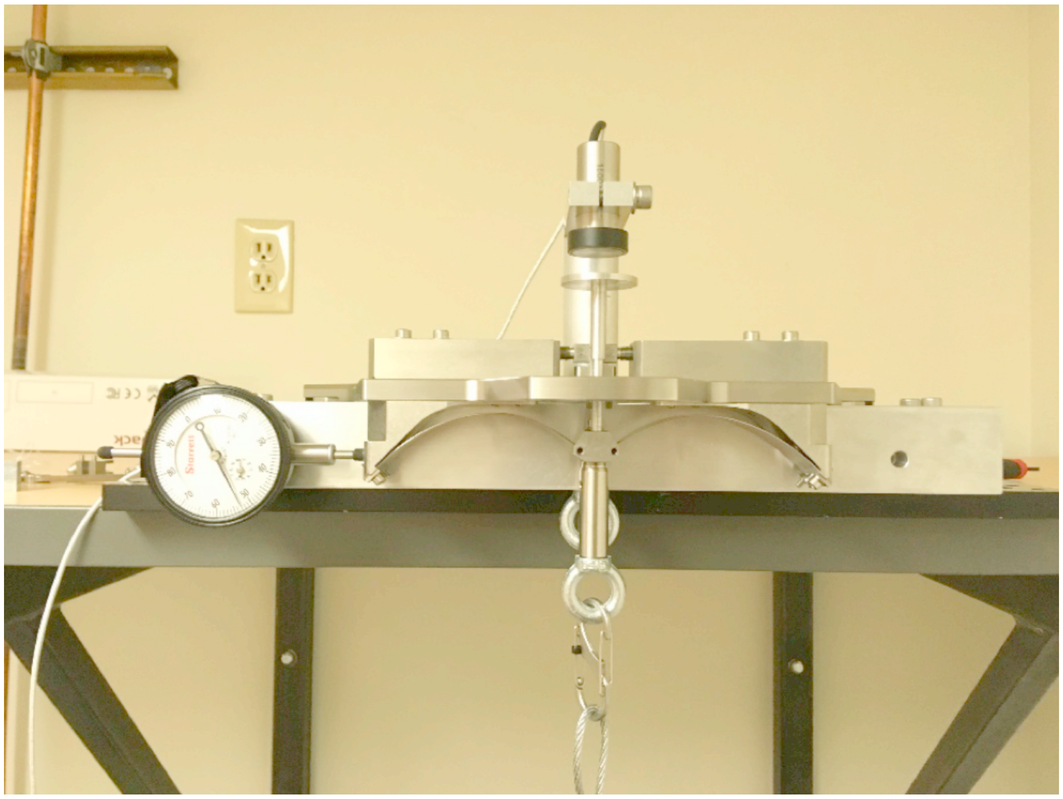


Figure 34. Cantilever blade configuration for testing both the maraging steel and vitrelloy 105 blade sets.

Once mounted, the set-up was attached to the table as shown in **Figure 34** and a mass was suspended at the center to bring the blades to float below the end-stop. The tuning process utilizes: a position gauge to determine the compression of the blade set (appearing to the left of the fixture in **Figure 34**), a capacitive sensor to measure the vertical position, an adjustable mass to change the load, and python script to control the electronics responsible for adjusting the system and data acquisition. This set-up automatically loads the blade set in incremental mass steps and applies an impulse to the fixture at each step to acquire ring down data at each load level. Both the maraging steel and the thinner set of LM105 blades were tested using the fixture shown above. Due to the size difference of the LM105 blades, the fixture could not tune the glassy blades past

1.4 Hz and thus spacers were designed by Caleb Schreibeis to allow for further compression of the blade set. Unfortunately, the blades shattered in the dismounting process, so this additional compression could not be tested. The design shown in **Figure 35** is ready for future testing of blades with shorter length. New glassy metal blades are currently being manufactured.

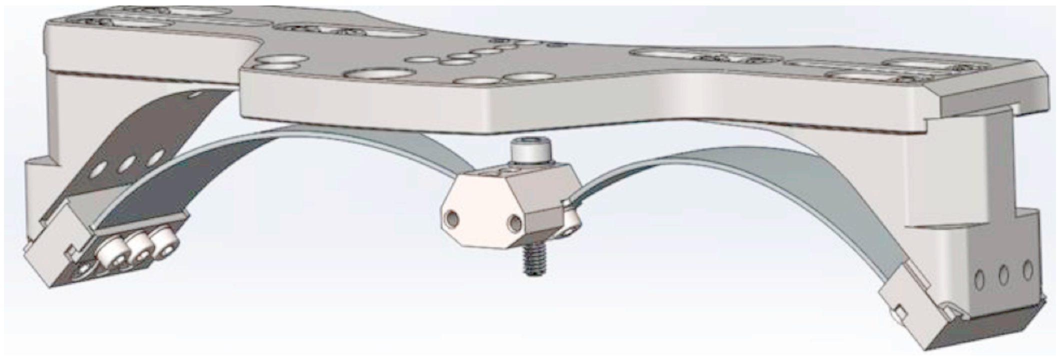


Figure 35. Cantilever blade set-up in geometric anti-spring configuration. The addition of spacers to the fixture to allow further compression of the LM105 cantilever blades.

5.2.4 Why the Glassy Metal Blades Broke

The LM105 material proved more brittle than previously tested glassy materials (a GAS spring built years ago is still operational). The expected fracture toughness (K_{1C}) should have been in the central region of the black ellipse drawn on **Figure 36** (around 150 MPa m$^{1/2}$) and should have been comparable to maraging steel in both fracture toughness and yield strength (σ_y). Thus, it should not have broken at the stress level imposed by the experiment.

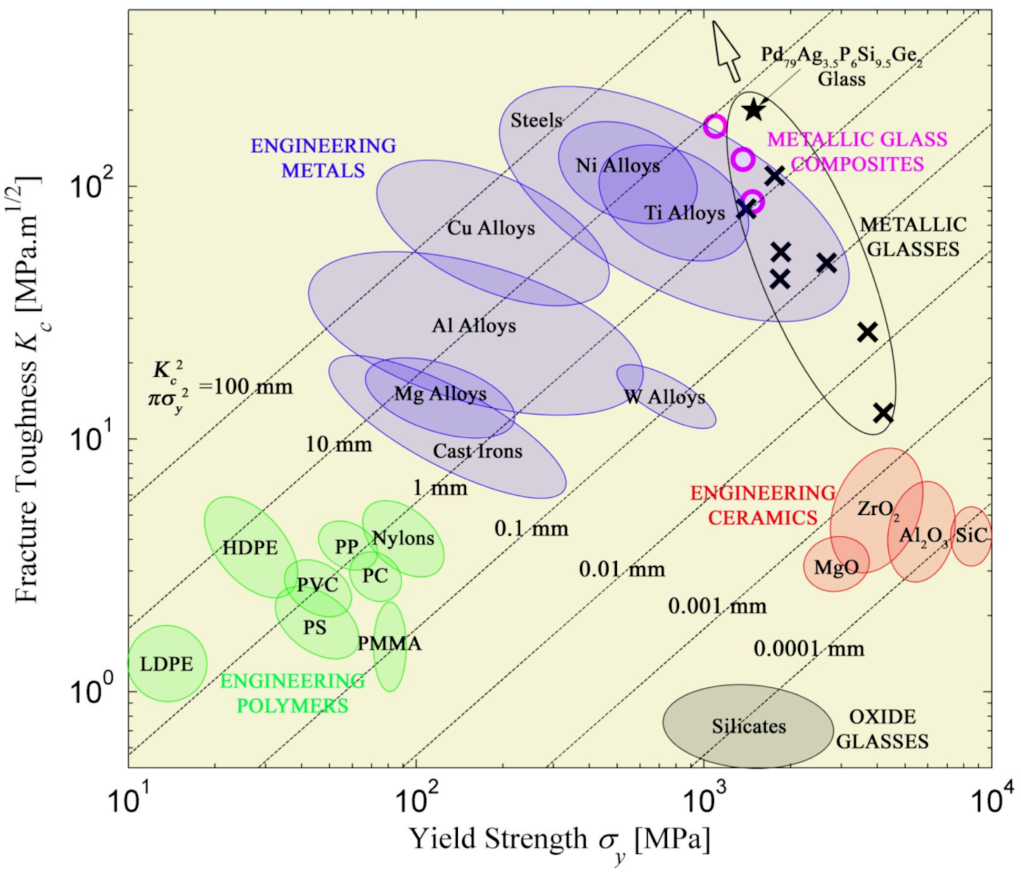


Figure 36. Plot of the fracture toughness versus the yield strength of many commonly used materials. The area of interest lies within the metallic glasses within the ellipse drawn in black [11].

Since the two samples of different thickness shattered, a direct comparison with maraging was not possible, however, a limited set of data was obtained with the second, thinner set of blades providing insight for future testing.

During the break, the blades gave off a flash of visible light due to surface oxidation ignited the newly exposed surface ignited by the sudden transient of stress concentration typical of amorphous metals. There are two possibilities we could think of as to why the blades shattered. First, the LM105 purchased was not good quality (the fracture toughness was much less than predicted).

$$R_{\text{critical}} = \frac{K_{1C}^2}{\pi\sigma^2} \quad (5.1)$$

Secondly, impurities (i.e. crystals, cracks, pockets of gas) formed during the casting process were larger than R_{critical} (the smallest a defect can be before causing catastrophic failure) and caused the material to shatter before reaching its yield point.

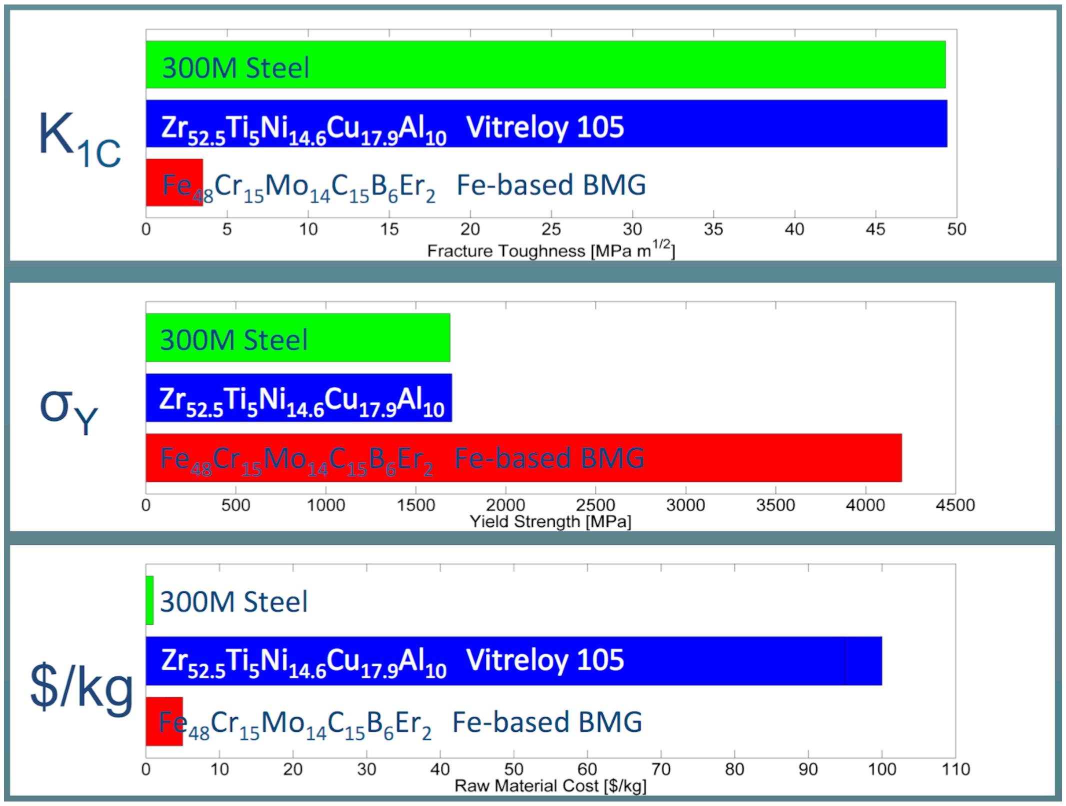


Figure 37. Comparison of the fracture toughness, yield strength, and cost of maraging steel to vitreloy 105 [10].

5.2.5 Q-Factor Analysis

5.2.5.1 Measurements of the Control Maraging Steel Blades

A large set of Q-Factor analysis were completed on a number of maraging steel blade sets.

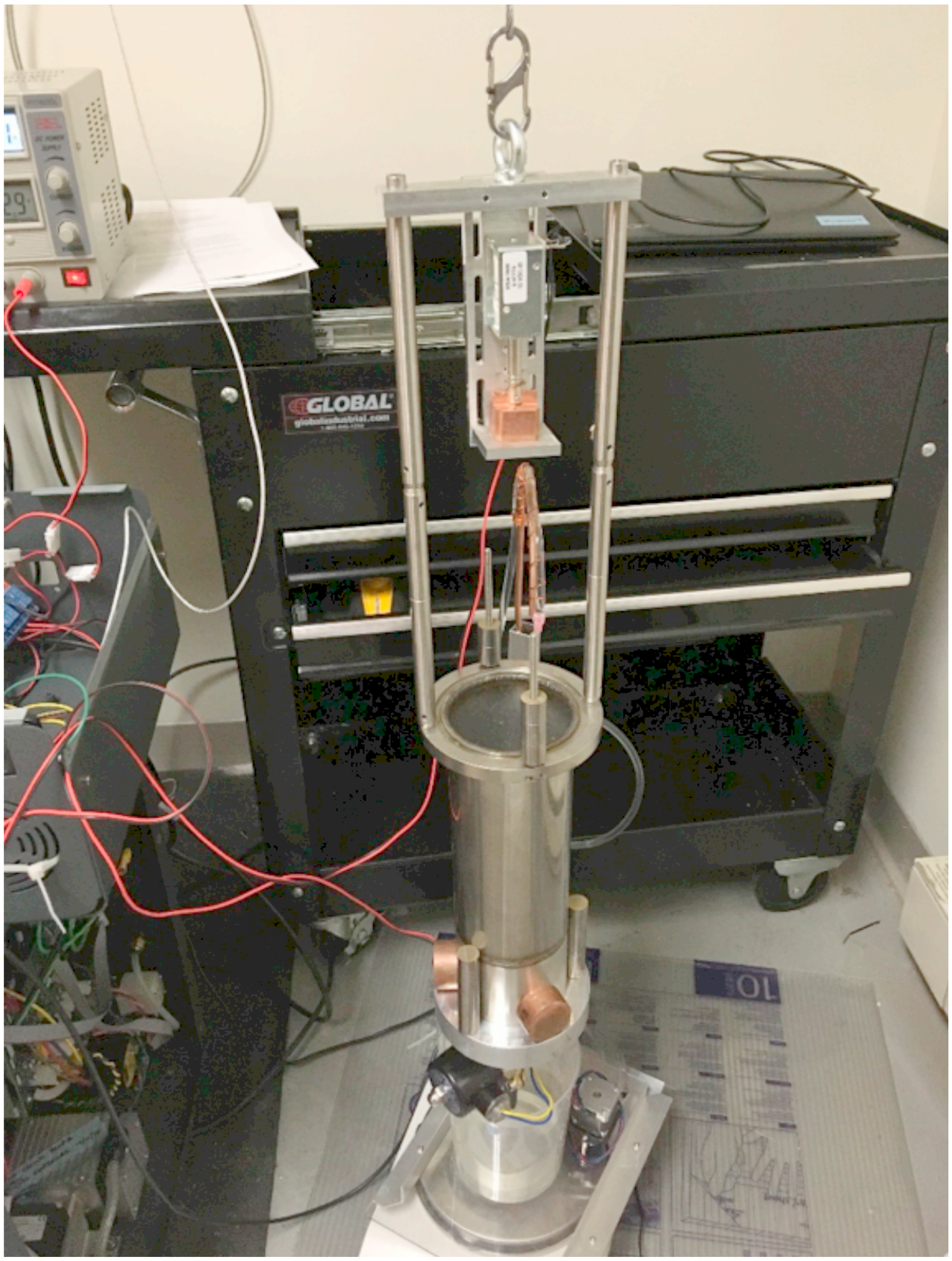


Figure 38. Image of the longer of the two suspended masses.

Tuning was completed using two separate configurations as the suspended mass (see **Figure 38**). The suspended payload behaves like a pendulum whose resonance

(**Equation 5.2**) renders sections of the data useless due to the appearance of beat frequencies around the pendulum's resonant frequency, see sample data set in **Figure 39**.

$$\frac{1}{2\pi}\sqrt{\frac{g}{L}} \tag{5.2}$$

Where the length of the pendulum, L, changes the region of coupling between the pendulum and the suspended mass. A shorter pendulum and a longer pendulum were used to fill in the areas of lost data and the complete set of data is shown in **Figure 40**.

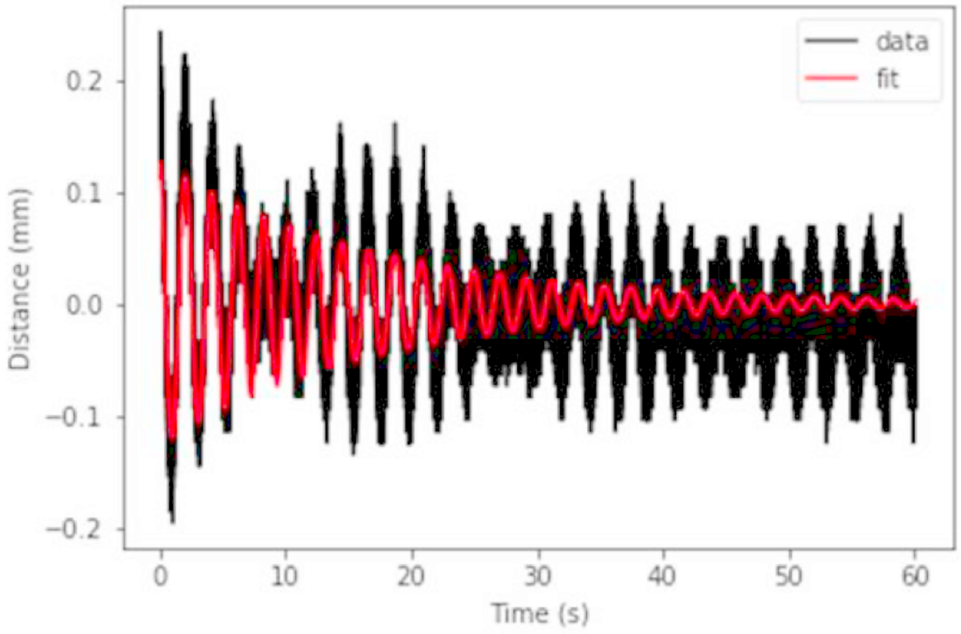


Figure 39. Beat frequencies appearing in the raw data where the pendulum's frequency couples with the frequency of the cantilever blade spring.

The expected behavior is shown when tuning the blades. The blades could only be tuned to roughly 0.43 Hz. A run using an electromagnetic voice coil for tuning the blades at lower frequency was performed and the system was able to be tuned to 0.38 Hz, which was the lowest frequency achieved during this investigation. More work is foreseen using this method to study the behavior at even lower frequencies.

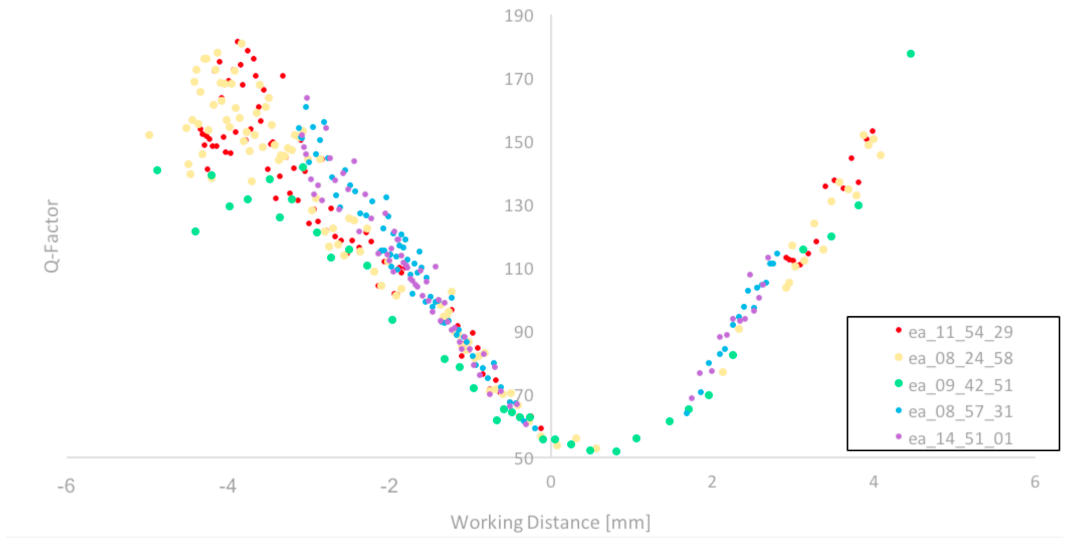


Figure 40. Data removed due to the appearance of beats at frequencies around the resonant frequency of the suspended mass, in green and yellow is the data taken with a shorter pendulum configuration.

5.2.5.2 Measurements of the Glassy Metal Blades

The data shown in **Figure 41** was taken when the blades were tuned to the largest radial compression allowed by the apparatus, which limited the lowest possible resonant frequency to a frequency of 1.4 Hz. Spacers were machined to allow for more compression as discussed previously, however, they remain untested since the blades shattered during disassembly.

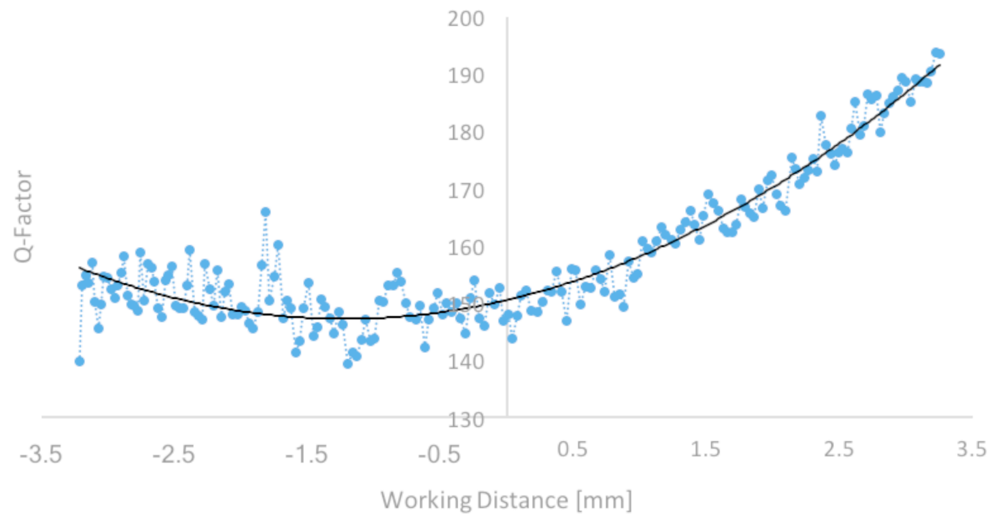


Figure 41. Q-Factor data from LM105 cantilever blade set.

Despite the interrupted experiment, an interesting phenomenon was observed in the Q-Factor. As the pay load increases, the resonant frequency first decreases and then increases again as expected. The same Q-Factor is expected at the same frequency, above or below the minimum, which is observed is not observed in either the maraging or with this sample of glassy metals, see **Figure 42** and **Figure 43**. The reason for this difference is still unknown, but it is suspected that the inertial change as mass is added may have some role here.

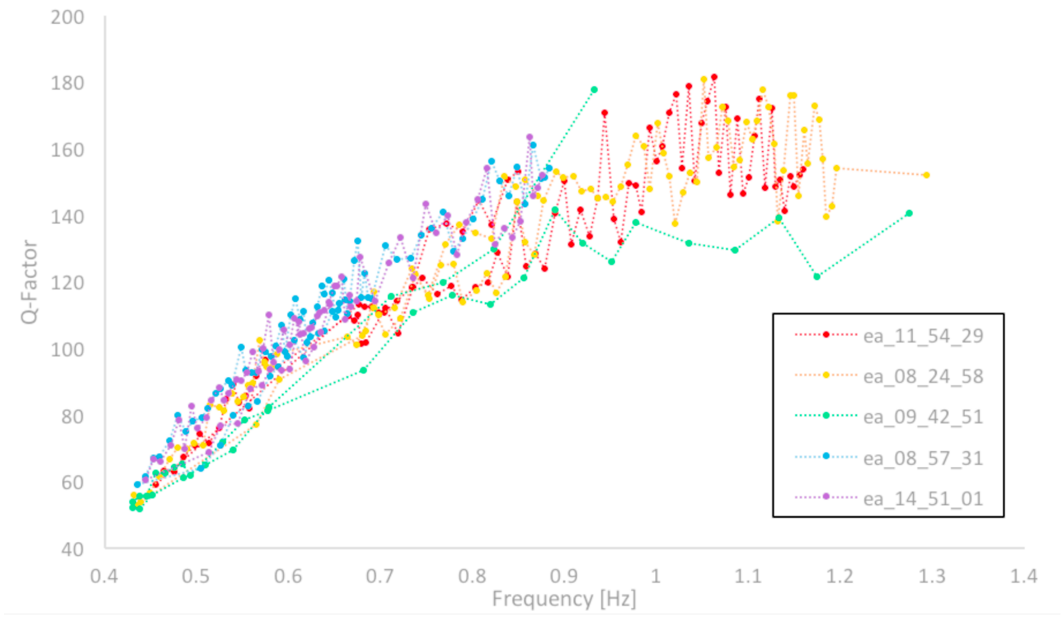


Figure 42. Q-Factor data from maraging steel cantilever blade set.

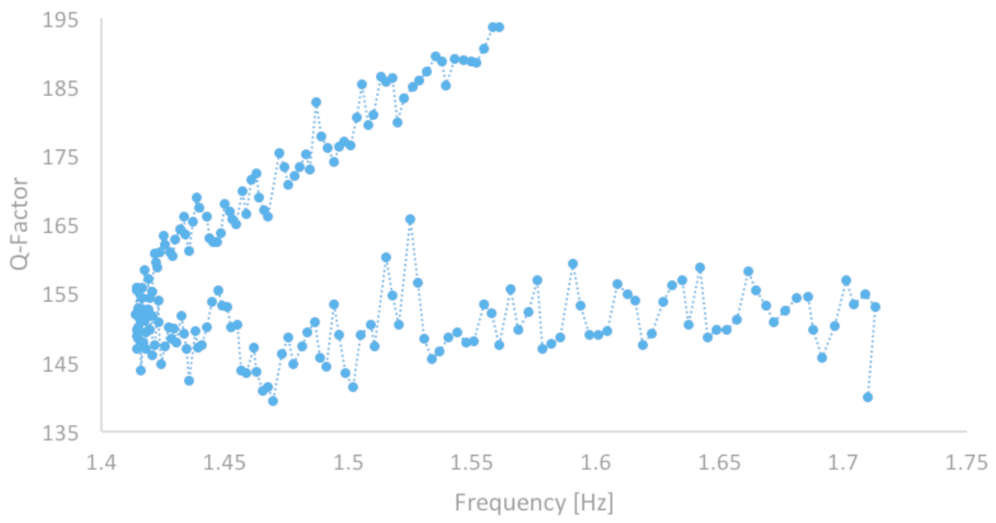


Figure 43. Q-Factor data from LM105 cantilever blade set.

5.2.6 Future Development of the Montana Instrument Test Set-Up

A PID loop was optimized for the tuning of the blades using an electromagnetic voice coil and it was able to tune the blades to frequencies below those achieved with the Reynold's Fixture alone. Once fully optimized and integrated this method will allow the exploration of the behavior of the Q-Factor at even lower frequencies, enriching the study already completed here.

The two remaining blades were taken back to the grinding company and ground to a smaller thickness. Tests were performed at lower load, but the data has yet to be analyzed. While the thinner blades do not carry the desired load, the measurements may give insight into the material's low frequency behavior.

Different kinds of glassy metals, including lab grade vitrelloy 105 are being considered.

CHAPTER 6

Conclusion

The first part of this thesis covers the development of an experiment to study low frequency mechanical noises from dislocation avalanches, if they exist. This experiment is designed to measure the loss angle of materials at arbitrarily low frequencies. While the experiment development is still far from complete, it has already successfully measured the loss angle of piano wire to within 5% of the value measured on the sample with traditional ring-down methods.

It is expected that if dislocation entanglement happens, larger loss angle and sudden deviations of the equilibrium point of the flexure will appear during the rod's rotation at very low frequencies [9]. In order to measure these deviations, further improvements need to be made to the experiment to remove excess noise and the precision of position measurements.

A second part of this thesis tests on the use of glassy metal flexures as a possible dislocation-free alternative. Should the theory of SOC prove to be valid, a new non-crystalline material needs to be developed in order to remove the 1/f noise from high sensitivity instruments using metal flexures. Self-organized criticality is believed to be present in all polycrystalline metals; therefore, one solution is to use a glassy metal. Initial tests using glassy metals gave discouraging results and a thinner set of blades has been prepared for future tests. The method for studying these materials at low frequency has been developed. Future work may need to wait for glassy metal technologies to improve.

REFERENCES

[1] R. DeSalvo, "Unaccounted source of systematic errors in mesurements of the Newtonian gravitational constant G," *Phys. Lett.,* pp. 379, 1202-1205, 2015.

[2] R. DeSalvo, A. DiCinGo and M. Lundin, "The role of Self-Organized Criticality in elasticity of metalic springs: Observations of a new dissipation regime," *Eur. Phys. J.,* p. 58, 2011.

[3] N. Virdone, J. Agresti, A. Bertolini, R. DeSalvo and R. Stellac, "Extended timescale creep measurement on Maraging cantilever blade springs," *Nuclear Instruments and Methods in Physics Research,* pp. 593, 597-607, 2008.

[4] R. DeSalvo, S. Marka, K. Numata, V. Sannibale, A. Takamori, H. Tariq, E. J. Ugas, T. Yoda, Y. Aso and A. Bertolini, "Study of quality factor and hysteresis associated with the state-of-the-art passive seismic isolation system for Gravitational Wave Interferometric Detectors," *Nuclear Instruments and Methods in Physics Research,* pp. 538, 526-537, 2005.

[5] C. Speake, T. Quinn, R. Davis and S. Richman, "Experiment and theory in anelasticity," *Meas. Sci. Technol.,* pp. 10(6), 430-434, 1999.

[6] P. Bak, C. Tang and K. Wiesenfeld, "Self-organized criticality," *Phys. Rev.,* pp. 38(1), 364, 1988.

[7] H. Cavendish, "Experiments to Determine the Density of Earth," *Philosophical Transcations of the Royal Society of London,* pp. Vol. 88, 469-526, 1798.

[8] C. Speake and T. Quinn, "The Search for Newton's constant," *Physics Today,* pp. 67(7), 27, 2014.

[9] C. Speake, "Newton's constant and the twenty-first century laboratory," *Phil. Trans. R. Soc.,* pp. 363, 1834, 2265-2287, 2005.

[10] P. Bak, C. Tang and K. Wiesenfeld, "Self-organized criticality," *Phys. Rev.,* pp. 38(1), 364, 1988.

[11] G. Cagnoli, L. Gammaitoni, F. Marchesoni and D. Segoloni, "On dislocation damping at low frequencies," *Philosophical Magazine,* pp. 68(5), 865-870, 1993.

[12] A. Kimball and D. Lovell, "Internal Friction in Solids," *Phys.,* 1927.

[13] Cagnoli, G., "Silica suspension and coating developements for Advanced LIGO," *Institute of Physics Publishing Journal of Physics: Conference Series,* pp. 32, 386-392, 2006.

[14] W. D. Callister and D. G. Rethwisch, Materials Science and Engineering: an Introduction, Wiley, 2014.

[15] "Trinamic Motion Control GmbH & Co. KG Hamburg, Germany," 4 December 2014. [Online]. Available: Hardware Version V1.00 TMC5130-EVAL EVALUATION BOARD MANUAL. [Accessed 25 April 2018].

[16] "QSH2818-51-07-012," Digi-Key Electronics, 19 October 2010. [Online]. Available: https://www.digikey.com/product-detail/en/trinamic-motion-control-gmbh/QSH2818-51-07-012/1460-1073-ND/4843424. [Accessed 25 April 2018].

[17] "Physics A-Level: Magnetic Fields and Induction," physbot: weebly, [Online]. Available: http://www.physbot.co.uk/magnetic-fields-and-induction.html. [Accessed 25 April 2018].

[18] "Eddy current brake," Wikimedia Foundation, 4 April 2018. [Online]. Available: https://en.wikipedia.org/wiki/Eddy_current_brake#/media/File:Eddy_current_brake_diagram.svg. [Accessed 25 April 2018].

[19] "HAM-SAS: Pictures and Movies," 24 December 2006. [Online]. Available: https://labcit.ligo.caltech.edu/~citsas/HAM-SAS/PicturesMovies/Filter-Assembly/filter-assembly-3.JPG. [Accessed 25 April 2018].

[20] R. DeSalvo, "Passive, Nonlinear, Mechanical Structures for Seismic Attenuation," *Journal of Computational and Nonlinear Dynamics,* 2007.

[21] A. Kass, "Thesis Title," California State University, Los Angeles, 2018.

[22] W. L. Johnson, "Towards a commercial metallic glass technology," in *TMS BMG-Symposium*, Nashville, February 2016.

[23] "Trinamic: Latest TMCL-IDE," 28 March 2018. [Online]. Available: https://www.trinamic.com/support/software/tmcl-ide/#c414 . [Accessed 25 April 2018].

APPENDIX A

Trinamic Stepper Motor

A.1. Running the Stepper Motor

The Trinamic software was downloaded from their website and can be found in reference [23]. The current version on the laptop in the lab is version 3.0 and was downloaded on June 30th 2017. To start the program, make sure that the stepper motor control board is connected to a power source, the motor and the computer before opening the application (the icon is shown in **Figure 45)**. If disconnected the program will not recognize the motor. As soon as the program opens the screen will look the same as the screen shown in **Figure 45**.

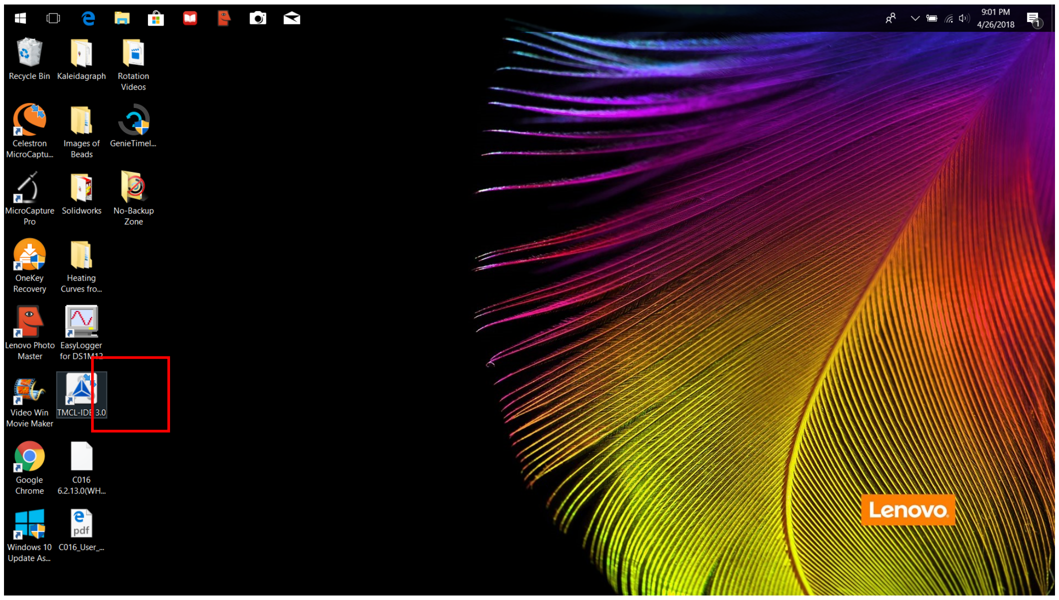
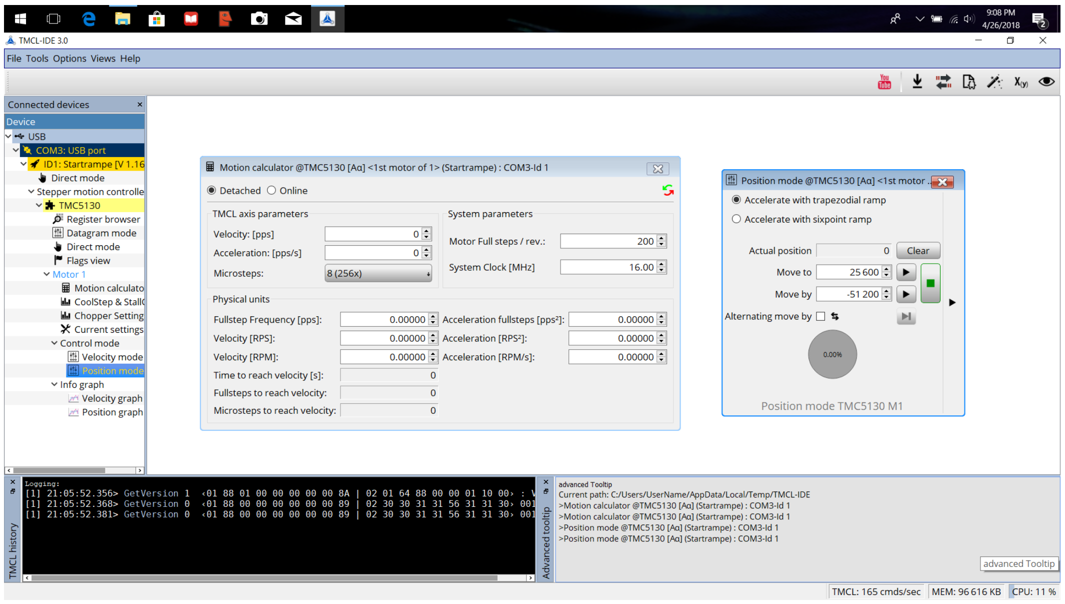


Figure 44. Desktop on the lab computer with the Trinamic software indicated with a red box.



Motion calculator @TMC5130 [Aα] <1st motor of 1> (Startrampe) : COM3-Id 1

(● Detached) ○ Online

TMCL axis parameters
Velocity: [pps] 0
Acceleration: [pps/s] 0
Microsteps: 8 (256x)

System parameters
Motor Full steps / rev.: 200
System Clock [MHz] 16.00

Physical units
Fullstep Frequency [pps]: 0.00000
Acceleration fullsteps [pps²]: 0.00000
Velocity [RPS]: 0.00000
Acceleration [RPS²]: 0.00000
Velocity [RPM]: 0.00000
Acceleration [RPM/s]: 0.00000
Time to reach velocity [s]: 0
Fullsteps to reach velocity: 0
Microsteps to reach velocity: 0

Position mode @TMC5130 [Aα] <1st motor ..

(● Accelerate with trapezodial ramp)
○ Accelerate with sixpoint ramp

Actual position 0 Clear
Move to 25 600
Move by -51 200
Alternating move by ☐
0.00%
Position mode TMC5130 M1

```
Logging:
[1] 21:05:52.356> GetVersion 1  ‹01 88 01 00 00 00 00 00 8A | 02 01 64 88 00 00 01 10 00› : V
[1] 21:05:52.368> GetVersion 0  ‹01 88 00 00 00 00 00 00 89 | 02 30 30 31 31 56 31 31 30› 001
[1] 21:05:52.381> GetVersion 0  ‹01 88 00 00 00 00 00 00 89 | 02 30 30 31 31 56 31 31 30› 001
```

```
advanced Tooltip
Current path: C:/Users/UserName/AppData/Local/Temp/TMCL-IDE
>Motion calculator @TMC5130 [Aα] (Startrampe) : COM3-Id 1
>Motion calculator @TMC5130 [Aα] (Startrampe) : COM3-Id 1
>Position mode @TMC5130 [Aα] (Startrampe) : COM3-Id 1
>Position mode @TMC5130 [Aα] (Startrampe) : COM3-Id 1
```

TMCL: 165 cmds/sec | MEM: 96 616 KB | CPU: 11 %

Figure 45. Start-up of the Trinamic stepper motor software. Both the Motion Calculator (left) and Position mode (right) windows open automatically.

At this point there are only three windows of interest to run the motor, however, the list of all possible windows can be found on the left hand side of the screen and a detail of this list can be found in **Figure 46**. The first window that should be used is the current settings window (see **Figure 47**). This window allows the user to edit the current supplied to the motor. The automatic current setting is 1.01 A. It was found that to currents as low as 0.08 A did not compromise the stepping accuracy of the motor (see **Appendix A.2**).

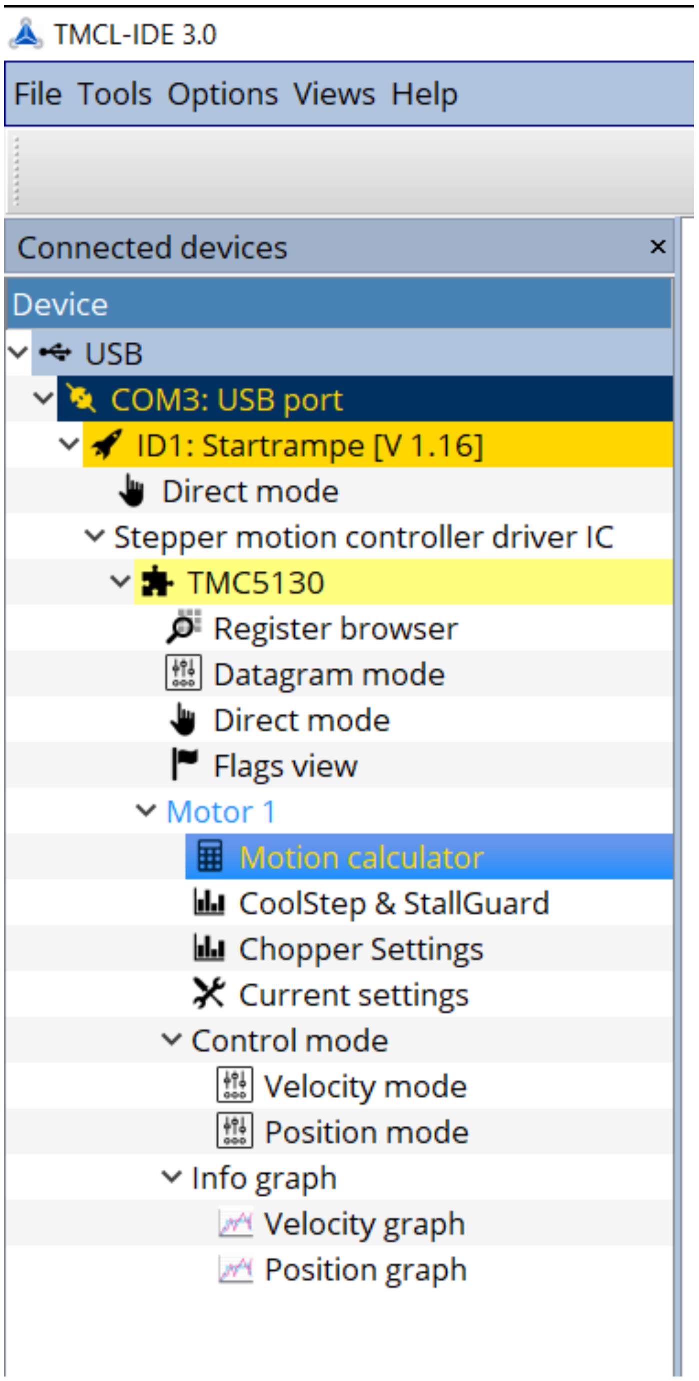
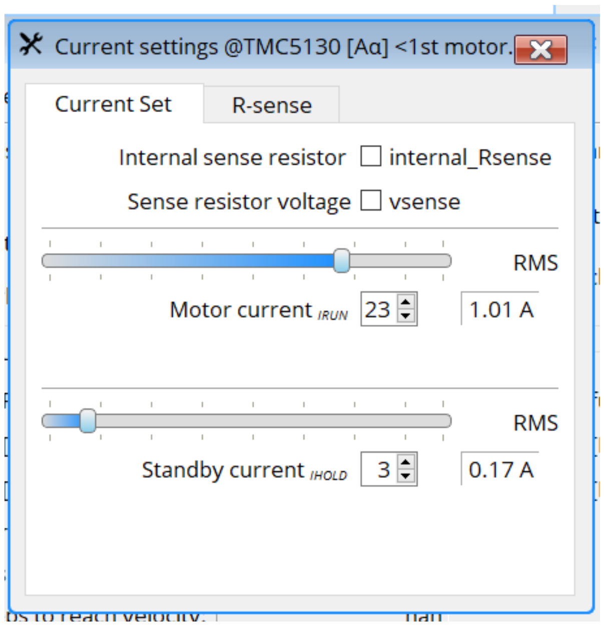


Figure 46. Detail of the option list on the left side of the window.



Current settings @TMC5130 [Aɑ] <1st motor.

Current Set | R-sense

Internal sense resistor ☐ internal_Rsense

Sense resistor voltage ☐ vsense

RMS

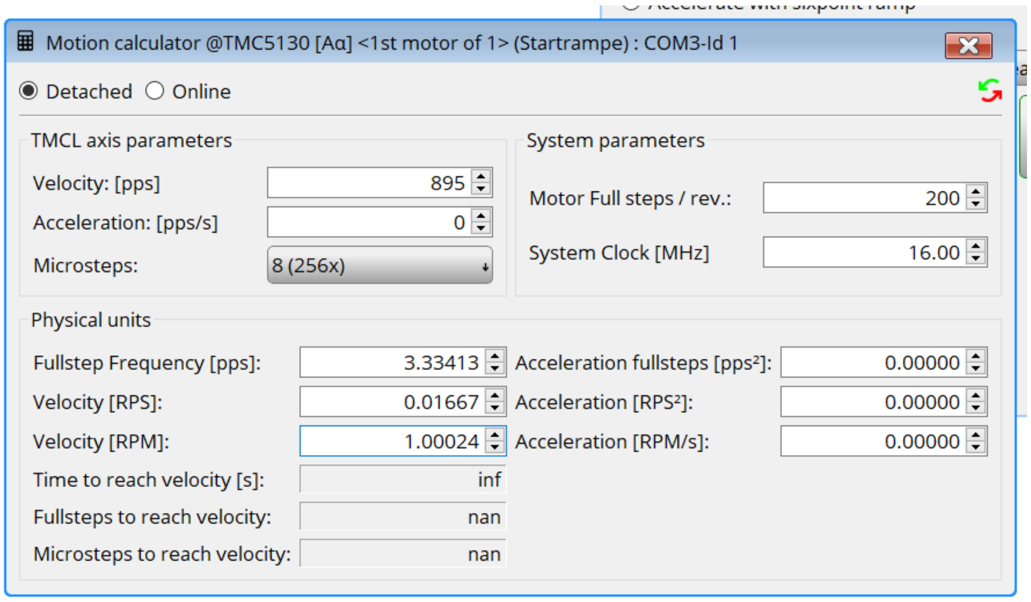
Motor current $_{IRUN}$ 23 | 1.01 A

RMS

Standby current $_{IHOLD}$ 3 | 0.17 A

Figure 47. Current settings window.

The next window of interest is the motion calculator window (see **Figure 48**). This window allows the user to calculate the velocity at different rotation speeds. A table of values used in this experiment can be found in **Table 3**.



Motion calculator @TMC5130 [Aɑ] <1st motor of 1> (Startrampe) : COM3-Id 1

● Detached ○ Online

TMCL axis parameters

Velocity: [pps]	895
Acceleration: [pps/s]	0
Microsteps:	8 (256x)

System parameters

Motor Full steps / rev.:	200
System Clock [MHz]	16.00

Physical units

Fullstep Frequency [pps]:	3.33413	Acceleration fullsteps [pps²]:	0.00000
Velocity [RPS]:	0.01667	Acceleration [RPS²]:	0.00000
Velocity [RPM]:	1.00024	Acceleration [RPM/s]:	0.00000
Time to reach velocity [s]:	inf		
Fullsteps to reach velocity:	nan		
Microsteps to reach velocity:	nan		

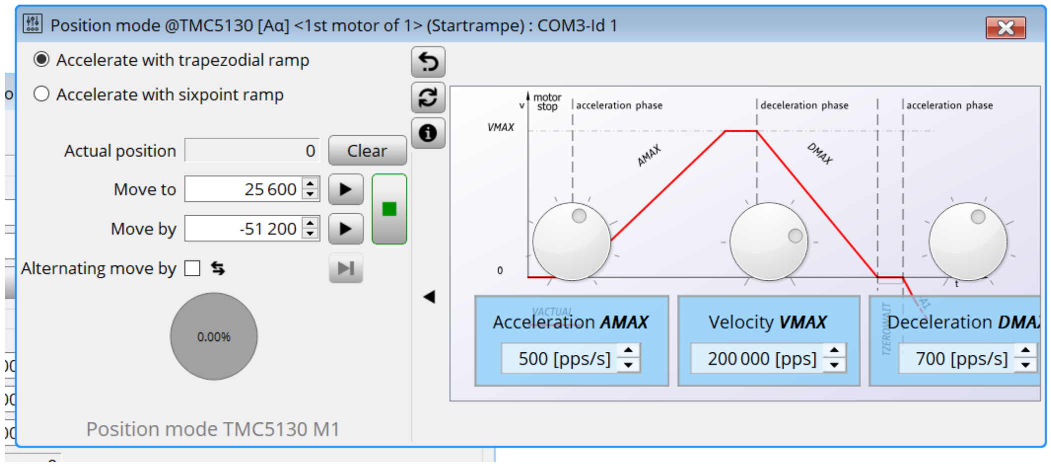
Figure 48. Detail of the motion calculator window.

Table 3

Velocities corresponding to each frequency run completed during the experiment.

Frequency [Hz]	R.P.M	Velocity [pps]
1	60	53687
0.75	45	40265
1	60	53687
0.5	30	26844
1	60	53687
0.4	24	21475
1	60	53687
0.3	18	16106
1	60	53687
0.2	12	10737
1	60	53687
0.1	6	5369
1	60	53687
0.05	3	2684
1	60	53687

After the correct velocity is calculated it must be entered into the position window before beginning the experiment. The position window can be found in **Figure 49**. The window is expanded here by clicking on the arrow appearing on the right of the window, here is appears in the center because it has been expanded. The velocity is automatically set to 200,000 pps and can be changed by deleting the value that appears in VMAX and typing in the correct velocity. Finally, to run the experiment the position must be changes in the move to section. For this data run, the rod was rotated through one repetition of five full rotations clockwise and five full rotations counter clockwise. Since one rotation would take the stepper motor from an actual position of 0 to 51,200, the value to enter for the first set of five clockwise rotations is 256,000. To complete the counterclockwise rotations, the motor would start at an actual position of 256,000 and thus the value to enter would be 0 to bring it back to the original position.


Position mode @TMC5130 [Aα] <1st motor of 1> (Startrampe) : COM3-Id 1

(●) Accelerate with trapezodial ramp
( ) Accelerate with sixpoint ramp

Actual position: 0
Move to: 25 600
Move by: -51 200
Alternating move by: ☐

Acceleration AMAX: 500 [pps/s]
Velocity VMAX: 200 000 [pps]
Deceleration DMAX: 700 [pps/s]

Figure 49. Detail of the position window.

A.2. Stepper Motor Testing

A series of tests were run on the stepper motor prior to acquiring data in order to learn more about how the motor heats up and whether or not steps were lost when running the motor at lower current. These tests were motivated by the desire to reduce the heating of the motor in order to reduce the heat transfer to the test wire. If this heat is transferred to the test wire a drift in the position will be seen which would interfere with any measurements taken.

The first of these two goals were achieved through the measurement of temperature while running the motor for 30 minutes and then shutting it off and watching the cooldown of the motor. An example of such a data run is shown in **Figure 50**.

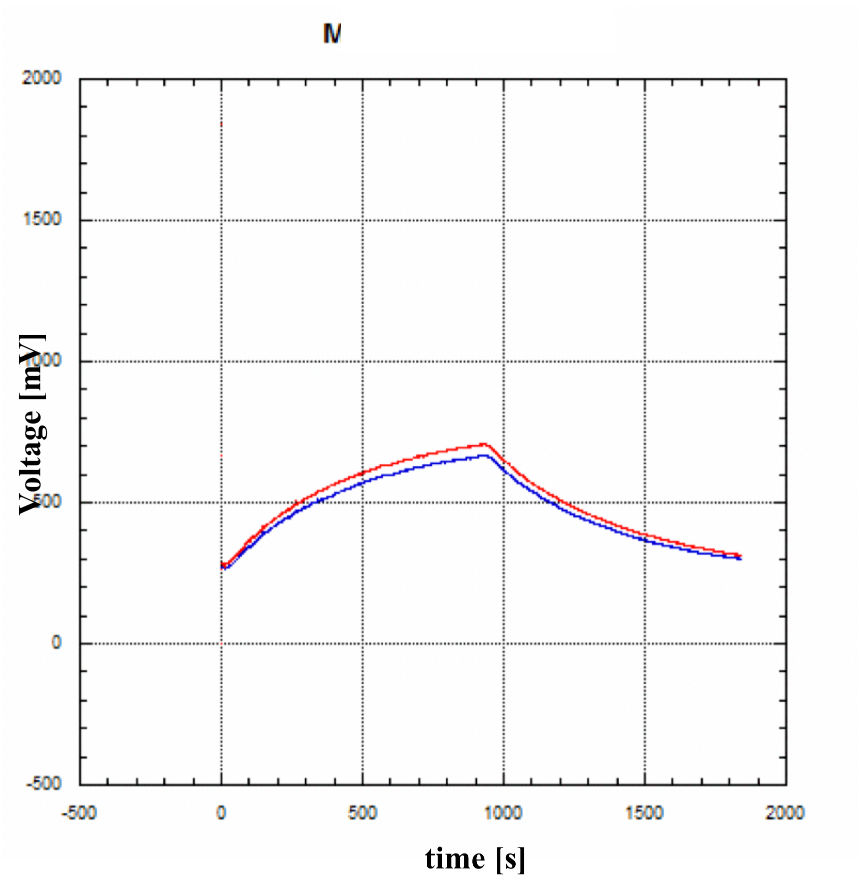
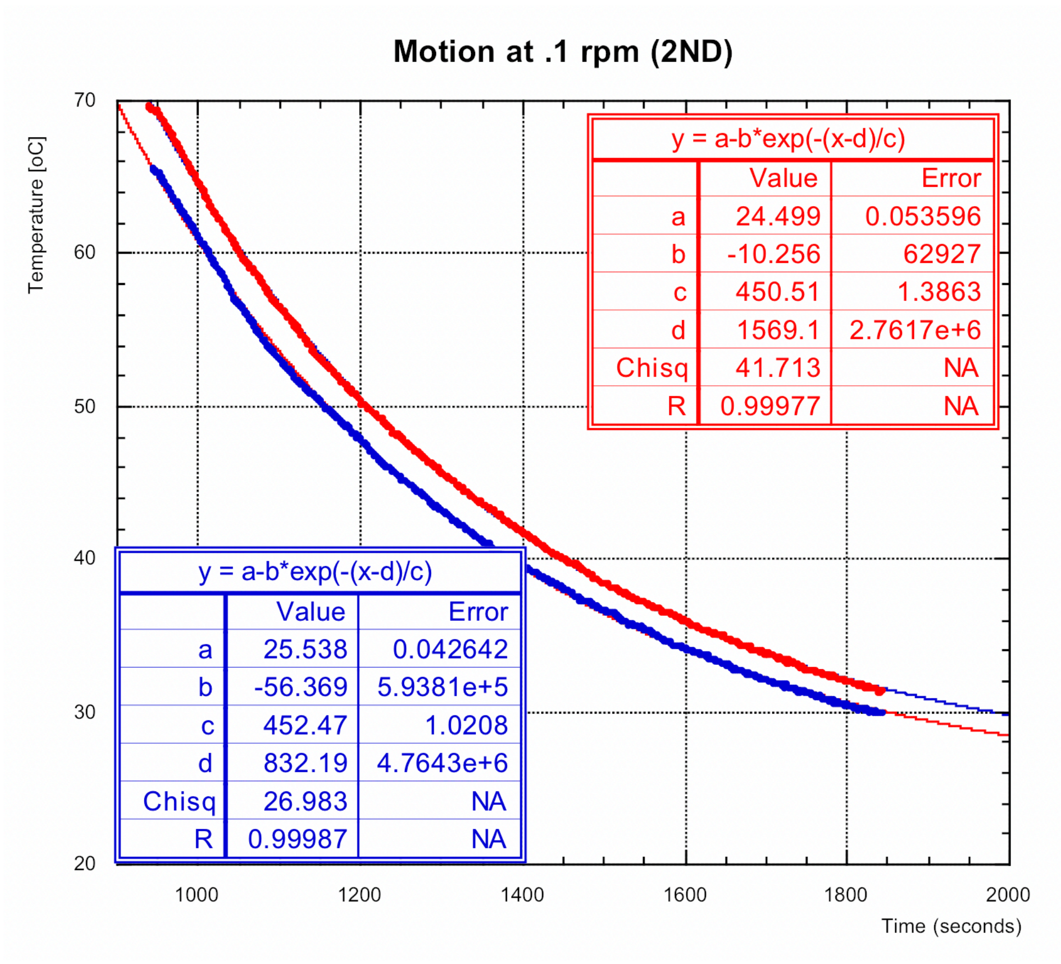


Figure 50. Voltage vs. time plot taken at 0.1 rotations per minute using two temperature sensors.

Both the rise in temperature while running the motor at a given rotational speed and the cooling of the motor after switching the motor off were fitted by separating the data at the shut off time. An example of the heating fit can be found in **Section 3.2** in **Figure 11** and the cooling can be found in **Figure 51**. The set of results can be found in **Table 4**.



	Value	Error
a	24.499	0.053596
b	-10.256	62927
c	450.51	1.3863
d	1569.1	2.7617e+6
Chisq	41.713	NA
R	0.99977	NA

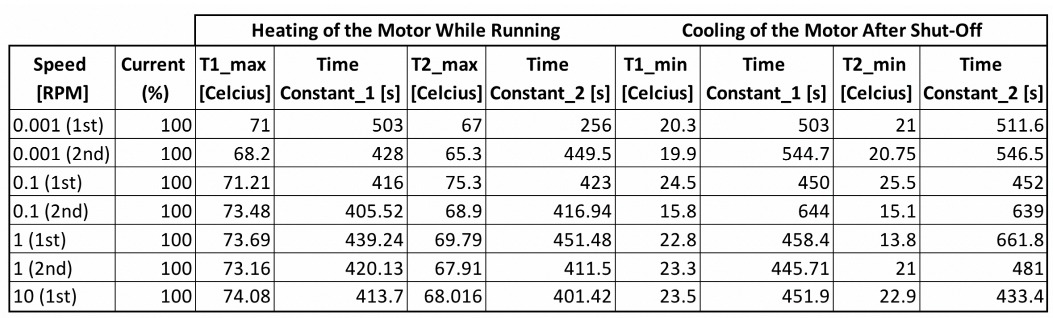
	Value	Error
a	25.538	0.042642
b	-56.369	5.9381e+5
c	452.47	1.0208
d	832.19	4.7643e+6
Chisq	26.983	NA
R	0.99987	NA

Figure 51. Fit to the cooling of the motor after shut-off. Plot and fit colors are reversed (i.e. Red data corresponds to the blue fit parameters).

In the cooling portion of the data the minimum temperature value approaches 24.5 ± 0.05 ^{o}C with the first temperature sensor and to 25.54 ± 0.04 ^{o}C with the second temperature sensor. The reason for the difference between the two sensors is thought to be from damage due to overheating the sensors in a previous experiment. The motor should thermalize between 37.5 and 37.7 minutes. These time frames are found by multiplying the time constant in seconds by five and then dividing by 60 to obtain the time for thermalization in minutes. With this information it can be expected that drift will appear in measurements within the first 35 minutes of running and it would be prudent to wait for 40 minutes before running a new set of data in order to ensure that enough time has passed to remove this drift.

Table 4

Results of testing how the motor heats at different rotation speeds.



		Heating of the Motor While Running				Cooling of the Motor After Shut-Off			
Speed [RPM]	Current (%)	T1_max [Celcius]	Time Constant_1 [s]	T2_max [Celcius]	Time Constant_2 [s]	T1_min [Celcius]	Time Constant_1 [s]	T2_min [Celcius]	Time Constant_2 [s]
0.001 (1st)	100	71	503	67	256	20.3	503	21	511.6
0.001 (2nd)	100	68.2	428	65.3	449.5	19.9	544.7	20.75	546.5
0.1 (1st)	100	71.21	416	75.3	423	24.5	450	25.5	452
0.1 (2nd)	100	73.48	405.52	68.9	416.94	15.8	644	15.1	639
1 (1st)	100	73.69	439.24	69.79	451.48	22.8	458.4	13.8	661.8
1 (2nd)	100	73.16	420.13	67.91	411.5	23.3	445.71	21	481
10 (1st)	100	74.08	413.7	68.016	401.42	23.5	451.9	22.9	433.4

It is apparent from the table of results that a change in rotation speed does not seem to have a large effect on the increase nor the decrease in temperature. The difference in time constants between the sensors might be due to the points of contact. Temperature data was also taken for a series of rotation speeds at 0.51 A, 0.25 A, 0.17 A, and 0.08 A to determine if the motor's peak temperature reduced with a reduction in current. Although none of the data has been analyzed, it was noted that the maximum temperature was roughly the same in all cases.

The second goal of determining the current at which the motor loses stepping accuracy was completed using a piece of tape fixed to the bras mount connecting the motor to the rod. This test determined that there was no loss in stepping accuracy at currents as low as 0.08 A. In the future it would be prudent to rerun this test in a more quantifiable way to determine if this is actually the case.

APPENDIX B

The Tracking Program and Testing

This section will first cover how to run the “Tracker” program used to track the 4.75 mm ball. Then the current state of the program being developed in the lab will be discussed, followed by the testing being done on the code to improve it.

B.1. Running the “Tracker” Program

The “Tracker” program is relatively easy to use, however, there are many pitfalls for someone starting out for the first time. To start the program, select the “Tracker” icon (see **Figure 52)** in the applications folder on your desktop. This will open the program an present a screen identical to the one shown in **Figure 53**.

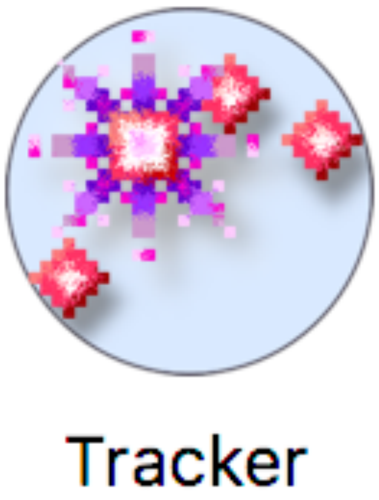
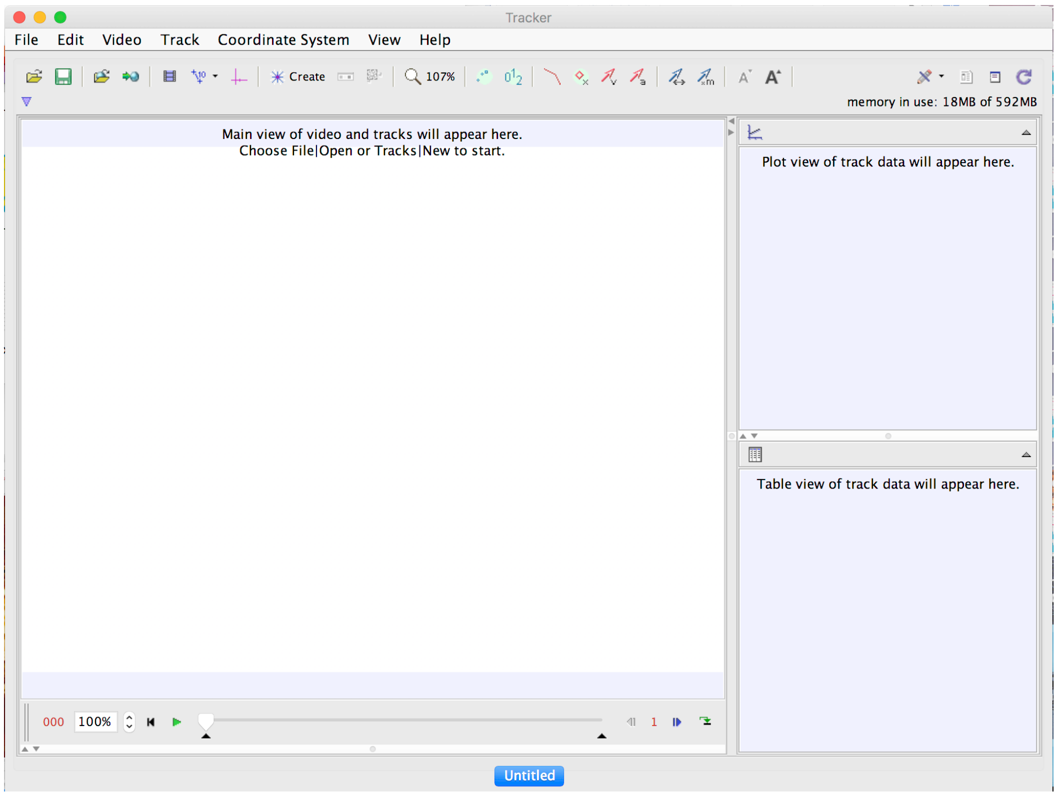


Figure 52. Icon for the "Tracker" program.



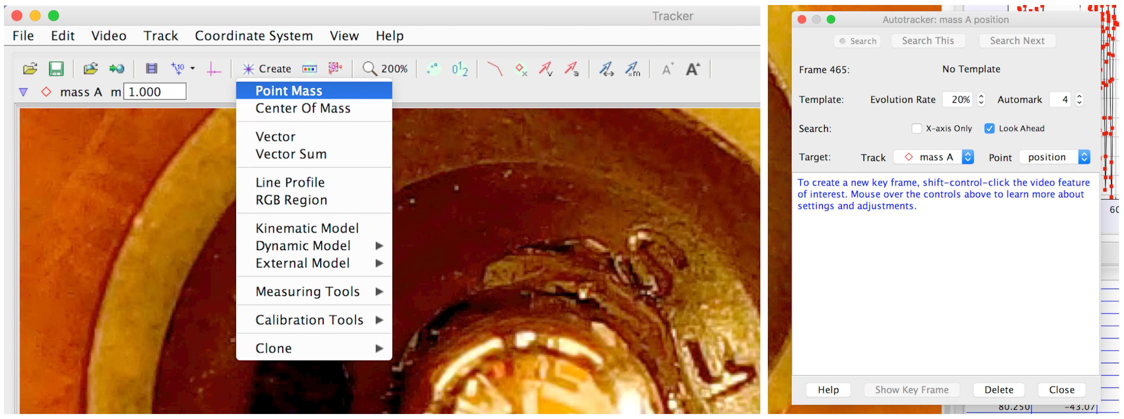
Main view of video and tracks will appear here.
Choose File|Open or Tracks|New to start.

Plot view of track data will appear here.

Table view of track data will appear here.

Figure 53. Start-up screen for the "Tracker" program.

The next step is to either click and drag the video of interest into the program window or open the file through the interface. Once uploaded, which may take a few minutes depending on the file size, a point mass must be created in order to start the auto-tracker. **Figure 54** illustrates clicking the “Create” button to select a point mass on the left. Once created, the auto tracker window is opened by selecting the button to the right of the magnifying glass and is shown on the right of **Figure 54**.



Autotracker: mass A position

Frame 465: No Template

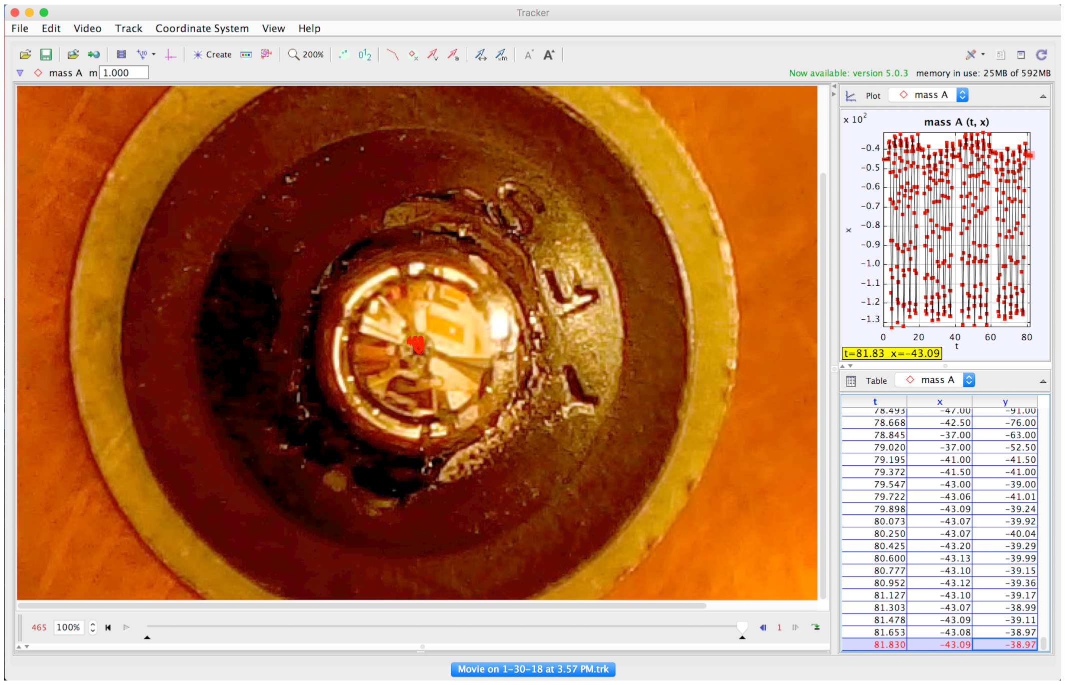
Template: Evolution Rate 20% Automark 4

Search: X-axis Only [✓ Look Ahead]

Target: Track mass A Point position

To create a new key frame, shift-control-click the video feature of interest. Mouse over the controls above to learn more about settings and adjustments.

Figure 54. Creating a point mass (left) and the auto tracker window (right).



t	x	y
78.493	-47.00	-91.00
78.668	-42.50	-76.00
78.845	-37.00	-63.00
79.020	-37.00	-52.50
79.195	-41.00	-41.50
79.372	-41.50	-41.00
79.547	-43.00	-39.00
79.722	-43.06	-41.01
79.898	-43.09	-39.24
80.073	-43.07	-39.92
80.250	-43.07	-40.04
80.425	-43.20	-39.29
80.600	-43.13	-39.99
80.777	-43.10	-39.15
80.952	-43.12	-39.36
81.127	-43.10	-39.17
81.303	-43.07	-38.99
81.478	-43.09	-39.11
81.653	-43.08	-38.97
81.830	-43.09	-38.97

Figure 55. Image of the "Tracker" program window once tracking of a video is complete.

When tracking the reflection of the camera in the ball, if the auto-tracking function does not work it is important to remember that the key combination shift-control-click only updates the template, shift-option-click selects the center and moves to the next frame. A way to check you have not made this mistake is to watch the plot of the x-position versus time on the right. The plot should look similar to the one displayed in **Figure 55**.

Upon completion the time, x-position, and y-position is easily extracted by selecting the entire table and copying it to an excel sheet. It is also important to save the tracker file as .trk in order to have access to the work completed during this process.

B.2. Running the Tracking Software in Development

In order to run the program both Matlab and the EE426DIPToolbox toolbox need to be present on the computer. The toolbox (obtained from Dr. Marina Mondin) does not appear in this thesis because it is too large to include its contents. Once those are successfully installed the file startup_rvc.m must be run in order to run the code. The code can be found under example_06_TemplateMatchingDaisy.m which contains the current code to track a template containing one daisy through a rotation. In line 9 the code requires a name for the daisy template being used. In this version of the code the file being used is DaisyTemplate1.jpg which appears on the left of **Figure 56**

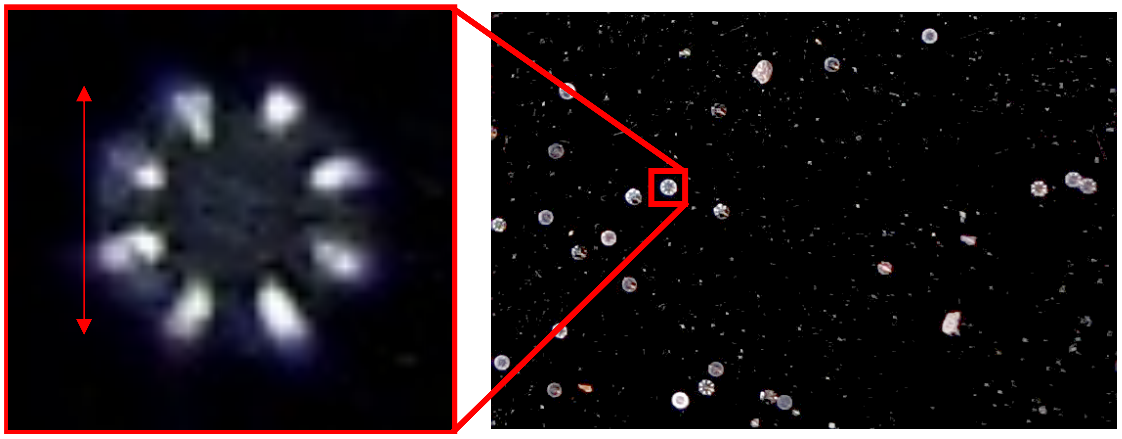


Figure 56. Template used to track a bead through its rotation.

Once the program has found a position for the daisy in the image it will return the center to the output window. The images produced by the code are shown in **Figure 57** starting with the template on the top right, the image in which the code is looking for a match to the template on the bottom right, the correlation map to the image on the bottom left, and the results of the code on the top left. The code only found one match to the template with a correlation threashold of 70% or higher, the output (below) has a correlation threshold of 95% or higher and also results in one center.

f = (1) area=1, cent=(731.0,370.0), theta=1.57, b/a=NaN, color=1, label=2, touch=0, parent=1

Currently the centers are extracted for each image and placed into an excel sheet. In the future, a batch method will be necessary, and the centers need to be printed to a text file for ease of use.

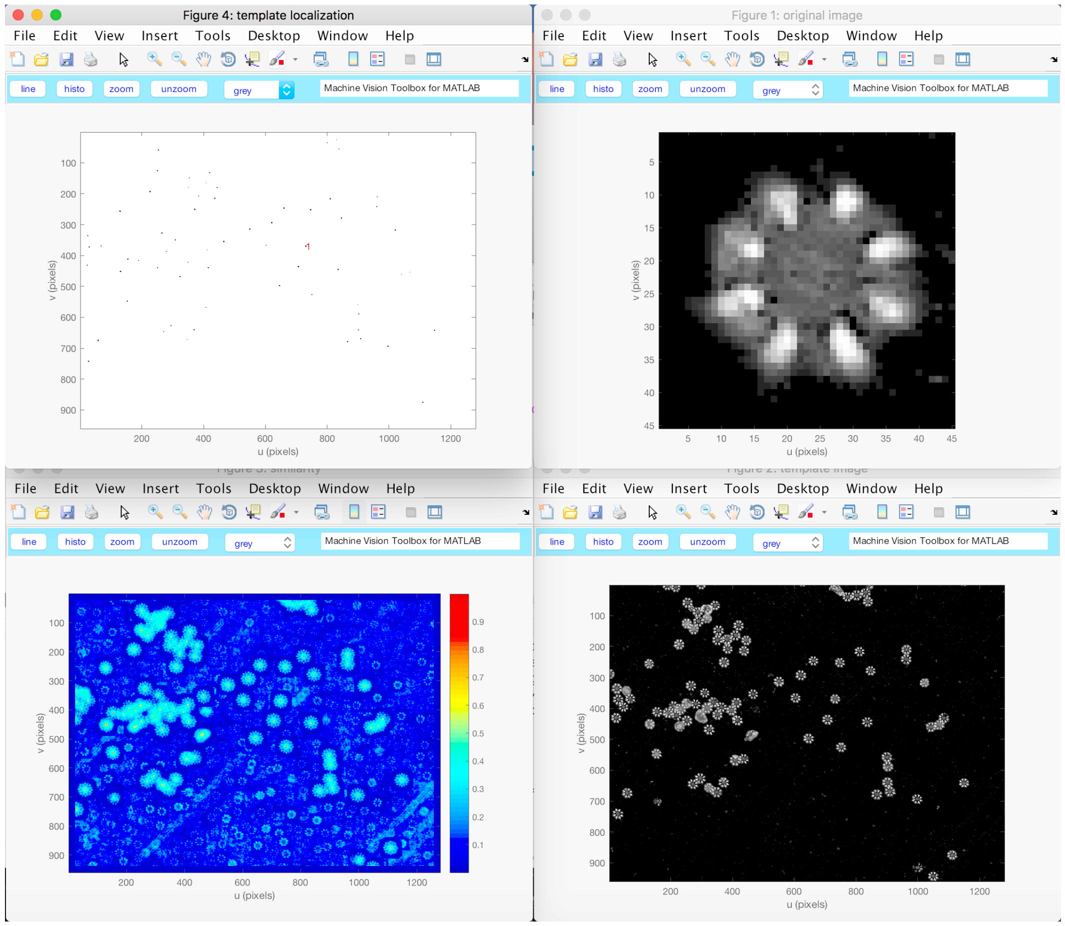


Figure 4: template localization
Figure 1: original image
Figure 3: similarity
Figure 2: template image
File Edit View Insert Tools Desktop Window Help
line histo zoom unzoom grey
Machine Vision Toolbox for MATLAB
u (pixels)
v (pixels)

Figure 57. Image product of the template matching code.

B.2.1. startup_rvc.m

```
disp('Robotics, Vision & Control: (c) Peter Corke 1992-2011 http://www.petercorke.com')
tb = false;
rvcpath = fileparts( mfilename('fullpath') );
robotpath = fullfile(rvcpath, 'robot');
if exist(robotpath,'dir')
    addpath(robotpath);
    tb = true;
    startup_rtb
end

visionpath = fullfile(rvcpath, 'vision');
if exist(visionpath,'dir')
    addpath(visionpath);
    tb = true;
    startup_mvtb
end

if tb
    addpath(fullfile(rvcpath, 'common'));
    addpath(fullfile(rvcpath, 'simulink'));
end

clear tb rvcpath robotpath visionpath
```

B.2.2. example_06_TemplateMatchingDaisy.m

```
%example0_Matching template
close all
clear all
%template = iread('template.jpg','double','grey');
%stars = iread('stars02.jpg','double','grey');

template1 = iread('DaisyTemplate1.jpg','double','grey');
template=template1(:,2:end-2);
stars = iread('Daisy01.jpg','double','grey');

S = isimilarity(template, stars, @ncc); %ncc is the normalized cross correlation

figure(1); idisp(template,'title','original image');
figure(2); idisp(stars,'title','template image');
figure(3); idisp(S,'title', 'similarity','colormap','jet','bar');
figure(4); idisp( S>=0.7,'invert','title','template localization');

out = S >= 0.95;
f = iblobs(out, 'area',[1,5])

[m,n] = size(f);
for k=1:n
    msg =sprintf('%d',k);
    text(f(k).uc, f(k).vc, msg,'Color','r');
end
```

B.3. Testing of the Current Tracking Software

Initial analysis of the tracking program has been completed. Images were taken of a sample prepared by Nicole and Riccardo over the summer which was glued to the end of a stepper motor shaft in order to simulate the actual motion of the rod. The sample was chosen to allow for enough points to track but not too many as to reduce noise in the correlation signal. The four samples are shown below. The image labelled 2 was the one chosen sample for this study.

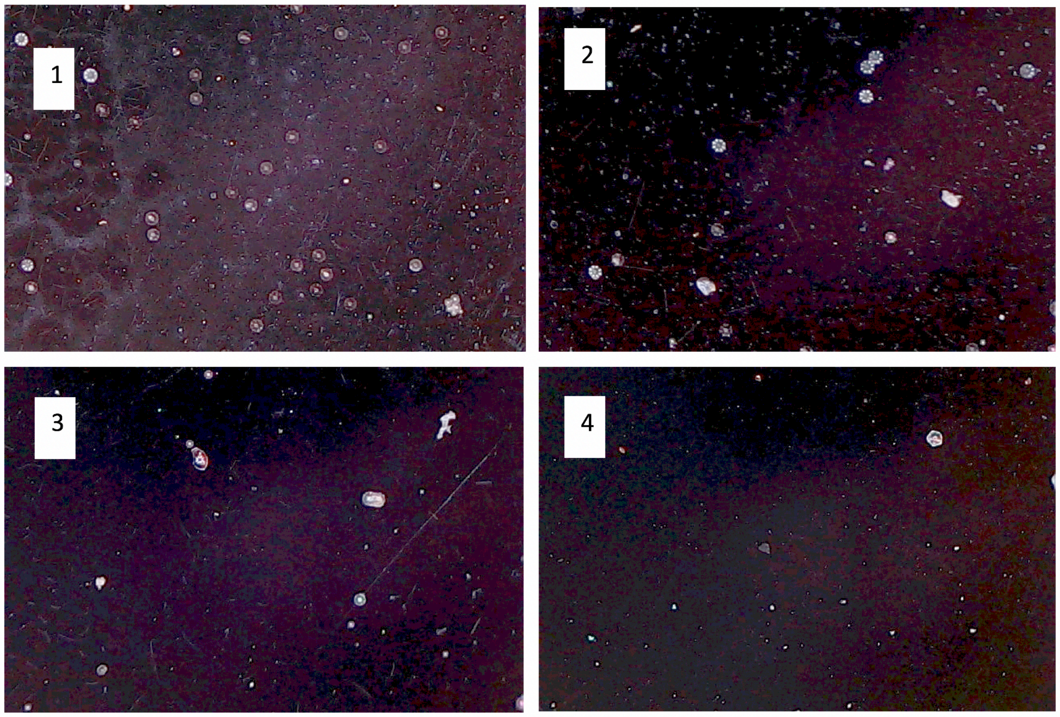


Figure 58. Four samples of wax-glued glass beads to a black glass plate.

In order to model the rotation of the rod, the stepper motor was rotated in various increments through one full rotation. At the start were very small steps of 0.0977% of the full rotation which increased up to a stable step in position of 1.95% the full rotation. Taking a total of 73 images of the sample through one rotation. Each image was processed using the correlation program. A bead that maintained a position within the frame for the full rotation was chosen and cropped from the initial photograph to use as the correlation sample and this crop is shown in **Figure 56**. This was used to generate a correlation map across each image and then another plot with a marker.

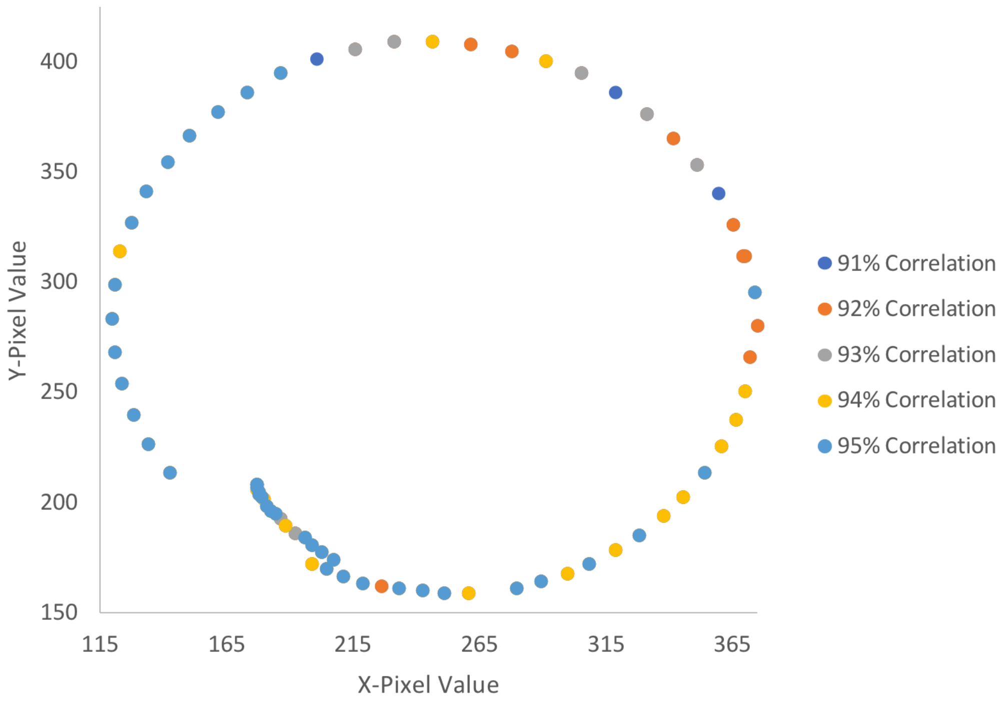


Figure 59. Results from testing the current tracking program.

Analysis was done using different minimum correlation values. The initial minimum was set to 95% and then slowly decreased. As the threshold was increased more noise was seen in the images as shown in

Table *5*. If an automation of the code is pursued, a loop that starts at a high threshold and then moves down slowly only looking at images where the initial run was a failure would be necessary. There should also be a minimum deviation implemented to rule out mislabeled objects within the image because as the threshold is increased, more objects begin to appear in the output.

Table 5

Results from increasing the correlation threshold.

Minimum Correlation Value	Number Found	Percent of Total	Number of Misreadings
95%	39	53	1
94%	54	74	1
93%	61	84	8
92%	70	96	14
91%	73	100	14

To remove the possibility of misreadings, the idea of tracking a constellation of glass beads has been discussed and attempted. Unfortunately, adding the functionality of rotating the template to find the best correlation value to determine the new position and its rotation from the initial point causes the program to run for hours as opposed to minutes. The idea of transferring this code to a faster programming language might need to become a reality for the future of this program. It is also clear that batch tracking will need to be implemented in the future to run through a file of images rather than having the user input each image individually since current experiments acquire hundreds of images per frequency run.

APPENDIX C

Raw Data and Ellipse Fitting Code

This appendix contains raw data in the form of plots for each frequency run, how to run the polynomial curve fitting code, fitting results, and the code used.

C.1. Raw Data

This section contains all raw data used to obtain the final results appearing in the Results section of this thesis. It is clear that contact is being made with the eddy current brake and is causing the bumps in the rotation data.

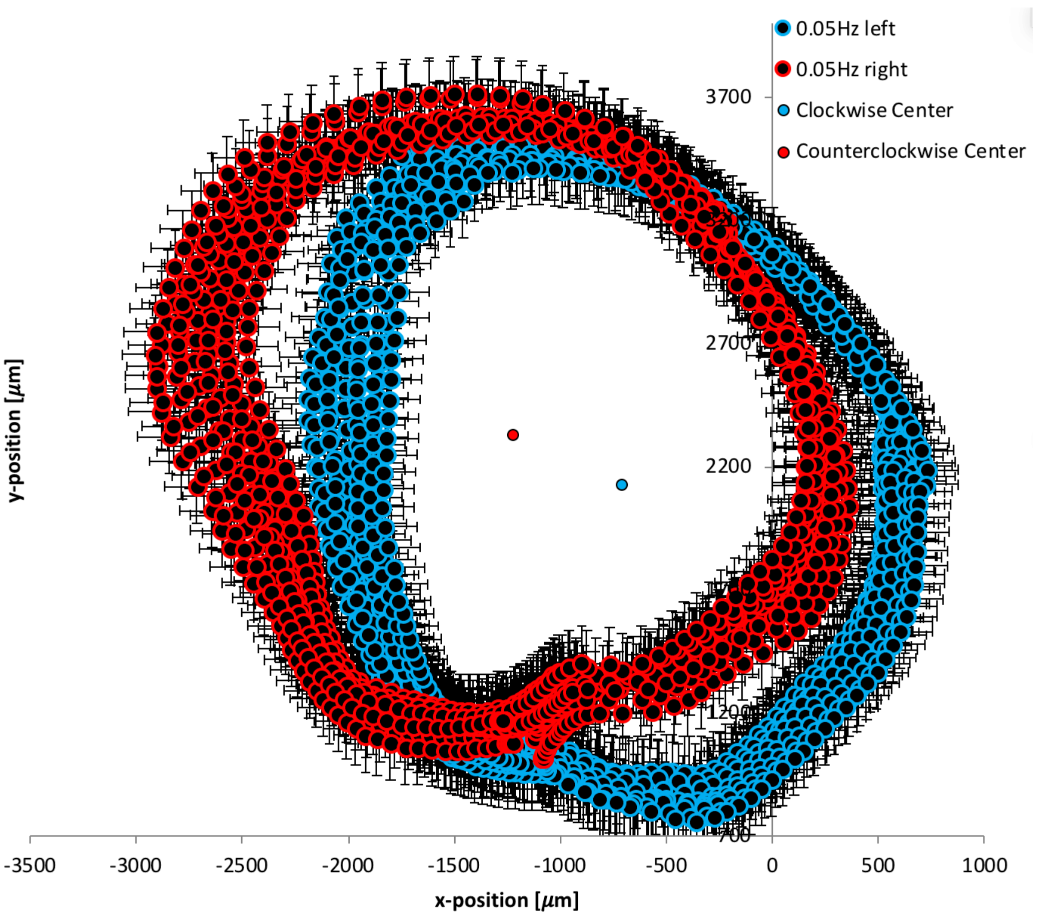


Figure 60. Rotation Data for 0.05 Hz.

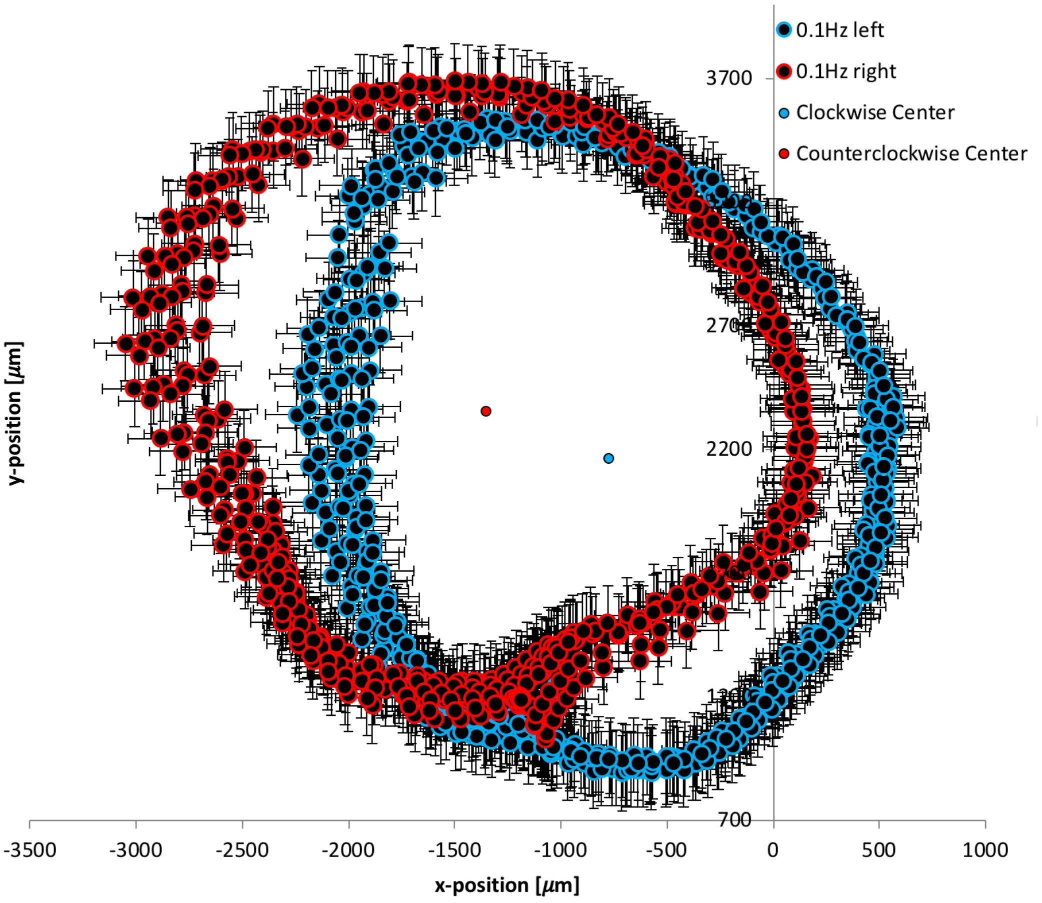


Figure 61. Rotation Data for 0.1 Hz.

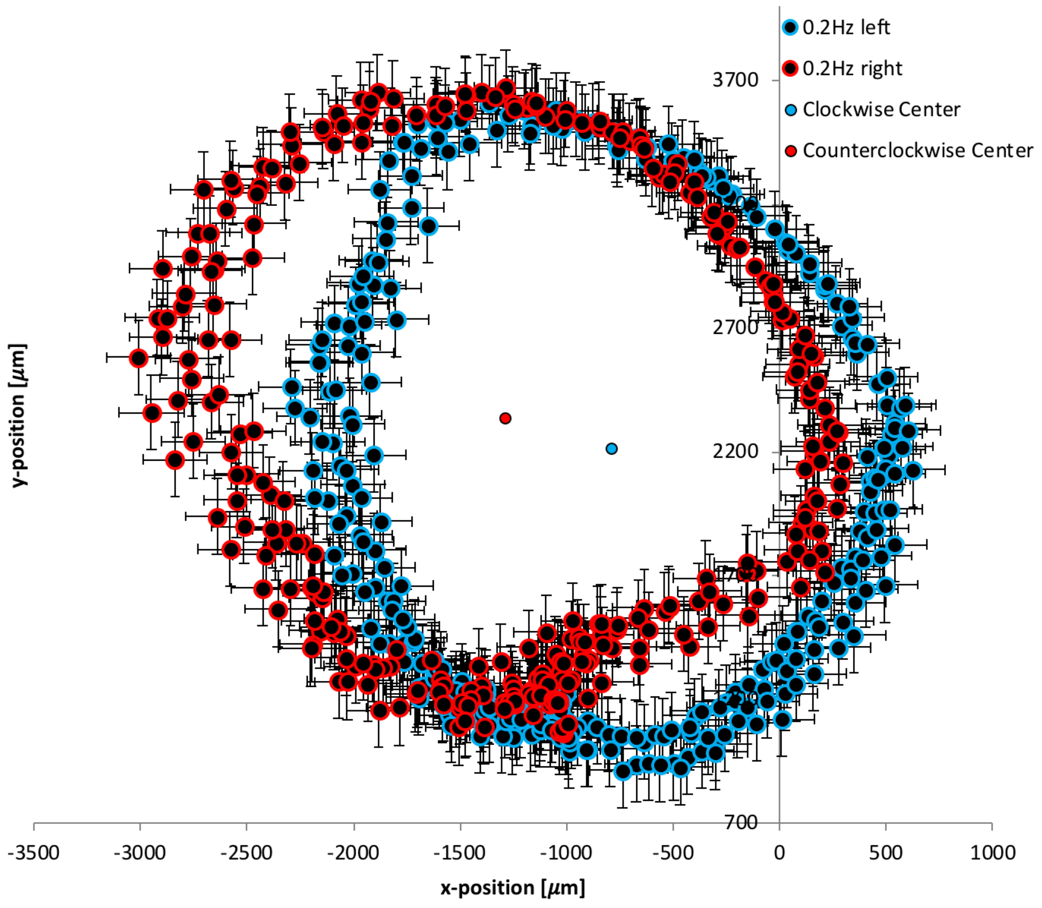


Figure 62. Rotation Data for 0.2 Hz.

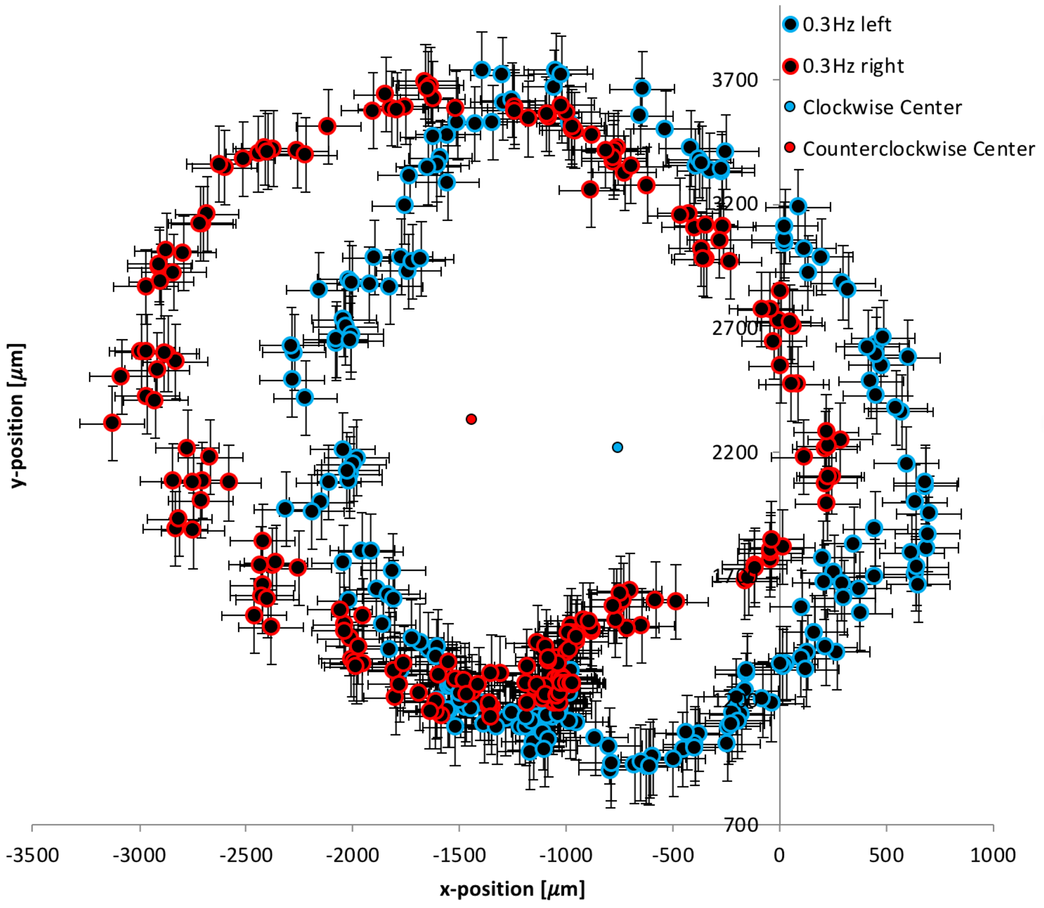


Figure 63. Rotation Data for 0.3 Hz.

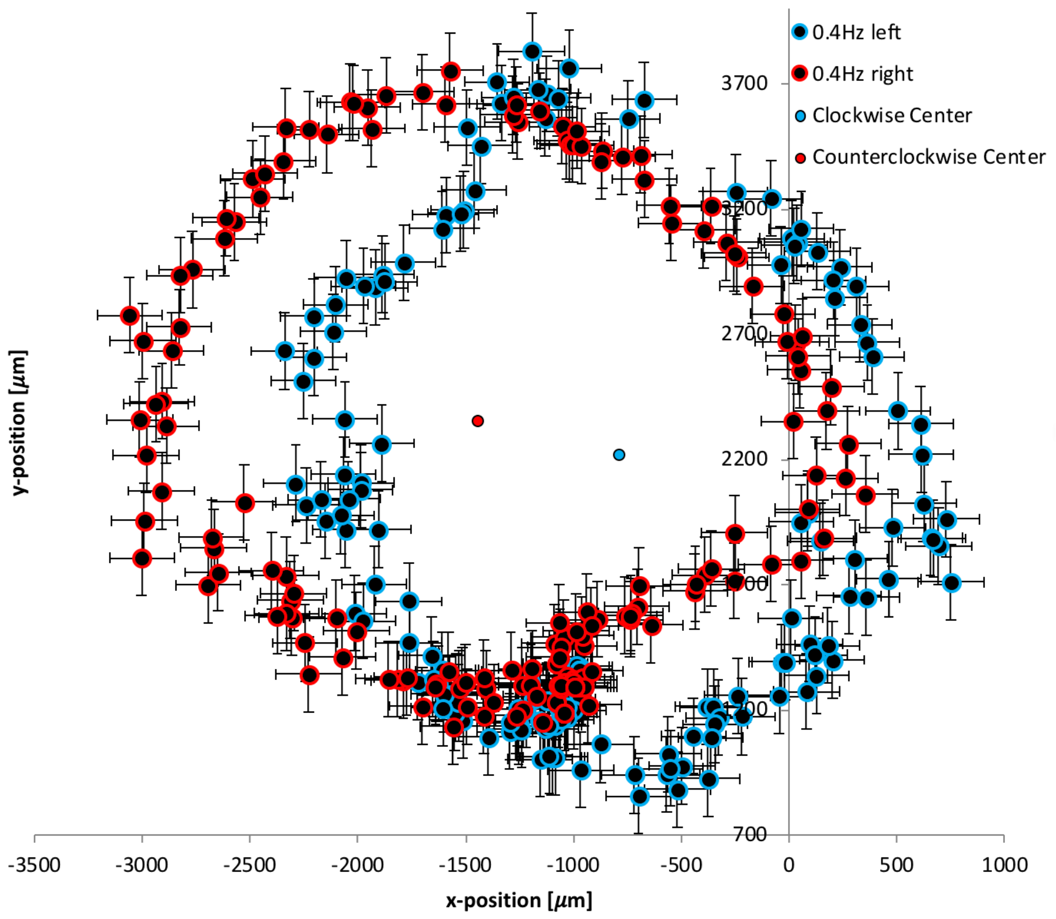


Figure 64. Rotation Data for 0.4 Hz.

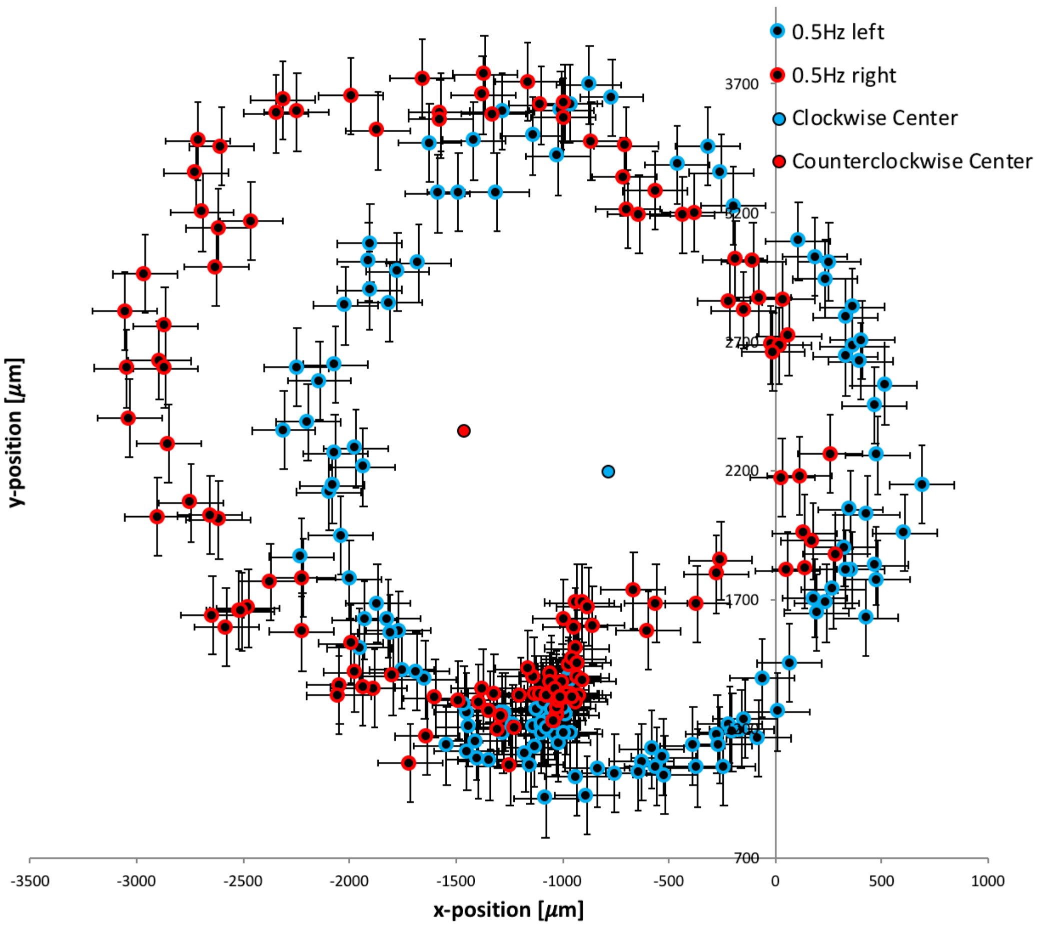


Figure 65. Rotation Data for 0.5 Hz.

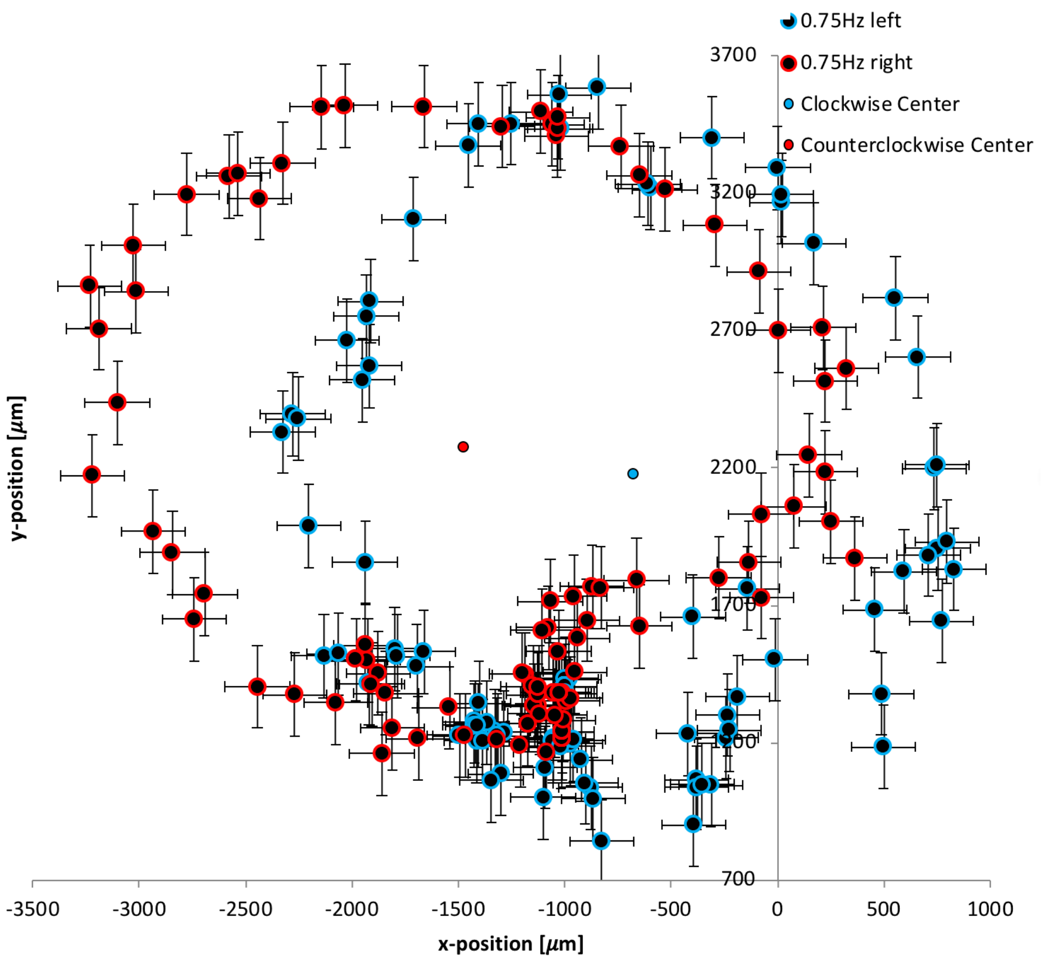


Figure 66. Rotation Data for 0.75 Hz.

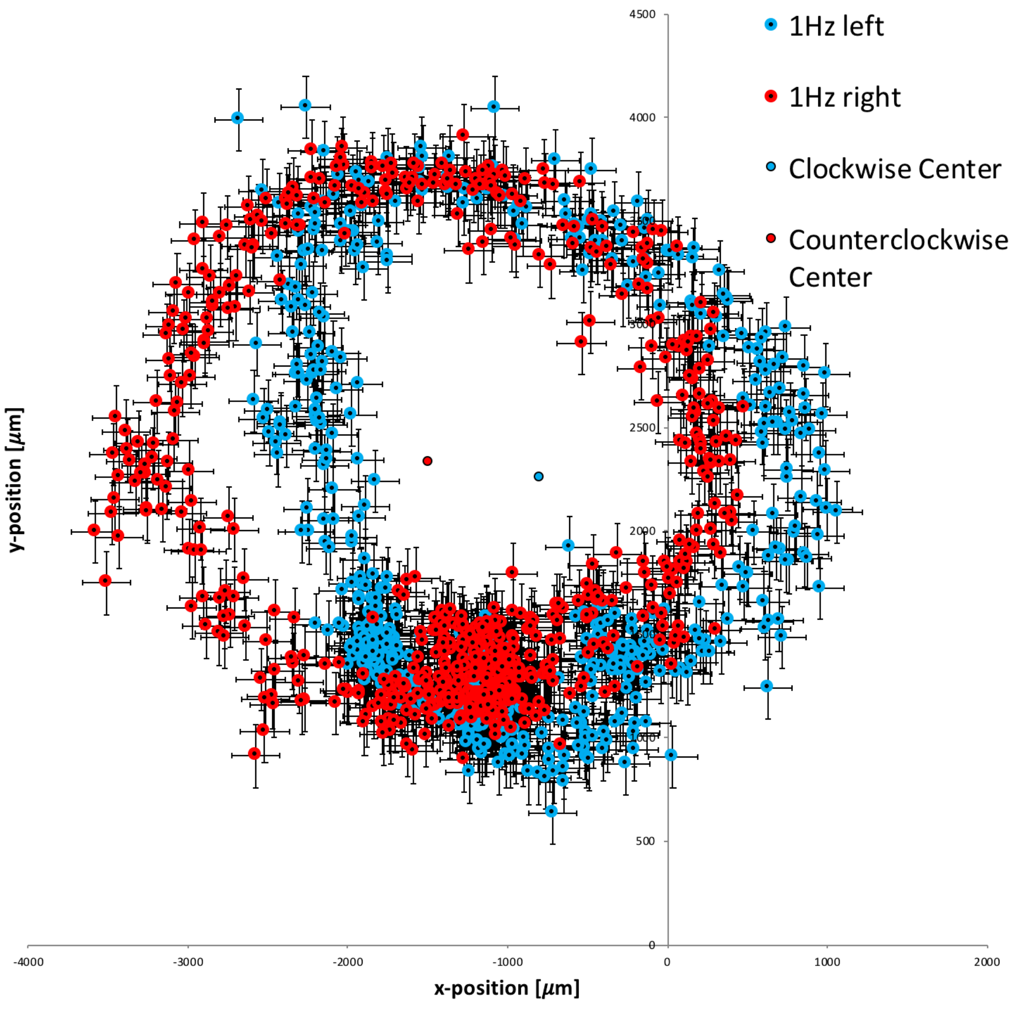


Figure 67. Rotation Data for 1 Hz.

C.2. Running the Ellipse Fitting Code

The ellipse fitting code was written by Samavarti Gallardo in python and can be run through any python platform. The instructions on how to run this code can be found in his documentation on the Laser Interferometer Gravitational Wave Observatory (LIGO) document control center (DCC) at: https://dcc.ligo.org/cgi-bin/private/DocDB/ShowDocument?docid=T1800205&version=

His technical paper contains a full explanation on the logic behind the code and how it generates the results found in the following section.

C.3. Result from Ellipse Fitting Code

The following images are the output from the polynomial curve fit. As part of the bootstrapping method, each parameter of the fit was recalculated for each selection of the data set. These values are plotted in a histogram to find the best fit value for each parameter and the error in this value in order to produce the final fit.

The output that is most important is outlines as follows. Starting with the first row on the top left is a plot containing the clockwise rotation data in red with the final fit in blue, two columns to the right is a plot containing the counterclockwise rotation data in red and the final fit in blue. Moving to the second row, plots of x and y center values appear with the peak value and its error (sigma value) appearing at the top of the plot. These values were used to calculate the loss angle of the material at the given rotation speed.

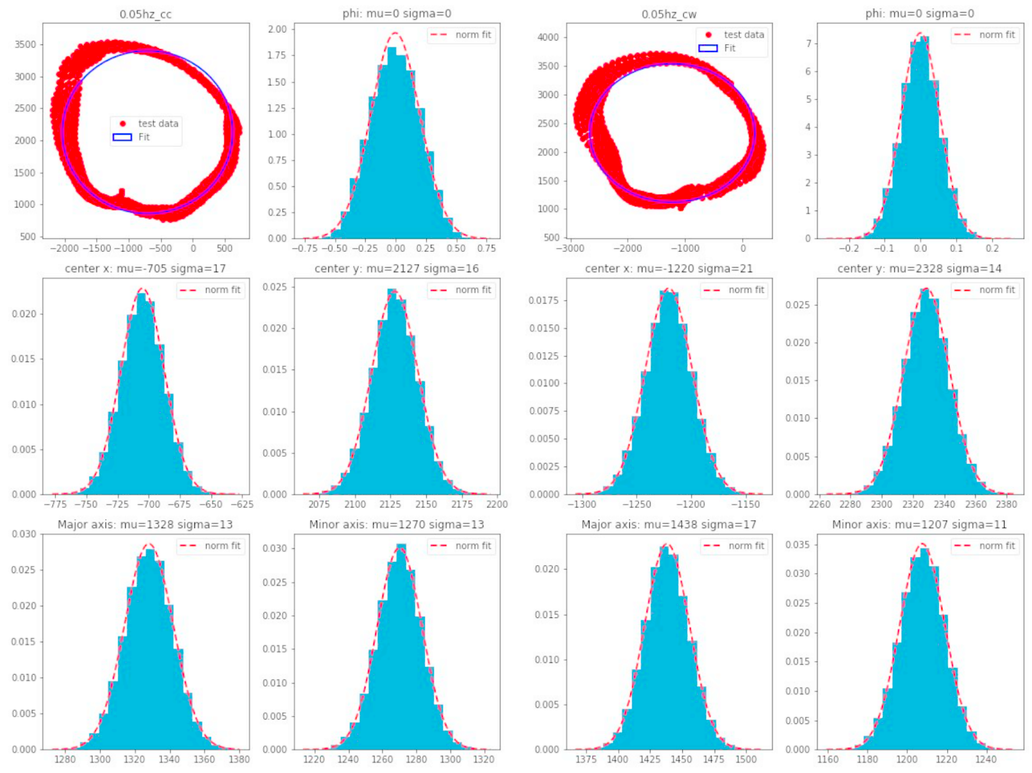


Figure 68. Fitting results for 0.05 Hz.

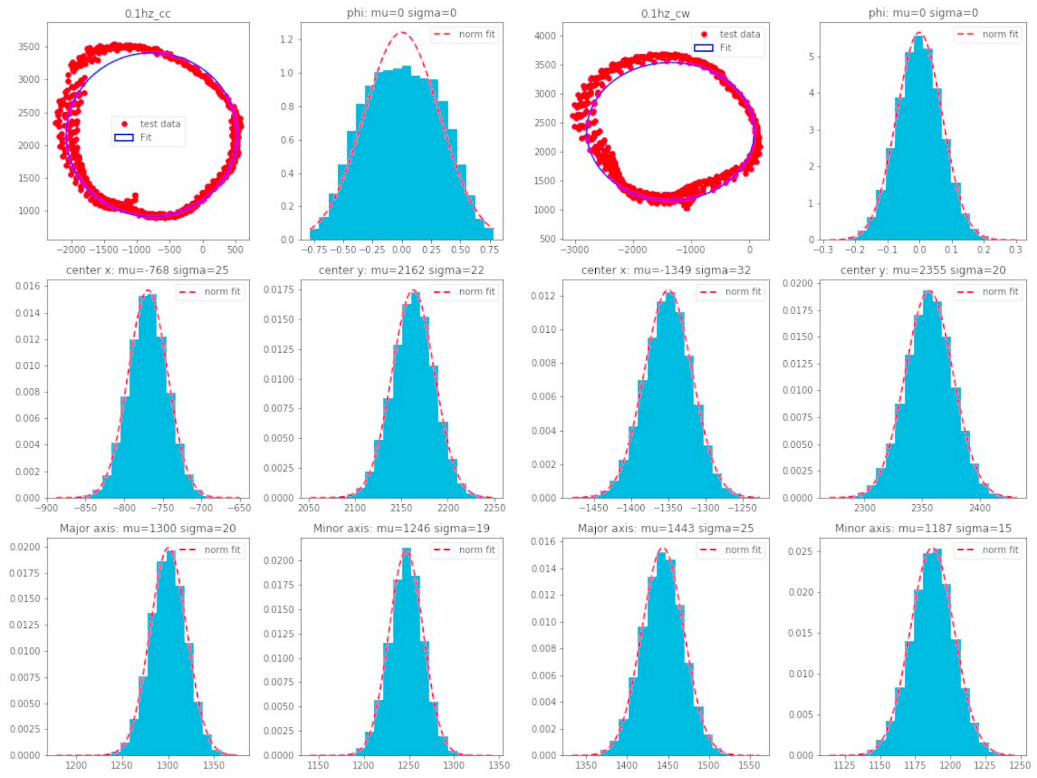


Figure 69. Fitting results for 0.1 Hz.

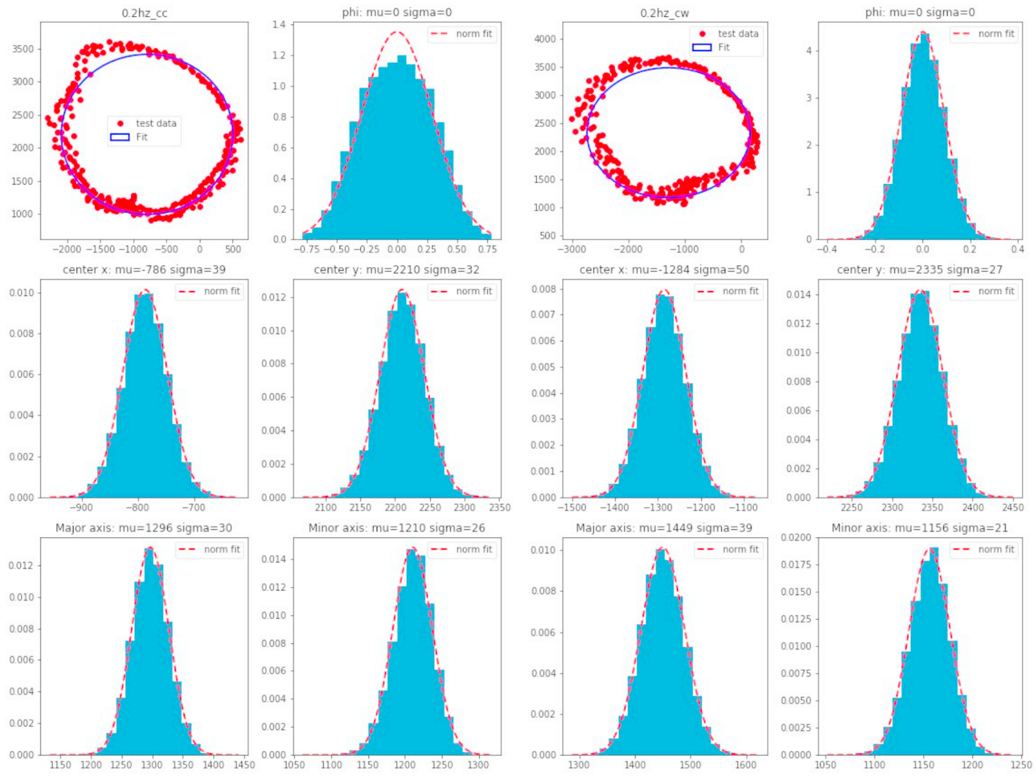


Figure 70. Fitting results for 0.2 Hz.

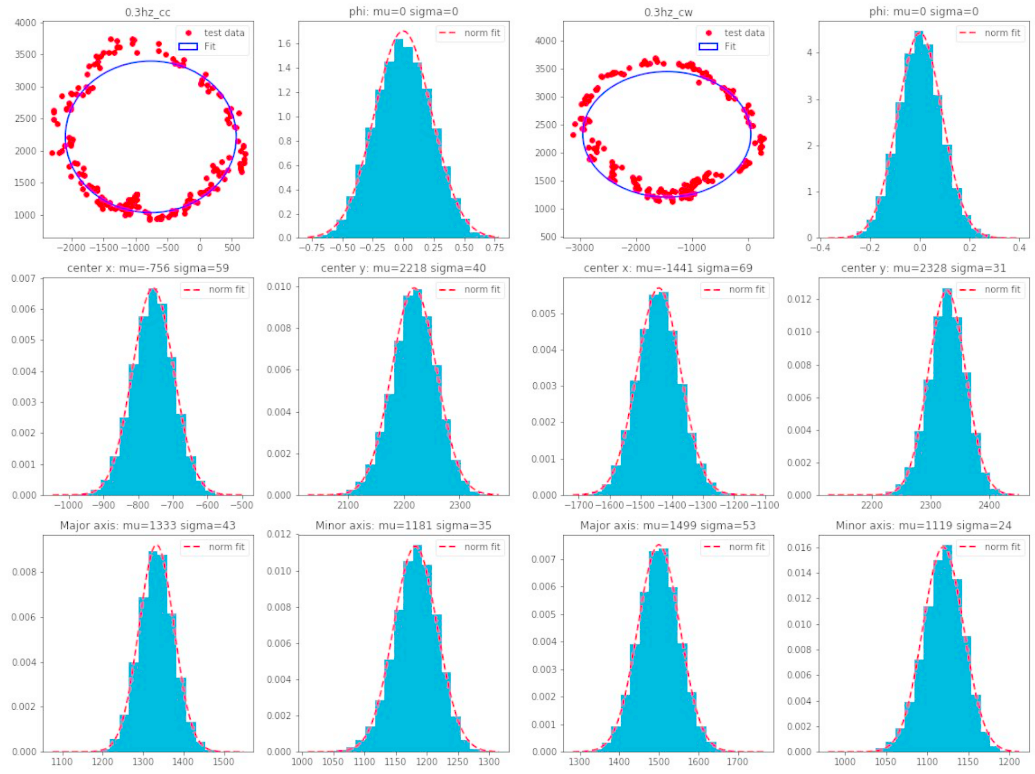


Figure 71. Fitting results for 0.3 Hz.

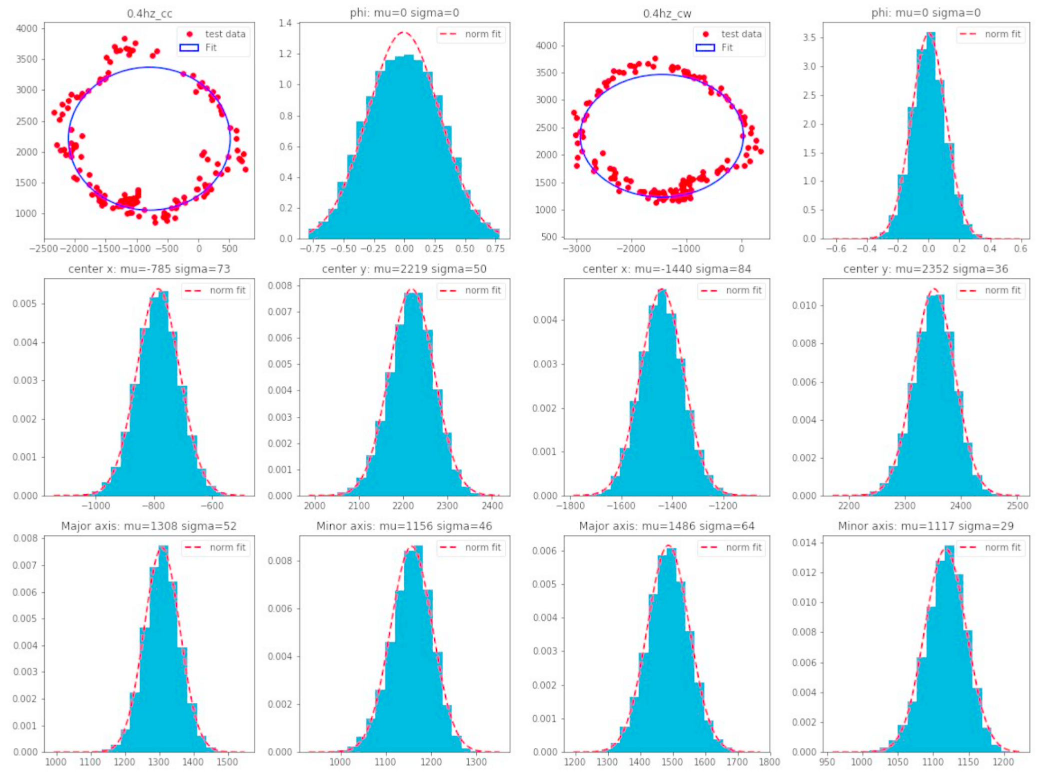


Figure 72. Fitting results for 0.4 Hz.

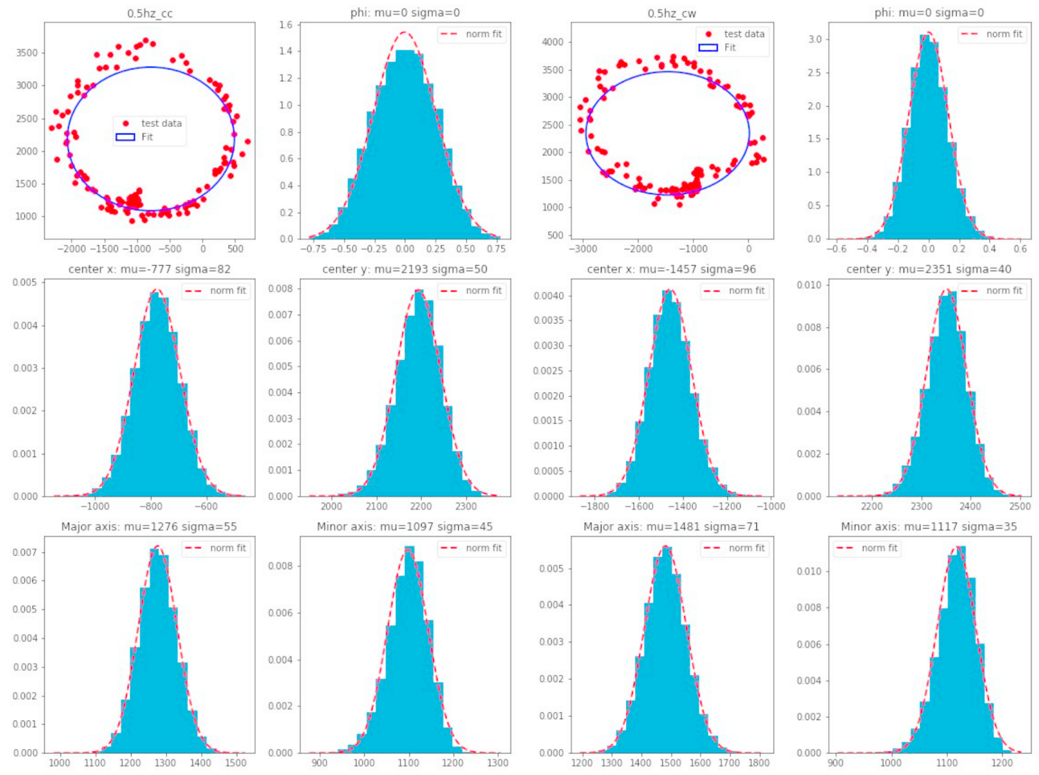


Figure 73. Fitting results for 0.5 Hz.

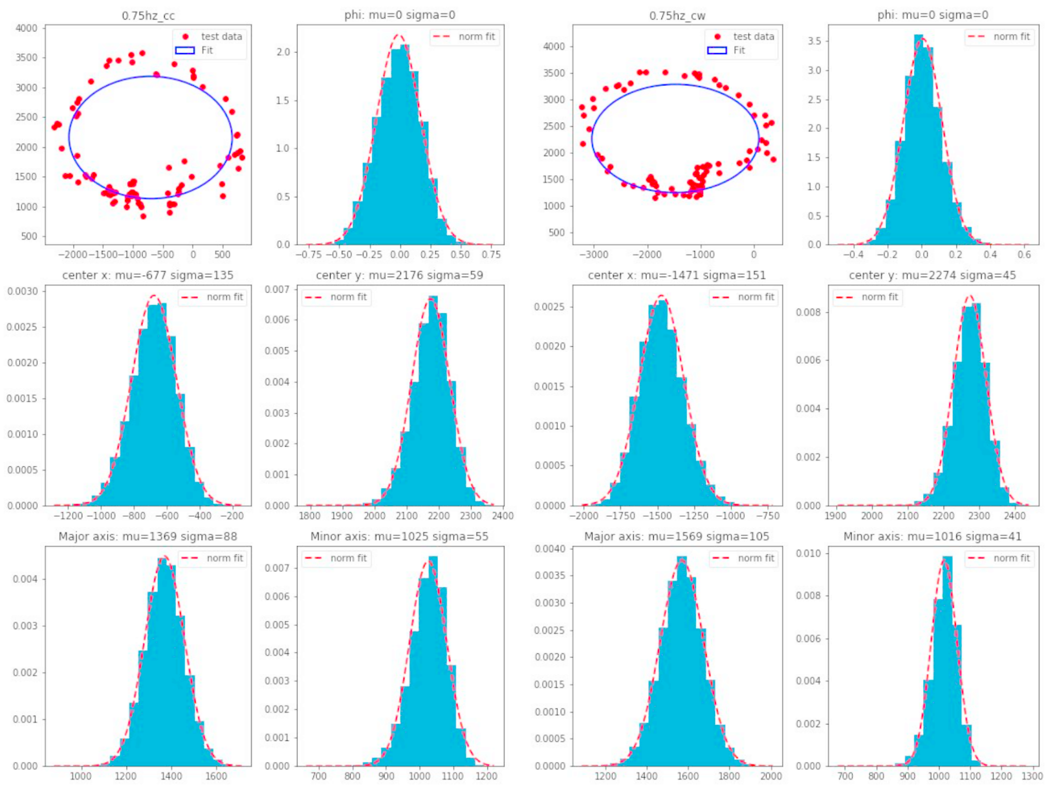


Figure 74. Fitting results for 0.75 Hz.

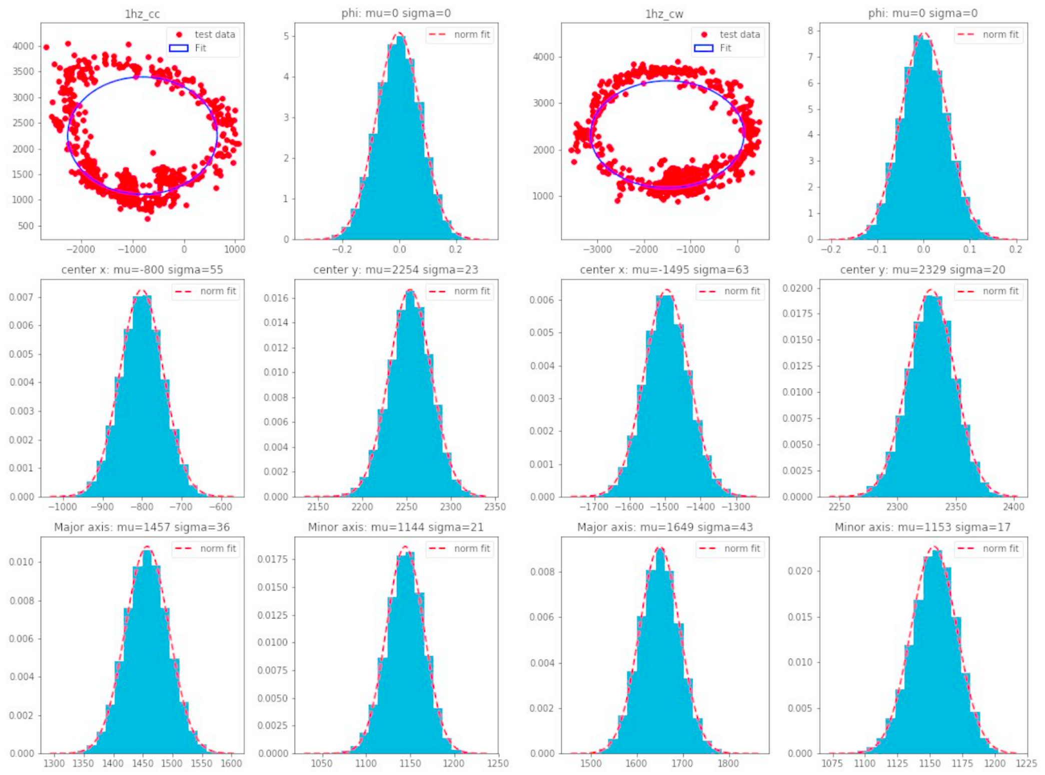


Figure 75. Fitting results for 1 Hz.

C.4. Ellipse Fitting Code

The code appears in Sam's technical paper on the LIGO DCC which can be found through the following link:https://dcc.ligo.org/cgi-bin/private/DocDB/ShowDocument?docid=T1800205&version=

APPENDIX D

Reynolds Fixture User Manual and Explanation of the Code

Written by Morgan Shaner to explain code compiled by Dr. William J Baker and David Schwarz on 06/22/2017.

D.1. Introduction

When writing scripts in python there are built in modules created for different functionalities. Outside these modules functions need to be created to customize functionality. The following is a document created with the aim of allowing any user to run the Reynolds Fixture and understand the output. The first section lists and describes all referenced code within the main script (referenced modules can be found in the appendix with links to detailed website explanations). This is followed by an outlined description of the main code, an explanation of how to run the Reynold's fixture and what to do in case of an error and/or failure. Finally, should the code be necessary, it appears in the appendix of this document.

D.2. Referenced Code

D.2.1. pump_operator

This is a class written to run the water pump used to add specified mass to the fixture's bucket. The class is called ***Pump_Operator()*** and has six defined functions within. These functions are listed below along with their functionality.

def DisplayDeviceInfo(self):	Displays the device information.
def _init_(self):	Initializes the pump.
def dispense(cls , mass_to_add):	Dispenses a set amount of water to the bucket.

def dispense_calc(cls , frequency_current):	Determines how much was dispensed.
def time_calc(start_mass , mass_to_add):	Calculates the time needed to dispense a particular amount of water to the bucket.
def close(cls):	Stops the pump.

D.2.2. fft_operator_u3

This is a class written to acquire data from the capacitive sensor connected to the LabJack, fit an FFT to the position and time data acquired, and find the resonant frequency of the spring system. The class is called ***Fft_Operator()*** and has seven defined functions within. These functions are listed below along with their functionality.

def _init_(self , seconds_to_measure):	Initializes the LabJack through u3 (a script developed to use the LabJack device) to acquire data.
def emptybucket(cls):	Empties water held in the bucket by opening and closing the solenoid attached to the bottom.
def impulse(cls , delay):	Applies an impulse to the bucket using the solenoid attached to the top of the bucket.
def close(cls):	Stops acquiring data from the LabJack using u3.

D.2.3. parabolic

This script was written to fit the FFT using a parabolic function to determine the resonant frequency as opposed to taking the max of the FFT. This code includes two functions ***parabolic(f,x)*** and ***parabolic_polyfit(f,x,n)***. The first is a quadratic interpolation for estimating the true position of an inter-sample maximum when nearby samples are known; and the latter find the peak of a parabola using the fit.

D.2.4. sartorious

This is the python interface provided for the scale and contains the class ***Sartorius(serial.Serial)*** which contains the six functions listed below with their functionality.

def _init_(self , com_port):	Initializes the scale.
def value(self):	Will return the displayed value on the scale.
def display_unit(self):	Will return the unit of measurement.
def tara_zero(self):	Tares and zeros the scale.
def tara(self):	Tares the scale.
def zero(self):	Zeros the scale.

D.2.5. checking_ports

This script contains a function named ***serial_ports()*** which lists the serial port names, raises and environment error on unsupported or unknown platforms, and returns a list of the serial ports available on the system.

D.3. Main Script

The main script utilizes the modules listed in the appendix and the scripts referenced above to control the Reynolds Fixture and tune blades for the HILA project. This section will explain the variables, functions, and finally the main code within the script, highlighting the functionality of the main code and pointing out areas that are commonly changed to serve different testing purposes. The following line numbers are based off the original version of the code; spaces were removed in the code appearing in the appendix of this document to shorten the length of the document and therefore, will not correspond to the lines within the document code.

Lines 8-24:	Import all the necessary modules, classes, and functions listed in the previous section.
Line 27:	Define a variable to be utilized later in the code to break if the initial TWP calibration has not been done.
Line 29:	Define a variable to be utilized to keep track if the test is a pass or a fail. (i.e Does the test pass the 1mm working point somewhere in the middle? Are we above 2Hz?)
Lines 31-34:	Making sure the calibration file exists, and if not create the file.
Lines 36-42:	Checks to see if the calibration is loaded and what the value is.
Lines 45-57:	Definitions of all the constants and user inputs within the script.
Lines 60-64:	Initializes all the necessary devices including the scale, fft, labjack, u3 and the pump.
Lines 66-374:	Defines all functions within the script which are listed below with their functionality.

***def* runMeasurement(user_mass)**: Runs a measurement.

- Creates a directory to which the information to be stored.
- Initializes variables to catch a failed fit or if a fall through occurs.
- Uses the function ***writeData()*** to collect and store the data.
- Reads in the starting mass of the scale and uses the function ***initial_mass_approach*** to bring the target into range to begin the measurement (currently set to 10mm).
- Loop in place to dispense the correct amount of water by determining the mass added to the bucket by finding the difference in mass read by the scale. If this loop reaches its 20th iteration without reaching half the amount of mass to add it will break and empty the bucket. This is to protect the pump from running without water.
- Test to see if the drift at low frequency can be eliminated.
- Impulse applied to the fixture.

- Prints the resonant frequency.
- Writes raw data to a .txt file (this is only necessary when acquiring data for a Q-Factor analysis, comment out if not necessary).
- Checks for failures.
- Uses the resonant frequency to determine the next mass to add to the fixture.
- Finish statement.
- Calculates and prints the time it took to run the measurement.
- Empties the bucket when finished.
- Returns the data.

def imputCompressionPoint(): Allows the user to input a value for the compression point read from the dial indicator fixed to the left of the fixture, see **Figure 77**.

def twp_grabber(): Asks the user if the theoretical working point (TWP) has been calibrated, if yes (y) it will take the user back to the code, if not (n) it will measure the target distance with the TWP fixture installed to calibrate the sensor.

def userInput(): Requires the user to input: the added mass to the fixture, the part number of the blades, the serial number of the blades in lower case, and then press enter to continue with the code. (This can be changed to a manual input if the user should want to run multiple tests on the same set of blades so you don't have to input the information each time).

def timeStamped(fname, fmt='{fname}_%m-%d-%Y-%H-%M-%S'): Gives the time stamp.

def SMS(res_freq , distance , mass , PFmsg): Sends an email and/or text to anyone listed in this part of the code.

def writeData(data): Initializes a dictionary for data collection, plots the data while the program is running, and saves the final plot of the data.

def close(): Closed the pump and FFT operator scripts.

def signt_handler(signal , frame): Writes the data if an interruption has occurred.

Lines 337-438: Contains the code necessary to run a full test. It starts with the implementation of ***signt_handler()*** to allow the program to abort if necessary. Then it asks for the user to input the blade parameters before running the program. After a test, it will ask the user if they would like to run again at a new compression point. If not, it will close the program and free resources. If yes, it will ask for the new compression point and ask if the user would like to overlay the last run.

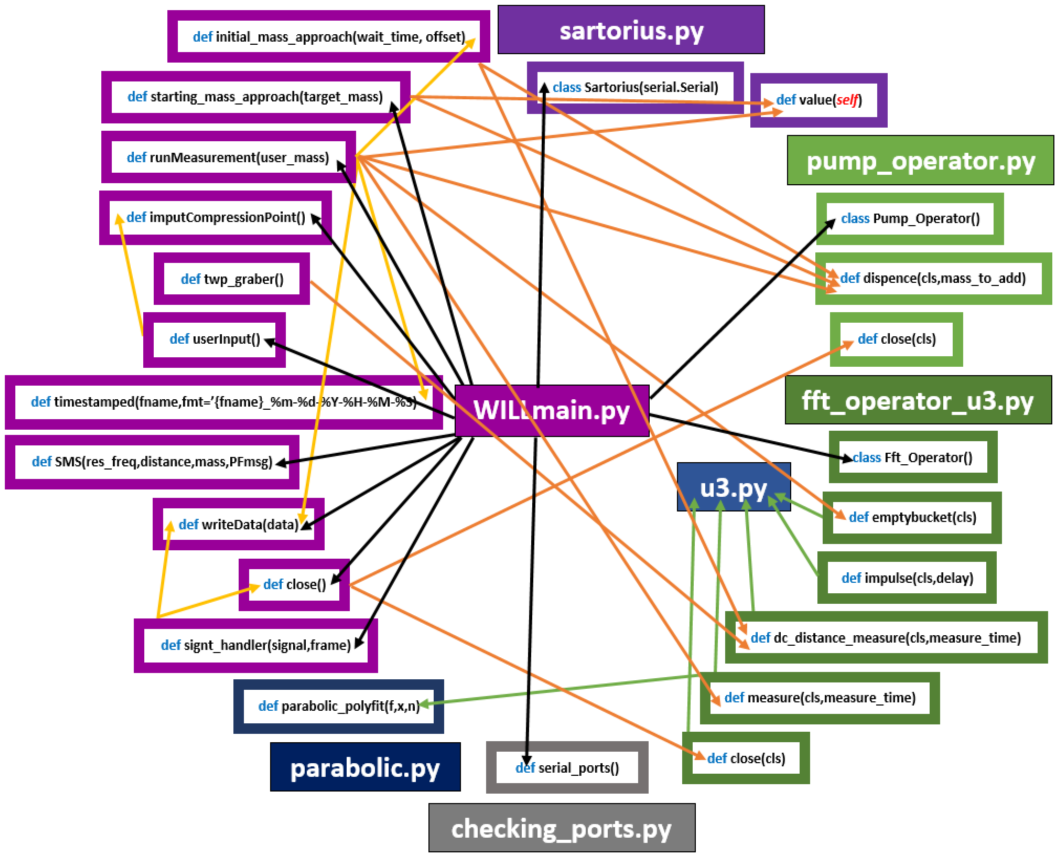

def initial_mass_approach(wait_time, offset)
sartorius.py
def starting_mass_approach(target_mass)
class Sartorius(serial.Serial)
def value(self)
def runMeasurement(user_mass)
pump_operator.py
def imputCompressionPoint()
class Pump_Operator()
def twp_graber()
def dispence(cls,mass_to_add)
def userInput()
def close(cls)
def timestamped(fname,fmt='{fname}_%m-%d-%Y-%H-%M-%S')
WILLmain.py
fft_operator_u3.py
def SMS(res_freq,distance,mass,PFmsg)
class Fft_Operator()
u3.py
def writeData(data)
def emptybucket(cls)
def close()
def impulse(cls,delay)
def signt_handler(signal,frame)
def dc_distance_measure(cls,measure_time)
def parabolic_polyfit(f,x,n)
def measure(cls,measure_time)
parabolic.py
def serial_ports()
def close(cls)
checking_ports.py

Figure 76. Web showing how the main code within WILLmain.py communicates with the other scripts.

In **Figure 76** black arrows indicate communication with the main script, yellow arrows indicate communication within the main script functions, orange arrows indicate communication with the main script functions and the other scripts, and finally the green arrows indicate communication between the fft_operator_u3.py script and the other scripts. An arrow head points to the function or class being utilized by the object from which it originates. There is no other communication out with this drawing with the understanding that u3, Phidget and python modules are left out to remove complexity. All unused functions within a script are also left out from the drawing above.

D.4. Step-by-Step How to Run

To run a spring tuning test the user must open both the WILLmain.py and fft_operator_u3.py scripts. It is good practice to have the stop_stream.py and empty.py scripts open as well should there be an error.

To start, run the fft_operator_u3.py script before starting the WILLmain.py script. Once started the script the terminal will ask the user for the inputs listed in the ***userInput()*** function. It will first call the function ***twp_grabber()***. If the theoretical working point (TWP) has been calibrated the user will answer "y" for yes to the question: "Has the TWP been calibrated for this spring set? In no, install TWP fixture and enter 'n' to calibrate. (y/n)". If not, the user needs to install the TWP fixture on the main fixture before entering 'n' for no. Once installed and no is entered, the capacitive sensor will read the current target distance and save it as the theoretical working point.

Once this has been completed, the terminal will prompt the user for the added amount of added mass to the system by asking the question: "How much additional mass was added(g)". The user must input the mass added in units of grams and press enter. The next question will be: "Part Number of Blades:" which requires the user to input the blade part number. This will either be 4109-905, 4109-906 or 4109-907 for maraging steel, or LM105 for glassy metal blades. The following question enquires for the serial number of the blade set by asking the question: "Blade Serial Number? (alpha/lower case):". The serial numbers appear on the blades themselves, but are covered once installed, the blade pairs were selected to ensure similar thicknesses and then inputted as pairs into an excel sheet. This is where the user will find the serial number for the selected position on the ring.

Next the user needs to remove the TWP fixture from the main fixture so the bucket is free to oscillate along the z-axis. The user input function will then call the ***imputCompressionPoint()*** function and the terminal will ask for the compression point which is read from the dial indicator fixed next to the main fixture, see **Figure 77**. The prompt is as follows: "With the TWP fixture removed, record the absolute compression point of blade Base(X.XXXin):". Once entered the terminal will finally ask the user to press enter to begin the measurement process.

The script will start by creating a unique file for the test run where it will record the suspended mass in grams in the first column, the resonant frequency in Hertz to the second column, and the working distance in millimeters to the third column. It will then initiate the measurement using the ***runMeasurement()*** function. The measurement will continue to run if all conditions are met. The script will search for a minimum around the 1mm and then continue adding mass until the resonant frequency of the system reaches halfway between the starting frequency and the minimum.

Once the measurement has finished the script will print "test was a success!" if the conditions were met (or "test was a fail!" if not) and display the minimum resonant frequency achieved along with the corresponding working point and added mass. It will also send the user a text with this information to allow the user to leave the room and return when the run has finished.

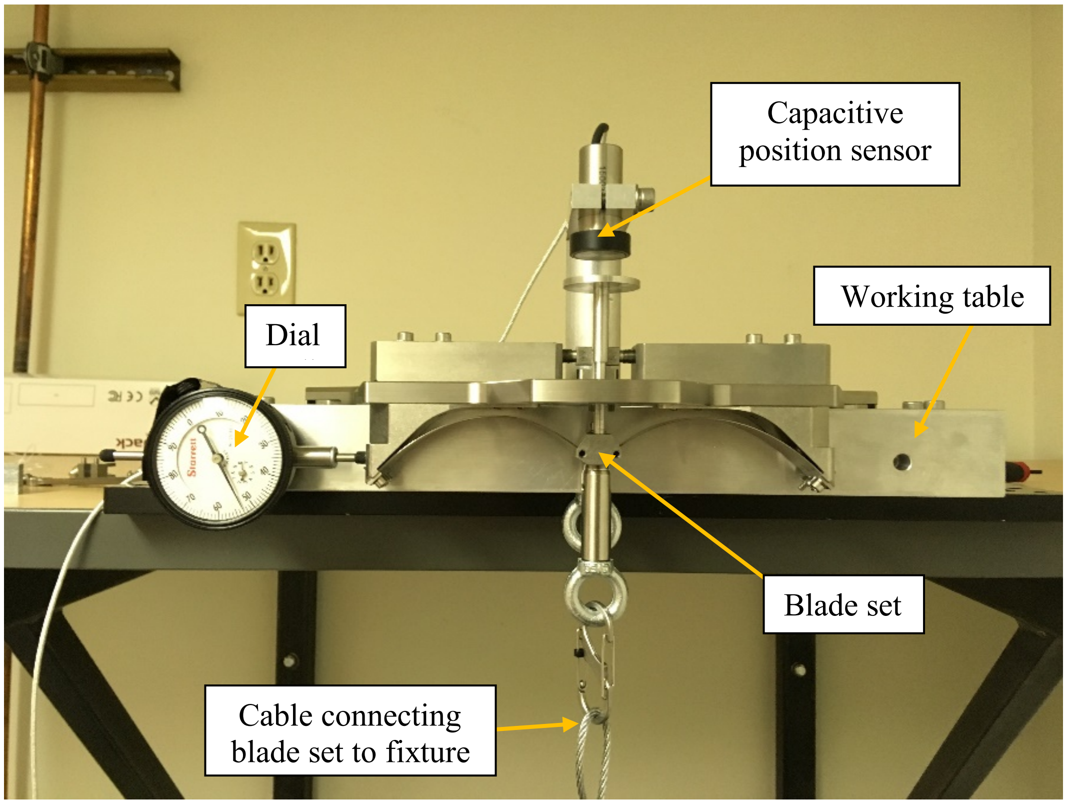

Capacitive position sensor
Working table
Dial
Blade set
Cable connecting blade set to fixture

Figure 77. On the left is the dial indicator, on the right is the fixture mounted to the working table.

Once the test is finished, the user has the option to run the test again (if for instance the blades are not compressed enough, corresponding to the system not reaching a low enough frequency (below 0.5Hz)). If the user enters 'y' for yes the user must enter the new compression point before running the code again as well as select if the last run should be overplayed on the graph. If the user enters 'n' for no, the program will close and save the data to the directory created.

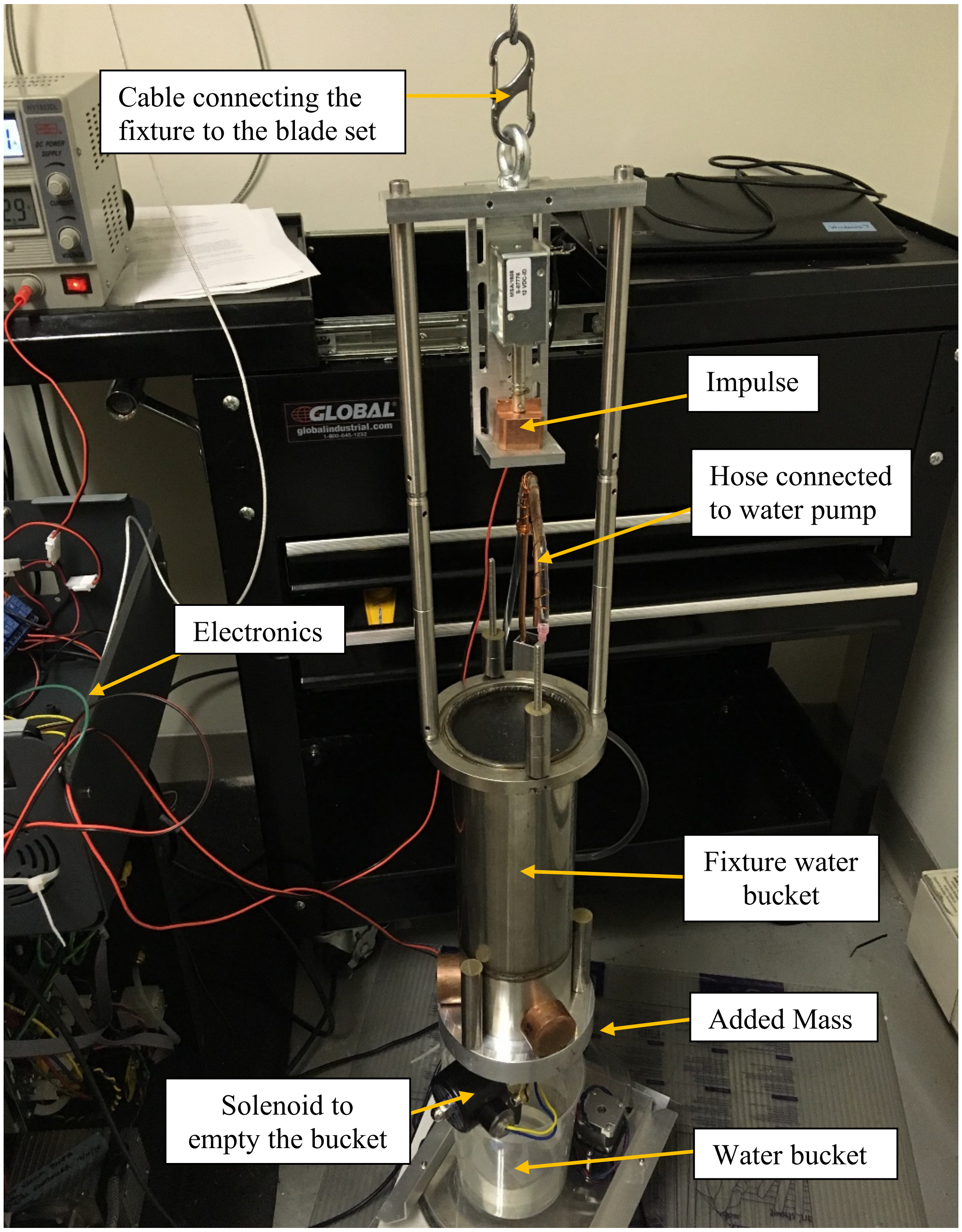
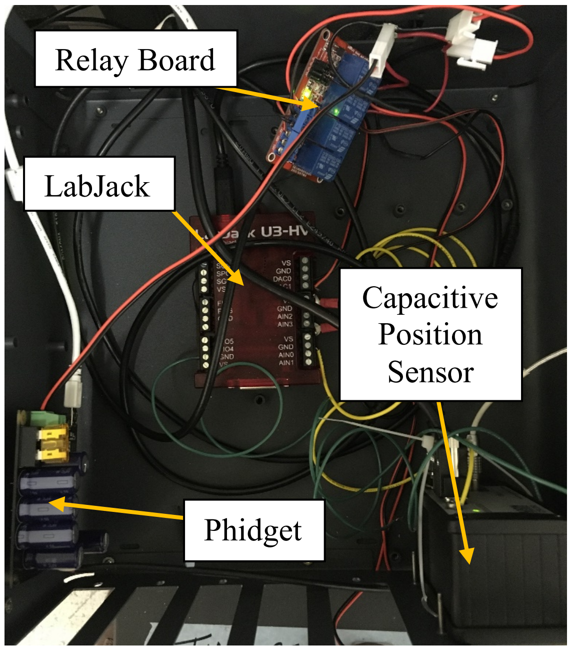

Cable connecting the fixture to the blade set
Impulse
Hose connected to water pump
Electronics
Fixture water bucket
Added Mass
Solenoid to empty the bucket
Water bucket

Figure 78. The Reynolds Fixture.



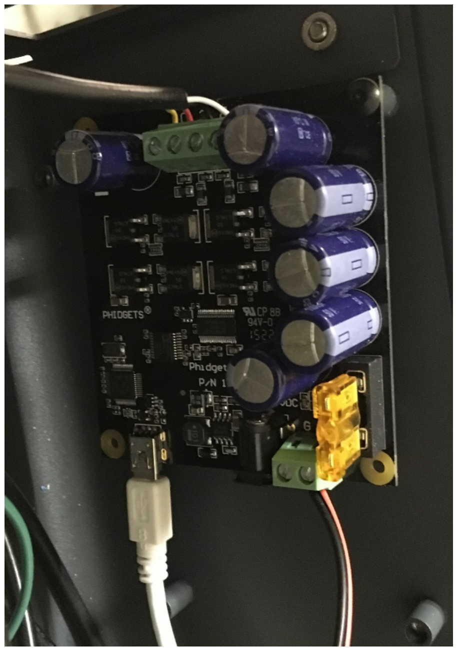


Figure 79. Electronics used to control the fixture (left) and the unit specifically used to control the water pump (right).

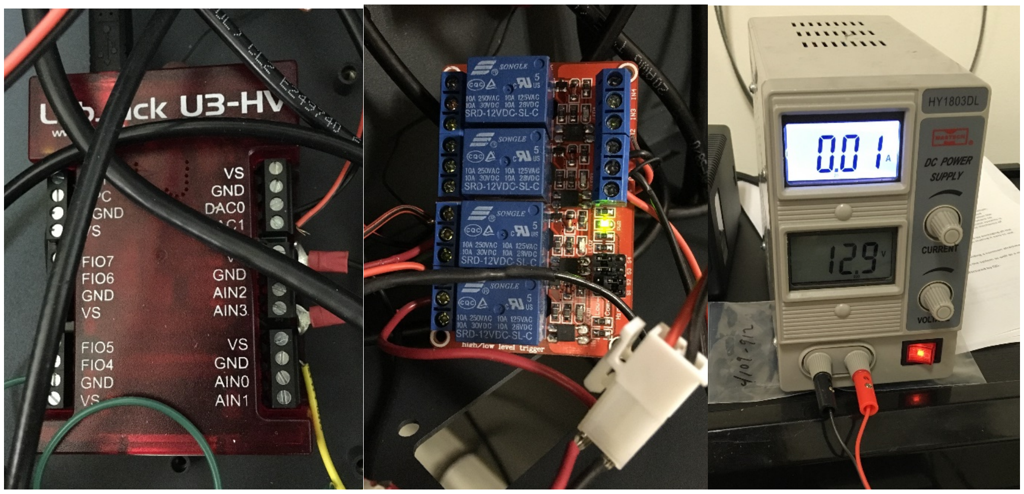


Figure 80. The electronics used to control both the impulse and the empty bucket solenoids. The left image is of the LabJack system, the middle image is the relay board connected to the LabJack, the solenoids and the power supply shown in the image on the right.

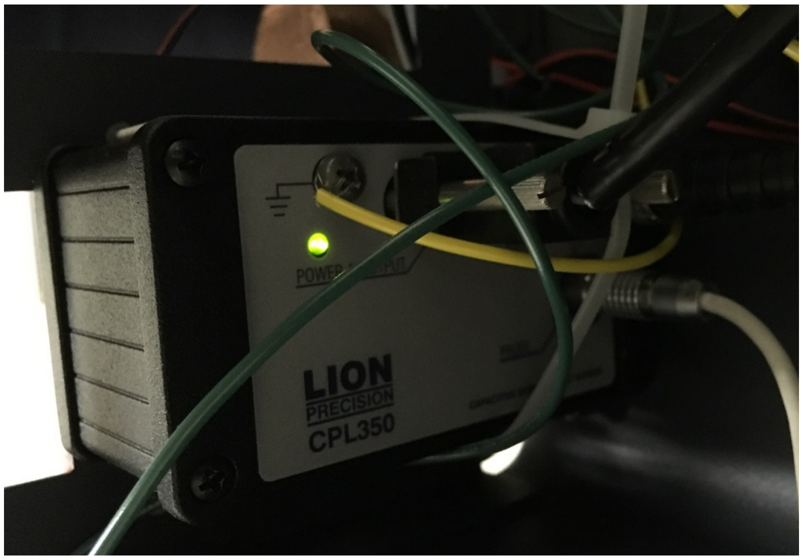


Figure 81. Electronics used to control the capacitive sensor.

D.5. In Case of Failure

There is no intelligent way to handle errors within the code. Therefore, in case of a mistake on the user's part (forgot to plug something in or remove the TWP fixture, etc.). The user should restart the kernel, empty the bucket by running the empty.py script, run the fft_operator_u3.py script, and then run the WILLmain.py script to start the next run. If an error message every appears stopping the program from completing a run, repeat this process.

D.6. Appendix

This section contains all scripts described in this document for reference along with a list of the python modules used with references to learn about their functionality.

D.6.1 WILLmain.py

```python
"""Created on Fri Mar 17 14:31:31 2017 @author: william.baker"""
import time
from time import gmtime, strftime
import datetime as dt
import numpy as np
from numpy import genfromtxt
from pump_operator import Pump_Operator
from fft_operator_u3 import Fft_Operator
#from Mass_step import next_mass
from sartorius import Sartorius
from checking_ports import serial_ports
from matplotlib.backends.backend_pdf import PdfPages #for outputting plots
import matplotlib.pyplot as plt
import os
import sys
import signal
#from subprocess import Popen

#this is used to break later if calibration is not done
kill_switch = 0
#ths is to keep track of a pass or fail
fail_switch = 0
#make sure the calibration file is there
cal_file ="TWP.txt"
if not os.path.exists(cal_file):
    file = open("TWP.txt", "w")
    file.close()
cal_check_val = genfromtxt('TWP.txt', delimiter=',')
#check to see if the calibration is loaded
if cal_check_val==0:
    print('system needs calibration')
    kill_switch = 1
else:
    calibratedworkingPoint = float(genfromtxt('TWP.txt', delimiter=','))
#constants
seconds_fft = 10; #seconds to sample data for the fft
mass_to_add = 5; #mass to add in each test
samples_to_run = 1000; #currently test runs a set number of times
```

```
base_mass = 5523.44;  #this is the mass of the mechanism and wet (caleb 3:30pm on
         6/11/2017)
impulse_time = 0.1; #time to run the silenoid
save_location = 'Z:\Engineering Design Files\MI-20 Floating Stage
Development\Testing\Individual GAS Tuning\Result Files\\'
TargetMass = 0;
#user inputs
extraMassString = '';
extraMass = 0.0;
partNumberBlades = '';
serialNumberBlades = '';
compressionPoint = '';
#initalize variables
port_check = serial_ports()
if 'COM3' in port_check: #checks to see if the scale is connected
   scale = Sartorius('COM3') #and conects to it
fft = Fft_Operator(seconds_fft); #initalizes the fft, also the labjack u3
a = Pump_Operator(); #initializes the pump

def initial_mass_approach(wait_time,offset):
   #wait_time = 5
   [distance] = Fft_Operator.dc_distance_measure(wait_time);
   #offset = 5 #this represents the near gap distance (ngd)
   while (distance < offset):
      Pump_Operator.dispence(20)
      time.sleep(1)
      [distance] = Fft_Operator.dc_distance_measure(wait_time);
      print("your current distance is "+ str(distance))

def starting_mass_approach(target_mass):
   global TargetMass
   global base_mass
   TargetMass = 0
   starting_mass = scale.value();
   mass = target_mass - base_mass - extraMass
   Pump_Operator.dispence(mass)
   time.sleep(5)
   current_mass = scale.value()
   mass_in_step = starting_mass - current_mass
   TargetMass = mass_in_step

def runMeasurement(user_mass):
   t0 = time.time()
   global seconds_fft;
   global compressionPoint;
   global mass_to_add;
```

```
    global impulse_time
    global fail_switch
    global partNumberBlades
    global serialNumberBlades
    global directory
    name = partNumberBlades + '_' +serialNumberBlades + '_' +compressionPoint + '_';
    directory = save_location + timeStamped(name)
    if not os.path.exists(directory):
        os.makedirs(directory)
    fail_count = 0
    #initialize the kill switch to later catch a failed exp
    fail_switch = 0
    #initialize a dict for data collection
Master_Data={'Mass(g)':[],'Res_freq(Hz)':[],'Distance(mm)':[],'Raw_data_flag(H:m:s)':[]
        ,'Spring Displacement(in)':compressionPoint}
    starting_mass = scale.value();
    pmass = starting_mass;
    #frequency_current = 10;  #dummy frequecy to feed into the calculation function
    #this step is inteded to bring the target into a range to begin the measurement (now is set
        to 10mm)
    initial_mass_approach(5,6)
    for i in range(samples_to_run):
        print("Starting Measurement")
#mass_to_add = Pump_Operator.dispense_calc(frequency_current);
#calculate the mass to add to the bucket
        #mass_to_add = 2.0;
        Pump_Operator.dispence(mass_to_add) #tell the pump to dispense that amount
        time.sleep(5) # let the scale settle
        mass = scale.value() #read the scale
        mass_added_in_step = pmass - mass; #calculate the mass added in the step
        print("Dispensed Mass: " + str(mass_added_in_step));
        j=0
        while mass_added_in_step < (0.5*mass_to_add):
            j+=1
            if j > 20:
                print("you have likely ran out of water, must abort and you should add course
                    mass")
                fail_switch = 1
                Fft_Operator.emptybucket();
                break;
            Pump_Operator.dispence((mass_to_add-
mass_added_in_step)+0.15)
            time.sleep(5) # let the scale settle
            mass_average = [0,0,0,0,0,0,0,0,0,0]
            for h in range(0,10):
                mass_average[h] = scale.value()
                time.sleep(2.0)
            mass = np.average(mass_average)
            mass_added_in_step = pmass - mass; #calculate the mass added in the step
            print("Dispensed Mass: " + str(mass_added_in_step));
        pmass = mass;
        total_mass = (starting_mass-mass)+base_mass+user_mass+TargetMass;
        #this line is a test to see if we can eliminate the drift-to-polyfit fail
        if mass_to_add == 1.0:
            time.sleep(1.0)
        if mass_to_add == 0.15:
            time.sleep(2.0)
        Fft_Operator.impulse(impulse_time);
        time.sleep(0.5)
        [frequency,abs_distance],RAW,RAW_time = Fft_Operator.measure(seconds_fft);
        print(RAW)
        ##saving the RAW voltage data in the chosen RAW data directory for furth
        #these lines will need to be un-commented to save the RAW data.
        time_RAW_aquired = strftime("%H-%M-%S", gmtime())
        RAW_text_location =  directory + '\\' + time_RAW_aquired + '.txt';
        file = open(RAW_text_location, "a")
        for l in range(len(RAW)):
            file.write(str(RAW[l]) +'\t'+str(RAW_time[l])+'\n')
        file.close()
        #this is a simple check for the failure of the polyfit x != y error, look to fft_operator
            to make sense of [0,0]
        if frequency == abs_distance ==0:
            fail_count += 1
            mass_to_add = 0
            if fail_count == 3:
                print('Most likely overcompressed, OR CAP sensor out of range? Try to un-
                        compress blades by a small amount')
                fail_switch = 1
                Fft_Operator.emptybucket();
                break;
        if frequency != abs_distance != 0:
            if frequency > 2.0:
                mass_to_add = 30.0
            if frequency < 2.0 and frequency > 1.26:
                mass_to_add = 25.0
            if frequency < 1.25 and frequency > 1.0:
                mass_to_add = 8.0
            if frequency < 0.99 and frequency > 0.70:
                mass_to_add = 3.5
            if frequency < 0.69 and frequency > 0.51:
                mass_to_add = 1.0
            if frequency < 0.5:
            mass_to_add = 0.2
          seconds_fft = 25.0* 1/frequency
        if mass_to_add != 0:
          fail_count = 0
        #grabbing the current theoretical working point (twp) from TWP.txt
        twp = genfromtxt('TWP.txt', delimiter=',')
        #this is where we will offset the abs distance to a reference to twp(calibration point)
        distance = abs_distance - twp
#         print('the current distance from FFT_Operator is..'+str(distance))
#         print('the current res freq from FFT_Operator is..'+str(frequency))
#         print('the current mass from FFT_Operator is..'+str(total_mass))
        #distance = abs_distance
        Master_Data['Mass(g)'].append(total_mass)
        Master_Data['Distance(mm)'].append(distance)
        Master_Data['Res_freq(Hz)'].append(frequency)
        Master_Data['Raw_data_flag(H:m:s)'].append(time_RAW_aquired)
        #this is checking for a collaped state, where the distance is not changing.
#         if i>5:
#           if (abs(((Master_Data['Distance(mm)'][i]) – (Master_Data['Distance(mm)'][i-5]))))
          < 0.1:
#               print('you likely have an overcompressed blade set...exiting ')
#               break;
        if frequency != 0:
          file = open("Data.txt", "r+")
          file.seek(0, 2)
          file.write( str(Master_Data['Mass(g)'][i]) + '\t'
                  +str(Master_Data['Res_freq(Hz)'][i])+'\t'+str(Master_Data['Distance(mm)'
                  ][i])+'\n' )
          file.close()
        #this is the finish statement, it requires that we pass the 1mm point(somewhere in
          middle) and also sees a res-frequency > 2Hz
        a = Master_Data['Res_freq(Hz)']
        a = np.array(a)
        min_frequency = np.min(a[np.nonzero(a)])
        if distance > 1.5 and frequency > ((Master_Data['Res_freq(Hz)'][0]-
min_frequency)/2 +  min_frequency):
        #if distance > 12 and frequency > 1.0:
          break
    time_duration=( time.time() - t0)/3600
    print("the time passed during measurment was "+str(time_duration)+"hours")
    Fft_Operator.emptybucket();
    #first try at a dump test, to see if the compression is passed a critical point, this should
        come after a (Fft_Operator.emptybucket();)
#   MIDindx = np.argmin(min(Master_Data['Res_freq(Hz)']))
#   MIDdistance = Master_Data['Distance(mm)'][MIDindx]
#   while (distance < MIDdistance):
```

```
#       Pump_Operator.dispence(50)
#       time.sleep(2)
#       [frequency, distance] = Fft_Operator.measure(wait_time);
#       if (distance > (MIDdistance + 5)):
#           print("failed the dump test!!")
#           break
    return Master_Data;

def imputCompressionPoint():
    global compressionPoint
    compressionPoint = raw_input("With the TWP fixture removed, record the absolute
        compression point of blade Base(X.XXXin): ")

def twp_grabber():
    global kill_switch
    ans = raw_input("Has the TWP been calibrated for this spring set? If no, install TWP
        fixture and enter 'n' to calibrate. (y/n)")
    if ans == 'n':
        measure_time=2
        #twp is the theoretical working point derived from the design of the GAS blades
        #the twp is defined by fixture that is bolted to isolation ring assembly and keystone
        twp = Fft_Operator.dc_distance_measure(measure_time)
        print (twp)
        twp=twp[0]
        file = open('TWP.txt','r+')
        file.write(str(twp))
        file.close()
        kill_switch = 0
        return;
    else:
        return;

def userInput():
    global extraMassString
    global extraMass
    global partNumberBlades
    global serialNumberBlades
    #the working point defines the theoretical (based of the GAS blade design) working
        point in the mechanical design. All distances are relative to the WorkingPoint
    twp_grabber()
    extraMassString = raw_input("How much additional mass was added(g):    ")
    extraMass = float(extraMassString);
    #extraMass = 2086.91
    partNumberBlades = raw_input("Part Number of Blades: ")
    #partNumberBlades = '4109-906'
    serialNumberBlades = raw_input("Blade seriel number?(alpha/lower
case):")
    #serialNumberBlades = 'ij'
    imputCompressionPoint();
    raw_input("Press enter to begin:")

def timeStamped(fname, fmt='{fname}_%m-%d-%Y-%H-%M-%S'):
    return dt.datetime.now().strftime(fmt).format(fname=fname)

def SMS(res_freq,distance,mass,PFmsg):
    import smtplib
    #fromaddr = 'email@gmail.com'
    #toaddrs  = 'email@gmail.com'
    msg = 'Run complete, and you have a res freq of'+str(res_freq)+'a working point of
            '+str(distance)+'and a supporting mass of '+str(mass)+'the test was a.. '+PFmsg
    username1 = 'email@gmail.com'
    password1 = 'password'
    server = smtplib.SMTP('smtp.gmail.com:587')
    server.starttls()
    server.login(username1,password1)
    #server.sendmail(fromaddr, toaddrs, msg)
    server.sendmail( 'email@gmail.com', 'number@txt.att.net', msg )
    server.sendmail( 'email@gmail.com', 'number@vtext.com', msg )
    server.quit()

def writeData(data):
    #initialize a dict for data collection
    global extraMassString
    global extraMass
    global directory
    for i in range(len(data)):
        Master_Data = data[i]
        displacement = Master_Data['Spring Displacement(in)']
        text_location = directory + '\\' + displacement + '.txt';
        file = open(text_location, "a")
     #   file.write('Blade Part Number: '+ partNumberBlades + serialNumberBlades+
            'compresssion point: '+ compressionPoint +'\n')

file.write('Mass(g)'+'\t'+'Res_freq(Hz)'+'\t'+'Distance(mm)'+'\t'+'Raw_data_flag(H:m:s)'+'
            \t'+'Spring Displacement(in)'+'\n')
        for i in range(len(Master_Data['Mass(g)'])):
            if i == 0:
                file.write( str(Master_Data['Mass(g)'][i]) + '\t' +
                        str(Master_Data['Res_freq(Hz)'][i])+'\t'+str(Master_Data['Distance(mm)'][
                        i])+str(Master_Data['Raw_data_flag(H:m:s)'][i] ) + '\t' + displacement + '\n'
                        )
            else:
```

```
            file.write(str(Master_Data['Mass(g)'][i]) + '\t'
                  +str(Master_Data['Res_freq(Hz)'][i])+'\t'+str(Master_Data['Distance(mm)'
                  ][i])+'\t'+str(Master_Data['Raw_data_flag(H:m:s)'][i])+'\n' )
        file.close()
        mass = Master_Data['Mass(g)'];
        frequency = Master_Data['Res_freq(Hz)'];
        distance = Master_Data['Distance(mm)'];
        fvm = plt.figure(3)
        ax1 = fvm.add_subplot(111)
        ax1.plot(mass, frequency, label = displacement )
        ax1.set_xlabel('Mass [grams]')
        ax1.set_ylabel('Frequency [Hertz]')
        ax1.legend(loc='upper center', shadow=True)
        ax1.set_yscale('log')#adjusts y-scale of live plotting
        fvd = plt.figure(2)
        ax2 = fvd.add_subplot(111)
        ax2.plot(distance, frequency, label = displacement )
        ax2.set_xlabel('Distance [mm]')
        ax2.set_ylabel('Frequency [Hertz]')
        ax2.legend(loc='upper center', shadow=True)
        ax2.set_yscale('log')#adjusts y-scale of live plotting
    pdf_location = directory + '\\' + 'plots.pdf';
    pp = PdfPages(pdf_location);
    pp.savefig(fvd)
    pp.savefig(fvm)
    pp.close();

def close():
    Pump_Operator.close();
    Fft_Operator.close();

def sigint_handler(signal, frame):
    print ('Interrupted Test, saving data')
    close();
    writeData(data);
    sys.exit(0)

signal.signal(signal.SIGINT, sigint_handler)
#gets the blade parameters from the user
userInput()
#run a test
data = []
os.remove("Data.txt")
file = open("Data.txt", "a")
file.write('Mass(g)'+'\t'+'Res_freq(Hz)'+'\t'+'Distance(mm)'+'\n')
file.close()
```

```
while True:
    if kill_switch == 1:
        print('system halted due to improper calibration')
        break;
    dc = runMeasurement(extraMass)
    dc['Spring Displacement(in)'] = compressionPoint
    data.append(dc)
    a = dc['Res_freq(Hz)']
    a = np.array(a)
    min_frequency = np.min(a[np.nonzero(a)])
    index_at_resFreq = np.argmin(a[np.nonzero(a)])
    working_point = dc['Distance(mm)'][index_at_resFreq]
    Supported_mass =  dc['Mass(g)'][index_at_resFreq]
    writeData(data);
    if fail_switch == 0:
        print('test was a success!')
        PFmsg = 'pass'
    if fail_switch == 1:
        print('test was a fail!')
        PFmsg = 'fail'
    print("The Minimun Frequency was:"+ str(min_frequency))
    print("With a working point of:"+ str(working_point))
    print("Supporting a mass of:"+ str(Supported_mass))
    SMS(min_frequency,working_point,Supported_mass,PFmsg)
    ans = raw_input('Would you like to go again? ("y" for yes, "n" for no): ')
    if ans == 'y':
        imputCompressionPoint();
        target_mass = Supported_mass - 100
        raw_input('Press Enter to Begin')
        erase_answer = raw_input("Overlay last? (y,n)")
        if erase_answer == 'n':
            os.remove("Data.txt")
            file = open("Data.txt", "a")
            file.write('Mass(g)'+'\t'+'Res_freq(Hz)'+'\t'+'Distance(mm)'+'\n')
            file.close()
        starting_mass_approach(target_mass)
    else:
        break;
#writeData(data);
#close and free resources
close();
```

D.6.2. pump_operator.py

```
"""Created on Thursday Mar 16 10:36:44 2017 @author: David Schwarz"""
import math
from Phidgets.Devices.Stepper import Stepper
from Phidgets.PhidgetException import PhidgetException

class Pump_Operator():
    stepper = 0;
        #Information Display Function
    def DisplayDeviceInfo(self):
        print("|--------------|--------------------------|----------------|--------------|")
        print("|- Attached -|-           Type           -|- Serial No. -|-  Version -|")
        print("|--------------|--------------------------|----------------|--------------|")
        print("|- %8s -|- %43s -|- %10d -|- %8d -|" %(Pump_Operator.stepper.isAttached(),
            Pump_Operator.stepper.getDeviceName(),Pump_Operator.stepper.getSerialNum(
            ), Pump_Operator.stepper.getDeviceVersion()))
        print("|--------------|--------------------------|----------------|--------------|")
        print("Number of Motors: %i" %
(Pump_Operator.stepper.getMotorCount()))

    def __init__(self):
        try:
            Pump_Operator.stepper = Stepper()
        except:
            print("failed to open stepper")
        try:
            Pump_Operator.stepper.openPhidget()
        except PhidgetException as e:
            print("Phidget Exception %i: %s" % (e.code, e.details))
            print("Exiting....")
            exit(1)
        print("Waiting for attach to Pump")

        try:
            Pump_Operator.stepper.waitForAttach(1000)
        except PhidgetException as e:
            print("Phidget Exception %i: %s" % (e.code, e.details))
            try:
                Pump_Operator.stepper.closePhidget()
            except PhidgetException as e:
                print("Phidget Exception %i: %s" % (e.code, e.details))
                print("Exiting....")
                exit(1)
            print("Exiting....")
            exit(1)
        else:
            #self.DisplayDeviceInfo()
            print('Connected to Stepper Motor')
    @classmethod
    def dispence(cls,mass_to_add):
        ml_step = 0.0022;
        gram_per_ml = 1;
        phidgits_step_unit = 0.0625;
        ml_to_add = mass_to_add*gram_per_ml;
        #this is the number of steps the pump will turn to put out the desired amount
        #in reality the pump is not perfect and doesnt have a linear output per step
        steps= int(ml_to_add/(ml_step*phidgits_step_unit));
        print("Dispensing: %s grams in %s steps" % ( mass_to_add,
float(steps)))
        try:
            Pump_Operator.stepper.setCurrentPosition(0, 0)
            Pump_Operator.stepper.setEngaged(0, True)
            Pump_Operator.stepper.setAcceleration(0, 3048)
            Pump_Operator.stepper.setVelocityLimit(0, 20000)
            Pump_Operator.stepper.setCurrentLimit(0, 1.00)
            Pump_Operator.stepper.setTargetPosition(0, steps)
            while Pump_Operator.stepper.getCurrentPosition(0) != steps:
                pass
        except PhidgetException as e:
            print("Phidget Exception %i: %s" % (e.code, e.details))
            print("Exiting....")
        try:
            Pump_Operator.stepper.setEngaged(0, False)
        except PhidgetException as e:
           print("Phidget Exception %i: %s" % (e.code, e.details))
           print("Exiting....")
        print("Done Dispensing")
        return steps;

    @classmethod
    def dispense_calc(cls,frequency_current):
        frequency = [0.5,1,2]
        mass_step = [0.1,3,9]
        for i in range(len(frequency)-1):
            if(frequency_current<frequency[i]):
                return mass_step[i];

    def time_calc(start_mass,mass_to_add):
        #all distances are measured in cm time in sec
        #radius of the holding container
        radius_big = 14.0
        #radius of port at bottom of holding tank
        radius_small = 0.5
        #desity of water in g/cm^3
        roh_water=1.0
        #constant of gravity in cm/sec^2
        g = 980.0
        height = start_mass/(roh_water*math.pi*radius_big**2)
        time_interval = mass_to_add/(roh_water*math.pi*radius_small**2*math.sqrt(2*g
          *height))
        return time_interval

    @classmethod
    def close(cls):
        Pump_Operator.stepper.closePhidget()
```

D.6.3. fft_operator_u3.py

```
"""Created on Fri Mar 17 14:31:31 2017 @author: david.schwarz"""
import u3
import traceback
from datetime import datetime
import time
import numpy as np
from numpy import arange
from numpy import argmax, mean, diff, log
import matplotlib.pyplot as plt
from parabolic import parabolic, parabolic_polyfit

limit = 100000; #number of samples to use for rising edge measurment
average_length = 400; #number of samples to forward average

class Fft_Operator():
    d = 0;
    seconds = 10;
    frequency_measurement_method = 1; # zero for fft, 1 for rising / fallling edge
    empty_time = 8; #seconds

    def __init__(self, seconds_to_measure):
        Fft_Operator.seconds = seconds_to_measure;
        Fft_Operator.d = u3.U3()
        # To learn the if the U3 is an HV
        Fft_Operator.d.configU3()
        #Fft_Operator.d.streamStop()
        # For applying the proper calibration to readings.
        Fft_Operator.d.getCalibrationData()
          #to find channel numbers go to this address
https://labjack.com/support/datasheets/u3/hardware-description/ain/channel_numbers
        print ("configuring U3 stream")
        Fft_Operator.d.streamConfig( NumChannels = 1, PChannels = [ 3 ], NChannels = [31
          ],
Resolution = 3, SamplesPerPacket=25, SampleFrequency=10000 )
  # Fft_Operator.d.getFeedback(u3.DAC1_8(Value = 255))
  # Fft_Operator.d.getFeedback(u3.DAC1_8(Value = 0))

    @classmethod
    def emptybucket(cls):
        print("Opening Empty Silenoid")
        #opening the solenoid
        cls.d.getFeedback(u3.DAC0_8(Value = 255))
        time.sleep(cls.empty_time)
        #time.sleep(0.5)
        cls.d.getFeedback(u3.DAC0_8(Value = 0))
        print('Closing Empty Silenoid')

    @classmethod
    def impulse(cls, delay):
        print("\a")
        #opening the solenoid
        print("Activating Impulse Silenoid")
        cls.d.getFeedback(u3.DAC1_8(Value = 255))
        time.sleep(delay) #wait for it to go up
        cls.d.getFeedback(u3.DAC1_8(Value = 0))
        print('De-Energizing Impulse Silenoid')
        time.sleep(delay)

    @classmethod
    def measure_rising_edge(cls,data, runTime):
        #time of measurment array
        #data = data[0:limit]
        time = np.linspace(0.0,runTime,num=len(data))
        #create normal data to read peaks from
        average_voltage = np.mean(data)
        normal_data = data-average_voltage; #centers the data at 0;
        range_voltage = np.max(normal_data)-np.min(normal_data);
        normal_data = normal_data/range_voltage; #creates a standard peak to peak (1)
        #average the readings to smooth them out
        for i in range(len(data) - average_length)[2:]:
            #average data
            normal_data[i] = np.mean(normal_data[i:i+average_length])
        #this should trim the data length should probably be fixed later
        normal_data = normal_data[:(len(data)-average_length)]
        time = time[:(len(data)-average_length)]
        plt.figure(10)
        plt.clf()
        plt.subplot(3,1,1)
        plt.plot(time,normal_data)
        #square the data
        top = np.where(normal_data>0);
        bottom = np.where(normal_data<=0)
        sq = np.zeros(len(normal_data))
        sq[top] = 0.5; #arbitrary,
        sq[bottom] = -0.5;
        #plot the square data in the second subplot
        plt.figure(10)
        plt.subplot(3,1,2)
        plt.plot(time,sq)
        #compute the rising edges of the square data
        pv = 0;
        pt = 0;
        ptp = [];
        for i in range(len(sq) - average_length)[2:]:
            #eliminate noise on edge of signal
            if sq[i] != pv:
                pv = sq[i];
                sq[i:(i+average_length)] = pv;
            if sq[i] < sq[i+1]:
                pt = time[i];
            if sq[i] > sq[i+1]:
                if pt>0: #ignore the begining
                    ptp.append(time[i]-pt)
        #plot the the period in the third subplot
        freqs = np.divide(1.0,2*ptp[2:]); #ignore the first value
        plt.figure(10)
        plt.subplot(3,1,3)
        plt.plot(freqs)
        distance = (5+ 6.250 - average_voltage/1.6);
        freq_in_hertz = np.mean(freqs);
        print('your res freq is ' + str(freq_in_hertz) + 'Hz' )
        ans=[freq_in_hertz, distance]
        return ans

    @classmethod
    def dc_distance_measure(cls,measure_time):
        Fft_Operator.seconds = measure_time;
        abs_distance = 0;
#cls.d.streamConfig( NumChannels = 1, PChannels = [ 3 ], NChannels = [ 31 ], Resolution
= 3, ScanFrequency = 200 )
        cls.d.streamConfig( NumChannels = 1, PChannels = [ 3 ], NChannels = [ 31 ],
            Resolution = 3, SamplesPerPacket=25,
SampleFrequency=20000)
        cls.d.streamStart()
        start = datetime.now()
```

```
print("start stream time: ")
print(start)
missed = 0
dataCount = 0
packetCount = 0
time_duration = 0
t0 = time.time()
data = []
try:
   for r in cls.d.streamData():
      if r is not None:
         # Our stop condition
         if time_duration >= cls.seconds:
            break
         if r['errors'] != 0:
            print( "Error: %s ; " % r['errors'], datetime.now())
         if r['numPackets'] != cls.d.packetsPerRequest:
            print("----- UNDERFLOW : %s : " % r['numPackets'], datetime.now())
         if r['missed'] != 0:
            missed += r['missed']
            print( "+++ Missed ", r['missed'])
         # Comment out these prints and do something with r
         #print r['AIN2']
         data = data + r['AIN3']
#           print "Average of" , len(r['AIN0']), "AIN0," , len(r['AIN1']) , "AIN1
            reading(s):",
#           print sum(r['AIN0'])/len(r['AIN0']) , "," , sum(r['AIN1'])/len(r['AIN1'])
         time_duration = time.time() - t0
         #print (time_duration)
         dataCount += 1
         packetCount += r['numPackets']
      else:
         # Got no data back from our read.
         # This only happens if your stream isn't faster than the
         # the USB read timeout, ~1 sec.
         print("No data", datetime.now())
except KeyboardInterrupt:
   return
except:
   print ("".join(i for i in traceback.format_exc()))
finally:
   stop = datetime.now()
   cls.d.streamStop()
   print ("stream stopped.")
   #print data
   sampleTotal = packetCount * cls.d.streamSamplesPerPacket
```

```
            scanTotal = sampleTotal / 2 #sampleTotal / NumChannels
        #   print "%s requests with %s packets per request with %s samples per packet = %s
            samples total." % (dataCount, (float(packetCount) / dataCount),
            d.streamSamplesPerPacket, sampleTotal )
            print("%s samples were lost due to errors." % missed)
            sampleTotal -= missed
            print("Adjusted number of samples = %s" % sampleTotal)
            runTime = (stop-start).seconds + float((stop-
start).microseconds)/1000000
            print("The experiment took %s seconds." % runTime)
            print("Scan Rate : %s scans / %s seconds = %s Hz" % ( scanTotal,
runTime, float(scanTotal)/runTime ))
            print("Sample Rate : %s samples / %s seconds = %s Hz" % (
sampleTotal, runTime, float(sampleTotal)/runTime ))
            data = np.array(data)
            #Time = np.linspace(0.0,runTime,num=len(data))
            #plt.axes(xlim=(0, runTime), ylim=(data.min(), data.max()))
            #plt.plot(Time,data)
            np.savetxt("foo.csv", data, delimiter=",")
            #return(cls.measure_rising_edge(data,runTime))
        #find the new distance of the fixture
        #"abs.distance" is the abosolute measurement from the face of the cap sensor to target
        average_voltage = np.mean(data)
        #currently using 12.5mm (C25-21)LION capacitive sensor: variable below represent
            it's specs
        ngd = 5 #near gap distance
        mid_range = 6.25 #distance to near gap at 0 volts (mm)
        sensitivity = 1.6 #sensitity of sensor (V/mm)
        abs_distance = (ngd+ mid_range - average_voltage/sensitivity);
        ans=[abs_distance]
        return ans

    @classmethod
    def measure(cls,measure_time):
    #start the measurment
        Fft_Operator.seconds = measure_time;
        kill_switch = 0;
        freq_in_hertz = 0;
        abs_distance = 0;
        #cls.d.streamConfig( NumChannels = 1, PChannels = [ 3 ], NChannels = [ 31 ],
            Resolution = 3, ScanFrequency = 200 )
        cls.d.streamConfig( NumChannels = 1, PChannels = [ 3 ], NChannels = [ 31 ],
            Resolution = 3, SamplesPerPacket=25,
SampleFrequency=20000)
        cls.d.streamStart()
        start = datetime.now()
```

```
print("start stream time: ")
print(start)
missed = 0
dataCount = 0
packetCount = 0
time_duration = 0
t0 = time.time()
data = []
try:
    for r in cls.d.streamData():
        if r is not None:
            # Our stop condition
            if time_duration >= cls.seconds:
                break
            if r['errors'] != 0:
                print( "Error: %s ; " % r['errors'], datetime.now())
            if r['numPackets'] != cls.d.packetsPerRequest:
                print("----- UNDERFLOW : %s : " % r['numPackets'], datetime.now())
            if r['missed'] != 0:
                missed += r['missed']
                print( "+++ Missed ", r['missed'])
                kill_switch = 1;
            # Comment out these prints and do something with r
            #print r['AIN2']
            data = data + r['AIN3']
#            print "Average of" , len(r['AIN0']), "AIN0," , len(r['AIN1']) , "AIN1
    reading(s):",
#            print sum(r['AIN0'])/len(r['AIN0']) , "," , sum(r['AIN1'])/len(r['AIN1'])
            time_duration = time.time() - t0
            #print (time_duration)
            dataCount += 1
            packetCount += r['numPackets']
        else:
            # Got no data back from our read.
            # This only happens if your stream isn't faster than the
            # the USB read timeout, ~1 sec.
            print("No data", datetime.now())
except KeyboardInterrupt:
    return
except:
    print ("".join(i for i in traceback.format_exc()))
finally:
    stop = datetime.now()
    cls.d.streamStop()
    print ("stream stopped.")
    sampleTotal = packetCount * cls.d.streamSamplesPerPacket
```

```
        runTime = (stop-start).seconds + float((stop-
start).microseconds)/1000000
        time_data=np.linspace(0,runTime,num=sampleTotal)
        if(kill_switch == 1):
           print("Some failure has occurred, packets missed")
           data = np.array(data)
           ans=[0, 0],data,time_data
           return ans;
        #print data
        scanTotal = sampleTotal / 2 #sampleTotal / NumChannels
     #   print "%s requests with %s packets per request with %s samples per packet = %s
        samples    total."    %    (dataCount,    (float(packetCount)    /    dataCount),
        d.streamSamplesPerPacket, sampleTotal )
        print("%s samples were lost due to errors." % missed)
        sampleTotal -= missed
        print("Adjusted number of samples = %s" % sampleTotal)
        print("The experiment took %s seconds." % runTime)
        print("Scan Rate : %s scans / %s seconds = %s Hz" % ( scanTotal,
runTime, float(scanTotal)/runTime ))
        print("Sample Rate : %s samples / %s seconds = %s Hz" % (
sampleTotal, runTime, float(sampleTotal)/runTime ))
        data = np.array(data)
        #Time = np.linspace(0.0,runTime,num=len(data))
        #plt.axes(xlim=(0, runTime), ylim=(data.min(), data.max()))
        #plt.plot(Time,data)
        np.savetxt("foo.csv", data, delimiter=",")
        #return(cls.measure_rising_edge(data,runTime))
        w = np.fft.fft(data)
        w[0]=0.0
        plt.axes(xlim=(0, 2), ylim=(0, 1000))
        sample_rate = float(sampleTotal)/runTime
        freqs = np.fft.fftfreq(len(w),1/sample_rate)
        idx = np.argmax(np.abs(w))
        #true_idx = parabolic(log(np.abs(w)), idx)[0]
        f = np.abs(w)
        x = idx
        n=2
        check_array1 = arange(x-n//2, x+n//2+1)
        check_array2 = f[x-n//2:x+n//2+1]
        len1 = len(check_array1)
        len2 = len(check_array2)
        if(len1 != len2):
           print("Some failure has occured, polyfit will fail under current conditions")
           ans=[0, 0],data,time_data
           return ans;
        true_idx = parabolic_polyfit(np.abs(w), idx, n)[0]
        True_idx_low = int(true_idx)
        True_idx_high = True_idx_low + 1
        percent_to_add = true_idx - True_idx_low
        difference = freqs[True_idx_high]-freqs[True_idx_low]
        amount_to_add = percent_to_add*difference
        freq = freqs[True_idx_low]+amount_to_add
        freq_in_hertz = abs(freq)
        plt.axes(xlim=(-5, 5))
        plt.plot(freqs,np.abs(w))
        print('you res freq is ' + str(freq_in_hertz) + 'Hz' )
    #    439.8975
    #find the new distance of the fixture
    #"abs.distance" is the abosolute measurement from the face of the cap sensor to target
    average_voltage = np.mean(data)
    #currently using 12.5mm (C25-21)LION capacitive sensor: variable below represent
        it's specs
    ngd = 5 #near gap distance
    mid_range = 6.25 #distance to near gap at 0 volts (mm)
    sensitivity = 1.6 #sensitity of sensor (V/mm)
    abs_distance = (ngd+ mid_range - average_voltage/sensitivity);
    ans=[freq_in_hertz, abs_distance]
    return ans,data,time_data

  @classmethod
  def close(cls):
    cls.d.close();
```

D.6.4. u3.py

Not included for the sake of the length of this document. If interested in the main code see: http://labjack.com/support/u3/users-guide/5.2

D.6.5. parabolic.py

```
"""Created on Mon Apr 10 16:41:37 2017 @author: will.b"""
from __future__ import division
from numpy import polyfit, arange

def parabolic(f, x):
"""Quadratic interpolation for estimating the true position of an inter-sample maximum
when nearby samples are known. f is a vector and x is an index for that vector. Returns (vx,
vy), the coordinates of the vertex of a parabola that goes through point x and its two
neighbors. Example: Defining a vector f with a local maximum at index 3 (= 6), find local
maximum if points 2, 3, and 4 actually defined a parabola. In [3]: f = [2, 3, 1, 6, 4, 2, 3, 1]
In [4]: parabolic(f, argmax(f)) Out[4]: (3.2142857142857144, 6.1607142857142856)"""
  xv = 1/2. * (f[x-1] - f[x+1]) / (f[x-1] - 2 * f[x] + f[x+1]) + x
  yv = f[x] - 1/4. * (f[x-1] - f[x+1]) * (xv - x)
  return (xv, yv)
```

```
def parabolic_polyfit(f, x, n):
"""Use the built-in polyfit() function to find the peak of a parabola. f is a vector and x is an
index for that vector. n is the number of samples of the curve used to fit the parabola."""
    a, b, c = polyfit(arange(x-n//2, x+n//2+1), f[x-n//2:x+n//2+1], 2)
    xv = -0.5 * b/a
    yv = a * xv**2 + b * xv + c
    return (xv, yv)

if __name__=="__main__":
    from numpy import argmax
    import matplotlib.pyplot as plt
    y = [2, 1, 4, 8, 11, 10, 7, 3, 1, 1]
    xm, ym = argmax(y), y[argmax(y)]
    xp, yp = parabolic(y, argmax(y))

    plot = plt.plot(y)
    plt.hold(True)
    plt.plot(xm, ym, 'o', color='silver')
    plt.plot(xp, yp, 'o', color='blue')
    plt.title('silver = max, blue = estimated max')
```

D.6.6. sartorius.py

```
"""Python Interface for Sartorius Serial Interface for EA, EB, GD, GE, TE scales. 2010-
2011 Robert Gieseke - robert.gieseke@gmail.com See LICENSE."""
import serial
class Sartorius(serial.Serial):
    def __init__(self, com_port):
        """Initialise Sartorius device. Example: scale = Sartorius('COM1')"""
        serial.Serial.__init__(self, com_port)
        self.baudrate = 9600
        self.bytesize = 7
        self.parity = serial.PARITY_ODD
        self.timeout = 0.5

    def value(self):
        """Return displayed scale value."""
        answer = 'na'
        while True:
            try:
                if self.inWaiting() == 0:
                    self.write('\033P\n'.encode())
                answer = self.readline()
                if len(answer) == 16: # menu code 7.1.1
                     answer = float(answer[0:11].decode(encoding='UTF-8').replace(' ', ''))
                else: # menu code 7.1.2
                    answer = float(answer[6:17].decode(encoding='UTF-8').replace(' ',''))
            except Exception:
                answer = "NA"
            if (type(answer) is float):
                return answer
            import time as tm
            tm.sleep(0.5)
        return answer

    def display_unit(self):
        """ Return unit."""
        self.write('\033P\n'.encode())
        answer = self.readline()
        try:
            answer = answer[11].decode(encoding='UTF-8').strip()
        except:
            answer = ""
        return answer

    def tara_zero(self):
        """Tara and zeroing combined."""
        self.write('\033T\n'.encode())

    def tara(self):
        """Tara."""
        self.write('\033U\n'.encode())

    def zero(self):
        """Zero."""
        self.write('\033V\n'.encode())
```

D.6.7. checking_ports.py

```
"""Created on Fri Feb 24 08:19:54 2017 @author: william.baker"""
import sys
import glob
import serial

def serial_ports():
""" Lists serial port names. raises EnvironmentError: On unsupported or unknown
platforms. returns: A list of the serial ports available on the system"""
    if sys.platform.startswith('win'):
        ports = ['COM%s' % (i + 1) for i in range(256)]
    elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):
        # this excludes your current terminal "/dev/tty"
        ports = glob.glob('/dev/tty[A-Za-z]*')
    elif sys.platform.startswith('darwin'):
        ports = glob.glob('/dev/tty.*')
    else:
        raise EnvironmentError('Unsupported platform')
    result = []
    for port in ports:
        try:
            s = serial.Serial(port)
            s.close()
            result.append(port)
        except (OSError, serial.SerialException):
            pass
    return result

if __name__ == '__main__':
    print(serial_ports())
```

D.6.8. empty.py

```
"""Created on Wed Apr 05 10:15:57 2017 @author: david.schwarz"""
from fft_operator_u3 import Fft_Operator
impulse_time = 0.2; #time to run the silenoid
seconds_fft = 15.0; #seconds to sample data for the fft
fft = Fft_Operator(seconds_fft); #initalizes the fft, also the labjack u3
Fft_Operator.emptybucket();
Fft_Operator.close();
```

D.6.9. stream_stop.py

```
"""Created on Wed May 31 11:04:36 2017 @author: william.baker"""
from fft_operator_u3 import Fft_Operator
#impulse_time = 0.2; #time to run the silenoid
seconds_fft = 15; #seconds to sample data for the fft
fft = Fft_Operator(seconds_fft); #initalizes the fft, also the labjack u3
#Fft_Operator.emptybucket();
#Fft_Operator.close();
fft.d.streamStop()
```

D.6.10. Python Modules

time:	"15.3. Time — Time Access and Conversions." 15.3. Time - Time Access and Conversions — Python 2.7.13 Documentation. Python, 27 May 2017. Web. 09 June 2017.
datetime:	"8.1. Datetime — Basic Date and Time Types." 8.1. Datetime - Basic Date and Time Types — Python 2.7.13 Documentation. Python, 27 Mar. 2017. Web. 09 June 2017.
numpy:	"NumPy." NumPy — NumPy. NumFocus, 2017. Web. 09 June 2017.

traceback:	Hellmann, Doug. "PyMOTW." Traceback – Extract, Format, and Print Exceptions and Stack Traces. - Python Module of the Week. N.p., 30 Apr. 2017. Web. 09 June 2017.
glob:	"10.7. Glob — Unix Style Pathname Pattern Expansion." 10.7. Glob - Unix Style Pathname Pattern Expansion — Python 2.7.13 Documentation. Python, 27 Mar. 2017. Web. 14 June 2017.
Matplotlib.backends.backend_pdf:	Sukhbinder. "Pdf With Matplotlib." SukhbinderSingh.com. N.p., 09 Sept. 2015. Web. 14 June 2017.
Matplotlib.pyplot:	Hunter, John, Darren Dale, Eric Firing, and Michael Droettboom. "Pyplot Tutorial." Pyplot Tutorial — Matplotlib 2.0.2 Documentation. N.p., 10 May 2017. Web. 14 June 2017.
os:	"15.1. Os — Miscellaneous Operating System Interfaces." 15.1. Os - Miscellaneous Operating System Interfaces — Python 2.7.13 Documentation. Python, 27 Mar. 2017. Web. 14 June 2017.
sys:	"28.1. Sys — System-specific Parameters and Functions." 28.1. Sys - System-specific Parameters and Functions — Python 2.7.13 Documentation. Python, 27 Mar. 2017. Web. 14 June 2017.
signal:	"18.8. Signal — Set Handlers for Asynchronous Events." 18.8. Signal - Set Handlers for Asynchronous Events — Python 3.6.1 Documentation. Python, 27 Mar. 2017. Web. 14 June 2017.
subprocess:	"17.1. Subprocess — Subprocess Management." 17.1. Subprocess - Subprocess Management — Python 2.7.13 Documentation. Python, 27 Mar. 2017. Web. 14 June 2017.
smtplib:	"20.12. Smtplib — SMTP Protocol Client." 20.12. Smtplib - SMTP Protocol Client — Python 2.7.13 Documentation. Python, 27 Mar. 2017. Web. 14 June 2017.