DESIGN AND VERIFICATION OF A LEVITATION DEVICE

by

Brittney Lambert

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Mechanical Engineering

Charlotte

2019

Approved by:

____

Dr. Steve Patterson

____

Dr. Kevin Lawton

____

Dr. Jim Conrad

ABSTRACT

BRITTNEY LAMBERT. Design and verification of a levitation device. (Under the direction of DR. STEVE PATTERSON)

This thesis describes the design and verification of a force measurement instrument using levitation through use of a voice coil actuator. The device can suspend an external force of up to 10 N, and determine that external force to within ±0.003 N. Mechanical design, electrical design both analog and digital, manufacturing processes, and the resulting measurements are described. Flexures, voice coil design, positioning sensors, and control systems are all integrated through use of an additively manufactured frame. Design objectives to limit manufacturing time and complexity, provide the ability to make design changes easily, and provide proof of concept feasibility are addressed.

DEDICATION

This thesis is dedicated to my husband and family who provided support and encouragement along the way.

ACKNOWLEDGEMENTS

I would like to acknowledge my advisors, Dr. Steven Patterson, for his never ending wisdom and patience, and Dr. Kevin Lawton for his wisdom, encouragement, and guidance along the way. A special thanks to Dr. Jeffrey Raquet, for all of his help and knowledge in the additive manufacturing lab, and to Joe Dalton for his guidance and help in the machine shop.

TABLE OF CONTENTS

LIST OF TABLES viii

LIST OF FIGURES ix

LIST OF ABBREVIATIONS 1

CHAPTER 1: INTRODUCTION 1

- 1.1. RELATED WORK 1
- 1.2. Force Measurement Systems 1
- 1.3. Watt Balance 2

CHAPTER 2: OVERALL DESIGN 4

CHAPTER 3: MECHANICAL DESIGN 7

- 3.1. Voice Coil and Magnetic Field Design 7
- 3.2. Flexures 8
  - 3.2.1. Flexure for Armature Movement 8
  - 3.2.2. Flexure for OKE Sensor Positioning 10

CHAPTER 4: ELECTRICAL AND CONTROLS DESIGN 12

- 4.1. DAQ 12
- 4.2. OKE Sensor Circuit 13
- 4.3. Power Amplifier Circuit 14
- 4.4. PI Analog Controller Circuit 15
- 4.5. Analog Summation Circuit 16
- 4.6. Digital and Analog Control 16

CHAPTER 5: MANUFACTURING 18

5.1. Coil 18

5.2. Additive Manufacturing 19

5.3. Assembly 19

CHAPTER 6: RESULTS 22

6.1. Calibration Methods 22

6.2. Testing 26

6.2.1. Force Constant 26

6.2.2. Flexure Stiffness 27

CHAPTER 7: CONCLUSIONS AND FUTURE WORK 29

REFERENCES 30

APPENDIX A: C++ Code 31

APPENDIX B: Mathcad Analysis 70

APPENDIX C: Matlab Analysis 71

LIST OF TABLES

TABLE 4.1: LabJack Inputs and Outputs 13

LIST OF FIGURES

FIGURE 2.1: Basic Design Concept 5

FIGURE 2.2: Overall Design 6

FIGURE 2.3: Voice Coil and OKE Sensor Housing Exploded View 6

FIGURE 3.1: Magnetic Circuit Design with Magnetic Field Lines Normal to Coil 7

FIGURE 3.2: FEMM Model 8

FIGURE 3.3: Opposing Modified Blade Flexure Design Allowing Single-Axis Motion 9

FIGURE 3.4: FEA analysis of flexure design with 5 N load applied to armature; units in mm 10

FIGURE 3.5: Position Sensor Bridge with Flexures and Micrometer 11

FIGURE 4.1: Control Diagram 12

FIGURE 4.2: OKE circuit 14

FIGURE 4.3: Power Amplifier Circuit 15

FIGURE 4.4: PI Analog Control Circuit 16

FIGURE 4.5: Analog Summation Circuit 16

FIGURE 4.6: Residual error for a succession of five loading masses: 272 g, 282 g, 292 g, 322 g and 372 g with digital integrator implementation 17

FIGURE 5.1: Manufactured Coil Inside ABS Coil Housing 18

FIGURE 5.2: Stops to prevent over extending flexures during assembly 20

FIGURE 5.3: Exploded View of System Components for Assembly 21

FIGURE 6.1: Voltage to Position of OKE Position Sensor moved with micrometer with Linear Fit line 23

FIGURE 6.2: Nonlinearity of OKE Position Sensor 23

FIGURE 6.3: Voice coil system set-up 25

FIGURE 6.4: Coil current and position for a succession of six loading masses: 272 g, 282 g, 292 g, 322 g, 372 g and 772 g 26

FIGURE 6.5: Force vs Current with best fit line for determining force constant C 27

FIGURE 6.6: Measured Flexure Stiffness with 5 Known Forces at 12 positions with linear fits to each set of measurements 27

CHAPTER 1: INTRODUCTION

A load cell is a mechanism that takes an unknown applied force and outputs a quantifiable signal that can be related to the unknown force. The unknown applied force is determined by looking at the signal change induced when a load is applied. A force balance uses a known force to balance an unknown applied force. When the force is balanced, the known force is equal to the unknown force. In an electromagnetic force balance, using a voice coil actuator, the voice coil balances the unknown applied force. When the unknown applied force is balanced, the current required to drive the voice coil can then be related to the unknown applied force. Measurements can also be taken while moving the coil through a magnetic field and used to determine the magnetic flux of the system. While strain gauge load cells are common, a force balance has the potential to be much more precise. An electromagnetic force balance is even more desirable because the system can be measured to determine the magnetic flux, and that can then be used to determine the applied force, making the measurement more precise.

1.1 RELATED WORK

1.2 Force Measurement Systems

One force balance apparatus created for atomic force microscope probe calibration at UNC Charlotte [1], uses flexures that suspend a rod. When a force is applied to the rod, the flexures displace. That displacement is then measured with a capacitance gauge, and then a signal is fed through a PID controller and a power amplifier provides current to a solenoid to keep the system at a set null point in position. The system has the ability to measure forces of up to 100 mN with a resolution of $70\,\mathrm{nN}/\sqrt{\mathrm{Hz}}$.

A patent by Feliks Bator shows a design for rapid weighing [2] through the use of a voice coil. A tray is used to hold a weight and is attached to a spring or flexure. A force is applied to the tray of the scale causing the spring to displace. A position sensor sends a signal to a control system, and current is then sent to the voice coil to bring the tray back to its original position. At that point the applied force is proportional to the current and can be determined.

A "nanopositioning system based on electromagnetic force compensated balances", discussed by C. Diethold, was used to determine "force to displacement curves" [3]. The system is able to measure force and displacement at the same time with only one sensor and is capable of measuring spring constants as high as $10^9\,\frac{\mathrm{N}}{\mathrm{m}}$, with a resolution of up to $0.01\,\frac{\mathrm{N}}{\mathrm{m}}$.

1.3 Watt Balance

Several types of electromagnetic force balances, known as Kibble (watt) balances, have been designed in the last forty years. A Kibble balance, named after its inventor Dr. Bryan Kibble, uses the balancing of electrical and mechanical power to determine force [4]. A current carrying coil in a magnetic field is used to restore the balance to a null point after a force is applied. Kibble balances have two modes of measuring, one is a force measurement and the other is a velocity measurement. In the force measurement mode, the balance is held at a constant displacement and current is measured to determine the force applied. In velocity mode, the velocity of the moving coil is held constant and the back emf is measured to determine the magnetic flux. Examples of some Kibble balances include the BIPM watt balance[5][6][7], NIST watt balance[8][9][10], and BNM watt balance[11].

The BNM watt balance is designed to measure forces through the use of a voice coil, feedback position, and feedback velocity [11]. An interferometer is used to measure the position and velocity of the coil. The BNM watt balance allows for a magnetic circuit with the same poles of the magnets facing each other. This causes a repelling force and causes the magnetic field lines to travel outward at 90 degrees. These magnetic field lines cross the voice coil perpendicular to the flow of current. Due to the Lorentz Force principal, this then gives a restoring force and motion to the system. In this system, the magnets are stationary and the coil moves. In force measuring mode, the current is measured to hold the system at a constant displacement to determine the applied force. When the system is moving, measurements can be taken to determine the magnetic flux.

CHAPTER 2: OVERALL DESIGN

A force measurement system is desired that can be additively manufactured to lower costs and to provide ease of manufacturing compared to commercially available options, while maintaining equal to or better precision than commercially available options. The force measurement system is also desired to be compact in size, and serve as a proof of concept for further projects. The overall design is shown in Figure 2.1. A voice coil with a stationary coil positioned in the middle of the system drives a pair of moving magnets. Four flexures connect the magnet housings to an outer frame, constraining motion to a single axis. The system is raised on supports to allow room for an external force to be applied from beneath. An opto-interrupter and razor blade optical knife edge sensor determine a position at the top. This optical knife edge (OKE) sensor serves as the position measurement instrument for the system.

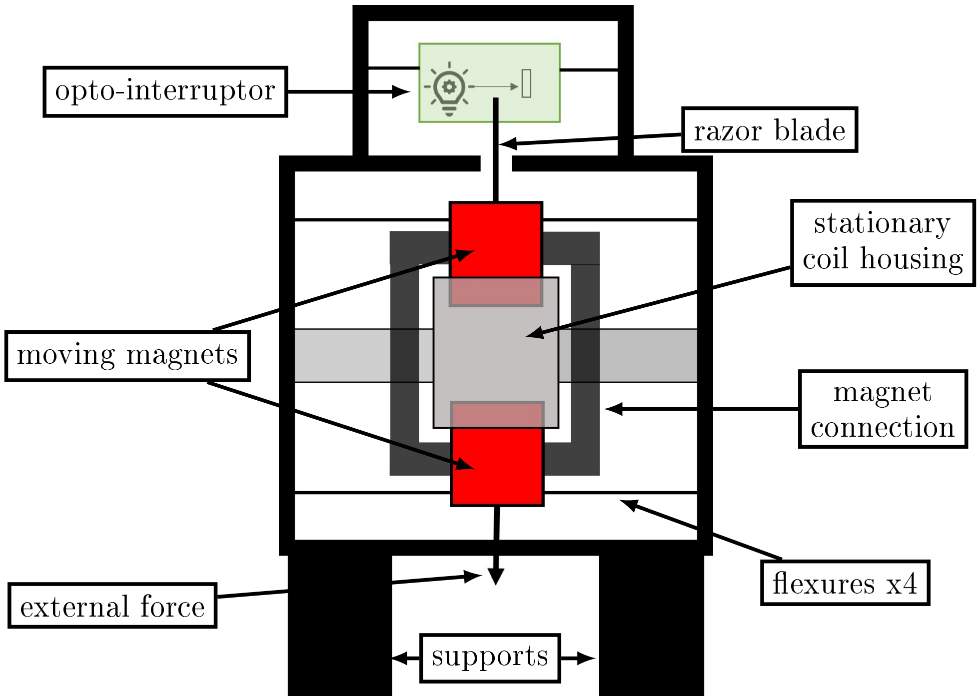


Figure 2.1: Basic Design Concept

The current required to hold a force at a known position through the use of a feedback control system provides the measure of the force. Measuring multiple known forces at a known fixed position for calibration, the system can then be used to measure unknown forces.

The armature, voice coil housing, and OKE sensor housing are additively manufactured with ABS. ABS was chosen based on analysis of the same flexure configuration and dimensions with different materials. ABS has the lowest Young's modulus, and therefore provides the lowest spring stiffness in the armature flexures. This analysis is shown in Appendix B. The final design can be seen in Figure 2.2. Figure 2.3, shows the terminology used for various parts of the system, and their location. The coil is stationary within the voice coil housing, and the magnets move within the coil on both of the outer edges. The magnets are held within the magnet housing and attached to the armature flexures to allow single-axis movement. The magnets are held within their respective housings by the repelling force between the two magnets.

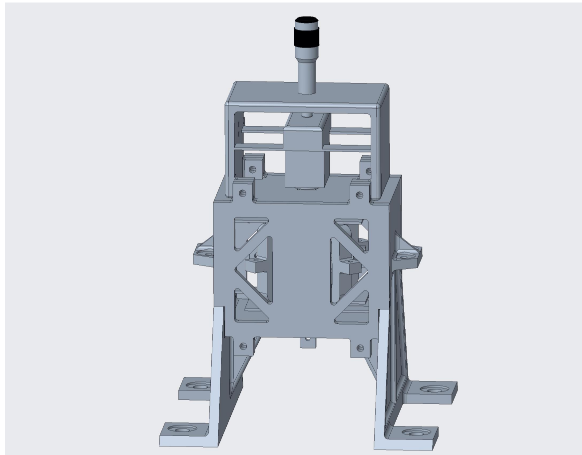


Figure 2.2: Overall Design

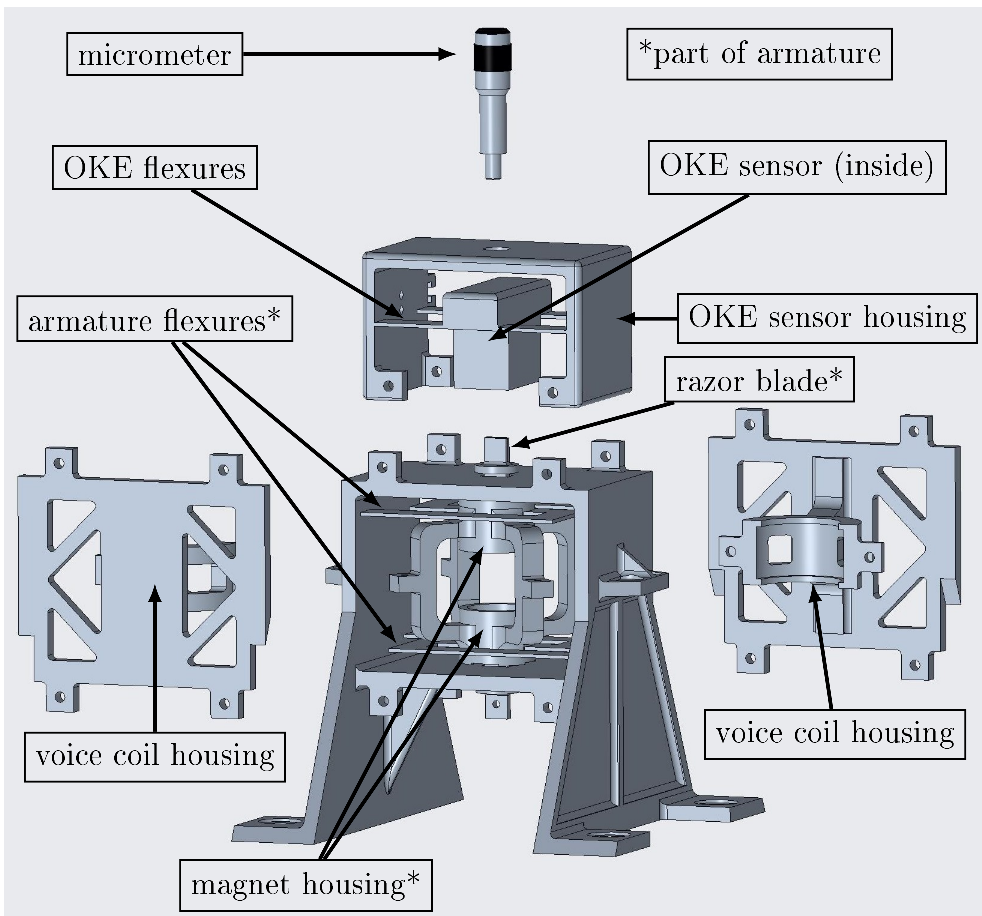


Figure 2.3: Voice Coil and OKE Sensor Housing Exploded View

CHAPTER 3: MECHANICAL DESIGN

3.1 Voice Coil and Magnetic Field Design

The voice coil driver is a stationary-coil moving-magnet system. The design of the magnet structure is influenced by that of the BNM and NIST watt balances. To achieve the desired maximum current capacity, the coil is wound with 20 ga. copper magnet wire. The coil length is 0.8 in, with an inner diameter of 1.04 in, and an outer diameter of 1.6 in. The voice coil was designed with a magnetic field direction aligned with the field of the moving magnets to produce an upward and downward force depending on the direction of the current. The design can be seen in Figure 3.1, where the two opposing magnets generate the magnetic field normal to the axis of motion.

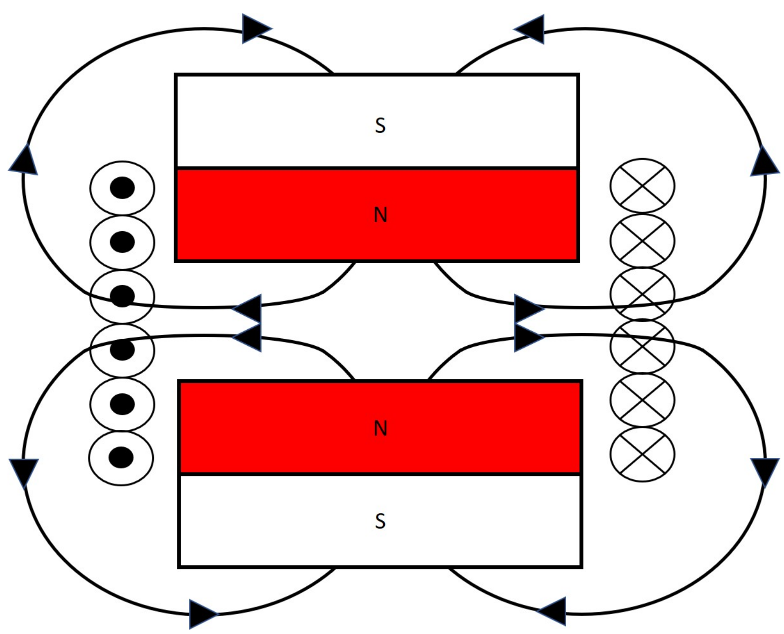


Figure 3.1: Magnetic Circuit Design with Magnetic Field Lines Normal to Coil

A free computer program called Finite Element Magnetics Modelling (FEMM) was used to calculate the magnetic field at points of interest, specifically at the position of the coil. Various magnet sizes and separations were calculated to determine the configuration of the magnetic system with the highest magnetic field at the coil, while maintaining the ability to assemble the system. The final design choice was two neodymium N-42 cylindrical magnets, each with a length of 1 in, and a diameter of 1 in, separated by 0.5 in. Figure 3.2 shows the FEMM model of the magnetic design.

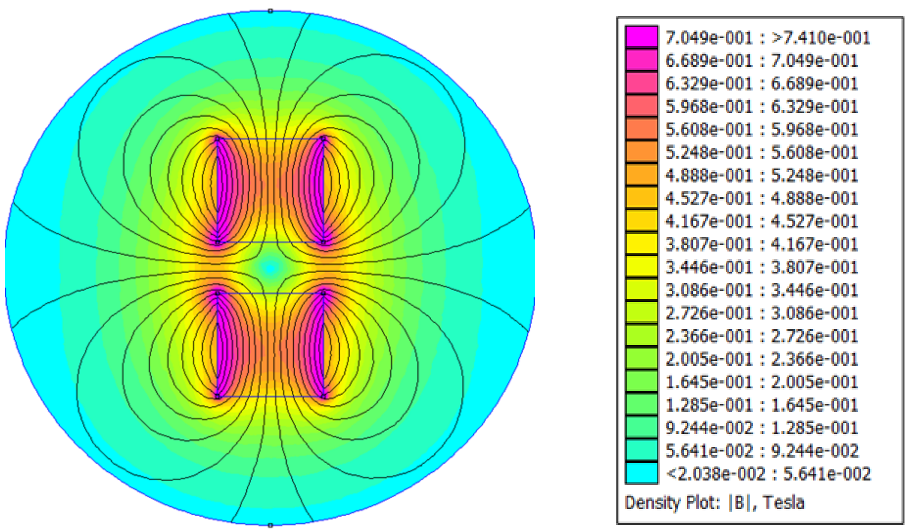


Figure 3.2: FEMM Model

3.2 Flexures

Flexures are used for both the armature movement and for the positioning of the OKE sensor with respect to the razor blade.

3.2.1 Flexure for Armature Movement

A modified blade flexure design supports the armature. Figure 3.3 shows the design of the armature flexure in a sectional view with the top of the system removed for viewing of the flexures. The armature flexures are in a J-shape and are symmetrically opposed side to side and top to bottom to eliminate torsion that would otherwise cause misalignment with the coil during movement. With the flexures opposing each other, a single linear degree of freedom was achieved.

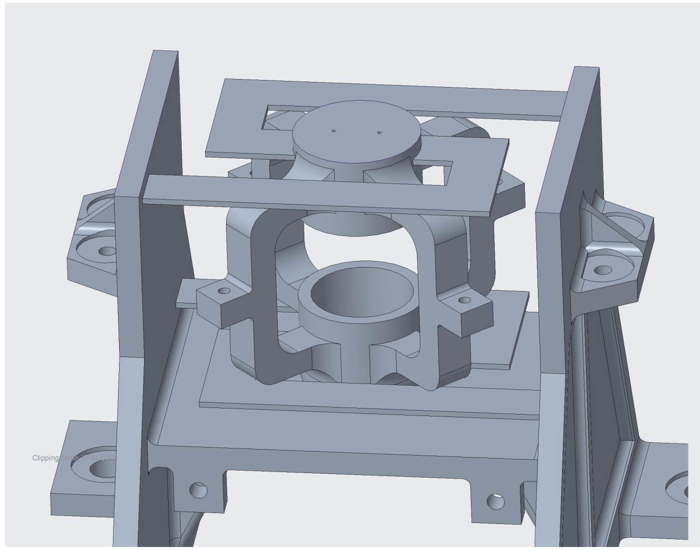


Figure 3.3: Opposing Modified Blade Flexure Design Allowing Single-Axis Motion

Calculations using simple beam theory (Appendix C) determined the stiffness of the armature flexure to be $0.552\,\frac{\text{N}}{\text{mm}}$. A finite element analysis approximately agreed with beam theory calculations and is shown in Figure 3.4, with the colors and table representing displacement in millimeters. A 5 N load was applied to the armature. Flexure stiffness through FEA was determined to be $0.449\,\frac{\text{N}}{\text{mm}}$.

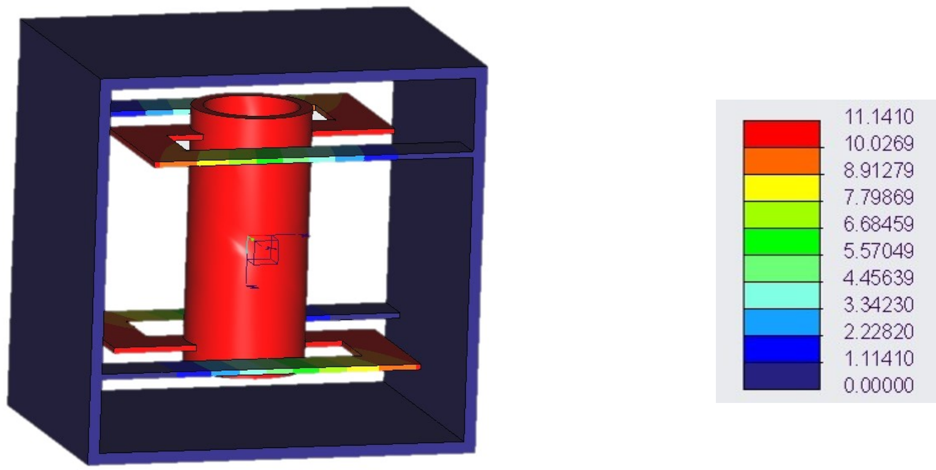


Figure 3.4: FEA analysis of flexure design with 5 N load applied to armature; units in mm

3.2.2 Flexure for OKE Sensor Positioning

A flexure system provides adjustment of the position sensor with respect to the razor blade. The OKE flexure has a range of 2 mm. This range is greater than the range of the opto-interrupter so that the sensor can be positioned at the edge of the razor blade and fine tuned as needed. This position is controlled with a micrometer fixed to the top of the OKE sensor housing. Figure 3.5 shows the micrometer at the top and its interface with the OKE flexure for the position sensor.

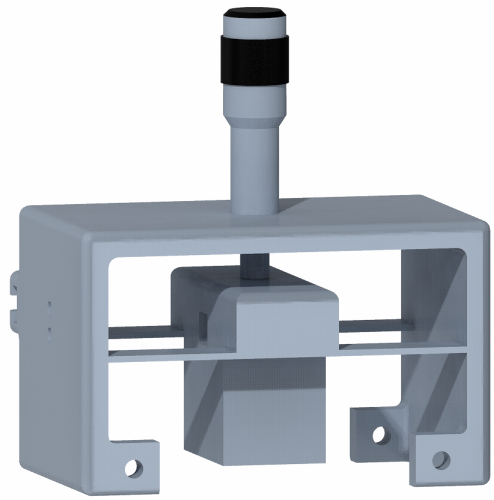


Figure 3.5: Position Sensor Bridge with Flexures and Micrometer

CHAPTER 4: ELECTRICAL AND CONTROLS DESIGN

A voice coil is driven by a current that is determined from holding the armature at a fixed position on the OKE sensor. Figure 4.1, shows the control system that produces the current to drive the voice coil based on positioning from the OKE sensor.

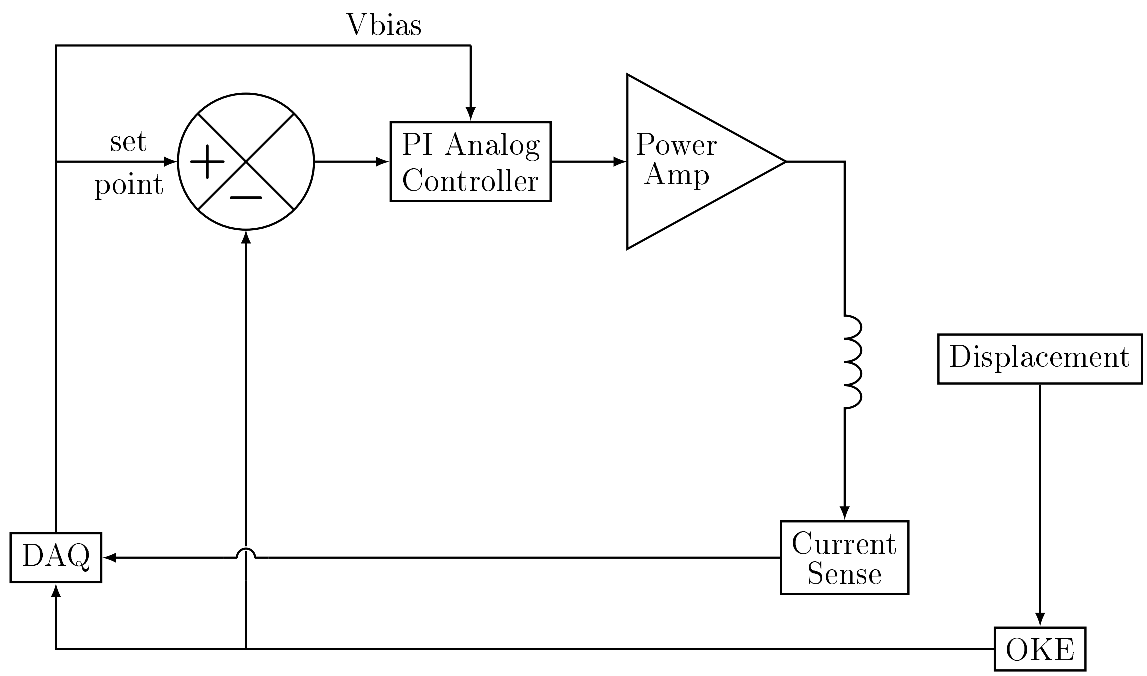


Figure 4.1: Control Diagram

4.1 DAQ

The DAQ is a LabJack T7 Pro that provides the set point based on input from the OKE sensor, and a constant Vbias, while also recording OKE sensor voltage, sense resistor voltage, commanded set point values, error term to the set point, and time. The LabJack has 24 bit resolution capability, which was utilized for recording long term data. For transient data collection the resolution can be adjusted based on the LabJack's resolution index to allow faster data collection. A resolution index of 5 was chosen for transient data collection, which is a 16 bit resolution. This allowed data collection to run at 1 kHz so that more data could be seen and aliasing could be limited. Five analog inputs were used on the labJack and 2 analog outputs. Table 4.1 below shows each analog input and output used and what they were measuring or sending.

Table 4.1: LabJack Inputs and Outputs

Value	**Definition**
AIN0	position sensor (volts)
AIN1	coil temp. sensor (volts)
AIN2	sense resistor (volts=amps)
AIN3	ambient temp. sensor (volts)
AIN4	error term (volts)
DAC0	V bias output
DAC1	V setpoint output

4.2 OKE Sensor Circuit

The optical knife edge sensor, OKE sensor, determines positioning of the armature. As the armature moves upward, the razor blade covers more of the photosensor, blocking the beam of light from hitting the photosensor. This changes the voltage output of the sensor which represents a position. This voltage signal is used as a feedback signal in the control system, so that current can be increased or decreased to the voice coil depending on if the armature is above or below the intended voltage setpoint.

The position signal from the OKE sensor in Figure 4.1, is derived from a trans-impedance amplifier coupled to an opto-interrupter as shown in Figure 4.2. The LM7805 is a voltage regulator to provide better voltage control for components throughout the circuit. The LTC-1046 is a voltage inverter to provide a −5 V to the transimpedance amplifier.

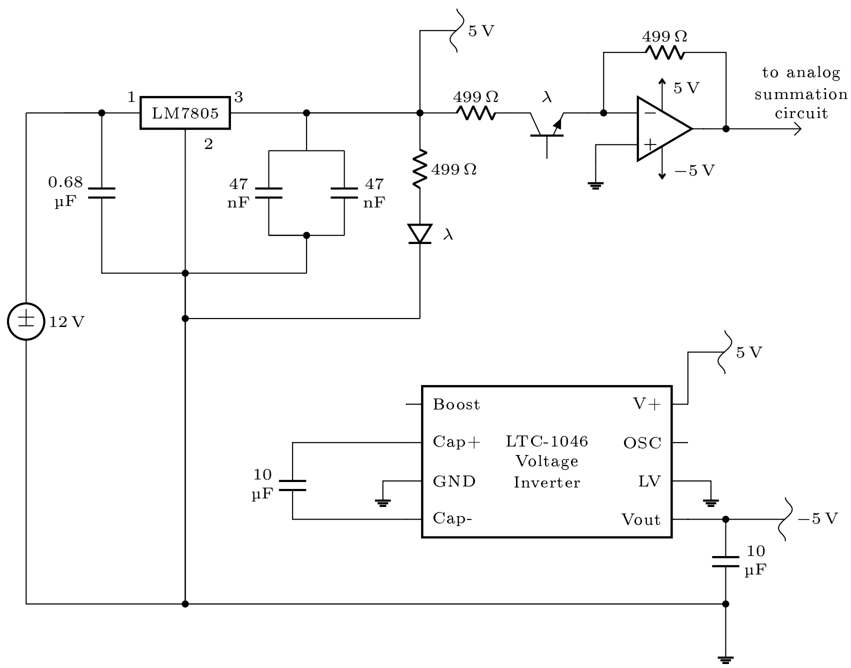


Figure 4.2: OKE circuit

4.3 Power Amplifier Circuit

The circuit for the power amplifier from Figure 4.1 is shown in Figure 4.3. An emitter-follower was created from two transistors to drive the voice coil. The 5 V power supply is a variable power supply set to 5 V, with a current limit manually set to 2.5 A. Voltage was measured across the 1 Ω sense resistor to determine the current required to drive the voice coil.

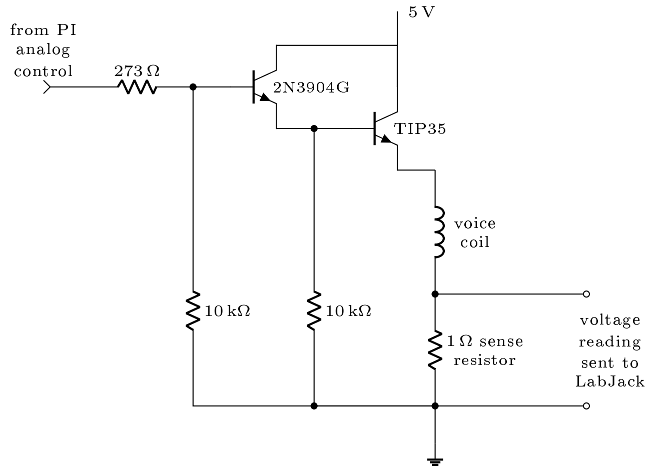


Figure 4.3: Power Amplifier Circuit

4.4 PI Analog Controller Circuit

The PI analog controller circuit, shown in Figure 4.4, is comprised of five LM741 operational amplifiers. A bias voltage signal from the LabJack is sent through a unity gain inverting op amp and then to the proportional op amp. The commanded set point voltage from the summing junction is sent to both a proportional gain stage and an integral gain stage. The signals from both the proportional and integral gain stages are then added together in the summing amplifier. To obtain a positive voltage value for the power amp, the signal is then fed through a unity gain inverting amplifier.

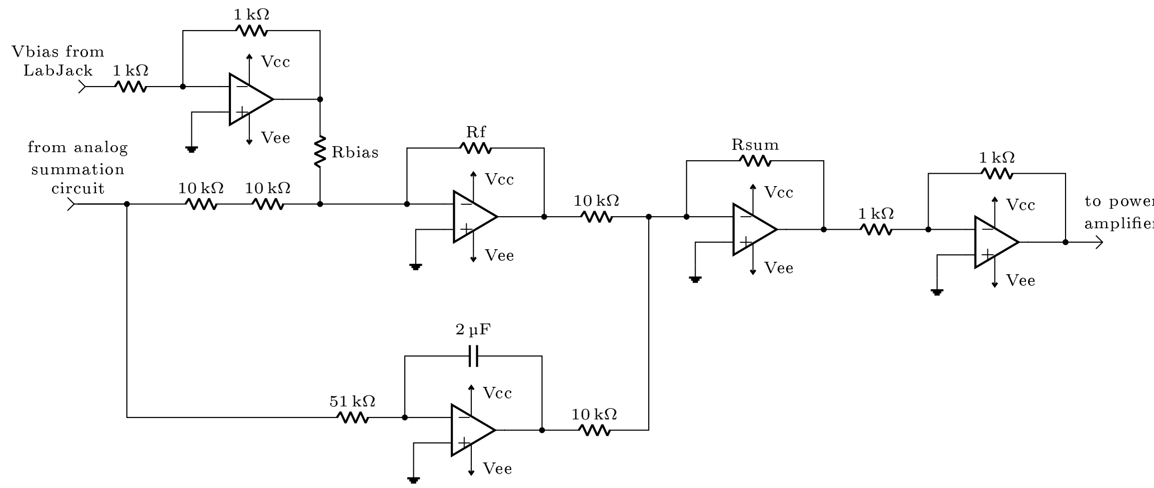


Figure 4.4: PI Analog Control Circuit

4.5 Analog Summation Circuit

The analog summation junction circuit, shown in Figure 4.5, takes the inverted set point voltage signal from the LabJack and subtracts it from the output voltage signal of the OKE sensor through a differencing amplifier. This provides a voltage error signal that is sent to the PI analog control circuit and the DAQ.

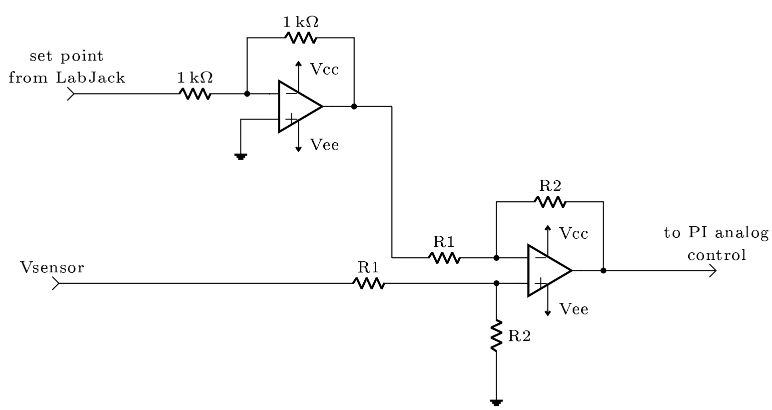


Figure 4.5: Analog Summation Circuit

4.6 Digital and Analog Control

A combination of both digital and analog control is used to eliminate latency issues with using the LabJack T7 Pro DAQ. Measurements were taken of the coil current and residual position for a succession of loadings, and were conducted with the analog PI control system. The measurements began with a 272 g mass, and an additional 10 g loading was added. The 10 g was removed and 20 g was added to the 272 g mass. This process was repeated by adding 50 g, 100 g, and 500 g to the 272 g mass. The steady state error observed as mass was added to the system indicated that the analog control system was not able to completely eliminate the residual error with increasing mass. This was due to imperfect integration in the analog PI controller.

To reduce this residual error, a digital integrator was implemented. The measurements were repeated with the modifications to the control system. Figure 4.6 shows the error as the mass is added to the system after implementation of the digital integrator. The residual error has been reduced to the noise limit of the position sensor by the introduction of the digital integrator.

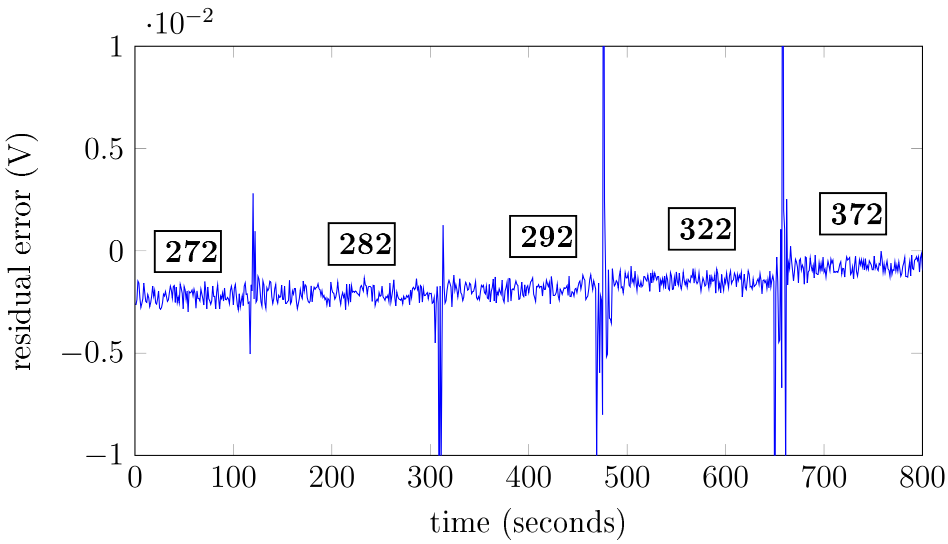


Figure 4.6: Residual error for a succession of five loading masses: 272 g, 282 g, 292 g, 322 g and 372 g with digital integrator implementation

CHAPTER 5: MANUFACTURING

The voice coil housing, armature, and OKE sensor housing were all manufactured using additive manufacturing, and the coil was wound on a lathe.

5.1 Coil

To manufacture the coil to necessary tolerances, a mandrel with a diameter equal to the desired 1.04 in inner diameter dimensions of the coil was made. The mandrel was made with walls spaced by the desired length, so that the wire could be wound to the length of the coil without repeatedly measuring length during the winding process. Speed Stick™ gel deodorant was applied directly to the mandrel as a mold release to ensure removal of the coil after winding. The coil was wound 300 times with 20 ga magnet wire. The wire was wrapped around the outer section of the mandrel multiple times to ensure it would not slip when the winding began. After each layer was wrapped, Devcon super glue gel was applied to hold the coil together. Once completed, the coil was allowed to set for thirty minutes and then it was removed from the mandrel. The finished coil mounted inside the ABS coil housing is shown in Figure 5.1.

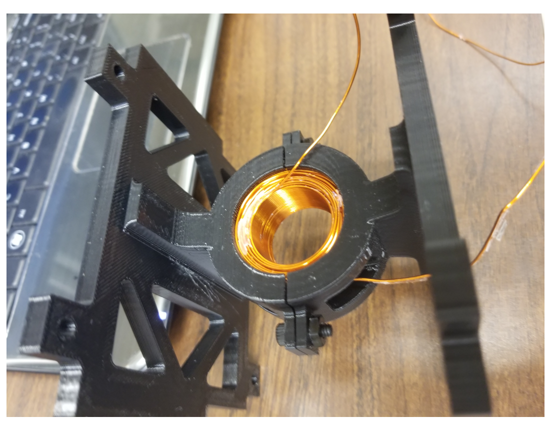


Figure 5.1: Manufactured Coil Inside ABS Coil Housing

5.2 Additive Manufacturing

Additive manufacturing is used to streamline any necessary changes, to reduce cost, and to allow ease of manufacturing for proof of concept purposes. The material of choice is black ABS plastic. The choice of black eliminates stray light entering through the ABS and affecting the position sensor reading. ABS is chosen out of ABS, Nylon, and polycarbonate as having the lowest Young's modulus, and therefore providing the lowest spring stiffness for the armature flexure. The system was printed using a Fortus 400 MC fused deposition printer.

5.3 Assembly

Brass heat-set threaded inserts were placed inside parts of the housing to allow for ease of assembly when putting the sections together. The coil housing is assembled first, placing the coil within each half of the coil housing and bolting the housing together. Care must be taken during assembly of the armature, as the magnets can have a repulsive force of over 50 lbs. Stops, shown in Figure 5.2, were integrated into the design on either side of the armature flexures to prevent the flexures from being over extended during assembly.

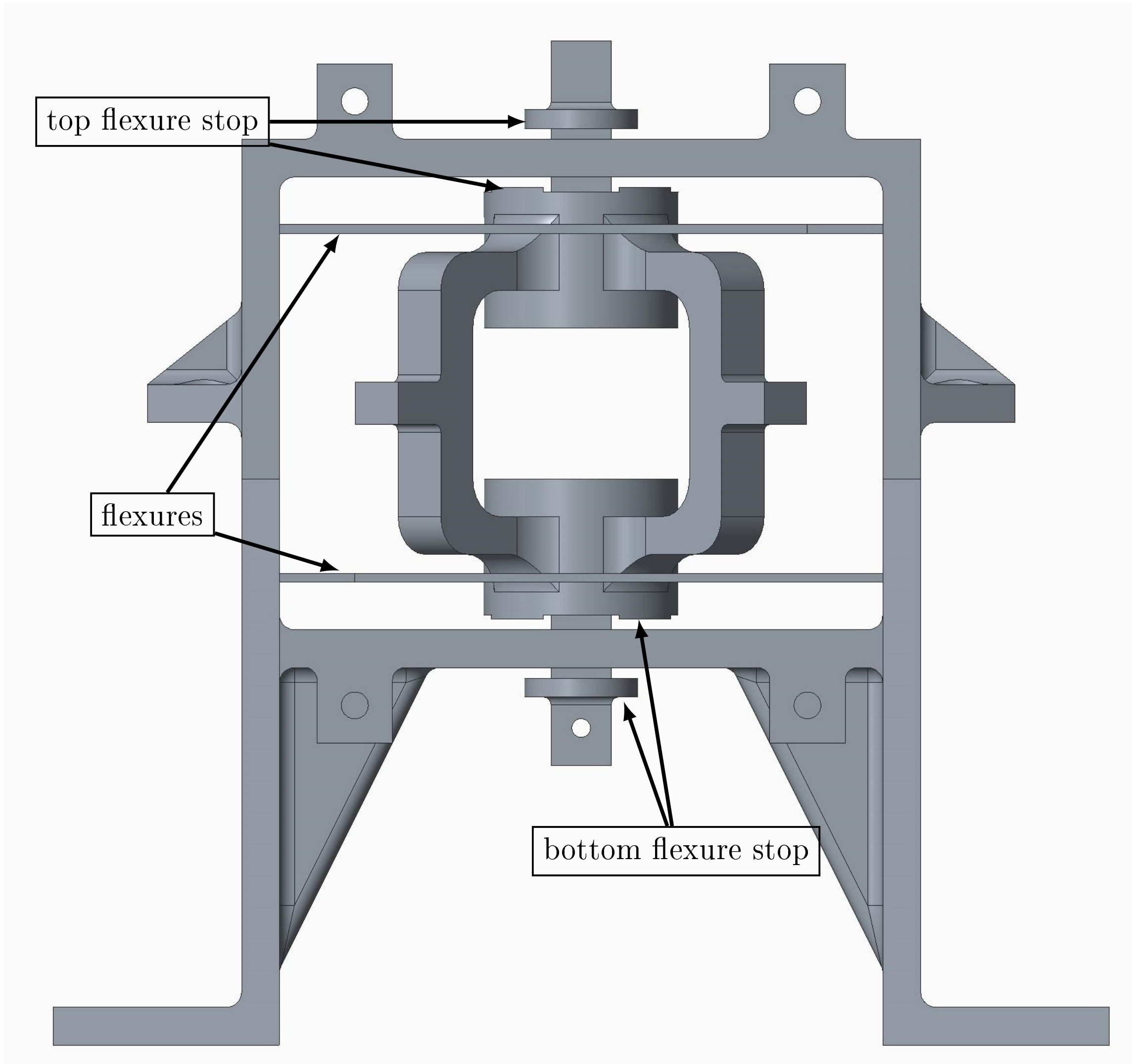


Figure 5.2: Stops to prevent over extending flexures during assembly

The top and bottom parts of the armature are clamped together around the coil housing while the two halves are bolted together. The OKE sensor housing is assembled separately and then placed on top of the system. An exploded view of the system is shown in Figure 5.3.

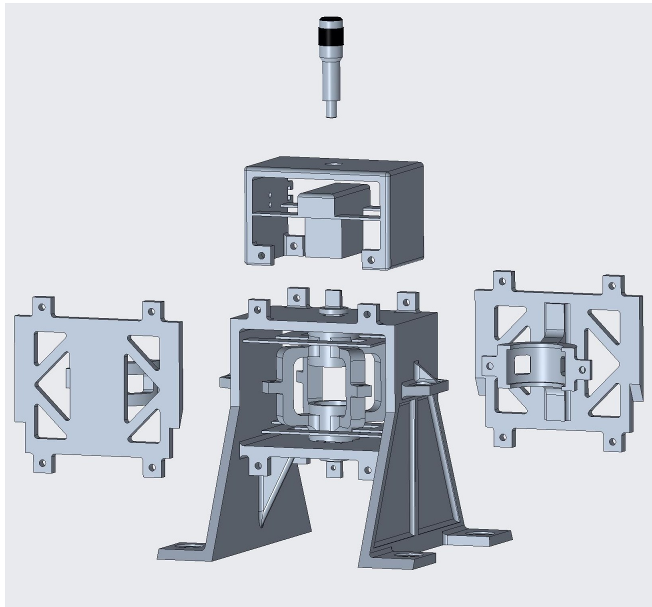


Figure 5.3: Exploded View of System Components for Assembly

CHAPTER 6: RESULTS

The balance of forces applied to the armature at equilibrium is shown in Equation 6.1, where k is the spring constant of the voice coil flexures, x is the position of the flexures with respect to the null point of the flexures, C is the force constant of the voice coil, i is the coil current, f_a is the force produced by the mass of the armature and spring stiffness of the flexures, and f_{ext} is the external force applied to the armature.

$$\sum F = k(x - x_0) - Ci + f_a + f_{ext} = 0 \tag{6.1}$$

Equation 6.1 can be solved for f_{ext} yielding,

$$f_{ext} = Ci - kx - f_a \tag{6.2}$$

C is determined by applying known external forces and determining the current required to maintain the armature at a constant position, x. Given a known value of C, k is determined by fitting position versus applied current at one or more constant values of f_{ext}.

6.1 Calibration Methods

The OKE position sensor used is described in Chapter 4 Section 3. In order to translate the voltage signal received from the sensor into a position value, the sensor must be calibrated. This position value is then used to calculate f_{ext} in Equation 6.2.

To calibrate the position sensor, the coil was not powered and the sensor was positioned over the razor blade by adjusting the micrometer on top. The micrometer reading and the corresponding voltage from the position sensor was recorded at each step across the full range of the position sensor. The sensitivity of the sensor was determined by a linear fit of the position sensor voltage to position. Based on the calibration data and linear fit, as seen in Figure 6.1, the sensitivity of the position sensor was determined to be $1.186(23)\,\frac{\mathrm{V}}{\mathrm{mm}}$.

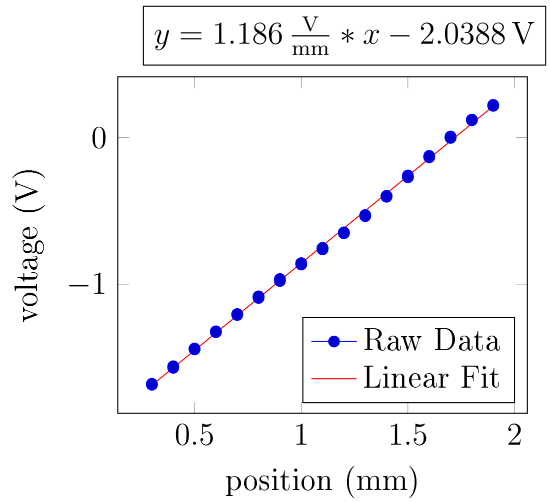


Figure 6.1: Voltage to Position of OKE Position Sensor moved with micrometer with Linear Fit line

Figure 6.2 shows that positions between 0.4 mm and 1.0 mm have the least non-linearity in the position sensor output data. Accordingly, 0.715 mm was chosen as the x_0 position for all measurements.

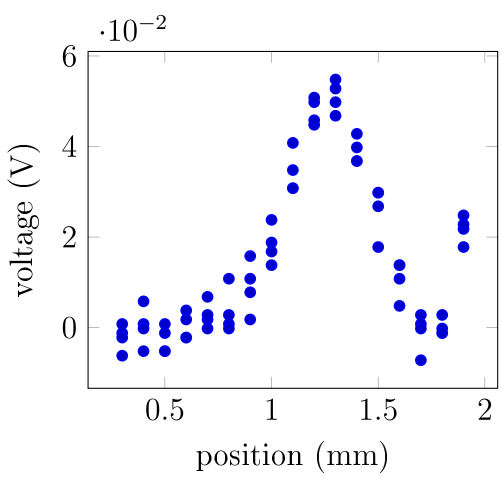


Figure 6.2: Nonlinearity of OKE Position Sensor

The OKE sensor housing design left little room for ambient light to reach the detector. Voltage levels at a known position were tested with both the lights on in the lab, and the lights off. Changes in voltage could not be seen outside of the noise limit of the sensor.

The force produced by the armature mass and spring stiffness of the voice coil flexures, f_a, determines the coil current required to suspend the armature without any external force. Two different methods were used to determine f_a.

The first method was to use a force gauge attached to a linear slide and placed under the flexure stage. An aluminum rod was attached to the end of the force gauge to reduce magnetic effects from the coil on the force gauge. The force gauge was moved upward until the position sensor was reading at x_0. The force gauge reading at this time was 251.2(10) g (2.4643(98) N). Figure 6.3 depicts the set-up used for this measurement method.

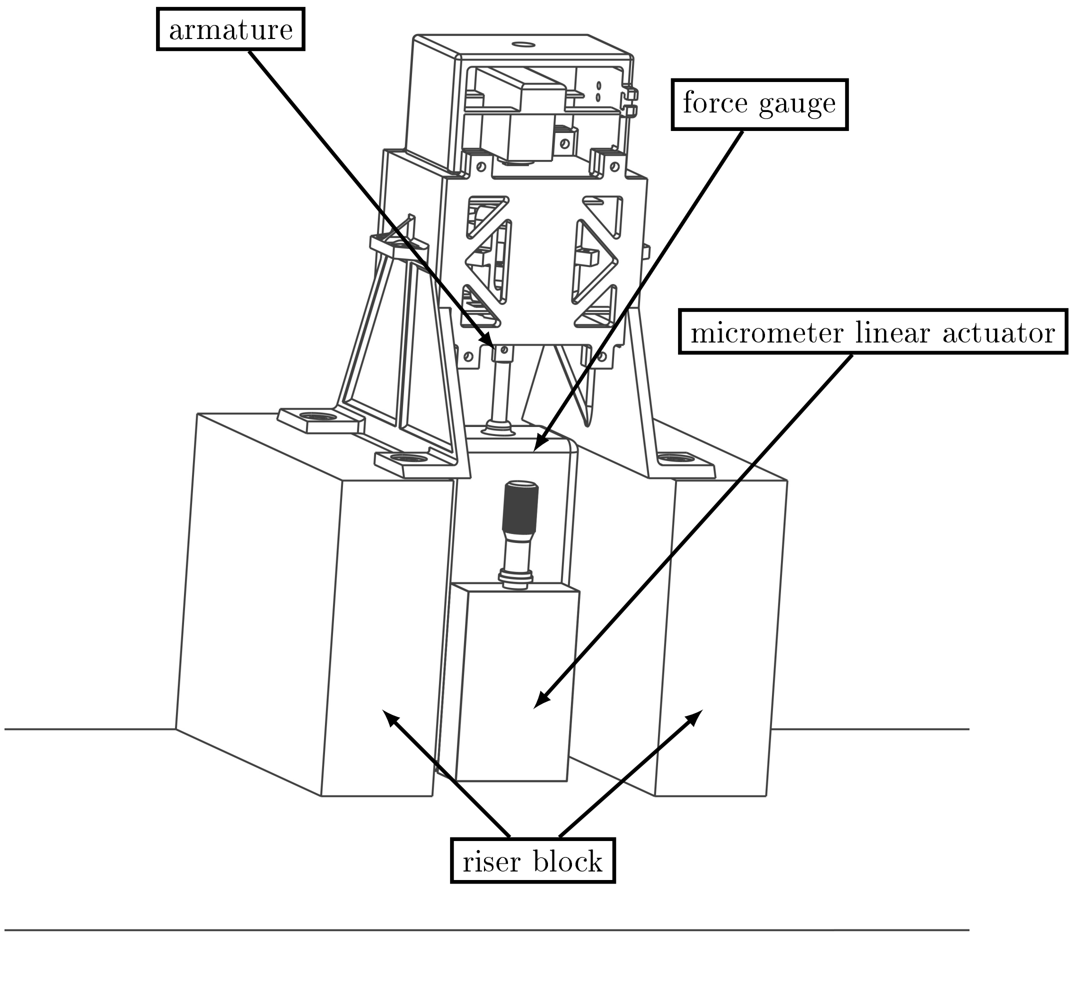


Figure 6.3: Voice coil system set-up

The second method was to use the linear fit of the current vs position data to calculate the force at x_0. Through this method, f_a was determined to be 2.2524(13) N. The first method had more room for errors, to include off axis forces due to poor alignment with the force gauge. Interference in the reading from the magnets used in the voice coil could also not be ruled out. The measured f_a from the second method was used in all calculations.

6.2 Testing

6.2.1 Force Constant

The force constant, C, used in Equation 6.2, was determined with the current of six successive loading masses at the same position. Figure 6.4 shows the position and current with respect to time, as mass is added to the system.

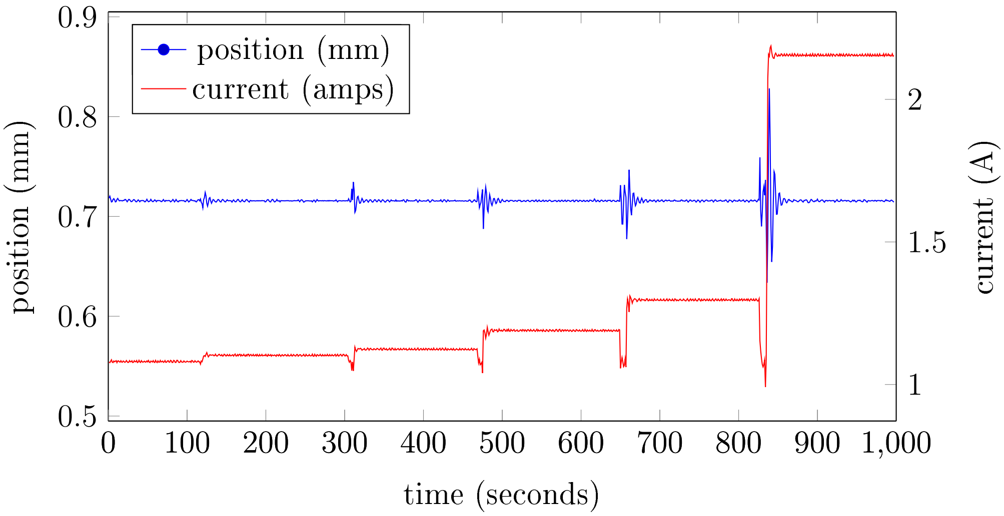


Figure 6.4: Coil current and position for a succession of six loading masses: 272 g, 282 g, 292 g, 322 g, 372 g and 772 g

Using the same data shown in Figure 6.4, Figure 6.5 shows the force versus current. The force constant, C, was determined to be $4.6015(12)\,\frac{\mathrm{N}}{\mathrm{A}}$.

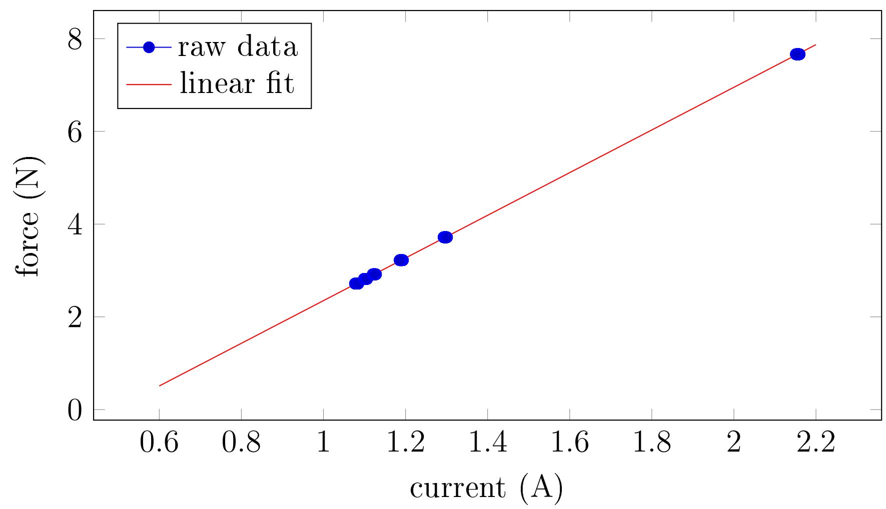


Figure 6.5: Force vs Current with best fit line for determining force constant C

6.2.2 Flexure Stiffness

To determine the voice coil flexures stiffness, k, in Equation 6.2, force versus position measurements of 5 known forces at 12 points were conducted, as shown in Figure 6.6. A linear fit of the force versus position data gives a flexure stiffness of $0.4532(19)\,\frac{\text{N}}{\text{mm}}$ with an R^2 value of 0.999. This value is in approximate agreement with an analytical approximation of $0.552\,\frac{\text{N}}{\text{mm}}$ discussed in Chapter 3 Section 2.1.

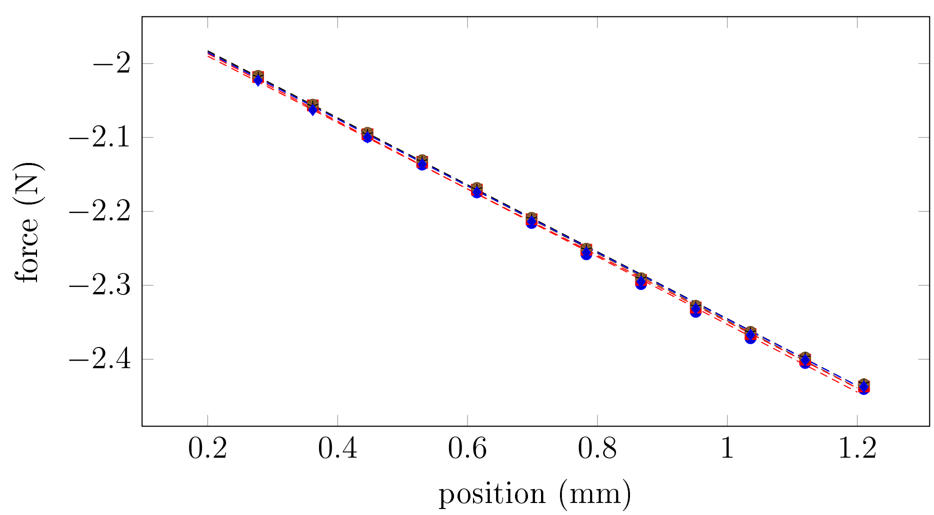


Figure 6.6: Measured Flexure Stiffness with 5 Known Forces at 12 positions with linear fits to each set of measurements

With the force constant, C, flexure stiffness, k, and combined force produced by the armature mass and flexure, f_a, determined, the measurements of the current and position for a known external force were taken. Equation 6.2 was used to calculate the known external force for comparison against measurements taken with a separate scale. With this verified, an unknown external force was applied to the system. The corresponding position and current was measured and the unknown external force was calculated using Equation 6.2.

To determine the uncertainty of the measured external force, Equation 6.3 was used. The uncertainty of the measured external force was determined to be $\pm 0.003\,\mathrm{N}$. Equation 6.2 was used to derive Equation 6.3. The sensitivities of each variable are multiplied by their respective uncertainty, and then summed in quadrature with all other variables to determine the overall uncertainty.

$$(\delta f_{ext})^2 = (i \cdot \delta C)^2 + (C \cdot \delta i)^2 + (x \cdot \delta k)^2 + (k \cdot \delta x)^2 + (1 \cdot \delta f_a)^2 \tag{6.3}$$

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

The voice coil actuated force measurement gauge is capable of determining an unknown force of up to 10 N, with an uncertainty of ± 0.003 N. Ease of manufacturing and design change is obtained through the use of additive manufacturing. Flexure design provides compliant springs for armature and OKE sensor movement. The use of an analog control system helped reduce latency issues with the LabJack DAQ, and the digital integrator implementation was critical in reducing residual steady state errors.

As a proof of concept the system demonstrates potential. It could be improved upon with several additions and modifications. The addition of a thermistor on the coil would help determine the temperature of the coil during measurements, and analysis of those temperature effects on the unknown force determination could be conducted. The magnetic circuit could potentially be improved with the addition of a yoke. This could help concentrate magnetic fields at points of interest to increase the force produced by the voice coil. Connection points for ease of assembly of the top and bottom portions of the voice coil flexure stage should be considered. It is also recommended to automate the process of switching between suspended masses during calibration in order to avoid thermal and mechanical disturbances due to handling.

REFERENCES

[1] D. Stuart-Watson and J. Tapson, "Simple force balance accelerometer/seismometer based on a tuning fork displacement sensor," *Review of Scientific Instruments*, vol. 75, no. 9, 2004.

[2] F. Bator, K. D. Hunter, and R. T. Durst Jr, "Weighing scale with voice coil," Feb. 7 1989. US Patent 4,802,541.

[3] C. Diethold, M. KÃŒhnel, F. Hilbrunner, T. FrÃ¶hlich, and E. Manske, "Determination of force to displacement curves using a nanopositioning system based on electromagnetic force compensated balances," *Measurement*, vol. 51, pp. 343–348, May 2014.

[4] S. Schlamminger and D. Haddad, "The Kibble balance and the kilogram," *Comptes Rendus Physique*, vol. 20, pp. 55–63, Jan. 2019.

[5] H. Fang, A. Kiss, E. de Mirandes, J. Lan, L. Robertsson, S. Solve, A. Picard, and M. Stock, "Status of the BIPM Watt Balance," *IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT*, vol. 62, no. 6, pp. 1491–1498, 2013.

[6] A. Picard, M. Stock, F. Hao, T. J. Witt, and D. Reymann, "The BIPM Watt Balance," *IEEE Transactions on Instrumentation & Measurement*, vol. 56, no. 2, 2007.

[7] A. Picard, M. P. Bradley, H. Fang, A. Kiss, E. De Mirandes, S. Solve, M. Stock, and B. Parker, "The BIPM watt balance: Improvements and developments," *IEEE Trans. Instrum. Meas. IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 7, pp. 2378–2386, 2011.

[8] A. D. Gillespie and K.-i. Fujii, "Alignment uncertainties of the NIST watt experiment," *IEEE Transactions on Instrumentation & Measurement*, vol. 46, no. 2, 1997.

[9] J. P. Schwarz, R. Liu, D. B. Newell, R. L. Steiner, E. R. Williams, D. Smith, A. Erdemir, and J. Woodford, "Hysteresis and Related Error Mechanisms in the NIST Watt Balance Experiment," *Journal of research of the National Institute of Standards and Technology*, vol. 106, no. 4, 2001.

[10] G. N. Stenbakken, R. Steiner, P. T. Olsen, and E. Williams, "Methods for Aligning the NIST Watt-Balance," *IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT IM*, vol. 45, no. 2, pp. 372–377, 1996.

[11] P. Gournay, G. GenevÃšs, F. Alves, M. Besbes, F. Villar, and J. David, "Magnetic Circuit Design for the BNM Watt Balance Experiment," *IEEE Transactions on Instrumentation & Measurement*, vol. 54, no. 2, 2005.

APPENDIX A: C++ Code

Listing A.1: C++ code main4rev3avgSLOW

```
/**
* Name: eWriteAddress.c
* Desc: Shows how to use the LJM_eWriteAddress function
**/

// For printf
#include <stdio.h>

// For the LabJackM Library
#include <LabJackM.h>

// input output stream that allows cin and cout to read
//continuously
#include <iostream>

// For LabJackM helper functions, such as OpenOrDie,
//PrintDeviceInfoFromHandle, ErrorCheck, etc.
#include "LJM_Utilities.h"

//to save to csv file
#include <fstream>
#include <string>

//for measuring time in hours minutes seconds
```

```
#include <ctime>
#include <chrono>

using namespace std;

//for measuring time in nanoseconds
//using  ns = chrono::nanoseconds;
//using get_time = chrono::steady_clock ;

int main()
{

// create an ofstream for the file output (see the link on
// streams for more info)
ofstream outputFile;

// create a name for the file output
//string filename = "exampleOutput.csv";

// create and open the .csv file
//outputFile.open("exampleOutput.csv"); //this one writes
//over itself everytime you run it
outputFile.open("exampleOutput.csv",fstream::app); //this one
//appends new data to end of old data everytime you run it

//write the file headers
outputFile<<"time & date"<<","<<"position sensor(volts)avg"
```

```
<< ","<< "sense resistor (amps) avg" << "," << "vbias" << ","
<< "v setpoint" << "," << "ambient temp volt avg" << ","
<< "coil temp volt avg" << "," << "error term volts" << endl;

int n=1;
//locks into a loop continuosly
while (true){

n=n+1;
int err, handle, i;

//to change the resolution
enum { NUM_FRAMES_CONFIG = 1 };
const char * aNamesConfig[NUM_FRAMES_CONFIG] = \
{"AIN_ALL_RESOLUTION_INDEX"};
const double aValuesConfig[NUM_FRAMES_CONFIG] = {10};
int errorAddress = INITIAL_ERR_ADDRESS;

// Set up for reading AIN value
double value; //measurement that the LabJack is reading
//from the OKE sensor
const char * NAME = "AIN0"; //voltage value

// Open first found LabJack
handle = OpenOrDie(LJM_dtANY, LJM_ctANY, "LJM_idANY");

// Read AIN from the LabJack
```

```
err = LJM_eReadName(handle, NAME, &value);
ErrorCheck(err, "LJM_eReadName");

double value2;
const char * NAME2 = "AIN2";
err = LJM_eReadName(handle, NAME2, &value2);
ErrorCheck(err, "LJM_eReadName");

//value4
const int ADDRESS = 1002; //DAC1
const int TYPE = LJM_FLOAT32;
double value4 = 1.18; //vsetpoint

err = LJM_Open(LJM_dtANY, LJM_ctANY, "LJM_idANY", &handle);
ErrorCheck(err, "LJM_Open");

err = LJM_eWriteAddress(handle, ADDRESS, TYPE, value4);
ErrorCheck(err, "LJM_eWriteAddress");

// write a voltage to DAC0
const int ADDRESS1 = 1000; // DAC0
const int TYPE1 = LJM_FLOAT32;
double value3 = 1.0; //vbias

// Open first found LabJack
err = LJM_Open(LJM_dtANY, LJM_ctANY, "LJM_idANY", &handle);
ErrorCheck(err, "LJM_Open");
```

```
err = LJM_eWriteAddress(handle, ADDRESS1, TYPE1, value3);
ErrorCheck(err, "LJM_eWriteAddress");

//to measure the time in hours minutes seconds
time_t now = time(0);
char* dt = ctime(&now);

double value5; //ambient temp sensor
const char * NAME5 = "AIN3";
err = LJM_eReadName(handle, NAME5, &value5);
ErrorCheck(err, "LJM_eReadName");

double value6; //temp sensor at coil
const char * NAME6 = "AIN1";
err = LJM_eReadName(handle, NAME6, &value6);
ErrorCheck(err, "LJM_eReadName");

double value7; //error term
const char * NAME7 = "AIN4";
err = LJM_eReadName(handle, NAME7, &value7);
ErrorCheck(err, "LJM_eReadName");

//Setup & call eWriteNames to configure AIN0 on LabJack.
err = LJM_eWriteNames(handle, NUM_FRAMES_CONFIG,
aNamesConfig,aValuesConfig,&errorAddress);
ErrorCheckWithAddress(err, errorAddress, "LJM_eWriteNames");
```

```

//how many data points it is averaging before writing
//a point to the csv file
int num=100;
double sum1=0, avgval1;
double sum2=0, avgval2;
double sum3=0, avgval5;
double sum4=0, avgval6;
double sum5=0, avgval7;

//cin >> num;

for(int i = 1; i <= num; i++){
err = LJM_eReadName(handle, NAME, &value);
ErrorCheck(err, "LJM_eReadName");
sum1 += value;

err = LJM_eReadName(handle, NAME2, &value2);
ErrorCheck(err, "LJM_eReadName");
sum2 += value2;

err = LJM_eReadName(handle, NAME5, &value5);
ErrorCheck(err, "LJM_eReadName");
sum3 += value5;

err = LJM_eReadName(handle, NAME6, &value6);
ErrorCheck(err, "LJM_eReadName");
```

```
sum4 += value6;

err = LJM_eReadName(handle, NAME7, &value7);
ErrorCheck(err, "LJM_eReadName");
sum5 += value7;
}

avgval1 = sum1 / num;
avgval2 = sum2 / num;
avgval5 = sum3 / num;
avgval6 = sum4 / num;
avgval7 = sum5 / num;
cout<<avgval1<<"        "<<avgval2<<"        "<<avgval5<<"        "
<<avgval6<<"        "<<avgval7<<endl;

outputFile<<dt<<","<<avgval1<<","<<avgval2<<","
<<value3<<","<<value4<<","<<avgval5<<","<<avgval6
<<","<<avgval7<<endl;
}}
```

Listing A.2: C++ code main14rev2AVGslowVsetpt24bitressteps0pt05

```
/**
* Name: eWriteAddress.c
* Desc: Shows how to use the LJM_eWriteAddress function
**/

// For printf
#include <stdio.h>

// For the LabJackM Library
#include <LabJackM.h>

//input output stream that allows cin and cout
//to read continuously
#include <iostream>

// For LabJackM helper functions, such as OpenOrDie,
//PrintDeviceInfoFromHandle,ErrorCheck, etc.
#include "LJM_Utilities.h"

//to save to csv file
#include <fstream>
#include <string>

#include <ctime>
#include <chrono>

```

```
using namespace std;

int main()
{
int reps=12;
int reps2= 30;
double start=0.85;
double start_adj= start -.05;
double delta;
double value3 = start_adj+ delta;
double curval=0;
double curval2=0;
double curval3=0;

// create an ofstream for the file output
ofstream outputFile;

// create and open the .csv file
outputFile.open("exampleOutput.csv");

// write the file headers
outputFile << "time and date" << ","
<<"position sensor(volts)avg" << ","
<< "sense resistor (amps) avg" << ","
<< "Vbias"<<","<<"v setpoint"<<","
<<"ambient temp volt avg"<<","<<"coil temp volt avg"
<<","<<"error term volts"<<endl;
```

```

int itr=1;

for (itr=1 ; itr<=4*reps;itr++){
delta=itr*.05;
int n=1;

n=n+1;

int err, handle, i;

enum { NUM_FRAMES_CONFIG = 1 };
const char * aNamesConfig[NUM_FRAMES_CONFIG] = \
{"AIN_ALL_RESOLUTION_INDEX"};
const double aValuesConfig[NUM_FRAMES_CONFIG] = {10};
int errorAddress = INITIAL_ERR_ADDRESS;

// Set up for reading AIN value
double value ; //measurement that the LabJack is
//reading from the OKE sensor the value is a voltage
//but it represents my position
const char * NAME = "AIN0";

// Open first found LabJack
handle = OpenOrDie(LJM_dtANY, LJM_ctANY, "LJM_idANY");

const int ADDRESS = 1000; //DAC1 i think
```

```
const int TYPE = LJM_FLOAT32;
double value4 = 1.0;  //V BIAS

err = LJM_Open(LJM_dtANY, LJM_ctANY, "LJM_idANY", &handle);
ErrorCheck(err, "LJM_Open");

err = LJM_eWriteAddress(handle, ADDRESS, TYPE, value4);
ErrorCheck(err, "LJM_eWriteAddress");

double value2;
const char * NAME2 = "AIN2";

// write to DAC0
const int ADDRESS1 = 1002; // DAC0 Vsetpoint
const int TYPE1 = LJM_FLOAT32;
if(itr<=reps)
{value3 = start_adj+ delta;
curval=value3;
}
// if(x>70 && x<80)
else if (itr > reps && itr <= reps*2)
{delta=(itr-(reps+1))*.05;
start_adj=curval;
value3 = start_adj- delta;
curval2 =value3;}

else if (itr > reps*2 && itr < reps*3)
```

```
{

delta=(itr -((2*reps)-1))*.05;
start_adj=curval2 -.05;
value3 = start_adj+ delta;
curval3=value3;}

else if (itr > reps*3 && itr < reps*4)
{delta=(itr -((3*reps)+1))*.05;
start_adj=curval3 -.05;
value3 = start_adj- delta;}


// Open first found LabJack
err=LJM_Open(LJM_dtANY,LJM_ctANY,"LJM_idANY",&handle);
ErrorCheck(err, "LJM_Open");

err = LJM_eWriteAddress(handle, ADDRESS1, TYPE1, value3);
ErrorCheck(err, "LJM_eWriteAddress");

for (int itr2=1 ; itr2<=reps2; itr2++){

time_t now = time(0);
char* dt = ctime(&now);

double value5; //ambient temp sensor
const char * NAME5 = "AIN3";
```

```
double value6; //temp sensor at coil
const char * NAME6 = "AIN1";

double value7; //error term
const char * NAME7 = "AIN4";

//Setup & call eWriteNames to configure AIN0 on LabJack.
err = LJM_eWriteNames(handle, NUM_FRAMES_CONFIG,
aNamesConfig, aValuesConfig, &errorAddress);
ErrorCheckWithAddress(err, errorAddress, "LJM_eWriteNames");

// Read AIN from the LabJack
err = LJM_eReadName(handle, NAME, &value);
//value=sensor voltage
ErrorCheck(err, "LJM_eReadName");

err = LJM_eReadName(handle, NAME2, &value2);
//value2=sense resistor voltage
ErrorCheck(err, "LJM_eReadName");

err = LJM_eReadName(handle, NAME5, &value5);
ErrorCheck(err, "LJM_eReadName");

err = LJM_eReadName(handle, NAME6, &value6);
ErrorCheck(err, "LJM_eReadName");

```

```
err = LJM_eReadName(handle, NAME7, &value7);
ErrorCheck(err, "LJM_eReadName");

int num=100;
double sum1=0, avgval1;
double sum2=0, avgval2;
double sum3=0, avgval5;
double sum4=0, avgval6;
double sum5=0, avgval7;

for(int i = 1; i <= num; i++){

//Setup & call eWriteNames to configure AIN0 on LabJack.
err = LJM_eWriteNames(handle, NUM_FRAMES_CONFIG,
aNamesConfig, aValuesConfig,&errorAddress);
ErrorCheckWithAddress(err, errorAddress, "LJM_eWriteNames");

err = LJM_eReadName(handle, NAME, &value);
ErrorCheck(err, "LJM_eReadName");
sum1 += value;

err = LJM_eReadName(handle, NAME2, &value2);
ErrorCheck(err, "LJM_eReadName");
sum2 += value2;

err = LJM_eReadName(handle, NAME5, &value5);
ErrorCheck(err, "LJM_eReadName");
```

```
sum3 += value5;

err = LJM_eReadName(handle, NAME6, &value6);
ErrorCheck(err, "LJM_eReadName");
sum4 += value6;

err = LJM_eReadName(handle, NAME7, &value7);
ErrorCheck(err, "LJM_eReadName");
sum5 += value7;
}

avgval1 = sum1 / num;
avgval2 = sum2 / num;
avgval5 = sum3 / num;
avgval6 = sum4 / num;
avgval7 = sum5 / num;

cout<<avgval1<<"     "<<avgval2<<"       "<<value3
<<"       "<<avgval7<<endl;

outputFile << dt /*<< "," << n */<< "," << avgval1
<< "," << avgval2 << "," << value3 << "," << value4
<< "," << avgval5 << "," << avgval6 << ","
<< avgval7 << endl;
}}
value3 = 0;
cout << "the voltage value is currently: " << value3<< endl;
```

```
}
```

Listing A.3: C++ code main15fastVsetptIncrement0pt2

```
/**
* Name: eWriteAddress.c
* Desc: Shows how to use the LJM_eWriteAddress function
**/

// For printf
#include <stdio.h>

// For the LabJackM Library
#include <LabJackM.h>

//input output stream that allows cin and
//cout to read continuously
#include <iostream>

// For LabJackM helper functions, such as OpenOrDie,
//PrintDeviceInfoFromHandle,
// ErrorCheck, etc.
#include "LJM_Utilities.h"

//to save to csv file
#include <fstream>
#include <string>

#include <ctime>
#include <chrono>
```

```
using namespace std;

using  ns = chrono::nanoseconds;
using get_time = chrono::steady_clock ;

int main()
{
int reps= 1;
int reps2= 10000;
double start=1.2;
double start_adj= start -.2;
double delta;
double value3 = start_adj+ delta;
double curval=0;
double curval2=0;
double curval3=0;

auto start_time = get_time::now();
//use auto keyword to minimize typing strokes

// create an ofstream for the file output
ofstream outputFile;

// create and open the .csv file
outputFile.open("exampleOutput.csv");

```

```
// write the file headers
outputFile << "time and date" << "," <<
"position sensor volts" << ","<< "sense resistor amps"
<< "," << "V setpoint"<<","<<"V bias"<<","
<<"ambient temp voltage"<<","<<"coil temp voltage"<<","
<<"error voltage"<<endl;
int itr=1;

for (itr=1 ; itr<=4*reps;itr++){
delta=itr*.2;
int n=1;


//locks into a loop continuosly

n=n+1;

int err, handle, i;


enum { NUM_FRAMES_CONFIG = 1 };
const char * aNamesConfig[NUM_FRAMES_CONFIG] = \
{"AIN_ALL_RESOLUTION_INDEX"};
const double aValuesConfig[NUM_FRAMES_CONFIG] = {5};
int errorAddress = INITIAL_ERR_ADDRESS;


```

```
// Set up for reading AIN value
double value ;  //measurement that the LabJack is
//reading from the OKE sensor
const char * NAME = "AIN0"; //voltage value

// Open first found LabJack
handle = OpenOrDie(LJM_dtANY, LJM_ctANY, "LJM_idANY");

const int ADDRESS = 1000; //DAC0 i think
const int TYPE = LJM_FLOAT32;
double value4 = 1.0;  //v bias

err=LJM_Open(LJM_dtANY,LJM_ctANY,"LJM_idANY",&handle);
ErrorCheck(err, "LJM_Open");

err=LJM_eWriteAddress(handle,ADDRESS,TYPE,value4);
ErrorCheck(err, "LJM_eWriteAddress");

double value2;
const char * NAME2 = "AIN2";

// write to DAC
const int ADDRESS1 = 1002; // DAC1 V setpoint
const int TYPE1 = LJM_FLOAT32;
if(itr<=reps)
{value3 = start_adj+ delta;
curval=value3;
```

```
}
// if(x>70 && x<80)
else if (itr > reps && itr <= reps*2)
{delta=(itr-(reps+1))*.2;
start_adj=curval;
value3 = start_adj- delta;
curval2 =value3;}

else if (itr > reps*2 && itr < reps*3)
{

delta=(itr-((2*reps)-1))*.2;
start_adj=curval2-.2;
value3 = start_adj+ delta;
curval3=value3;}

else if (itr > reps*3 && itr < reps*4)
{delta=(itr-((3*reps)+1))*.2;
start_adj=curval3-.2;
value3 = start_adj- delta;}

// Open first found LabJack
err = LJM_Open(LJM_dtANY, LJM_ctANY, "LJM_idANY", &handle);
ErrorCheck(err, "LJM_Open");

err = LJM_eWriteAddress(handle, ADDRESS1, TYPE1, value3);
ErrorCheck(err, "LJM_eWriteAddress");
```

```
for (int itr2=1 ; itr2<=reps2; itr2++){
auto end = get_time::now();
auto diff = end - start_time;
double value5; //ambient temp sensor
const char * NAME5 = "AIN3";
double value6; //temp sensor at coil
const char * NAME6 = "AIN1";
double value7; //temp sensor at coil
const char * NAME7 = "AIN4";
err = LJM_eWriteNames(handle, NUM_FRAMES_CONFIG,
aNamesConfig, aValuesConfig, &errorAddress);
ErrorCheckWithAddress(err,errorAddress,"LJM_eWriteNames");
// Read AIN from the LabJack
err = LJM_eReadName(handle, NAME, &value);
//value is the sensor voltage
ErrorCheck(err, "LJM_eReadName");
err = LJM_eReadName(handle, NAME2, &value2);
//value2 is the sense resistor voltage
ErrorCheck(err, "LJM_eReadName");
```

```

err = LJM_eReadName(handle, NAME5, &value5);
ErrorCheck(err, "LJM_eReadName");

err = LJM_eReadName(handle, NAME6, &value6);
ErrorCheck(err, "LJM_eReadName");

err = LJM_eReadName(handle, NAME7, &value7);
ErrorCheck(err, "LJM_eReadName");

outputFile<<chrono::duration_cast<ns>(diff).count() /*
<< "," << n */<< "," << value << "," << value2 << ","
<< value3 << "," << value4 << "," << value5 << "," <<
value6 << "," << value7 << endl;
}}
value3 = 0;
cout << "the voltage value is currently: " << value3<< endl;
}
```

Listing A.4: C++ code main4rev3avgSLOWwithDigIntegRev2

```
/**
* Name: eWriteAddress.c
* Desc: Shows how to use the LJM_eWriteAddress function
**/

// For printf
#include <stdio.h>

// For the LabJackM Library
#include <LabJackM.h>

//input output stream that allows cin and
//cout to read continuously
#include <iostream>

// For LabJackM helper functions, such as OpenOrDie,
//PrintDeviceInfoFromHandle,
// ErrorCheck, etc.
#include "LJM_Utilities.h"

//to save to csv file
#include <fstream>
#include <string>

#include <ctime>
#include <chrono>
```

```
#include <unistd.h>
#include <vector>
#include <numeric>

using namespace std;

int main()
{
double value4 = 1.3; //vsetpoint commanded
double error_old = 0;

// create an ofstream for the file output
ofstream outputFile;

// create and open the .csv file
outputFile.open("exampleOutput.csv",fstream::app);

//write the file headers
outputFile << "time and date"<< ","<<
"position sensor (volts) avg" << ","<<
"sense resistor (amps) avg" << "," << "vbias" << ","
<< "v setpoint" << "," << "ambient temp volt avg"
<< "," << "coil temp volt avg" << "," <<
"error term volts" << "," << "commanded setpoint"
<<endl;

```

```
double j=0;
vector<double> vec;
double b;
double a;
double c;
double d;
double e;
double k;

int n=1;

while (true){

n=n+1;

int err, handle, i;

enum { NUM_FRAMES_CONFIG = 1 };
const char * aNamesConfig[NUM_FRAMES_CONFIG] = \
{"AIN_ALL_RESOLUTION_INDEX"};
const double aValuesConfig[NUM_FRAMES_CONFIG] = {9};
int errorAddress = INITIAL_ERR_ADDRESS;

// Set up for reading AIN value
double value ; //measurement that the LabJack
//is reading from the OKE sensor
const char * NAME = "AIN0"; //voltage value
```

```
// Open first found LabJack
handle = OpenOrDie(LJM_dtANY, LJM_ctANY, "LJM_idANY");

// Read AIN from the LabJack
err = LJM_eReadName(handle, NAME, &value);
ErrorCheck(err, "LJM_eReadName");

j++;
sleep(1); //sleeps for 1 second

vec.push_back(value);
b=accumulate(vec.begin(),vec.end(),0.0);
// ^^adds the values that are being stored
a=b/50; //the average when accumulating the past 50 values

if (j>=50)
{vec.erase (vec.begin());
// ^^erases all values past the last 50
c=0.7*value+0.3*a; //weighting the average

double value2;
const char * NAME2 = "AIN2";
err = LJM_eReadName(handle, NAME2, &value2);
ErrorCheck(err, "LJM_eReadName");

double value8 = -1.3; //desired setpoint
```

```
double error_new = value8 - c;

double ki = 0.01;

const int ADDRESS = 1002; //DAC1
const int TYPE = LJM_FLOAT32;
value4 = value4 - (ki*error_new + error_old);
// ^^commanded setpoint

err = LJM_Open(LJM_dtANY, LJM_ctANY, "LJM_idANY", &handle);
ErrorCheck(err, "LJM_Open");

error_old = error_new;

err = LJM_eWriteAddress(handle, ADDRESS, TYPE, value4);
ErrorCheck(err, "LJM_eWriteAddress");

// write to DAC0
const int ADDRESS1 = 1000; // DAC0
const int TYPE1 = LJM_FLOAT32;
double value3 = 1.0; //vbias

// Open first found LabJack
err = LJM_Open(LJM_dtANY, LJM_ctANY, "LJM_idANY", &handle);
ErrorCheck(err, "LJM_Open");
```

```
err = LJM_eWriteAddress(handle, ADDRESS1, TYPE1, value3);
ErrorCheck(err, "LJM_eWriteAddress");

time_t now = time(0);
char* dt = ctime(&now);

double value5; //ambient temp sensor
const char * NAME5 = "AIN3";
err = LJM_eReadName(handle, NAME5, &value5);
ErrorCheck(err, "LJM_eReadName");

double value6; //temp sensor at coil
const char * NAME6 = "AIN1";
err = LJM_eReadName(handle, NAME6, &value6);
ErrorCheck(err, "LJM_eReadName");

double value7; //error term
const char * NAME7 = "AIN4";
err = LJM_eReadName(handle, NAME7, &value7);
ErrorCheck(err, "LJM_eReadName");

//Setup & call eWriteNames to configure AIN0 on LabJack.
err = LJM_eWriteNames(handle, NUM_FRAMES_CONFIG,
aNamesConfig, aValuesConfig, &errorAddress);
ErrorCheckWithAddress(err, errorAddress, "LJM_eWriteNames");

int num=1; //for averaging
```

```
double sum1=0, avgval1;
double sum2=0, avgval2;
double sum3=0, avgval5;
double sum4=0, avgval6;
double sum5=0, avgval7;

for(int i = 1; i <= num; i++){
err = LJM_eReadName(handle, NAME, &value);
ErrorCheck(err, "LJM_eReadName");
sum1 += value;

err = LJM_eReadName(handle, NAME2, &value2);
ErrorCheck(err, "LJM_eReadName");
sum2 += value2;

err = LJM_eReadName(handle, NAME5, &value5);
ErrorCheck(err, "LJM_eReadName");
sum3 += value5;

err = LJM_eReadName(handle, NAME6, &value6);
ErrorCheck(err, "LJM_eReadName");
sum4 += value6;

err = LJM_eReadName(handle, NAME7, &value7);
ErrorCheck(err, "LJM_eReadName");
sum5 += value7;
}
```

```

avgval1 = sum1 / num;
avgval2 = sum2 / num;
avgval5 = sum3 / num;
avgval6 = sum4 / num;
avgval7 = sum5 / num;

cout<<value<<"     "<<c<<"       "<<error_new<<endl;

outputFile << dt << "," << avgval1 << "," << avgval2
<< "," << value3 << "," << value4 << "," << avgval5
<< "," << avgval6 << "," << avgval7 << "," << c << endl;
}}}
```

Listing A.5: C++ code main4rev3avgSLOWwithDigIntegRev3

```
/**
* Name: eWriteAddress.c
* Desc: Shows how to use the LJM_eWriteAddress function
**/

// For printf
#include <stdio.h>

// For the LabJackM Library
#include <LabJackM.h>

// input output stream that allows cin and
//cout to read continuously
#include <iostream>

// For LabJackM helper functions, such as OpenOrDie,
//PrintDeviceInfoFromHandle,
// ErrorCheck, etc.
#include "LJM_Utilities.h"

//to save to csv file
#include <fstream>
#include <string>

#include <ctime>
#include <chrono>
```

```

#include <unistd.h>
#include <vector>
#include <numeric>

using namespace std;

int main()
{
int initial_lpd_set_pt=1;
int final_lpd_set_pt=12;
double scaled_lpd_set_point;

for (initial_lpd_set_pt;initial_lpd_set_pt<=final_lpd_set_pt;
initial_lpd_set_pt++){
scaled_lpd_set_point=initial_lpd_set_pt*.1;
cout<<"scaled looped setpoint: "<<scaled_lpd_set_point<<endl;
double total_lpd_set_point= 1+scaled_lpd_set_point-.5;
cout<<"total looped setpoint: "<< total_lpd_set_point<< endl;

double j=0;

double value4 = total_lpd_set_point; //vsetpoint
double error_old = 0;

// create an ofstream for the file output
ofstream outputFile;
```

```

// create and open the .csv file
outputFile.open("exampleOutput.csv",fstream::app);

//write the file headers
outputFile << "time and date"<< ","<<
"position sensor (volts) avg" << ","<<
"sense resistor (amps) avg" << "," << "vbias" << ","
<< "v setpoint" << "," << "ambient temp volt avg" << ","
<< "coil temp volt avg" << "," << "error term volts"
<< "," << "commanded setpoint" <<endl;

vector<double> vec;
double b;
double a;
double c;
double d;
double e;
double k;

int n=1;

while (j<150){

n=n+1;

int err, handle, i;
```

```

enum { NUM_FRAMES_CONFIG = 1 };
const char * aNamesConfig[NUM_FRAMES_CONFIG] = \
{"AIN_ALL_RESOLUTION_INDEX"};
const double aValuesConfig[NUM_FRAMES_CONFIG] = {9};
int errorAddress = INITIAL_ERR_ADDRESS;

// Set up for reading AIN value
double value ;
//measurement that the LabJack is reading from the OKE sensor
const char * NAME = "AIN0"; //voltage value

// Open first found LabJack
handle = OpenOrDie(LJM_dtANY, LJM_ctANY, "LJM_idANY");

// Read AIN from the LabJack
err = LJM_eReadName(handle, NAME, &value);
ErrorCheck(err, "LJM_eReadName");

j++;
cout<< "iterator j: "<< j<< endl;
sleep(1); //sleeps for one second

vec.push_back(value);
b=accumulate(vec.begin(),vec.end(),0.0);
// ^^adds the values that are being stored
a=b/50; //the average when accumulating the past 50 values
```

```
if (j>=50)
{vec.erase (vec.begin());
// ^^erases all stored values past the last 50
c=0.7*value+0.3*a; //weighting the average

double value2;
const char * NAME2 = "AIN2";
err = LJM_eReadName(handle, NAME2, &value2);
ErrorCheck(err, "LJM_eReadName");

double value8 = -total_lpd_set_point;

double error_new = value8 - c;

double ki = 0.01;

const int ADDRESS = 1002; //DAC1
const int TYPE = LJM_FLOAT32;
value4 = value4 - (ki*error_new + error_old);

err = LJM_Open(LJM_dtANY, LJM_ctANY, "LJM_idANY", &handle);
ErrorCheck(err, "LJM_Open");

error_old = error_new;

err = LJM_eWriteAddress(handle, ADDRESS, TYPE, value4);
```

```
ErrorCheck(err, "LJM_eWriteAddress");

// write to DAC0
const int ADDRESS1 = 1000; // DAC0
const int TYPE1 = LJM_FLOAT32;
double value3 = 1.0; //vbias

// Open first found LabJack
err = LJM_Open(LJM_dtANY, LJM_ctANY, "LJM_idANY", &handle);
ErrorCheck(err, "LJM_Open");

err = LJM_eWriteAddress(handle, ADDRESS1, TYPE1, value3);
ErrorCheck(err, "LJM_eWriteAddress");

time_t now = time(0);
char* dt = ctime(&now);

double value5; //ambient temp sensor
const char * NAME5 = "AIN3";
err = LJM_eReadName(handle, NAME5, &value5);
ErrorCheck(err, "LJM_eReadName");

double value6; //temp sensor at coil
const char * NAME6 = "AIN1";
err = LJM_eReadName(handle, NAME6, &value6);
ErrorCheck(err, "LJM_eReadName");
```

```
double value7; //error term
const char * NAME7 = "AIN4";
err = LJM_eReadName(handle, NAME7, &value7);
ErrorCheck(err, "LJM_eReadName");

//Setup & call eWriteNames to configure AIN0 on LabJack.
err = LJM_eWriteNames(handle, NUM_FRAMES_CONFIG,
aNamesConfig, aValuesConfig, &errorAddress);
ErrorCheckWithAddress(err, errorAddress, "LJM_eWriteNames");

int num=1; //for averaging
double sum1=0, avgval1;
double sum2=0, avgval2;
double sum3=0, avgval5;
double sum4=0, avgval6;
double sum5=0, avgval7;

for(int i = 1; i <= num; i++){
err = LJM_eReadName(handle, NAME, &value);
ErrorCheck(err, "LJM_eReadName");
sum1 += value;

err = LJM_eReadName(handle, NAME2, &value2);
ErrorCheck(err, "LJM_eReadName");
sum2 += value2;

err = LJM_eReadName(handle, NAME5, &value5);
```

```
ErrorCheck(err, "LJM_eReadName");
sum3 += value5;

err = LJM_eReadName(handle, NAME6, &value6);
ErrorCheck(err, "LJM_eReadName");
sum4 += value6;

err = LJM_eReadName(handle, NAME7, &value7);
ErrorCheck(err, "LJM_eReadName");
sum5 += value7;
}

avgval1 = sum1 / num;
avgval2 = sum2 / num;
avgval5 = sum3 / num;
avgval6 = sum4 / num;
avgval7 = sum5 / num;
cout<<value<<"    "<<c<<"      "<<error_new<<endl;

outputFile << dt << "," << avgval1 << "," << avgval2
<< "," << value3 << "," << value4 << "," << avgval5
<< "," << avgval6 << "," << avgval7 << "," << c << endl;
}}}}
```

APPENDIX B: Mathcad Analysis

Flexure Stiffness Calculation for a Notch-Hinge Flexure

$E_{abs} := 2.5 \cdot 10^9 \cdot \frac{N}{m^2} = 2.5\ GPa$ Modulus of Elasticity - ABS

$E_{nylon} := 3 \cdot 10^9 \cdot \frac{N}{m^2}$ Modulus of Elasticity - Nylon

$E_{polycarb} := 2.6 \cdot 10^9 \cdot \frac{N}{m^2}$ Modulus of Elasticity - PC

$b := 6.35 \cdot mm$ depth of the spring

$t := 1.016 \cdot mm$ thickness

$a_x := 6.35\ mm$ radius of circular hinge

$\beta := \frac{t}{2 \cdot a_x} = 0.08$ dimensionless factor representing the hinge geometry

$$K_{abs} := \left(\left(\frac{3}{2 \cdot E_{abs} \cdot b \cdot {a_x}^2} \right) \cdot \left(\frac{1}{2 \cdot \beta + \beta^2} \right) \cdot \left(\left(\frac{3 + 4 \cdot \beta + 2 \cdot \beta^2}{(1+\beta) \cdot (2 \cdot \beta + \beta^2)} \right) + \left(\frac{6 \cdot (1+\beta)}{(2 \cdot \beta + \beta^2)^{\frac{3}{2}}} \right) \cdot \operatorname{atan}\left(\sqrt{\frac{2+\beta}{\beta}} \right) \right) \right)^{-1} = 0.473\ N \cdot m$$

$K_{linear\_abs} := 0.923\ \frac{N}{mm}$

$$K_{nylon} := \left(\left(\frac{3}{2 \cdot E_{nylon} \cdot b \cdot {a_x}^2} \right) \cdot \left(\frac{1}{2 \cdot \beta + \beta^2} \right) \cdot \left(\left(\frac{3 + 4 \cdot \beta + 2 \cdot \beta^2}{(1+\beta) \cdot (2 \cdot \beta + \beta^2)} \right) + \left(\frac{6 \cdot (1+\beta)}{(2 \cdot \beta + \beta^2)^{\frac{3}{2}}} \right) \cdot \operatorname{atan}\left(\sqrt{\frac{2+\beta}{\beta}} \right) \right) \right)^{-1} = 568.032\ N \cdot mm$$

$K_{linear\_nylon} := 1.108\ \frac{N}{mm}$

$$K_{pc} := \left(\left(\frac{3}{2 \cdot E_{polycarb} \cdot b \cdot {a_x}^2} \right) \cdot \left(\frac{1}{2 \cdot \beta + \beta^2} \right) \cdot \left(\left(\frac{3 + 4 \cdot \beta + 2 \cdot \beta^2}{(1+\beta) \cdot (2 \cdot \beta + \beta^2)} \right) + \left(\frac{6 \cdot (1+\beta)}{(2 \cdot \beta + \beta^2)^{\frac{3}{2}}} \right) \cdot \operatorname{atan}\left(\sqrt{\frac{2+\beta}{\beta}} \right) \right) \right)^{-1} = 492.294\ N \cdot mm$$

$K_{linear\_PC} := 0.961\ \frac{N}{mm}$

APPENDIX C: Matlab Analysis

Analysis of the flexure design is based on considering each segment of the flexure as a simple cantilever beam. Delta bending 1, 2, and 3 calculate the displacement of each segment given a force. The displacement of each segment is then added together to give a total displacement of the flexure.

Contents

- range of forces
- delta bending 1
- delta bending 2
- delta bending 3
- delta total


```
clc
clear

E=320000; %units psi
G=118500; %units psi

s=1; %units inches
b=0.5; %units inches
h=0.06; %units inches (h=a)
d=1.25; %units inches
L=2.0; %units inches
```

range of forces

```
f=0:0.01:0.05;
f_t=f'
```

```
f_t =

         0
    0.0100
    0.0200
    0.0300
    0.0400
    0.0500
```

delta bending 1

```
d_b1=[f.*s.^3]./[0.25.*E.*b.*h.^3];
```

delta bending 2

```
d_b2=[f.*L.^3]./[0.25.*E.*b.*h.^3];
```

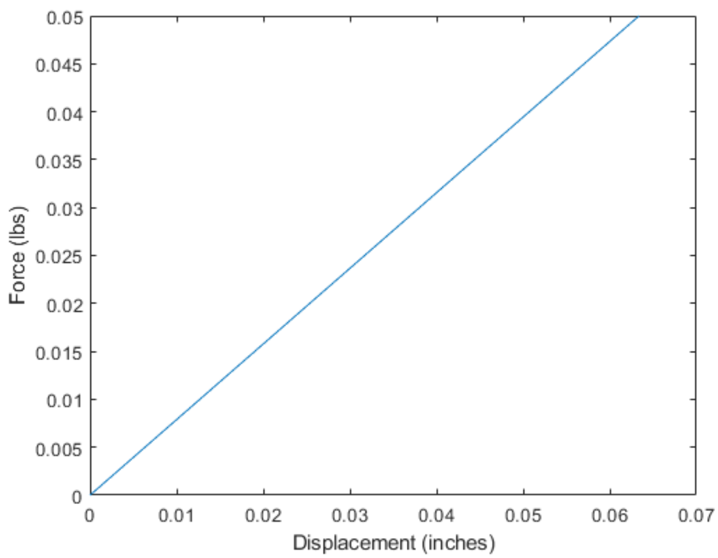
delta bending 3

```
d_b3=[f.*d.^3]./[0.25.*E.*b.*h.^3];
```

delta total

```
d_t=d_b1+d_b2+d_b3;
d_tt=d_t'

I=(b.*h.^3)/12    %units in^4

plot(d_t,f)
%ylim([0 0.08])
ylabel('Force (lbs)')
xlabel('Displacement (inches)')

fit1=fit(d_tt,f_t,'Poly1')
```

```
d_tt =

         0
    0.0127
    0.0254
    0.0380
    0.0507
    0.0634


I =

   9.0000e-06


fit1 =

     Linear model Poly1:
     fit1(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =      0.7888  (0.7888, 0.7888)
       p2 =  -1.004e-17  (-3.719e-17, 1.71e-17)
```



Published with MATLAB® R2018b