Wesleyan University Physics Department

Development of Raspberry Pi Magnetic Feedback Control System to Explore the Dynamics of a Parity Time-Symmetric Two-Dipole Model

by

Ekram Towsif

Class of 2021

An honors thesis submitted to the
faculty of Wesleyan University
in partial fulfillment of the requirements for the
Degree of Bachelor of Arts
with Departmental Honors in Physics

Middletown, Connecticut April, 2021

Dedication

I am dedicating this thesis to four beloved people who mean so much to me; my father, mother, brother, and sister.

Acknowledgements

This project would not have been possible without the support of many people. First, many thanks to my adviser, Fred M. Ellis, who took me into the lab as a freshman and steered me in the right direction during each obstacle I faced. Thank you for teaching me how to be a researcher and helping me make some sense of complicated geometrical relationships in magnetic systems.

Second, I would like to thank the McNair Scholars Program's directors, Ronnie Hendrix and Professor Erick A. Taylor, for all their support and funding, especially during the Covid-19 pandemic. They have supported my project since my sophomore year. Many aspects of the experimental results for the experiment would not have been possible without them.

I would also like to thank all the professors in the physics department who have taught me so much in the last four years. Finally, thanks to the numerous friends who always offered support and love throughout this long process. Thank you all for listening to my poster presentations and giving me feedback.

Abstract

Parity-Time (PT) symmetry describes systems that are balanced with their environments, and the flow of energy is dictated by the geometric relationship of differential equations. These equations of motion allow for gain-loss parameters to interact within systems. Previous work on PT symmetry includes PT-lasers, PT-superconducting wires, and PT-nuclear magnetic resonance. However, the behavior of large-scale magnetic PT-symmetry is unknown. Here we address the large-scale equations of motions corresponding to a coupled magnetic system in a background field analogous to a simplification of PT NMR. The numerical equations of motion were solved, leading to computational and experimental exploration of the system. Loss in a system can be implemented by geometrically placing a piece of aluminum near the dipole causing magnetic drag induced by eddy currents. To establish gain, we implemented a feedback loop system where the frequency is picked up by the pick-up coil and amplified by a Pre-Amp to the drive coil, which makes the dipole spin faster. Due to mutual inductance, the coils were restricted geometrically around the dipole such that the mutual inductance was zero. To drive the dipole at its resonant frequency, we used a Raspberry Pi module with a HiFi Berry DAC/ADC sound card to filter electromagnetic waves in python code to control the phase shifts of the system. We were finally able to devolve a code that filters the waves continuously without any errors, and we hope to get experimental results to confirm the simulations we have done. We anticipate our study to indicate the behavior of magnetic two and one-way signal transmissions in hopes of controlling electromagnetic waves. The addition of multi-coupled magnetic systems can give further insight into electromagnetic systems. Furthermore, the application of magnetic fields in the quantum regime of photons and electrons will be relevant for further development. [2,8,11,12]

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

In 2010, a group collaboration between professor Ellis and professor Kottos started leading to the study of Non-Hermitian Hamiltonians and Parity-Time (PT) symmetry in different systems. The Non-Hermiticity of these PT-symmetric systems is dependant on the magnitude of a γ parameter. The γ parameter introduces a new degree of freedom that is tunable and controls the dynamical behavior energy in a system. Values of γ below the γ_{PT} result in the real solutions for dynamics of the system. This γ region is known as the exact PT-phase region. If the values of γ are above γ_{PT}, then the solutions to the system's dynamics will be complex. This condition is known as the broken phase region. In our lab, many symmetric systems have been examined by previous research students in an electronic framework. However, my project attempts to shed light on the potential applications of implementing PT-symmetry in a magnetic framework. As well as the integration of electronics with a PT-symmetric magnetic system. [15]

1.2 Overview

Just as symmetry considerations have guided the development of many physics fields, I am exploring the effects of PT-symmetry on various magnetic systems. PT-symmetry describes

a system's balance with its environment, where the geometric relationship between each element in the system dictates the flow of energy, including the gain and loss parameter. The system I am studying is a coupled two-dipole system where gain and loss are time-reversible while the left and right dipoles are parity reversible. Time reversible refers to the behavior of the system based on the flow of time. The dynamics of the two-dipole system get modeled by the differential equations of motion, which will remain well-defined regardless of the directional flow of time. Parity reversible is a flip in the signs of the terms of the differential equations, either from positive to negative or vice versa, which results in a mirror image of the system. The two dipoles can oscillate freely in the X-Y plane. Gain or loss gets implemented onto the dipoles by taking energy away from one dipole via damping its oscillations while supplying energy to the other dipole such that its oscillatory behavior is either sustained or increased. Figure 1.1 shows an example of the system we are exploring. In the two-dipole system reversing time changes the flow of gain and loss. The dipole that had loss implemented will now have gain, and the dipole with gain now has a loss. Additionally, flipping the signs of the spatial coordinates of the system will switch the position of the dipoles. Using the time or parity operator independently will change the system, but applying both operations gives back an identical system. [2,15]

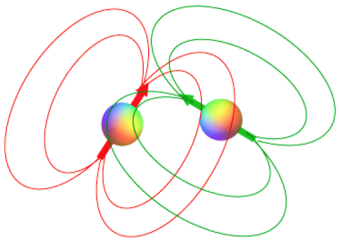

Figure 1.1: The image is an example of a two-dipole pair coupled to one another. Energy gets supplied to the dipole with green magnetic field lines such that it oscillates at its resonant frequency. The second dipole with red magnetic field lines is analogous to a dipole which is damped such that its oscillations slow down. The two dipoles communicate with each other through their magnetic field lines resulting in a coupling behavior. The distance is what determines the strength of the coupling between the two dipoles. [1]

Ultimately, the identifications of new behaviors in this two-dipole system can potentially have applications in Nuclear Magnetic Resonance (NMR) Spectroscopy, signal transfer technology, hypersensitive sensors, nonlinear motors, etc. The aim is to use PT-symmetry to improve the sensitivity of NMRs by converting them into PT-NMRs. The principles of PT-symmetry

have been applied successfully to electrical circuits, which have enabled more sensitive detection circuits. Optical experiments and other theoretical investigations have led to the development of unidirectional and multi-directional optical valves. Complete control of the flow of energy in any system can result in multiple novel applications. However, before we can have total control over a system, there are design and engineering challenges that need to get resolved. Such challenges include implementing, controlling, and sustaining the gain and loss of energy for a given system. In the PT-symmetric dipole system shown in figure 1.1, implementing energy gain and loss is relatively easy computationally, but experimentally it becomes a challenge. Before going into the computational and experimental methods used to analyze the two-dipole system of interest, let's examine the simple case of implementing gain and loss to a harmonic oscillator using a feedback loop system. [2,14]

1.3 Gain/Loss of Simple Harmonic Oscillator

The most common example of a simple harmonic oscillator is a mass on a spring, as shown in figure 1.2. The one-dimensional mass on a spring has a linear restoring force that obeys Hooke's Law ($F = -kx$). From Hooke's Law, we can see that the restoring force is proportional to the displacement of the mass from the equilibrium position. The equation of motion for the simple harmonic oscillator can is obtainable with Newton's Second Law ($F = ma$) and Hooke's Law. Rewriting the acceleration in terms of the second derivative of position and setting the two forces equal to each other produces the second-order linear ordinary differential equation that describes the dynamics of the harmonic oscillator shown in equation 1.1. [14]

$$F = m\frac{d^2x}{dt^2} = -kx \tag{1.1}$$

Solving the differential equation above results in a sinusoidal solution with respect to time for the position $x(t) = Acos(\omega t - \phi)$, velocity $v(t) = -A\omega sin(\omega t - \phi)$, and acceleration $a(t) = -A\omega^2 cos(\omega t - \phi)$. The ϕ in the solutions to the differential equation indicates a horizontal translation of the sine or cosine function left or right along the x-axis, which is also referred to as a phase shift. Additionally, the square proportionality of the spring constant (k) and the mass of the object attached produce the angular frequency of oscillation of the system. Thus, the harmonic oscillator has a angular frequency of $\omega = \sqrt{\frac{k}{m}}$. From this the period of the harmonic

oscillator can be obtained since $\omega = 2\pi f$ and $T = \frac{1}{f}$, which means that $T = \frac{2\pi}{\omega} = 2\pi\sqrt{\frac{m}{k}}$. [14]

Now the motion of the ideal simple harmonic oscillator, represented in equation 1.1, is understood. Further parameters such as dampening or driving the system can get introduced. In the dampening case, the energy of the system is dissipated into the environment continuously. Generally, frictional forces dissipate the energy in the simple harmonic oscillator. There are many ways of introducing dampening in the system, such as considering a larger air resistance, friction on the floor the mass is sliding on, or even changing the surrounding medium the mass and spring are from air to a viscous fluid. Either way, the result is the same; energy dissipates. The dampening force is proportional to the velocity and is negative since it acts against the direction of motion. Thus, the new equation of the system becomes $F = ma = -bv - kx$. The solution to this differential equation is still a sinusoidal wave, but this time it oscillates in the envelope of an exponentially decaying function $x(t) = Ae^{-\alpha t}cos(\omega t - \phi)$ where $\alpha = \frac{b}{2m}$. The solution also has an angular frequency shifted from the natural angular frequency by α^2. Therefore, $\omega = \sqrt{\omega_0^2 - \frac{b}{2m}^2}$ where $\omega_0 = \sqrt{\frac{k}{m}}$. Based on the different values of b, the oscillatory response of the system will differ. When $b = \sqrt{4mk}$ than $\omega = 0$ which indicates that for any value of $b < \sqrt{4mk}$ the system will be underdamped, for $b = \sqrt{4mk}$ the system is critically damped, and for $b > \sqrt{4mk}$ the system is overdamped. [14]

Since we understand the dynamical response of dampening a simple harmonic oscillator, let us consider the oscillator with an additional driving force. For this situation, we can model the system as shown in figure 1.2. [14]

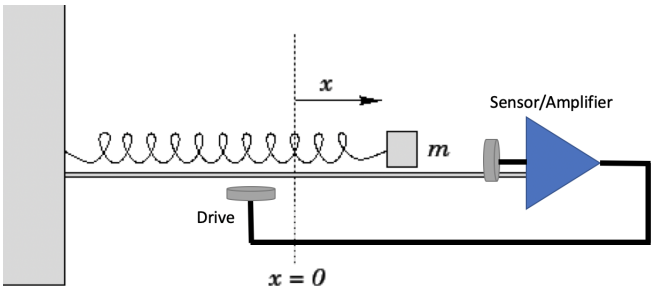

Figure 1.2: Image of a mass on spring with a controllable feedback loop. The feedback loop has sensors that measure the displacement, a pre-amp that amplifies the signal, and a driver that applies a force to the mass/spring system proportional to the measured signal. The applied force affects the motion of the entire system based on the complex phase of the applied force. The amplifier magnifies the effects of the complex phase on the system. [14]

Let's assume, for the sensor and amplifier set up, that the feedback loop applies a force at a specific angular frequency independent of the angular frequency of the harmonic oscillator. The frequency of the driving force wave will also have a phase and amplitude associated with it. Thus, the driving force gets simplified into the form of $Acos(\omega t + \phi)$ which will be different from the solution to the harmonic oscillator with no dampening or driving force, which means that we can alter equation 1.1 such that it looks like equation 1.2. [14]

$$F = m\frac{d^2x}{dt^2} + kx = A_d cos(\omega_d t + \phi_d) \tag{1.2}$$

The subscript d represents the amplitude, frequency, and phase of the driving force. The solution to the differential equation with no dampening in the system would be in the form of $x(t) = Asin(\omega t + \phi) + A_d cos(\omega_d t + \phi_d)$. The first term of the solution is the transient solution which is determined by the initial conditions of the system. The second term of the solution corresponds to the steady-state solution, which is determined by the driving force. If the harmonic oscillator gets damped, the bv term would be reintroduced into equation 1.2, and the transient solution would have an exponential function that would determine the envelope in which the sinusoidal solution to the system would oscillate inside. From the solution x(t), we can see that if in steady-state solution is in phase with the transient solution, then there will be energy added to the system proportional to the amplitude of the steady-state driving wave. However, if the phase of the steady-state solution is out of phase with the transient solution,

then there will be energy loss in the system proportional to the amplitude of the steady-state driving wave. Therefore, based on the phase, amplitude, and frequency of the driving force, full control over the motion of the simple harmonic oscillator can be achieved through the feedback loop. The feedback loop can theoretically be attached to a digital or analog amplifier, phase shifter, and frequency filter in-between the sensor and driver to gain control over each parameter to a specific degree of accuracy. [14]

Let's examine, in more detail, how the simple harmonic oscillator interacts with the feedback loop without external dampening. The feedback loop senses the motion of the oscillator, amplifies it, and sends it back to the oscillator, which directly influences the dynamics of the system. Specifically, the sensor measures the change in position for the mass (x); the displacement. The amplifier multiplies the measurement by a complex exponential in the form of $Ge^{i\phi}$. Thus, the drive will apply a force onto the mass and spring proportional to $Ge^{i\phi} * x$. The response in the system will depend on the amplitude G and phase ϕ that multiplies the input signal of the displacement of the harmonic oscillator. Rewriting Newton's equation with the new feedback gain term and dampening should give $F = ma = -kx + Ge^{i\phi}x$. Solving the force equation with an ansatz of $x = Ae^{i\omega t}$, Newton's equation becomes $-m\omega^2 A = -kA + Ge^{i\phi}A$. Solving for the frequency of the system, $\omega = \sqrt{\frac{k}{m} - \frac{G}{m}e^{i\phi}}$. Since the natural frequency (ω_0) of the harmonic oscillator is know, it can be factored out such that $\omega = \omega_0\sqrt{1 - \gamma e^{i\phi}}$, where $\gamma = \frac{G}{k}$. The frequency shows that the feedback loop multiplies the natural frequency of the harmonic oscillator by some scalar relative to the complex frequency of the system. Therefore, the phase that the input signal gets multiplied by determines how the feedback loop affects the system, as shown in figure 1.3. Controlling the γ parameter will amplify the effects of the phase on the system.

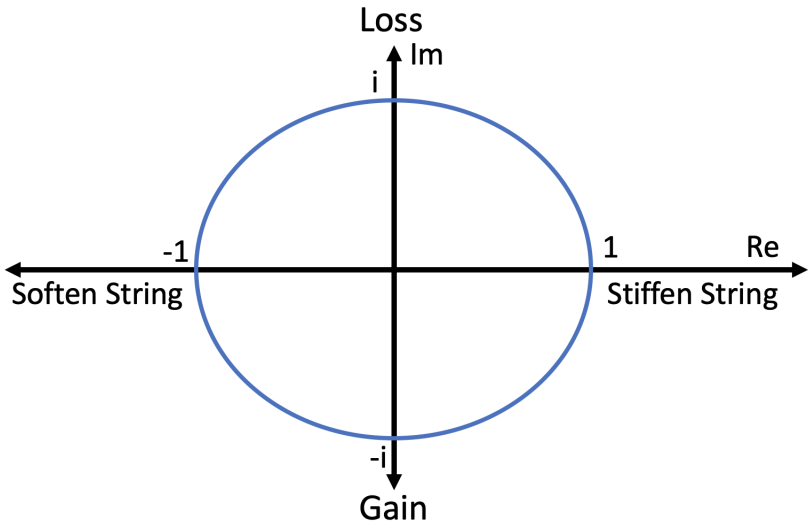

Figure 1.3: Image of the physical effects on harmonic oscillator from the feedback loop due to the phase. Using the feedback loop at a ϕ of $\frac{\pi}{2}$ or $\frac{3\pi}{2}$ will implement a maximum loss or a maximum gain into the system, respectively. Other phases will implement loss and gain too, but they will also affect the stiffness of the spring by changing the spring constant. The γ parameter amplifies these effects on the system.

Figure 1.3, demonstrates the power of the feedback loop on a harmonic oscillator. The phase of the feedback can apply a loss or gain of energy proportional to γ. It can also modify the spring constant k by either stiffening it or softening the spring. The loss and gain exponentially decays or grows in the system until a limit is reached by the oscillator or the feedback module. A similar feedback loop will is constructed and analyzed to control a magnetic oscillator.

1.4 Computational Research

The gain-loss (γ) parameter can be implemented computationally by adding or subtracting energy from equations of motion of the dipoles (shown in chapter two). The γ parameter describes the equal or unequal strength of the gain and loss on the dipoles. The magnitude of the γ parameter determines the nature of unique behaviors observed in the system. The equations of motion for the two-dipole system, which are first-order differential equations, can be solved using initial conditions and an approximation method such as the Euler method or the fourth-order Runge Kutta method implemented via a fast compiled program such as C. The (γ) parameter can equally drive/dampen the dipoles or unequally drive/dampen them, giving us the

ability to simulate, explore, analyze the dynamical behavior that arises from various conditions. [6]

Analytically identifying the limitations of the equations of motion provides us with the parameter ranges to explore. Using the parameter ranges and looping through all possible initial conditions for every gamma value will produce a complete numerical solution for the dipoles. Additionally, the program can estimate the boundary between the symmetric and chaotic motions of the system. Graphically analyzing the numerical solutions of the differential equations demonstrated the four-dimensional nature of the systems exhibits. The mapping of the boundary of symmetrical and chaotic motion is also graphically depicted. To further analyze the system, I used varied amplitude conditions for one dipole and forced cyclical behaviors to arise in the phase space by setting the other initial conditions based on the first dipole angle and a fixed gamma value. Graphically under these conditions, the relationship between the initial starting angles and the angular frequency ω resulted in bifurcations appearing. In section 2.8, there is further analysis of the boundary graph, phase space graphs, bifurcations graphs.

1.5 Experimental Research

Experimentally, gain and loss of energy can be implemented onto the dipoles physically via feedback loops. The feedback loop uses the resonant frequency of the dipole and coils to apply controlled magnetic waves such that the dipole maintains and sustains its oscillatory behavior. The feedback loop can also dampen the oscillatory behavior of a dipole by offsetting the electromagnetic wave phase. To gain complete control over the phase, amplitude, and gain/loss parameters, the integration of a raspberry Pi module was required such that the gain or loss of energy applied to a dipole gets manipulated utilizing a custom-designed python code. However, using a conductor such as copper or aluminum is another method of dampening the oscillatory response. A sold piece of a conductor, when positioned near an oscillating dipole, can induce eddy currents. The eddy currents essentially act as a magnetic drag since they produced a magnetic field equal and opposite to the dipoles. The eddy current drag can be controlled by positioning and orienting the conductor near the dipole such that the dampening force applied is related to some geometrical relationship correlating with the placement of the conductor. The geometric relationships control the amount of energy dissipated from the dipole. Hypothetically, once controllable gain and loss get applied to the two dipoles, we can experimentally confirm

the analytical and simulated results of the system. However, the experimental model would be a Pseudo-PT-Symmetric two dipole system since the equations of motion changed after integrating the coils and the raspberry pi module. Nonetheless, such a system can provide crucial information about possible novel applications of PT-symmetry on magnetic apparatuses. Ultimately, we were not able to develop the Pseudo-PT-symmetric two-dipole model. Still, we thoroughly explored the various mechanism of applying gain and loss on a rotating dipole and laid the foundation for such a system to be built. [7]

Chapter 2

Two Dipole System

2.1 Overview

In our study, we investigate the behavior of two coupled magnetic dipoles. The dipoles can spin freely with no outside influences such as gravity or the Earth's magnetic field. The equations of motion of the system are differential equations that predict how the dipoles change with time by utilizing initial conditions. They describe the dynamics of the two dipoles mathematically by using θ_1 and θ_2 to distinguish each of their positions relative to one another while ω_1 and ω_2 describe their angular spins. In the system we are studying, the two dipoles are modeled by two magnets. The magnets can be coupled to one another through their magnetic fields. If one dipole changes it's position the second dipole can respond to that change accordingly. The reason the second dipole responds is due to the magnetic fields of the first dipole changing at the same time it's position is changed. The changing magnetic filed induces a the second magnetic dipole to move. The induced motion due to coupling between the two magnets can be easily shown experimentally. First, hang two magnets with strings near each other, but not to close so they do not snap together. The dipoles should be stationary at this point and have no motion at all. Now if you perturb one of the dipoles the second one will also move, which demonstrates the concept of magnetic coupling between two magnetic objects. The coupling strength between the two dipoles is dependant on the distance of separation between them. The two magnetic dipole system is the simplest model that can be used to apply PT-symmetry. Adding more

dipoles to the system complicates it and may break symmetries which would lead to difficulties in analytical and computational analysis. Thus, for the remainder of the this thesis we will be focusing on the two dipole system. Figure 2.1 shows a 2D view of the dipoles on a coordinate system.

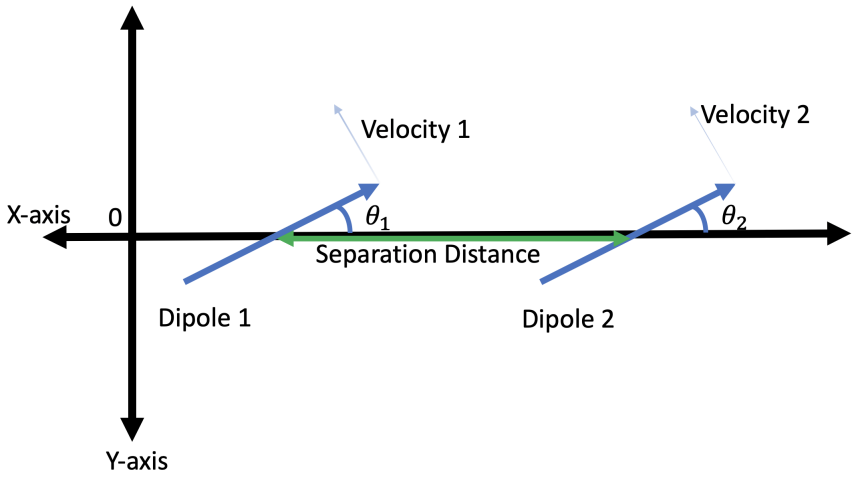

Figure 2.1: The image above is a 2D representation of a two-dipole system. The two dipoles are only allowed to rotate in the XY plane. The rotation causes an angle to be formed with respect to the x-axis for each dipole. Applying an external force to cause the dipole to rotate applies a perpendicular velocity, which over time changes the angle representing the dipole's position with respect to the x-axis. The change in the angle over the change in time gives each dipole an angular velocity. Since we are working with magnetic dipoles, they are coupled to one another by their magnetic fields and the separation distance represents the strength of the coupling.

Upon finding the equations of motion, a gain/loss parameter is integrated into the system. Adding an arbitrary γ value multiplied by the angular velocity of the first dipole, ω_1, introduces a gain parameter onto the dipole. A loss of energy is introduced to the second dipole by subtracting an arbitrary γ value multiplied by the angular velocity of the dipole ω_2. Below the equations of motion describing the PT-symmetric two dipole system with the gain and loss implemented are shown in section 2.3. Using the equations of motion trajectories of the system can be simulated using initial conditions. Examining the system dynamics under various γ values could provide insight into the effects of gain and loss on the system. The trajectories will produce a list of both angles and angular acceleration, which are graphically analyzed. Further exploring unique trajectories or initial conditions of the system can lead to potential novel applications.

2.2 Setup

Either Lagrangian or Newtonian mechanics need to be used to find the differential equations of motion. Using Lagrangian mechanics requires determining the potential and kinetic energy of the system. Newtonian Mechanics requires calculating the forces on the system. Once the equations for the energies or forces are identified, we can figure out the two dipole systems' equations of motion. The Newtonian route is shown above in section 1.2 using a simple harmonic oscillator. If we take the Lagrangian path, then $\mathcal{L}$ = T - V. In this equation, $\mathcal{L}$ is the Lagrangian, T is the kinetic energy, and V is the potential energy. The Lagrangian equation is very similar to the Hamiltonian equation, which is $\mathcal{H}$ = T + V, where T + V is the total energy in the system. However, using the Lagrangian will enable us to obtain the two dipole equations of motion; if we utilize the Euler - Lagrange equation shown in equation 2.1. [9]

$$\frac{d}{dt}(\frac{\partial \mathcal{L}}{\partial \dot{q}}) = \frac{\partial \mathcal{L}}{\partial q} \tag{2.1}$$

Taking the Lagrangian for a specific system and plugging it into the Euler - Lagrange equation, we will find the equation of motion for our system. The Euler - Lagrange equation is consistent with Newtonian classical Mechanics. However, using Newtonian Mechanics requires you to identify all the forces acting in the system. Using the Lagrangian method for a simple harmonic oscillator is not ideal since it only depends on linear forces. It is much easier to use Newtonian Mechanics in that situation, but the Lagrangian will essentially give you the same output; the motion of a harmonic oscillator system is dependant on Hooke's Law (-kx). When dealing with more complex systems, Lagrangian Mechanics is much more practical, in the sense that we can derive the equations of motion for the system just by considering the energies without considering the numerous forces the system may have in multiple dimensions. In the two dipole system, motion is happening in the XY plane, so we have to consider forces in multiple coordinates. The Euler-Lagrange equation can give us equations of motion for the multiple coordinate systems, so we will end up with an equation for the x and y-direction. However, switching to polar coordinates to represent the position of the dipoles can simplify things further, as shown in section 2.3. [9]

2.3 Equations of Motion

Due to the complexity of our system we need to take the Lagrangian Mechanics route. First we need to identify the potential and kinetic energy for the two dipoles to get $\mathcal{L}$ = T - V. Using figure 2.1 as a reference we can see that the system will essentially be a dipole dipole interaction. The potential energy of one dipole in a background magnetic field is going to be $U(\theta) = -m \cdot B = -mBcos(\theta)$. The potential energy of the dipole is dependant on the relative orientation of the dipole with respect to the background field indicated by the angle θ. Now introducing a second dipole into the system the potential energy will change into the potential energy for a dipole-dipole interaction. The potential energy of two dipoles irregardless of the type of dipole, electric or magnetic, is shown in equation 2.2. [7]

$$U = \frac{\mu_0}{4\pi r^3}(\vec{m_1} \cdot \vec{m_2} - 3(\vec{m_1} \cdot \hat{r})(\vec{m_2} \cdot \hat{r})) \tag{2.2}$$

For the potential energy, m_1 and m_2 are the two magnetic dipole moments in the system. The two magnetic dipole moments are related to each other by the distance they are separated from one another. In a background magnetic field the dipole's would align themselves along the x-axis once they are at rest. The separation distance determines the strength of the coupling in the system as mentioned in section 2.1. The position of the two dipoles care determined by their individual angles created relative to the x-axis. Expanding out the dot products in equation 2.2 will reveal how the angles determine the potential energy of the dipoles. Evaluating the dot products in equation 2.2 we get that the potential energy for the two electric dipole system should be represented by equation 2.3. [7]

$$U(\theta_1, \theta_2) = \frac{m_1 m_2 \mu_0}{4\pi r^3}(cos(\theta_1 - \theta_2) - 3cos(\theta_1)cos(\theta_2) \tag{2.3}$$

Now we need to determine the kinetic energy of the two dipoles. The dipoles move in the X-Y plane, and the simple motion can be written as the torque of the system. The definition of the rotational torque is the product of the rotational mass (I = Inertia) and the angular acceleration (Change in velocity over time = $\frac{d\omega}{dt}$). Now we can identify the torque of the magnetic dipoles and equate it to be equal to the $I * \frac{d\omega}{dt}$. However, since we know that the dipoles rotate, they must have rotational kinetic energy, which can be derived from the work of the dipoles; $Work = \tau * \theta$. Going back to the Lagrangian we need to account for the rotational

kinetic energy of each dipole. To do so we need to us $E_{rotational} = \frac{1}{2}I\omega^2$ for both dipoles. Now that the potential energy and the kinetic energy of the two magnetic dipole system has been identified we can write the entire Lagrangian out as shown in equation 2.4. [9]

$$\mathcal{L} = \frac{1}{2}I_1\omega_1^2 + \frac{1}{2}I_2\omega_2^2 + \frac{m_1 m_2 \mu_0}{4\pi r^3}(-cos(\theta_1 - \theta_2) + 3cos(\theta_1)cos(\theta_2)) \tag{2.4}$$

In equation 2.4 the negative sign for the potential was distributed to the two angular terms. Now that we have the Lagrangian equation, we can extrapolate the four equations of motion for the two dipole system using the Euler - Lagrange equation shown in equation 2.1. Due to the two angles in the system, we will have two separate Euler - Lagrange equations, one for the θ_1 dependence and another for the θ_2 dependence, which will result in four equations of motion as shown below. [7,9]

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\theta_1}} = \frac{d}{dt}(I\dot{\theta_1}) = M_1 r^2 \ddot{\theta_1} \tag{2.5}$$

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\theta_2}} = \frac{d}{dt}(I\dot{\theta_2}) = M_2 r^2 \ddot{\theta_2} \tag{2.6}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_1} = \frac{m_1 m_2 \mu_0}{4\pi r^3}(sin(\theta_1 - \theta_2) - 3sin(\theta_1)cos(\theta_2)) \tag{2.7}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_2} = \frac{m_1 m_2 \mu_0}{4\pi r^3}(-sin(\theta_1 - \theta_2) - 3cos(\theta_1)sin(\theta_2)) \tag{2.8}$$

Using the Euler-Lagrange equation requires taking derivatives with respect to angles and time for each term in $\mathcal{L}$. When evaluating the derivatives accounting for every negative signs is important or else the equations of motion will be incorrect. Also, the inertia can be expanded to $I = Mr^2$ if we consider the dipoles to be point masses, where M is the mass of each individual dipole and r is the distance from the axis of rotation. Now that we have all the terms for both sides of the Lagrangian for each dipole evaluated, we can equate equation 2.6 to 2.8 and equation 2.5 to 2.7. When equating the equations to each other we end up the two equations below. The equations show how one dipole will respond due to a change in the other dipole's angle over time and vice versa. [9]

$$M_1 r^2 \frac{d^2\theta_1}{dt^2} = \frac{m_1 m_2 \mu_0}{4\pi r^3}(sin(\theta_1 - \theta_2) - 3sin(\theta_1)cos(\theta_2)) \tag{2.9}$$

$$M_2 r^2 \frac{d^2\theta_2}{dt^2} = \frac{m_1 m_2 \mu_0}{4\pi r^3}(-sin(\theta_1 - \theta_2) - 3cos(\theta_1)sin(\theta_2)) \tag{2.10}$$

2.4 Nondimensionalization

From the equations of motions determined in equations 2.9 and 2.10, we can now scale away the physical dimensions. This allows us to study just the interactions of the dipoles without any external factors. To nondimensionalize the equations we can define the time scale such that $t = \tau \cdot t_0$. Where both t and τ are variable time scales but t_0 is a constant that is comprised of all the constants in the equations of motion. Plugging in the new time scale for both the derivatives results in equations 2.11 and 2.12.

$$M_1 r^2 \frac{d^2\theta_1}{d(\tau \cdot t_0)^2} = \frac{m_1 m_2 \mu_0}{4\pi r^3}(sin(\theta_1 - \theta_2) - 3sin(\theta_1)cos(\theta_2)) \tag{2.11}$$

$$M_2 r^2 \frac{d^2\theta_2}{d(\tau \cdot t_0)^2} = \frac{m_1 m_2}{4\pi r^3}(-sin(\theta_1 - \theta_2) - 3cos(\theta_1)sin(\theta_2)) \tag{2.12}$$

Now we can rearrange the equations of motion in such a way that all the constants are grouped together and can be cancelled by setting the time scale of t_0 as shown below.

$$\frac{d^2\theta_1}{d\tau^2} = \frac{m_1 m_2 \mu_0 t_0^2}{4\pi r^3 M_1 r^2}(sin(\theta_1 - \theta_2) - 3sin(\theta_1)cos(\theta_2)) \tag{2.13}$$

$$\frac{d^2\theta_2}{d\tau^2} = \frac{m_1 m_2 \mu_0 t_0^2}{4\pi r^3 M_2 r^2}(-sin(\theta_1 - \theta_2) - 3cos(\theta_1)sin(\theta_2)) \tag{2.14}$$

For equation 2.13 and 2.14 we see that almost all the terms are equivalent except that some have a subscript 1 or 2 distinguishing between the two dipoles. However, if we essentially use the same magnetic dipoles with the same mass and dipole moment it simplifies the equations since $m_1 = m_2$. Doing so simplifies the equations of motion constants so the two dipoles are purely distinguishable by the angles θ_1 and θ_2. Additionally, using the same magnetic dipoles

enables us to set t_0 equal to the square root of the reciprocal of the constants. Thus, using $t_0 = \sqrt{\frac{4\pi r^3 M r^2}{m^2}}$ simplifies the constants to just one. Now the nondimensionalized equations of motion will be:

$$\frac{d^2\theta_1}{d\tau^2} = (sin(\theta_1 - \theta_2) - 3sin(\theta_1)cos(\theta_2)) \tag{2.15}$$

$$\frac{d^2\theta_2}{d\tau^2} = (-sin(\theta_1 - \theta_2) - 3cos(\theta_1)sin(\theta_2)) \tag{2.16}$$

To simplify the equations of motion further we can use the trigonometric angle difference identity $sin(\theta_1 - \theta_2) = sin(\theta_1)cos(\theta_2) - sin(\theta_2)cos(\theta_1)$. Following the negative signs carefully for each equation of motion and applying the trig identity we get the final form of the nondimensionalized equation of motion shown below.

$$\frac{d^2\theta_1}{d\tau^2} = (-sin(\theta_2)cos(\theta_1) - 2sin(\theta_1)cos(\theta_2)) \tag{2.17}$$

$$\frac{d^2\theta_2}{d\tau^2} = (-sin(\theta_1)cos(\theta_2) - 2cos(\theta_1)sin(\theta_2)) \tag{2.18}$$

The equations are now nondimensionalized to just polar coordinates and the motion of the two dipoles depend solely on the angles. The second order differential equations in 2.17 and 2.18 can be turned into first order differential equations. Since, we know that for a simple pendulum, the angular velocity $\omega = \frac{d\theta}{dt}$. So the second derivative with respect to the angle should just be the first derivative of the angular velocity; $\frac{d^2\theta}{dt^2} = \frac{d\omega}{dt}$. Which is how the four equations below were derived. Once we rewrite the equations we can implement gain an loss by adding and subtracting $\gamma \cdot \omega$.

$$\frac{d\theta_1}{dt} = \omega_1, \tag{2.19}$$

$$\frac{d\theta_2}{dt} = \omega_2, \tag{2.20}$$

$$\frac{d\omega_1}{dt} = -sin(\theta_2)cos(\theta_1) - 2sin(\theta_1)cos(\theta_2) - \gamma \cdot \omega_1 \tag{2.21}$$

$$\frac{d\omega_2}{dt} = -sin(\theta_1)cos(\theta_2) - 2sin(\theta_2)cos(\theta_1) + \gamma \cdot \omega_2 \tag{2.22}$$

Using these equations of motion we can simulate the system in a computer program to obtain information about the behaviour of the dipoles at various starting initial conditions for $\theta_1, \theta_2, \omega_1$, and ω_2. For each starting initial condition, different values of the γ parameter will change the system dynamics. We will like to explore every possible value for γ applied to both dipoles with an equal weight or unequal weight to identify any configurations of the system that may have novel applications.

2.5 Eigenmode Analysis

Before running simulations blindly we would like to analytically analyze the system for it's normal modes. Using the nondimensionalized equations of motion and the small angle approximations of $sin(x) \approx x$ and $cos(x) \approx 1$, the characteristic eigen frequencies of the system can be determined. Applying the small angle approximation we should get:

$$\frac{d\omega_1}{dt} = -2\theta_1 - \theta_2 + \gamma\omega_1 \tag{2.23}$$

$$\frac{d\omega_2}{dt} = -2\theta_2 - \theta_1 - \gamma\omega_2 \tag{2.24}$$

The angles and angular velocity can be written in terms of an initial amplitude and an exponential phase factor. Since θ_1, θ_2, ω_1, and ω_2 are proportional to $e^{i\Omega t}$ the equations of motion can be simplified further to find the characteristic eigen frequencies (Ω_1 and Ω_2) of the two dipoles. Since we know $\frac{d\theta}{dt} = \omega$ we can plug in $\theta = \theta e^{i\Omega t}$. Taking the derivative we can see that $\omega = \frac{d\theta}{dt} = i\Omega\theta$, which means that $\omega_1 = i\Omega\theta_1$ and $\omega_2 = i\Omega\theta_2$. Similarly, we can write $\omega = \omega e^{i\Omega t}$ now taking the derivative we see that $\frac{d\omega}{dt} = i\Omega\omega$, which means that $\frac{d\omega_1}{dt} = i\Omega\omega_1$ and $\frac{d\omega_2}{dt} = i\Omega\omega_2$. These terms can be plugged into equations 2.23 and 2.24 to simplify the equation of motion as shown below.

$$i\Omega\omega_1 = -2\theta_1 - \theta_2 + \gamma\omega_1 \tag{2.25}$$

$$i\Omega\omega_2 = -2\theta_2 - \theta_1 - \gamma\omega_2 \tag{2.26}$$

Now plugging in values determined for ω_1 and ω_2 into 2.25 and 2.26 turns the equations into a linear eigenvalue problem as shown below.

$$-\Omega^2\theta_1 = -2\theta_1 - \theta_2 + i\gamma\Omega\theta_1 \tag{2.27}$$

$$-\Omega^2\theta_2 = -2\theta_2 - \theta_1 - i\gamma\Omega\theta_2 \tag{2.28}$$

Due to the linear nature of the system at small angles, equation 2.27 and 2.28 can be written as the characteristic eigen matrix as shown in equation 2.29.

$$\begin{bmatrix} \Omega^2 - 2 + i\gamma\Omega & -1 \\ -1 & \Omega^2 - 2 - i\gamma\Omega \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = 0 \tag{2.29}$$

Solving the determinant of the 2 by 2 matrix, than setting it equal to zero, and solving for the roots of Ω will produce the characteristic eigen frequencies for the two dipole system. The determinant equation is shown below in equation 2.30.

$$\Omega^4 - 4\Omega^2 + \gamma^2\Omega^2 + 3 = 0 \tag{2.30}$$

The determinant can be solved by grouping the same power polynomials, doing a u-substitution with $u = \Omega^2$, than solving for u with the quadratic formula, and finally back substituting to find Ω. Solving for Ω ultimately generates four characteristic frequencies of the system. The four solutions are shown below.

$$\sqrt{\frac{-\gamma^2 + 4 + \sqrt{\gamma^4 - 8\gamma^2 + 4}}{2}} \tag{2.31}$$

$$-\sqrt{\frac{-\gamma^2 + 4 + \sqrt{\gamma^4 - 8\gamma^2 + 4}}{2}} \tag{2.32}$$

$$\sqrt{\frac{-\gamma^2+4-\sqrt{\gamma^4-8\gamma^2+4}}{2}} \tag{2.33}$$

$$-\sqrt{\frac{-\gamma^2+4-\sqrt{\gamma^4-8\gamma^2+4}}{2}} \tag{2.34}$$

The normal characteristic eigen frequencies occur when there is no gain or loss, thus we can derive them by setting $\gamma = 0$ for the four equations above. Doing so gives us the result of $\pm\sqrt{3}$ for equation 2.31 and 2.32 when $\gamma = 0$ as well as $\pm\sqrt{1}$ or equation 2.33 and 2.34 when $\gamma = 0$. Thus, the normal characteristic eigen frequencies for the two dipole system when there is no gain or loss are $\pm\sqrt{3}$ and $\pm\sqrt{1}$. The frequencies correspond a particular motion of the two dipoles. The lower frequencies represent the motion of the dipoles when they move synchronously as shown in figure 2.1. In the figure both the dipoles are moving in the same angular direction such that the arrows point parallel to one another. The higher frequency represent the motion of the dipoles when they move asynchronously. In terms of the figure it is when the dipoles have the opposite angular direction such that the arrows point perpendicularly to each other. All other motion of the two dipoles are in-between the low frequency and high frequency modes of the system. The analytical analysis of the two dipole system, shows that the motion of the two dipoles are only dependant on the angles and the angular velocity. There are no other parameters to consider for the system since they will not affect the underlying frequencies in the linear regime. However, our linear eigen frequencies do include a γ parameter, which is the addition or subtraction of energy from each dipole. Thus, γ values can change the oscillatory dynamics of the system giving us control over the frequency of oscillation of each dipole.

2.6 PT-Boundary

Since we have identified the determinant equation for the linear eigen value problem of the two dipole system with the gain/loss parameter, we can find the $\gamma_{Critical}$ and γ_{PT} values. $\gamma_{Critical}$ and γ_{PT} determine the boundary of the γ values allowed in the system before the system enters the broken phase region. The γ_{PT} is the specified boundary of the gain-loss term. Any γ values used in the system that are above γ_{PT} will result in a transition to the broken phase region of the system leading to symmetry breaking and chaos in the system. While $\gamma_{Critical}$ is spontaneous PT-symmetry breaking, which is the so called “exceptional point”, where

the system transitions to a new phase associated with complex eigenvalues. To determine the $\gamma_{Critical}$ and γ_{PT} we can use the equation 2.32 obtained from. [15]

$$\Omega = \frac{\pm\sqrt{\gamma_{Critical}^2 - \gamma^2} \pm \sqrt{\gamma_{PT}^2 - \gamma^2}}{2\sqrt{1}} \tag{2.35}$$

Using this equation we can solve for γ_{PT} and $\gamma_{Critical}$ by equating one of the four characteristic eigen equations incorporating γ in equations 2.31 - 2.34. Using the eigenfrequency in equation 2.31 we obtain the equality in equation 2.36.

$$\sqrt{\frac{-\gamma^2 + 4 + \sqrt{\gamma^4 - 8\gamma^2 + 4}}{2}} = \frac{\pm\sqrt{\gamma_{Critical}^2 - \gamma^2} \pm \sqrt{\gamma_{PT}^2 - \gamma^2}}{2} \tag{2.36}$$

Evaluating equation 2.36 at $\gamma = 0$ we can derive $\gamma_{Critical}$ and γ_{PT}. However, doing the same analysis with all four eigen frequency equations we will get the general result that $\sqrt{2 \pm 1} = \frac{\gamma_{Critical} \pm \gamma_{PT}}{2}$. The values of $\gamma_{Critical}$ and γ_{PT} that satisfy the equation are $\gamma_{Critical} = \sqrt{4 + 2\sqrt{3}} = \sqrt{3} + 1$ and $\gamma_{PT} = \sqrt{4 - 2\sqrt{3}} = \sqrt{3} - 1$. Based on the results any value of γ greater than $\gamma_{PT} = \sqrt{4 - 2\sqrt{3}} \approx 0.7320508076$ will break the symmetry in the system. The PT symmetry breaking point can be verified by simulating the equations of motion and incriminating the γ values until the symmetrical motion of the dipoles break. The specified γ value should be around 0.7320508076. Graphing the real and imaginary parts of the eigen frequencies as a function of γ we can obtain the PT-symmetry graph for the two dipole system with the γ_{PT} and $\gamma_{Critical}$ points labeled as shown in figure 2.2.

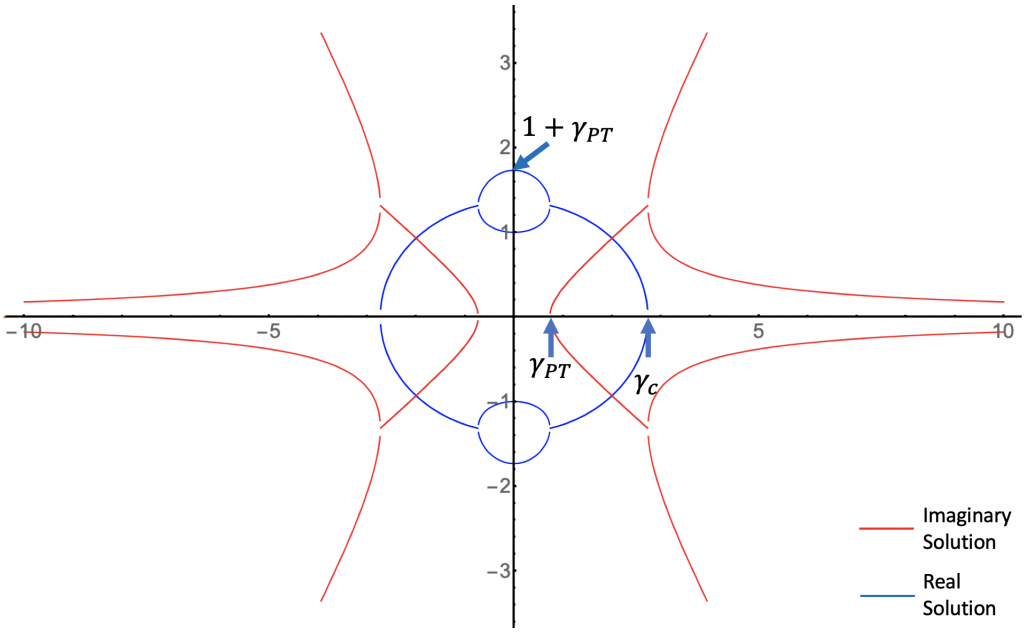

Figure 2.2: Graph of the Real and Imaginary solutions to the eigenfrequencies as a function of γ with the γ_{PT} and $\gamma_{Critical}$ points labeled. The γ_{PT} and $\gamma_{Critical}$ points are fixed for a two dipole system since there is no coupling parameter. All parameters in the system were scaled away, and the underlying physics of this system depends on the initial starting conditions. The magnetic fields of the dipoles determine the coupling, and the distance affects the strength of that coupling. In other coupled systems, like two pendulums attached by a spring, the coupling gets adjusted by changing the spring constant K. In the two dipole system, there is no analogous spring constant to be changed, so γ_{PT} and $\gamma_{Critical}$ are always the same.

2.7 Simulations

To simulate the differential equations quickly, we used a C compiler. A fourth-order Runge Kutta method was used in the compiler to solve the first-order differential equations of motion above. Once the differential equations are solved correctly, the program can be initiated with starting conditions and looped until a specified time. After the program finishes running through all the variables, we can analyze the results by graphing the outputs for both the angles and angular velocity. Thus, we can produce a θ_1 vs ω_1 and θ_2 vs ω_2 phase plot as shown in figure 2.3. [6]

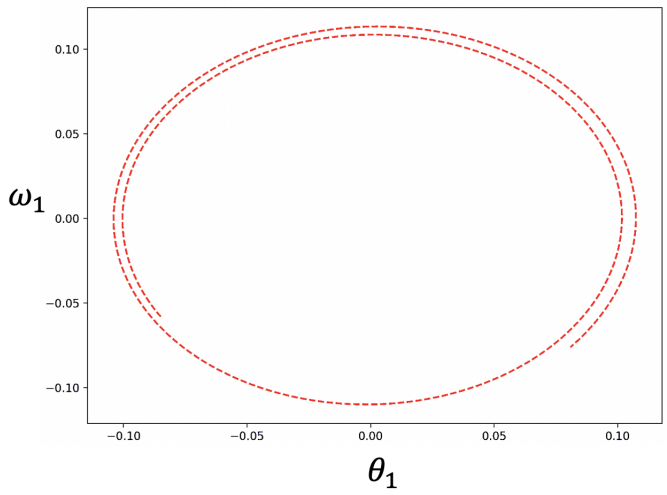

Figure 2.3: Example of the phase plot of the dipole motion. Produced by simulating the equations of motion with initial conditions for θ_1, ω_1, θ_2, ω_2, and γ and graphing its angle and angular velocity. The simulation uses a 4th order Runge Kutta that approximates the solutions to the differential equations. The simulation uses the current initial conditions and calculates the next values using the average of four increments. The smaller the increment size, the more accurate the results are, but the longer the program takes for a given range. Over time the error in the approximation increases, which is why we see that the trajectory in phase space has an error associated with it. The error, at the end of one cycle, can be fixed using linear interpolation.

We want to simulate the system to understand the precise initial conditions that allow the magnetic dipoles to come back to the same starting position, which produces a cyclical behavior. The phase plot starts at one location and comes back to that same location regardless of the duration of the simulation or initial conditions. We will consider a closed phase plot as symmetrical behavior, and an open phase plot indicates non-cyclical symmetry-breaking behavior for the two dipoles due to chaos or bifurcations. Over time, the system is slowly deviating away from symmetrical behavior. At small amplitudes, the solutions to the equations of motion that produce a cyclical behavior are linear solutions, which enable us to use a linear interpolation method at the beginning and end of one cycle. To restrict the simulation to find cyclical solutions, we can derive a formula relating θ_2 and ω_2 to the initial condition of θ_1, ω_1, and γ.

2.7.1 Linear Interpolation

We can use the linear equations in the matrix equation 2.29 and solve for θ_2 in terms of θ_1. Using the terms defined above and simplifying we ultimately obtain the relationship $\theta_2 = (\Omega^2 - 2 + i\Omega\gamma)\theta_1$ and taking the real components of the equation we can see that $\theta_2 = (\Omega^2 - 2)\theta_1$.To find the restriction for ω_2 we can take the derivative of θ_2 which will result in $\omega_2 = (\Omega^2 - 2 + i\Omega\gamma)\frac{d\theta_1}{dt}$. Since we defined the derivative $d\frac{\theta_1}{dt}$ being equal to $i\Omega\theta_1$ above, we can multiply the whole term by $i\Omega\theta_1$ and take the real components to apply in the simulation. The first two terms will be disregarded since they have an imaginary component, while in the last term, the imaginary component will become a negative number leaving us with $\omega_2 = -\gamma\Omega^2\theta_1$. We take the real component of the equation since the simulation is only exploring the real eigenfrequencies of the system as shown in 2.2. It is equally valid to take the imaginary component if the simulation is exploring the imaginary eigenfrequencies by starting with imaginary initial conditions to produce imaginary phase plots. [5]

Thus, the restrictions required in the simulation are represented by equations 2.37 and 2.38. In the equations, x_0 is used in place of the initial θ_1 value to limit the assignment errors when executing the simulation. One of the eigenfrequencies got plugged in for Ω^2, which enables us to solve for ω_2 and θ_2 using the initial conditions assigned to γ and θ_1 in the simulation.

$$\omega_2 = (\gamma * (\frac{\gamma^2 - 4 \pm \sqrt{(\gamma^4 - 8 * \gamma^2 + 4)}}{2}) * x_0) \tag{2.37}$$

$$\theta_2 = (-(\frac{\gamma^2 \pm \sqrt{(\gamma^4 - 8 * \gamma^2 + 4)}}{2}) * x_0) \tag{2.38}$$

Using these equations with any value of θ_1 and γ the simulation will provide a cyclical solution that always has one of the eigenfrequency of the linear system. However, restricting the θ_2 and ω_2 values will only work for small values of γ as shown in figure 2.4.

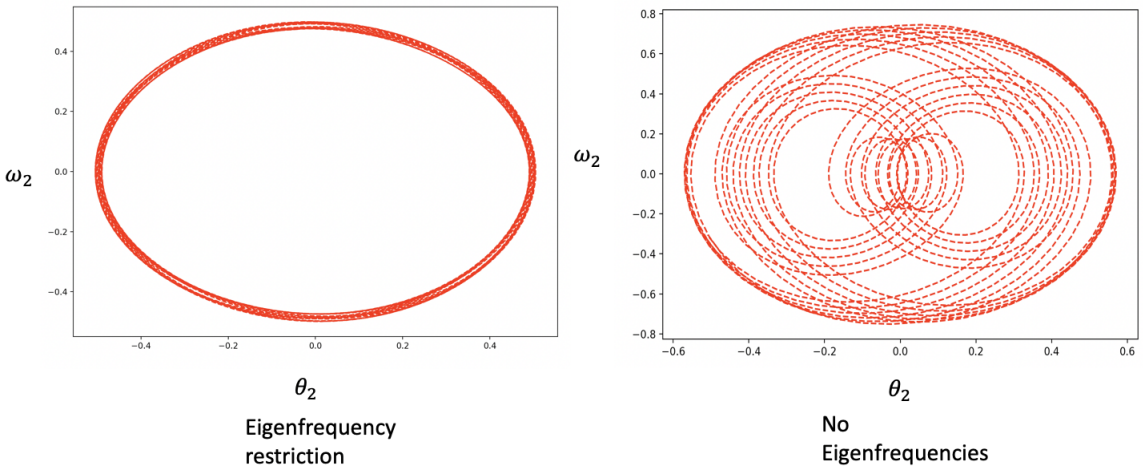

Figure 2.4: The image shows how restricting the values of ω_2 and θ_2 have on the system. Both graphs were plotted using the RK4 function of the simulation at a $\gamma = 0.3$ The starting conditions for the system were $\theta_1 = 0.5$ and a time step dt $= 0.1$. The first phase plot shows the eigenfrequency restriction. The second phase plot shows no restriction present. For the second plot the user has to input values for θ_2 and ω_2. The values plugged in were 0 for both. As you can see the eigen frequencies restrict the phase plots to cyclical behavior but over time at larger γ values the cycles grow bigger. This is why the interpolation is done at the end to prevent the cycle from expanding and only giving cyclical graphs as shown in figure 2.5.

Thus, implementing this method with linear interpolation will fix the error shown in figure 2.3, which means that regardless of the number of phase plot cycles we go through, the program will output closed cyclical plots, as shown in figure 2.4 with an eigenfrequency. In the index, the C code shows how the interpolation was done. Using the cycleprint() function, which produces figure 2.3 since it is not the end points interpolating. Using the Excatcycle() function gives figure 2.5. The Excatcycle() does a weighted average of two points to the left and right of the starting position to make a new point in-between. The program forces the ends to be come back on itself such that over time at any γ value, we will always have one cyclical phase plots. The cyclical phase plots will always have a linear eigenfrequency associated with it due to the restrictions on ω_2 and θ_2.

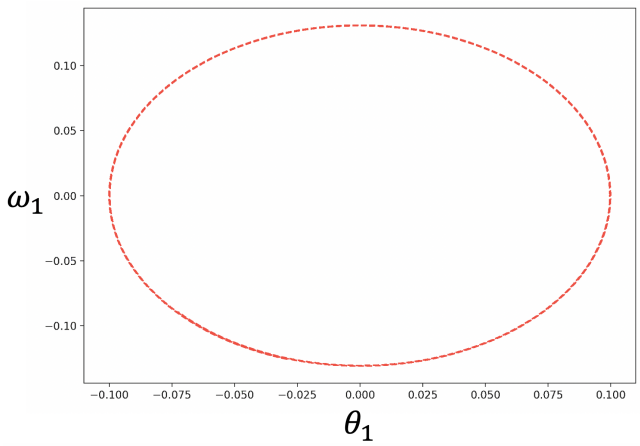

Figure 2.5: Phase plot example with the linear interpolation method applied in the program with no errors in the cyclical behavior. The trajectories return to the original starting position, which results in perfect symmetrical behavior in the system. Pushing the initial conditions for θ_1 and γ until the symmetric phase plots break will produce a graph of the boundary of the symmetry and chaos.

With the linear interpolation complete, the system can be explored at various initial θ_1 and γ values and analyzed by observing variations in the phase plot graphs produced. Figure 2.6 shows an example of a unique behavior seen in the phase plots due to the symmetry restriction imposed above by equations 2.37, 2.38, and the linear interpolation method.

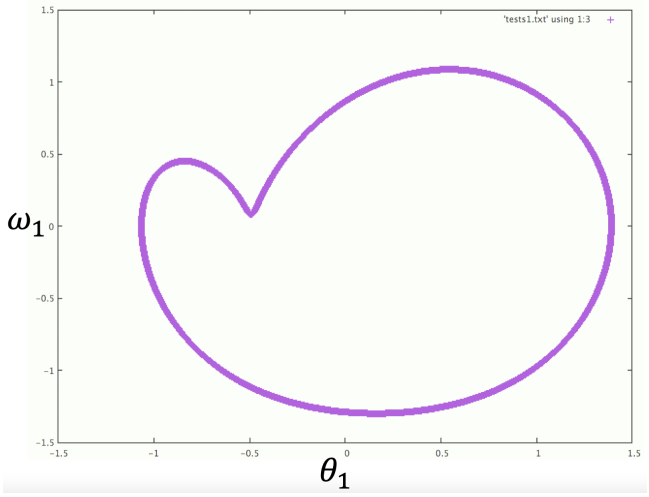

Figure 2.6: A phase graph plotted at $\gamma = 0.5$ showed a kink in one region. The unique phase plots arose after implementing the symmetry restrictions. The point where the phase plot kink occurs seems to be cusp. A cusp is not differentiable so the simulation may be breaking symmetry at those values of θ_1 and ω_1.

The phase plot graph in figure 2.6 shows a closed cycle with a kinked behavior occurring in one region. The behavior persists in the program for various initial conditions. The kinked behavior seems to break symmetry due to the phase plot potentially having a discontinuous hole at the kink position. If this is the case, the symmetrical motion of the dipoles breaks at the values of ω_1 and θ_1. To further analyze the kinked locations on the phase plots, we tried to graph them in three dimensions as shown in figure 2.7.

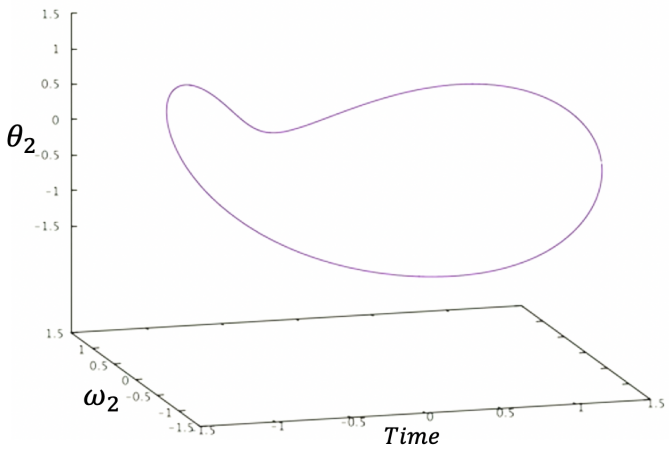

Figure 2.7: The same numerical results from the simulation as figure 2.5, but graphed in 3D. The results showed that the phase plot, viewed from another angle in three dimensions, have cycles that are closed loops. This indicated that the symmetry restrictions implemented in the simulation forced the cycles to go out of the page or into the page. Thus, the 2D phase plots from the program are only showing one slice of the 3D solution of the 4D problem we are investigating.

Graphing ω_1, θ_1, and the iteration step showed that the cycles are coming out of the page or into the page. Ultimately, indicating that the symmetrical restrictions imposed on θ_2 and ω_2 forces the phase plots to become three-dimensional cycles to keep the symmetry behavior of the two dipoles in the system.

2.7.2 PT Boundary Graph and Amoeba

The program can identify the γ_{PT} point by iterating through γ values from 0 to γ_{PT} and checking all the inputs of θ_1 that will enable symmetry at that γ value before symmetrical behavior breaks. Doing this nested for loop is computationally expensive since the iteration steps for variables can cause the simulation to go through millions of initial conditions that will

get plugged into the differential equations with the fourth-order Runge Kutta. To minimize the time required for the program to execute, we embedded a downhill simplex algorithm (amoeba method). The amoeba method is a numerical method used to find the local minimum or maximum of a function in multidimensional space. The amoeba can be implemented in the program to find the initial conditions that produces the smallest error in the phase plot. The program can now minimize the time it takes when iterating through multiple variables. With the downhill simplex algorithm implemented along with the linear interpolation, with the initial values of θ_1 and γ, the program can find θ_2, ω_2, and the smallest error possible in the fastest time possible. Now iterating through many initial conditions for θ_1 and γ with a nested for loop, we will create the graph in figure 2.8 showing the boundary between symmetry and chaos for the two dipole system. The iteration and full code for producing the boundary graphs are shown in appendix B. [5,6,10]

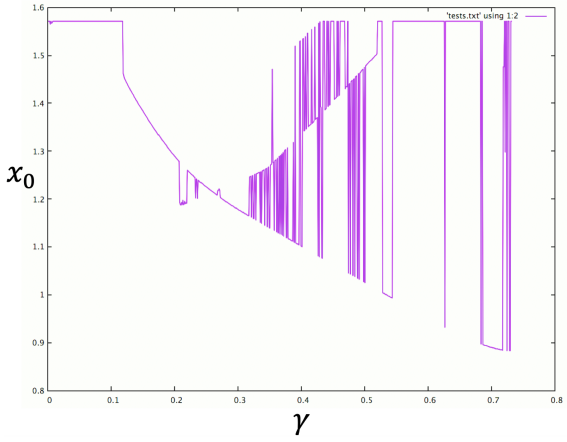

Figure 2.8: The symmetry chaos boundary for the two-dipole system. The graph made by iterating γ and θ_1 values. The maximum θ_1 value was $\frac{\pi}{2}$ and the maximum γ was γ_{PT}. The line connects the largest $\theta_1(X_0)$ points for each successive γ value that the program could not produce a symmetrical phase plot. Thus, for all $\theta_1(X_0)$ above the line, the system will be in a chaotic state. For all $\theta_1(X_0)$ below the line, the system will be in symmetric motion. In the middle of the graph, we can see regions of bifurcation for the boundary, and some other parts of the graph exhibit well-like and constant behavior.

Due to the 360-degree restriction to the θ_1 (X_0) parameter, the boundary will give symmetrical results for the four quadrants of the unit circle. Thus, in the first attempt at producing the boundary graph, we restricted θ_1 iterations between 0 and $\pi/2$. The γ restriction was from 0 to about 0.8. As we can see in figure 2.8, the gamma parameter halted the simulation once it

reached a value between 7 and 8. That value was either $\gamma_{PT} \approx 0.7320$ or slightly bigger than that. In figure 2.7, we can see what the maximum θ_1 value is at each γ value. Any value above that maximum will push the system in chaotic behavior, while any value less than that would result in the system being in a symmetrical state of motion. However, we can see that for the first few γ values and some in the middle of the graph have the maximum θ_1 being at $\pi/2$, so it is not clear if a θ_1 value greater than $\pi/2$ will result in symmetrical or chaotic behavior. Thus, rerunning the simulation enabled us to get a complete picture of the boundary. However, this time the θ_1 (X_0) parameter had a range of 0 to π. Figure 2.9 shows the expanded boundary condition in addition to the same γ value the program terminates; γ_{PT}.

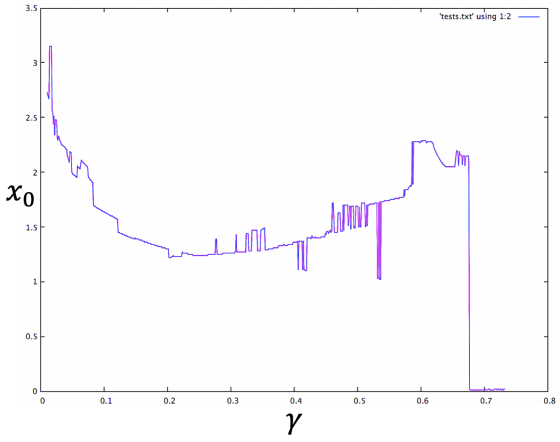

Figure 2.9: Figure 2.7 shows the expanded symmetry-chaos boundary for the two-dipole system. The graph was produced in the same way as figure 2.6 by iterating gamma and θ_1 values. However, this time the maximum $\theta_1(X_0)$ point the program went to was π. Expanding the graph made the boundary look linear, but the bifurcation and the well-like regions were still present.

With the expanded boundary graph, the first few initial γ values have a maximum value for X_0 greater than $\pi/2$. The expanded boundary graph also shows the bifurcations behavior in figure 2.4 seems to be still present but less noticeable after expanding the X_0 range. We have confirmed that the $\gamma_{PT} \approx 0.7320508076$ is the value at which the system symmetry breaks and the initial conditions that enable symmetry. We can analyze specific configurations in the system to explore novel applications of the two dipole system.

2.8 Graphical Analysis

The simulation is now completed and can be used to further understand specific conditions in the two dipole system. Some unique phase plots obtained from the simulation are shown in figure 2.10. The images show how different the two dipole motion can be once symmetry restrictions are imposed in the simulation. The last image shows the phase plot diagram of the two dipole systems when the simulation is pushed into the chaotic boundary region.

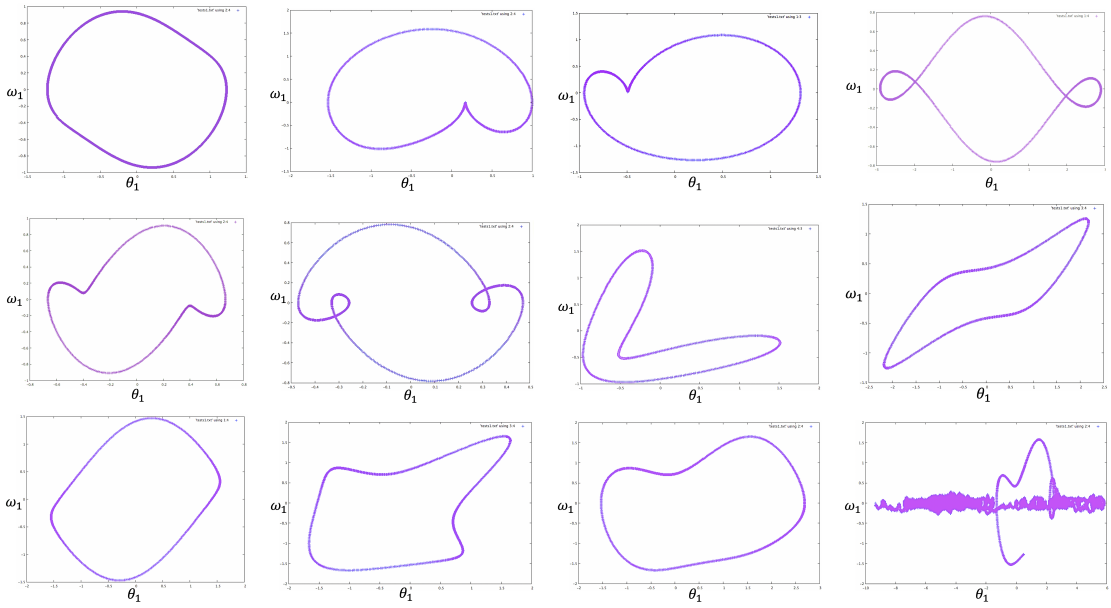

Figure 2.10: The above images are different phase plots from the simulation. Starting the simulation at various initial conditions produced the unique phase plots above. The bottom right graph shows how the symmetry in the system breaks at γ values bigger than γ_{PT}. A similar response in the system happens with higher initial values of $\theta_1(X_0)$. The increased $\theta_1(X_0)$ values cause more errors in the approximation, which the downhill simplex method can not resolve. Leading to the program halting and deciding there are no more values of $\theta_1(X_0)$ at this specific γ which will produce symmetric phase plots.

Based on the images, it seems like the symmetry restriction on θ_2 and ω_2 makes the phase plots sensitive to the initial conditions the program is started at. To explore the behaviour of θ_2 and ω_2 we can gradually change the $\theta_1(X_0)$ value at a specified γ value. This procedure will enable us to look at what is gradually occurring in parameter/phase space by graphing the $\theta_1(X_0)$ against both θ_2 and ω_2. In order to explore how things are changing in parameter space I ran the simulation with an initial $\theta_1(X_0) = 0.1$, $\omega_1 = 0$, and $\gamma = 0.403$ since that is where

we observed interesting bifurcation behavior occurring in figures 2.8 and 2.9. After running the program at these starting conditions, I obtained figure 2.11 below.

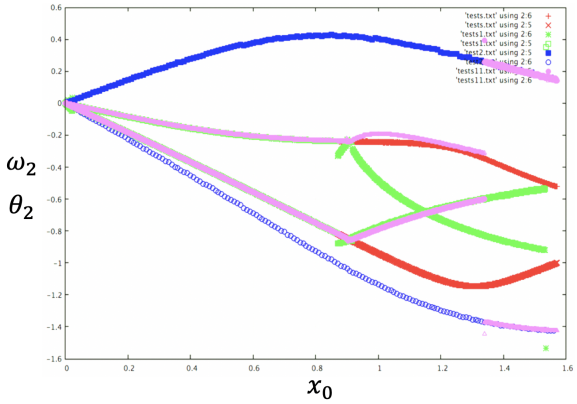

Figure 2.11: Deviations in parameter space plot. The image shows how θ_2 and ω_2 change at $\gamma = 0.403$ as the program increments through $\theta_1(X_0)$. Bifurcations in the parameter values are shown at $\theta_1(X_0) \approx 0.9$ which is approximately equal to $\sqrt{3}/2$. The bifurcations indicate that there are two symmetrical solutions, phase plots, for the system. Slight deviations from the initial starting $\theta_1(X_0)$ value results in the program taking one path or another. In some instances, the graph of θ_2 and ω_2 at $\gamma = 0.403$ as the program increments through $\theta_1(X_0)$ instantaneously jumps from one solution to another as seen in the pink line.

The image in figure 2.11 is a composite of various plots of θ_2 and ω_2 vs $\theta_1(X_0)$. Individually, the bifurcation behavior was not clearly observable, it is only after combining two graphs with slight deviations in the initial $\theta_1(X_0)$ value from 0.01 to 0.011. The slight initial condition difference produced two different trajectories like the light green lines and the red lines. However, once combined together they reveal the bifurcation point, which is between $\theta_1(X_0)$ of 0.8 and 1. Graphically, the point looks very close to 0.9 but I suspect due to the symmetry restrictions imposed onto the simulation with equations 2.34 and 2.35 that $\theta_1(X_0) \approx \sqrt{3}/2 \approx 0.866025$. Towards the end of the pink line, we can see that the solution jumps to the trajectory of the outside blue line in the graph. This behavior shows that the system can instantly jump to another solution of the equations of motion behavior that we would like to explore experimentally.

2.9 Summary of Findings

In this chapter, we established the theory behind a two dipole PT-symmetric system we are investigating and how gain and loss can be implemented onto the dipoles to create a PT-

symmetric system. We have also demonstrated the derivation of the equations of motion of the system using Lagrangian mechanics. We demonstrated how the equations of motions can be nondimensionalized such that the system can be modeled by the angle of the two dipoles. In the linear regime, the equations of motion can be simplified using the small-angle approximations, which enables us to use matrix analysis to find the four characteristic eigenfrequencies of the two dipole system. From this, we were able to extrapolate the normal modes of the system when there is no γ dependence to be $\pm\sqrt{3}$ and $\pm\sqrt{1}$. Afterwards using the characteristic eigenfrequencies and equation 2.35 we were able to solve for the values of $\gamma_{PT} = \sqrt{4-2\sqrt{3}}$ and $\gamma_{Critical} = \sqrt{4+2\sqrt{3}}$.

After analytically analyzing the equations of motion for the two dipole system we used a fourth-order Runge Kutta method and initial conditions for the variables to simulate the two dipoles. Using the numerical output of the simulation for each variable phase plots of the position vs angular velocity were constructed. Some initial conditions pushed the system out of a perfectly closed phase plot indicating those initial conditions will eventually lead the system into chaotic motion. To impose symmetrical motion on the system through perfect closed phase plots the end of the cycle was linearly interpolated and the restrictions were imposed onto the initial values of ω_2 and θ_2 based on the values the user plugged in for θ_1 and γ. The restrictions for both θ_2 and ω_2 were derived from the linear eigenfrequencies. Further simulations with various initial conditions demonstrated new kinked behavior in the phase plots. This behavior was later found to be a result of looking at a 2D image of the 3D phase plot solution of the 4D dynamical problem of the PT-symmetric two dipole system.

Pushing the simulation further to uncover the symmetry chaos boundary in the system required the implementation of a gradient descent algorithm to reduce the error in identifying symmetrical cycles and increase the speed of the program when iterating through multiple variables by determining the change in the error during each step enabling the simulation to move the step direction that reduces the error further ending in a local minimum. Implementing the gradient descent algorithm improved the speed of the program and enable us to iterate through every possible $\theta_1(X_0)$ and γ combination for the two dipole system resulting in the boundary graphs shown in figures 2.8 and 2.9. The figures demonstrated that the analytical derivation of γ_{PT} was correct since the program stopped functioning once the loop iterating through all the γ values reached $\gamma_{PT} \approx \sqrt{4-2\sqrt{3}} \approx 0.7320508076$. Finally, analyzing the boundary diagrams

showed bifurcations behaviors occurring in the simulation leading to further investigation of the parameter space. This time using a fixed γ value while iterating through $\theta_1(X_0)$ and overlaying multiple graphs of ω_2, θ_2 vs $\theta_1(X_0)$ showed more bifurcation behaviours analogous to chaotic systems. Figure 2.11 shows multiple points of bifurcation in the system as well as solutions to the system jumping from one symmetrical trajectory to another.

2.9.1 Future Computational Analysis

Our study using a simulation of the PT-symmetric two dipole system has shed light onto the dynamics of magnetic systems. However, the computational analysis can be pushed further. For instance, additional studies on the phase diagrams can give insight into more unique conditions the system can exhibit with potential novel applications. By saving more output text files of the solutions for various initial conditions and overlaying them will produce the entire phase space region of solutions for the system. Plotting all the phase space graphs for every initial condition possible in three dimensions will create the entire three dimensional solution for the forth dimensional problem of the PT-symmetric two dipole system.

Additionally, further simulation can be done to study additional bifurcational behaviour in the system liked the ones observed in figure 2.11 at $\gamma = 0.403$. More simulations can be done at various other γ values to see if the bifurcational behaviour in the parameters still persists. The simulation can be altered to start backwards from the initial conditions given and slowly approach zero to confirm the bifurcations points are indeed bifurcations in the solutions to the differential equations and not just artifacts of the simulations. Finally, further investigation into similar parameter space graphs like figure 2.11 may produced initial conditions where the system will start jumping through very possible solution and become discontinuous over the entire parameter range. Such a behaviour would have unique novel applications that would have to be tested experimentally. I suspect a discontinuous stable system maybe possible at the boundary regions or near γ_{PT}.

Finally, the boundary conditions can be simulated for the maximum possible ranges for γ and $\theta_1(X_0)$. The full range simulation would make sure that the entire picture for the boundary conditions would graphically depicted. The graph of the full range simulation can possibly provide more information on the two dipole system's dynamics.

Chapter 3

Electronic Feedback Loop - Single Dipole System

3.1 Overview

Ultimately, the simulations revealed how the two dipole system would behave in an ideal world, but we would like to understand how the PT-symmetric two dipole system behaves experimentally. Thus, I began to develop a method of producing an experimental example for the two dipole system. After a model of the system is constructed, it could be used for verifying the findings of the analytical and simulated results obtained in chapter 2. In this chapter, we demonstrate how a feedback loop mechanism can apply rotational gain and loss of energy to a single magnetic dipole.

However, building an experimental model is easier said than done. The difficulty with making a prototype of the PT-symmetric system is discovering the best way to implement the gain and loss of energy. There have been PT-symmetric experiments done where one part of the system is over-damped compared to another. Subsequently, this means that the other part of the system is considered to have a gain of energy relative to the first part of the system that has more loss. Thus, the system is analogous to a PT-symmetric system without dealing with an external gain of energy. Experimentally, it is easy to work with losses, but the gain is more complicated

to implement. As shown in chapter one, the increase of energy in a simple mass with a spring system can be implemented with a sensor and drive mechanism through a feedback loop. The feedback allows greater control over the system input energy and enables phase shifting of the output wave affecting the motion of the mass on the spring. In our experimental model, we will not be dealing with two dipoles with losses. We will explore different techniques of applying rotational gain of energy to a magnetic dipole that freely rotates in the XY plane, similar to that of the mass on a spring with a positive feedback loop.

3.2 Setup

To represent one dipole in our experiment, we initially used two bar magnets clamped around a super-thin and durable piece of string. The opposite end of the wiring was attached to a cork that fit on top of the test tube. This setup allowed the dipole to hang freely inside the test tube and rotate in the XY plane, as shown in figure 3.1. Before tackling the rotational gain problem, we'll have to distinguish the best method of inducing rotational loss on the dipole. The easiest way that we found that provides complete control over the loss of energy applied was using a conductor such as aluminum or copper that can induce eddy currents onto the rotating magnet. An eddy current is an electrical current that is produced in a conductor by a changing magnetic field. The current, in turn, generates a magnetic field that opposes the changing magnetic field produced by the dipole. Thus, the conductor and its eddy currents act like a magnetic dampener. Based on the position and orientation of the conductor, the magnitude of the opposing magnetic field can be changed, resulting in a controllable method of applying magnetic oscillatory loss based on the geometrical coordinates of the aluminum conductor in our experiment. The conductor can induce light dampening on the dipoles oscillations. We were unable to experimentally get critical dampening or over-dampening on the dipole since they require the conductor to be close to the magnet resulting in mechanical interference of its rotation.

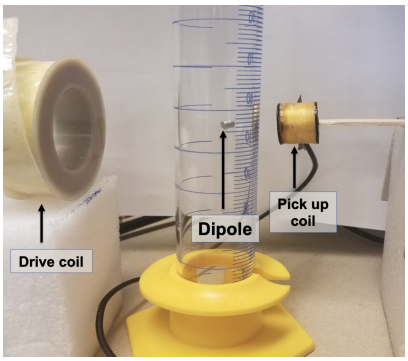

Figure 3.1: The experimental setup to apply gain with a coil (Drive coil) to a bar magnet dipole. The second coil (pick-up coil) measures the oscillations of the dipole and observes the behavior of the dipole on an oscilloscope. The oscillations of the dipole are sent from the pick-up coil to an amplifier and then sent back to the dipole using the drive coil. The dipole at the center of the test tube is two bar magnets clamped together between a string. Due to this setup, the dipole has a slight tilt due to the Earth's magnetic field. Another magnet has to be positioned around the test tube to cancel out the effects on the dipole.

Now that we have a method of tuning the loss on one of the dipoles, we need to develop a way of applying gain to a second dipole. Electronically, an exponential increase of energy is achievable using an RLC resonant circuit with a negative resistor. However, we cannot buy the elements for the RLC circuit, so components have to be made from scratch. The values of the inductor, capacitor, and resistor have to be analytically or computationally identify. We can construct the RLC circuit once the values that work best for the two-dipole system are known. We successfully found the numerical results for each circuit element, but the necessary Q values of the LC resonance at the experimental frequencies are not achievable. It would take a lot of time and resources to build the custom components for the circuit at the low frequencies of our dipole in the experiment. Thus, we had to use another approach. Instead of producing a tunable physical RLC resonant circuit, we decided to use a feedback loop system similar to the one shown with the mass on a spring in chapter one but with coils, as shown in figure 3.1. One coil measures the magnetic dipoles oscillations (pick-up), and another one produces magnetic torque (drive). Using a filtering amplifier to amplify the measured signals of the dipole from the pick-up and sending it back to the drive coil completes a feedback loop. The loop will sustain dipole oscillations at its resonant frequency if the phase is chosen correctly in the filtering. Ultimately, we utilized the feedback loop system to apply gain in the experimental model. Upon integrating the hypothetical RLC circuit or the feedback loop system, the equations of motion

of the system change. The characteristic eigenfrequencies have to be analytically re-derived in sections 3.3 and 3.4.

3.3 +RL Circuit Equations of Motion

Even though the RLC resonant circuit cannot be built and used to apply gain to one dipole in the experimental model, we can obtain analytical and computational results for one dipole and the RLC resonant circuit. The analytical results may apply to higher frequency situations such as NMR since the resonant RLC circuit can work for higher frequencies in the radio frequency range (20 kHz - 300 GHz). Applying the RLC circuit to the dipole will result in additional equations of motion depending on the current going through the circuit. First, we can analyze one inductor coil and a positive resistor (RL circuit) with a dipole. This setup essentially will act like a magnetic dampener, just like the eddy currents in the conductor. Thus the dipole near the inductor will experience loss of energy from its angular rotation. To apply gain to the dipole, we will need to implement a negative resistance in the circuit. However, the negative resistor cannot be by itself and requires a capacitor to maintain a non-instantaneously growing voltage proportional to $e^{t/(RC)}$. Thus, applying gain to a dipole using a negative resistor requires us to have a resonant RLC circuit. To understand the new system, we can rederive the equations of motion for the positive RL circuit, which behaves like a controllable dipole dampener. We can rederive the equations for the negative RLC circuit that acts as a controllable dipole driver. The eigenfrequencies of the system will show how the circuits affect the motion of the dipole. However, since this system is not PT-symmetric, there is no γ_{PT} or $\gamma_{Critical}$ points in the system. Subsequently, solving the one dipole gain problem with the equations of motion and coupling it to the equations of motion of a second dipole with loss will result in a PT-symmetric problem. The system is dependant on γ, ω, and the initial angle of the dipole θ. The γ is dependent on the current, voltage, and 3D position/orientation of the coils, circuit, and conductor. [19]

Implementing a positive RL component to on dipole doesn't change the θ dependence of the angular velocity as shown in the equation of motion for the one dipole. However, since there is no second dipole, the derivative of ω simplifies, and there is no need to distinguish with subscripts. Furthermore, since the current through the positive RL circuit determines the loss applied to the dipole, the equation for $\frac{d\omega}{dt}$ will now need to depend on I*R. Deriving the equations of motion from the Lagrangian once again can be done. However, we know that the

dipole will have two differential equations for the θ and ω dependence, while the Rl circuit will have just one differential equation for the current dependence. We can identify the relationship between the current and angular velocity to find the differential equations of motion for this system. We know already that for this system that $\frac{d\theta}{dt} = \omega$ and for the RL circuit the differential equation for the current is $\frac{dI}{dt} = \frac{V(t)-IR}{L}$. The V(t) can be represented by Faraday's law where the voltage induced is $V(t) = -N\frac{\Delta B \cdot A}{\Delta t}$ and the $B \cdot A$ is the magnetic flux Φ_m. The dot product of B and A would result in $V(t) = \frac{-NBAcos(\theta)}{\Delta t}$. Since our θ is changing over time in the system we can take the time derivative of the flux to get $V(t) = NBAsin(\theta)\frac{d\theta}{dt}$. Plugging into the current differential equation and substituting ω we get $\frac{dI}{dt} = \frac{NBAsin(\theta)\omega - IR}{L}$. We can substitute in the area of the coil, which is πa^2 and the magnetic field of the dipole when the magnetic moment is aligned at the z-axis above the coil; $\frac{\mu_0|m|}{2\pi r^3}$. To obtain the final differential equation of motion, we need to look into the torque of the dipole in the magnetic field. Thus, $\tau = m \times B_z = G\alpha = G\frac{d\omega}{dt}$. In this equation, G is the inertia of the dipole, m is the magnetic moment, α is the angular acceleration, which can be written as $\alpha = \frac{d\omega}{dt}$, and B_z is the magnetic field produced by the coil. The cross product of $m \times B_z$ should yield $-B_z sin(\theta)$, which results in $\frac{d\omega}{dt} = \frac{-B_z sin(\theta)}{G}$. The magnetic field, B_z, can be rewritten as $\frac{\mu_0 NIa^2}{2r^3}$, which gives us the last equation of motion $\frac{d\omega}{dt} = \frac{-\mu_0 NIa^2 sin(\theta)}{2r^3 G}$. The three equations of motion for the one dipole and the positive RL coil are shown below. [7,19]

$$\frac{d\theta}{dt} = \omega \tag{3.1}$$

$$\frac{d\omega}{dt} = \frac{-\mu_0 NIa^2 sin(\theta)}{2r^3 G} \tag{3.2}$$

$$\frac{dI}{dt} = -\frac{RI}{L} + \frac{\mu_0 mNa^2}{2r^3 L}\omega sin(\theta) \tag{3.3}$$

Using the small angle approximation for the dipole angle $sin(\theta) = \theta$. Now we can nondimensionalize the equations of motion by scaling with the time, which results in the equations below. Setting $t = \bar{t} * t_0$ will enable us to once again analyze the system in the linear regime to find the characteristic eigenfrequencies of the system. Then setting the time scale to $t_0 = \frac{2r^3}{\mu_0 Na^2}$ we can simplify the equations of motion to equations 3.4 to 3.6.

$$\frac{d\theta}{d\bar{t}} = \omega t_0 \tag{3.4}$$

$$\frac{d\omega}{d\bar{t}} = -\frac{Isin(\theta)}{G} \tag{3.5}$$

$$\frac{dI}{d\bar{t}} = -\frac{RIt_0}{L} + \frac{m\omega}{L} sin(\theta) \tag{3.6}$$

To push the system one step further, we can analyze how the dipole behaves with the RL circuit in a background magnetic field. This situation will model our dipole in an NMR machine with a coil that can apply gain to if there is a negative resistance. The background magnetic field can be applied to the equations of motion by understanding how the background field will affect the dipole. Ultimately, the background field is perpendicular to the system will just affect the dipole angular velocity by dampening it with a factor of $Bcos(\theta)$. This value can be implemented into equation 3.6. Afterward, doing the matrix and determinant calculations, we can find the characteristic eigenfrequencies of the one dipole and positive RL coil in a background magnetic field system.

$$\frac{d\omega}{d\bar{t}} = -\frac{Isin(\theta)}{G} - B_0 * cos(\theta) \tag{3.7}$$

Since we now have all the terms for the equations of motion we can pick an orientation of the dipole to make the calculations easier. We will consider the dipole aligned perpendicular to the coil axis and parallel to the background magnetic field. This will change the $cos(\theta)$ to $sin(\theta)$ and vice versa in the equations of motion enabling the linearization of the system to be much similar. The final equations of motion based on the new set up are shown below.

$$\frac{d\theta}{d\bar{t}} = \omega t_0 \tag{3.8}$$

$$\frac{d\omega}{d\bar{t}} = -\frac{Icos(\theta)}{G} - B_0 * sin(\theta) \tag{3.9}$$

$$\frac{dI}{d\bar{t}} = -\frac{RIt_0}{L} + \frac{m\omega}{L} cos(\theta) \tag{3.10}$$

With the equations of motion, we can create the characteristic matrix and then solve for the characteristic eigenfrequencies of the system. Before making the matrix we have to use the small angle approximation for $sin(\theta) = \theta$ and $cos(\theta) = 1$. For the angle, current, and angular velocity, we can correlate them to some exponential formula like $X = X_0 e^{i\Omega t}$. Taking the time derivative will result in $\frac{dX}{dt} = i\Omega * X$. Using this simplification for the derivatives allows us to rewrite the equations of motion in the form of a linear eigenvalue problem, as shown in the matrix below. Finding the determinant of the 3x3 matrix shown below will enable us to solve the eigenfrequencies. Simplifying using $\lambda = i\Omega$ results in:

$$\begin{bmatrix} \lambda & -t_0 & 0 \\ B_0 & \lambda & \frac{1}{G} \\ 0 & -\frac{m}{L} & \lambda + \frac{Rt_0}{L} \end{bmatrix} \begin{bmatrix} \theta \\ \omega \\ I \end{bmatrix} = 0 \tag{3.11}$$

Solving for the determinant gives us the equation 3.12. In the equation we can simply terms by equating them to constants like $\chi = \frac{m}{GL}$, $\kappa = \frac{Rt_0}{L}$, and $\beta = t_0 B_0$, which results in the equation 3.13.

$$\lambda^2(\lambda + \frac{Rt_0}{L}) + \lambda\frac{m}{GL} + t_0 B_0(\lambda + \frac{Rt_0}{L}) = 0 \tag{3.12}$$

$$\lambda^2(\lambda + \kappa) + \lambda\chi + \beta(\lambda + \kappa) = 0 \tag{3.13}$$

Solving for λ should give us the characteristic eigenfrequencies of the positive RL circuit with one dipole in a background magnetic field. However, the equation above is a third degree polynomial and it requires using the cubic formula to solve for the eigenfrequencies of the equations of motion. We can however use the quadratic equation if we set the background magnetic field to zero. This would allow us to understand how the dipole behaves with just the RL coil. Setting $B_0 = 0$ would mean that $\beta = 0$, so the last term of the determinant becomes zero and we get $\lambda^3 + \lambda^2\kappa + \lambda\chi = 0$ as out new equation. Now factoring out λ the trivial solution would be $\lambda = 0$, but the other solution would be $\lambda = \frac{-\kappa \pm \sqrt{\kappa^2 - 4\chi}}{2}$. There will always be three or less eigenfrequencies for the determinant of this system even when reintroducing the background field. The background field will shift the solutions to the determinant as shown in the solution to the cubic formula below. To simplify the long cubic formula I set $a_1 = (\frac{\kappa^3}{27} + \frac{\kappa\chi}{6} - \frac{\beta\kappa}{2})$,

$a_2 = \frac{\beta\kappa}{3} - \frac{\kappa^2}{9}$, and $a_3 = \frac{\kappa}{3}$. Thus, the cubic solution to the eigenfrequencies will become equation 3.14.

$$\lambda = \sqrt[3]{a_1 + \sqrt{a_1^2 + a_2^3}} + \sqrt[3]{a_1 - \sqrt{a_1^2 + a_2^3}} - a_3 \tag{3.14}$$

Thus, based on the values of the inductor (L), resistor (R), mass, inertia, and the initial angle (θ) of the dipole, we will be able to determine the dynamics of the system. Now that we have successfully modeled the RL coil and analytically understand the behavior of the dipole in the presence of the coil, we apply gain to the dipole using a negative resistance in the coil instead. However, the negative resistance cannot be physically by itself and requires that the circuit have a parallel capacitor present in the system. This turns the circuit into a negative RLC resonant circuit. The analysis of the circuit is shown in the next section.

3.4 -RLC Resonant Circuit Equations of Motion

A schematic of the circuit in question is shown below in figure 3.2 of the RLC resonant circuit with the magnetic dipole near the inductor. This entire setup would be surrounded by a background field. Additionally, due to the integration of a parallel capacitor, there will be another equation of motion that will be describing the changing voltage. This indicates that for a one dipole RLC resonant circuit, a 4X4 matrix will be constructed, and solving the determinant will result in the new characteristic eigenfrequencies for the circuit.

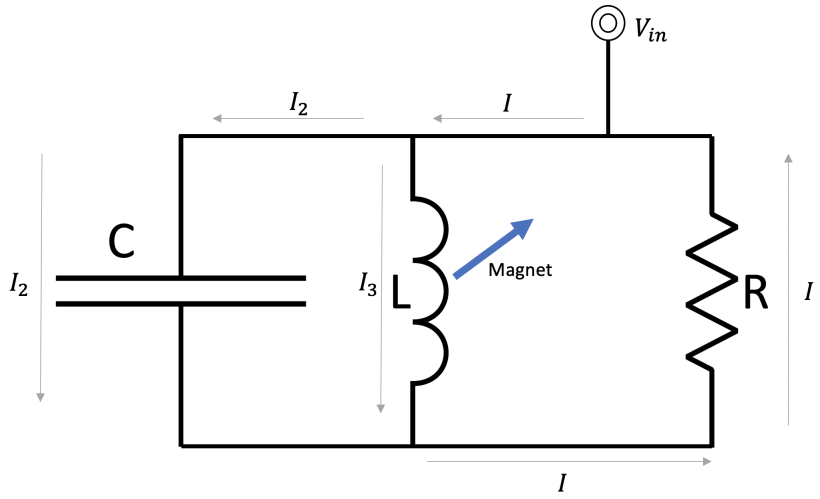

Figure 3.2: Theoretical design of the RLC circuit with the currents labeled and the magnet positioned near the inductor so energy can be applied to the dipole from the inductor once there is a negative resistor. The calculated values of the capacitance, resistance, and inductance were not physically possible. Thus, implementing a feedback loop was the most ideal method of applying gain to the dipole.

Based on this diagram, we can develop new equations of motion that will describe the oscillations of the dipole with the circuit. The three equations of motion above will stay the same, but now there needs to be a voltage differential equation due to the addition of the capacitor. The current can be modeled through a capacitor as $I = C\frac{dV}{dt}$. If we use Kirchhoff's current law, we can identify the differential equation for voltage since $I_c + I_r + I_l = 0$. Since there is a constant current flow through the inductor, the voltage is zero, so $I_l = 0$ due to $I_l = L\frac{di}{dt}$, however, the current through the resistor is found from Ohm's law I = V/R. Also, there is an induced current ($I_{induced}$) in the inductor from the dipole's magnetic field, which is the differential equation modeled above for the current. From these relations we obtain that $C\frac{dV}{dt} + \frac{V}{R} + I_{induced} = 0$. Thus, the four unscaled equations below represent the differential equations for the one dipole and RLC resonant circuit model. [19]

$$\frac{d\theta}{dt} = \omega \tag{3.15}$$

$$\frac{d\omega}{dt} = -\frac{\mu_0 a^2 N I sin(\theta)}{2r^3 G} \tag{3.16}$$

$$L\frac{dI_{induced}}{dt} = V - V_{induced} \tag{3.17}$$

$$C\frac{dV}{dt} = -I_{induced} - \frac{V}{R} \tag{3.18}$$

The $V_{induced}$ in equation 3.17, is the voltage in the inductor produced by the dipoles magnetic field calculated from the magnetic flux (EMF) in the system as shown in equation 3.2. The V term in equation 3.18 is just the voltage through the circuit resistor represented by Ohm's law once again is $V = IR$. Representing the differential equations as exponential terms as we have done before and using the small-angle approximations with the new coordinate system will produce another linear eigenmode problem for the system. First, we need to nondimensionalize the equations once again, so the characteristic eigenfrequency solutions are dependant on scaled versions of the voltage, current, angular acceleration, and the initial position of the dipole. Expanding the equation's results in.

$$\frac{d\theta}{dt} = \omega \tag{3.19}$$

$$\frac{d\omega}{dt} = IA_1 cos(\theta) - B_0 sin(\theta) \tag{3.20}$$

$$\frac{dI_{Induced}}{dt} = \frac{v}{L} - A_2\omega cos(\theta) \tag{3.21}$$

$$\frac{dv}{dt} = \frac{-I_{Induced}}{C} - \frac{v}{RC} \tag{3.22}$$

In the equations above, $A_1 = \frac{\mu_0 a^2 N}{2r^3 G}$ and $A_2 = \frac{\mu_0 a^2 Nm}{2h^3}$. The linearized version of the equations of motion are shown below where for a small angle θ cos(θ) is 1 and $sin(\theta)$ is θ. Additionally, using the following definitions $u = Iz_0$, $\phi = tw_0$, $w_0 = A_2$, and $z_0 = A_1$ will simplify the differential equations further since we can set $I = \frac{u}{z_0}$ and t = $\frac{\phi}{w_0}$. Thus, we obtain the following linearized equations.

$$\frac{d\theta}{d\phi} = \frac{\omega}{\omega_0} \tag{3.23}$$

$$\frac{d\omega}{d\phi} = \frac{uz_0}{\omega_0 z_0} - \frac{B_0}{\omega_0}\theta \tag{3.24}$$

$$\frac{du}{d\phi} = \frac{vz_0}{\omega_0 L} - \omega z_0 \tag{3.25}$$

$$\frac{dv}{d\phi} = -\frac{u}{z_0\omega_0 C} - \frac{v}{\omega_0 CR} \tag{3.26}$$

Now we can simplify the linearized version by grouping constants into new constants. I will be substituting the following constants below: $\beta = B_0$, $\delta = \frac{z_0}{L}$, $\alpha = \frac{1}{z_0 C}$, and $\gamma = \frac{1}{RC}$.

$$\frac{d\theta}{d\phi} = \frac{\omega}{\omega_0} \tag{3.27}$$

$$\frac{d\omega}{d\phi} = \frac{u}{\omega_0} - \frac{\beta\theta}{\omega_0} \tag{3.28}$$

$$\frac{du}{d\phi} = \frac{\delta v}{\omega_0} - \omega z_0 \tag{3.29}$$

$$\frac{dv}{d\phi} = -\frac{\alpha u}{\omega_0} - \frac{\gamma v}{\omega_0} \tag{3.30}$$

Now we have the constants all condensed, however we still have ω_0 in the equations so in order to remove them I will create new variables for them instead of trying to condense the ω_0 into the ones above. The new variables will be $\overline{\omega} = \frac{\omega}{\omega_0}$, $\overline{u} = \frac{u}{\omega_0}$, $\overline{v} = \frac{v}{\omega_0}$, and $\overline{\theta} = \frac{\theta}{\omega_0}$. Then the final equations of motion should be the ones shown below.

$$\frac{d\overline{\theta}}{d\phi} = \overline{\omega} \tag{3.31}$$

$$\frac{d\omega}{d\phi} = \overline{u} - \beta\overline{\theta} \tag{3.32}$$

$$\frac{d\overline{u}}{d\phi} = \delta\overline{v} - \overline{\omega}\omega_0 z_0 \tag{3.33}$$

$$\frac{d\overline{v}}{d\phi} = -\alpha\overline{u} - \gamma\overline{v} \tag{3.34}$$

We know that ω_0 and z_0 are both constants so we can group them together and call it σ. Now that we have the simplified linearized equations of motion we can now put it into matrix representations as shown in equation 3.32 and then find the determinant to obtain the eigenvalues of the system.

$$\begin{bmatrix} \lambda & -1 & 0 & 0 \\ \beta & \lambda & -1 & 0 \\ 0 & \sigma & \lambda & -\delta \\ 0 & 0 & \alpha & \lambda+\gamma \end{bmatrix} \begin{bmatrix} \overline{\theta} \\ \omega \\ \overline{u} \\ \overline{v} \end{bmatrix} = 0 \tag{3.35}$$

The determinate of the matrix above is calculated using Mathematica's determinate function for matrices and the output is:
Det $= \alpha\beta\delta - \gamma\sigma\lambda + \beta\gamma\lambda - \sigma\lambda^2 + \beta\lambda^2 + \alpha\delta\lambda^2 + \gamma\lambda^3 + \lambda^4 = 0$
Where $\lambda = i\Omega$

From the determinant we can see that $\alpha * \delta$ are always together so we can set $\alpha * \delta = \chi$. This finally simplification results in the determinant below.

Det $= \chi\beta - \gamma\sigma\lambda + \beta\gamma\lambda - \sigma\lambda^2 + \beta\lambda^2 + \chi\lambda^2 + \gamma\lambda^3 + \lambda^4 = 0$

Solving the determinate requires a quartic equation since the determinant is a fourth-order polynomial. However, it is much easier to graph the function with values for each constant to obtain what the eigenfrequencies for the system will be. Thus, this is the farthest analytical analysis of the RLC resonant circuit. Afterward, we transitioned into determining if the circuit can be physically constructed. The first limitation in building the circuit is that our inductor has a copper wire that is too fine, leading to high resistance in the coil and a smaller capacitance. Using experimental data we obtained from the voltage measurements of the inductor, we used an RLC fitting routine to fit the data using a least-squares method to find the optimal parameters of the inductor to work in the circuit setup. The routine was done at low frequencies around 38 - 40 Hz since that was the resonant frequency of the dipole we used at the time. The results we obtained showed that the inductor internally would need a resistance of 13.90 Ohms, 0.1575 µF, and 23.73 mH. Due to the parameters required for the inductor, we are unable to make or find

one that fits all three criteria due to materials and the degree of precision needed to make such an inductor. However, instead of using a negative resistor, we will be using the model 113 Pre-Amp to create a feedback loop with the inductor to apply the gain to the neodymium-magnet.

3.5 Coil Integration and New Equations of Motion

Due to the unrealistic values of resistance, capacitance, and inductance, we had to go with developing a magnetic feedback loop. The original loop incorporated two coils and a preamp. The first coil picks up the dipole oscillations and then sends it to the preamp, which amplifies the signal to send through the second coil, which generates electromagnetic oscillations. However, the coils will produce more complex behavior in the system by changing or affecting some of the differential equations of motion. Thus, for the two coils one dipole feedback loop system, the equations of motion need to be re-derived once more. However, we will have equations representing the interaction of both coils with the magnet and with each other based on their positions. The diagram of the dipole with the coils is shown in figure 3.3. From the image, we obtain the following differential equations. [19]

$$\frac{d\theta}{dt} = \omega \tag{3.36}$$

$$V_d = G(\Omega)V_p \tag{3.37}$$

$$V_p = V_{pi} - I_p R_c \tag{3.38}$$

$$V_{pi} = L_p \frac{dI_p}{dt} + M\frac{dI_d}{dt} + N_p \pi R_{effp}^2 \frac{d}{dt}(B_m \cdot U_p) \tag{3.39}$$

$$\frac{V_p}{Z_{in}} + C_c \frac{dV_p}{dt} + I_p = 0 \tag{3.40}$$

$$G\frac{d\omega}{dt} = (m \times B_d)_z - B_0 sin(\theta) \tag{3.41}$$

After completing the cross product the new expression for $\frac{d\omega}{dt}$ will become the following:

$$\frac{d\omega}{dt} = I_d \frac{\mu_0 N R_{eff}^2}{4Gd^3}(2sin(\theta - \theta_1)cos(\theta_2) + cos(\theta - \theta_1)sin(\theta_2)) - \frac{B_0}{G}sin(\theta) \tag{3.42}$$

Now that we have all the terms of the equations of motion we can linearize with $sin(\theta) = \theta$ and $cos(\theta) = 1$. We also need to solve the time derivative of the field of the magnet and the pickup coil. The dot product, when simplified and derived, gives us 0. Thus, the final linearized equations of motion are as followed:

$$\frac{d\theta}{dt} = \omega \tag{3.43}$$

$$V_d = G(\Omega)V_p \tag{3.44}$$

$$V_p = V_{pi} - I_p R_c \tag{3.45}$$

$$i\Omega L_d I_d = V_d - I_d R_s \tag{3.46}$$

$$\frac{V_p}{Z_{in}} + C_c i\Omega V_p + I_p = 0 \tag{3.47}$$

$$\frac{d\omega}{dt} = I_d \frac{\mu_0 N R_{eff}^2}{4Gr_d^3}(2sin(\theta_1)cos(\theta_2) - cos(\theta_1)sin(\theta_2)) - \frac{B_0}{G}\theta \tag{3.48}$$

$$V_{pi} = i\Omega L_p I_p + i\Omega I_d M + \frac{\mu_0 m_p}{4\pi r_p^3}\omega(2cos(\phi_2)sin(\theta_1 + \phi_1) - sin(\phi_2)cos(\theta_1 + \phi_1)) \tag{3.49}$$

From the equations above we see that there are two sets of equations of motion; mechanical and electronic. The mechanical equations of motion relate to the dipole's motion. Equation 3.43 describes the dipole's angular velocity, which is the change in the angle formed with the x-axis over time. Equation 3.48 describes torque on the dipole due to the background magnetic field as well as the position and orientation of the two coils. The electronic equations of motion are divided among the drive coil and the pick-up coil. In the equations, the drive coil ignores the

voltages induced by the moving permanent magnet (PM) dipole moment and the small pickup coil current. For the drive coil, equation 3.44 describes the feedback amplifier gain where G(Ω) is what amplifies the pick-up coil signal. The amplification can be a complex number based on filtering and phase shifting parameters. Equation 3.46 describes the drive coil and current relationship. The last three equations above correlate to the pick-up coil. Equation 3.45 is relating the internal voltage drop of the pick-up coil. The pick-up voltage is reduced by the internal coil resistance R_c. Equation 3.47 shows the relationship between the pick-up current and voltage. The internal pick-up current the amplifier input impedance Z_{in} and the coil capacitance. The final equation, 3.49, describes the internal pick-up coil induction from self-inductance, mutual inductance with the drive coil, and the motion of the permanent magnetic dipole. Using the new feedback equations of motion we can explore how the feedback controls dipole's motion. However, initializing the feedback with the dipole, there was interference from the mutual inductance of the two coils. To limit the mutual inductance the spatial orientation and positions of both coils need to be determined analytically. [19]

3.6 Feedback Loop Coil Orientation Optimization

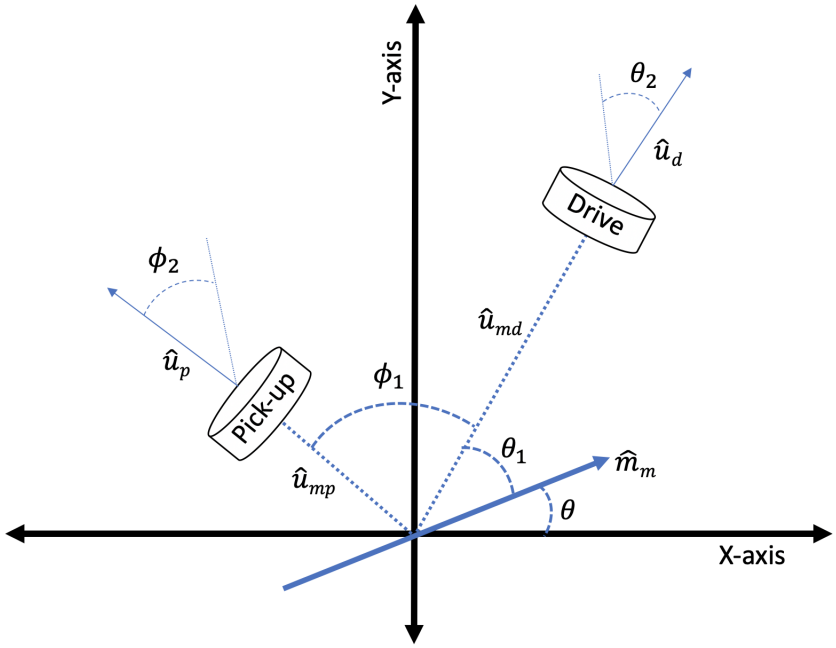

Figure 3.3: The image shows a 2D geometric representation of the one dipole two coil feedback loop system. The position and orientation of all components are represented by the angles formed. The drive coil is further away since it can affect the dipole from farther distances. The pick-up coil is closer to the dipole to measure its oscillations. θ_1 and θ_2 represent the position and orientation of the drive coil. ϕ_1 and ϕ_2 represent the position and orientation of the pick-up coil. θ is the position of the dipole relative to the X-axis. The diagram also shows the unit vectors for the pick-up and drive coils by themselves and with the dipole. The unit vectors aid in calculating the mutual inductance in the system. Ideally, the mutual inductance between the coils should be zero, and the interaction between the coils and dipole should be maximal.

Now since we know the position of the drive coil with respect to the neodymium center we can find the orientation of the pick-up coil with respect to the drive coil so there is no interaction between the two coils. In order to do this, we can find the mutual inductance term for the coils which can be done in a similar way that we found B_d above using the following equation and substituting the correct vectors and unit vectors relating to the individual coils and using the dot product between the vectors instead of the cross product. The general equation is shown below. [7]

$$B(r) = \frac{\mu_0 m}{4\pi} \frac{3U_r(U_r \cdot U_m) - U_m}{r^3} \tag{3.50}$$

Using this similar equation we were able to find all the relations between the coils and the magnet in terms of orientation and position in a 2D space since were are doing the experiment on a flat table. The mutual inductance is given by the B(r) of the coils dot U_m of either of the coils. By expanding out the dot products we were able to obtain the expression for part of the numerator being $3(U_{21} \cdot U_{m1}) \cdot (U_{21} \cdot U_{m2}) - (U_{m1} \cdot U_{m2})$. After obtaining this expression we can set the expression equal to 0 and then substitute using trig identities the corresponding cos and angle terms for each part of the equation and then simplify to solve for one of the angles; $\theta_1 or \theta_2$. Solving the expression by substituting the correct unit vectors in we obtain the following value for the mutual inductance corresponding to the orientations of the coils with respect to each other:

$$M = C((3cos(\theta_2 - \phi_1 - \phi_2) + cos(\theta_2 + \phi_1 + \phi_2))(1 + r_2^2) - (2cos(\phi_1)cos(\theta_2 - \phi_1 - \phi_2) + 6cos(\theta_2 + \phi_2))r_2) \quad (3.51)$$

$C = \frac{\pi \mu_0 N_1 N_2 R_1^2 R_2^2}{r_{21}^3}$

After solving for the orientations of the coils with respect to the neodymium-magnet at the center we can now experiment analytically and physically if the angles allow us to minimize the high-frequency feedback oscillations and maximize the feedback gain or essentially the oscillation of the magnets.

Afterward, we used the equations of motion and put everything in matrix form to find the determinant of the matrix, so we can get the roots to the polynomial expression the determinant gives us. Since we wanted to find the roots of a very simple situation, we used the case where the mutual inductance was 0. From this, we got the polynomial form:

$$Q\lambda - (\beta + \lambda^2)(\lambda L_d + R_s) \quad (3.52)$$

The original matrix was the following before we set the mutual inductance(M) equal to 0.

$$\begin{bmatrix} \lambda & -1 & 0 & 0 & 0 & 0 & 0 \\ \beta & \lambda & -A(2sin(\theta_1)cos(\theta_2) - cos(\theta_1)sin(\theta_2)) & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda L_d + R_s & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -g & 0 \\ 0 & 0 & 0 & R_c & 0 & 1 & -1 \\ 0 & -B(2cos(\phi_2)sin(\theta_1+\phi_1) - cos(\theta_1+\phi_1)sin(\phi_2)) & -\lambda M & \lambda L_p & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \omega \\ I_d \\ I_p \\ V_d \\ V_p \\ V_{pi} \end{bmatrix} = 0 \quad (3.53)$$

Once we set the mutual inductance to 0 and found the determinant, and simplified the output, we get the following expression for the polynomial that provides us with the roots:

$$BgA\lambda(2cos(\theta_2)sin(\theta_1) - cos(\theta_1)sin(\theta_2))(2cos(\phi_2)sin(\theta_1+\phi_1) - cos(\theta_1+\phi_1)sin(\phi_2)) - (\beta+\lambda^2)(\lambda L_d + R_s) \quad (3.54)$$

In the full polynomial g is the gain term, β is the background field over the inertia G, A is the $\frac{\mu_0 N R_{eff}^2}{4Gr_d^3}$, and B is $\frac{\mu_0 m_p}{4\pi r_p^3}$. Within B, the m_p is the dipole moment of the pick-up coil which is just $NI_p\pi R_{eff}^2$. Since we know all the constant variables and their meanings now as well as the fact that we want zero mutual inductance, we solve for the angles orientations that allow for zero mutual inductance with the expression for the mutual inductance, equation 3.51, and then analytically find an equation or numerically find the angles that give the maxi um gain in the system.

To solve the maximum orientations of the system, I solved for ϕ_2 from the mutual inductance term and then plugged that value into the ϕ_2 in the gain term. From there I used Mathematica's maximization function to find the maximum value given by the orientations that the gain will multiply which was 3.54 in terms of the other three angles and since I know that the ϕ_2 expression I derived analytically shown below we can back substitute the values of the other three angles that give the maximum that allows me to find the value of ϕ_2 without effecting the maximum gain and leaving the mutual inductance in the system to be zero.

$$\phi_2 = -ArcTan\left(\frac{-4cos(\theta_2)cos(\phi_1) + 2sin(\theta_2)sin(\phi_1) + (7cos(\theta_2) + cos(\theta_2 - 2\phi_1))r_2 - 4cos(\theta_2 - \phi_1)r_2^2}{2cos(\phi_1)sin(\theta_2) + 4cos(\theta_2)sin(\phi_1) + (-5sin(\theta_2) + sin(\theta_2 - 2\phi_1))r_2 + 2sin(\theta_2 - \phi_1)r_2^2}\right) \quad (3.55)$$

Using the ϕ_2 expression in conjunction with the spatial orientation parameters that determine the effect of the gain, I was able to identify the ideal positions and orientations of the

drive and pick-up coils. The distance ratio of the drive to the pick-up has been set to be 5:1 respectively on the 2D image of the system above. In order the get the maximum gain of about 3.37, we need the specific angles below:

$\theta_1 = 1.0307577357826916$

$\theta_2 = -0.5148402449919208$

$\phi_1 = 0.8177559661104004$

$\phi_2 = 0.02668032651015571$

These values for the angles are shown in figure 3.4. Now that we have the single spin system locked down mathematically, we can simply develop a double spin system simply by adding another dipole on the same axis as the one before but respectively positioned parallel to the field lines so it doe not feel any actual effects from the drive and pick up coil only from the damping factor. We ultimately can make the damping factor anything we want as long as it does not affect the geometrical relationships of the single spin and coil system we solved above. This being said we can place an aluminum sheet next to the second dipole far enough away from the dipole getting driven but close to damp the dipole that's not driven. Then having the two dipoles coupled will create the double spin system we simulated before.

The feedback loop has a restriction on the orientation and position of the pick-up coil, drive coil, and magnetic dipole. In the system, we want the dipole to interact maximally with the two coils, while the two coils have zero or minimal interaction with each other (zero mutual inductance). To obtain the orientation and position of all three components of the system, we can use the 2D schematic shown in figure 3.3. The figure takes into consideration the dipole being the center of the entire system and the two dipoles being some ratio of distance away from each other.

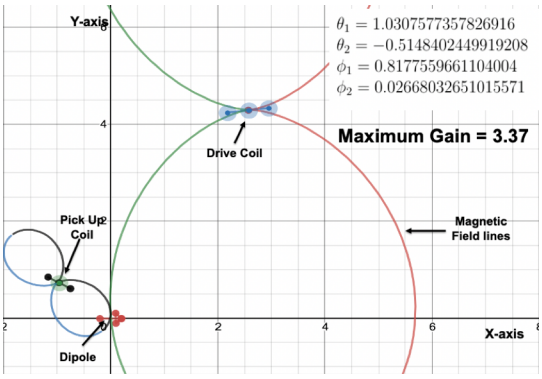

Figure 3.4: The image shows the optimal position and orientation for the angles of the one dipole two-coil system. The angles generate zero mutual inductance between the two coils and the most interaction with the dipole. The ratio of the distance of the coils from the dipole is 1 to 5. The theoretical maximum gain factor for the system is 4, but due to the zero mutual inductance restriction, it is 3.37. The setup shown above is one of many that can produce a gain of 3.37. Based on the calculations, the coils need to be perpendicular to each other while the magnetic field lines of the coils are parallel to each other but perpendicular to the dipole.

3.7 Summary of Findings

In this chapter, we have shown how the gain of rotational energy can be implemented in our experimental system. The gain of rotational energy has been theoretically and analytically examined for an RLC resonant circuit with negative resistance. The equations of motion for the RLC resonant circuit with a negative resistor have been identified above. We have found the characteristic eigenfrequencies for the RLC circuit. Each frequency corresponds to a particular mode. Once we extrapolated the optimal resistor, inductor, and capacitor values for the circuit, we realized that the circuit could not be built based on the values we obtained. This leads us to construct a feedback back loop with two coils. The equations of motion for the two coil and one dipole system were found. Afterward, the characteristic eigenfrequencies were obtained from the matrix analysis. Once the mathematical analysis was completed, the two coils connected to a preamp resulted in a feedback loop. Starting the magnet with an external impulse resulted in the feedback loop kicking in to sustain the oscillatory behavior. We also were able to show that the magnet had a slight tilt in its orientation inside the tube due to the earth's magnetic field. We were able to use an external magnet to straighten the dipole to an XY axis and then using two additional larger background magnets symmetrically positioned away from the dipole resulted in a background field behavior similar to an NMR machine.

After testing the feedback loop at various conditions, a mutual inductance behavior between the two coils was observable. Intuitively, a coil that measures magnetic oscillations and another coil that produced magnetic oscillations would be able to communicate with each other. This communication between the coils is not desired since it can complete a feedback loop that overpowers the feedback loop with the dipole. Thus, the geometric constraints of the two coils corresponding to the dipole had to be derived such that there is zero mutual inductance between the two coils and maximum interaction between each coil and the dipole. An optimization algorithm was used in Mathematica by constraining one angle relative to the other three in the determinant. Afterward, identifying the angles that gave maximum interaction between the coils and the dipole we were able to back-calculate the fourth angle such that there was zero mutual inductance in the system. This resulted in the graph in figure 3.4. However, the constraint angles force the maximum gain on the dipole to be 3.37 instead of 4.

3.7.1 Future Experimental Analysis

In our experimental analysis, we have demonstrated how resonance can apply gain into a system through a feedback loop. However, the same behavior can be done with an RLC circuit with a negative resistance implemented. In the future, the circuit can be built and tested to verify the analytical derivations, the previous results of the simulations, and potential technological novel applications. The circuit can be custom-built today for radio frequencies (20 kHz - 300 GHz), but it may take substantial time and resources to build and tune each custom part to work at low frequencies (0 Hz - 600 Hz). If the circuit is built with the dipole another dipole can be spaced a specific distance away with either a conductor to impose magnetic drag or an RL/RLC circuit with a positive resistor, whichever method is easier to implement loss in the system. The orientation and positional angles identified for the two coils are just one solution that results in maximum interaction between the coils and dipole while having zero mutual inductance. Thus, all the other angles corresponding to the positions and orientations of the coils can be identified using the equations shown above with a similar optimization method used or a better method. Once we started obtaining experimental data for the one dipole two-coil system, there were pendulum motions observed in the dipole due to it being suspended from a string of wire. To zero out the pendulum motion a clamp can be constructed such that the dipole cannot sway in any direction forcing it to oscillate in the XY plane. Any experimental method to get rid of the pendulum oscillations would work. However, in chapter 4, we will

introduce a custom-designed C-clamp specifically designed to hold the dipole and enable it to rotate using jewel bearings.

Chapter 4

Raspberry Pi Control System

4.1 Overview

The proof of concept of applying gain applied to an oscillating dipole is shown in chapter 3. However, as we discussed in chapter 1 with the simple harmonic oscillator, there needs to be greater control over the dipole motion by manipulating the phase of the applied gain either through a digital or analog method. Furthermore, the pendulum motion from the current setup needs to be addressed as well. Once these changes get implemented into the system, measurements can be taken of the dipole's motion to observe how each component changes the equations of motion. The data will enable us to characterize the transient and steady-state behavior of the system when there is gain applied and when the gain gets turned off. Finally, once a complete understanding of how the dipole motion will change due to the gain apparatus, another dipole can be placed nearby with loss to set up a pseudo-PT-symmetric coupled two dipole system similar to the one computationally simulated in chapter 2.

4.2 Raspberry Pi Integration

A python code is designed and implemented in the feedback loop to obtain further control of the dipole's oscillations. The code continuously filters the waves picked up by the pick-up coil digitally. Before developing the code, we first had to identify hardware capable of inputting

and outputting electromagnetic waves, which would allow us to interface with the coils. We ultimately found out that a Raspberry Pi 3 B+ with a HiFi Berry DAC/ADC sound would be the cheapest and most versatile option. The Raspberry Pi has Python 3 built-in, which allows us to develop a code to manipulate the electromagnetic waves inside. To send the electromagnetic waves into the Raspberry Pi, we utilize the 8th-inch jack port on the HiFi Berry sound card using BNC to 8th-inch adapter shown in Figure 4.1. The HiFi berry sound card sits on top of the raspberry pi general-purpose input/output (GPIO) pins. The information obtained by the pick-up coil gets converted and then travels through the audio ports into the GPIO pins and then into the Python 3 code for manipulation. Once the code completes filtering, it sends out the filtered wave to the drive coil or oscilloscope via the GPIO pins to the RCA ports on the HiFi berry sound card and finally to the BNC to RCA adaptors, as shown in figure 4.1. [3,4,8,11,12,13]

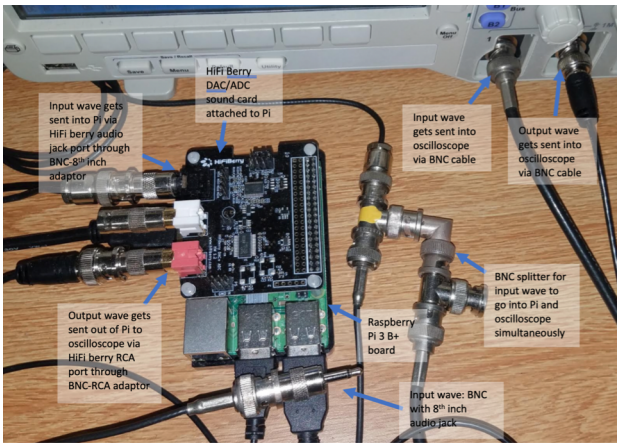

Figure 4.1: The image shows the Raspberry Pi B+ integrated setup with the two coils. The HiFi berry ADC DAC sound card is sitting on top of the GPIO pins of the Raspberry Pi. The coils were custom built with BNC cables attached. Thus, to connect them to the Pi-DAC module, BNC to RCA and BNC to 8th-inch adaptors were used. The adaptors convert the signal to be sent into the Raspberry Pi to get manipulated and sent back out. [3,4,8,12]

4.3 Python Assisted Control of Dipole Oscillations

We used a raspberry pi and hifi berry DAC/ADC sound card module to gain control of the feedback loop. When attaching the sound card, you need to load all the drivers onto the boot menu, or else the input and output channels will not work. Additionally, it is necessary

to import the pyaudio library from the terminal for the Python 3 code to work. The pyaudio library facilitates the input and output of audio signals into python through a NumPy array. Array manipulation can allow us to change the input signal in any way and send it back out to the drive coil to change the oscillatory behavior of the dipole. Using each element of the NumPy array, a recursive band-pass filter can be applied such that the input signal is filtered by a resonant frequency so only that frequency is sent out. Looping the code for the duration the script runs enables real-time filtering of the pick-up coil signal. However, the filtering process is delayed, which causes the output signal to be phase shifted. Adding additional internal delays to the code can enable a greater degree of phase shift to happen. Furthermore, we can multiply the NumPy array by scalar before the filtering process, so the signal amplitude gets amplified. Further, taking steps to automate the filtering procedure remotely, we utilized a virtual network to connect to the raspberry pi remotely. Now the python codes can be altered and run remotely from anywhere.

4.3.1 Recursive Filter

The signal coming into the python code is in byte format and gets converted in numerical float values using the command numpy.fromstring(array, dtype = np.float32). Afterward, a recursive filter is applied to each element of the NumPy array of the signal. The filter uses the previous two elements and weights them by scalars predetermined by the resonant frequency, Q factor, and the sampling rate of the input signal. Equation 4.1 shows the form of the recursive filter.

$$x[i] = a * array[i] + b * x[(i-1)\%CHUNK] + c * x[(i-2)\%CHUNK] \tag{4.1}$$

There is a lot of subtle things occurring in this filter. First, x is a predefined array the same size as the signal array, but with all zeros. The constants a, b, and c are the constants that weigh each term based on the factors mentioned above. The definitions for each term are $c = -e^{-\frac{2\pi f_0}{Qf}}$, $b = 2e^{-\frac{\pi f_0}{Qf}} cos(2\pi \frac{f_0}{f})$, and $a = \frac{1-b-c}{Q}$. Thus, before running the filter, the scalars need to be calculated in the same order they are defined. In each definition, f is the sampling rate of the audio card which is set to 22050 for the program, f_0 is the resonant frequency, and the quality factor Q determines the bandwidth of the filter around the resonant frequency. The array is the NumPy array with the signal from the pick-up coil. The index i indicates that the

filter loops through every element of the signal array, change the value of that element based on the filter parameters, and saves the value to the x array by overwriting the zero at that same specified index. The second two terms use modular arithmetic with the previous two values of the x array since modular arithmetic gives the remainder of division in python. The CHUNK parameter is the size of the signal sample array, which for our code can be set to any power of 2, such as 1024 or 8192. Thus, taking the modular of an index returns that same index, but if the index is negative, it returns the last or second to last element of the array. This is necessary to use since, at i = 1, the first element of the signal sample array does not have two previous points that can be used to filter the current point. Using the modular index allows us to use the last two elements of the x array, which are zero, to filter the first element of the signal sample array. Once the filter is done changing all the elements in the sample array, the x array can be converted back into bytes and sent out again as bytes using the command x.astype(numpy.float32).tostring(). [11,16,18]

4.3.2 PyAudio Blocking Vs Callback Mode

The program we developed to filter uses the python package pyaudio. The first iteration of the filtering code was designed in the blocking mode of pyaudio. The blocking mode requires an audio stream to be initialized. The audio stream is what defines the sampling rate, sample chunk size, and input/output channels of the audio device. Once the audio stream is defined, the sample data array needs to be read into the program using the command stream.read(CHUNK), and if there is overflow in data, then overflow can be set to false to avoid this with stream.read(CHUNK, expectation_on_overflow = False). The read data has to be converted into numerical values using the same numpy.fromstring() command above. The data is read into the program continuously using a while loop that runs until the program is stopped. In the while loop, we implemented the recursive filtering process such that frequency response to the filter is a bandwidth around the target resonant frequency while all other frequencies far from the target go to zero once the filtering is done the stream has to be called again to send out the data using the command stream.write(x). Initial tests of the program showed it working fine. However, after a while, we observed occasional glitches in the output on the oscilloscope. Glitches in the output could have been due to the filtering happening in a nested loop. Another possibility could be there is an internal audio buffer filling up, which wasn't accounted for in the code. The glitches could have occurred due to opening and closing the streams continuously in the while loop. The final

prospect is compiling or timing errors occurring within python, which we have not considered or accounted for in the code. [11,16,18]

To fix the glitching problem, we need to redesign the code such that we are running the filtering process in the callback mode of pyaudio. The callback mode does not require the code to read in or write out samples of data using the stream. The new mode initiates a callback function embedded in the stream definition. Thus, the PyAudio response is to automatically handle the input of sample data using the callback function. Inside the callback function, we can manipulate the sample data using the recursive filter defined above. Once the data is filtered, the callback function can return the filtered data without calling the stream. Thus, the definition of the callback enables the audio input/output to be automated by the code without requiring a loop that calls the stream to read and write data. Testing the new callback mode showed no glitching behavior occurred, and the Raspberry Pi module was able to handle continuous input, filtering, and output of data from the feedback system for prolonged periods. [11,16,18]

To make the dipole oscillate, I either had to physically spin it or use an external magnet and swipe by it quickly. Eventually, without any external gain applied to the dipole, the oscillations would exponentially decay. However, starting the callback version of the recursive filter after dipole oscillations get initiated, they are sustained indefinitely. However, to obtain self-sustained oscillatory behavior with the feedback loop and raspberry pi required me to get the dipole oscillating at the right frequency, the recursive filter to kick in at the right time, and the geometric gain to be just stronger than the exponential decay the dipole experiences. Subsequently, the code needed to be modified such that the input signal can be amplified and phase-shifted such that the amplitude of the dipole exponentially increases. Due to the non-zero nature of the input signal, even when there are no dipole oscillations, the correct amplification and phase can theoretically cause the dipole's oscillations to exponentially grow even if it is not initially moving.

4.3.3 Phase Shifting and Amplification

Initially, I tried to implement code to phase shift the input wave before sending it back out with the NumPy Fourier Transform packages. However, this method was computationally expensive to do, so another way of adding time delays to cause a phase shift in the signal had to be identified. Before adding additional time delays to cause phase shifts to occur, the

internal processing time delay of the recursive filter and pyaudio callback loops have to be determined. The internal phase shift of the raspberry pi module changes based on the input frequency. Equation 4.2 shows the relationship between the phase and the input frequency and an additional processing time delay.

$$\frac{d\phi}{df} = \frac{2Q}{f_0} + 2\pi\tau \tag{4.2}$$

Thus, using a signal generator and measuring the phase shift difference between the input and output signal would help to determine the internal processing time delay (τ). Measurements of phase difference due to small changes in the input frequency were obtained, giving a linear result. Using the slope from the data, we found (τ) to be 92 milliseconds. The 92 milliseconds corresponds to a phase shift of 0.578 radians which is about 33.12 degrees. Capturing the input/output signal after the first few seconds upon initiating the python code the oscilloscope shows that the 92-millisecond delay corresponds to a 3-4 cycle processing delay, as shown in figure 4.2.

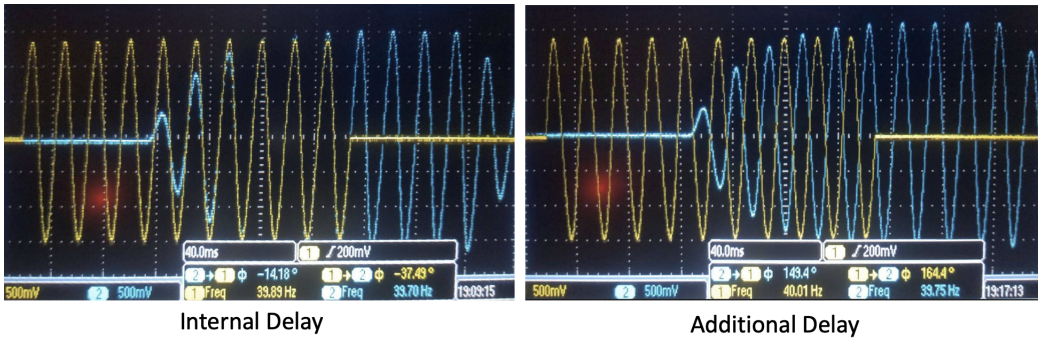

Figure 4.2: The two graphs show the initial wave in yellow and the processed outputted wave in blue on the oscilloscope. The input yellow wave gets processed and the blue output wave exponentially grows to match the input and exponentially decays once the input it off. The first graph shows the natural delay due to sending the input signal through the filter. The second graph shows additional delays added to the output signal by incriminating the delay parameter. There are 180 degrees of phase added to the system is observed on the oscilloscope.

Adding additional time delays with the delay parameter enables the output to be at any phase from 0°-360° or any multiple of those phases. The delay parameter can shift the data by a discrete number of zero points such that it adds to the 3-4 cycle delay by some phase factor.

Figure 4.2 shows a schematic of how the audio processing is complete, so a controlled phase shift is applied.

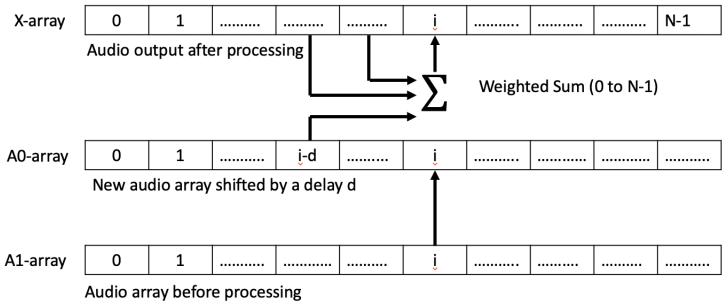

Figure 4.3: The audio processing enables a controllable phase shift to be get applied to the output wave. Utilizing an intermediate array preserve the output and input data as the python code is filtering and phase shifting. The A1 array is the input array from the pick-up coil. The summation represents the recursive filter, which calculates the filtered output data. The middle array (A0) is where the delay parameter gets implemented. The delay parameter shifts the current element of the signal that gets processed by the summation. This shifting is like adding zeros to the start of the input array. The zeros cause additional time delays to occur in the program leading to a different phase between the input and output arrays.

Each element from the input signal array gets sent to a new intermediate array one at a time as the filtering process is occurring. However, the filter is using the i-d element of the A0 array, and the previous two elements in the output array determine the new value to place in the output array for the current i index. The delay in processing the intermediate array is what causes a phase shift in the output dependant on the value of the delay parameter d. The phase shift is proportionality related to the delay and can be calculated using equation 4.3.

$$d\phi = 2\pi(d)\frac{f_0}{f} \tag{4.3}$$

Any phase shift can be predicted by the formula above for any resonant frequency, sampling rate, or delay value in the program. The correct phase, if chosen enables maximum electronic gain of the system or induces electronic loss of oscillatory behavior, essentially dampening the dipole. Any other phase will determine an amplitude of oscillation proportional to the maximum. In the electronic feedback system the total delay is $(\tau + d\phi)$. Now if the right phase is chosen, then the dipole will exponentially grow its oscillations from zero motion to the maximum amplitude of oscillation. However, the time required for the oscillatory motion to

grow from no visible motion may take up to five minutes. Additionally, the optimal phase of the system is not known thus every delay parameter would have to be tested to see which would start the dipole's oscillatory behavior from zero motion. Thus, the number of points in each sample array, CHUNK = 1024, times five minutes results in 85 hours of experimentally testing each delay parameter to determine the optimal phase. This is not an ideal method, so the input signal needs to be amplified before filtering. Inside the callback function before the filter, the input signal once converted into numerical values from bytes, can be multiplied by a scalar. The amplification acts as a kick-starter for the dipole since it only amplifies the amplitude of the resonant frequency being sent out after filtering and phase shifting. Thus, the time to observe the exponential growth of the dipole is cut down proportionally to the amplification. Once the dipole starts oscillating, its amplitude gets picked up in the input signal and then amplified again. This continues to happen until the maximum output of the sound card is hit, which is about 3.3 volts. The digital amplification also has limitations since the largest value a signed 16-bit number can be is ± 32767. Ultimately, now the dipole can self-start its oscillations and sustain them indefinitely using the feedback loop. [11,12,13]

4.3.4 Results of Raspberry Pi Integration

The integration of the raspberry pi module has shown to be capable of implementing a phase shift and amplification onto the dipole. However, the most significant result of the experiment was demonstrating that raspberry pi modules are capable of replacing more conventional electronic instruments used in physics research worldwide. We have proven that these raspberry pi modules can digitally amplify, phase shift, and filter signals continuously with minimal delay, as shown in figure 4.2. However, due to the method of phase-shifting implemented, the minimum and maximum frequency that theoretically can be shifted are calculated from RATE/f0. The input and output waves are capable of having a 360 phase shift, but that phase gets divided by the chunk size. If the rate in the program is 22050 and chunk size is set to 1024, then the maximum frequency that can have every possible degree of phase shift is roughly 21.5 Hz. Thus, this means that applying a phase shift with the delay parameter d at 21.5 Hz will result in about one degree of phase shift per integer delay. Running the program for higher frequencies will make the phase shift per delay more than one degree. The maximum frequency that would allow 360 degrees of phase shifting is 612.5 Hz. At 612.5 Hz, the program would have a 28.44-degree phase shift per delay parameter since 22050/612.5 = 36 and 1024/36 = 28.44 degrees

per delay. At 612.5 Hz, the integer delay range would be d from 0 - 36, and any other integers above that would result in greater than 360 degrees phase shift. Subsequently, the electronic feedback loop module is limited to working with low-frequency systems. Nonetheless, if at higher frequencies the phase-shifting can not be done digitally, the program is still capable of filtering and amplifying. As long as the user is capable of coding and understanding the dynamics of the audio processing methods, then they can utilize this electronic feedback loop module with any low-frequency system by interchanging the pick-up sensor and drive coil.

4.4 C-Clamp

As discussed earlier in chapter 3, the experimental setup shown in figure 3.1 produced pendulum motion at high gain values since the dipole was attached to a string. Thus, to limit the dipole's full motion on the XY plane and have no pendulum motion in the system, another method to position the dipole needed to be constructed. This method also requires that the dipole have very little friction, and the only dampening behavior is air resistance. Thus, using solid works, a C-clamp apparatus was designed with the specifications shown in figure 4.4.

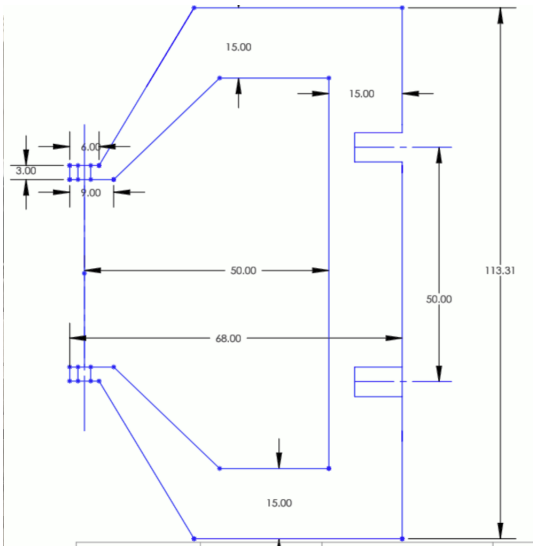

Figure 4.4: The image shows the custom-designed clamp for the dipole such that there is no more pendulum motion occurring when we go to higher frequencies. The C-clamp got designed in SolidWorks and machine-made. All measurements are in millimeters. The clamp needed to be strong enough to hold the dipole without breaking when the dipole oscillated, so the clamp was made of solid aluminum. However, having aluminum close to the dipole produces eddy currents, so to avoid that, the clamp was designed into a C bracket. The C bracket design reduced the interaction between the dipole and the aluminum while still making the clamp strong enough to hold the dipole. There are two screw holes in the tiny arms of the clamp enabling two jewel bearings to be screwed in. The other two screw holes on the side attach the clamp to an optical table.

Now that we can control the frequencies we drive the dipole at and the magnitude of the drive based on the orientations of the coils, we need to hold the magnet at a fixed position, still being able to rotate instead of hanging from a string. To do this, we have crafted in the Wesleyan University Machine shop a C-clamp of our design to hold the magnet but allow it to rotate with minimal friction with the help of jewel bearings. The schematic of the C-clamp is shown below in figure 4.4. Figure 4.5 shows how we attached the magnet to the clamp to give it minimal friction.

The C-clamp has two holes opposing each other, which are threaded to match the size of two jewel bearings that can be screwed into the holes. Jewel bearings are used in old wristwatches to reduce the friction of the mechanical moving parts. The jewel bearings were obtained from SwissJewel Company, and the specific model was VS-100, length 0.25 inches, a radius of .003/.004 inches, jewel size V1.25, and thread size of 5-40 UNS-2. The jewel bearings are utilized in the

C-clamp to enable a very sharp double-sided needle to rotate with minimal friction. The friction applied to the pin is determined by how tight the screws are placed. The other two holes in the C-clamp that are opposing each other are used to mount the clamp onto a stand which gets mounted onto an optical table such that when the dipole oscillates the clamp does not move. Placing the two magnets of the dipole on opposite sides of the needle held together by their magnetic attraction is not ideal since the needle thickness causes a V-shaped gap on one side. To get rid of the gap and make sure the dipoles are positioned flat across the needle on both sides a 3D printed holder was made in Wesleyan's Ideas Lab. The magnet holder is shaped like the magnets, but slightly bigger with a hole on both sides for the magnets to sit. In the middle is a thin film to separate the magnets by the thickness of the needle. A cylindrical hole runs through the 3D printed magnet holder, which the needle can fit through. An image of the new setup is shown in figure 4.4. [17]

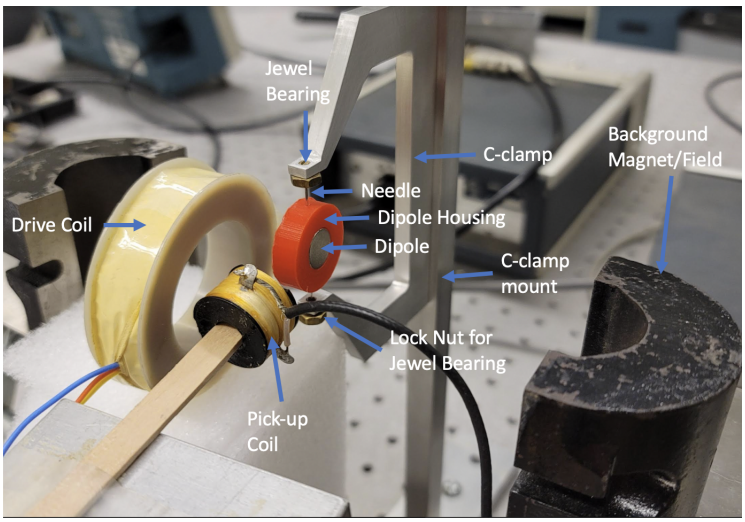

Figure 4.5: The image shows the custom-designed clamp for the dipole such that there is no more pendulum motion occurring when we go to higher frequencies. The two magnets that form the dipole sit opposite to one another inside the 3D printed housing shown in red/orange. There is a thin 3D printed membrane with a thickness slightly larger than the width of the needle on the inside. The needle slides through the housing in-between the two dipoles without causing any gaps. The needle sits on jewel bearings, which are screwed into the holes in the C-clamp. A lock nut is used to lock the jewel bearings in place since the oscillations of the dipole loosen them. The clamp is attached to a mount and positioned onto an optical table. The drive coil and pick-up coil are perpendicular to one another, so there is zero mutual inductance. The background magnets align the dipole along one axis. Since the dipole is not hanging on a string, there is no slight tilt, so there is no need to use another magnet to cancel the Earth's magnetic field in this setup.

Testing the new setup showed that the dipole is only oscillating in the XY plane of the C-clamp. All other vibrational modes that occur get dampened out by the C-clamp and the optical table. At very low frequencies the dipole has noticeable oscillations, but at higher frequencies, the dipole oscillates too quickly to be noticeable. However, the most efficient energy transfer from the drive coil to the dipole happens at the resonant frequency. At resonance, the amplitude exponentially increases the fastest as well. To maintain a specific amplitude the python program has to know the current amplitude of the dipole and change the parameters in the code to adjust and sustain the amplitude of oscillation. To sustain a specific amplitude of oscillation in real-time, an internal digital feedback loop needs to be designed. In the next section, we will go further in-depth into the digital feedback loop (Hover code) designed to dynamically sustain a specified amplitude.

4.5 Hover Code Development

By changing two variables in the code, the delay parameter (d) and the amplification applied to the input wave, we can control the amplitude level. However, to dynamically control the amplitude of the oscillating dipole, the current amplitude needs to be determined from the input signal, and the two variables need to be adjusted accordingly. The range of the delay parameter depends on the resonant frequency and sample size, while the range of the amplification is $\pm$ 32767. To sustain a specific amplitude of oscillation the input signal amplitude needs to be identified and compared to the specified amplitude of oscillation. If the amplitude of the input signal is too high or too low, then the delay parameter and the signal amplification are changed such that the dipole's oscillations are slowly brought to the desired amplitude. The amplitude can be determined by finding the maximum value in the Fourier transform of the signal or by reading the voltage input from the HiFi berry ADC/DAC sound card. In the program, I decided to calculate the amplitude with the Fourier transform of the signal using the formula 2*numpy.fft.rfft(signal)/CHUNK. Once the Fourier transform gets generated, taking the absolute value gives positive amplitudes, and this enables the python max function to compare all the amplitude values to find the maximum amplitude associated with the signal to use in the hover code. The calculation for the maximum amplitude can be placed inside the callback routine, so for every signal that gets processed, there is a maximum amplitude associated with it. Knowing the amplitude of oscillation for the dipole at any given time means an internal

feedback loop can be created to hover around a specified amplitude of oscillation. The hover code works by dynamically adjusting the variables and continuously checking the amplitude level with the desired target amplitude. Testing the hover code demonstrated that the dipole can lock in at a specified amplitude of 100. However, the program showed that the same hover amplitude could be reached for various delay parameters and amplification values. This is due to the current amplitude not being accurately calculated from the input chunk of data. The input chunk of data may not have the current max amplitude inside since it may not contain a full cycle of data. This means that the hover code is checking the current amplitude of the dipole, and it may be over the target of 100, but the sample data gives a value under 100. In this situation, the hover code readjusts to get back the target, inadvertently changing the optimal parameters. The internal feedback loop of the hover code amplitude measurement doesn't match the amplitude of the dipole with the external feedback loop. Thus, further investigation into how the dipole interacts with the electronic feedback loop and the variables in the python code would help to stabilize the hover code.

4.6 Summary of Findings

In this chapter, we have successfully shown the integration of a raspberry pi module with the two coils one dipole feedback loop system. The system was able to convert the electromagnetic oscillations into a digital signal array, which can be manipulated through array multiplication and addition such that the input signal can be amplified, filtered to a resonant frequency, and phase-shifted by a specified angle. This results in complete digital control over the oscillatory behavior of the dipole. Based on the resonant frequency we were able to identify the internal inherent phase shift/time delay that the filtering and amplifying programs imposed onto the input and output waves. Afterward, we incorporated a delay parameter that enables further phase shifting to occur, and based on the delay, resonant frequency, and sampling rate we can calculate the additional phase shifting/time delay the program can introduce into the system. Also, a C-clamp was custom machined to restrict the magnet from having pendulum oscillations. The magnet was positioned onto a needle placed inside of two jewel bearings on the opposite ends of the clamp allowing the needle and magnet to spin freely in the XY plane. The raspberry pi python-assisted controls demonstrated sustained oscillatory feedback behavior for 24+ hours. Thus, the system is durable enough to work for long periods unless disturbed

by an external factor or the power to the raspberry pi lost. Furthermore, we demonstrated that the raspberry pi python code can auto-start the dipole oscillations by amplifying the non-zero magnetic fluctuations that the pick-up coil measures from its surroundings. This information gets amplified by a scalar and then sent at the dipole. Eventually, after some time, the dipole starts oscillating and the pick-up coil input gets overridden by the dipole oscillations. The time it takes the dipole to auto-start its oscillations can be altered by changing the scalar that is multiplying the initial non-zero information from the pick-up coil or by changing the phase slightly to increase exponential growth or decrease exponential growth behavior of the oscillations. Finally, a hover code was developed that enables us to dynamically control the variables in the code such that a specific dipole oscillation amplitude can be sustained instead of letting the amplitude exponentially grow to the maximum value.

4.6.1 Future Experimental Work: Pseudo PT-Symmetric System

Even though we developed a raspberry pi module capable of filtering, phase shifting, and amplifying, the system has its limits. The maximum frequency this system can work for is about 612.5 Hz. For all frequencies above 612.5 Hz, a 360-degree phase shift is not possible. However, a frequency of 21.5 Hz is when each delay corresponds to one degree of phase shift between the input and output waves. Thus, the program can only filter low-frequency systems like the magnetic dipole system and can not filter higher frequencies. Consequently, finding another method of phase-shifting implemented into the system that is not dependent on integer delay values, then the raspberry pi module can be implemented into higher frequency systems. Furthermore, a new phase-shifting code can enable a one-to-one correlation in the delay parameter and the amount of phase-shifting happening to the input and output independent of the dipole's resonant frequency.

The equations of motion for the one dipole and two coil system were derived and explored. However, the equations of motion for the system with the integrated raspberry pi and python-assisted controls were not incorporated into the equations of motion. The motion of the dipole depends on the resonant frequency, delay parameter, sampling rate, chunk size initialized in the program, which indicates that they will affect the equations of motion. Therefore, a new set of equations of motion need to be derived and analytically analyzed for the raspberry pi module with the two coils, python code, and dipole. Once the equations of motion are re-derived and the

characteristic eigenfrequencies are identified then it will be much more clear how each parameter in the python code affects the motion of the dipole. The equations of motion may lead to a better method of developing a hover code resulting in the dipole having a predefined constant amplitude gain applied to it that can dynamically adjust to sustain itself even in the presence of another dipole with loss applied.

Combining all the dynamical results of our analytical and experimental observations, we can create a Pseudo PT-Symmetric two dipole system. To do so their needs to be another C-Clamp built with the same magnets in an identical 3D printed magnetic holder spaced apart from one another. This will result in symmetry in the system we wanted. The first dipole will have gain imposed by the feedback loop raspberry pi module and the second dipole will have loss either by introducing a conductor geometrically positioned in 3D space or another feedback loop raspberry pi module system having a phase parameter set to dampen the oscillatory behavior. Introducing a second feedback loop will require another optimization analysis to be done, like in section 3.4, of the orientation and position of the four coils relative to the two dipoles. Once the eight angles of the four coils are identified the Pseudo PT-symmetric two dipole system can be created. The system can also be created with the RL and RLC negative resistance circuit expanded in section 3.2. Any of these methods can be used to implement gain and or loss to either of the dipoles. The gain and loss can be controlled such that they are equal or one is greater than the other just like we did in the c program in chapter 2. After the set up of the pseudo-PT-symmetric two dipole system is complete then initial conditions can be explored to observe the deviations from the simulated results. Furthermore, if the equations of motion for each dipole is coupled together then we can once again analytically derive the γ_{PT} and $\gamma_{Critical}$ points such that they are dependant on the parameters of the geometry of the coils and the raspberry pi python code. However, I suspect that the γ_{PT} and $\gamma_{Critical}$ will only be scaled by the parameters similar to how the gain was scaled down when the two coils were geometrically restricted. Therefore, the underlying physics of the system will inherently stay the same if it can be shown that all geometric and program parameters scale away.

If the underlying physics in the experimental system is not the same in the simulated system then further studies will have to be conducted. However, regardless of whichever conclusion for the system is reached through further future research, the novel applications of the systems should be explored. The results of the two dipole system can be used to create a PT-symmetric

NMR instrument, a magnetic signaling device, or improving the sensitivity of current instrumentation. Due to the modular design of the feedback loop with the raspberry pi module, it can be customized by interchanging the coils with different sensors such that the code can be used on other electromagnetic, mechanical, acoustic, optical, biological, computational, thermal, and or electronic systems.

Chapter 5

Conclusion

Our analyses of the single-body magnetic PT-symmetric system fundamentally prove gain can be implemented physically onto a magnetic dipole with a feedback loop and modulated/filtered with python programs on a raspberry pi device. The optimal parameters for the maximum gain of 3.37 and zero mutual inductance get obtained mathematically using substitution within maximization programs. Figure 5 demonstrates the angular parameters identified for the single spin system with the two coils. The dipole must be perpendicular to the parallel field lines from the coils, so interactions are maximal. As we continue working with the one spin system, we will expand to solve the differential equations for a dual-spin system. However, geometrically the positioning of the second dipole can be along the same axis as the first one at a point where the field lines are parallel to each other and the second dipole.

Ultimately, the Raspberry Pi provides greater control of the gain applied to a single dipole through the feedback system. The filtering/phase shifting code developed has been proven to work with the hardware currently being used for the experiment. The code can filter any input wave sent into the raspberry pi, such as sound waves and electromagnetic waves at any resonant frequency. Optimization of the code for the two-dipole system is underway, but the general code can be utilized for any purpose requiring a recursive bandwidth filter. Upon assembly of the entire system, more data will matriculate, which will provide a greater understanding of the behavior of the magnetic systems. Exploration of unique modes or boundary conditions within the chaotic and symmetrical regions of the system will give insight into future applications for

novel technologies.

Bibliography

[1] AD Elster, ELSTER LLC. (2021). Dipole-dipole interaction. Retrieved April 21, 2021, from http://www.mri-q.com/dipole-dipole-interactions.html

[2] Bender, C. M. (2005). Introduction to PT-symmetric quantum theory. Contemporary Physics, 46(4), 277-292.

[3] BNC female to RCA male adapter. from https://www.showmecables.com/bnc-female-to-rca-male-adapter?

[4] BNC female to 3.5mm MONO male adapter. from https://www.showmecables.com/bnc-female-to-3-5mm-mono-male-adapter?

[5] Brett, M. (2016). Linear interpolation¶. Retrieved April 21, 2021, from https://matthew-brett.github.io/teaching/linear_interpolation.html

[6] Cheever, E. (2005). Swarthmore College - Fourth order Runge-Kutta. Retrieved April 21, 2021, from https://lpsa.swarthmore.edu/NumInt/NumIntFourth.html

[7] Griffiths, David J. (David Jeffery), 1942-. Introduction To Electrodynamics. Boston :Pearson, 2013.

[8] HiFiBerry DAC+ adc. (2021, April 19). Retrieved April 21, 2021, from https://www.hifiberry.com/shop/boards/hifiberry-dac-adc/

[9] Morin, D. (2007). The Lagrangian Method. Retrieved April 21, 2021, from https://scholar.harvard.edu/files/david-morin/files/cmchap6.pdf

[10] Numerical Recipes in C : the Art of Scientific Computing. Cambridge [Cambridgeshire] ; New York :Cambridge University Press, 1992.

[11] PyAudio Documentation. (2020). Retrieved June 5, 2020, from https://people.csail.mit.edu/hubert/pyaudio/docs/

[12] Raspberry Pi. Buy a Raspberry Pi 1 Model B+. Retrieved April 21, 2021, from https://www.raspberrypi.org/products/raspberry-pi-1-model-b-plus/

[13] Raspberry Pi Foundation. Raspberry Pi DOCUMENTATION. Retrieved April 21, 2021, from https://www.raspberrypi.org/documentation/

[14] Richard Fitzpatrick, R. (2013, April 8). Mass on a spring. Retrieved April 21, 2021, from http://farside.ph.utexas.edu/teaching/315/Waves/node3.html

[15] Schindler, J., et al. (2012). PTsymmetric electronics. Journal of Physics A: Mathematical and Theoretical, 45(44), 444029.

[16] Smith, S. W. 1997. The scientist and engineer's guide to digital signal processing. California Technical Publishing, USA.

[17] VS-100: Jewel Bearings, Vee Jewel BEARING ASSEMBLIES. (2019, March 04). Retrieved April 21, 2021, from https://www.swissjewel.com/product/jewel-bearings/vee-jewel-assemblies/vs-100/

[18] Wickert, M. (2018). Real-Time Digital Signal Processing Using pyaudio_helper and the ipywidgets. 91-98. 10.25080/Majora-4af1f417-00e.

[19] Meyer, C. (2015). Basic electronics: an introduction to electronics for science students (Second edition.). [Curtis A. Meyer].

Appendices

Appendix A

Experimental Parts

Images of each individual component used in the experimental portion of the lab is shown below. Combing all the hardware components, with the software code, and the analytical results enables full control over the motion of the dipole.

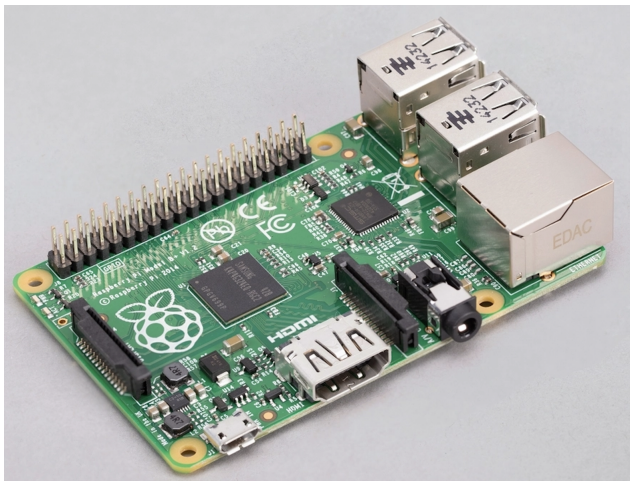

Figure A.1: Image of the Raspberry Pi B+ used in the experiment. The Pi contains 40 general-purpose input-output (GPIO) pins. It comes with 4 USB 2.0 ports, 100 Base Ethernet, power consumption of 0.5W - 1W, 1.4GHz 64-bit quad-core processor, dual-band wireless LAN, full-size HDMI, and Bluetooth 4.2/BLE. The raspberry pi also has a Micro SD port for loading your operating system and storing data and 5V/2.5A DC power input. [12]

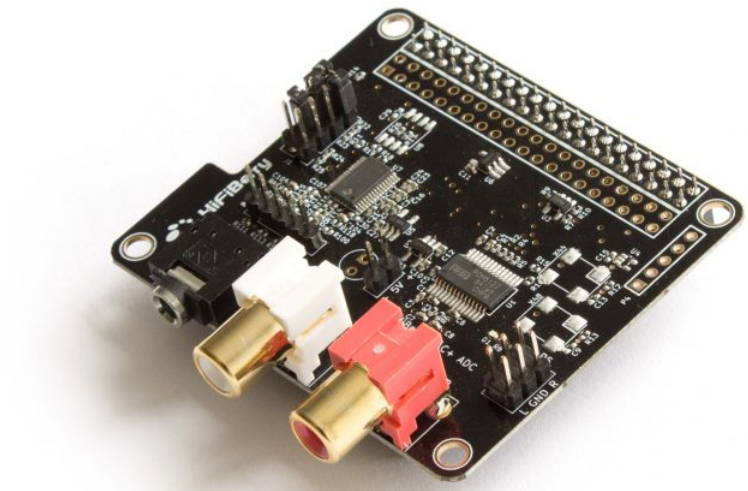

Figure A.2: Image of HiFi Berry DAC+ ADC which is compatible with the Raspberry Pi B+ model and sits on top of the GPIO pins on the Raspberry Pi. There is no additional cables or soldering needed to connect the parts. The HiFi Berry DAC+ ADC has an 8th-inch audio input port and two RCA audio output ports. Both the input and output support sample rates up to 192kHz. It enables stereo input and output through a 192kHz/24bit high-quality Burr-Brown DAC and ADC. Once placed on top of the raspberry pi the kernel needs to be updated with type the command dtoverlay=hifiberry-dacplusadc inside the config.txt of the terminal [8]

Figure A.3: The BNC Female to 3.5mm Mono Male Adapter which enables the input signal from the pick up coil to be sent into the HiFi Berry DAC+ ADC and then the Raspberry Pi. [4]

Figure A.4: The BNC Female to RCA Male Adapter which enables the output signal to get sent out through the RCA ports on the HiFi Berry DAC+ ADC to the drive coil. [3]

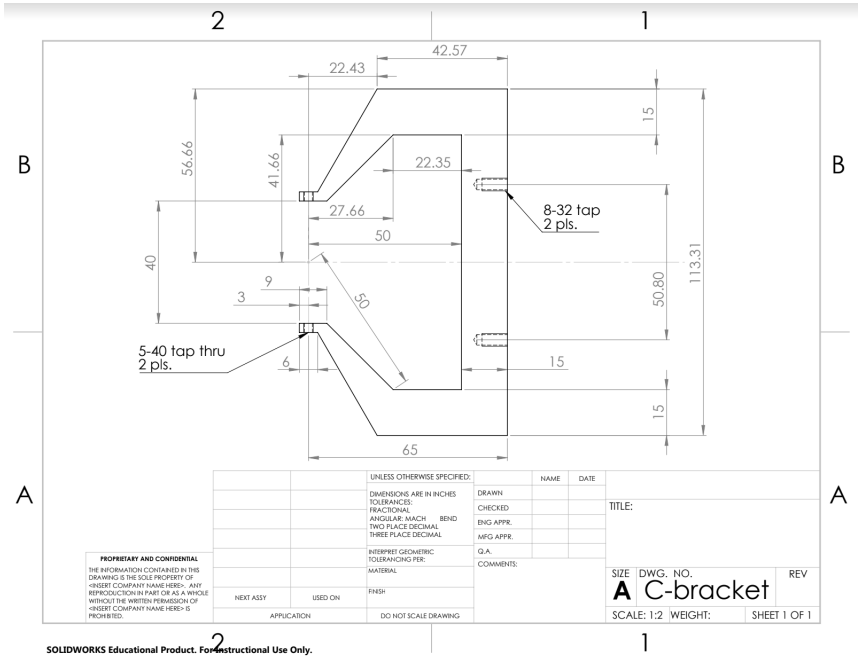

Figure A.5: The solid-works specifications for the C-clamp. All measurements are set to millimeters. The screw tap through are also labeled so they match the jewel bearings and the screws on the mount for the optical table.

Figure A.6: A jewel bearing is a plain bearing in which a metal spindle turns inside a jewel-lined pivot hole. The image above is the jewel bearings used to reduce the friction on the needles when the dipole was oscillating. The jewel model is VS-100, length of .250", a radius of .003/.004", jewel size of V1.25, and thread size of 5-40 UNS-2. [17]

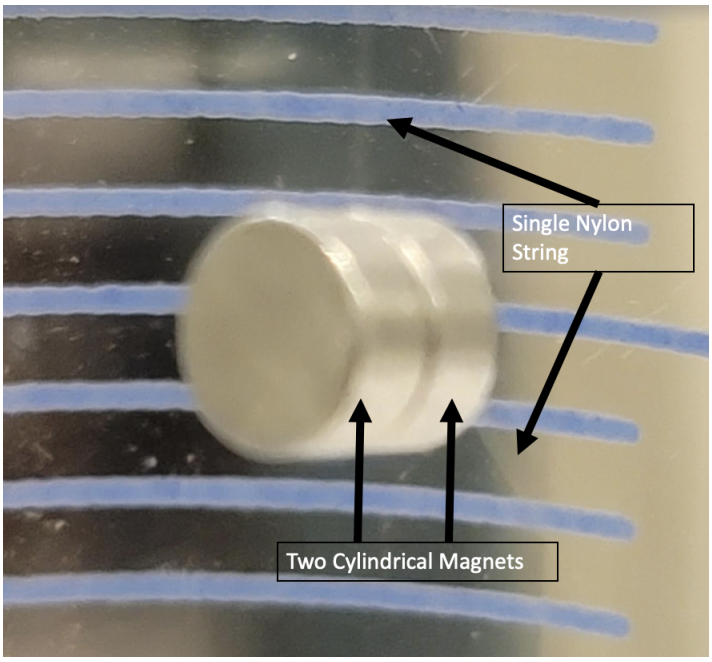

Figure A.7: Image of first dipole used inside of test tube. The two magnets making the dipole are clamped to a single piece of Nylon string. String was obtained from a PVC network communication cable.

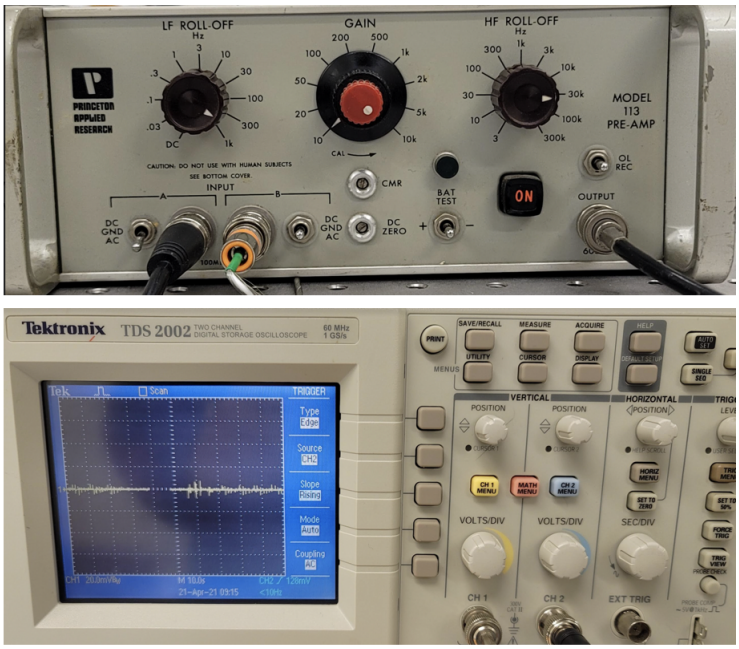

Figure A.8: The top image is of the 113 pre-amp used to amplify and filter the signals from the pick-up coil. The bottom image is the oscilloscope used to make measurements of the dipole oscillations and the input and output wave phase shifts.

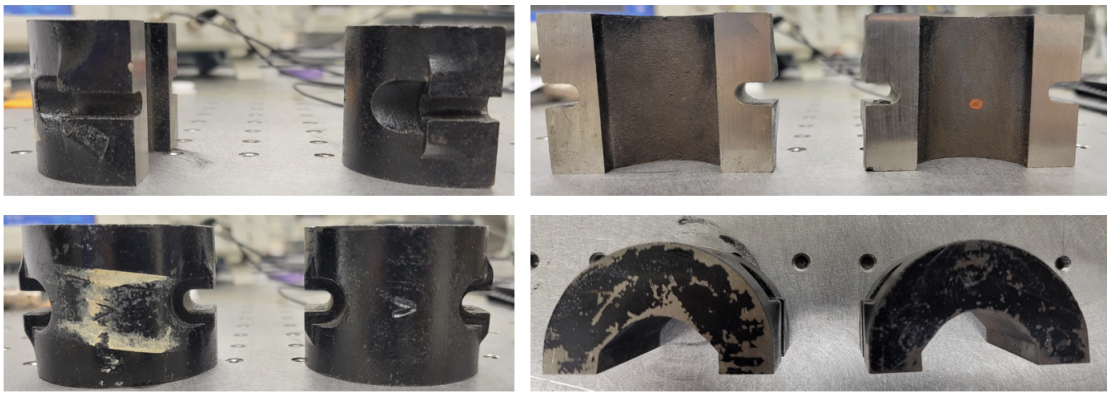

Figure A.9: The four images above show the two larger magnets used in the experiment to produce a background field to align the dipole.

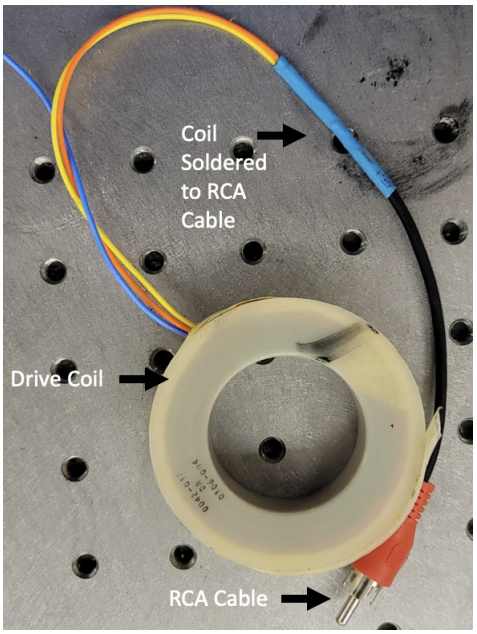

Figure A.10: Image of the drive coil with it's cables soldered to an RCA cable.

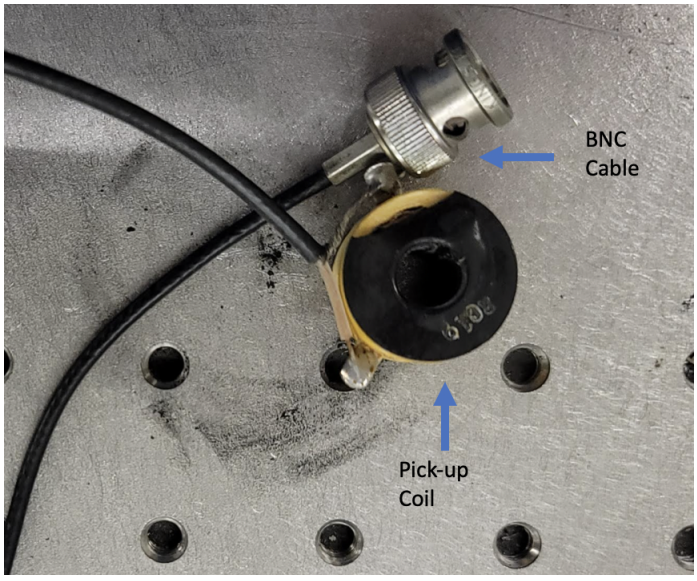

Figure A.11: Image of the pick-up coil used soldered to a BNC cable.

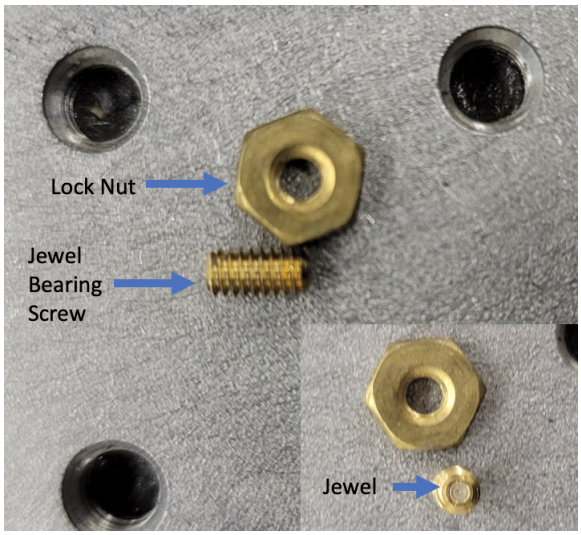

Figure A.12: The image shows a close up of the jewel bearing screw and the lock nut used to prevent it from loosening. The smaller image is showing the jewel bearing standing up with the jewel pocket showing. The other side of the screw is the flat head side used to turn the jewel bearing to increase or decrease the friction on the dipole.

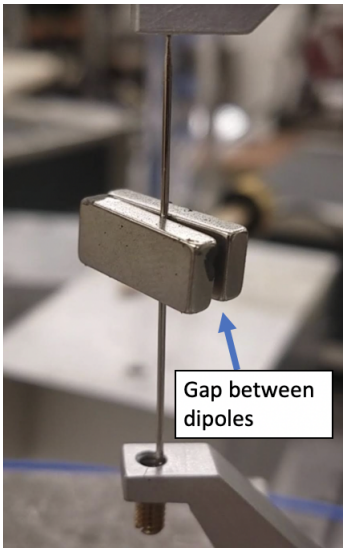

Figure A.13: The image shows the gap that is created between the two magnets of the dipole when they are placed between the needle.

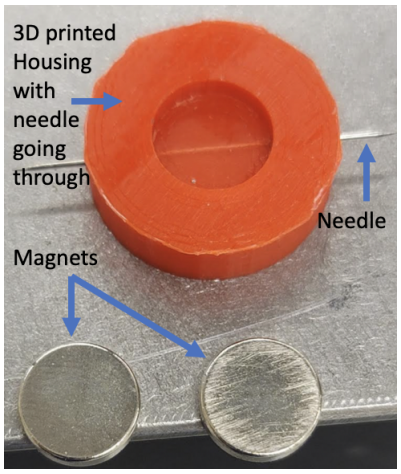

Figure A.14: The image shows the 3D printed magnet housing. The housing enables the double ended needle to pass through the center. The housing has a thin film in the center that is just the thickness of the needle. There are pockets the size of the two magnets on both sides. The magnets sit flush in the pockets with out creating a gap in the dipole.

Appendix B

Simulation Code

B.1 C Code

All graphs of the simulated equations of motion can be produced using the functions defined below and initial conditions. Using multiple functions in an iterative loop enables the program to reduce the errors while exploring various parameters and limits to the system. All the definitions listed below were used to explore the boundary graphs shown in figure 2.8 and 2.9. The code ramps through γ values and iterates θ_1 at each of the γ values. However, each individual function like the RK4 and the exact cycle with the limits on ω_2 and θ_2 from the eigen frequencies can be used individually to produce the phase plot and the parameter graphs shown in chapter 2.

B.1.1 RK4 Function

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

double functn(double a[]);
```

```
long amoeba(double p[][16], double y[], int ndim, double ftol);

void show_simplex(int ndim, double p[][16], double y[]);
void set_guesses(int ndim, double a[]);
void show_params(int ndim, double a[]);

double t1, t2, g, w1, w2, x0, dt, period1, error, ot2, ow2;


//The 4th order runge kutta function with time step input
void rk4step(double dt){
//initiating the variables needed for the 4th order Runge Kutta
double t1t, t2t, w1t, w2t, m1w1, m1w2, m1t1, m1t2, m2w1, m2w2, m2t1, m2t2;
double m3w1, m3w2, m3t1, m3t2, m4w1, m4w2, m4t1, m4t2, dt2;
    //below each step is calculated with equations of motion

        dt2 = dt/2;
        t1t = t1;
        t2t = t2;
        w1t = w1;
        w2t = w2;

        m1w1 = -sin(t2t)*cos(t1t)-2*sin(t1t)*cos(t2t)+g*w1t;
        m1w2 = -sin(t1t)*cos(t2t)-2*sin(t2t)*cos(t1t)-g*w2t;
        m1t1 = w1t;
        m1t2 = w2t;

        t1t = t1 + m1t1*dt2;
        t2t = t2 + m1t2*dt2;
        w1t = w1 + m1w1*dt2;
        w2t = w2 + m1w2*dt2;
```

```
    m2w1 = -sin(t2t)*cos(t1t)-2*sin(t1t)*cos(t2t)+g*w1t;
    m2w2 = -sin(t1t)*cos(t2t)-2*sin(t2t)*cos(t1t)-g*w2t;
    m2t1 = w1t;
    m2t2 = w2t;

    t1t = t1 + m2t1*dt2;
    t2t = t2 + m2t2*dt2;
    w1t = w1 + m2w1*dt2;
    w2t = w2 + m2w2*dt2;

    m3w1 = -sin(t2t)*cos(t1t)-2*sin(t1t)*cos(t2t)+g*w1t;
    m3w2 = -sin(t1t)*cos(t2t)-2*sin(t2t)*cos(t1t)-g*w2t;
    m3t1 = w1t;
    m3t2 = w2t;

    t1t = t1 + m3t1*dt;
    t2t = t2 + m3t2*dt;
    w1t = w1 + m3w1*dt;
    w2t = w2 + m3w2*dt;

    m4w1 = -sin(t2t)*cos(t1t)-2*sin(t1t)*cos(t2t)+g*w1t;
    m4w2 = -sin(t1t)*cos(t2t)-2*sin(t2t)*cos(t1t)-g*w2t;
    m4t1 = w1t;
    m4t2 = w2t;

// the final output is the average of the steps
    t1 = t1 + ((m1t1+2*(m2t1+m3t1)+m4t1)/6)*dt;
    t2 = t2 + ((m1t2+2*(m2t2+m3t2)+m4t2)/6)*dt;
    w1 = w1 + ((m1w1+2*(m2w1+m3w1)+m4w1)/6)*dt;
    w2 = w2 + ((m1w2+2*(m2w2+m3w2)+m4w2)/6)*dt;

    }
```

B.1.2 Print Cycle No Interpolation

```
void cycleprint(double dt, double x0){
        //this is where the outputs are printed without interpolation


        t1 = x0;
        t2 = (-(((g*g+sqrt(g*g*g*g-8.0*g*g+4.0))/2.0))*t1);
        w1 = 0;
        w2 = (g*(((g*g-4.0+sqrt(g*g*g*g-8.0*g*g+4.0))/2.0))*t1);

    // open a text to save the data to graph
        FILE *in_file  = fopen("tests.txt", "w");

    //run through RK4 function for half the phase
        while (w1 <= 0 ){
        rk4step(dt);
    //print to the file
        //printf("%lf\t%lf\t%lf\t%lf\n", t1, t2, w1, w2);
        //fprintf(in_file, "%lf\t%lf\t%lf\t%lf\n", t1, t2, w1, w2);


        }
    //run through RK4 function for other half the phase
        while (w1 > 0){

        rk4step(dt); // run the rk4 function and print terms below
    //print to the file
        //printf("%lf\t%lf\t%lf\t%lf\n", t1, t2, w1, w2);
        //fprintf(in_file, "%lf\t%lf\t%lf\t%lf\n", t1, t2, w1, w2);


        }
        //close the file and save to graph
        fclose (in_file);
        }
```

B.1.3 Exact Conditions Function

```
void exactconditions (double dt, double x0)
{//this does one cycle without printing but with interpolation

double t10, t20, w10, w20, period1, intertheata1, fraction1, intertheata2;
double counter1, interw1, interw2, x1, y1, y0;

        t1 = x0;
        w1 = 0;

        x1 = t2; //this needs to be changed to the new theata2 times x0
        y0 = w1;
        y1 = w2;//this needs to be changed to the new w2 times x0

        FILE *in_file  = fopen("tests1.txt", "w");

        while (w1 <= 0 )
        {
        rk4step(dt);
        counter1++;
        fprintf(in_file, "%lf\t%lf\t%lf\t%lf\n", t1, t2, w1, w2);
        }

        while (w1 > 0)
        {
        t10 = t1;
        t20 = t2;
        w10 = w1;
        w20 = w2;
        rk4step(dt);
        fprintf(in_file, "%lf\t%lf\t%lf\t%lf\n", t1, t2, w1, w2);
        counter1++;
```

```
}
fclose (in_file);

///this is where the interpolation happens
fraction1 = (-w10/(w1-w10));
period1 = counter1 * dt + fraction1 * dt;
interw1 = w10 + fraction1*(w1-w10);
interw2 = w20 + fraction1*(w2-w20);
intertheata1 = t10 + fraction1*(t1-t10);
intertheata2 = (t20 + fraction1*(t2-t20));

Perror = (1/x0)*(sqrt(1/4*(intertheata1 - x0)*(intertheata1 - x0)+
(intertheata2 - x1)*(intertheata2 - x1)+(interw1 - y0)*(interw1 - y0)+
(interw2-y1)*(interw2 -y1)));
}
```

B.1.4 Exact Cycle

```
double excatcyle (double t21, double w21)
//exact conditions needs to be analyzed very carefully
{
///this does one cyle without printing but with interpolation
double t10, t20, w10, w20, period1, intertheata1, fraction1, intertheata2;
double counter1, interw1, interw2, x1, y1, y0, Perror;

t1 = x0;
w1 = 0;
t2 = t21;
w2 = w21;
Perror = 0;
//FILE *in_file  = fopen("tests1.txt", "w");
while (w1 <= 0 )
{
```

```
rk4step();
counter1++;
//fprintf(in_file, "%lf\t%lf\t%lf\t%lf\n", t1, t2, w1, w2);
if (fabs(t2)>4||fabs(t1)>4){
        Perror = 1;
        goto skip;
}

}
while (w1 > 0)
{
t10 = t1;
t20 = t2;
w10 = w1;
w20 = w2;
rk4step();
//fprintf(in_file, "%lf\t%lf\t%lf\t%lf\n", t1, t2, w1, w2);
counter1++;
}
skip:

//fclose (in_file);
//printf("\n%lf, %lf, %lf, %lf, %lf\n\n\n",t10, t20, w10, w20, counter1);
//printf("\n%lf\n\n",counter1);
if (Perror == 0){
///this is where the interpolation happens
fraction1 = (-w10/(w1-w10));
period1 = counter1 * dt + fraction1 * dt;
interw1 = w10 + fraction1*(w1-w10);
interw2 = w20 + fraction1*(w2-w20);
intertheata1 = t10 + fraction1*(t1-t10);
intertheata2 = (t20 + fraction1*(t2-t20));
```

```
        //printf("The fraction and period is \t%lf\t%lf\n", fraction1, period1);

        //printf("Interpolated w1 is %lf\nInterpolated w2 is
        %lf\nInterpolated theata1 is %lf\nInterpolated theata2 is %lf\n",
        interw1, interw2, intertheata1, intertheata2);

        Perror = (1/(x0))*sqrt(0.25*((intertheata1 - x0)*(intertheata1 - x0)+
        (intertheata2 - t21)*(intertheata2 - t21)+
        (interw2-w21)*(interw2 - w21)));
        //printf("\nPercent error is %lf\n", Perror);
        }
                        error = Perror;
                        return Perror;
        }
```

B.1.5 Function

```
double functn(double a[])
{
// This 3D test function is a sharp spherical shell well at radius 1 with
//      a weak minimum in the 111 direction. Sigma is the width, dd is the
//      weak tilting toward the 111 direction.  -- Min at ~ (0.577, 0.577, 0.577)
         return excatcyle(a[0], a[1]);
// Here's an alternatve simpler parabolic minimum -- min of 1.0 at (2, 3, 4)
//     return 1.0+(a[0]-2.0)*(a[0]-2.0) + (a[1]-3.0)*(a[1]-3.0) +
(a[2]-4.0)*(a[2]-4.0);
}
```

B.1.6 Ameoba

```
long amoeba(double p[][16], double y[], int ndim, double ftol)

{
```

```
double const alpha = 1.0, beta = 0.5, gamma = 2, delta = 0.5;
double pr[16], prr[16], pbar[16];     // these will be used ndim +1
double ypr, yprr;
int i, j, ilo, ihi, inhi;
int mpts = ndim +1;
long iter = -1;
double rtol = 10.;
//printf("\n");

while(rtol > ftol)
{
        iter = iter + 1;
        if(iter >10000)
        {
                printf("\nNot converging! Stopped at 10000 iterations.\n");
                return 10000;
        }
        ilo = 1;

// IDENTIFY HIGHEST, NEXT HIGHEST, LOWEST
        if(y[0] > y[1])
        {
                ihi = 0;
                inhi = 1;
        }
        else
        {
                ihi = 1;
                inhi = 0;
        }

        for(i = 0; i < mpts; i++)
```

```
        {
                if(y[i] < y[ilo])
                {
                        ilo = i;
                }

                if(y[i] > y[ihi])
                {
                        inhi = ihi;
                        ihi = i;
                }
                else if(y[i] > y[inhi])
                {
                        if(i != ihi)
                        {
                                inhi = i;
                        }
                }
        }

// evaluation of tolerance -- based on the difference between the highest
//and lowest simplex values

        rtol = fabs(y[ihi] - y[ilo]);
        // actual function error between hi and lo points
// rtol = 2.0*fabs(y[ihi] - y[ilo])/(fabs(y[ihi]) + fabs(y[ilo]));

// error relative to function value
// report the progress
//printf("%ld  hi = %14.6E  lo = %14.6E  rtol = %10.4E\n", iter,
y[ihi], y[ilo], rtol);
// AVERAGE OF ALL VERTICES BUT THE HIGHEST.  THIS IS THE CENTER OF THE
```

```
// "FACE" ACROSS FROM THE HIGHEST AND THE RAY ALONG WHICH TO EXPLORE
        for(j = 0; j< ndim; j++)
        {
                pbar[j] = 0;
        }
        for(i = 0; i< mpts; i++)
        {
                if(i != ihi)
                {
                        for(j = 0; j < ndim; j++)
                        {
                                pbar[j] = pbar[j] + p[i][j];
                        }
                }
        }

// REFLECT THE HIGHEST POINT THROUGH THIS FACE WITH A FACTOR ALPHA
        for(j = 0; j < ndim; j++)
        {
                pbar[j] = pbar[j]/(double)ndim;
                pr[j] = pbar[j] - alpha*(p[ihi][j] - pbar[j]);
        }
// CHECK the FUNCTION
        ypr = functn(pr);
// IF IT'S BETTER, TRY AGAIN WITH AN ADDITIONAL FACTOR OF GAMMA

        if(ypr <= y[ilo])
        {
                for(j = 0; j < ndim; j++)
                {
                        prr[j] = gamma*pr[j] + (1.0 - gamma)*pbar[j];
```

```
                }
// CHECK FUNCTION
                yprr = functn(prr);

// IF THIS IS BETTER, REPLACE THE HIGHEST
                if(yprr < y[ilo])
                {
                        for(j = 0; j < ndim; j++)
                        {
                                p[ihi][j] = prr[j];
                        }
                        y[ihi] = yprr;
                }
// OTHERWISE USE THE FIRST EXTRAPOLATION
                else
                {
                        for(j = 0; j < ndim; j++)
                        {
                                p[ihi][j] = pr[j];
                        }
                        y[ihi] = ypr;
                }
// IF THE FIRST FAILED TO BETTER THE NEXT HIGHEST

        }
        else if(ypr > y[inhi])
        {

// REPLACE THE HIGHEST IF ITS BETTER
                if(ypr < y[ihi])
                {
                        for(j = 0; j < ndim; j++)
```

```
                {
                        p[ihi][j] = pr[j];
                }
                y[ihi] = ypr;
        }

// EITHER WAY, LOOK FOR A BETTER POINT BY CONTRACTING TOWARD THE AVERAGE

        for(j = 0; j < ndim; j++)
        {
                prr[j] = beta * p[ihi][j] + (1.0 - beta) * pbar[j];
        }

// CHECK FUNCTION
        yprr = functn(prr);

// ACCEPT AN IMPROVEMENT

        if(yprr < y[ihi])
        {
                for(j = 0; j < ndim; j++)
                {
                        p[ihi][j] = prr[j];
                }
                y[ihi] = yprr;
        }

// HIGH POINT PERSISTS — CONTRACT ABOUT THE LOWEST

    else
    {
                for(i = 0; i < mpts; i++)
```

```
                {
                        if(i != ilo)
                        {
                                for(j = 0; j < ndim; j++)
                                {
                                        pr[j] = delta * (p[i][j] + p[ilo][j]);
                                        p[i][j] = pr[j];
                                }
                                y[i] = functn(pr);
                        }
                }
        }
}

// IF ORIGINAL REFLECTION GAVE A MIDDLE SIZE, REPLACE HIGH POINT
        else
        {
                for(j = 0; j < ndim; j++)
                {
                        p[ihi][j] = pr[j];
                }
                y[ihi] = ypr;
        }
}
// show_simplex(ndim, p, y);
return iter;
}  // end of amoeba
```

B.1.7 Show Simplex

```
void show_simplex(int ndim, double p[][16], double y[])
{
        int i, j;
```

```
        printf("\nSimplex points...\n");
        for(i = 0; i <= ndim; i++)      // simplex index
        {
                printf("\n(");
                for(j = 0; j < ndim; j++)  // parameter space coordinate
                {
                        printf("p[%d][%d] = %14.6E,",i, j, p[i][j]);
                }
                printf(")  y[%d] = %14.6E",i ,y[i]);
        }
        printf("\n");
}
```

B.1.8 Show Parameters and Set Guesses

```
// fills the parameter array with guesses
void set_guesses(int ndim, double a[])
{
        int i;

        printf("\nEnter initial guesses for the %d function parameters
        ...\n\n", ndim);
        for(i = 0; i < ndim; i++)
        {
                printf("a[%d] = ",i);
                scanf("%lf", &a[i]);
        }
}

// shows the parameter array
void show_params(int ndim, double a[])
{
```

```
        int i;
        for(i = 0; i < ndim; i++)
        {
                printf("  a[%d] = %14.6E\n", i, a[i]);
        }
}
```

B.1.9 Ameobacycle

```
void ameobacycle (){
double sz0 = .25;

double counter;
double p[16][16], y[16];
double a[16], b[16], sz[16];
double ftol, guessf;
int ndim, i, j, satisfied, imin, iter;
char resp[8];

ndim = 2;
ftol = 1.0E-14;
satisfied=0;

        for(i = 0; i < ndim; i++)
        {
                sz[i] = sz0;  // all dimansion the same

        }

        for(i = 0; i <= ndim; i++)   // note <= for ndim + 1 simplex points
        {
                for(j = 0; j< ndim; j++)
                {
```

```
                        if(j == i)          // bump this one up by sz
                        {
                                p[i][j] = a[j] + sz[j];
                        }
                        else
                        {
                                p[i][j] = a[j];
                        }
                        b[j] = p[i][j];
                }
// y stores the current function values at each simplex point
                y[i] = functn(b);
        }
// All set up. Run the minimization...
        iter = amoeba(p, y, ndim, ftol);

// SHOW RESULTS (SMALLEST OF THE SIMPLEX POINT)
//    show_simplex(ndim, p, y);
// Use to show all of the simplex content for debugging.

// find the index of the smallest valued simplex point
        imin = 0;
        for(i = 0; i <= ndim; i++)
        {
                if(y[i] < y[imin]){imin = i;}
        }
// recover the best parameters from the above point
        for(i = 0; i < ndim; i++)
        {
                a[i] = p[imin][i];
        }
```

```
        //printf("\n Minimized parameters after %d simplex steps:\n\n", iter);
                //show_params(ndim, a);
                //printf("\n  Function value = %14.6E\n",functn(a));
}
```

B.1.10 Gamma Ramp

```
void gammaramp (){

for (g = .001; g < .732; g = g +.001){
for (x0 = .01; x0 < 1.57; x0 = x0 + .001){

        ameobacycle();

        if (error == 1){
                goto gammap;
        }
        }
        gammap:
                printf("%lf\t%lf\n", g, x0);
                t2 = ot2;
                w2 = ow2;
        }
//printf("%lf\t%lf\t%lf\t%lf\t%lf\t%lf\n", x0, t2, w2, m, Perror, counter);
//fprintf(in_file, "\n%lf\t%lf\t%lf\t%lf\t%lf\n", x0, t2, w2, m, Perror);
}
```

B.1.11 Main Function

```
int main()
{
                dt = .01;
                w2 = (g*(((g*g-4.0+sqrt(g*g*g*g-8.0*g*g+4.0))/2.0))*x0);
```

```
        t2 = (-(((g*g+sqrt(g*g*g*g-8.0*g*g+4.0))/2.0))*x0);

        ot2 = t2;
        ow2 = w2;

        gammaramp();
}
```

Appendix C

Raspberry Pi Integration Codes

Complete audio processing to complete a feedback loop can be done with the code below with the raspberry pi setup shown in chapter 4. The callback mode pyaduio is shown below. Blocking mode is similar setup but requires writing and reading input and output signals.

C.1 Callback Mode Code

```
# All necessary imports for the program [11]
import pyaudio
import time
import numpy as np
import math
import cmath
from tkinter import Tk, Button, Text, END, Label
from time import sleep


##############################################################################
"""Rate/f0 = 22050/f0. The lowest f0 that gives 360 degree phase
control is 22 Hz. The maximum f0 would have to be 22050 leading to 1
```

```
unit of delay giving a 360 degree so controlling phase won't be possible
beyond the range of 22 Hz and 600 Hz. A new method of phase shifting
needs to be implemented to accommodate other frequencies. An increased
Chunk size or Rate might be away to solve the issue."""
##############################################################################


# channel needs to be set to 1 for sound card
# Processing rate, sample size (chunk), and the type of audio stream is defined
CHANNELS, RATE, CHUNK, FORMAT = 1, 22050, 1024, pyaudio.paInt16

p = pyaudio.PyAudio() # This is were pyaudio system is abbreviated to p

# The values below are for the recursive filter
f, f0, Q = RATE, 16.44, 5
c = -math.exp((-2*math.pi*f0)/(Q*f))
b = 2*math.exp((-math.pi*f0)/(Q*f))*math.cos(2*math.pi*f0/f)
a = (1-b-c)/(Q)

# The two empty array of zeros CHUNK times for filtering and phase shifting
x = np.zeros(CHUNK)
a0 = np.zeros(CHUNK)
d = 0 # this is the delay factor that phase shifts output wave
amp = 32700 #This is the amplification value

#The callback definition allows the audio stream to handle inputting and outputting
#The blocking mode requires the program to read and write data inside a while loop
#Using while loop creates timing errors, but callback fixes them automatically
def callback(in_data, frame_count, time_info, flag):
    #The global variables pass values from the callback to outside callback
    global a0, x, a, b, c, f0, Q, d
```

```
    # convert data to numpy array
    audio_data = np.fromstring(in_data, dtype=np.int16)

    #Amplify the signal by a scalar
    audio_data = audio_data * amp

    # Multiply data by first parameter of filter a to speed up filter
    a1 = audio_data * a

    # Start the recursive filter loop
    for i in range(0, CHUNK):
        if i == d:
            a0=a1
        # Run the recursive filter equation
        # Implement a delay in the input array
        x[i] = a0[i-d] + b * x[(i-1)%CHUNK] + c * x[(i-2)%CHUNK]

        # output data at this point becomes np.float64
    #Convert the data into 16 bit integers and a string of bits
    audio = x.astype(np.int16).tostring()

    return (audio, pyaudio.paContinue) # returning data and continue the callback

# setting up the stream with callback and predetermined values above
stream = p.open(format=FORMAT,
                channels=CHANNELS,
                rate=RATE,
                output=True,
                input=True,
                stream_callback=callback,
                frames_per_buffer=CHUNK)
```

```
stream.start_stream() # starting the stream

#Creating buttons and Tkinter to change variables in real time
def f0_add1(value):
    global f0, a, b, c
    f0 = f0 + value
    c = -math.exp((-2*math.pi*f0)/(Q*f))
    b = 2*math.exp((-math.pi*f0)/(Q*f))*math.cos(2*math.pi*f0/f)
    a = (1-b-c)/(Q)
    f0_str.config(text="F0 = " + str(f0))
def d_add1(value):
    global d
    d = d + value
    d_str.config(text="D = " + str(d))
def Q_add1(value):
    global Q, a, b, c
    Q = Q + value
    c = -math.exp((-2*math.pi*f0)/(Q*f))
    b = 2*math.exp((-math.pi*f0)/(Q*f))*math.cos(2*math.pi*f0/f)
    a = (1-b-c)/(Q)
    Q_str.config(text="Q = " + str(Q))
def f0_sub1(value):
    global f0, a, b, c
    f0 = f0 - value
    c = -math.exp((-2*math.pi*f0)/(Q*f))
    b = 2*math.exp((-math.pi*f0)/(Q*f))*math.cos(2*math.pi*f0/f)
    a = (1-b-c)/(Q)
    f0_str.config(text="f0 = " + str(f0))
def d_sub1(value):
    global d
    d = d - value
    d_str.config(text="D = " + str(d))
```

```
def Q_sub1(value):
    global Q, a, b, c
    Q = Q - value
    c = -math.exp((-2*math.pi*f0)/(Q*f))
    b = 2*math.exp((-math.pi*f0)/(Q*f))*math.cos(2*math.pi*f0/f)
    a = (1-b-c)/(Q)
    Q_str.config(text="Q = " + str(Q))

root = Tk()

f0_str = Label(root)
f0_str.pack()
d_str = Label(root)
d_str.pack()
Q_str = Label(root)
Q_str.pack()
f0_str.config(text="F0 = " + str(f0))
Q_str.config(text="Q = " + str(Q))
d_str.config(text="D = " + str(d))

Button(root, text='F0 + 1',command=lambda *args: f0_add1(1)).pack()
Button(root, text='F0 - 1',command=lambda *args: f0_sub1(1)).pack()
Button(root, text='D + 1',command=lambda *args: d_add1(1)).pack()
Button(root, text='D - 1',command=lambda *args: d_sub1(1)).pack()
Button(root, text='Q + 1',command=lambda *args: Q_add1(1)).pack()
Button(root, text='Q - 1',command=lambda *args: Q_sub1(1)).pack()
```