Mechanical Chaotic Oscillator

Brandon M. Thacker
Dr. Eric Ayars, Advisor

8/5/2015

EXECUTIVE SUMMARY

In this paper we explain the procedure for building a mechanical chaotic oscillator or MCO. The objective of this project was to create an apparatus for the Advanced Lab that enables students to observe nonlinear dynamics and chaos in an efficient and intuitive way .

The apparatus uses two Helmholtz coil pairs perpendicular to each other, one to provide a constant magnetic field within a given region of space and the other to give a perpendicular time varying field component in that same region. This region contains a magnetic dipole connected to a rotating shaft that supports an aluminum inertial disk. The inertial disc supplies some mass and also provides a means for magnetic damping. The MCO uses an optical encoder with a quadrature output signal to track the position of the magnetic dipole. A microcontroller collects and processes the position data while also generating a signal for the time varying field. The user has complete control over system parameters using a computer with a USB interface. System commands and inquiries can be made in real time. The MCO is a nonlinear damped oscillator capable of producing a chaotic trajectory given specific system parameter values.

The Advanced Lab is an excellent environment for undergraduate physics students to learn about nonlinear dynamics and chaos. Students can learn this concept more efficiently if they can control and track the systems behavior. While chaos can be observed in many experiments, few of them allow the control and precise measurement capabilities provided with the MCO. Because of these features, the MCO is a valuable tool for analyzing nonlinear dynamics in the Advanced Lab.

MCO Student Skill Set

By Brandon Thacker

8/30/2015

The student will obtain the following skills while preforming the "Chaotic Rotor" lab using the MCO.

- Phase diagrams
- Poincaré plots
- Bifurcation maps
- LabVIEW/Python programming
- Computational modeling
- Error analysis

Historical Discussion of the Mechanical Chaotic Oscillator

By Brandon Thacker

8/30/2015

Sometime in the 90's the Daedalon Corporation produced a chaotic pendulum apparatus for the undergraduate physics lab. This came with the apparatus and an interface board. While the chaotic pendulum apparatus was effective, it was expensive. The Interface board was designed for IBM PC's and required an ISA slot and windows 3.1, both of which are obsolute. The Chaotic pendulum made by Daedalon is no longer being produced.

The Mechanical Chaotic Oscillator (MCO) can provide a replacement for Daedalon's pendulum while taking advantage of some technological advancements. This MCO is inexpensive and easy to manufacture. We use an on-board microchip that is many times smaller than Daedalon's interface board so the MCO can be connected to any computer via USB. The low cost build of the MCO makes it a reasonable way to study nonlinear systems in the undergraduate Advanced Lab.

Chaotic Rotor Student Manual

Overview of the Instrument

The Chaotic Rotor apparatus allows quantitative analysis of the motion of a rotating magnetic dipole in an oscillating magnetic field. This motion is analogous to the motion of the driven damped pendulum. Nearly every parameter on the apparatus is adjustable, including the strength of the fixed field, the frequency and amplitude of the drive field, the damping parameter, and the rotational inertia of the rotor. Precise time and position information is reported to a computer via USB at intervals spaced evenly throughout the drive cycle, allowing the experimenter to generate not only phase-space plots of the motion but also a series of Poincaré sections for the motion.

The instrument is controlled via a Cortex-M4 microprocessor (on an Arduino-compatible "Teensy 3.1" development board) which manages all communications with the computer, generates the oscillating drive field, tracks time and position of the rotor, and allows complete user control over the behavior of the apparatus via GPIB-style commands.[1]

Powering the Instrument

There are two sets of Helmholtz coils affecting the rotor. They are powered by two separate DC power supplies: an adjustable DC supply of up to 12V for the fixed-field coils, and a bipolar ± 12VDC supply for the control board, which in turn supplies current to the drive coils. 3.3V logic-level voltage is supplied by a 5V-3.3V converter on the Teensy 3.1 board, powered by the USB connection to the computer.

The larger of the two coil pairs generates a fixed magnetic field, which establishes an equilibrium position for the rotor. This field is analogous to the gravitational field for the driven damped pendulum; the advantage with this apparatus is that by changing the current in these field coils one can effectively adjust the strength of "gravity". Field strength is set by adjusting the voltage on an external supply connected directly to the coils. Please adjust the current limit on this supply to no more than 1.0 A.

The smaller inner coils (drive coils) are controlled by the microprocessor and the associated amplifier circuit on the controller board. The amplifier circuit requires ±12VDC from a supply capable of driving up to 1.0 A.

[1]The instrument is not GPIB/IEEE 488.2-compliant as of the current firmware version, but the commands will be quite familiar in format and usage to anyone familiar with GPIB or SCPI communications.

Communicating with the Instrument

Communications between the Chaotic Rotor apparatus and the computer are via a virtual serial port on USB at a bauad rate of 115.2 kBd. Parameters are controlled (or polled) using 4-character GPIB-style commands sent from any serial-capable program. (LabVIEW is an obvious choice, but Python is also quite good for this and one can even control it by typing commands directly into a terminal emulator program.) Commands are case- and whitespace-insensitive.

HELP Return a summary of available commands.

***IDN** Return a description of the apparatus and the firmware version.

***RST** Reset instrument to default conditions: Coil off, Report off, f = 1.0 Hz, A = mid-range, phase and position both set to 0.

***TST** Test for proper centering adjustment. This command is included for SCPI compatibility; there is no way at this point for the hardware to do a self-test so the response will invariably be '0'.

***ESR** Report the Error Status Registor.

***CSR** Clear Error Status Registor Any error will turn on the Error light on the controller board: *CSR turns it off.

FREQ ? | (float) If followed by '?', the frequency command returns the current frequency setting for the function generator. If followed by a floating-point number, it sets the frequency to approximately that value in Hz. The precision of the internal function generator is limited by the method of generating the output waveform. The microcontroller adjusts the output 256 times per drive cycle. It calculates from the frequency the number of microseconds to wait between each step; at higher frequencies the quantization of this microsecond count seriously messes with the output frequency and above 3,906 Hz the output breaks entirely. For this apparatus, though, 10 Hz is a "ridiculously high" frequency so it's not a problem.

AMPL ? | (0-1000) The amplitude is set on a scale from 0 to 1000, where 1000 is close to an amplitude of 10 V_p. A query '?' will return the current amplitude, an integer value will set the amplitude. Values less than 0 or greater than 1000 will set the amplitude to 0 or to 1000, respectively.

TRAK ? | (1 | 0) The user can turn tracking on or off with 'TRAK 1' or 'TRAK 0', respectively. 'TRAK ?' returns the current tracking setting. The default value is 1, and I can't think of any reason anyone would turn this off but the option is there if needed.

REPT ? | (1 | 0) setting report to 1 turns on data reporting, 0 turns it off. '?' returns the current report setting. Default value is 0.

COIL ? | (1 | 0) Setting coil to 1 turns on the coil drive output, 0 turns it off. '?' returns the current coil drive output state. Default value is 0.

ZERO Defines current position as 0 degrees.

POSN? Returns current position, in degrees.

TIME? Returns current time, in microseconds.

SAVE (0-9) Saves current frequency and amplitude to non-volatile memory in memory slot 0-9.

LOAD (0-9) Sets frequency and amplitude to the values previously saved in memory slot 0-9.

As noted before, this instrument is not fully IEEE 488.2 compatible. One thing that does not work yet is the ability to string commands together with semicolons: "FREQ 2.0 ; AMPL 750" correctly sets the frequency to 2.0 Hz but then sets ESR bit 5 without changing the amplitude to 750. Another issue is that the command parser *always* takes the first four characters of the command. For a GPIB-compatable instrument the commands 'FREQ' and 'Frequency' should work equally well, but the latter will cause an error on this device.[2]

Other Adjustments

The damping parameter is adjusted by using the micrometer screw at the top of the instrument to bring the magnet near the rotor flywheel. Useful values of damping seem to occur for distances of 5 mm or less, although this is an area of active research still.

The inertia of the rotor can be changed by physically changing the flywheel. The default flywheel is a fairly lightweight aluminum disk, but there is a much heavier brass disk available also and of course you can use the

[2] "It's on my list!" –Shreck

lathe or 3D printer to make something different. It might be interesting to make an attachment to fit just below the aluminum flywheel that would allow controllable fine adjustment of the rotational inertia.

Pre-lab

Before beginning experimental work on this apparatus, you should:

- Derive the equation of motion for the apparatus. You may assume that damping is primarily due to eddy currents in the flywheel rather than friction in the bearings.
- Understand the minimum requirements for chaotic motion, and how one would determine whether a motion is chaotic or not.
- Become comfortable with Phase Space plots. Understand what they show and how to interpret them.
- Understand what is meant by Period Doubling, and how it relates to the onset of chaotic motion.
- Understand Poincaré sections: how are they created, what do they indicate, and how do they relate to phase space plots?

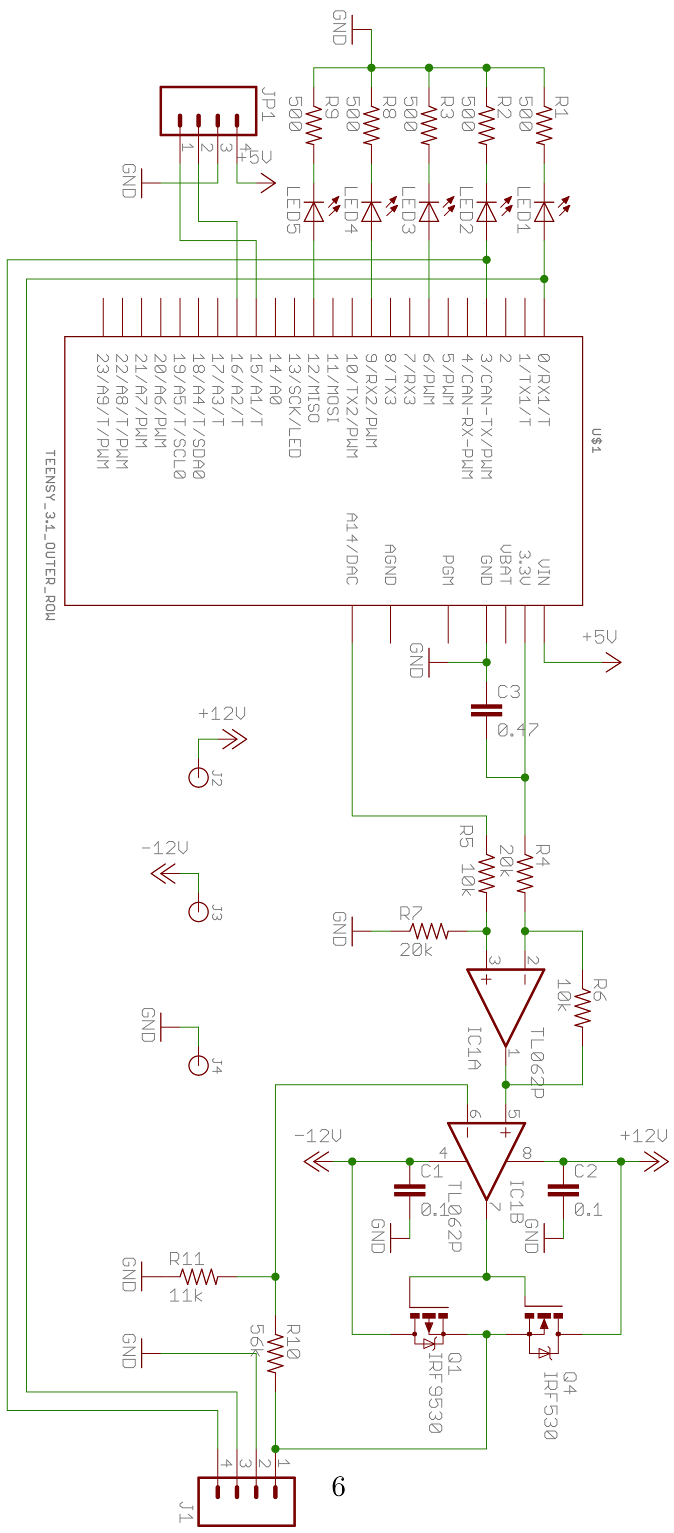
Experiment

Here are some things that you can try, in order of increasing impressiveness.

1. Write a LabVIEW or Python program to control and collect data from the device. (This is the bare minimum, really.)
2. Find an "interesting" frequency-amplitude area and plot phase-space diagrams for the rotor.
3. Plot a Poincaré section for the rotor, showing either 'normal' or 'strange' attractors depending on the motion.
4. Make a movie showing the evolution of a strange attractor through the drive cycle.
5. Compare your results with a numerical solution of the equations of motion for the apparatus.
6. Make a bifurcation map for a given frequency (amplitude) showing period-doubling as amplitude (frequency) is swept. (This is difficult, but quite impressive.)

Version

Experiment manual version 1.0, by Eric Ayars. Written to accompany firmware version Teensy3.1_20160608 or greater.



U$1
TEENSY_3.1_OUTER_ROW
0/RX1/T
1/TX1/T
2
3/CAN-TX/PWM
4/CAN-RX-PWM
5/PWM
6/PWM
7/RX3
8/TX3
9/RX2/PWM
10/TX2/PWM
11/MOSI
12/MISO
13/SCK/LED
14/A0
15/A1/T
16/A2/T
17/A3/T
18/A4/T/SDA0
19/A5/T/SCL0
20/A6/PWM
21/A7/PWM
22/A8/T/PWM
23/A9/T/PWM
VIN
3.3V
VBAT
GND
PGM
AGND
A14/DAC
JP1
+5V
GND
R1 500
R2 500
R3 500
R8 500
R9 500
LED1
LED2
LED3
LED4
LED5
C3 0.47
R4 20k
R5 10k
R6 10k
R7 20k
IC1A TL062P
IC1B TL062P
C1 0.1
C2 0.1
+12V
-12V
J2
J3
J4
R10 56k
R11 11k
Q1 IRF9530
Q4 IRF530
J1

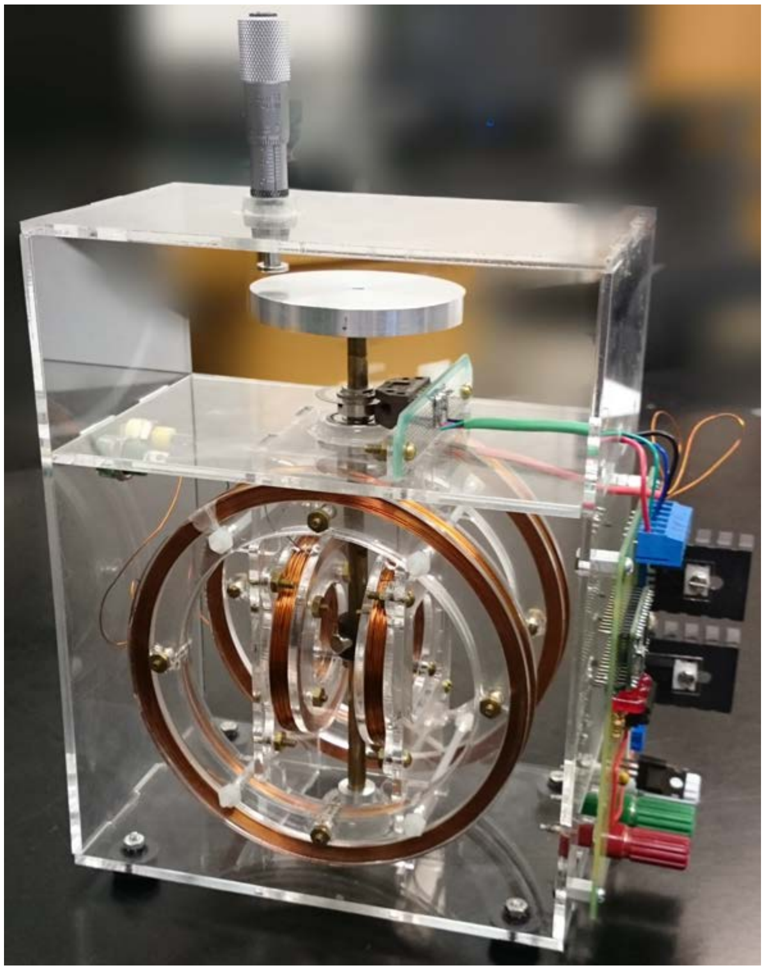
Mechanical Chaotic Oscillator

Brandon M. Thacker
Dr. Eric Ayars, Advisor

7/21/2016

Abstract

This project objective is to create a tool to analyze the behavior of a chaotic system in the advanced lab environment. Here we explain the design and modeling of a chaotic oscillator along with fabrication and assembly methods. Processes for obtaining and processing data, as well as electrical design are discussed. Using a USB and computer, the user is given control over system parameters and access to system diagnostics. The resulting apparatus is a nonlinear oscillator capable of producing a chaotic trajectory while providing a unique hands-on way to study chaos.



Contents

1 Introduction

For most undergraduate physics students, chaos theory is a concept of very little if any exposure. Perhaps a student may get an opportunity to study chaos in a computational physics course where numerical methods provide a manageable treatment of nonlinear systems. Nonlinear dynamics and chaos are prominent features in nature and deserve a thorough introduction in the undergraduate physics curriculum. The Advanced Lab is an excellent environment for undergraduate students to explore chaotic systems in detail. The ability to observe a nonlinear system approach chaos gradually through a succession of period doubling bifurcations would make this introduction more intuitive[1]. This would also give a student a better understanding of what it means for a system to be deterministic versus random. This paper explains the process of building a mechanical chaotic oscillator that can serve this purpose.

It is not difficult to build a system that exhibits some form of chaos, a damped driven pendulum would do the trick [2]! However, it is no trivial matter to create an apparatus that does so while allowing for complete user control over system parameters and precise measurement of the state of the system as time evolves. These are some of the requirements needed for this project to be successful in the Advanced Lab setting. The mechanical chaotic oscillator or MCO presented fulfills these needs in an effective and elegant way.

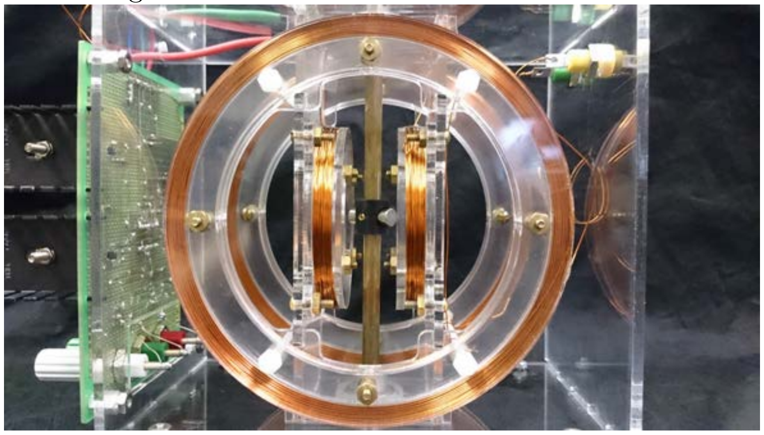
[1]A simple experiment for studying the transition from order to chaos Meissner, H. and Schmidt, G., American Journal of Physics, 54, 800-804 (1986), DOI:http://dx.doi.org/10.1119/1.14449

[2]Chaos and the simple pendulum De Jong, Marvin L., The Physics Teacher, 30, 115-121 (1992), DOI:http://dx.doi.org/10.1119/1.2343491

2 Concept of Design

The Helmholtz coil has always been a popular lab technique for creating a uniform magnetic field for experimentation. It also has the quality that the strength of the field it produces is linearly proportional to the current through its coils. This provides the convenience of precise field control, a feature we make significant use of with the MCO.

Figure 1: The Mechanical chaotic oscillator.



The apparatus is one involving magnetic interactions so nonmagnetic materials were used wherever possible to minimize interference. The apparatus uses two Helmholtz coil pairs, one to provide a constant magnetic field within a given region of space and the other to give a perpendicular time varying field component in that same region. This region contains a magnetic dipole connected to a rotating shaft that supports an aluminum inertial disk. This dipole motor configuration has been observed to produce chaotic behavior[3]. The inertial disc supplies both rotational inertia and a means for magnetic damping. An optical encoder with a quadrature output signal tracks the position of the dipole. This signal is read by a microcontroller programmed to retrieve and process this information, as well as generate the signal for the time varying field. The microcontroller is able to receive specific commands through a serial connection. This lets the user control system parameters, obtain system diagnostics and access a help menu to

[3]The bipolar motor: A simple demonstration of deterministic chaos Ballico, M. J. and Sawley, M. L. and Skiff, F., American Journal of Physics, 58, 58-61 (1990), DOI:http://dx.doi.org/10.1119/1.16320

learn how to use the apparatus from a computer. The position and time data for the system is sent from the microcontroller to a computer for further processing and use. This apparatus is a damped non-linear oscillator that allows the user control of the system through a serial connection.

3 Modeling the System

The equation of motion was determined in the variable ϕ and using numerical methods a phase space diagram was produced along with a strange attractor. The theoretical model provided a magnificent poincaré plot showing this strange attractor. This occurred after adjusting system parameters such as drive frequency ω_d, damping coefficient β, magnetic moment μ and field strengths B_{field} and B_{drive}. Equation (1) below is the equation of motion we obtained for the MCO system where I is the moment of inertia determined by the mass distribution of the inertial disc and shaft.

$$\ddot{\phi} = \frac{\mu}{I}[B_{field}\sin\phi + B_{drive}\cos\phi\sin\omega_d t] - \frac{\beta\dot{\phi}}{I} \tag{1}$$

This second-order nonlinear differential equation was then written as a system of two first order equations, (2) and (3). Here we denote the rotational frequency of the dipole as ω_ϕ to differentiate from the drive frequency ω_d. These equations where then plotted in phase space using a Python program.

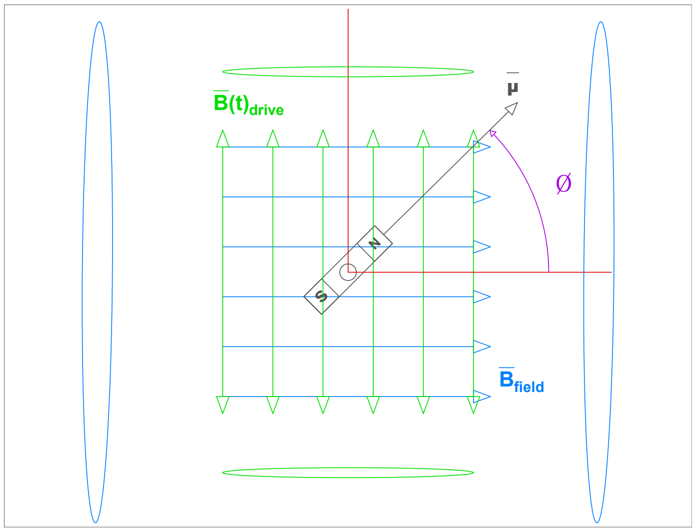
$$\dot{\omega}_\phi = \frac{\mu}{I}[B_{field}\sin\phi + B_{drive}\cos\phi\sin\omega_d t] - \frac{\beta\omega_\phi}{I} \tag{2}$$

and

$$\dot{\phi} = \omega_\phi \tag{3}$$

The figure below shows the resulting phase space plot of our theoretical model for 4000 laps of the drive frequency. The dark blue set of points are obtained by plotting one dot at the same point in each period; they show the strange attractor for this chaotic motion.

Figure 2: Diagram of magnetic dipole in uniform field region.



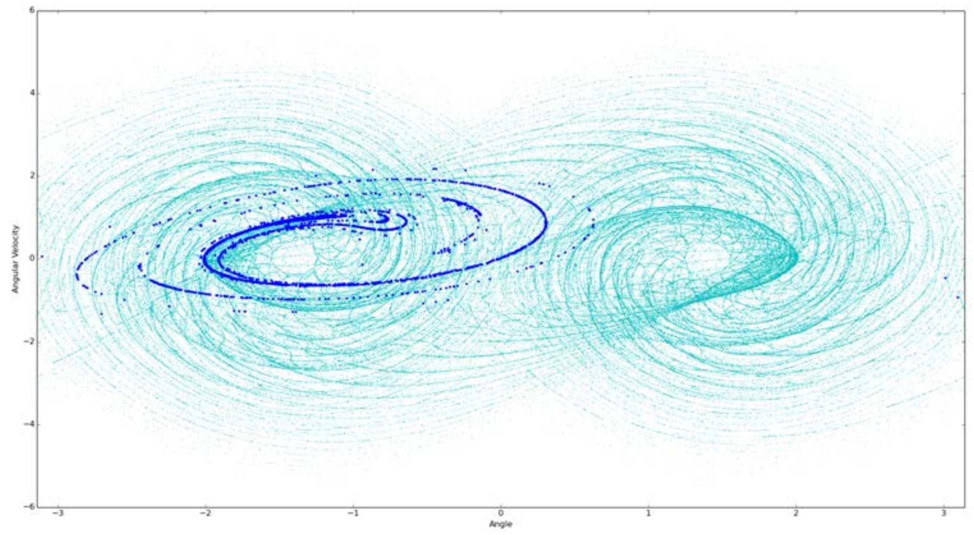
4 Materials and Fabrication

The MCO is made up of many components and it is important to choose the right materials for each. The system is magnetic and so magnetic materials would effect the resulting trajectory if they where in close proximity to the dipole. While in some cases magnetic materials were used, the majority of the apparatus is nonmagnetic and no magnetic materials are present on or inside the coils other than the rotating dipole.

Case and Coils

The majority of the apparatus is made of acrylic. We were able to design these parts in AutoCAD and cut them from acrylic sheets using a laser cutter. This includes the case which uses a tongue and grove technique for assembly (bonded together using solvent welding) and the coils which are

Figure 3: Chaotic Oscillator phase space plot with strange attractor.



each comprised of four layers of acrylic held together with brass fasteners. Once assembled, the coils were wound on a lathe using 26 gauge magnet wire with 170 turns each. The two sets of coils are secured together using coil mount plates with brass and nylon fasteners.

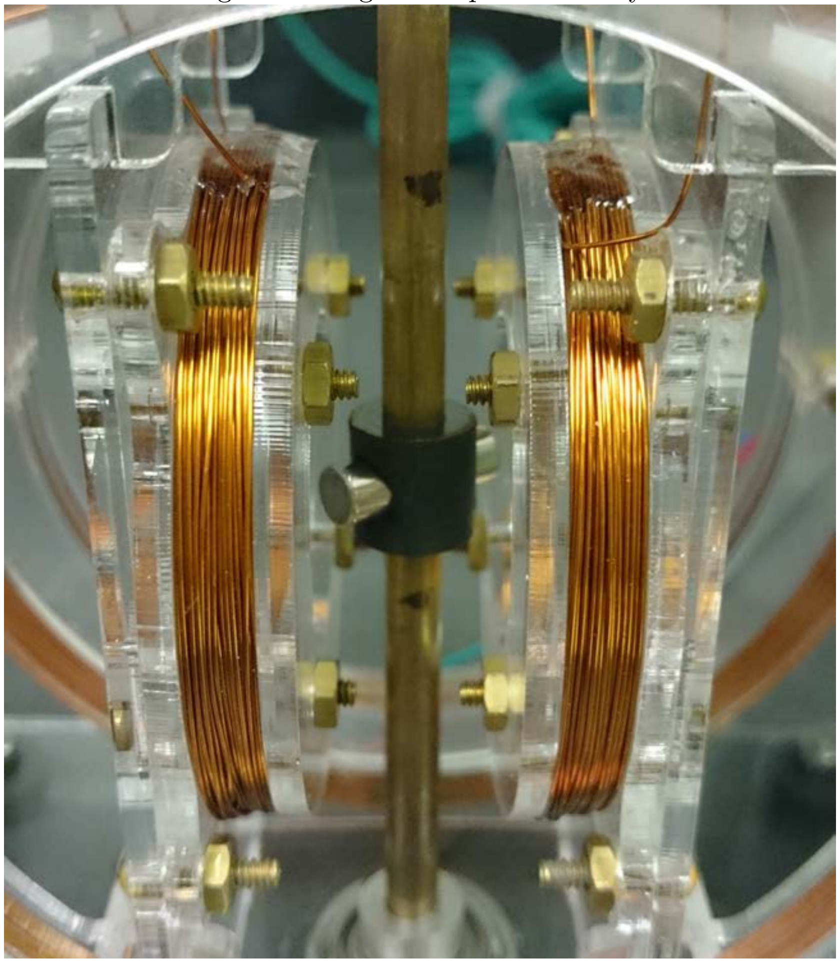
The Shaft and Its Components

For the shaft, again we needed a nonmagnetic material and so we used brass. The shaft serves three main purposes, support the inertial disk, hold the optical encoder disk in place and hold the magnetic dipole in place, while rotating freely. The shaft was chosen to be quarter inch in diameter because the bearings, encoder disk and inertial disk all had a quarter inch drive.

Bearings and friction

The bearings that hold the shaft in place are mounted in two locations on the case. One bearing is set in the base of the case centered below the coils and the other is set in the shelf just above the coils. These are made of steel and could have some small effects on the dipole's trajectory; the effect should be minimal though due to the radial symmetry. It is worth mentioning here that the rotational axis is vertical, thus eliminating gravitational effects. The only uncontrolled torque on the rotor is from the bearings. We assume that these small frictional contributions are negligible compared to the magnetic damping.

Figure 4: Magnetic dipole assembly.

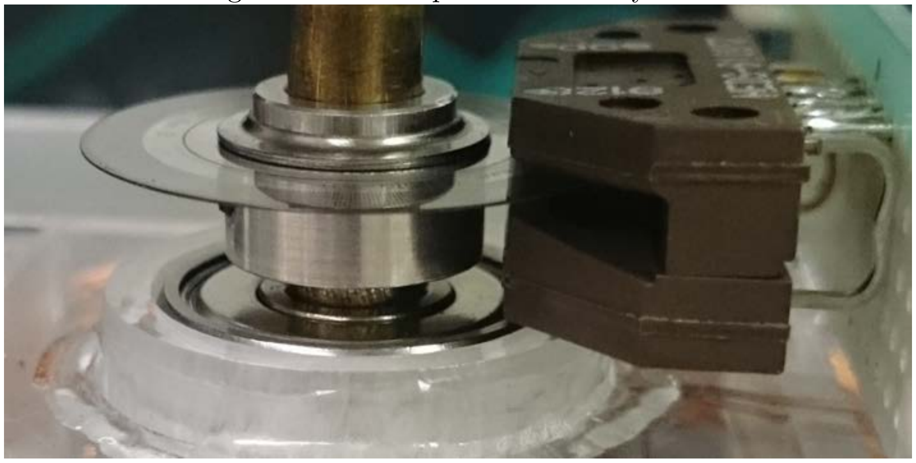


The magnetic dipole

The magnetic dipole is mounted on the shaft at the center of the uniform field region between the coils. The dipole is made of two cylindrical magnets with a depth, diameter and spacing of a quarter inch. The mount that holds the magnets in place was designed in AutoCAD and printed in ABS on a

3D printer. The mount slides onto the shaft into position and held in place with a brass set screw.

Figure 5: HEDS optical encoder system.



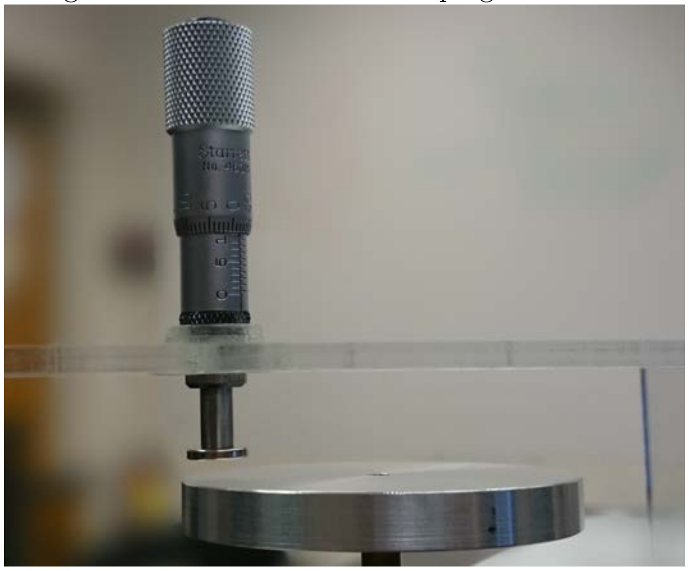
Optical encoder disk and module

Located just above the shelf at the top of the coils is an encoder disc that is used to track the position of the shaft. The disc is paired up with a HEDS optical encoder module that is mounted next to the shaft on the upper shelf face. The HEDS module gives a quadrature output signal that is processed by a microcontroller.

Inertial disk and damping

Above the optical encoder at the top of the shaft is an aluminum inertial disk. As the name suggests the inertial disk provides the majority of our mass. We were able to use it as a means of magnetic damping as well. This was achieved by mounting a micrometer screw adjuster in the top face of the case with a magnet attached to the extending end. The micrometer screw allows for a precise and measurable approach to the aluminum disk. When the disk rotates in proximity to the magnet the changing magnetic flux through the disc will create eddy currents within the disk and according to Lenz's law a drag will occur on the disk opposing the motion.

Figure 6: Inertial disk and damping mechanism.



5 Electronics

The MCO has an on-board microcontroller that is programmed to perform three main processes: produce a user controlled drive signal to the circuitry which supplies a current to the drive coils, process the quadrature signal from the optical encoder to get the trajectory for the dipole and provide an interface between the MCO and the computer.

Drive coil circuit

The output signal from our microcontroller's DAC is an analog sine wave varying between 0 and 3.3V. We however would like to produce an oscillating current through the coils capable of reaching 1 to 2 amps. This requires some circuitry and external power. The method used goes as follows.

First we shifted and amplified the DAC signal using a summing amplifier. The amplifier has two inputs, one from the DAC signal at 0-3.3V and the other connected to a -5V regulator through a trimpot and buffer circuit. The trimpot allows for manual zeroing of the summed signal. The signal is then amplified to $\pm$ 10V by putting the appropriate valued resistor on the feedback loop of the op-amp.

Next we use a push pull circuit to drive the coils. The now 10V peak

Figure 7: Electrical schematic for the MCO.

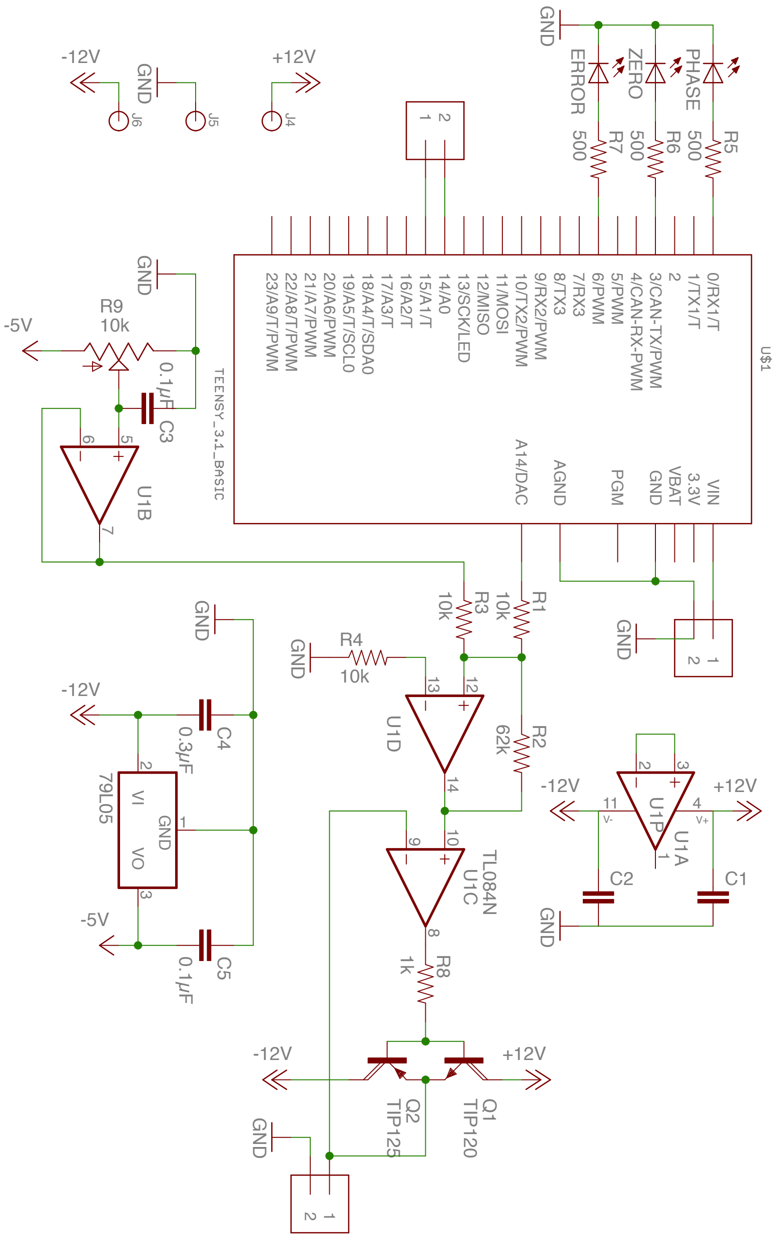
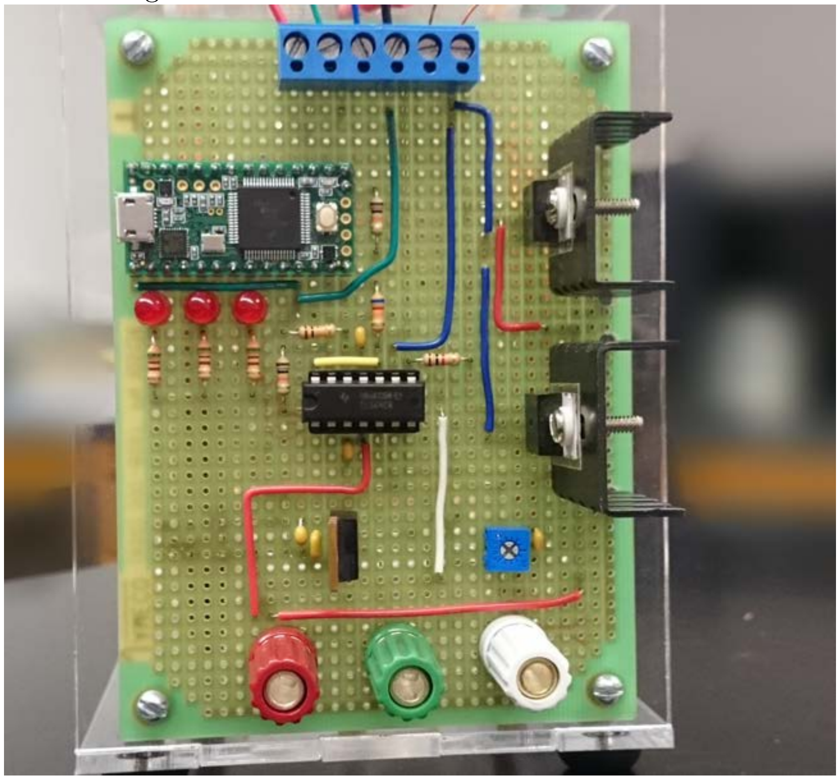


Figure 8: Electrical circuit and microcontroller.



AC signal is fed to the positive input of an op-amp. The output is sent to the base of two transistors simultaneously. The MCO circuit uses the TIP 120(NPN) and TIP 125(PNP) power transistors supplied with +12V and -12V respectively. Both emitters are connected together and out to the coils. The coil lead is also connected to the negative output of the op-amp, so as as the input voltage varies between +10V and -10V the power transistors are letting through a smoothly varying alternating current for supplying the coils. Even though this process is not normally linear for transistors, in this case the feedback from the op-amp forces a linear response.

Data Processing and Serial Communication

The microcontroller is tasked with resolving the quadrature signal from the optical encoder module into position data. It also tracks time, so velocity data can be obtained. The MCO program developed by Dr. Ayars uses one of the quadrature lines to trigger an interrupt to track the motion of the dipole and the other one to specify the rotational direction at the time of the interrupt.

The microcontroller can report this time, drive phase and position data to the computer in real time using the IEEE 488.2 serial command protocol. The microcontroller also allows for the user to submit commands to and request information from the MCO in real time. This is done through the serial protocol, and can be used with many software options such as LabVIEW. The table below lists the specialized commands created for the MCO.

Table 1: List of commands for the mechanical chaotic oscillator.

Command	Explanation
*IDN?	Equipment and firmware ID
∗RST	Reset to default condition
∗TST	Test for centering adjustment (not yet implemented)
∗ESR	Report (and clear) error status register
FREQ (?) or (value)	Request (?) or set (value) drive frequency
TRAK (?) or (1/0)	Request tracking status (?) or set tracking on (1) or off (0)
REPT (?) or (1/0)	Request reporting status (?) or set reporting on (1) or off (0)
COIL (?) or (1/0)	Request coil-drive status (?) or set coil on (1) or off (0)
AMPL (?) or (value)	Request (?) or set (value) amplitude, 0-1000
ZERO	Define current position as zero
POSN?	Request current position
TIME?	Request current time in microseconds
SAVE (0-9)	Save current parameters in slot (0-9)
LOAD (0-9)	Load saved parameters from slot (0-9)

6 Results

Considerations

The MCO requires the ability to collect large data sets. The main reason for this is that chaotic motion only repeats over very large periods, so the true characteristics of the motion is best seen when the long term behavior of the system is observed. Another experimental consideration is the effects of a transient on the systems trajectory. The initial state of the system will most certainly introduce a transient element into the motion of the system. If the initial state of the system does not coincide exactly with the systems natural trajectory, there will be some more complicated behavior. The transient will eventually die out and the system will settle into its natural trajectory; whether the systems motion is simple harmonic or chaotic.

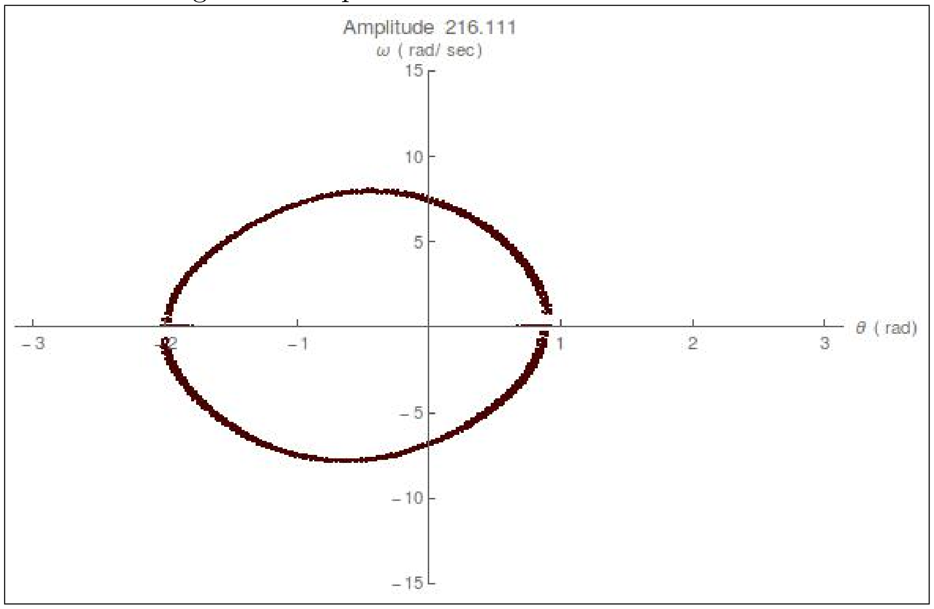
Data Acquisition

The MCO data was obtained using two different serial interface methods, a LabVIEW GUI (Graphical User Interface) and a Python program using PySerial. The GUI was designed to communicate with the MCO and retrieve a stream of data. This data was stored in a string within LabVIEW, and written to a text file once the collection process was stopped. The data was then imported into Mathematica for further analysis. While LabVIEW provided a simple way to communicate with the MCO it also had some difficulty retrieving the data, it would dump chunks of data in a unpredictable manner. The Python program does essentially the same thing while adding even more options for collecting the data. As before a stream of data is stored in memory and written out to a text file, however, using PySerial the data dumps were no longer an issue. The program also allows the user to sweep through parameter values, specify the number of data points desired and adjust the step size of the sweeps. With these modifications the MCO could collect a range of system trajectories for many hours or even days on its own. The Python program does not offer the user friendly GUI experience but given its stability and flexibility it is currently the method being used for data acquisition.

Data Visualization

The data is best represented in phase space (also referred to as state space). For the MCO, phase space is a two dimensional graph of angular velocity vs. angular position. These two quantities are conveniently provided by the microcontroller for every drive phase step. In phase space a simple harmonic motion corresponds to an ellipse traced out over and over. Figure (9) below is a phase space plot of the MCO data with the system driven at its natural frequency.

Figure 9: Simple harmonic motion of the MCO

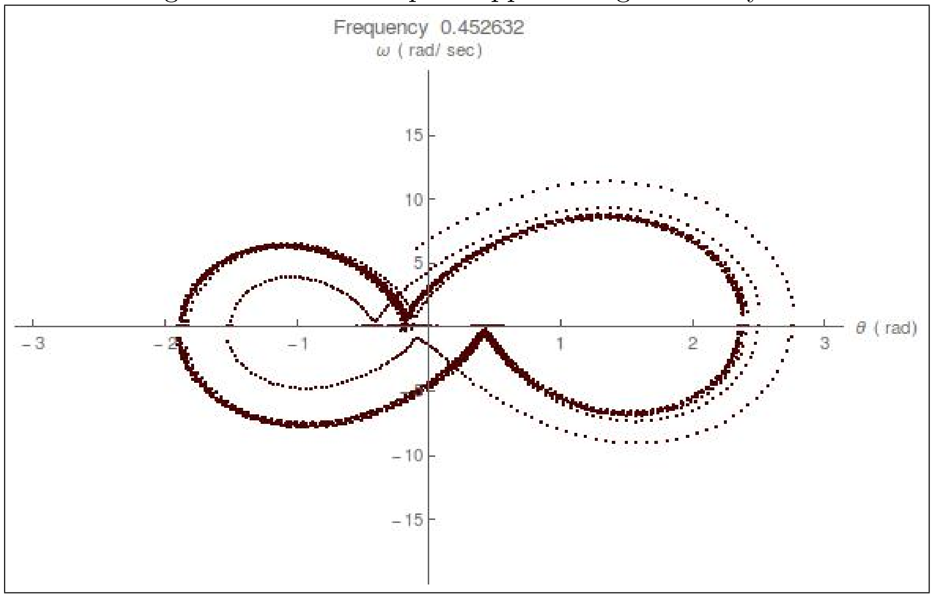


This closed trajectory is a limit cycle of the system for the current parameter configuration. As mentioned before, if the initial conditions (values of ϕ and $\dot{\phi}$) correspond to a point in phase space that is not on this limit cycle it will approach this closed trajectory asymptotically, this is the dying out of the transient behavior. In figure (10) we can see a transient converging to the natural limit cycle for another parameter configuration of the MCO. For this specific case the transient is almost completely diminished after two periods.

Chaos

Though there are many interesting characteristics of nonlinear dynamical systems the main purpose of the MCO is to allow for a user to observe chaos. The new question becomes; where do we start looking? Chaos will appear discretely for special combinations of parameter values, the game

Figure 10: Transient path approaching a limit cycle



is to fix all but one parameter and observe the long term behavior of the system for small variations of the free parameter. So we set the MCO to give us 50,000 data pairs for every sweep step of the frequency. The sweep range was 0.1 - 0.94(rads/sec) and the step size was 0.06(rads/sec) giving us 14 phase space portraits. The MCO had collected 750,000 data pairs in less than 30 minutes. As it turned out we did find an abrupt transition to chaos going from 0.76(rads/sec) to 0.82(rads/sec). These portraits are shown in figures (11) and (12).

The random looking trajectory in figure (12) is theoretically just as predictable as the motion seen in figure (11), it is a deterministic system in either case. This means is that if the initial state of the system is defined "exactly", equation (1) determines uniquely the state at any moment in time. This is of course if you can find the solution, an impossible task for most second order nonlinear differential equations.

Figure 11: Steady periodic motion at 0.76(radians/second)

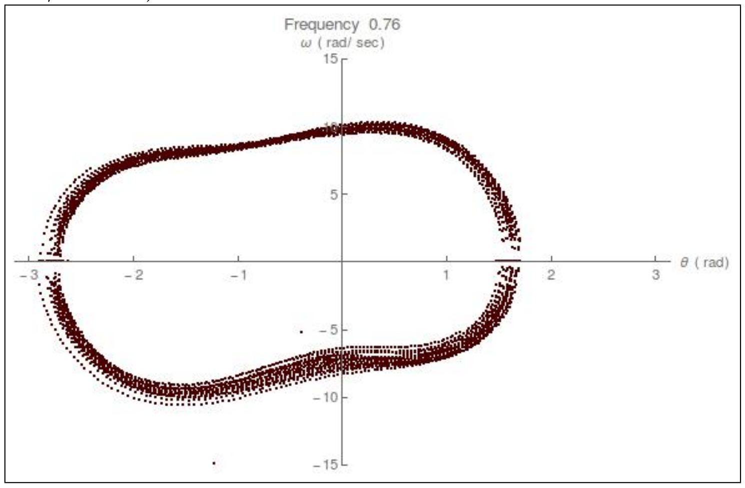
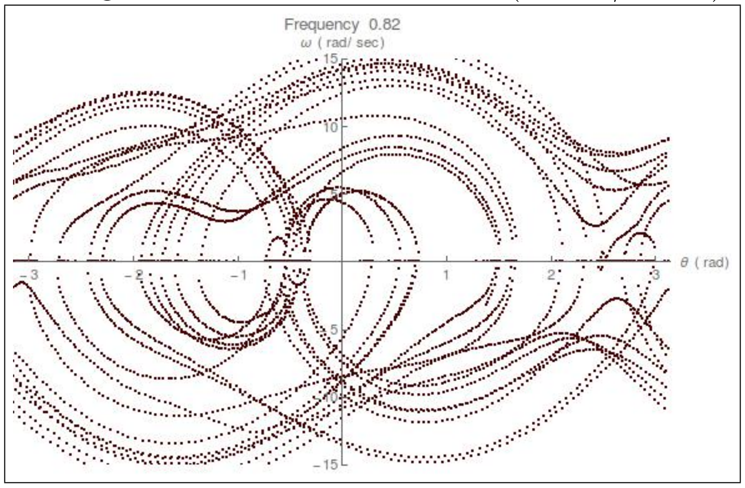


Figure 12: Chaotic motion at 0.82(radians/second)



7 Discussion

Issues and improvements

The MCO has shown its ability to provide a way to observe chaos in the Advanced lab as intended. The design however can be improved in many ways. The bearings used in the beta model were not of instrumentation grade and are imposing friction that is nonuniform over all angles. The next version will have low friction instrumentation bearings. One other issue is that the accuracy of our data is not as clear as we would like. This will be addressed by replacing the optical encoder assembly with a higher resolution disc and encoder. The method for zeroing the swing of the current through the drive coils is currently done manually using a trim-pot on the board, this is a touchy means for controlling an important aspect of the MCO. Eventually this will be done automatically in the electronics and the board will be sent out for professional assembly.

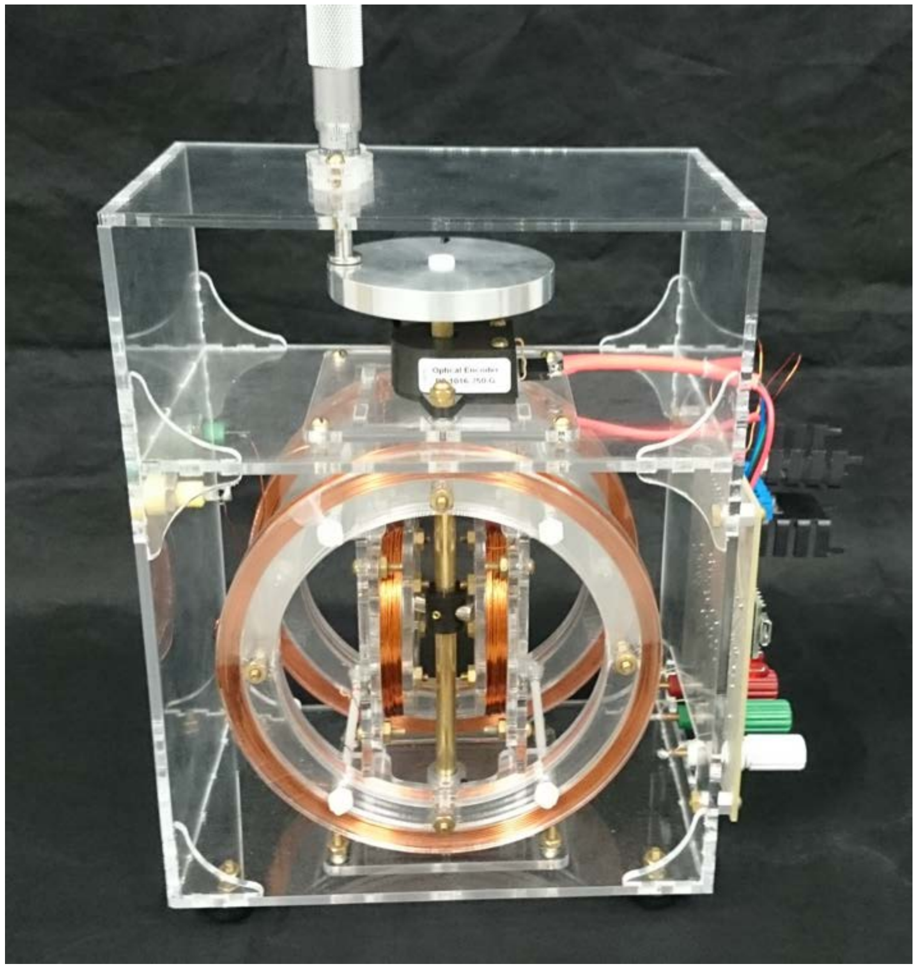
Experimental plans for the MCO

Now that we can produce chaos with the MCO there are many interesting ways to analyze the data. We can start by driving the system at resonance and run through the period doubling sequence and create a bifurcation diagram. poincaŕe sections can be made for all 256 steps of the drive phase and animated. A program can be written to determine the Liapunov exponents for small differences in initial conditions. This could be a method for locating chaos while sweeping through parameter values. The MCO provides many opportunities for exploring chaos through analyzing the data.

8 Modifications and the New MCO

While the original version of the MCO did what it was intended for, it revealed some issues that could effect its performance and the accuracy of the data it produced. For this reason we rebuilt the MCO to solve these issues before moving forward with data collection. Below is an image of the new MCO with its modifications.

Figure 13: The New MCO



Modifications

The modifications include structural supports for the case, new precision bearings, optical encoder and encoder disk system and significant electrical modifications.

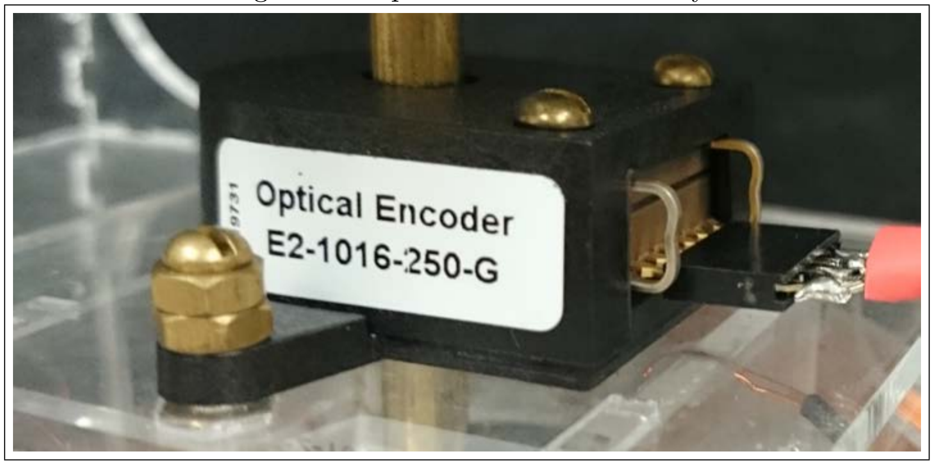
Support

The first and perhaps most obvious modification is that of structural integrity. From the image above we can see that the bottom corners and shelf of the MCO now have supports to help prevent any shear stresses from putting unwanted torque on the shaft. There is also an improved alignment mount for the damping micrometer and plates on the floor and shelf for stabilizing the bearings.

Encoder

The optical encoder and disk were upgraded to provide a resolution of 1016 steps per revolution. This is a significant improvement on the previous 360 step resolution. The effects are particularly noticeable when inspecting the smoothness of the data. We also included an enclosure for the encoder and disk to prevent error due to environmental debris and external light sources.

Figure 14: Optical Encoder Assembly



New Circuitry

Dr. Ayars redesigned the circuit board to maintain the drive signal with great precision and accuracy. In place of the manual zero point adjustment used in the old model, the zero point is now held constant using a voltage divider with precision resisters. The overall design itself was simplified and the board was sent out for a professional print. Figure (15) is an image of the new board and figure (16) is the new schematic.

Figure 15: Circuitry

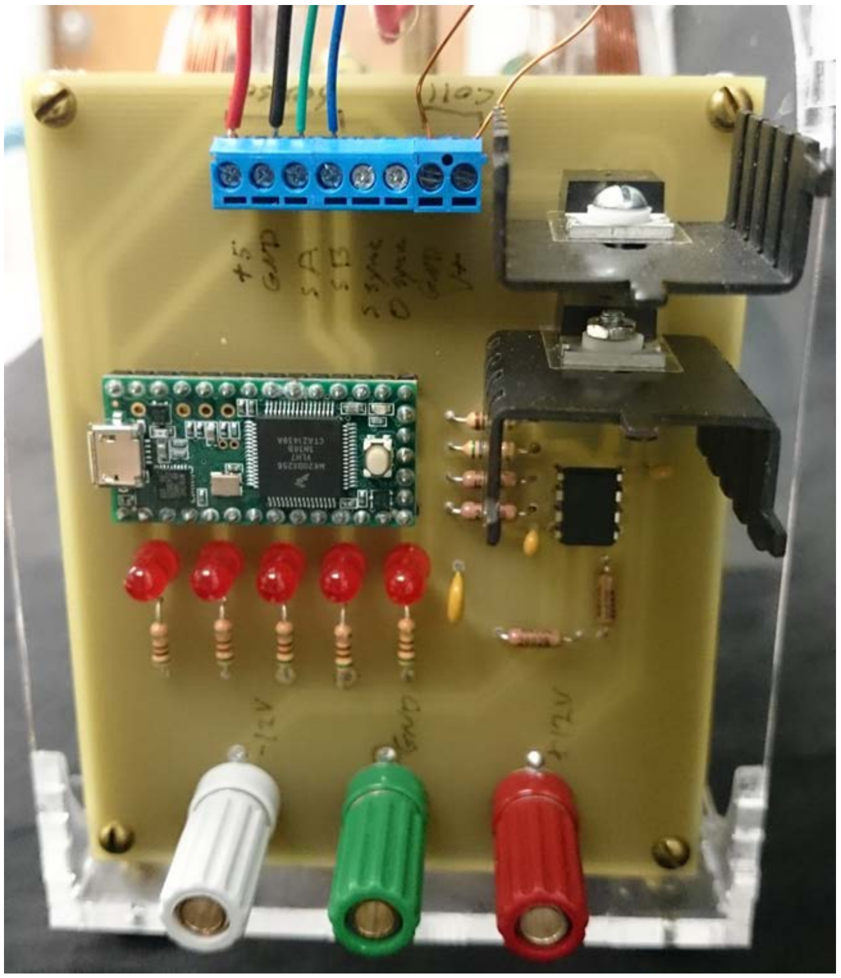
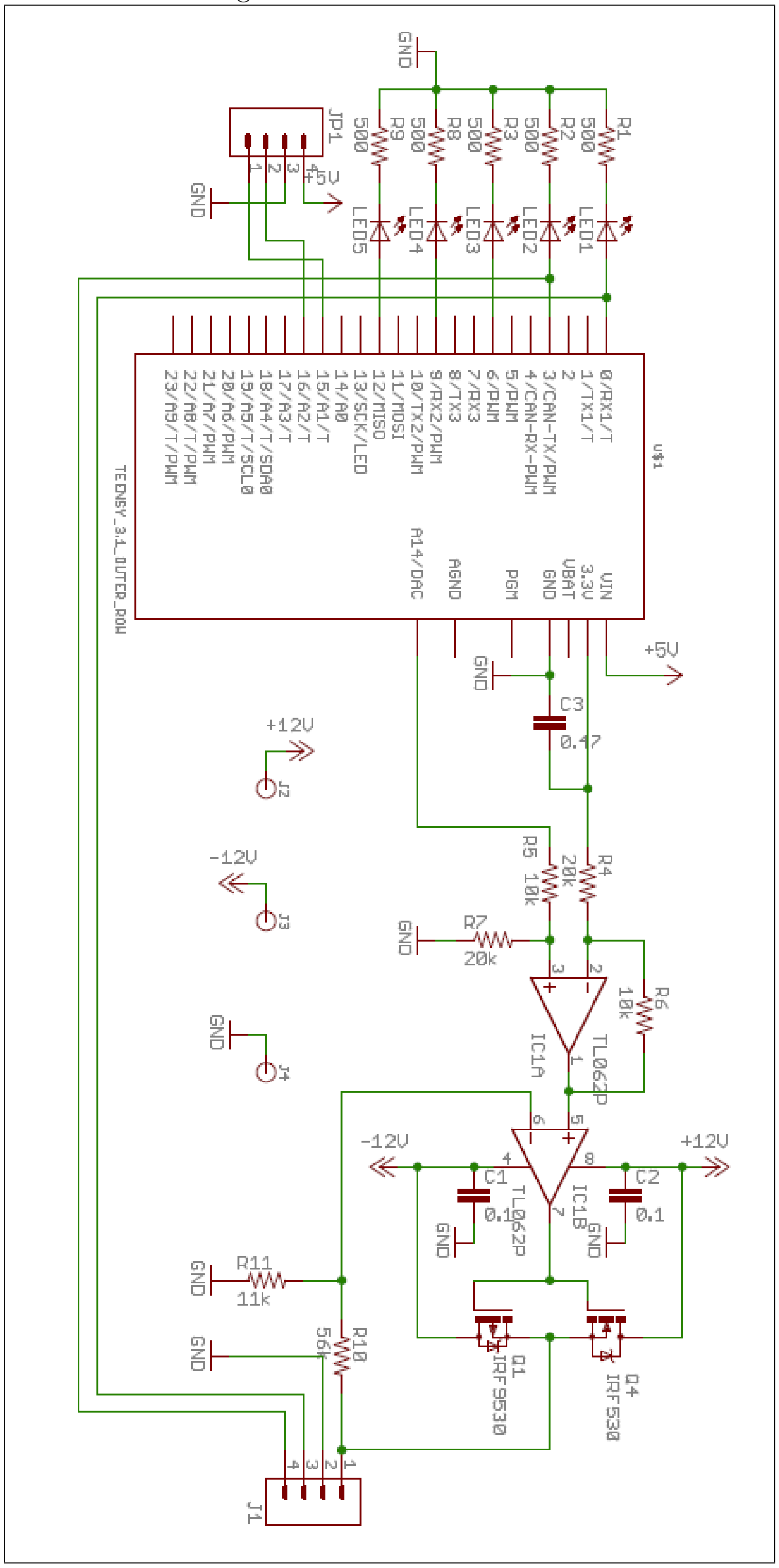


Figure 16: New MCO Circuit



The -5V regulator was done away with completely and a difference amplifier was used to subtract off half of the 3.3V reference voltage provided by the Teensy, giving a signal varying between ±1.56V. This signal is then fed to a class B current amplifier with voltage amplification on the feedback. The result is a ±10V signal capable of producing way more current then we could ever need. The MCO's drive circuit is currently outputting a maximum of 700mA.

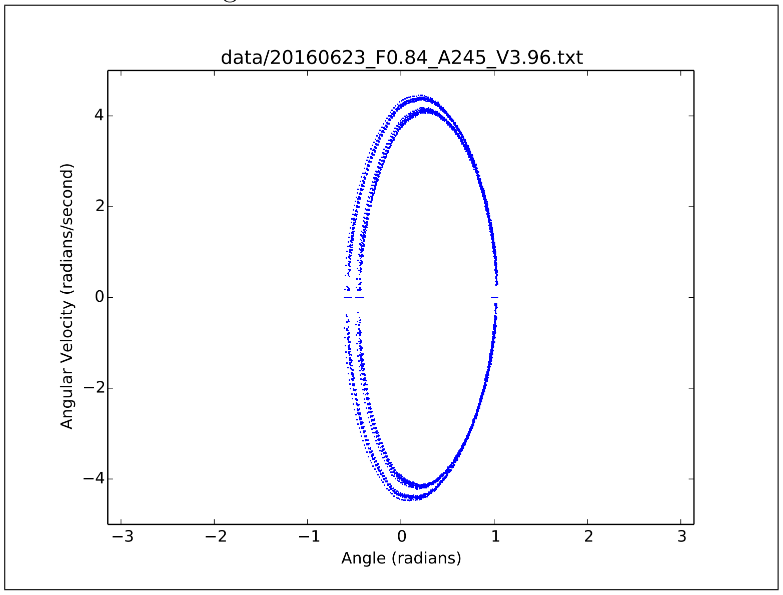
bearings

The new MCO was outfitted with high precision ABEC 7 instrumentation grade bearings to decrease frictional effects as much as possible. The bearings were also embedded in the shelf and floor of the MCO as to provide more radial stability when operating.

New MCO Data

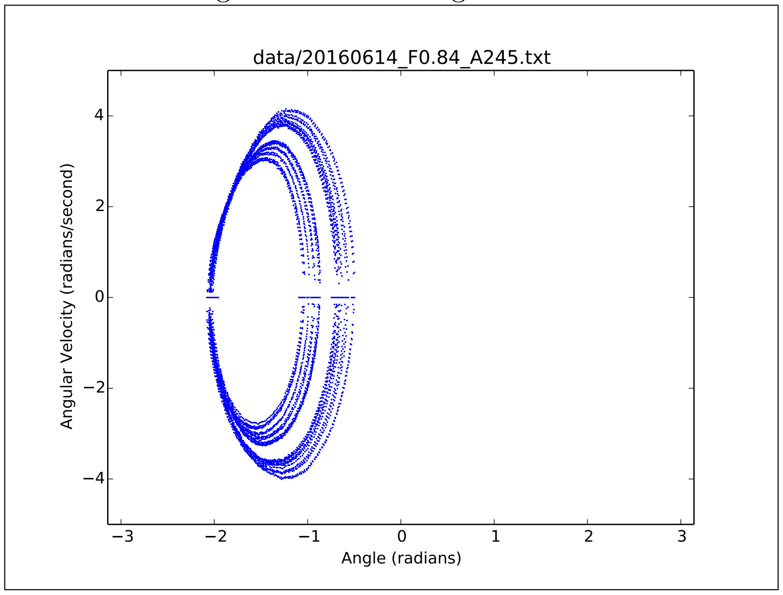
Once all of the modifications were in place we decided to focus on looking for evidence of period doubling specifically. The first step was to find a region of parameter space that admits what could be a transition to chaos.

Figure 17: Period Two Motion



This was not hard to find, we just started with a simple harmonic oscillation driven at the natural frequency and started varying control parameters individually, i.e the drive field amplitude $B_{drive}(t)$[4]. Eventually the SHO broke and gave way to the period two motion seen in figure (17). The dipole is alternating between two simple harmonic trajectories with small differences in their period. This is a noteworthy achievement since period doubling is characteristic of a chaotic system. While finding subsequent period doubling becomes a tedious process, we have found some motions that are promising. Figure (18), although not quite as clear, is what looks to be a period eight motion.

Figure 18: Period Eight Motion

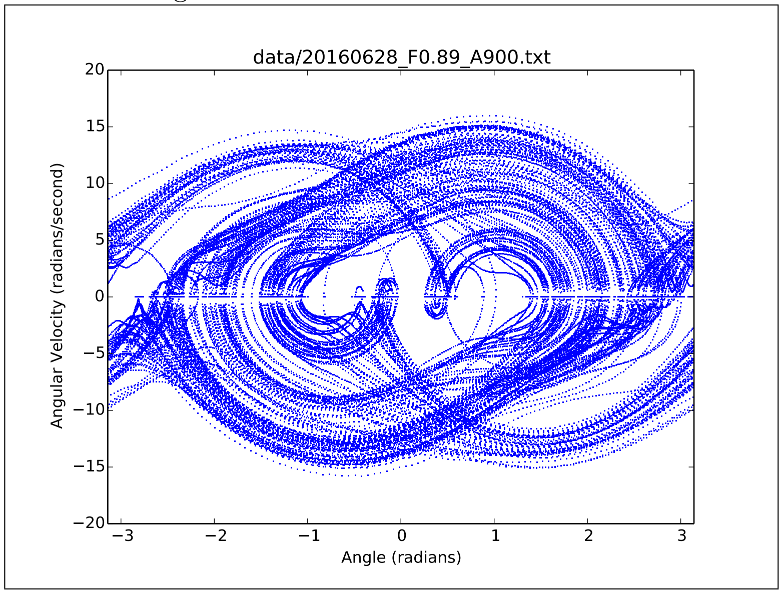


Eventually we get an abrupt transition to an unpredictable trajectory we are assuming is chaos. Most often we are confronted with something

[4]Harmonic generation and chaos in an electromechanical pendulum Marega Jr., E. and Ioriatti, L. and Zilio, S. C., American Journal of Physics, 59, 858-859 (1991), DOI:http://dx.doi.org/10.1119/1.16742

that looks like figure (19). To further exemplify chaotic behavior with the MCO we will look at another feature of chaos, the poincaré section. This is done by plotting every data point that corresponds to a common drive phase value; it could be π, $3\pi/2$ or any value between 0 and 2π.

Figure 19: Chaos With the New MCO



The drive cycle is built out of 256 values (0-255) within the firmware and is provided by the Teensy upon request. This means once we have the data we can program a "for loop" to pluck out and plot all data points with a specific phase value. For simple harmonic motion we would get all data at one point in phase space, for period two we get two points and so on. However, if we are dealing with chaos we get something called a strange attracter. Figures (20) and (21) are two strange attracters for drive phase values 0 and 127 respectively, this is a 90^o drive phase difference. Another way to visualize the strange attracter is to create a movie that shows the strange attracters for each of the 256 drive phase steps evolving through the drive cycle. A poincaré evolution of the MCO in a chaotic state can be viewed at: `www.youtube.com/watch?v=WKRFZYFWcJA`

Figure 20: poincaŕe Section, Drive Phase 0

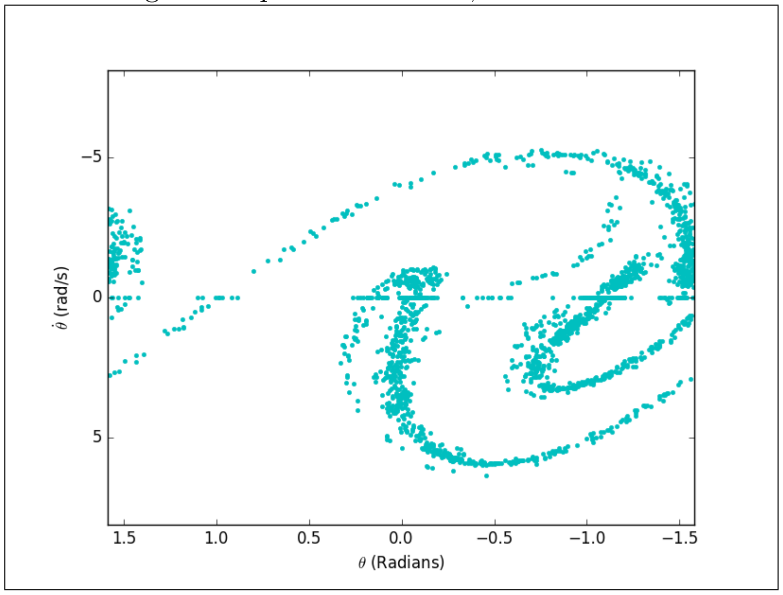
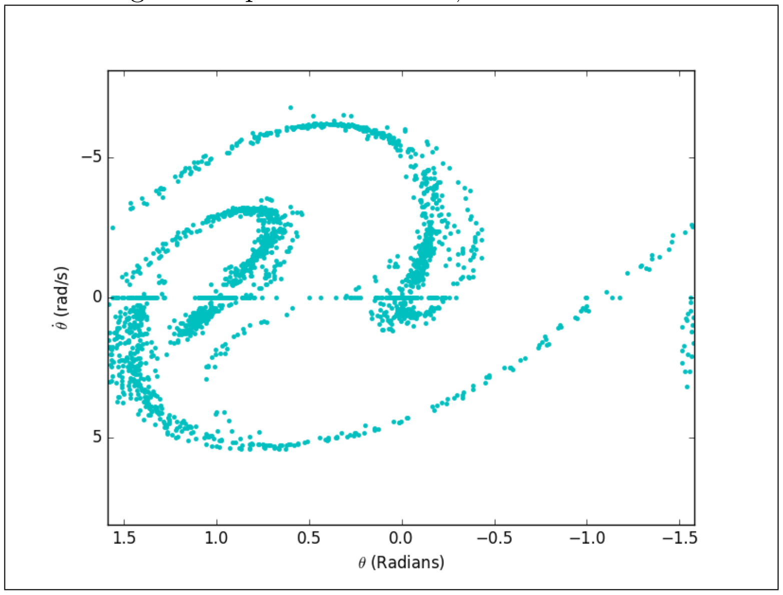


Figure 21: poincaŕe Section, Drive Phase 127



9 Conclusion

The Mechanical Chaotic Oscillator offers the student a way to learn about a nonlinear dynamical system while having substantial control of the systems parameters. Characteristics such as period doubling, poincaré sections and strange attracters can be observed. The MCO apparatus provides the user with the tools for acquiring precise data while observing the nonlinear dynamical processes with their own eyes.

Technical References

Brandon M. Thacker

August 8, 2016

1. The bipolar motor: A simple demonstration of deterministic chaos Ballico, M. J. and Sawley, M. L. and Skiff, F., American Journal of Physics, 58, 58-61 (1990), DOI:http://dx.doi.org/10.1119/1.16320

2. A simple experiment for studying the transition from order to chaos Meissner, H. and Schmidt, G., American Journal of Physics, 54, 800-804 (1986), DOI:http://dx.doi.org/10.1119/1.1444

3. Harmonic generation and chaos in an electromechanical pendulum Marega Jr., E. and Ioriatti, L. and Zilio, S. C., American Journal of Physics, 59, 858-859 (1991), DOI:http://dx.doi.org/10.1119/1.16742

4. Chaos and the simple pendulum De Jong, Marvin L., The Physics Teacher, 30, 115-121 (1992), DOI:http://dx.doi.org/10.1119/1.2343491

5. The route to chaos in a dripping water faucet Dreyer, K. and Hickey, F. R., American Journal of Physics, 59, 619-627 (1991), DOI:http://dx.doi.org/10.1119/1.16783

The Mechanical Chaotic Oscillator

BY: BRANDON THACKER

What is Chaos?

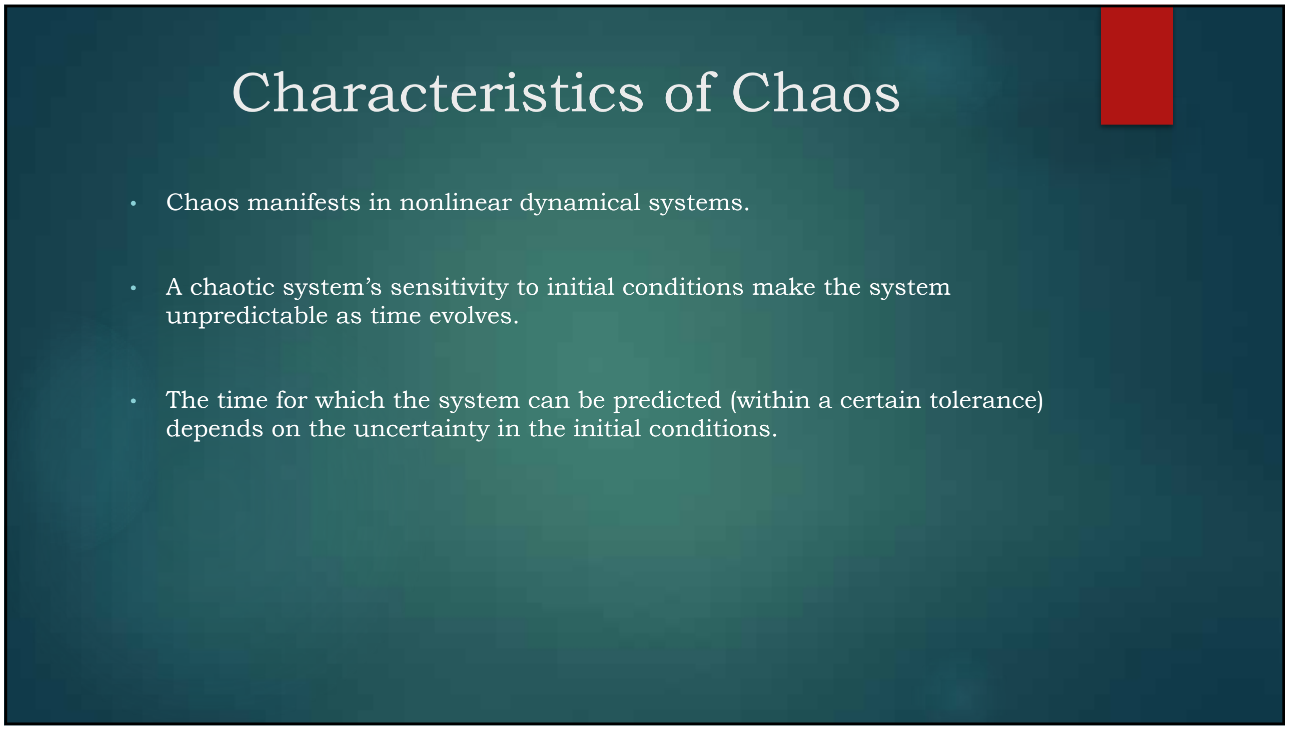
Deterministic vs. Stochastic Systems

Stochastic Systems

- Random process
- Every possible state of a system is equally probable.
- The state of a system is independent of its past or future states.

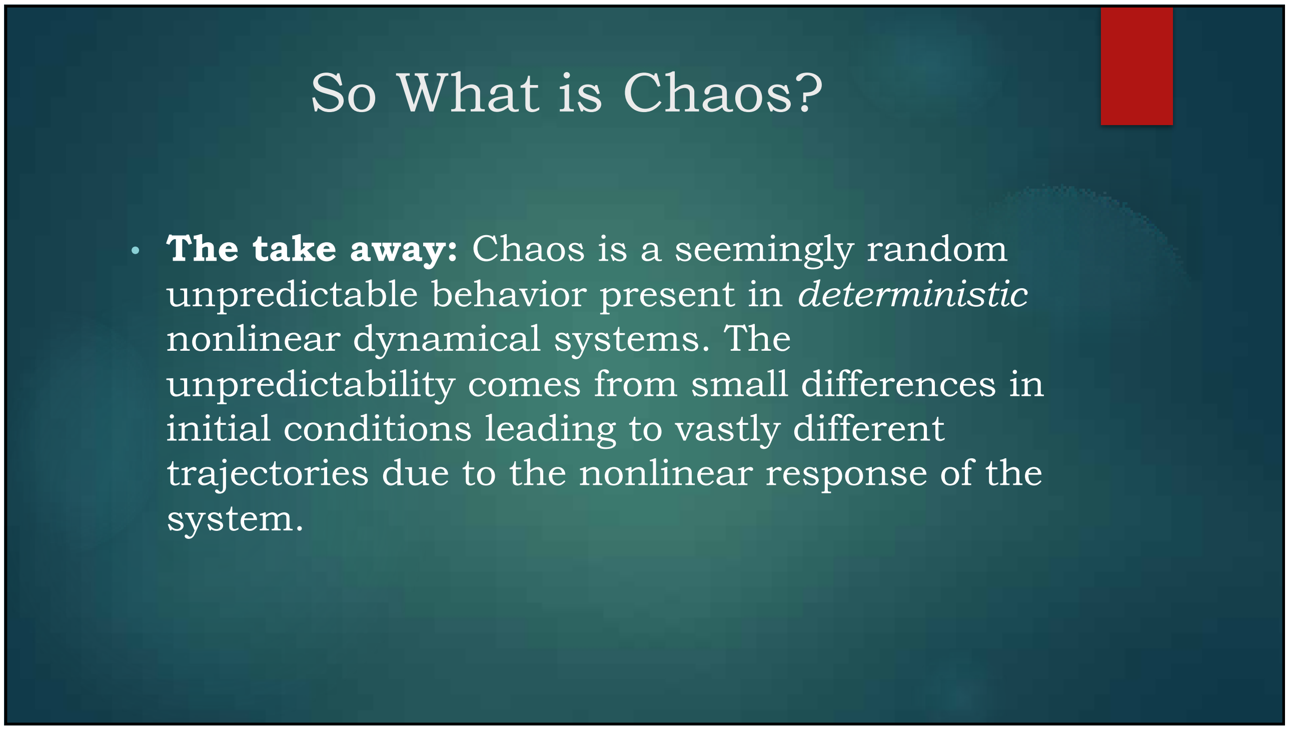
Deterministic Systems

- A deterministic system is allowed only one unique time evolution of states for a given set of initial conditions. **Proof:** “The Uniqueness Theorem.”
- If you have complete knowledge of the state of a system (state variables), all possible future and past states can be determined.
- Deterministic systems are “in theory” completely predictable.



Characteristics of Chaos

- Chaos manifests in nonlinear dynamical systems.
- A chaotic system’s sensitivity to initial conditions make the system unpredictable as time evolves.
- The time for which the system can be predicted (within a certain tolerance) depends on the uncertainty in the initial conditions.



So What is Chaos?

- **The take away:** Chaos is a seemingly random unpredictable behavior present in *deterministic* nonlinear dynamical systems. The unpredictability comes from small differences in initial conditions leading to vastly different trajectories due to the nonlinear response of the system.

Nonlinear dynamics

- **Nonlinear:** A nonlinear system is one that gives a response that is not directly proportional to its stimulus.
- **Dynamical:** A dynamical system is one that evolves in time.
- The MCO is a nonlinear dynamical system.

Modeling the MCO

$$I\ddot{\phi} = \sum \Gamma$$

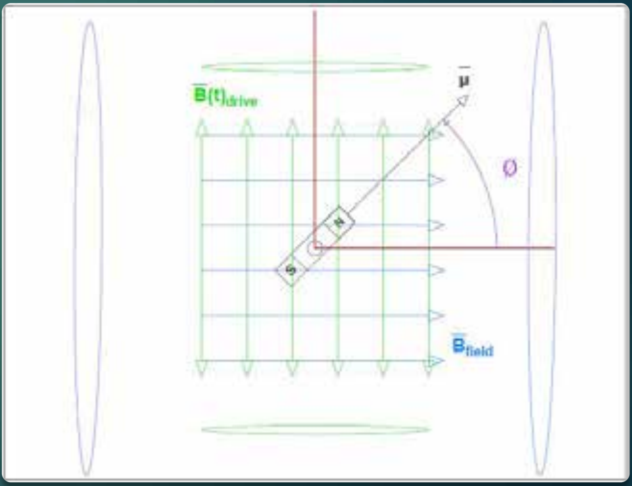
$$\Gamma_{damp} = -\beta\dot{\phi}$$

$$\Gamma_{field} = \vec{\mu} \times \vec{B}_{field} = \mu B_{field} \sin\phi$$

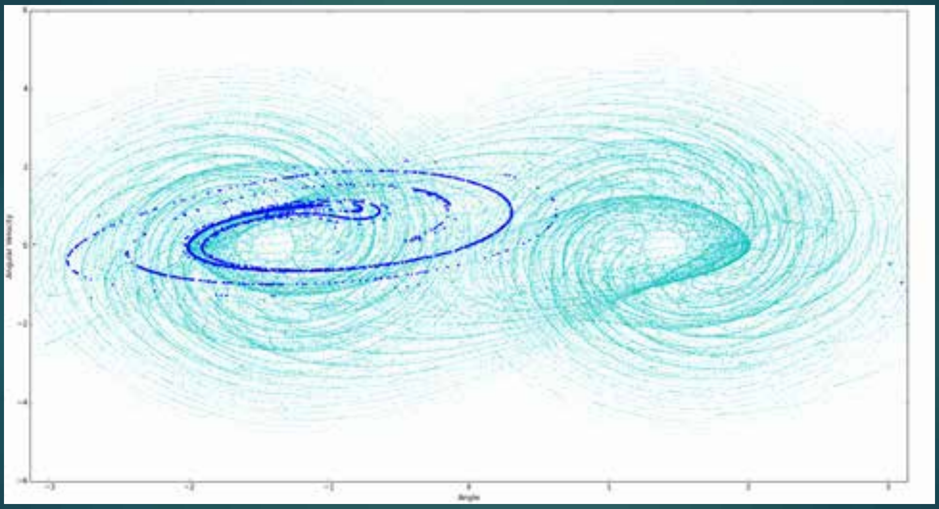
$$\Gamma_{drive} = \vec{\mu} \times \vec{B}(t)_{drive} = \mu B_{drive} \sin\omega t \sin(\frac{\pi}{2} - \phi)$$

$$= \mu B_{drive} \sin\omega t \cos\phi$$

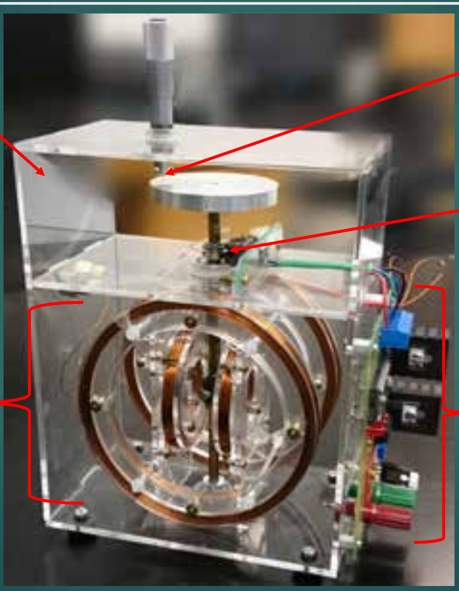
$$\ddot{\phi} = \frac{\mu}{I}[B_{field}\sin\phi + B_{drive}\cos\phi\sin\omega_d t] - \frac{\beta\dot{\phi}}{I}$$



The Phase Space Portrait



Key features of the MCO -1.0



Case and materials

Magnetic damping
(micrometer, inertial disc, and magnet)

System tracking
(Optical encoder and disk)

Helmholtz chamber
(field coil, drive coil, shaft and dipole)

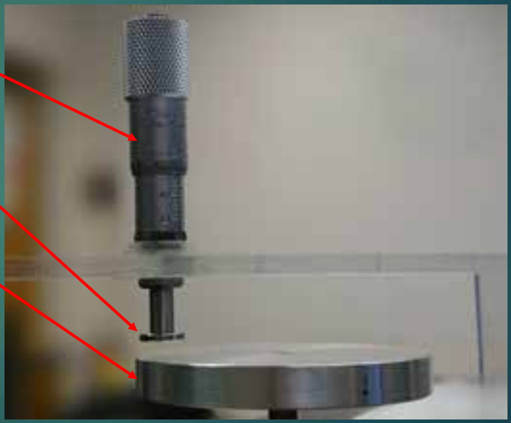
Electronics and communication
(microcontroller, push pull circuit and serial communication)

Case and materials

Non magnetic materials

- Brass and nylon fasteners
- Acrylic case and coil spools
  - Designed in AutoCad
  - Cut using laser
- Brass shaft

Magnetic Damping



Lenz's Law

$$|\mathcal{E}| = \left|\frac{d\Phi_B}{dt}\right|$$

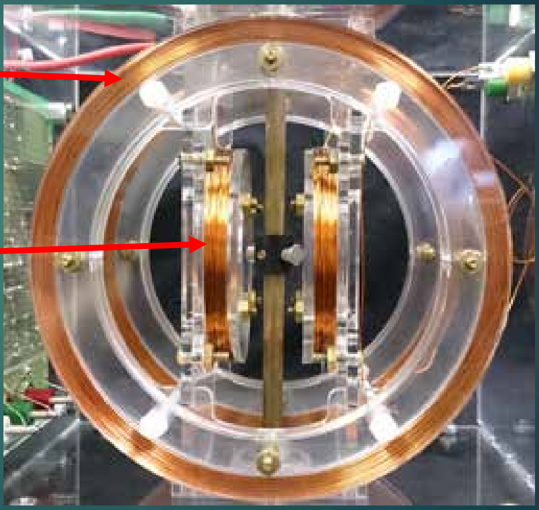
Helmholtz Chamber

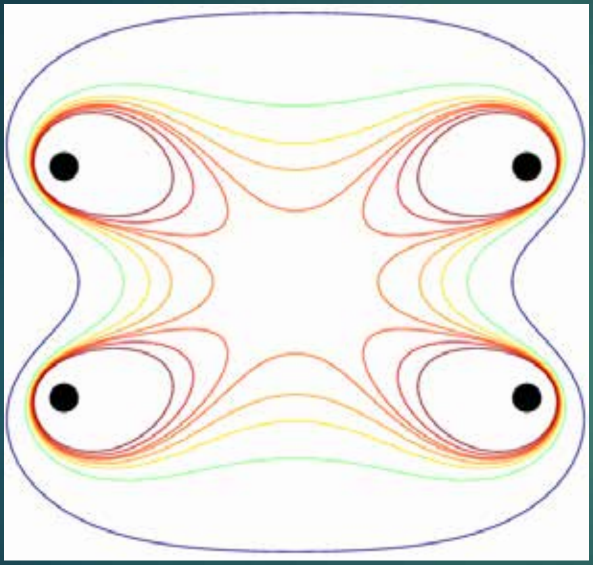
Field coils

$$B_{f_o} = \frac{8\mu n I_f}{5\sqrt{5}R_f}\hat{x}$$

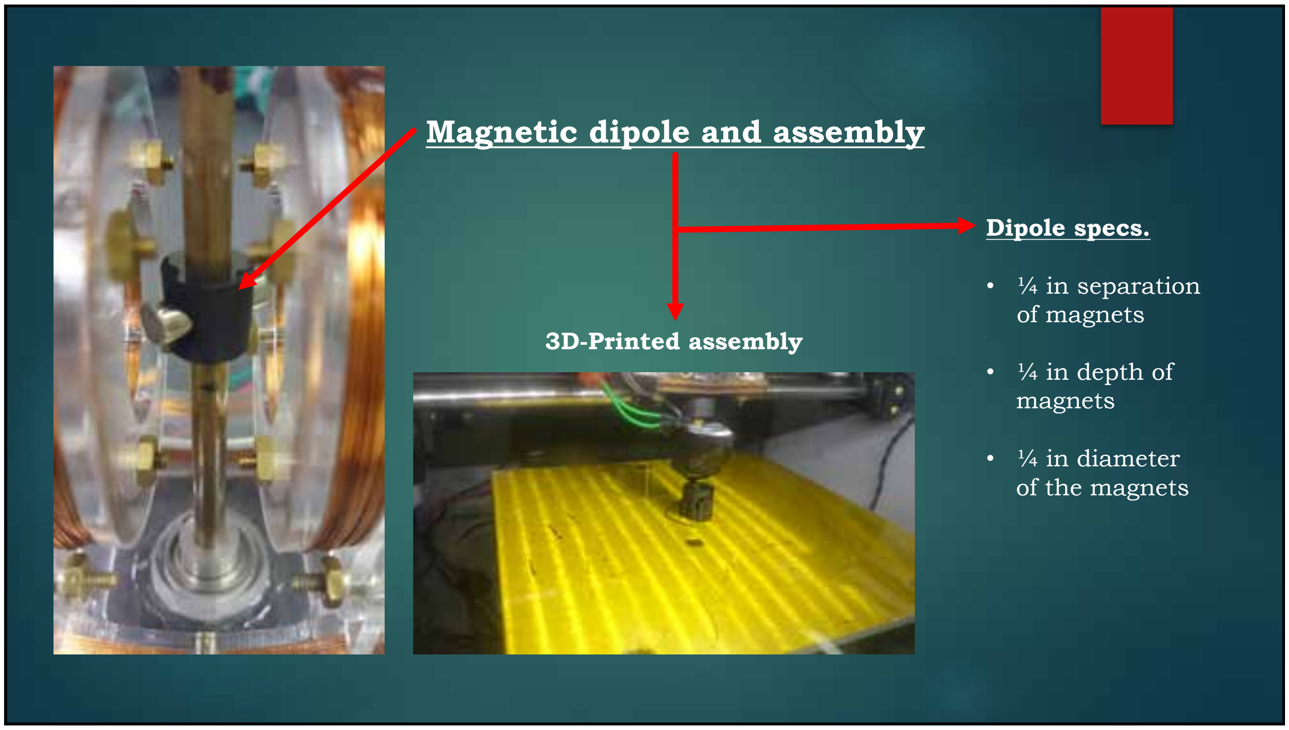
Drive coils

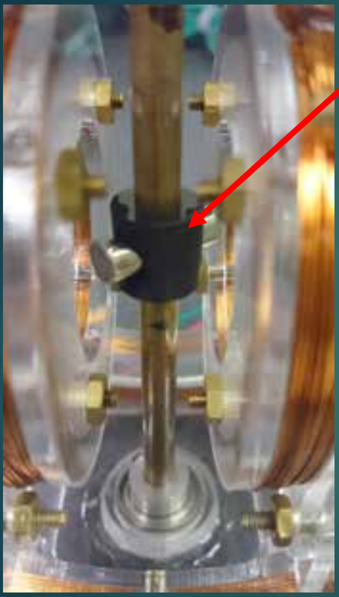
$$B_{d_o}(t) = \frac{8\mu n}{5\sqrt{5}R_d} I_d \sin(\omega_d t)\hat{y}$$





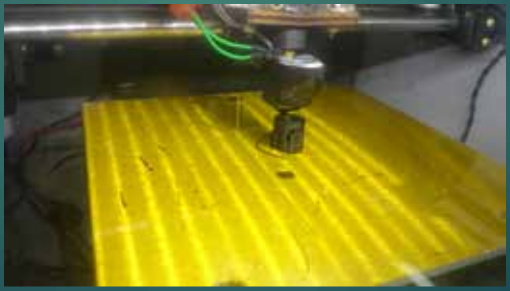
By Morn - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=32750723





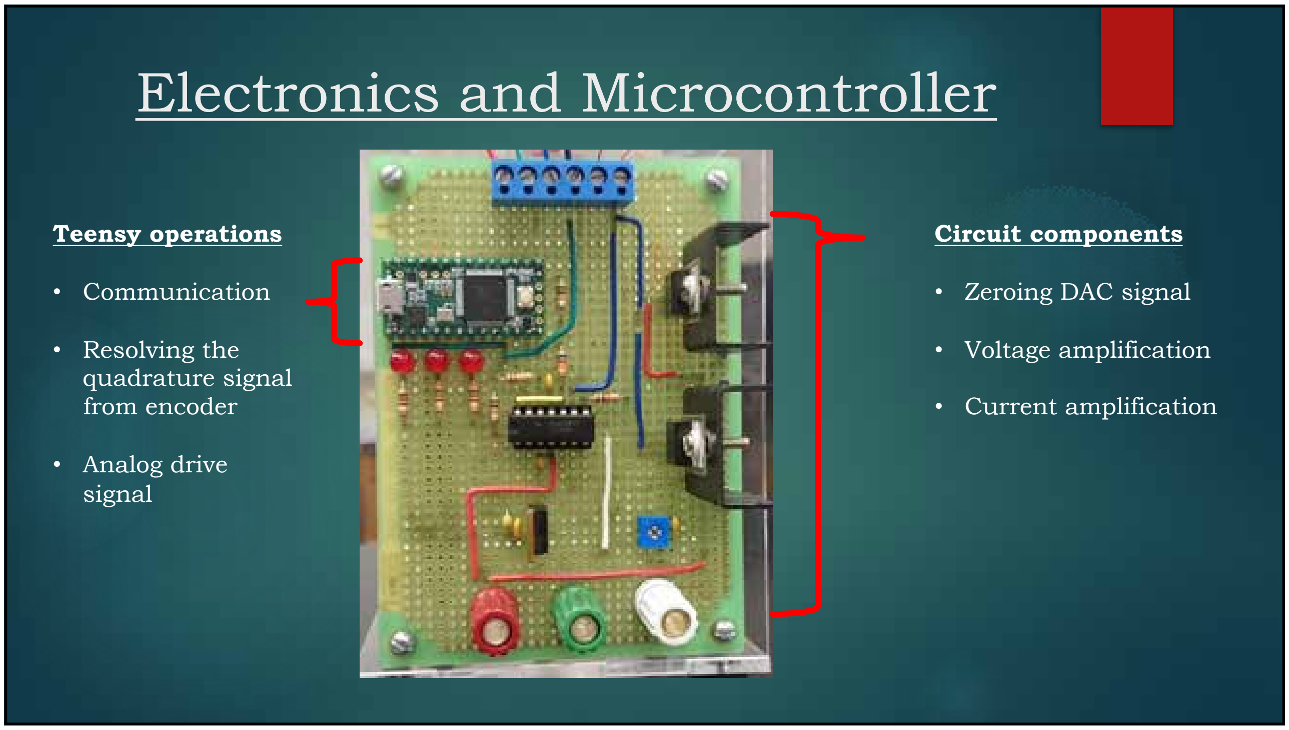
Magnetic dipole and assembly

3D-Printed assembly



Dipole specs.

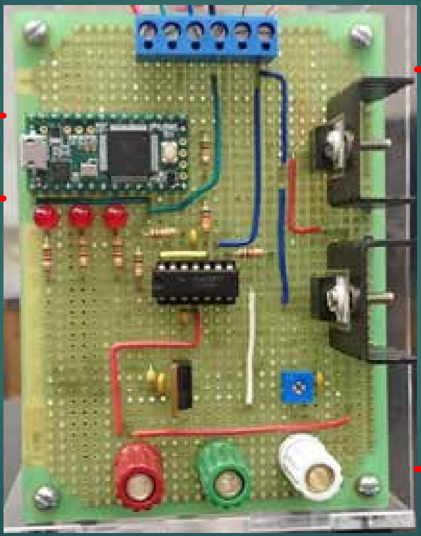
- ¼ in separation of magnets
- ¼ in depth of magnets
- ¼ in diameter of the magnets



Electronics and Microcontroller

Teensy operations

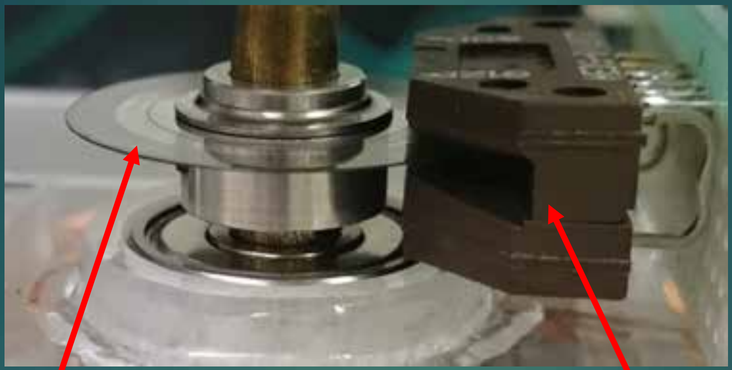
- Communication
- Resolving the quadrature signal from encoder
- Analog drive signal



Circuit components

- Zeroing DAC signal
- Voltage amplification
- Current amplification

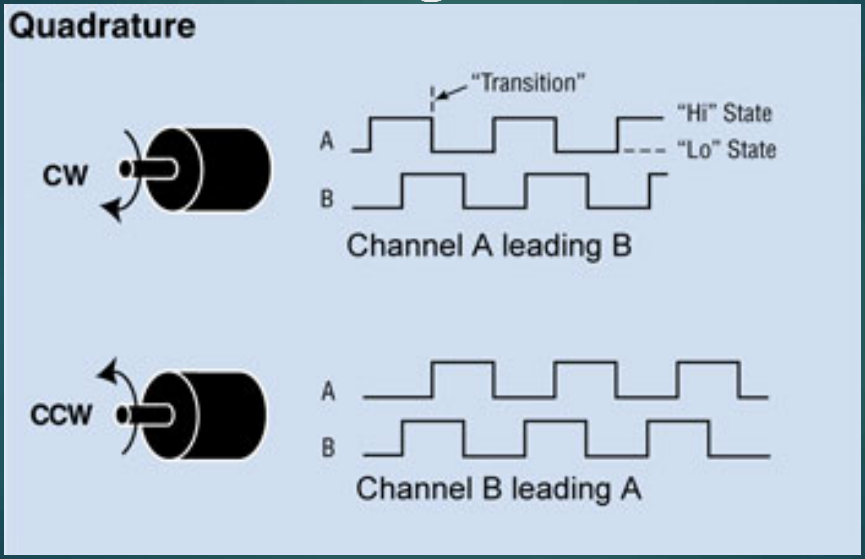
System Tracking



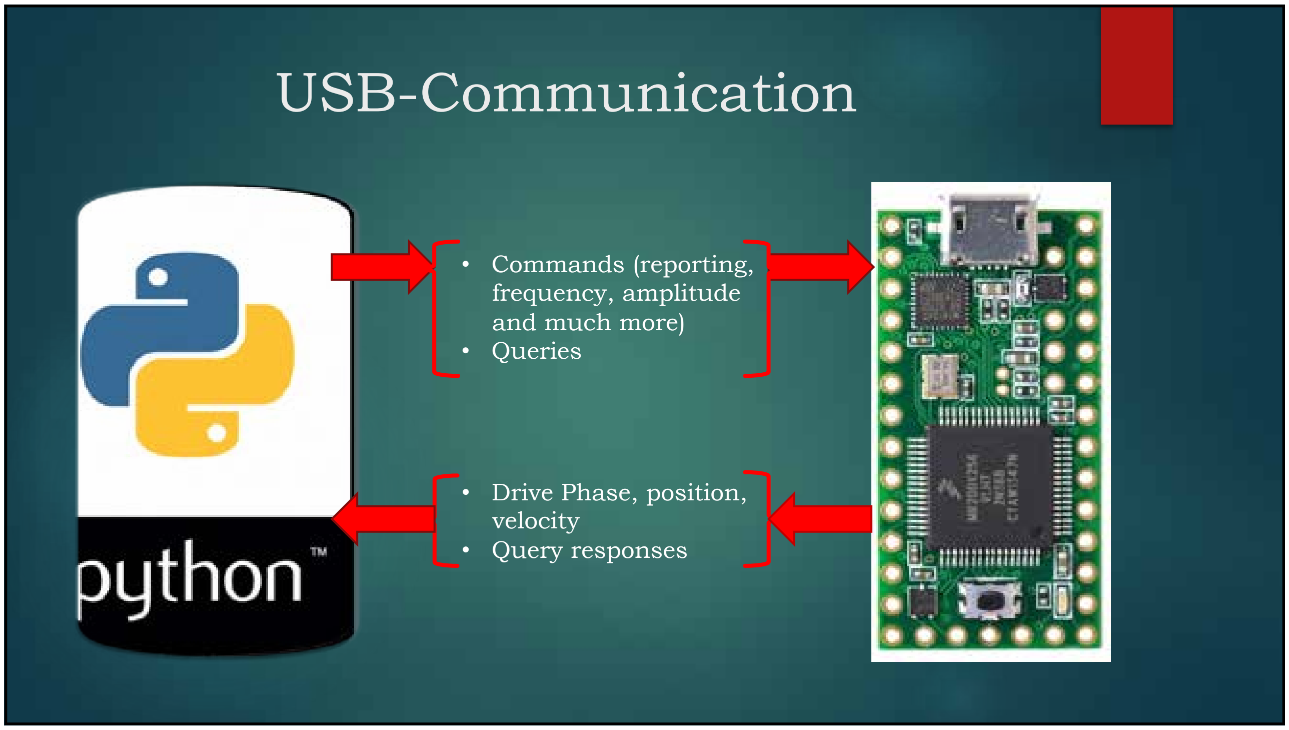
Encoder wheel
Old resolution: 360 steps
New resolution: 1024 steps

Optical encoder
Quadrature output signal

Quadrature Signal Resolution



USB-Communication



- Commands (reporting, frequency, amplitude and much more)
- Queries

- Drive Phase, position, velocity
- Query responses

Table of commands for the MCO

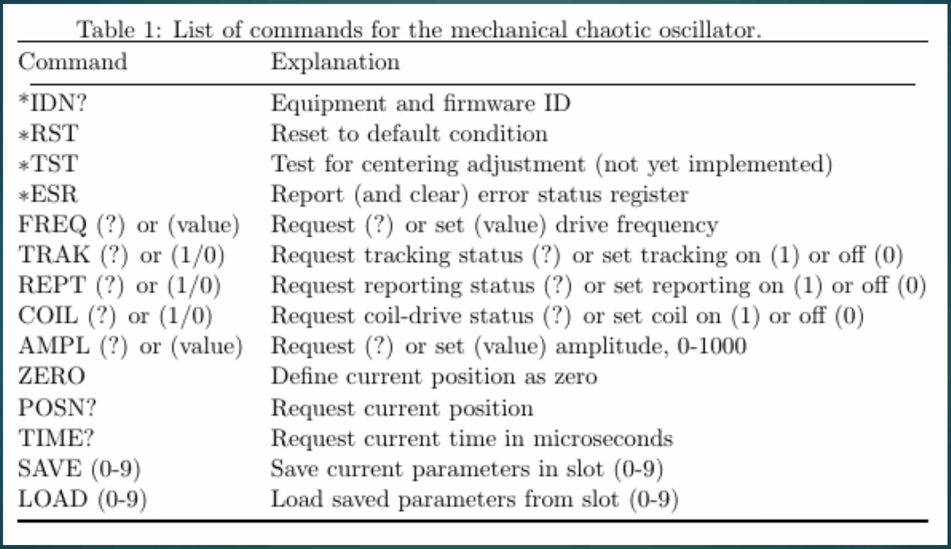
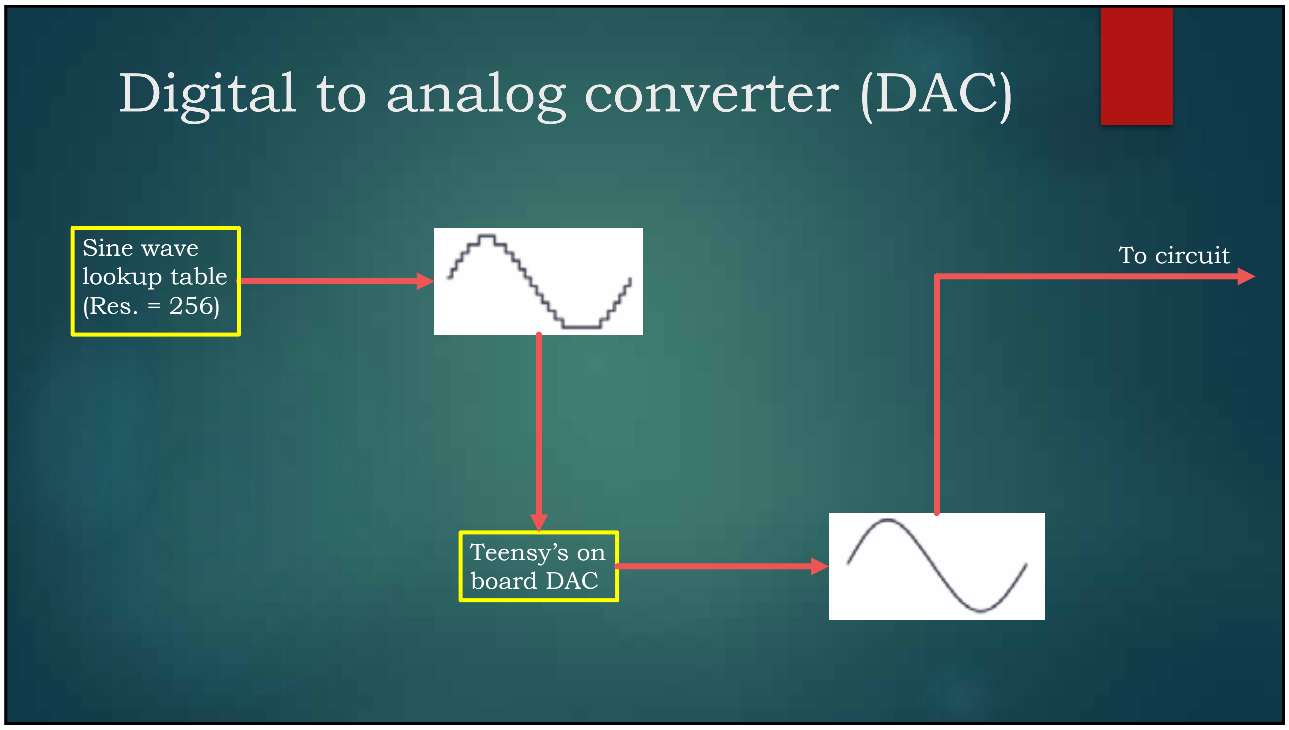


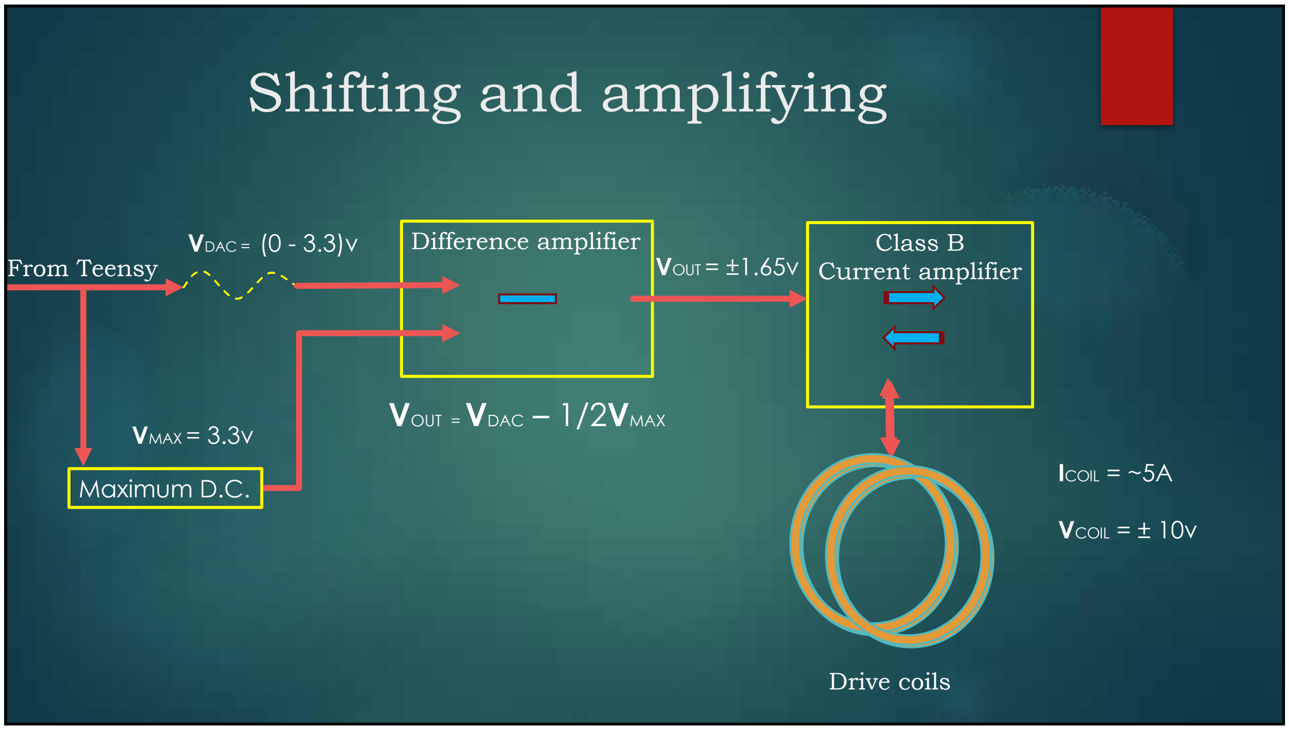
Table 1: List of commands for the mechanical chaotic oscillator.

Command	Explanation
*IDN?	Equipment and firmware ID
∗RST	Reset to default condition
∗TST	Test for centering adjustment (not yet implemented)
∗ESR	Report (and clear) error status register
FREQ (?) or (value)	Request (?) or set (value) drive frequency
TRAK (?) or (1/0)	Request tracking status (?) or set tracking on (1) or off (0)
REPT (?) or (1/0)	Request reporting status (?) or set reporting on (1) or off (0)
COIL (?) or (1/0)	Request coil-drive status (?) or set coil on (1) or off (0)
AMPL (?) or (value)	Request (?) or set (value) amplitude, 0-1000
ZERO	Define current position as zero
POSN?	Request current position
TIME?	Request current time in microseconds
SAVE (0-9)	Save current parameters in slot (0-9)
LOAD (0-9)	Load saved parameters from slot (0-9)

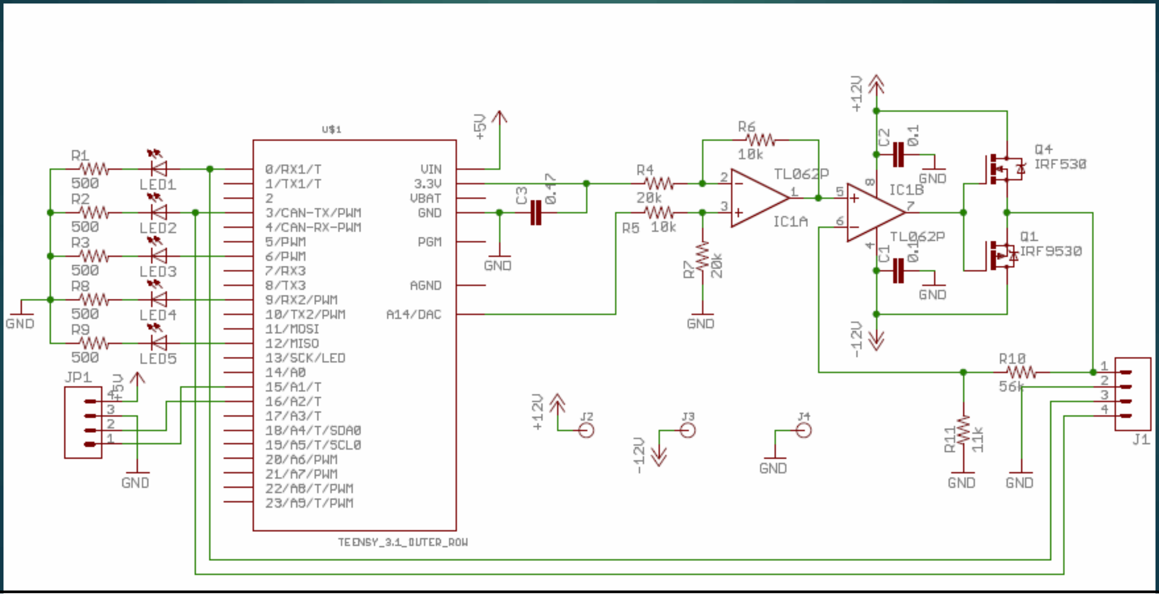
Digital to analog converter (DAC)



Shifting and amplifying

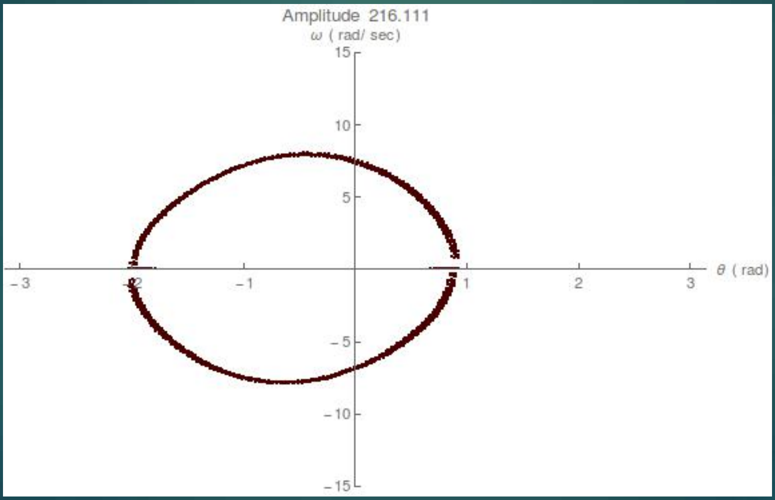


Circuit Diagram



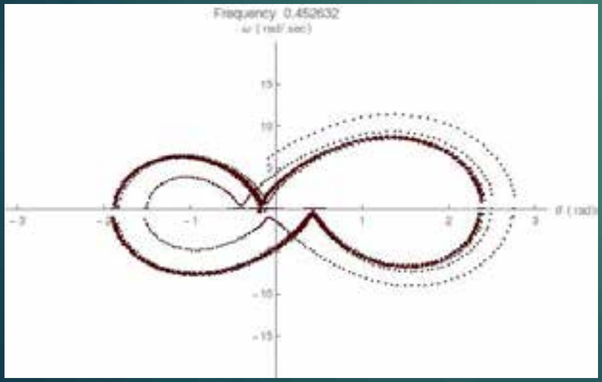
MCO (1.0) Results

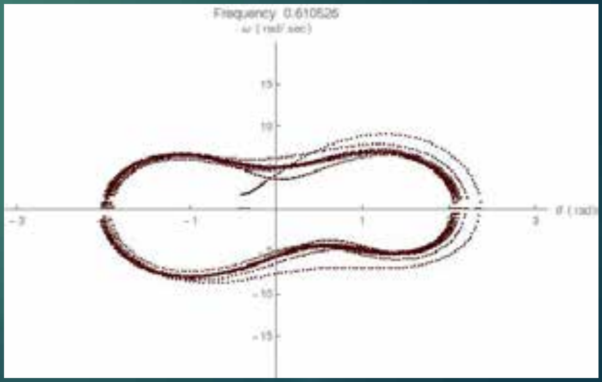
Simple harmonic motion driven at the natural frequency



Limit cycles and transient behavior

The dying out of transient motion

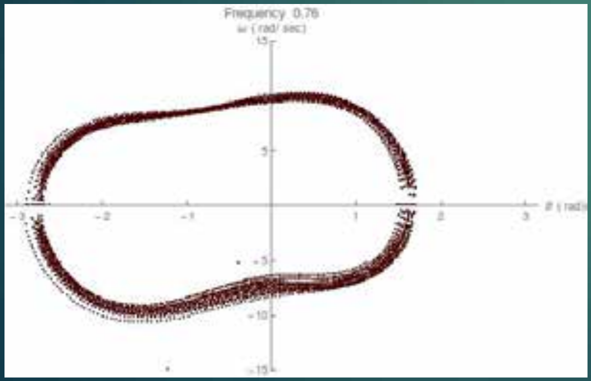


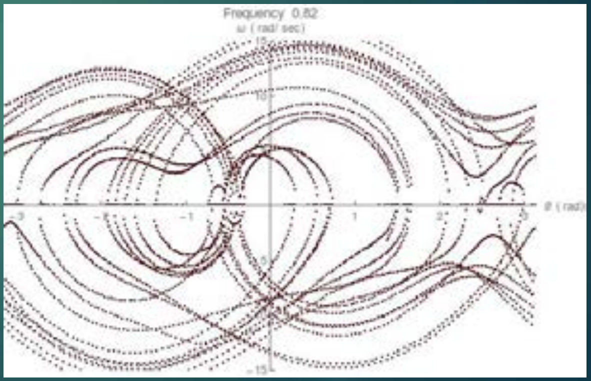


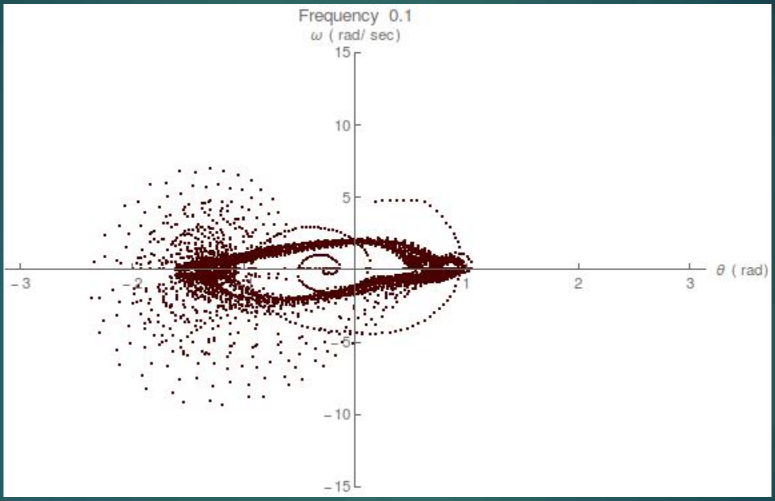
From simple motion to chaos

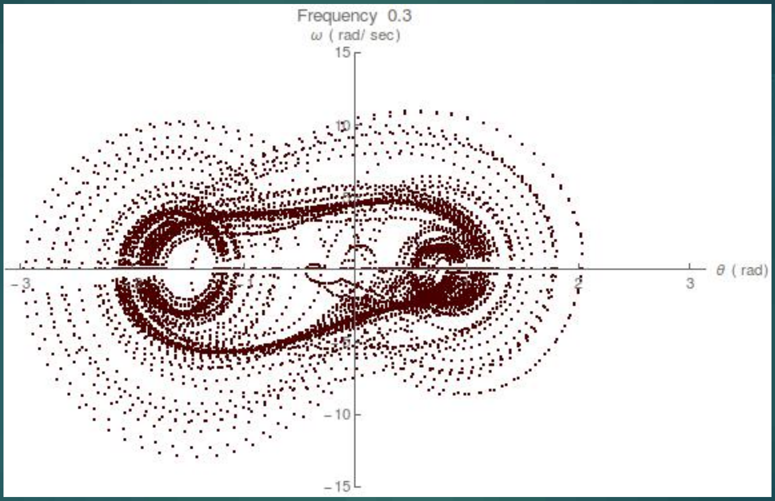
Simple motion at 0.76 (rads/sec)

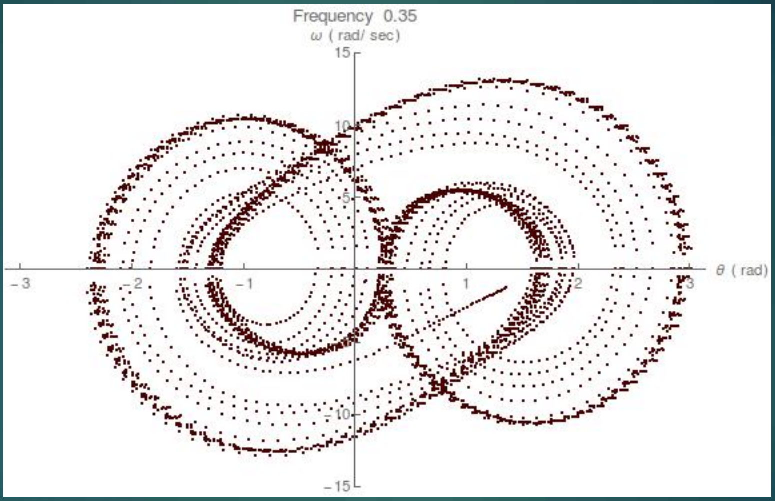
Chaos at 0.82 (rads/sec)

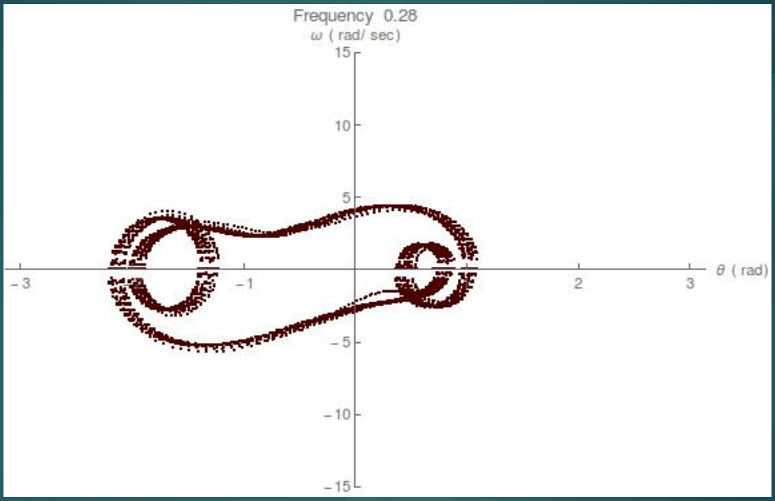


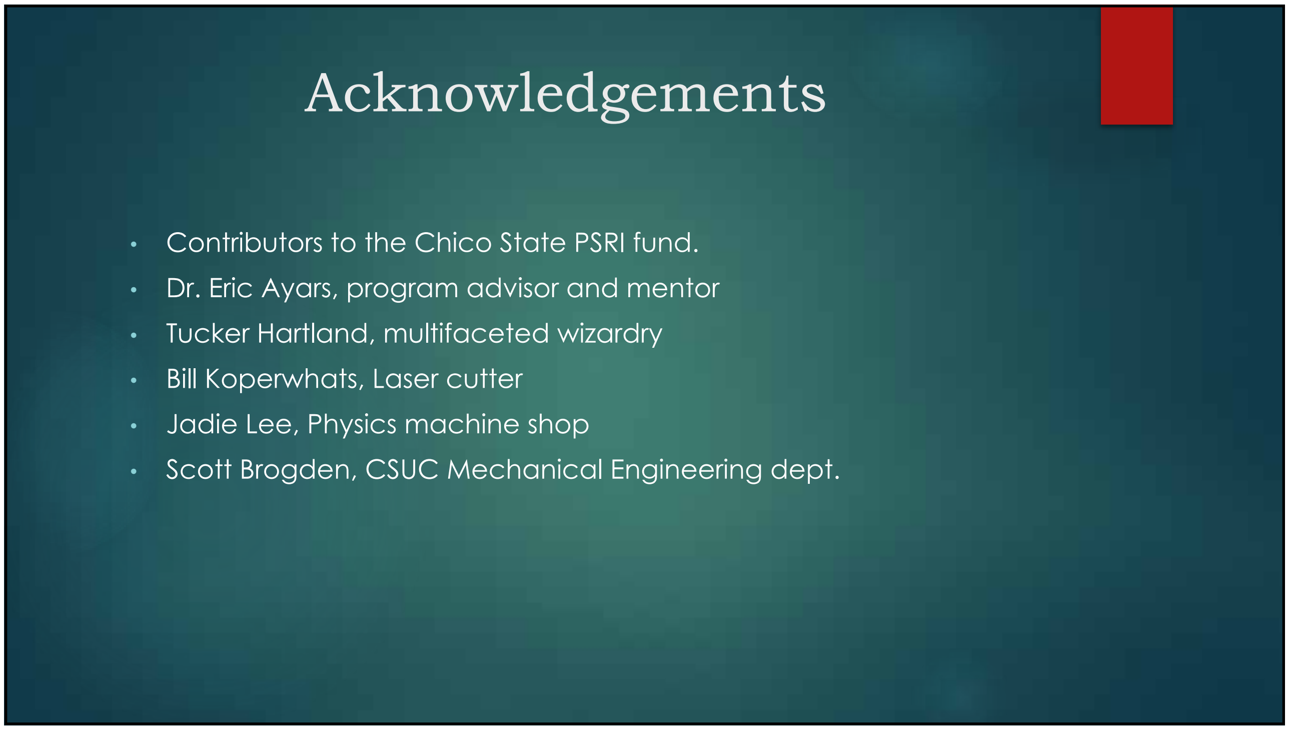



Frequency 0.1
ω (rad/ sec)
15
10
5
-5
-10
-15
-3
-2
1
2
3
θ (rad)


Frequency 0.3
ω (rad/ sec)
15
10
5
-5
-10
-15
-3
2
3
θ (rad)

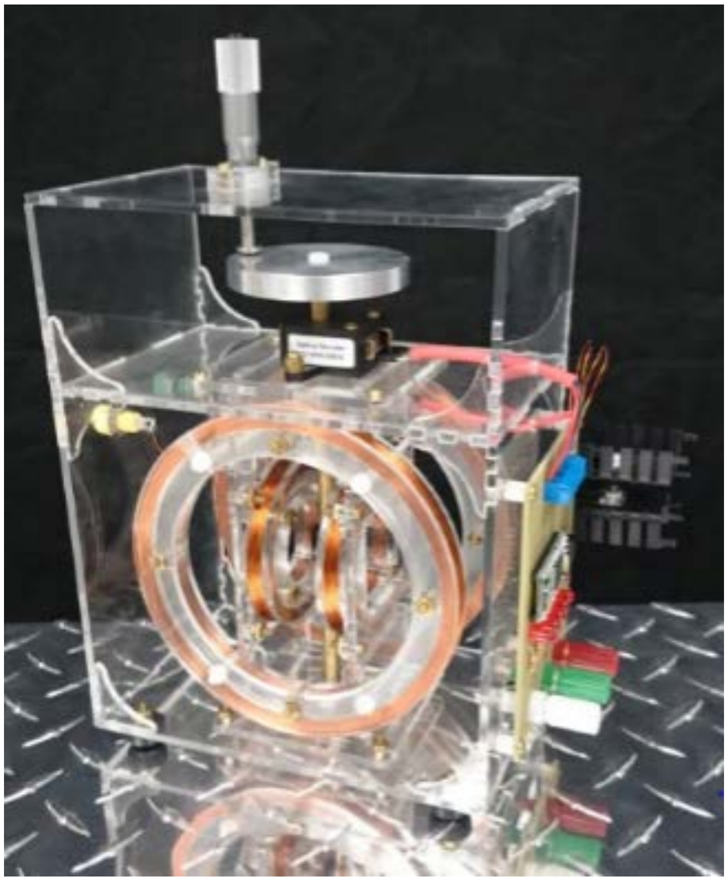
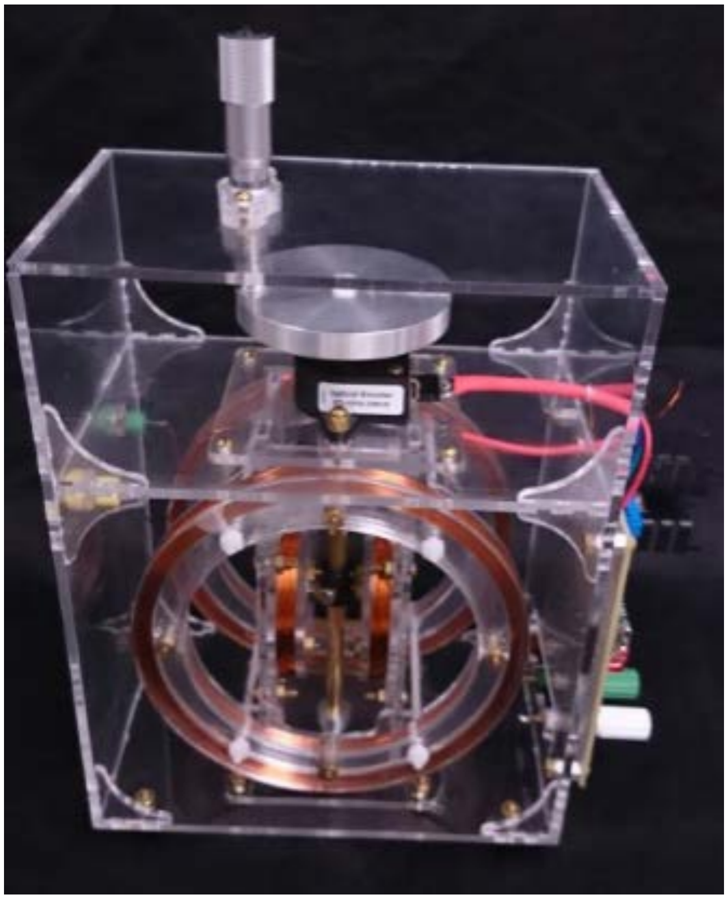
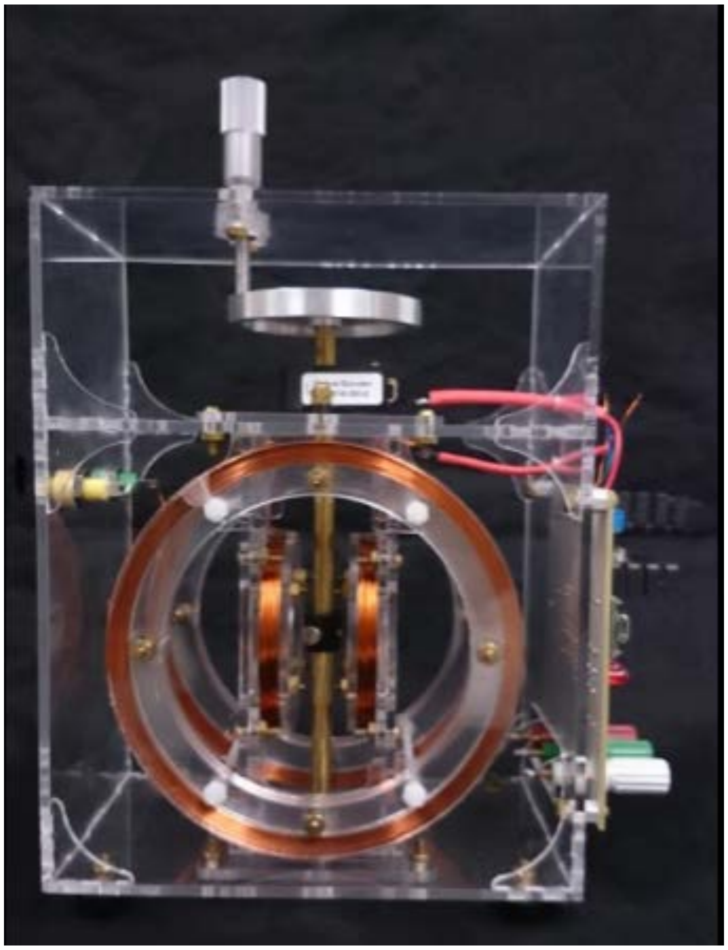
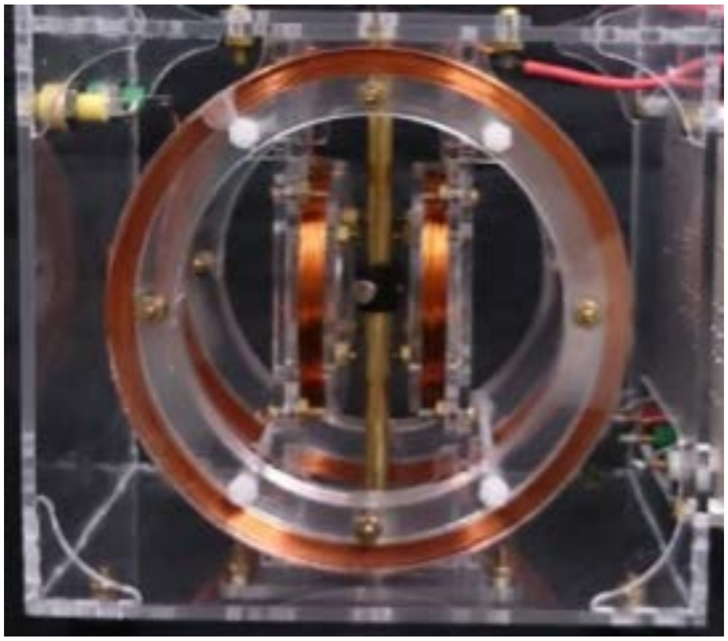

Frequency 0.35
ω (rad/ sec)
15
10
5
−10
−15
−3
−2
−1
1
2
3
θ (rad)

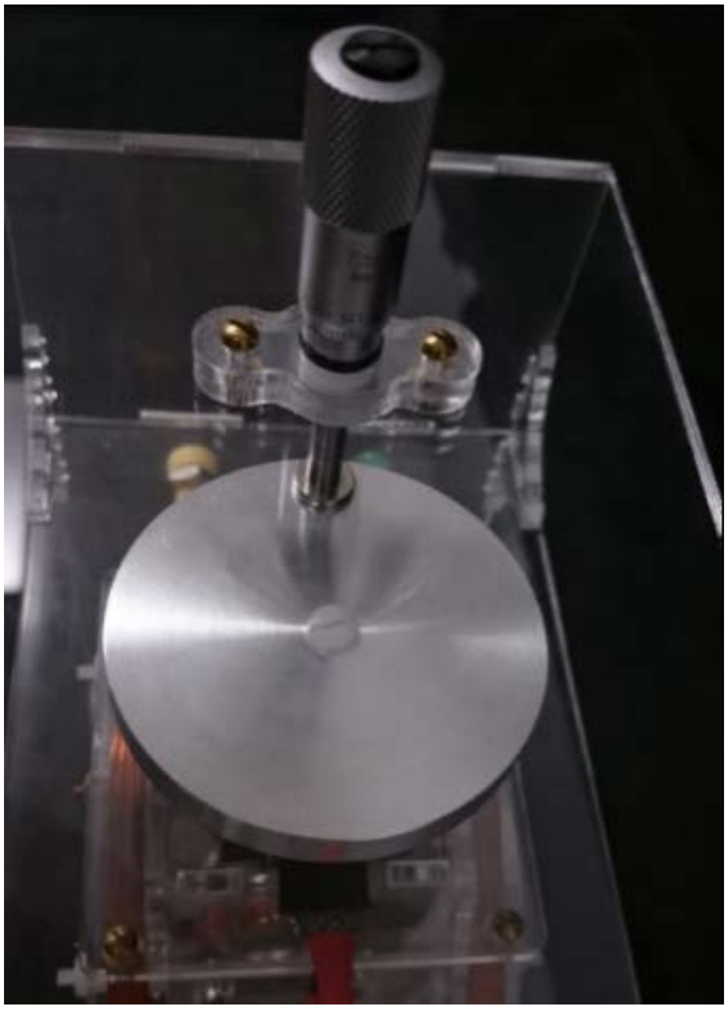

Frequency 0.28
ω (rad/ sec)
15
10
5
−5
−10
−15
−3
−1
2
3
θ (rad)

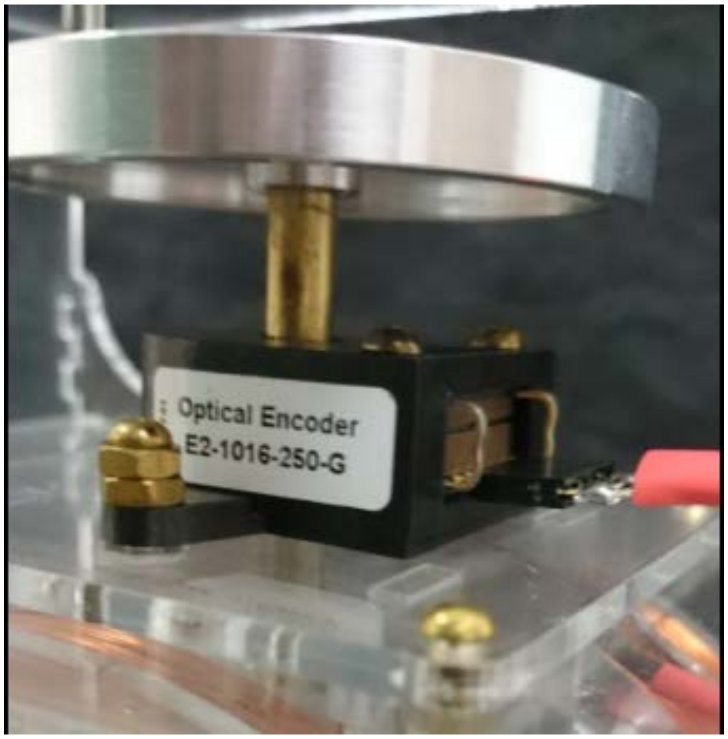


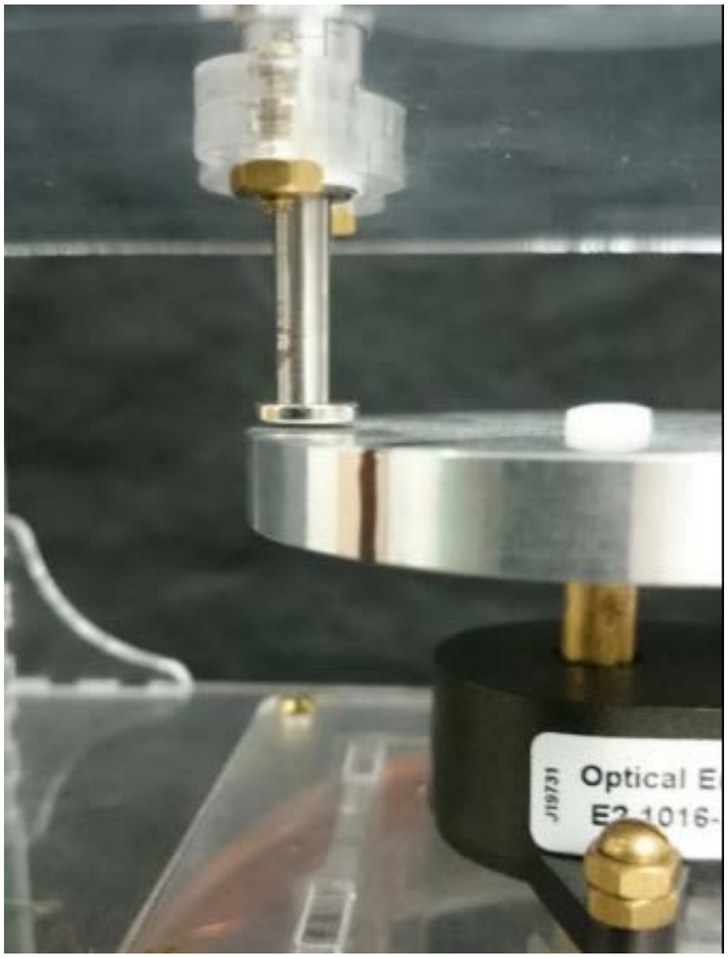
Acknowledgements

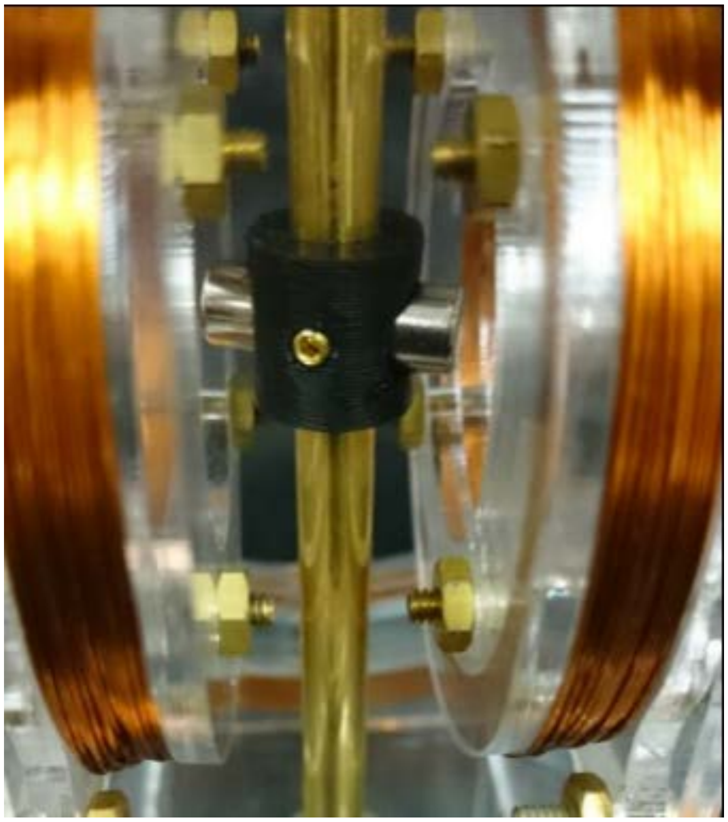
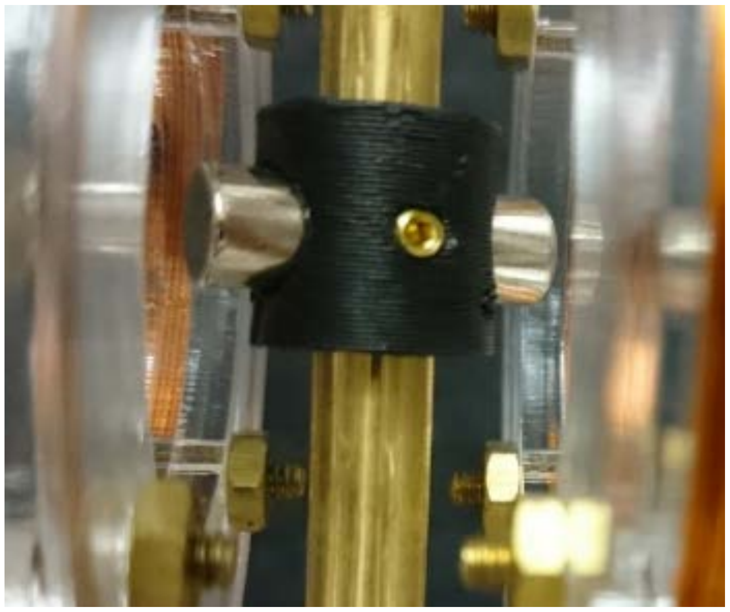
- Contributors to the Chico State PSRI fund.
- Dr. Eric Ayars, program advisor and mentor
- Tucker Hartland, multifaceted wizardry
- Bill Koperwhats, Laser cutter
- Jadie Lee, Physics machine shop
- Scott Brogden, CSUC Mechanical Engineering dept.




Optical Encoder
E2-1016-250-G


J19731
Optical E
E2-1016-

```
/*
ChaosDrive.ino
Eric Ayars
8/13/15

Drive program for the all-in-one chaos control board.
Version for Teensy 3.1 control

Differences from prior work are largely due to hardware improvements.
Use of the Teensy allows direct D/A conversion since the 3.1 includes
a real analog output. The sensor we're using is just quadrature, without
PASCO's extra board converting to step-up and step-down pulses.
(We now get to use direct quadrature signals, which is actually easier
since we can use one of the quadrature lines as an interrupt and the
other quadrature line as direction indicator for that interrupt. We lose
the 1440 resolution, since there are only 360 marks on this encoder.)
In addition, Teensy has much higher computational power, so we could
do on-board velocity calculation (we don't) and amplitude adjustment
(we do) without loss of sensitivity.

Do-list:
    x Sine output
    x frequency control
    x Drive phase zero indication via LED
    x Position zero indication via LED
    x Position reporting via serial
    x 488.2 compatibility
    x Gain control via 488.2
    x Zero via 488.2
    get multiple commands working (;)
    x save parameters in eeprom (see ~line 386)
    self-test capability (*TST)
    sweep amplitude
    sweep frequency

Versions:
    20150827: added range-checking on FREQ and AMPL
    20150817: first "working" version
*/

//#define DEBUG

String VERSION = "CSU Chico Physics Advanced Lab\nChaotic Rotor (prototype)\nfirmware
Teensy3.1_20150827";

#include <avr/io.h>
#include <avr/interrupt.h>
#include <EEPROM.h>

// Board constants
#define AOUT A14              // analog output
const byte SENSOR = 14;       // interrupt line from rotary encoder.
const byte DIRECTION = 15;    // direction indicator from rotary encoder
const byte PHASELED = 0;      // Indicator LED for drive zero-crossing
const byte ROTORLED = 3;      // Indicator LED for rotor zero-crossing
const byte EXTRALED = 6;      // Indicator LED for something TBD.
const byte TESTLED = 13;      // Indicator LED for something TBD.
const byte saveSize = sizeof(float)+sizeof(int);     // EEPROM save interval

// IEEE 488.2 variables
byte SESR = 0;                // IEEE 488.2 SESR
String inMessage = "";        // incoming text message
String outMessage = "";       // outgoing text message
String command = "";          // parsed command
boolean query = false;        // '?' in command
```

```
// Position parameters
boolean trackOn = true;     // whether to track position or not
boolean reportOn = false;   // whether or not to report position
volatile int position = 0;  // actual position
const int resolution = 360; // Depends on rotary encoder.

// Drive-related variables
const float defFreq = 1.0;  // default value
float freq = defFreq;       // variable value
boolean coilOn = false;     // whether coil drive is on or off
const int maxAmplitude = 1000;  // maximum allowed amplitude
const int defAmplitude = 500;   // default value
int amplitude = defAmplitude;
int coilV;                  // holds current output voltage
unsigned long dt;           // time step, microseconds.
unsigned long residual = 0; // residual microseconds to add in for each step
unsigned long M;            // micros
unsigned long lM;           // last micros
unsigned long dM;           // change in micros
byte phase = 0;             // Byte so it wraps correctly with 8-bit LUT.

// 12-bit Look-Up Table (LUT) for the sine wave.
int DACLookup_FullSine_8Bit[256] =
{   0,   50,  100,  150,  200,  250,  300,  350,
  399,  448,  497,  546,  594,  642,  689,  736,
  783,  829,  875,  920,  965, 1009, 1052, 1095,
 1137, 1178, 1219, 1259, 1298, 1337, 1375, 1411,
 1447, 1482, 1517, 1550, 1582, 1614, 1644, 1674,
 1702, 1729, 1756, 1781, 1805, 1828, 1850, 1871,
 1891, 1910, 1927, 1944, 1959, 1973, 1986, 1997,
 2008, 2017, 2025, 2032, 2037, 2041, 2045, 2046,
 2047, 2046, 2045, 2041, 2037, 2032, 2025, 2017,
 2008, 1997, 1986, 1973, 1959, 1944, 1927, 1910,
 1891, 1871, 1850, 1828, 1805, 1781, 1756, 1729,
 1702, 1674, 1644, 1614, 1582, 1550, 1517, 1482,
 1447, 1411, 1375, 1337, 1298, 1259, 1219, 1178,
 1137, 1095, 1052, 1009,  965,  920,  875,  829,
  783,  736,  689,  642,  594,  546,  497,  448,
  399,  350,  300,  250,  200,  150,  100,   50,
    0,  -51, -101, -151, -201, -251, -301, -351,
 -400, -449, -498, -547, -595, -643, -690, -737,
 -784, -830, -876, -921, -966,-1010,-1053,-1096,
-1138,-1179,-1220,-1260,-1299,-1338,-1376,-1412,
-1448,-1483,-1518,-1551,-1583,-1615,-1645,-1675,
-1703,-1730,-1757,-1782,-1806,-1829,-1851,-1872,
-1892,-1911,-1928,-1945,-1960,-1974,-1987,-1998,
-2009,-2018,-2026,-2033,-2038,-2042,-2046,-2047,
-2048,-2047,-2046,-2042,-2038,-2033,-2026,-2018,
-2009,-1998,-1987,-1974,-1960,-1945,-1928,-1911,
-1892,-1872,-1851,-1829,-1806,-1782,-1757,-1730,
-1703,-1675,-1645,-1615,-1583,-1551,-1518,-1483,
-1448,-1412,-1376,-1338,-1299,-1260,-1220,-1179,
-1138,-1096,-1053,-1010, -966, -921, -876, -830,
 -784, -737, -690, -643, -595, -547, -498, -449,
 -400, -351, -301, -251, -201, -151, -101,  -51,
};

/*****************************************
 *  help function                        *
 *****************************************/

void handleHELP() {
    Serial.println(VERSION);
    Serial.println();
    Serial.println("Available Commands:");
    Serial.println("  *IDN? - Equipment and firmware ID");
    Serial.println("  *RST - Reset to default condition");
    Serial.println("  *TST - Test for centering adjustment (not yet implemented)");
    Serial.println("  *ESR - Report (and clear) Error Status Register");
    Serial.println("  FREQ ?|(value) - request (?) or set (value) frequency");
    Serial.println("  TRAK ?|(0|1) - request tracking status (?) or set tracking on (1) or off
    (0)");
    Serial.println("  REPT ?|(0|1) - request reporting status (?) or set reporting on (1) or
    off (0)");
    Serial.println("  COIL ?|(0|1) - request coil-drive status (?) or set coil on (1) or off
    (0)");
    Serial.println("  AMPL ?|(value) - request (?) or set (value) amplitude, 0-1000");
    Serial.println("  ZERO - define current position as zero");
    Serial.println("  POSN? - request current position");
    Serial.println("  TIME? - request current time in microseconds");
    Serial.println("  SAVE (0-9) - save current parameters in slot 0-9");
    Serial.println("  LOAD (0-9) - load saved parameters from slot 0-9");
}

/*****************************************
 *  tracking functions                   *
 *****************************************/

void positionChange() {
    /*  This is the tracking interrupt-handler. It runs when
        quadrature line 'A' (AKA 'SENSOR') fires.
    */
    if (digitalRead(DIRECTION)) {
        position++;
    } else {
        position--;
    }
}

#ifdef DEBUG
void flash() {
    // possibly useful for debugging / program tracing
    digitalWrite(TESTLED, HIGH);
    delay(200);
    digitalWrite(TESTLED, LOW);
}
#endif

/*****************************************
 *  IEEE 488.2 functions                 *
 *****************************************/

void trimSpace() {
    // trims leading whitespace off inMessage.
    while (inMessage.length() > 0) {
        if (inMessage[0] == ' ' or inMessage[0] == '\t') {
            inMessage = inMessage.substring(1);
        } else {
            break;
        }
    }
}

void popChar(byte N) {
    // removes N characters from the front of inMessage.
    inMessage = inMessage.substring(min(inMessage.length(),N));
}

void parse() {
    // Parse device-specific commands.
    // uses (and changes) global inMessage as well as others.
    String command;
    String params;

    // Clear outMessage
    outMessage = "";

    while (inMessage.length() > 0) {

        // Determine command and parameters
        command = inMessage.substring(0,4);
        popChar(4);

        query = false;
        if (inMessage.length() > 0) {
            if (inMessage[0] == '?') {
                query = true;
                popChar(1);
            }
            trimSpace();
            int restOfCommand = inMessage.indexOf(';');
            if (restOfCommand == -1) {
                // not found, use rest of string.
                params = inMessage;
                inMessage = "";
            } else {
                params = inMessage.substring(0,restOfCommand);
                popChar(restOfCommand);
            }
        }

        // start sorting...

        if (command == "*IDN" && query) {             // *IDN?
            outMessage = VERSION;

        } else if (command == "*RST") {               // *RST
            handleRST();

        } else if (command == "*TST" && query) {     // *TST?
            handleTST();

        } else if (command == "*ESR" && query) {     // *ESR?
            handleESR();

        } else if (command == "*OPC") {               // *OPC
            handleOPC(params, query);

        } else if (command == "FREQ") {               // FREQ
            handleFREQ(params.toFloat(), query);

        } else if (command == "TRAK") {               // TRAK
            handleTRAK(params, query);

        } else if (command == "REPT") {               // REPT
            handleREPT(params, query);

        } else if (command == "COIL") {               // COIL
            handleCOIL(params, query);

        } else if (command == "AMPL") {               // AMPL
            handleAMPL(params, query);

        } else if (command == "ZERO") {               // ZERO
            position = 0;

        } else if (command == "POSN" && query) {     // POSN?
            outMessage = String(position);
        } else if (command == "TIME" && query) {    // TIME?
            outMessage = String(micros());

        } else if (command == "SAVE") {             // SAVE
            handleSAVE(params.toInt());

        } else if (command == "LOAD") {             // LOAD
            handleLOAD(params.toInt());

        } else if (command == "HELP") {             // HELP
            handleHELP();

        } else {                                    // bad command
            SESR = SESR | 32;                       // set command error bit
        }

        if (outMessage.length()>0) {                // Send requested message
            Serial.println(outMessage);
        }
    }
}

void handleRST() {                                  // *RST
    // clear position,
    // set amplitude to default
    // set phase to zero
    // turn off DAC
    // turn off position reporting
    position = 0;
    amplitude = defAmplitude;
    handleFREQ(defFreq, false);
    phase = 0;
    coilOn = false;
    reportOn = false;
    M = lM = micros();
}

void handleTST() {                                  // *TST
    // There's CURRENTLY no way this thing can self-test,
    // so report that all is good. BUT... You could run
    // the coil current monitor output to one of the AI
    // pins and have this set a current and report on the
    // result. That might be a good idea for the next
    // board version.

    /*  Run the TIP120 base signal through spare op-amp to an analog input,
        change this to check that 2048 output -> 0 V and ?? -> 3.0V.

    // For now, return 0.
    */
    outMessage = "0";
}

void handleESR() {                                  // *ESR?
    outMessage = String(SESR);
    SESR=0;
}

void handleOPC(String params, boolean query) {      // *OPC x and *OPC?
    outMessage = "Not yet implemented.";
}

void handleFREQ(float frequency, boolean query) {   // FREQ
    if (query) {
        outMessage = String(freq);
    } else {
        // Note: This will throw an error if there are non-numeric characters
        // in the remains of the command. Fix this, eventually.
        freq = frequency;
        if (freq < 0) freq = 0;
        dt = int(1000000.0/(freq*256.0));
    }
}

void handleTRAK(String params, boolean query) {     // TRAK
    if (query) {
        outMessage = trackOn? "1": "0";
    } else {
        if (params[0] == '0') {
            trackOn = false;
            detachInterrupt(SENSOR);
        } else {
            trackOn = true;
            attachInterrupt(SENSOR,positionChange,RISING);
        }
    }
}

void handleREPT(String params, boolean query) {     // REPT
    if (query) {
        outMessage = reportOn? "1": "0";
    } else {
        reportOn = (params[0]=='0')? false: true;
    }
}

void handleCOIL(String params, boolean query) {     // COIL
    if (query) {
        outMessage = coilOn? "1": "0";
    } else {
        if (params[0] == '0') {
            coilOn = false;
            analogWrite(AOUT, 2048);
        } else {
            coilOn = true;
        }
    }
}

void handleAMPL(String params, boolean query) {     // AMPL
    if (query) {
        outMessage = String(amplitude);
    } else {
        // Note: This will throw an error if there are non-numeric characters
        // in the remains of the command. Fix this, eventually.
        amplitude = params.toInt();
        if (amplitude > 1000) amplitude = 1000;
        if (amplitude < 0) amplitude = 0;
    }
}

void handleSAVE(byte slot) {                        // SAVE
    if ((slot < 0) || (slot > 9)) {
        // invalid save slot
        SESR |= (1<<5);
    } else {
        byte startPoint = slot*(sizeof(float)+sizeof(int));
        EEPROM.put(startPoint, freq);
        EEPROM.put(startPoint+sizeof(float), amplitude);
    }
}
```

```
void handleLOAD(byte slot) {                          // LOAD
    if ((slot < 0) || (slot > 9)) {
        // invalid save slot
        SESR |= (1<<5);
    } else {
        byte startPoint = slot*(sizeof(float)+sizeof(int));
        EEPROM.get(startPoint, freq);
        EEPROM.get(startPoint+sizeof(float), amplitude);
    }
}

void setup() {

    // initialize variables
    lM = micros();

    // set up LEDs
    pinMode(PHASELED, OUTPUT);
    pinMode(ROTORLED, OUTPUT);
    pinMode(EXTRALED, OUTPUT);
    pinMode(TESTLED, OUTPUT);
    pinMode(SENSOR, INPUT_PULLUP);
    pinMode(DIRECTION, INPUT_PULLUP);

    // analog resolution
    analogWriteResolution(12);

    // Retrieve saved values
    // How do I save a float in eeprom?
    // or an integer for that matter? (2 bytes?)

    // Start serial
    Serial.begin(115200);

    // make sure lights are out
    digitalWrite(PHASELED, LOW);
    digitalWrite(ROTORLED, LOW);
    digitalWrite(EXTRALED, LOW);
    digitalWrite(TESTLED, LOW);

    // counter interrupts off by default, for now.
    handleTRAK('1', false);

    // start the internal function generator, even if coil is off
    handleFREQ(freq, false);

    // set analog output to adjusted zero
    analogWrite(AOUT, 2048);

}

void loop() {

    // Check for incoming messages
    while (Serial.available() > 0) {
        char inChar = Serial.read();
        inMessage += inChar;
    }
    if (inMessage.length() > 0) {
        // New message arrived. Deal with it.

        // clean up message
        inMessage.toUpperCase();

#ifdef DEBUG
```

```
        // Verify message --- debug only!
        Serial.print("Message Received: ");
        Serial.println(inMessage);
#endif

        while (inMessage.length() > 0) {
            // clean up remaining message
            trimSpace();
            while (inMessage[0] == ';') popChar(1);
            trimSpace();

            // Parse remaining message
            // For starters, the message must be at least 5 characters.
            // Otherwise, don't bother even parsing it.
            if (inMessage.length() > 3) {
                parse();                            // uses global inMessage
            } else {                                // bad message length
                SESR = SESR | 32;                   // set command error bit
                inMessage = "";
            }
        }
    }

    // Now check the time and adjust output accordingly
    M = micros();
    dM = M - lM + residual;
    if (dM > dt) {
        // we just crossed a drive cycle time step, so...

        // adjust output.
        phase = phase + dM/dt;      // step forward requisite number of steps.
        if (coilOn) {
            coilV = DACLookup_FullSine_8Bit[phase] * amplitude / maxAmplitude + 2048;
            analogWrite(AOUT, coilV);
        }
        residual = dM % dt;         // save "unused" microseconds for next dM.
        lM = M;

        // Check how position is doing
        if (trackOn) {
            if (position > (resolution - 1)) position -= resolution;// wrap backwards
            if (position < 0) position += resolution;               // wrap forwards
        }

        if (reportOn) {
            // Send information to the computer
            Serial.print(M, DEC);           // microseconds
            Serial.print(" ");
            Serial.print(phase, DEC);       // phase angle (0-255)
            Serial.print(" ");
            Serial.println(position, DEC);  // position (in units of resolution)
            //Serial.print(" ");
            //Serial.println(T, DEC);           // Drive period (ms)
        }

        // Update position indicator light
        if (position==0) {
            digitalWrite(ROTORLED, HIGH);
        } else {
            digitalWrite(ROTORLED, LOW);
        }

        // Update phase indicator light
        if (phase==0) {
            digitalWrite(PHASELED, HIGH);
        } else {
            digitalWrite(PHASELED, LOW);
        }
    }

    // Show Error status (for debugging... may be useful overall though!
    digitalWrite(EXTRALED, SESR);
}
```