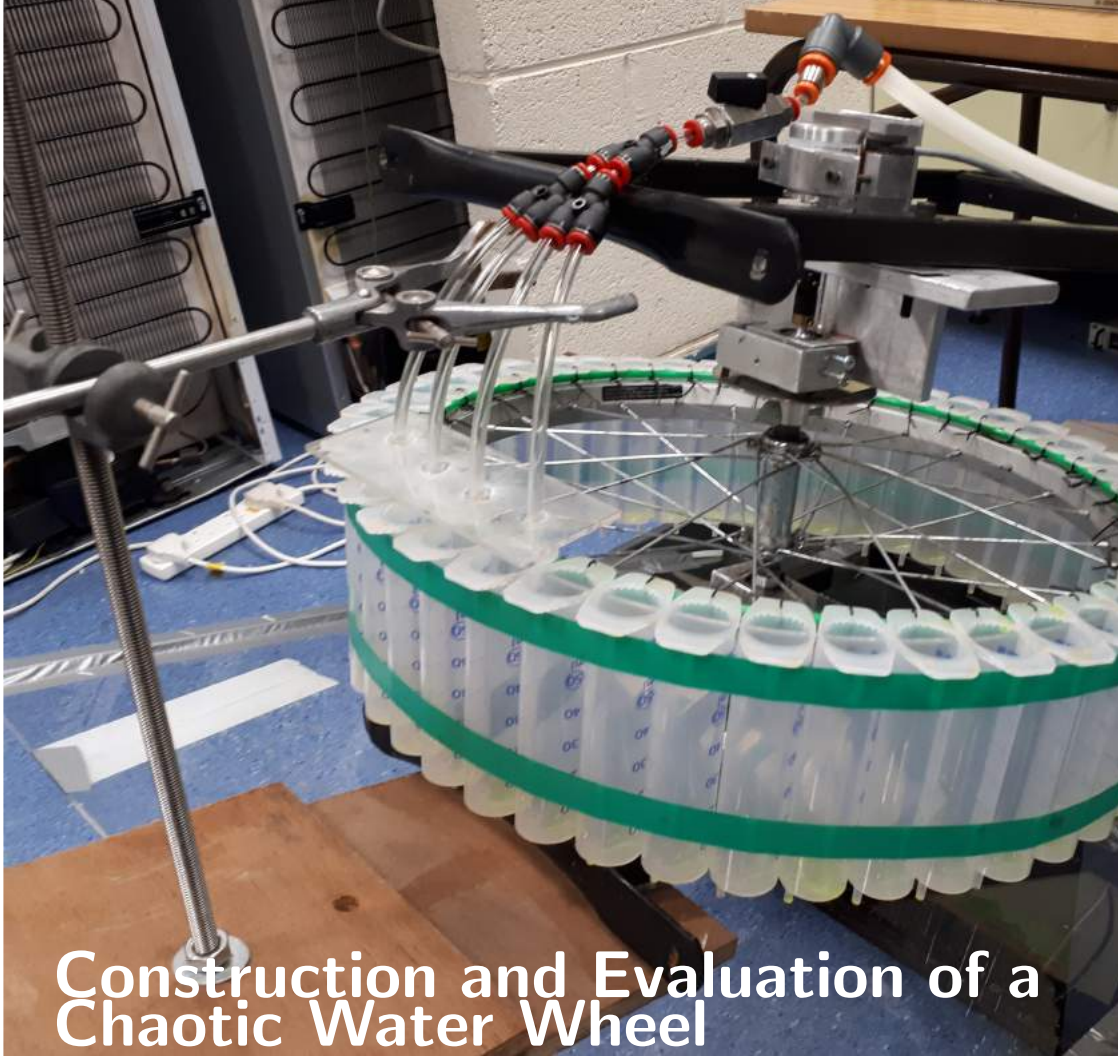


Construction and Evaluation of a Chaotic Water Wheel

Maynooth University
National University of Ireland Maynooth

Sean Cummins
Department of Experimental Physics
seancummins16@gmail.com

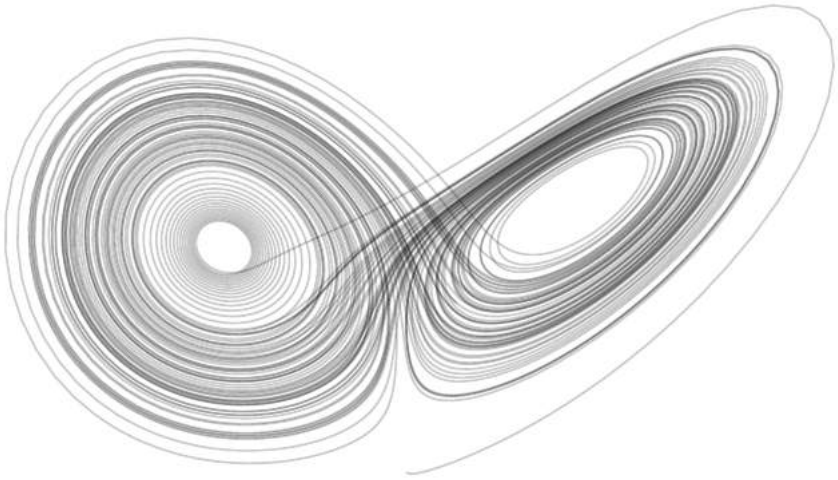
Dr Michael Cawley
Department of Experimental Physics
Michael.Cawley@nuim.ie

Abstract

The aim of this project is the evaluation and construction of a Malkus-Howard-Lorenz chaotic water wheel. We start from an unideal 4-bucket water wheel. While exploring the range of motion possible using a computer program, we proceed to build and run tests on an experimental 4-bucket water wheel apparatus. We attempt to match the motion observed with the software predictions. From this we move on to the more well known Malkus-Howard-Lorenz chaotic water wheel whose governing equations are a subset of the Lorenz system. We construct a 45 bucket water wheel using syringes tightly bound in a ring with as few gaps as possible. We integrate the Lorenz system to predict the apparatus's motion. Furthermore we numerically generate a map of the type of motion expected for a particular choice of parameter values using Lyapunov exponents. Again we seek to match the type of motion generated by a particular choice of experimental parameter values set on the apparatus with the same type of motion predicted by the parameter map. This report is as self contained and self sufficient as possible with significant background material, detailed explanations of the design and construction process of both apparatuses, evaluation of data, suggested improvements and code used.

Contents

1 Introduction



Chaos: When the present determines the future, but the approximate present does not approximately determine the future.

Edward Lorenz (1917 - 2008)

The story[1] is as famous as the field of research itself. No doubt it has been told an innumerable amount of times and formed the introduction of countless papers on the topic but it always feels like the most appropriate place to begin....

MIT, Winter, 1961 and mathematician and meteorologist Edward Norton Lorenz is running a climate model whose convection equations (in simplified form) consist of :

$$\dot{x} = \sigma(y - x)$$

$$\dot{y} = \rho x - y - xz$$

$$\dot{z} = xy - bz$$

where σ, ρ and b > 0 are parameters. The three dimensional system is itself a simplified model of a twelve-variable computer weather model of convection rolls in the atmosphere. Lorenz decided to rerun one of the sequences and (to save time and space) rounded the 6 digit inputs to 3 digits.

However the Royal McBee LGP-30 electronic computer churned back out quite a surprise. Expecting to see a similar trajectory plot in the phase space as the first run, Lorenz found himself staring at a trajectory that bared little resemblance to the one before. Both trajectory looked initially identical but they both quickly diverged on radically different paths. Clearly there was a something wrong.

Initially suspecting there was a problem with the computer, Lorenz quickly realised that it was the very act of rounding the inputs that created the two drastically different trajectories. Lorenz realised that such *sensitivity to initial conditions* would make long-term predictions of the weather impossible. Lorenz published his conclusions in the now historic paper *"Deterministic Nonperiodic Flow"*[2] in 1963.

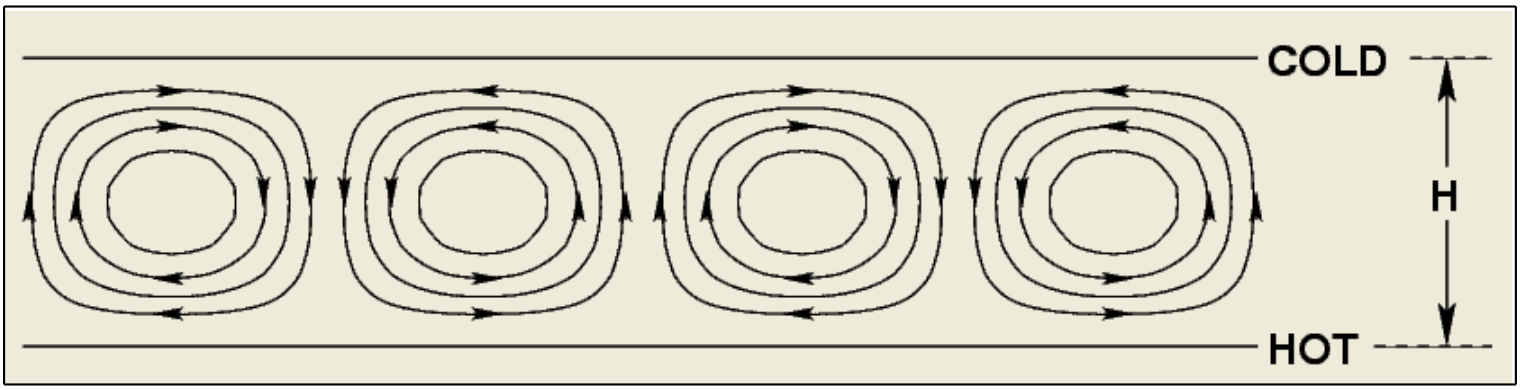
Thus began the modern field of **Chaos Theory**.

*"**Chaos** is aperiodic long-term behaviour in a deterministic system that exhibits dependence on initial conditions."*[3]

Since Lorenz first wrote down the three equations whose system now bears his name, their structure has arisen in models of lasers, dynamos an many other systems[3]. In this report we consider one such model that is an exact mechanical analogue to the Lorenz system - a **Malkus - Howard - Lorenz chaotic water wheel**[3][4][5].

2 Background Research

2.1 Lorenz System



The Lorenz equations in their original form are a model for convection rolls in the atmosphere.[2]

$$\dot{x} = \sigma(y - x)$$

$$\dot{y} = \rho x - y - xz$$

$$\dot{z} = xy - bz$$

where

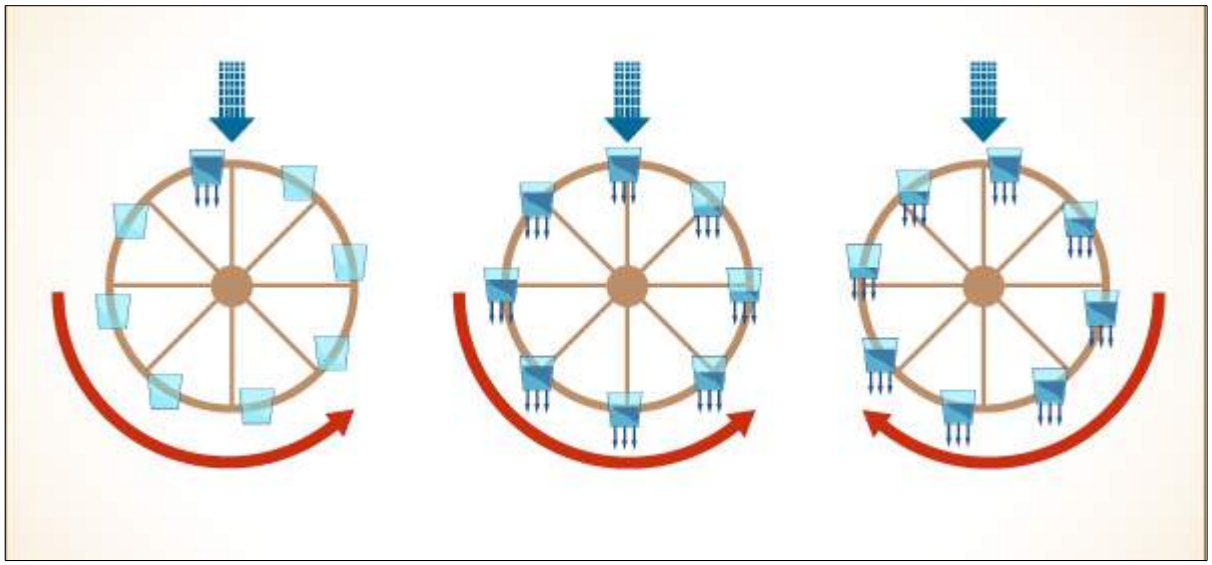
x :	Fluid Motion
y :	Horizontal Temperature
z :	Vertical Temperature
σ :	Prandtl Number
ρ :	Rayleigh Number
b :	Aspect ratio of the rolls

and ρ, σ, b > 0.

Lorenz saw that when a liquid or gas is heated from below, the fluid tends to circulate in a steady pattern of rolls, as shown in the figure above. Hot fluid rises on one side, loses heat, and descends on the other side. This is the process of convection. When the difference in temperature is slightly larger, an instability sets in, and the rolls develop a wobble that moves back and forth along the length of the cylinders. This is the region of deterministic chaos. At even higher temperatures, the flow becomes wild and turbulent and is very difficult to describe.[6]

2.2 Malkus-Howard-Lorenz Water Wheel

A Malkus-Lorenz water wheel is a type of water wheel devised in the 1970s at MIT by Willem Malkus and Lou Howard[3]. It is an exact mechanical analogue to the Lorenz system, that is to say it simulates the Lorenz equations.



In a Malkus-Lorenz water wheel water pours in from the top into a leaking cup. As the cup fills the wheel becomes top heavy and begins to rotate under the force of gravity.

As the wheel rotates, water begins to fill the other cups as they pass under the flow. The leakage in each cup is proportional to the mass of water in each cup. If the inflow is too small, the wheel won't turn. If the inflow is increased such that inflow exceeds outflow the wheel will rotate in a steady motion.

Even greater inflow destabilises the motion of the wheel and its rotations become chaotic. The wheel will rotate clockwise for a while before unpredictably reversing direction and rotating counter clockwise. The wheel can be vertical or tilted.

There are two types of Malkus water wheel - ideal and non-ideal.

Non-Ideal The most basic model pictured above consists of a small number of leaking cups arranged around a wheel. As will be shown, this wheel's motion is not described by the Lorenz system. Not all of the water that pours out of the spouts enter into one of the cups. This difference results in an arguably more complex to describe motion.

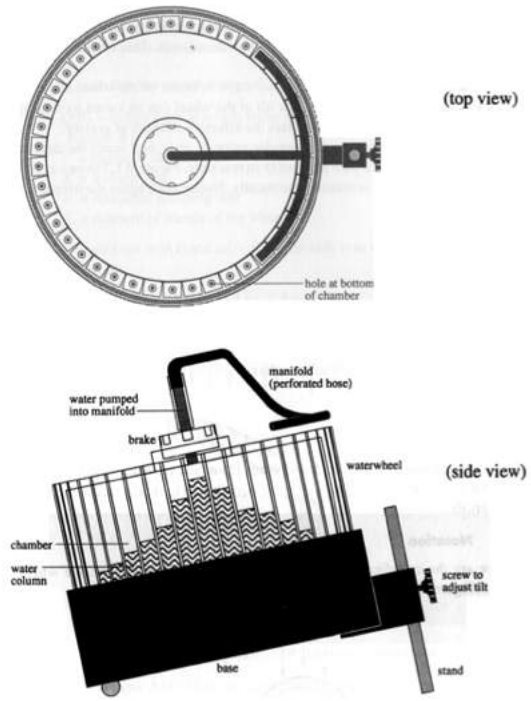
Ideal This model consists of a continuous rim of cups with no gaps between each cup. All of the inflow water must enter into one of the cups. This type of water wheel is described by the Lorenz system.

More complex models of realistic water wheels have sought to incorporate real life behaviour like cup overflow, leakage water pouring into other cups etc.[7] Such additional factors will not be considered here.

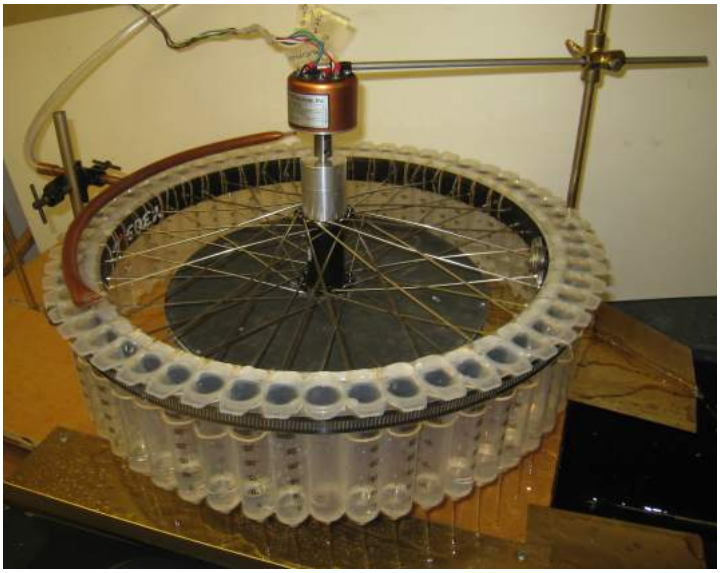
The original water wheel built by Malkus and Howard consisted of 10 leaking cups arranged around a wheel. The toy model was made famous by its inclusion in Steven Strogatz's classic *"Nonlinear Dynamics and Chaos"* where Strogatz's uses the water wheel to introduce Chaos and the Lorenz system to the reader. Strogatz's derivation of the the water wheel equations is the most well known. This report will include both Strogatz's method but also Matson's method, both of which yield the same set of equations considered from two different points of view. Its left up to the reader to decide which they consider easier to follow and understand.

2.3 Example Designs

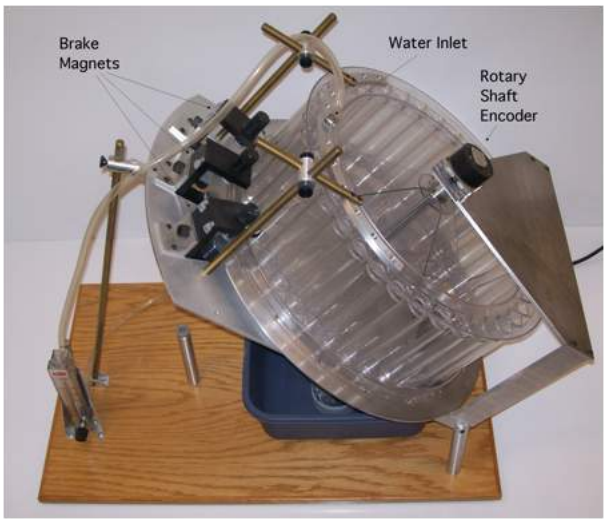
Many physics departments and hobby building enthusiasts have followed in Malkus, Howard and Lorenz's footsteps, designing and building their own chaotic waterwheels. Over the years, many of the designs have become quite sophisticated. Here is a snapshot of some of the more recent designs.



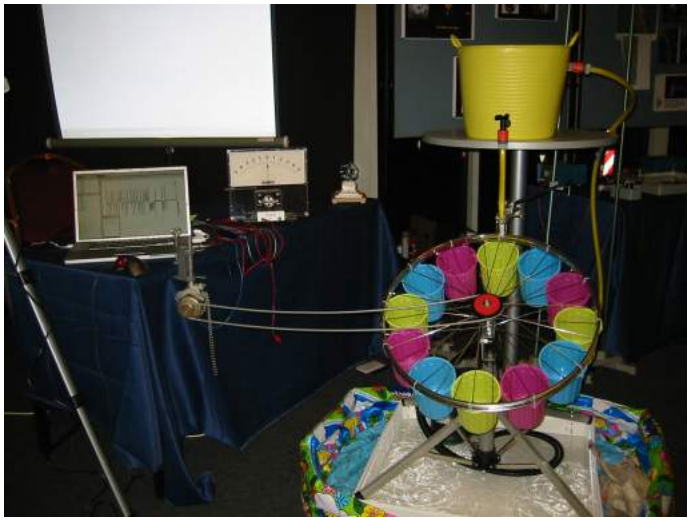
(a) Schematic of Strogatz's ideal waterwheel built at MIT



(b) Ideal waterwheel built at Reed College



(c) This ideal water wheel was built at Illinois State University



(d) This unideal water wheel built by Planeten Paultje for the Dutch Annual Physics Teacher Conference in December 2005

Figure 1: A small selection of the the different designs for both ideal and unideal Malkus water wheels

2.4 Water wheel equations of motion

Here we present the water wheel equations of motion and consider the transformation that takes us from the waterwheel equations to the Lorenz equations. We discuss terms and assumptions before finishing with a discussion on some of the simple properties of the Lorenz system. For a full derivation of the water wheel equations by the Strogatz method and the Matson method see Appendix.

The Malkus water wheel differential equations derived by the Strogatz method are

$$\begin{aligned} \frac{d\omega}{dt} &= -\frac{\nu + Q_{tot}R^2}{I_{tot}}\omega(t) + \frac{\pi R g \sin\alpha}{I_{tot}}a_1(t) \\ \frac{da_1}{dt} &= -Ka_1(t) + \omega(t)b_1(t) \\ \frac{db_1}{dt} &= q_1 - Kb_1 - \omega(t)a_1(t) \end{aligned} \tag{1}$$

To map these equations unto the Lorenz system, introduce dimensionless parameters

$$\sigma = \frac{1}{K}\frac{\nu + Q_{tot}R^2}{I_{tot}}, \quad \rho = \frac{q_1}{K^2}\frac{\pi R g \sin\alpha}{\nu + Q_{tot}R^2} \tag{2}$$

and dimensionless time

$$s = kt \tag{3}$$

With this we introduce the co-ordinates

$$x = \frac{\omega}{K}, \quad y = \frac{\rho K}{q_1}a_1, \quad z = \rho - \frac{\rho K}{q_1}b_1 \tag{4}$$

In the new co-ordinates, Eq.() becomes

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= xy - z \end{aligned} \tag{5}$$

If we compares these to the Lorenz equations

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= xy - bz \end{aligned} \tag{6}$$

We see that the Malkus water wheel equations are equal to the Lorenz equations when $b = 1$. Hence the Malkus water wheel system is a subset of the Lorenz system.

If one has followed the Matson derivation then the Malkus water wheel equations look like the following

$$\frac{d\omega}{dt} = ay - f\omega \tag{7a}$$
$$\frac{dy}{dt} = \omega z - \lambda y \tag{7b}$$
$$\frac{dz}{dt} = -\omega y + \lambda(R - z) \tag{7c}$$

where

$$a = \frac{Mg\sin\phi}{I}, \quad f = \frac{\alpha}{I} + \lambda\frac{I_w}{I} \tag{8}$$

Matson rewrites (7) and (8) in dimensionless form by defining an appropriate distance and time scale, D and T such that two of the water wheel parameters are 1.

Matson chooses to set a* and R* in the new parameterless form equal to zero by letting

$$D = R \quad T = 1/\sqrt{Ra}$$

In doing so, the four parameters become two

$$f^* = \frac{f}{\sqrt{Ra}} \quad \lambda^* = \frac{\lambda}{\sqrt{Ra}}$$

and the water wheel equations (7) become

$$\frac{d\omega}{dt} = y - f^*\omega \tag{9a}$$
$$\frac{dy}{dt} = \omega z - \lambda^* y \tag{9b}$$
$$\frac{dz}{dt} = -\omega y + \lambda^*(1 - z) \tag{9c}$$

It can be shown that the transformation relation between (9) and Lorenz system is

$$\sigma = \frac{f^*}{\lambda^*} \tag{10a}$$
$$\rho = \frac{1}{f^*\lambda^*} \tag{10b}$$
$$b = 1 \tag{10c}$$

2.5 Notation

The Malkus water wheel possesses many parameters and variables. The notation used in Strogatz's derivation differs from that in Matson's. The following are two tables containing all parameters, variables and notation used by both author. On the left is Strogatz's notation. On the right is Matson's notation

Symbol	Meaning
ω	Angular velocity of the wheel
K	Leakage rate
Q_{tot}	Water inflow
R	Radius of wheel
I_{tot}	Inertia of wheel and water in cups
$g \sin \alpha$	Effective gravity
a_1	Fourier coefficient - 1st mode
b_1	Fourier coefficient - 1st mode
ν	Viscous damping coefficient
q_1	Inflow Fourier coefficient - 1st mode

Symbol	Meaning
ω	Angular velocity of the wheel
λ	Leakage rate
Q	Water inflow
$M = Q/\lambda$	Mass of water (after transients)
R	Radius of wheel
I	Inertia of wheel and water in cups
$g \sin \phi$	Effective gravity
I_w	Inertial of water
α	Viscous damping coefficient
y	Position co-ordinate
z	Position co-ordinate

2.6 Key Assumptions Made

The derivation of the Malkus water wheel equations involve important assumptions that must be satisfied by the experimental apparatus if the results collected are to be considered valid

- **All** water from the nozzles must enter into the system (i.e into one of the cups)
- $Q(\theta)$ must be *symmetric*. That is to say how you choose to arrange your water distribution from the nozzles is irrelevant so long as the water flow distribution is symmetric with respect to the top point of the wheel (0,R).
- The cup leakage parameter (K or λ) (whatever its exact structure may be) is proportional to the mass m. By choosing cylindrical syringes or cups with cylindrical outflow tubes, we can invoke Poiseuille's Law for the leakage flow (which is is a equation proportional in m)
- As explicitly outlined in Matson's derivation and implicitly in Strogatz's derivation, we assume that, with a time constant of $1/\lambda$ or 1/K, the total mass in the system M will reach a constant. We therefore discard the first few minutes of motion as transients.

2.7 Properties of the Lorenz System

In this section, we examine some simple properties[3] on the Lorenz equations. We recall the Lorenz equations

$$\dot{x} = \sigma(y - x)$$

$$\dot{y} = \rho x - y - xz$$

$$\dot{z} = xy - bz$$

where σ, ρ and b > 0 are parameters.

Non-linearity: Two quadratic terms, xz and xy.

Symmetry: If $(x, y) \to (-x, -y)$ then the equations are the same.

Volume Contraction: The Divergence of the Lorenz system is negative. Hence volumes in phase shrink over time and the system is dissipative. As a result only stable, periodic and chaotic not quasi periodic solutions can exist.

2.8 Lyapunov Exponents

The Malkus water wheel subspace is vast and complex and the Lorenz system even more so. A question arises as to how to properly explore this space.

By changing ρ, σ and the initial position (x_0, y_0, z_0) and plotting the resultant motion in a y-z phase space, we can obtain stable motion, periodic motion, chaotic motion and everything in-between.

A question arises

"Is there a way to tell if a particular set of chosen parameter values and initial conditions will generate chaotic motion and if so can we quantify the degree of chaos that system is exhibiting?"

The answer (perhaps surprisingly) is **yes**.

Consider *fig 2.* Two points in the phase space are separated by initial distance $||\delta(0)||$. We evolve their trajectories in the phase space and consider again their separation vector after a time t. Within a linearised approximation, the separation vector of the trajectories grows as

$$||\delta(t)|| \approx ||\delta(0)||e^{\lambda t} \tag{11}$$

where λ is called the **Maximal Lyapunov Exponent**.[3][8]

The word maximal is due to the fact that there is not one but a **spectrum of Lyapunov exponents**[8] equal in number to the dimension of the system. For the Malkus water wheel and the Lorenz system that number is 3.[3]

However the maximal exponent dominates the behaviour of the system.

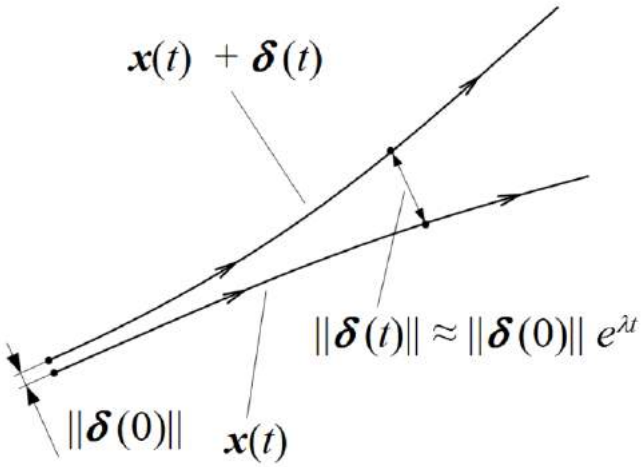


Figure 2: Evolution of two nearby trajectories in a two dimensional phase space

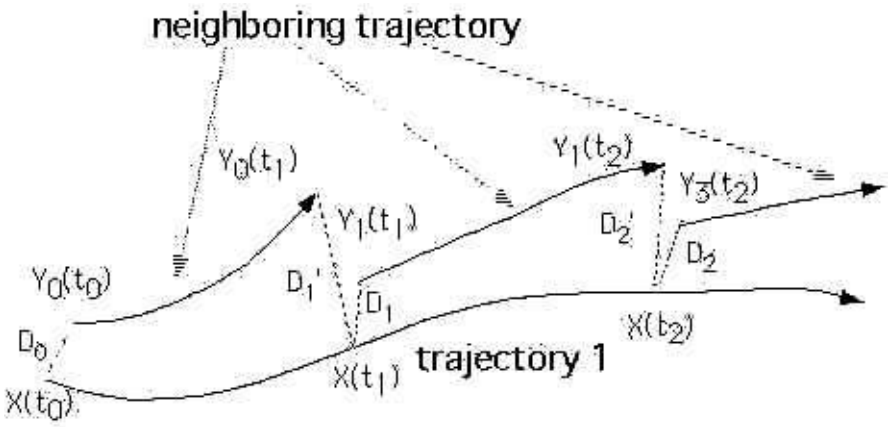
The sign of the largest Lyapunov characteristic exponent LLCE determines the motion of the system.[4] Its magnitude determines how chaotic or stable the motion is.

LLCE Sign	Motion	Effect
$\lambda < 0$	Stable	Perturbations decay exponentially
$\lambda = 0$	Periodic	Perturbations neither grow nor decay
$\lambda > 0$	Chaotic	Perturbations grow exponentially

It can be shown by ergodic theory that the set of Lyapunov exponents is the same for all initial conditions (x_0, y_0, z_0)[8]. Therefore, in our calculations, each set of exponents for a chosen ρ and σ is calculated from a random initial condition.[8]

The two main algorithms for calculating Lyapunov exponents are **orbit separation**[9] and **Gram-Schmidt orthonormalisation procedure**[4][8][10][11]. Code has been provided for both methods in this report. However it was found that the re-ortho method was faster and as result was used for the purpose of generating the parameter map.

2.8.1 Orbit Separation



The method[9] of orbit separation can be used to calculate the largest Lyapunov characteristic exponent. (**LLCE**).

1. Two orbits $\vec{x}$ and $\vec{y}$ are initialised so $x_0 = y_0$.
2. Then y_0 is perturbed by a small ϵ.
3. Both orbits are integrated for a small time step dt.
4. The magnitude of their displacement vector $d = ||\vec{x} - \vec{y}||$ is recorded.
5. The LLCE for that time step is calculated by LLCE= $\frac{1}{dt} \log\left(\frac{d}{\epsilon}\right)$.
6. Scale y orbit vector such that the magnitude of the displacement vector d is once again ϵ but the direction is preserved from the integration from (3) i.e

   $y_{new} = y + \frac{\epsilon}{d}(\vec{x} - \vec{y})$
7. To get an accurate answer for the LLCE, repeat 1-6 approximately 10000 times and discard the first 2000 values to insure the LLCE's for the orbits are on the attractor.
8. Average for final answer.

2.8.2 Gram-Schmidt Orthonormalisation Procedure

This method can be used to calculate the entire Lyapunov spectrum[8]. Although only the LLCE is needed to determine if a particular choice of ρ and σ gives rise to chaotic motion, it can be useful to know the full spectrum.

We consider in our three dimensional space, three orthonormal vectors. For simplicity we choose the three traditional unit vectors

$$\begin{bmatrix}1\\0\\0\end{bmatrix} \quad \begin{bmatrix}0\\1\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\1\end{bmatrix}$$

As stated before the Lorenz equations are

$$\dot{x} = \sigma(y - x)$$

$$\dot{y} = \rho x - y - xz$$

$$\dot{z} = xy - bz$$

We now consider the **Jacobian** of the Lorenz system (also called the flow matrix or tangent space map).

$$J(x, y, z) = \begin{bmatrix} -\sigma & \sigma & 0 \\ -z + \rho & -1 & x \\ y & x & -b \end{bmatrix}$$

whose elements are the partial derivatives of the Lorenz system. This matrix completely describes how nearby trajectories to a point change in space. i.e how infinitely small displacement vectors to a point in space evolve in space.

The variation equation

$$\frac{d\vec{P}}{dt} = J\vec{P}(t)$$

describes in turn how the Jacobian evolves in time i.e how infinity close trajectories evolve both in space and time. Therefore the variation equations contains all the information we need to obtain the Lyapunov spectrum.

For a three dimensional system, one needs to take into account that the rate of separation of two trajectories in phase system can be different for different orientations of the initial separation vector. Our three dimensional system has three directions in which to either stretch or shrink and hence three Lyapunov exponents for a three dimensional phase space.

We would like to use each of of our three orthonormal vectors to calculate the LCE associated with that dimension. To that end, it would seem the simplest approach is to integrate the variational equation and the Lorenz equations as a coupled system with I (equivalent to our unit vectors) as the initial condition. We could then take the log of the magnitudes of the column vectors and divide by the time elapsed to obtain our spectrum. Unfortunately this runs into several problems.

- We know from the theory of Lyapunov exponents that these separation vector's magnitudes will grow exponentially, quickly exceeding computer representation.
- All of these vectors will quickly align with the direction of maximal growth with only the LLCE possible to calculate (this is why orbit separation works). So if we can't integrate very far in time we can't obtain an accurate answer for the spectrum...

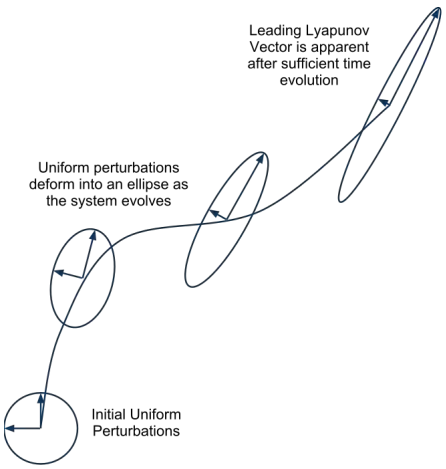


Figure 3: A perturbation asymptotically aligns with the Lyapunov vector associated with the LLCE

However in the 70s and 80s, it was shown that this numerical instability could be remedied with repeated application of the **Gram-Schmidt Orthonormalisation Procedure**.[8]

Recall the 1D linearised approximation of a small separation or perturbation ϵ to x_0. After dt, the perturbation u_t will become

$$u_t \approx D_{x_0} f(x_0) \cdot u_0$$

The LCE is therefore

$$\lambda = \lim_{t\to\infty} \frac{1}{t} \ln||D_{x_0} f(x_0) \cdot u_0||$$

Generalising this to n-dimensions it can be shown[8] that the sum of n Lyapunov exponents describes the mean rate of growth of an n-dimensional parallelepiped

$$\sum_{i=1}^{k} \lambda_k = \lim_{t\to\infty} \frac{1}{t} \ln[V_k] \quad (1 \leq k \leq n)$$

The Gram-Schmidt procedure takes a set $(u_1, ..., u_n)$ of n linearly independent vectors and returns an orthonormal set $(v_1, ..., v_n)$ which spans the same subspace as $(u_1, ..., u_n)$.

The vectors are constructed as follows

$$w_1 = u_1, \quad v_1 = \frac{w_1}{||w_1||}$$

$$w_2 = u_2 - \langle u_2, v_1\rangle v_1, \quad v_2 = \frac{w_2}{||w_2||} \quad ...$$

Generalising

$$w_n = u_n - \sum_{i=1}^{n-1} \langle u_n, v_i\rangle v_i, \quad v_n = v_n = \frac{w_n}{||w_n||}$$

Geometrically the procedure can be thought as the following

- Define one vector as first axis
- Normalise to unit length
- Take second vector. Remove all components of first vector from second vector. Now they are both orthogonal. Normalise second vector to unit length.
- Repeat for n vectors by removing all components of the previous $n-1$ vectors from n^{th} and normalise n^{th} vector to unit length. In doing so we are "straightening" the parallelepiped back into a cube.

The key to the success of the procedure is that the volume of the new orthogonal set $(w_1, ..., w_n)$ is the same as the old set $(u_1, ..., u_n)$. The volume which was once a parallelepiped is now a cube again given by

$$\text{Vol } \{u_1, ..., u_n\} = ||w_1|| \cdots ||w_n||$$

To calculate the LCE spectrum, we preform the following algorithm

- Integrate the Lorenz system and the variation equation as a coupled system for a few time steps, $\approx 10 \cdot dt$.
- Evolve the set of orthonormal vectors $(u_1, ..., u_n)$ (corresponding to our traditional unit vectors) according to the solution of the variation equation for the time $t = 10 \cdot dt$ and final position (x, y, z) i.e

$$(u_1, ..., u_n) = P(x, y, z, t) \cdot (u_1, ..., u_n)$$

- We now preform the Gram-Schmidt Orthnormalisation procedure on the set $(u_1, ..., u_n)$. The set $(w_1, ..., w_n)$ we collect for the calculation of the LCE spectrum. We collect the orthonormal set $(v_1, ..., v_n)$ to use for our next iteration.
- In this time step, the volume of $(u_1, ..., u_n)$ has increased by a factor $||w_1|| \cdots ||w_n||$
- Oseledec in 1968 showed that one can find n linearly independent vectors $u_1, ..., u_n$ such that for any initial point x_0 and initial set of orthonormal vectors U_0

  $$\lambda^n(x_0, U_0) = \lambda_1 + ... + \lambda_n$$

  However

  $$\lambda^n(x_0, U_0) = \lim_{t\to\infty} \frac{1}{t} \ln(||w_1|| \cdots ||w_n||)$$

  Implying

  $$\sum_{i=1}^{n} \lambda_n = \lim_{t\to\infty} \frac{1}{t} \ln(||w_1|| \cdots ||w_n||)$$

  This is restating above replacing V_k by the set $(w_1, ..., w_n)$.

  By the rules of logs this implies for our $n = 3$ system

  $$\lambda_1 \approx \lim_{t\to\infty} \frac{1}{t} \ln(||w_1||) \quad \lambda_2 \approx \lim_{t\to\infty} \frac{1}{t} \ln(||w_2||) \quad \lambda_3 \approx \lim_{t\to\infty} \frac{1}{t} \ln(||w_3||)$$

- Using the orthonormal set $(v_1, ..., v_n)$ as our new starting point we repeat the algorithm for either a large t or until one or more convergence tests (see Properties of the Lyapunov Spectrum) are satisfied.
- For an $n = 3$ system, the dynamics of the attractor is either a Torus T^2 corresponding to a LCE spectrum $(\lambda_1, \lambda_2, \lambda 3) = (0, 0, -)$ or Chaos C^1 corresponding to a LCE spectrum $(\lambda_1, \lambda_2, \lambda_3) = (+, 0, -)$

In terms of geometrically understanding the process we must keep the vectors orthogonal so that they measure stretching in one direction and shrinking in the complementary direction. If we didn't subtract out the part of the second vector that was being stretched, then it's change in size would be dominated by stretching. What we do is subtract out the component of the first vector, which is measuring the stretching, from the second vector. The result is that the latter only points in the direction of shrinking and so its change in size is used to estimate the second LCE. We in turn extend this to the third possible direction of stretching or shrinking for our third vector. [11]

2.8.3 Properties of the Lyapunov Spectrum

Only the LLCE is needed for predicting if the chosen values of ρ and σ will result in stable, periodic or chaotic motion. However the spectrum possess properties that are useful for convergence tests. [4]

1. If the system is conservative then $\sum_{i=1}^{3} \lambda_i = 0$. If the system is dissipative then $\sum_{i=1}^{3} \lambda_i < 0$.
2. More specifically $\sum_{i=1}^{3} \lambda_i = \text{tr}[\mathbf{J}] = -\sigma - 2$
3. There is always a zero Lyapunov exponent if the system is a flow (that does not converge to a fixed point) corresponding to the vector pointing in the direction of the flow.

2.8.4 Assessment of Methods

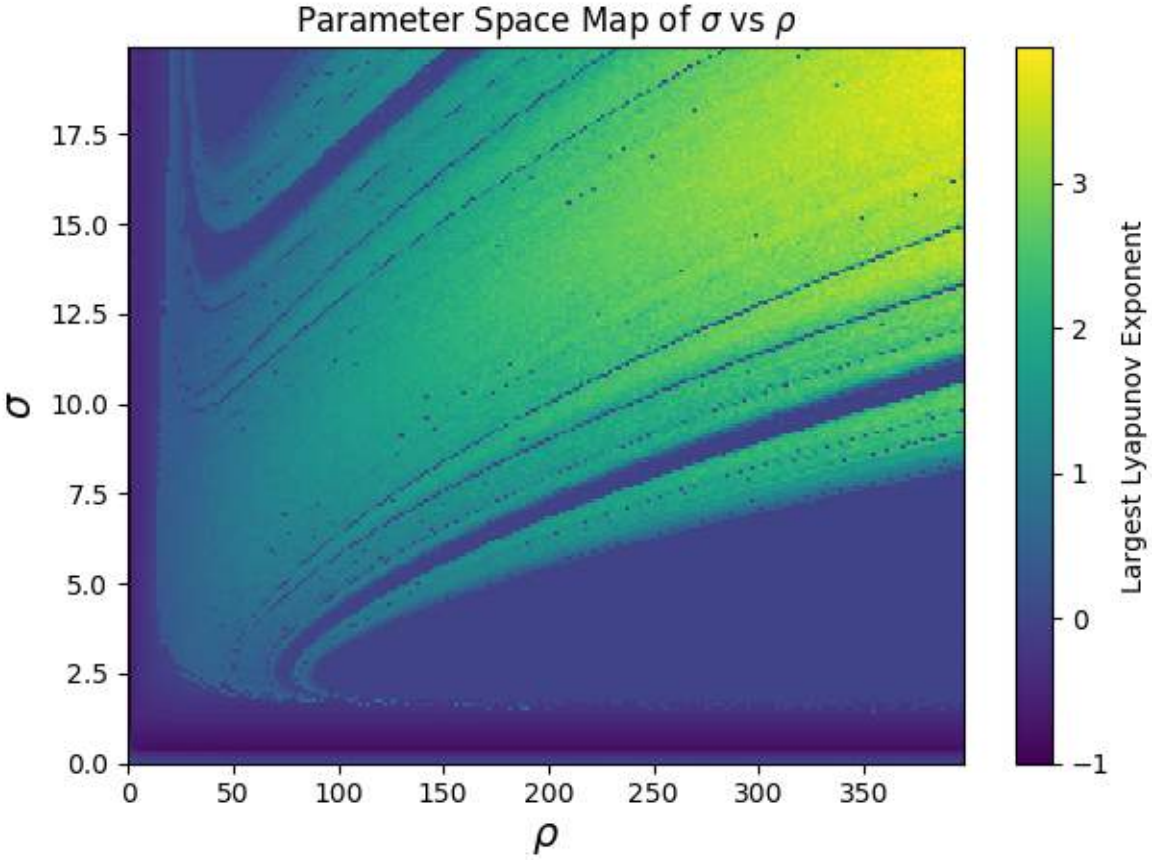
Both methods were tested and the Gram-Schmidt method was deemed to be faster. Even though by selecting this method we enabled ourselves access to convergence tests they were not used as convergence to $\approx 1 \times 10^{-3}$ not only proved too slow but wasn't necessary. By integrating in steps of 10 for 100 iterations, the final value for the LLCE was deemed accurate enough for this project. This was based on a test with "classic" parameter values $\rho = 28, \sigma = 10, b = 8/3$, which gave a λ_1 corresponding to the LLCE of ≈ 0.9 consist with attempted values[3]. Furthermore $\lambda_2 \approx 0$ for on average 3-4 significant figures, again consistent as $\lambda_2 = 0$ for the Lorenz system which is a flow.

2.9 Parameter Space Map

By using the Gram-Schmidt method, the LLCE was calculated for a 200×200 grid of values for ρ and σ (b=1) and displayed as a colour map. Each box corresponds to $\Delta\rho = 2$, $\Delta\sigma = 0.1$. The program used was LLCE Parameter Map Generator. The run-time was ≈ 10 hours.

A larger full page image can be seen in the Appendix.

As it can be seen, the parameter space map is complex, structured and rather beautiful.



;

Examining the plot we see that, for small ρ and σ, the motion is only stable or periodic as shown by the purple and dark blue regions.

Similarly if one parameter is kept small and other large, the type of motion is again stable or periodic.

For large ρ and σ the motion is dominated by chaos as indicated by the yellow and green regions.

However within the regions dominated by chaos, there are small enclaves of stable and periodic motion as indicated by the blue windows dotting the center to upper right corner of the image.

As mentioned by Illing et al (who produced a similar plot)

"Interestingly, the numerical calculations reveal there are periodic windows embedded inside the chaotic region and that these windows seem to form a fractal set, with an increasing number of ever smaller periodic windows, each organised along a line in parameter space"

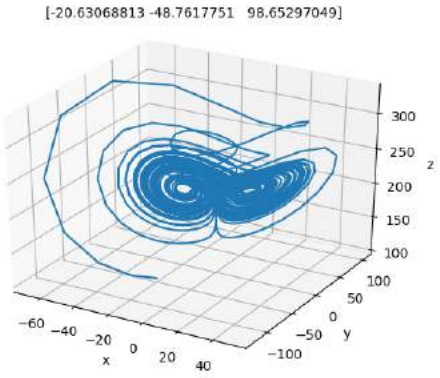
As described we see blue curves of order streaking through regions of pure chaos. These curves seem to thin as one approaches the center of the image.

2.9.1 Hidden Oscillations

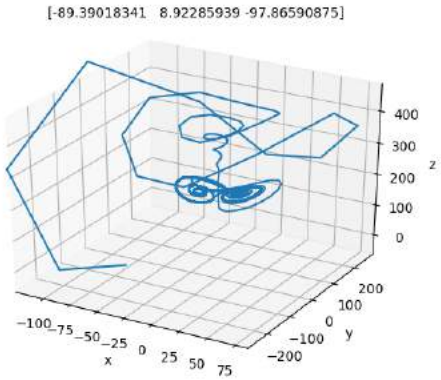
At the beginning of this section it was stated that, regardless of the initial conditions, for a chosen ρ and σ, the spectrum of Lyapunov exponents will be the same. This is true for nearly every value of ρ and σ. For these values there exists a single, global attractor whose basin of attraction is the set of all initial conditions. This attractor is either a fixed point (stable), limit cycle (periodic) or strange attractor (chaotic)

However for an unpredictable number of ρ, σ there exists more then one attractor as a solution! These hidden attractors are called **hidden oscillations** and always correspond to periodic motion. For these parameter values, the initial conditions do matter since (x_1, y_1, z_1) could be on the basin of the strange attractor giving a resultant $\lambda_1 \approx 0.9$, while (x_2, y_2, z_2) could be on basin of the hidden oscillation giving a resultant $\lambda_1 \approx 0$.

The following plots were generated for $\rho = 216$, $\sigma = 17.642$, $b = 1$ and two random initial positions. This choice of ρ and σ corresponds to one of the periodic windows on the parameter space map.



(a) Strange Attractor

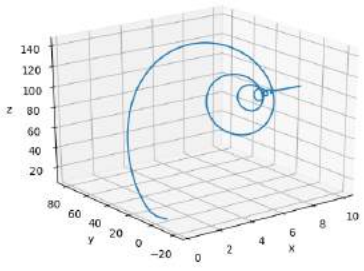


(b) Hidden Oscillation

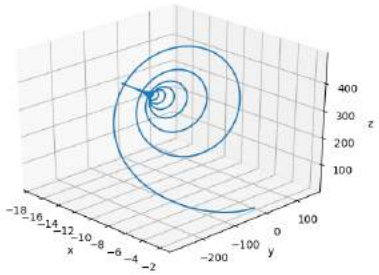
As one can see,the initial condition for (a) is in the basin of a strange attractor while the initial condition for (b) is the basin of a limit cycle. This corresponds to a hidden oscillation.

2.10 Attractors of the Lorenz System

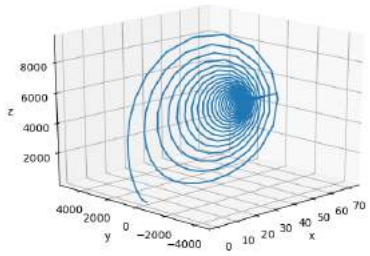
The solutions to the Lorenz system $b = 1$ corresponding to the Malkus water wheel result in varying types of attractors ranging from fixed points, limit cycles and strange attractors. Near the boundaries of two different regions of motion in the parameter map, the attractor plot often has mixtures of both, e.g transient chaos giving way eventually to periodic motion on a limit cycle. The following pages contain plots in the phase space of a variety of different attractors that can appear by choosing ρ and σ from different areas of the parameter map.



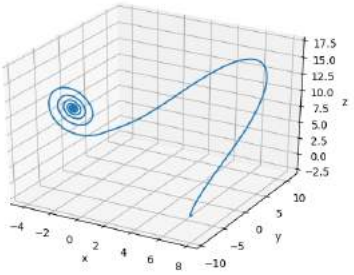
(a) $\rho = 100, \sigma = 0.1$, Fixed Point



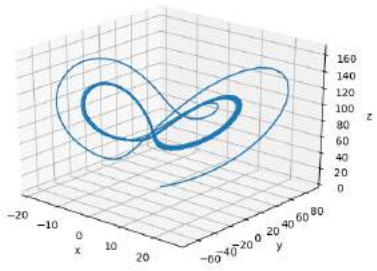
(b) $\rho = 300, \sigma = 0.1$, Fixed Point



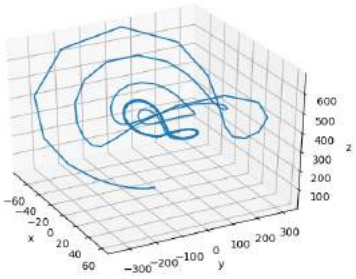
(c) $\rho = 5000, \sigma = 0.1$, Fixed Point



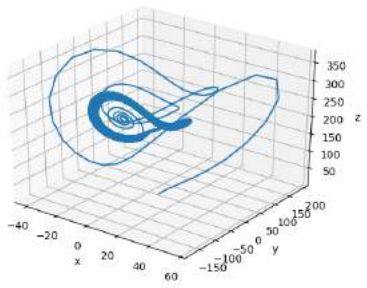
(d) $\rho = 10, \sigma = 2$, Limit Cycle



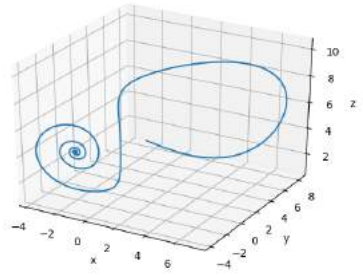
(e) $\rho = 90, \sigma = 2.5$, Limit Cycle



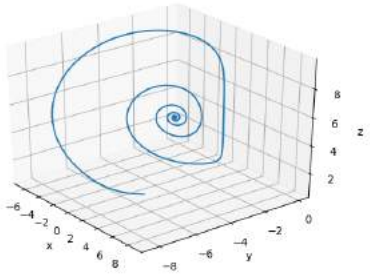
(f) $\rho = 360, \sigma = 5$, Limit Cycle



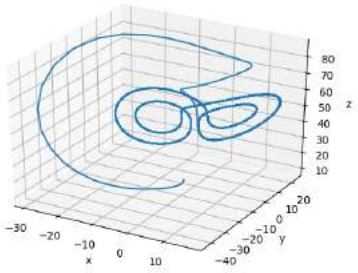
(a) $\rho = 200, \sigma = 5.5$, Unusual Limit Cycle



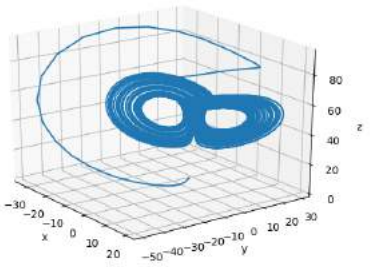
(b) $\rho = 5.0, \sigma = 13$, Limit Cycle



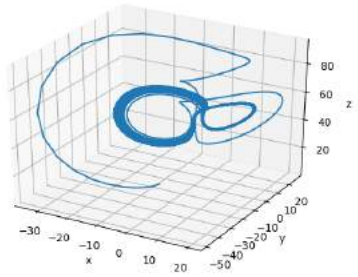
(c) $\rho = 5, \sigma = 14$, Limit Cycle



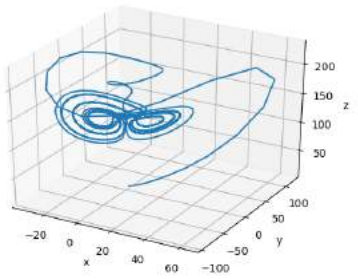
(d) $\rho = 50, \sigma = 14$, Limit Cycle



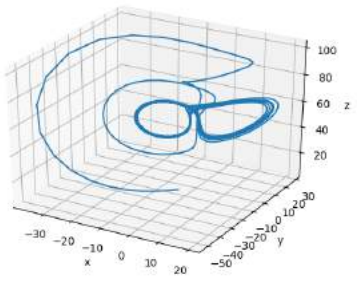
(e) $\rho = 50, \sigma = 15$, Strange Attractor



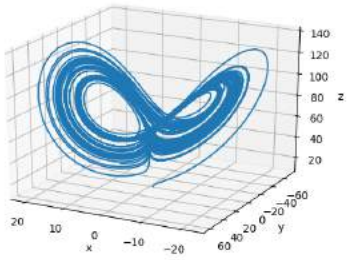
(f) $\rho = 50, \sigma = 17.5$, Unusual Limit Cycle



(a) $\rho = 125, \sigma = 18$, Unusual Limit Cycle



(b) $\rho = 55, \sigma = 19$, Unusual Limit Cycle



(c) $\rho = 80, \sigma = 2.5$, Strange Attractor

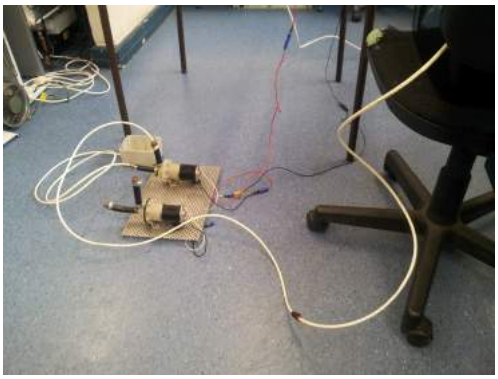
The main purpose of this map is to allow the user to know which values of ρ and σ to choose so as to obtain stable, periodic, or chaotic motion and to what degree.

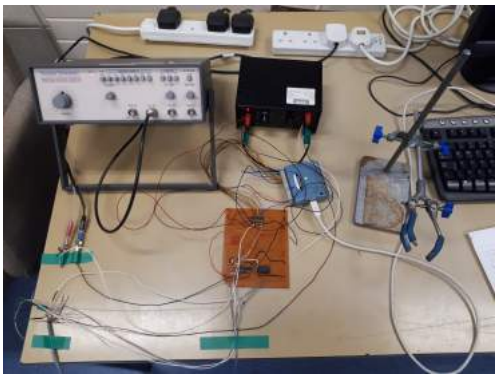
3 Construction of the Chaotic Water Wheel

3.1 Frame

3.1.1 4 Bucket Water Wheel

Figure 8: 4 bucket water wheel





3.1.2 45 Bucket Water Wheel

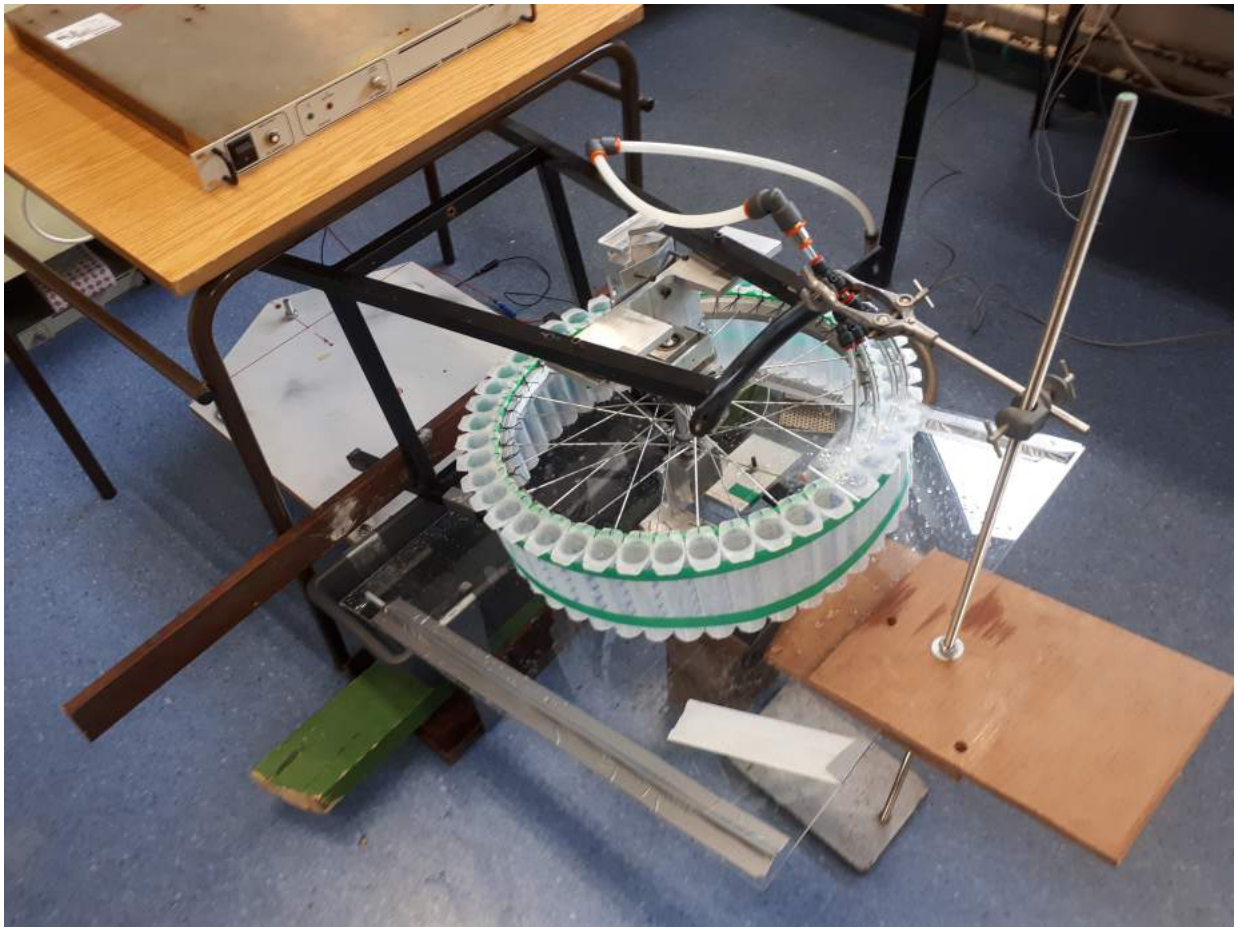
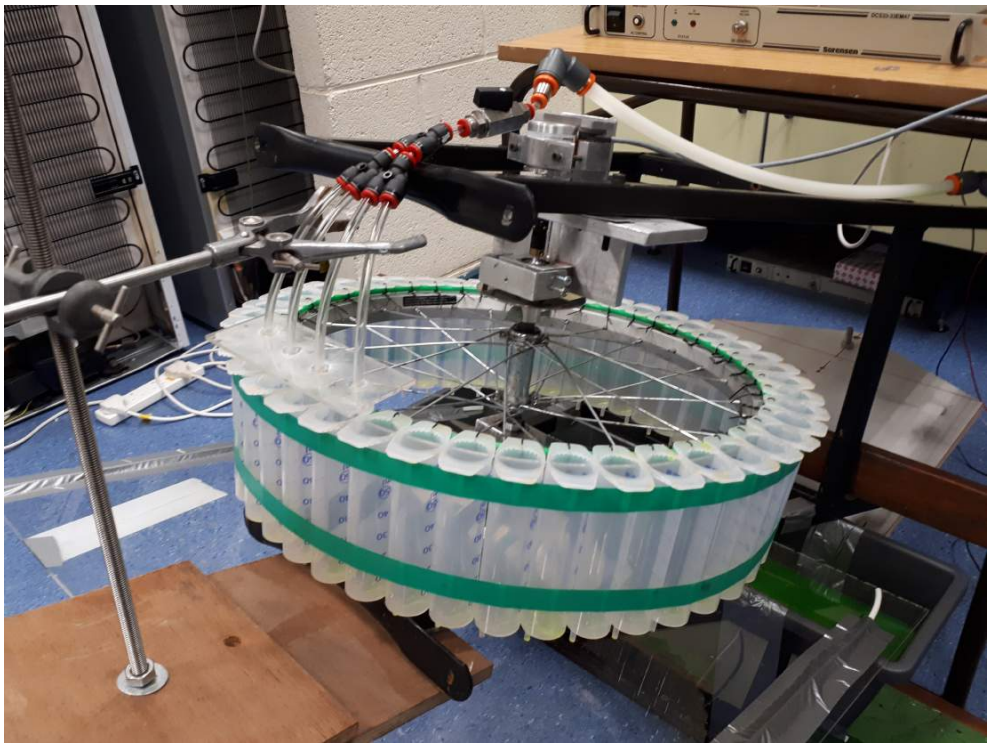
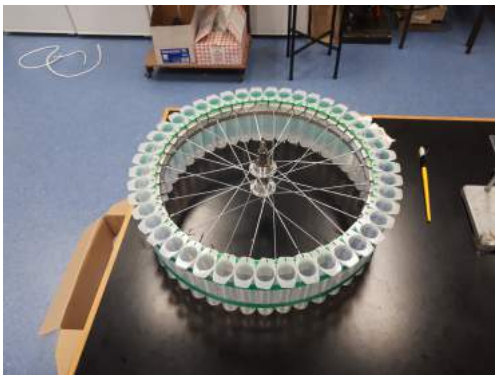
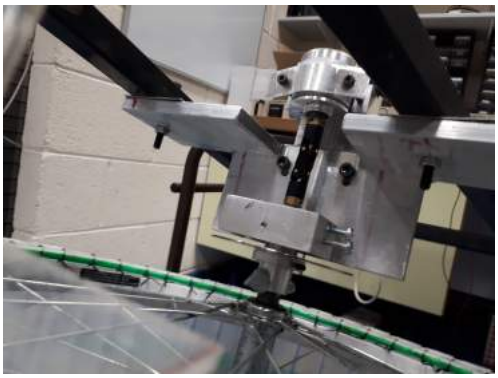


Figure 9: 45 bucket water wheel frame







3.2 Buckets/Syringes

There was 4 types of buckets used for the 4 bucket water wheel. Each type was found to have its advantages and disadvantages, which are listed below.

1. These large, long cylindrical buckets were never used as they proved to be too large to fit and freely allow the wheel to rotate.

2. These smaller cyclindrical buckets were made out of vitamin containers and orginally intended for a smaller toy model water wheel. However they proved to be suitable for the water wheels used in this report. Holes were drilled in the bottom and tubing attached to insure the leakage was a laminar flow. However the leakage was too great and (as seen from fig 10) wire was placed in the tubing to reduce flow. The advantage was air bubbles did not form to prevent leakage but a disadvantage was a flat and not slanted bottom prevented all of the water leaking out.

3. These buckets are made from the same type of syringe as the 45 bucket water wheel. Improvements on 2. included transparent plastic which enabled visual measurement of the water content in each bucket via video recording. Furthermore the outflow tube was small enough that leakage did not exceed inflow and a slight slant in the bottom insured that all water could leak out. However this type was far more prone to air bubbles forming in the leakage tube causing up to 20ml of water collecting in the syringe before leakage again.

4. These buckets bare similar resemblance to 2. However the leakage tubes have a smaller radius such that the resultant leakage does not exceed inflow. However they still possess all other disadvantages of 2.

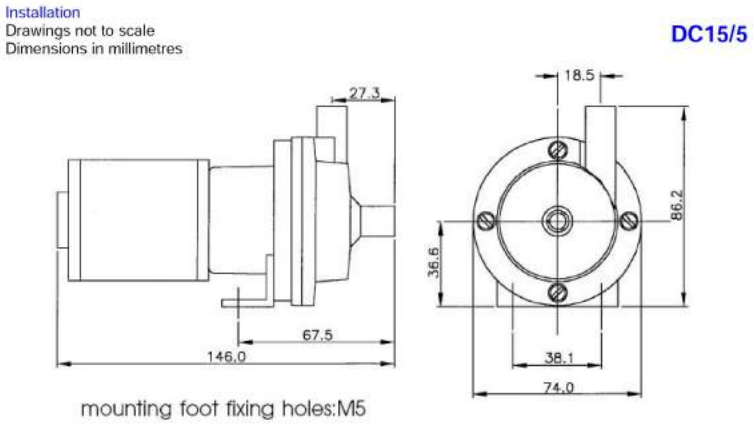
Figure 10: The 4 types of bucket tried for the 4 bucket water wheel

All rods that connect the buckets to the wheel were sprayed with a thin coating of lacquer to combat oxidation and rust.

3.3 Pump system / Float Relay system

3.3.1 Pump system

The pump used in this project was a **Totton Pumps (now Xylem Flojet) Magnetic Coupling Centrifugal Pump, 23L/min, 12 V dc, Model Number: 073983**



(a) Pump used in project placed against a cooling fan

(b) Schematic diagram of pump used

Key Features

The following is a brief summary of the key features of the pump used in this project:

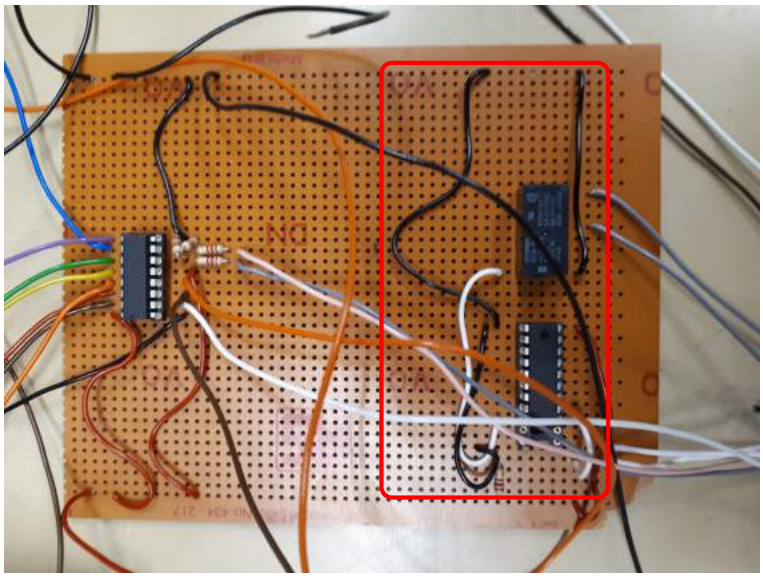
- Pump is not self-priming meaning water must be pushed through initially by gravity before pumping action can begin.
- **Not designed to run dry**
- Cooling fan required as the pump quickly overheats with prolonged use
- Sorensen 20V power supply used

3.3.2 Float - Relay System

The float - relay system was designed for the 4 bucket system. It consists of a **float switch** and a **1114P relay** controlled by a **2981A driver**.

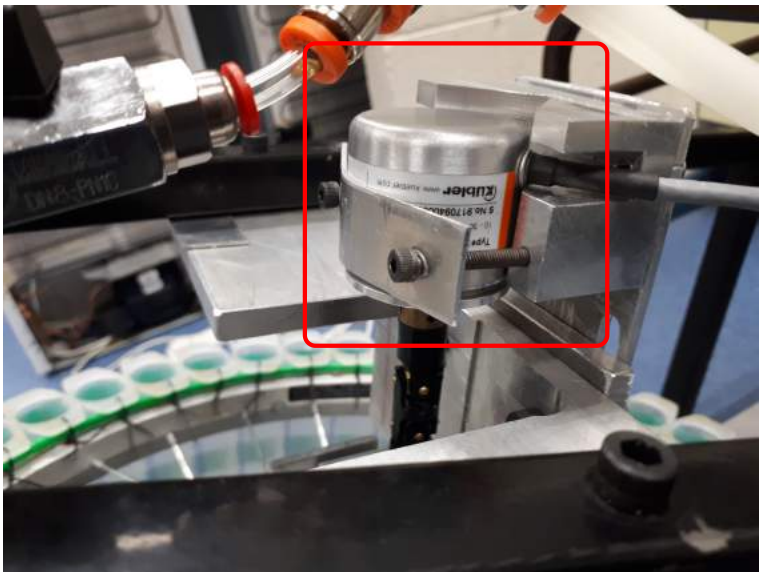
In the 4 bucket water wheel, water is pumped from the bottom to the top tank. However since water is pumped up to the top tank faster then it drains into the bottom tank, the bottom tank will eventually empty. A circuit is needed to control power to the pumps since they cannot be run dry.

(a) Float switch

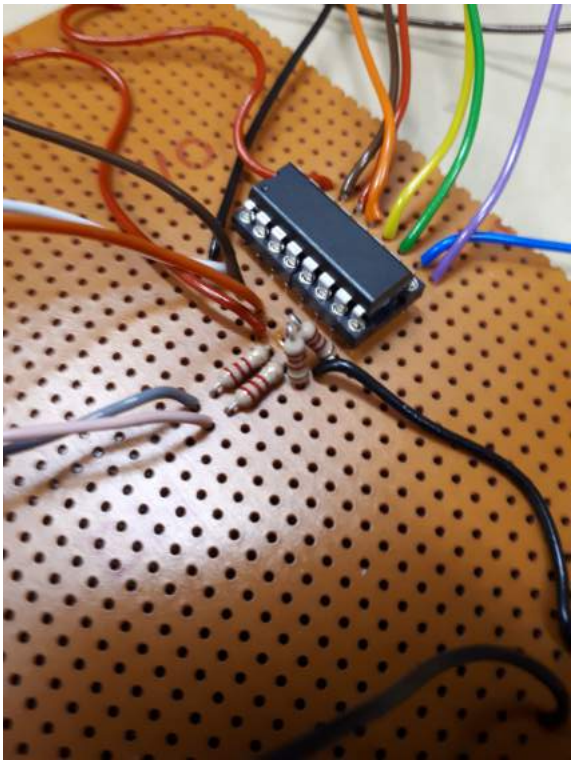


(b) Relay circuit

3.4 Rotary Encoder/Decoder system



(a) Rotary Encoder



(b) Quadrature Decoder

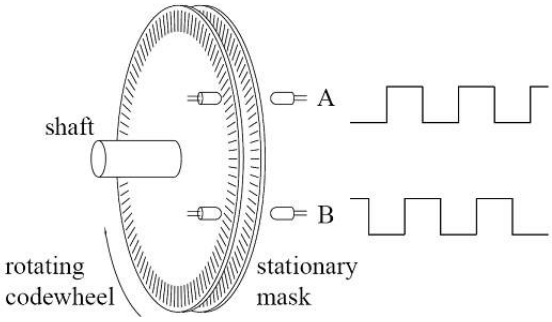
3.4.1 Rotary Encoder

Rotary Encoder Theory: A rotary encoder is a sensor that converts the angular position of a axle into an electrical signal - either analog or digital[12]. There are two types - **incremental** and **absolute**.

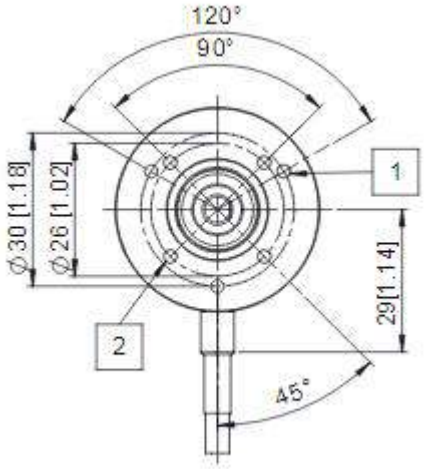
An absolute encoder has an unique code pattern for each position so it remembers its position even if power is removed, whereas an incremental encoder determines position by generating pulses[14]. There are many different ways by which the angular position can be converted into a signal - mechanical, optical or magnetic. We will discuss the the type used in this project - an **incremental**, **optical** encoder.

Incremental rotary encoders can only measure changes in angular position. For an optical encoder, a disc with slits and bars is placed between an LED and a receiver. The light emitted by the LED is modulated by the disc producing light pulses. The number of pulses generated is proportional to the angle rotated by the axle. An incremental encoder employs two outputs called A and B. Two channels are used where the direction of rotation is needed. This requirement will be further explained in the quadrature decoder section. In addition, a third channel called Z is sometimes included which outputs an index pulse once every turn. This provides an absolute reference if required.[13]

Figure 14: The principal of operation of an incremental rotary encoder.

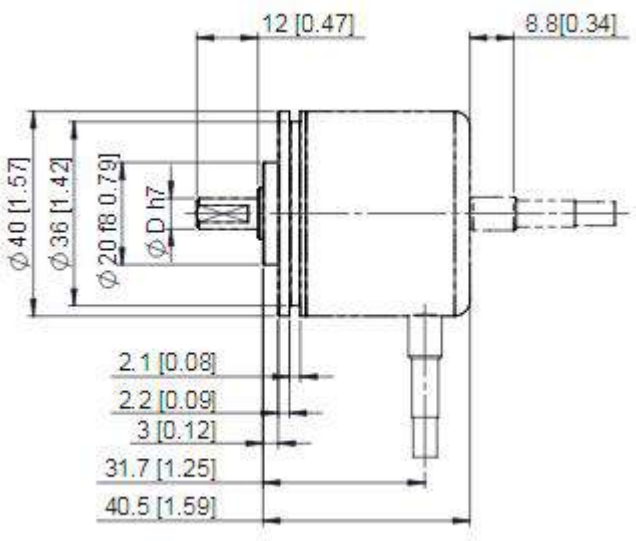


The **resolution** of an encoder is the smallest change in angle required to output a pulse. Encoder resolutions are described in ppr (pulses per revolution). Therefore, if a encoder has a resolution of 2000 ppr, then the smallest change in angle detectable by the encoder is $\frac{360^\circ}{2000} = 0.18^\circ$. These light pulses form the input for the quadrature decoder which outputs a 12 bit value for the USB to receive. This project used an optical incremental encoder. The optical type enables higher speeds to be measured accurately.[12]



;

(a) 8.KIS40.1342.2500 (front)



(b) 8.KIS40.1342.2500 (side)

Kübler 8.KIS40.1342.2500 Incremental Encoder: The rotary encoder used for this project was a Kübler 8.KIS40.1342.2500 Incremental Encoder. This particular encoder is part of Kübler's KIS40 range.

The data sheet for this range of encoders is included in the next two pages with information about the encoder used for this project as well as important general information common to all encoders of this range highlighted and commented.

The following is a brief summary of the key details of the encoder used in this project:

Figure 16: 8.KIS40.1342.2500

- Optical
- Shaft version
- Resolution of 2500 pulses per revolution
- Push-Pull output circuit
- Inverter signal (not used)
- 10...30 V DC power supply
- 2m radial connection cable
- Max pulse frequency of 250kHz

Signal	0V	+V	A	$\bar{A}$	B	$\bar{B}$	0	$\bar{0}$
Cable Colour	WH	BN	GN	YE	GY	PK	BU	RD
Used?	✓	✓	✓	X	✓	X	✓	X

Compact optical | **Sendix Base KIS40 / KIH40 (shaft / hollow shaft)** | **Push-pull / RS422 / Open collector**

The incremental encoders type Sendix Base KIS40 / KIH40 with optical sensor technology have been designed for highest cost-effectiveness. They are available with a resolution of up to 2500 pulses per revolution.

They are particularly suitable for tight mounting spaces and small machines and appliances.

Compact and robust

- Only 40 mm outer diameter.
- Ideally suited for use where space is tight.
- Sturdy bearing construction in Safety Lock™ design.
- Safe commissioning: reverse polarity protection and short-circuit proof.

Flexible

- Maximum resolution of 2500 pulses per revolution.
- Power supply 5 V DC or 10 ... 30 V DC.
- Push-pull, RS422 or open collector
- Radial or axial cable.

1342 2500

Order code
Shaft version

8.KIS40	.	1	X	X	X	.	XXXX	.	PXX [1]
Type		a	b	c	d		e		f

a *Flange*
1 = clamping-synchro flange, ø 40 mm [1.57"]

b *Shaft (ø x L)*
3 = ø 6 x 12.5 mm [0.24 x 0.49"], with flat
5 = ø 1/4" x 12.5 mm [1/4" x 0.49"], with flat
6 = ø 8 x 12.5 mm [0.32 x 0.49"], with flat

c *Output circuit / power supply*
3 = open collector (with inverted signal) / 10 ... 30 V DC
4 = push-pull (with inverted signal) / 10 ... 30 V DC
6 = RS422 (with inverted signal) / 5 V DC
7 = open collector (without inverted signal) / 10 ... 30 V DC
8 = push-pull (without inverted signal) / 10 ... 30 V DC

d *Type of connection*
1 = axial cable, 2 m [6.56'] PVC
2 = radial cable, 2 m [6.56'] PVC

e *Pulse rate*
25, 100, 200, 360, 500, 512, 600, 1000, 1024, 2000, 2048, 2500
(e.g. 500 pulses => 0500)

f *Special signal format*
P03 = see page 58

Stock types

8.KIS40.1342.0360	8.KIS40.1362.0500
8.KIS40.1342.0500	8.KIS40.1362.1024
8.KIS40.1342.1000	8.KIS40.1362.2048
8.KIS40.1342.1024	
8.KIS40.1342.2048	
8.KIS40.1342.2500	

Optional on request
- other pulse rates

1) Is only necessary when a special output signal format is required.

Compact optical	Sendix Base KIS40 / KIH40 (shaft / hollow shaft)	Push-pull / RS422 / Open collector

Order code Hollow shaft

8.KIH40 . X X X X . XXXX . PXX [1]
Type — a b c d — e — f

a *Flange*
2 = with spring element, long
5 = with stator coupling, ø 46 mm [1.81"]

b *Blind hollow shaft (insertion depth max. 18 mm [0.71"])*
4 = ø 8 mm [0.32"]
3 = ø 1/4"

c *Output circuit / power supply*
3 = open collector (with inverted signal) / 10 ... 30 V DC
4 = push-pull (with inverted signal) / 10 ... 30 V DC
6 = RS422 (with inverted signal) / 5 V DC
7 = open collector (without inverted signal) / 10 ... 30 V DC
8 = push-pull (without inverted signal) / 10 ... 30 V DC

d *Type of connection*
1 = axial cable, 2 m [6.56'] PVC
2 = radial cable, 2 m [6.56'] PVC

e *Pulse rate*
25, 100, 200, 360, 500, 512, 600, 1000, 1024, 2000, 2048, 2500
(e.g. 500 pulses => 0500)

f *Special signal format*
P03 = see page 58

Stock types

8.KIH40.2442.1024	8.KIH40.5442.0360
8.KIH40.2462.1000	8.KIH40.5442.0500
8.KIH40.2462.1024	8.KIH40.5442.1024
	8.KIH40.5442.2048
	8.KIH40.5442.2500
	8.KIH40.5462.0500
	8.KIH40.5462.2048

Optional on request
- other pulse rates

Mounting accessory for shaft encoders		Order no.
Coupling	bellows coupling ø 15 mm [0.59"] for shaft 6 mm [0.24"]	**8.0000.1202.0606**

Connection technology		Order no.
Connector, self-assembly (straight)	M12 female connector with coupling nut, 8-pin	**05.CMBS 8181-0**

Further accessories can be found in the accessories section or in the accessories area of our website at: www.kuebler.com/accessories.
Additional connectors can be found in the connection technology section or in the connection technology area of our website at: www.kuebler.com/connection_technology.

Technical data

Mechanical characteristics

Maximum speed		4500 min^{-1}
Mass moment of inertia		approx. 0.2 x 10^{-6} kgm^2
Starting torque – at 20°C [68°F]		< 0.05 Nm
Shaft load capacity	radial	40 N
	axial	20 N
Weight		ca. 0.17 kg [6.00 oz]
Protection acc. to EN 60529		IP64

Working temperature range		-20°C ... +70° [-4°F ... +158°F]
Materials	shaft	stainless steel
	flange	aluminum
	housing	aluminum
	cable	PVC
Shock resistance acc. to EN 60068-2-27		1000 m/s^2, 6 ms
Vibration resistance acc. to EN 60068-2-6		100 m/s^2, 55 ... 2000 Hz

Electrical characteristics

Output circuit		RS422 (TTL comp.)	Push-pull [2] (7272 comp.)	Open collector (7273)
Power supply		5 V DC (±5 %)	10 ... 30 V DC	10 ... 30 V DC
Power consumption with inverted signal (no load)		typ. 40 mA max. 90 mA	typ. 50 mA max. 100 mA	100 mA
Permissible load / channel		max. +/- 20 mA	max. +/- 20 mA	+/- 20 mA sink at 30 V DC
Pulse frequency		max. 250 kHz	max. 250 kHz	max. 250 kHz
Signal level	HIGH LOW	min. 2.5 V max. 0.5 V	min. +V - 2.0 V max. 0.5 V	
Rising edge time t_r		max. 200 ns	max. 1 µs	
Falling edge time t_f		max. 200 ns	max. 1 µs	
Short circuit proof outputs [3]		yes [4]	yes	yes
Reverse polarity protection of the power supply		no	yes	yes
UL approval		file 224618		
CE compliant acc. to		EMC guideline 2014/30/EU RoHS guideline 2011/65/EU		

1) Is only necessary when a special output signal format is required.
2) Max. recommended cable length 30 m [98.43'].
3) If power supply correctly applied.
4) Only one channel allowed to be shorted-out:
at +V= 5 V DC, short-circuit to channel, 0 V, or +V is permitted.
at +V= 5 ... 30 V DC, short-circuit to channel or 0 V is permitted.

Compact optical | **Sendix Base KIS40 / KIH40 (shaft / hollow shaft)** | **Push-pull / RS422 / Open collector**

Terminal assignment

Output circuit	Type of connection	Cable (isolate unused wires individually before initial start-up)								
3, 4, 6 with inv. signal	1, 2	Signal:	0 V	+V	A	$\overline{A}$	B	$\overline{B}$	0	$\overline{0}$
		Cable color:	WH	BN	GN	YE	GY	PK	BU	RD

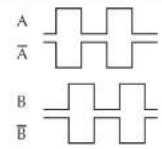
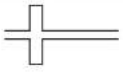
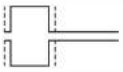
Output circuit	Type of connection	Cable (isolate unused wires individually before initial start-up)								
7, 8 without inv. signal	1, 2	Signal:	0 V	+V	A	–	B	–	0	–
		Cable color:	WH	BN	GN	–	GY	–	BU	–

Wiring Infomation

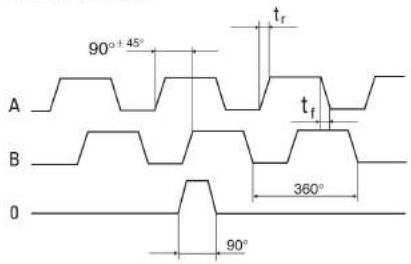
+V: Encoder power supply +V DC
0 V: Encoder power supply ground GND (0 V)
A, $\overline{A}$: Incremental output channel A
B, $\overline{B}$: Incremental output channel B
0, $\overline{0}$: Reference signal

Output signal formats

All Kübler encoders come standard with six channels where A leads B in the clockwise direction and the standard index is gated with A & B. The tolerance of the wave form affects the control and, in some cases, may affect the smoothness of system operation.

A leads B when the shaft is rotated in the clockwise direction viewing the shaft or collet end. This is the Kübler standard. This format applies to the pin key codes listed below.		
Order code ❶		
standard	0 gated with A & B. This is the Kübler standard. 0 is 90° wide.	
P03	0 ungated. 0 is 330° to 360° wide.	

Wave form tolerances



t_r = rising edge time
t_f = falling edge time

3.4.2 Quadrature Decoder system

Quadrature Decoder theory: A quadrature decoder contains two input channels, A and B. These are connected to the output channels A and B of the incremental encoder. Both of the encoder's channels have 0 and +5V states. If light passes through one of the slits in front of one of the channels, it will generate a +5V square pulse through that channel.

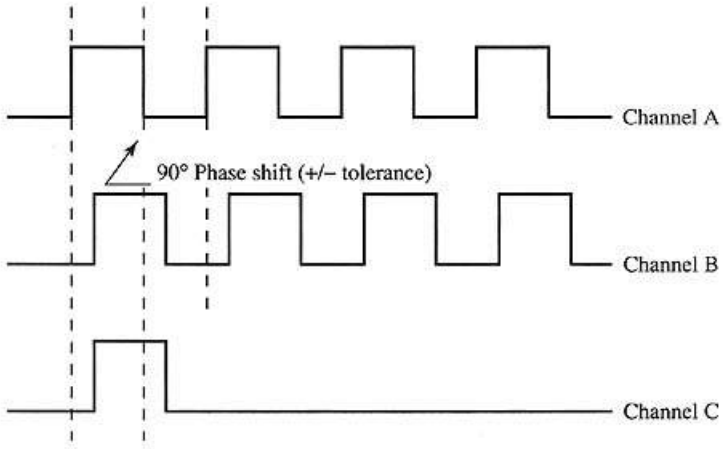
The quadrature decoder collects these pulses from the two encoder channels A and B. The encoder can fully update the decoder once every **quadrature cycle**, that is one pulse output from A and B (giving both change in angle and direction). This also corresponds as mentioned above to the smallest increment in angle the encoder is able to measure. If we know the total number of quadrature cycle outputs in 360 deg, we can infer the angular rotation from any number of quadrature cycles collected.

The encoders channels are staggered by 1/4 of a cycle, thus the word quad. Two channels with one offset from the other is needed to identify the direction of rotation. The direction of rotation (clockwise or counter-clockwise) is determined by the level of one signal during an edge transition of the second signal.

As indicated by the dashed line in figure 5, if channel A transitions from 0 to +5V while channel B is 0 or from +5V to 0 while channel B is +5V, this corresponds to a clockwise rotation. This is a result of channel B "leading" channel A.

If the roles were reversed and A led B, then a transition from 0 to +5V while channel B is +5V or from +5V to 0 while channel B is +0 would correspond to a counter-clockwise rotation.

Figure 17: The two channels A and B are phase shifted by 1/4 cycle. This offset enables the decoder to prescribe a direction. Channel C corresponds to the index pulse, also called Z which is output once every revolution.

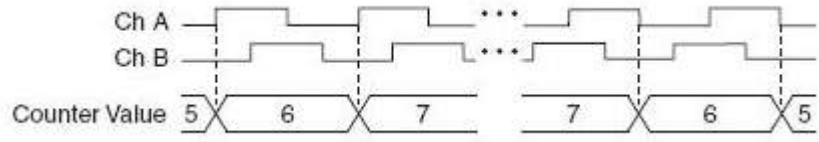


3.4.3 Pulses and the decoder counter

X1, X2, and X4 position encoding With quadrature output, three types of encoding can be used: X1, X2, or X4. The difference between these encoding types is simply which edges of which channel are counted during movement, but their influence on encoder resolution is significant.

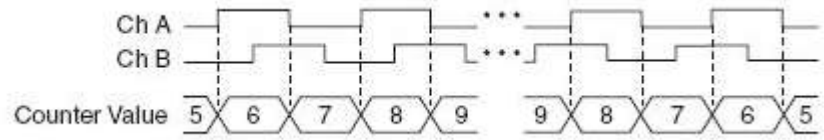
With X1 encoding, either the rising (aka leading) or the falling (aka following) edge of channel A is counted. If channel A leads channel B, the rising edge is counted, and the movement is forward, or clockwise. Conversely, if channel B leads channel A, the falling edge is counted, and the movement is backwards, or counterclockwise.[12]

Figure 18: X1 encoding



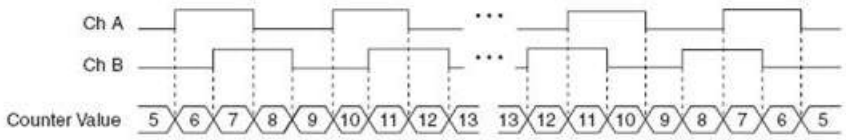
When X2 encoding is used, both the rising and falling edges of channel A are counted. This doubles the number of pulses that are counted for each rotation or linear distance, which in turn doubles the encoders resolution.[12]

Figure 19: X2 encoding



X4 encoding goes one step further, to count both the rising and falling edges of both channels A and B, which quadruples the number of pulses and increases resolution by four times.[12]

Figure 20: X4 encoding



In general, for a given quadrature edge count output Edge Count, encoding type x and encoder resolution N, the change in angle in degrees from the encoder $\Delta\theta$ is:

$$\Delta\theta = \frac{\text{Edge Count}}{\text{x}N} \times 360^\circ$$

Note: The quadrature decoder used in this experiment uses X4 encoding.

3.4.4 HCTL - 2001 - A00

Description The quadrature decoder chip used in this project was a HCTL - 2001 - A00 by AVAGO Technologies.

The decoder is a 12-Bit counter while the USB is 8-Bit. This incompatibility is solved by the chips ability to output the first 8 Bits as a Low Byte, followed by the final 4 Bits as a High Byte.

The following is a brief summary of the key features of the quadrature decoder used in this project:

- Interfaces Encoder to USB
- 14 MHz Clock Operation
- 8-Bit Tristate Interface
- 12-Bit Operating Modes
- X4 encoding

Figure 21: HCTL - 2001 - A00

Figure 22: Pinout A

Symbol	Name
D0	Output pin
CLK	Clock
SEL	Select
OE	Output Enable
RST	Rest
CH B	Channel B
CH A	Channel A
VSS	Source supply
VDD	Drain supply
D1-7	Output pin

CLK - Oscillator Clock The external signals from channel A and B from the encoder are synchronized with an internal clock provided by a signal generator seen in fig .

RST - Rest Pin This active low Schmitt-trigger input clears the internal position counter and the position latch. In this project, this pin was grounded and not used.

OE - Output Enable The Output Enable pin when active enables the tri-state output buffer. Since there are only 8 output pins on the decoder and the USB receiver, the 12-bit count value must be sent as two 8-bit outputs sequentially. The decision as to which of the two bytes are loaded onto thee tri-state output buffer is controlled by enabling of the SEL pin.

SEL - Select Pin The Select pin controls which of the two 8-bit outputs (High byte or Low byte) is loaded onto the 8-bit tri-state output buffer.

SEL	BYTE SELECTED
0	High
1	Low

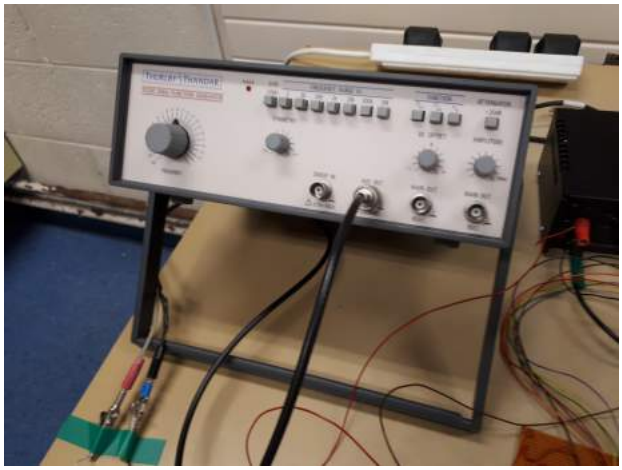
For a 12-bit counter, the possible values range from $0 - 4095$ before repeating.

For our encoder with resolution $N = \frac{360^\circ}{2500} = 0.144^\circ$, decoder type X4, the counter wraps around every:

$$\theta = \frac{4095}{4} \times 0.144^\circ = 147.42^\circ$$

This means that the decoder will cycle through $\Delta\theta$ until 147.42° after which it will return to 0°. To this end, the software must keep track of the total angle and know when to add or subtract 147.42° to that total.

Figure 23: Thurlby Thandar TG210 2MHz Function Generator



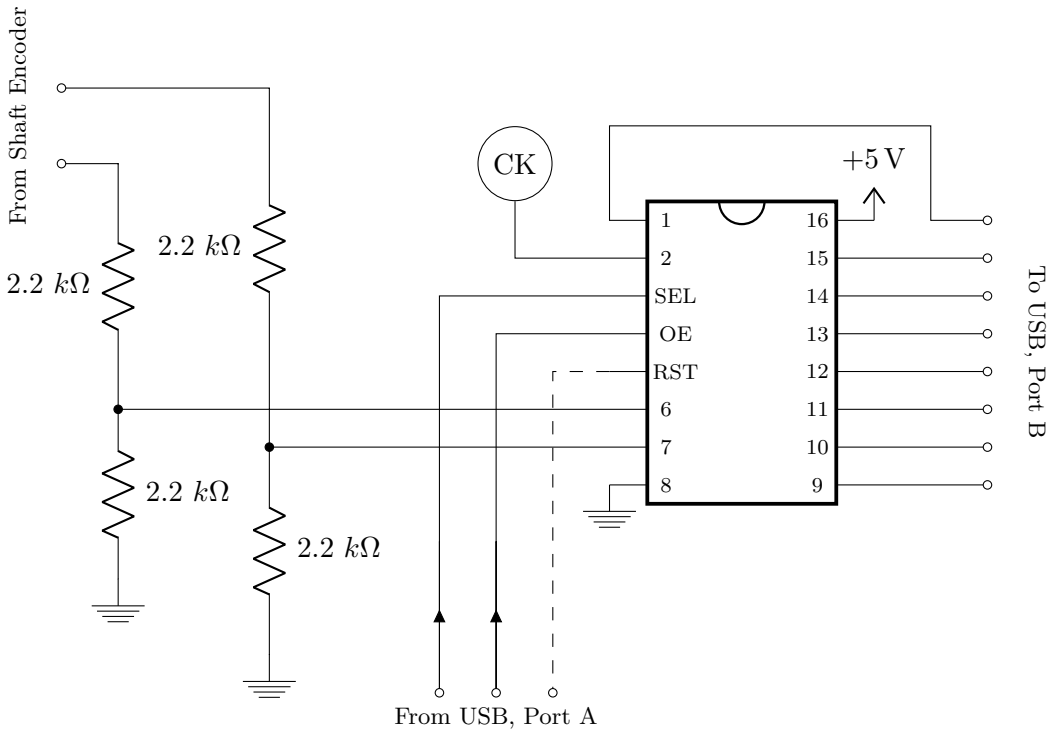


Figure 24: Rotary decoder circuit diagram.

The following table summarises the functional description for each pin used.

Pin	Symbol	Explanation	Wire Colour
1	D0	Bit Output	Black
2	CLK	Schmitt-trigger input for external clock signal	White
3	SEL	Enable either high byte (0) or low byte (1) to pass to USB	Black
4	OE	Enable tri-state buffer	Orange
5	RST	Enable counter to reset (connected but not used)	Red
6	CH B	Input from encoder	Gray
7	CHA	Input from encoder	Pink
8	VSS	Ground	Black
9	D7	Bit Output	Purple
10	D6	Bit Output	Blue
11	D5	Bit Output	Green
12	D4	Bit Output	Yellow
13	D3	Bit Output	Orange
14	D2	Bit Output	Red
15	D1	Bit Output	Brown
16	VDD	+5V	Red

Note: Though a bit confusing, encoder channel A (green) is wired to decoder Pin 7 (pink).

3.5 Measurement Computing USB

3.5.1 USB-1208LS

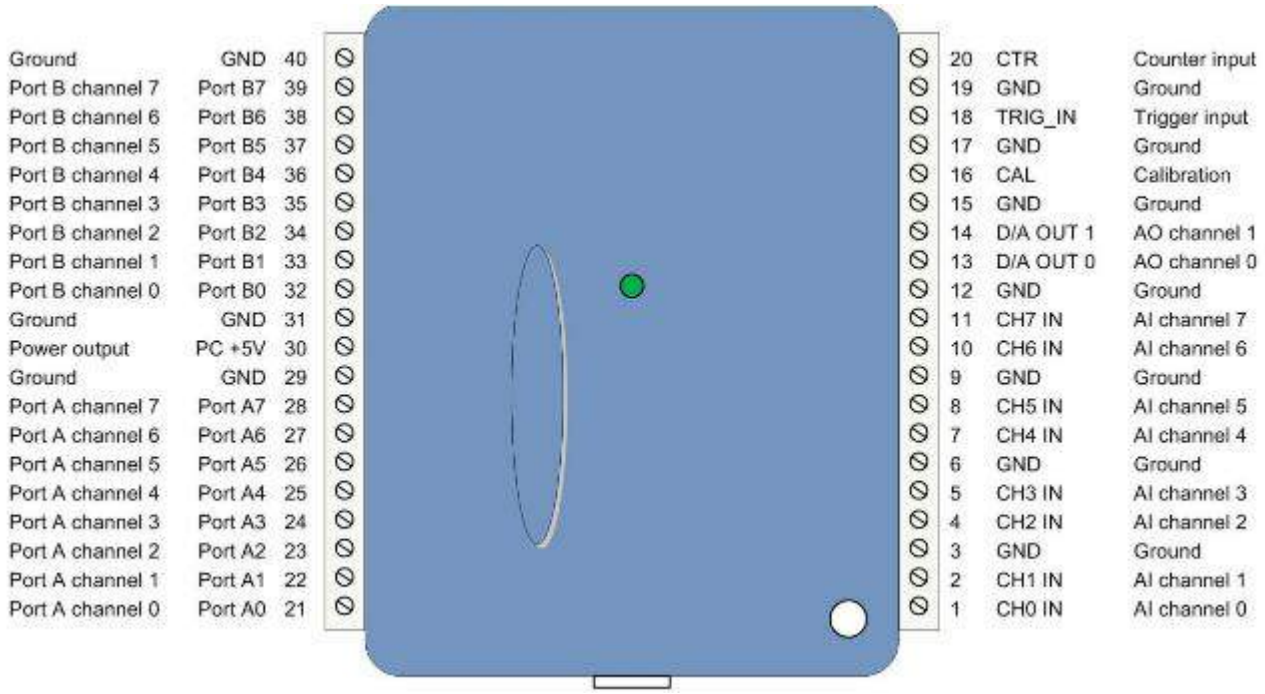


Figure 25: USB-1208LS pin diagram

The data acquisition device used in this project was a Measurement Computing USB-1208LS. The device is both a digital-to-analog and analog-to-digital converter.

The USB-1208LS is powered by the +5 volt USB supply from the computer. No external power was required.

InstaCal software is used to configure the device i.e the ports as single channels.

3.5.2 Particular Functions Used

The USB-1208LS Universal Libraries contains many functions. However for this project, only several key functions were needed. Their descriptions are given as follows.

cbDConfigBit()

Configures a specific digital bit as Input or Output. This function treats all DIO ports of the AUXPORT type on a board as a single port.

```
int cbDConfigBit(int BoardNum, int PortType, int BitNum, int Direction)
```

Arguments:

`BoardNum`

The number associated with the board when it was installed with the InstaCal configuration program. BoardNum may be 0 to 99.

`PortType`

The port (AUXPORT) whose bits are to be configured. The port specified must be bitwise configurable.

`BitNum`

The bit number to configure as input or output.

`Direction`

DIGITALOUT or DIGITALIN configures the specified bit for output or input, respectively.

---

cbDConfigPort()

Configures a digital port as input or output

```
int cbDConfigPort(int BoardNum, int PortNum, int Direction)
```

Arguments:

`BoardNum`

The number associated with the board when it was installed with the InstaCal configuration program. BoardNum may be 0 to 99.

`PortType`

The specified port must be configurable.

`Direction`

DIGITALOUT or DIGITALIN configures entire eight or four bit port for output or input.

---

cbFlashLED()

Causes the LED on a USB device to flash.

```
int cbFlashLED(int BoardNum);
```

Arguments:

`BoardNum`

The board number of the USB device whose LED will flash.

---

cbDBitOut()

Sets the state of a single digital output bit. This function treats all of the DIO ports of a particular type on a board as a single very large port. It lets you set the state of any individual bit within this large port. If the port type is not AUXPORT, you must use cbDConfigPort() to configure the port for output first. If the port type is AUXPORT, you may need to use cbDConfigBit() or cbDConfigPort() to configure the bit for output first.

```
int cbDBitOut(int BoardNum, int PortType, int BitNum, unsigned short BitValue)
```

Arguments:

`BoardNum`

The number associated with the board when it was installed with the InstaCal configuration program. BoardNum may be 0 to 99.

`PortType`

There are three general types of digital ports - ports that are programmable as input or output, ports that are fixed input or output and ports for which each bit may be programmed as input or output. For the first of these types, set PortType to FIRSTPORTA. For the latter two types, set PortType to AUXPORT.

`BitNum`

Specifies the bit number within the single large port. The specified bit must be in a port that is currently configured as an output.

`BitValue`

The value to set the bit to. Value will be 0 or 1. A 0 indicates a logic low output, a 1 indicates a logic high output. Logic high does not necessarily mean 5V.

---

cbAIn()

Reads an A/D input channel. This function reads the specified A/D channel from the specified board. If the specified A/D board has programmable gain then it sets the gain to the specified range. The raw A/D value is converted to an A/D value and returned to DataValue

```
int cbAIn(int BoardNum, int Channel, int Range, unsigned short
*DataValue);
```

Arguments:

`BoardNum`

The number associated with the board when it was installed with the InstaCal configuration program. BoardNum may be 0 to 99.

`Channel`

A/D channel number. The maximum allowable channel depends on which type of A/D board is being used. For boards with both single ended and differential inputs, the maximum allowable channel number also depends on how the board is configured.

`Range`

A/D range code. If the selected A/D board does not have a programmable gain feature, this argument is ignored. If the A/D board does have programmable gain, set the Range argument to the desired A/D range.

`DataValue`

Pointer or reference to the data value.

---

cbToEngUnits()

Converts an integer count value to an equivalent single precision voltage (or current) value. This function is typically used to obtain a voltage value from data received from an A/D with functions such as cbAIn().

```
int cbToEngUnits(int BoardNum, int Range, unsigned short DataVal, float *EngUnits)
```

Arguments:

`BoardNum`

The number associated with the board when it was installed with the InstaCal configuration program. BoardNum may be 0 to 99.

`Channel`

A/D channel number. The maximum allowable channel depends on which type of A/D board is being used. For boards with both single ended and differential inputs, the maximum allowable channel number also depends on how the board is configured.

`Range`

A/D range code. If the selected A/D board does not have a programmable gain feature, this argument is ignored. If the A/D board does have programmable gain, set the Range argument to the desired A/D range.

`DataVal`

An integer count value (typically, one returned from an A/D board)

`EngUnits`

The single precision voltage (or current) value that is equivalent to DataVal is returned to this variable. The value will be within the range specified by the Range argument using the resolution of the A/D on the board referenced by BoardNum (if any).

Returns:
EngUnits: the engineering units value equivalent to DataVal is returned to this variable.

3.6 Software

```
// This program:
// -controls the float relay circuit
// -reads in counts from quadrature decoder
// -converts those counts into angular position and velocity
// -keeps track of total angular position
// -stores measurements in a .txt file


#include <time.h>// time library for clock()
#include <cbw.h>
#include <stdio.h>
#include <conio.h>// measurement computing library
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <math.h>
#include <koolplot.h>// plotting package
#include <fstream>
#include <iostream>
#include <windows.h>

void InitialStates(void);// Initialises USB board

int BoardNum=0;// Assigns an identity number to the board. Only relevant when one is using multiple
                // USBs
int ULStat=0; // Variable to read the USBs pins and ports
int Chan=0;
int Gain=BIP10VOLTS;
WORD datavalue=0;
float EngUnits;
USHORT testin;
unsigned short BitValue=0;

/*****************************************************************************/

int main(int argc,char* argv[])
{
        float vout0, switchvalue, count, omega;
        float theta_i=0.0, theta_f=0.0;
        float d_t=0.0, d_theta;
        float theta_total, t_total;

    int ii, ndata;
    int bytelow, bytehigh=0;
    int thresholdlow=0, thresholdhigh=5, value=0, cycle_value=0, n=0;

    unsigned short BitValueHigh=1;
    unsigned short BitValueLow=0;
    unsigned int t1=0,t2=0;// to use the clock function, variable must be unsigned int

    const double Pi=2*acos(0.0);// Defines Pi for measurement of angular position in radians
```

```
// Enables user to create a .txt file to save measurements
ofstream myfile;
string filename;

cout<<"Please enter a file name to write: ";
getline(cin,filename);

filename += ".txt";

myfile.open (filename.c_str(), std::ios_base::app);

// Format of .txt file
myfile <<"index" << "\t\t\t" << "time" << "\t\t\t" <<"angle"<<
         "\t\t\t" <<"omega"<<endl;


InitialStates();// Initialise USB

plotdata xplot(0.0,1.0), yplot(0.0,1.0); // Initializes plotting variables (dummy args)
clear(xplot);
clear(yplot);

ndata=10000;// The number of measurements to take

ULStat=cbDBitOut(BoardNum,FIRSTPORTA,0,BitValueHigh); // Both OE and SEL pulled high
ULStat=cbDBitOut(BoardNum,FIRSTPORTA,1,BitValueHigh);

ULStat=cbDBitOut(BoardNum,FIRSTPORTA,2,BitValueLow);

// The following is a loop for the measurement of an angle

for (ii=0;ii<ndata;ii++){


     t1=clock();// Record time at start of measurement

     ULStat=cbAIn(BoardNum, Chan, Gain, &datavalue);

     ULStat=cbToEngUnits(BoardNum, Gain, datavalue, &EngUnits);

     vout0=EngUnits;

     switchvalue=vout0;

     // Condition to check if water level has dropped below level determined by float switch
    // and provide power to pumps if true
     if (vout0 = thresholdhigh)
        ULStat=cbDBitOut(BoardNum,FIRSTPORTA,2,BitValueLow);

     if (vout0 = thresholdlow)
        ULStat=cbDBitOut(BoardNum,FIRSTPORTA,2,BitValueHigh);
```

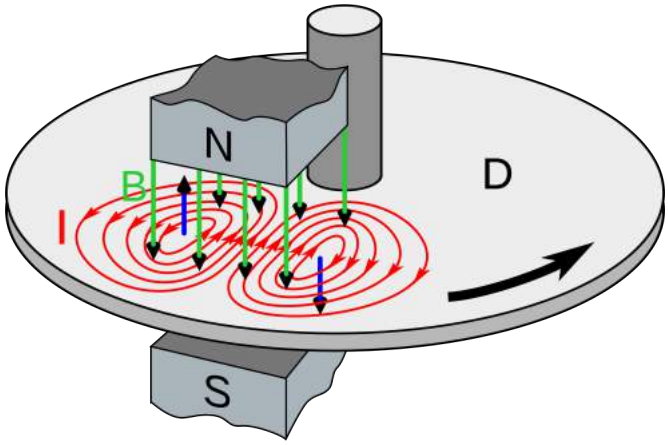
```
    // The following code reads the high byte, then the low byte
ULStat=cbDBitOut(BoardNum,FIRSTPORTA,0,BitValueLow); // Pull OE low

delay(0);// An option for a delay between measurements.

ULStat=cbDBitOut(BoardNum,FIRSTPORTA,1,BitValueLow); // Pull SEL low to read high byte

delay(0);

ULStat=cbDIn(BoardNum, FIRSTPORTB, &testin);

cycle_value=bytehigh;// This stores the previous highbyte value to be compared with the
                     // current one for determining if a cycle needs to be added to theta_f
bytehigh=testin;

ULStat=cbDBitOut(BoardNum,FIRSTPORTA,1,BitValueHigh); // Pull SEL high to read low byte

delay(0);

ULStat=cbDIn(BoardNum, FIRSTPORTB, &testin);

bytelow=testin;

t2=clock();// Record time at end of measurement

ULStat=cbDBitOut(BoardNum,FIRSTPORTA,0,BitValueHigh); // Both OE and SEL high
ULStat=cbDBitOut(BoardNum,FIRSTPORTA,1,BitValueHigh);

value=(bytehigh<<8)|bytelow;// Adds the High Byte and Low Byte together

// Condition to check if count output value has passed 4095 and rolled back around to 0.
// If true then add one cycle of angle (147.495 deg) to all future measurements.

// The condition is based on the fact that unless the wheel is rotating very fast, the
// highbyte does not change by much.

// The high byte cycles from 0-15 and back to 0. Therefore unless the wheel is rotating
// very fast, the only time the highbyte will change from 0 to 15 or 15 to 0 is when
// the count output value has passed 4095.

if (cycle_value-bytehigh>=9)
   n=n+1;
if (bytehigh-cycle_value>=9)
   n=n-1;

count = static_cast<float>(value);// Value must be recast as float for division
theta_f=(count*360/10000)+147.495*n;// Calculation to turn count into angular position

// For first datavalue since there is no previous theta_i but cannot be inititaied to 0
if (ii<=0)

    theta_i=theta_f;
```

```
        d_theta=(theta_f-theta_i)*(Pi/180);// Calculate d_theta in radians

        // Calculates d_t

        // CLOCKS_PER_SEC converts clock() function values to nearest second in seconds.

        // Dividing  CLOCKS_PER_SEC by 1000 gives clock() function values to nearest millisecond
                // in millisceonds

        // Division by 1000 again gives time in seconds to nearest millisecond.

        d_t=((t2-t1)/(CLOCKS_PER_SEC/1000))/1000.0;

        omega=(d_theta/d_t);// Calculates angular velocity

        theta_total=theta_f*(Pi/180); // Calculates total angular position in radians

        t_total=t_total+d_t; // Calculates total time

        theta_i=theta_f;

        // Plots omega vs time
        xplot << t_total;
        yplot << omega;

        // Writes time, angular position and angular velocity to .txt file
        printf("ii,Time,Angle,Omega(rads/s) %10i, %10f, %10f, %10f, \n",
                ii, t_total, theta_total, omega);
        myfile <<ii <<"\t\t\t"<<t_total <<"\t\t\t" << theta_total<<"\t\t\t"
               <<omega<<endl;

}
   myfile.close();// Close .txt file
   plot(xplot,yplot);// Display plot of omega vs time
   system("pause");


}

/*****************************************************************************/
// Initialises USB by configuring bits and ports as either inputs and outputs
void InitialStates(void)
{

        cbDConfigPort(0, FIRSTPORTA, DIGITALOUT);
        cbDConfigPort(0, FIRSTPORTB, DIGITALIN);

        cbDConfigBit(BoardNum,AUXPORT,0,DIGITALOUT);
        cbDConfigBit(BoardNum,AUXPORT,1,DIGITALOUT);
        cbDConfigBit(BoardNum,AUXPORT,2,DIGITALOUT);

        cbFlashLED(BoardNum);
}
```

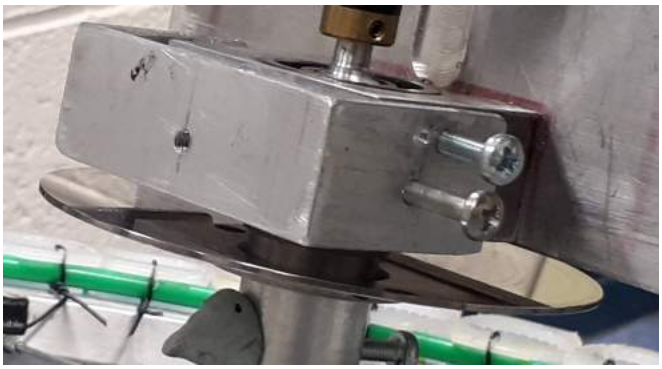
3.7 Magnetic Brake system

As discussed in Background Theory, axle torque with damping coefficient α is one of the adjustable parameters of the system. To vary α, a magnetic braking system (also called an **eddy current brake**) was devised. This section will discuss the theory and construction of such a brake. Its success will be discussed in the evaluation.

3.7.1 Theory



(a) The principle of a magnetic braking system



(b) The magnetic braking system used in project. The aluminium disc is shown. The fixed magnets are not shown.

An eddy current brake consists of a conductive sheet of (ideally non-ferromagnetic) material, usually copper or aluminium which moves through a magnetic field. The brakes usually come as either a linear piece of rail or a circular disc, that latter being the choice of this project.[15]

The disc rotates between the two poles of a magnet as seen in (a). As the disc rotates, the area within the magnetic field has **eddy currents** induced by **Faraday's Law**. These eddy currents swirl around inside the metal in a direction that, by **Lenz's Law**, induces a force that opposes the motion of the disc. This damping force is *proportional in the velocity* like all drag forces, with α being that proportionality constant. To vary α we must vary the drag force induced by the eddy currents.[16]

We would like to obtain a model that describes the torque that acts on the disc as a function of current, B-field, distance from magnet to disc etc. Unfortunately such a derivation is not trivial and could in itself be a project of its own. Although there proved no time to implement, the best and simplest approach is to experimentally obtain a model by taking measurements and fitting the data[4].

In this project, a hard disk drive platter was used for an aluminium disc. It was attached so that it co-rotated with the wheel. 6 small neodymium magnets were stacked, fixed to the frame and placed close to the disc. However for this project, a magnetic brake was not top priority and so qualitative rather quantitative measurements were made. As would be discovered, its effects were too small, indicating more magnets are needed.

4 Evaluation of the Chaotic Water Wheel

4.1 4-Bucket Water Wheel

The 4-bucket water wheel was explored using a simulation in Python. Following Matson, the inflow Q was modelled using the function[5]

$$Q_k = Q\cos^{1000}\theta_k/2$$

where $\theta_k = \theta + k2\pi/N$, k is the index of the bucket and N is he number of buckets. This results in a curve that resembles a thin stream of water.

Since we are dealing with a series of discrete buckets, each must be simulated separately and the resultant conservation of mass and balance of torques equations becomes more complicated[5]

$$\frac{dM_k}{dt} = Q\cos^{1000}\theta_k/2 - LM_k \tag{12a}$$

$$\frac{d\omega}{dt}\left(I_0 + \sum_{k=1}^{N} R^2 M_k\right) + \omega\sum_{k=1}^{N} R^2 Q\cos^{1000} Q_k/2 = R\sin\phi\sum_{k=1}^{N} M_k g\sin\theta_k - \alpha\omega \tag{12b}$$

We include below example plots from the program 4-Bucket Simulation and Animation. It is highly recommended that users run and explore the parameter space themselves. The plots parameter values were the following

- $N = 4$
- $dt = 0.01s$
- ntrials = 20000
- $t_{tot} = dt \cdot \text{ntrials} = 200s$
- $I_{wheel} = 0.1kgm^2$
- $R = 0.2m$
- $\phi = \pi/12$
- $g = 9.81ms^{-2}$
- $Q_0 = 0.039kgs^{-1}$
- $L = 0.13s^{-1}$
- $\alpha = 0.01$

The resultant motion is steady stable motion. Since the total mass of the system is not constant, we cannot package the above experimental parameters into two dimensionless parameters ρ and σ like we can for the 45-bucket water wheel. This means it is impossible to create a corresponding Parameter Map for the 4-bucket water wheel.

Also included in the program is the code to generate a **Python animation** also created using the *matplotlib.pyplot* package. The program helps to visualise the motion of the waterwheel by virtually simulating a water wheel to follow the same motion as the static plots.

Included are stills from an example animation.

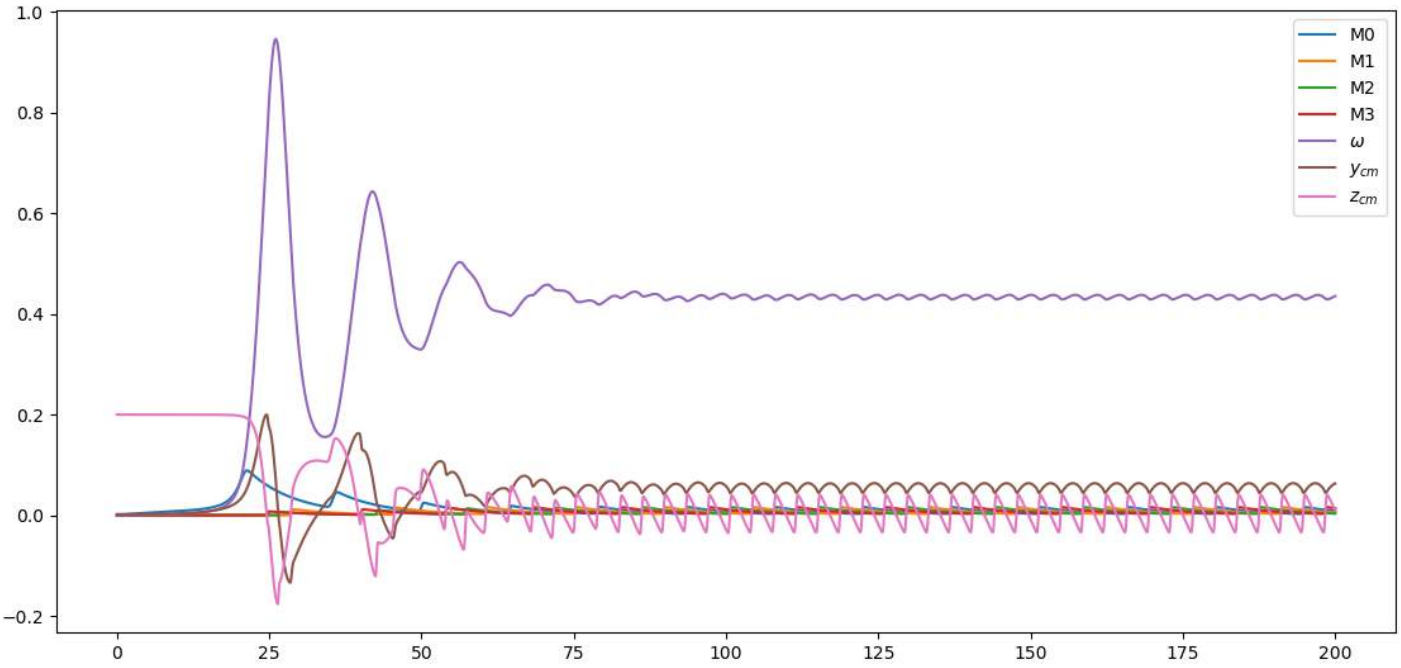


Figure 27: Motion of 4-bucket water wheel for parameter settings above

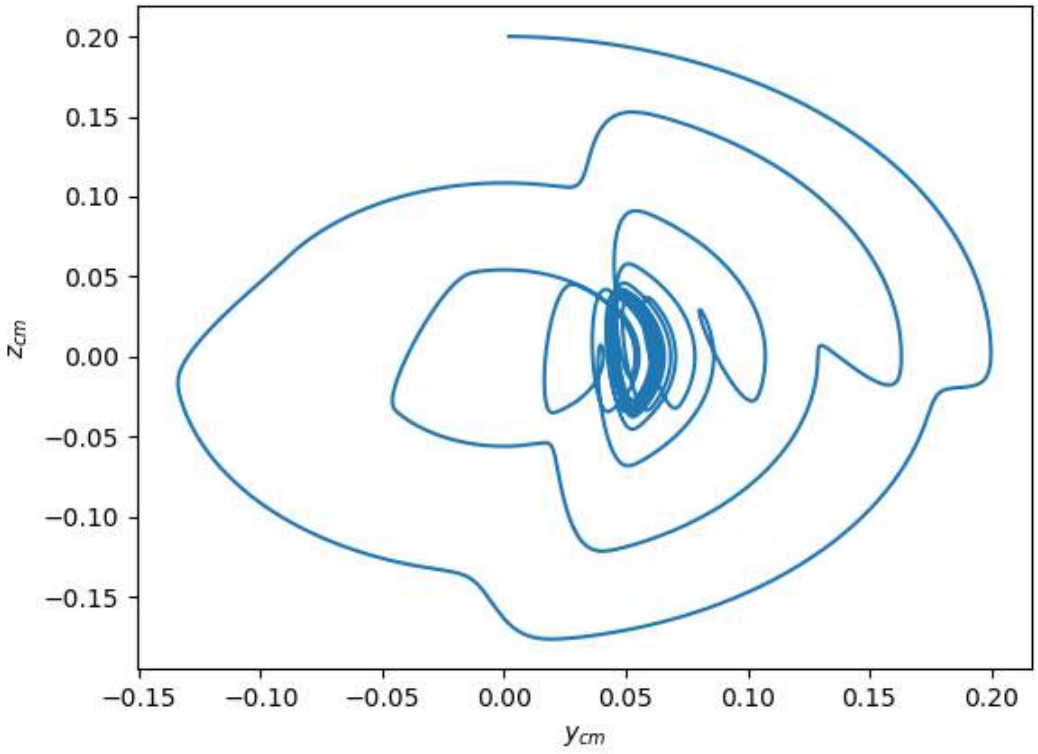
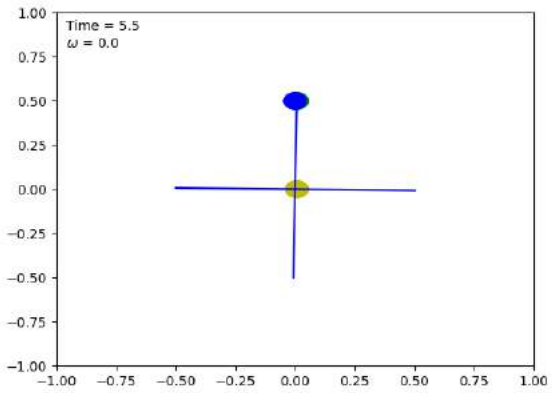
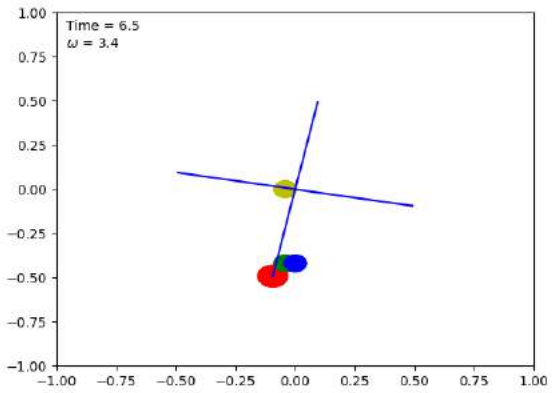


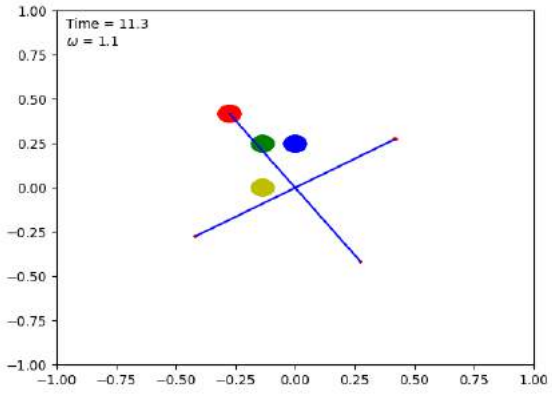
Figure 28: Phase space plot for 4-bucket water wheel



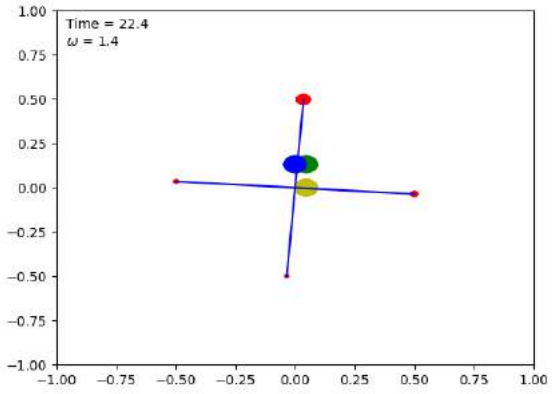
(a)



(b)



(a)



(b)

Figure 30: An example of a Python animation of a 4-bucket water wheel

4.2 45 Bucket Water Wheel

In the course of all experimental runs, not all parameters could be measured accurately or at all. This is of course a limitation. However the following is a list of estimated parameter values for a setting of the water wheel that resulted in periodic motion.

- Magnetic braking co-efficient ν: not known but ≈ 0.01 (due to smooth ball bearings between axle and wheel)
- Tilt angle $\alpha \approx 9°$ (experimentally measured)
- R $= 0.2m$ (experimentally measured)
- Leakage parameter $K \approx 0.13$ (measured by applying transformation $\omega = K \cdot x$ to simulation of x values and varying K until simulated ω fitted to ω data)
- $M_{tot} \approx 0.3kg$ (estimated by examining video footage of water wheel)
- $Q_{tot} = KM_{tot} = 0.039kgs^{-1}$
- $I_{wheel} = 0.1kgm^2$ (estimation based on Ill et al who used a similar sized wheel)
- $I_{tot} = I_{wheel} + M_{tot}R^2$
- $g = 9.81ms^2$ (acceleration due to gravity)

Recall

$$\sigma = \frac{1}{K}\frac{\nu + Q_{tot}R^2}{I_{tot}}, \quad \rho = \frac{Q_{tot}}{K^2}\frac{Rg\sin\alpha}{\nu + Q_{tot}R^2}$$

Here for simplicity we say $q_1 \cdot \pi = Q_{tot}$

From this we obtain $\rho \approx 273, \sigma \approx 0.2$.

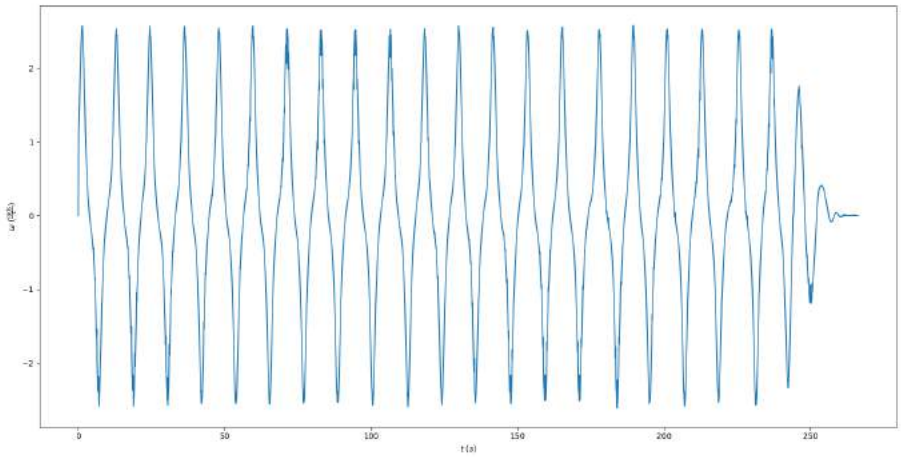


Figure 31: Experimental time series of ω vs t for parameter settings above. The sampling rate was 10Hz. The parameter settings correspond to $\rho \approx 273, \sigma \approx 0.2$. The motion is periodic and corresponds to the water wheel rotating back and forth.

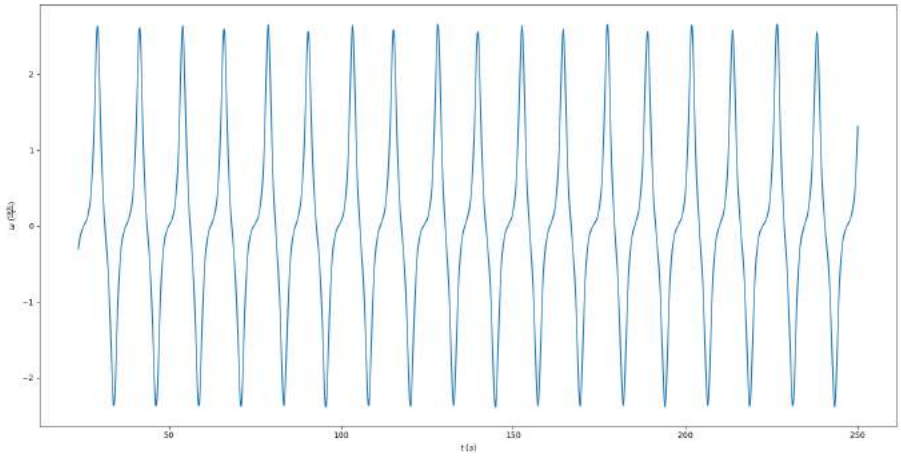


Figure 32: Simulated ω vs t achieved by solving Lorenz system for $\rho \approx 120, \sigma \approx 1.75$ and transforming x-co-ordinate to ω by $\omega = xK$ where $k = 0.13$. This choice of ρ and σ corresponds to periodic motion.

However this raises a problem. The calculated values of ρ and σ for the experimental data are too low to correspond to periodic motion. Examining the Parameter Map, we see that such values should correspond to steady motion whereas the motion obtained more clearly resembles Figure 28, whose values of ρ and σ put it clearly in a periodic region.

This therefore implies that the values of ρ and σ for the experimental data set are incorrect and that one or more experimental parameters must be measured to a greater degree of accuracy.

The tilt angle α was then raised to $\approx 14°$, keeping all other parameter values the same. The result was *steady* motion as seen in Figures below. The thickness of the curve is a result of a small variance $d\omega$ that shrinks in time as ω tends toward $2s^{-1}$.

By only changing the tilt angle α we are able to only change ρ. As a result the parameter values are $\rho \approx 422$ and $\sigma \approx 0.2$.

Comparing this to the expect mode of motion predicted by the Parameter Map the motion should indeed be steady. However as stated above, the values for ρ and σ are likely inaccurate due to one or more inaccurate experimental parameter values and as a results, this "correct" comparison cannot be trusted.

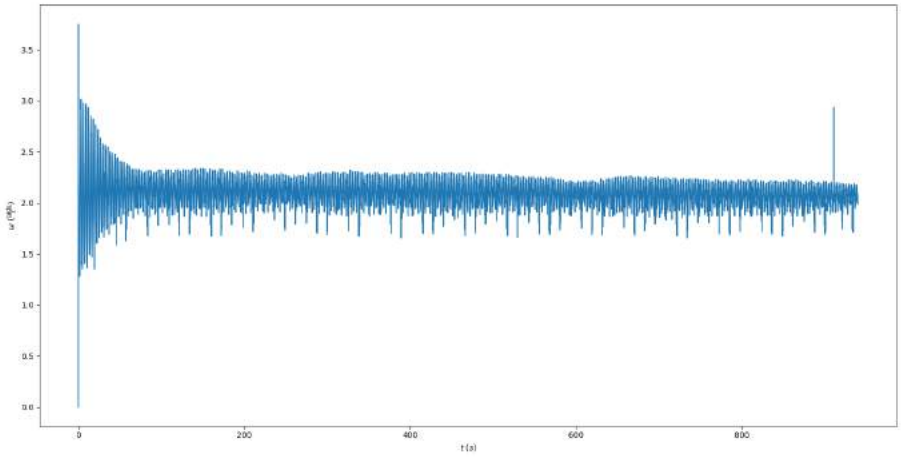


Figure 33: Experimental time series of ω vs t. All experimental parameters values are the same except for tilt angle α which is now $\approx 14°$. The sampling rate was 10Hz. The parameter settings correspond to $\rho \approx 423, \sigma \approx 0.2$. The motion is steady and corresponds to the water wheel rotating in a counter-clockwise unidirectional manner with approximately constant ω. Visible here are the inaccurate spikes in ω which are a consequence of the measurement devices and are not part of the motion

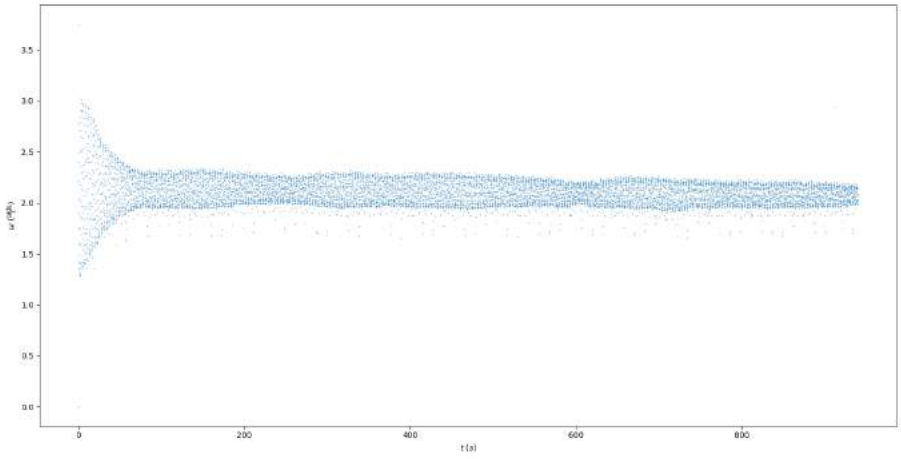


Figure 34: Same data as Fig 34 displayed in a scatter plot. By plotting in this way the values corresponding to inaccurate spikes in ω are not as greatly emphasised.

4.2.1 Problems Encountered in Measuring dt Accurately Using C++

The decision to plot both a continuous curve of values and a scatter plot is to highlight a problem that arose while collecting data for this and similar experimental runs while using the C++ platform.

To measure the time elapsed between subsequent angular measurements, the function

```
clock()
```

from the $< time.h >$ library was used along with a CLOCKS PER SEC macro to convert to milliseconds. However this proved to be a poor choice for a timer. The decision to use it was based on a lack of an alternative.

The *clock*() function in C++ can be used to measure the amount of processor time that a program consumes. However occasionally the time was less and as a result the measured dt (which should be $0.1s$) was less resulting in inaccurate spikes in ω as can be seen in Figure below.

While this can result in a messy graph if one plots the data as a continuous curve, the general trend can be seen more easily and these flaws more easily ignored if one plots the data as a scatter plot.

In future, it is highly recommended to phase out the *clock*() function and replace with either the high precision time keeping offered by the *chronos* library requiring C++ 11 or migrating the software used to collect data from C++ to Python. This will be further discussed in Improvements.

4.3 Magnetic Brake

The aluminium disc which would act as our magnetic brake was first attached to the 4-bucket water wheel. However before attaching, a series of tests were run on the wheel undamped so as to have a comparison.

The buckets were removed leaving only the wheel. There was assumed to be very little friction between the wheel and the axle. Starting from various initial angular velocity values ω_0, the wheel was allowed to freely rotate until it came to a stop. The motion was recorded and plotted.

The aluminium disc was attached to the the wheel so it co-rotated with the wheel. 6 small neodymium magnets were stacked onto of each other, brought a distance d to the wheel and attached to the frame. The wheel again was spun from various ω_0 and allowed to freely rotate until stopping. The motion was recorded for $d \approx 1cm$ and $d \approx 1mm$ and plotted.

The resulting motion can be seen below in Figures. The motion curves slightly as the wheel approaches $\omega = 0s^{-1}$ but most of the curve can be approximated linearly. In order to determine if the magnetic brake produces any noticeable effect , a line of best fit was added to each curve. If there was any noticeable drag from our 6 neodymium magnets, then, starting from approximately the same ω_0, the slope of the damped curve should be steeper then the slope of the undamped curve. This would correspond to the wheel stopping faster.

However comparing slope values across all Figures, one can see that there is little noticeable difference in slope for a ω_0.

Due to this, it was concluded that our 6 neodymium magnets are not strong enough to have a noticeable drag effect on the wheel without buckets let alone with 4 buckets filled water or worse an entire ring of buckets. With not enough magnets or time to pursue this section of the project any further, it was left unfinished for future completion by another. With no way to quantify α, we continued on under the reasonable assumption that due to our apparatus $\alpha \approx 0$.

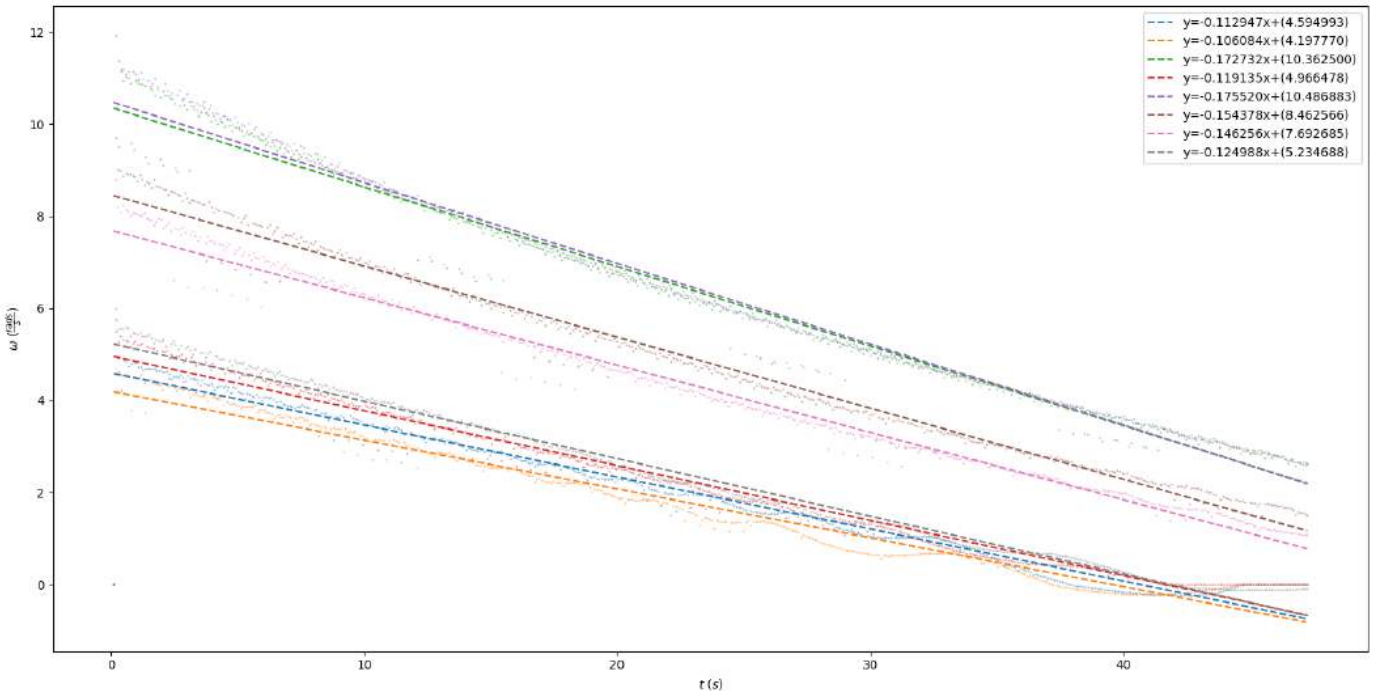


Figure 35: Undamped

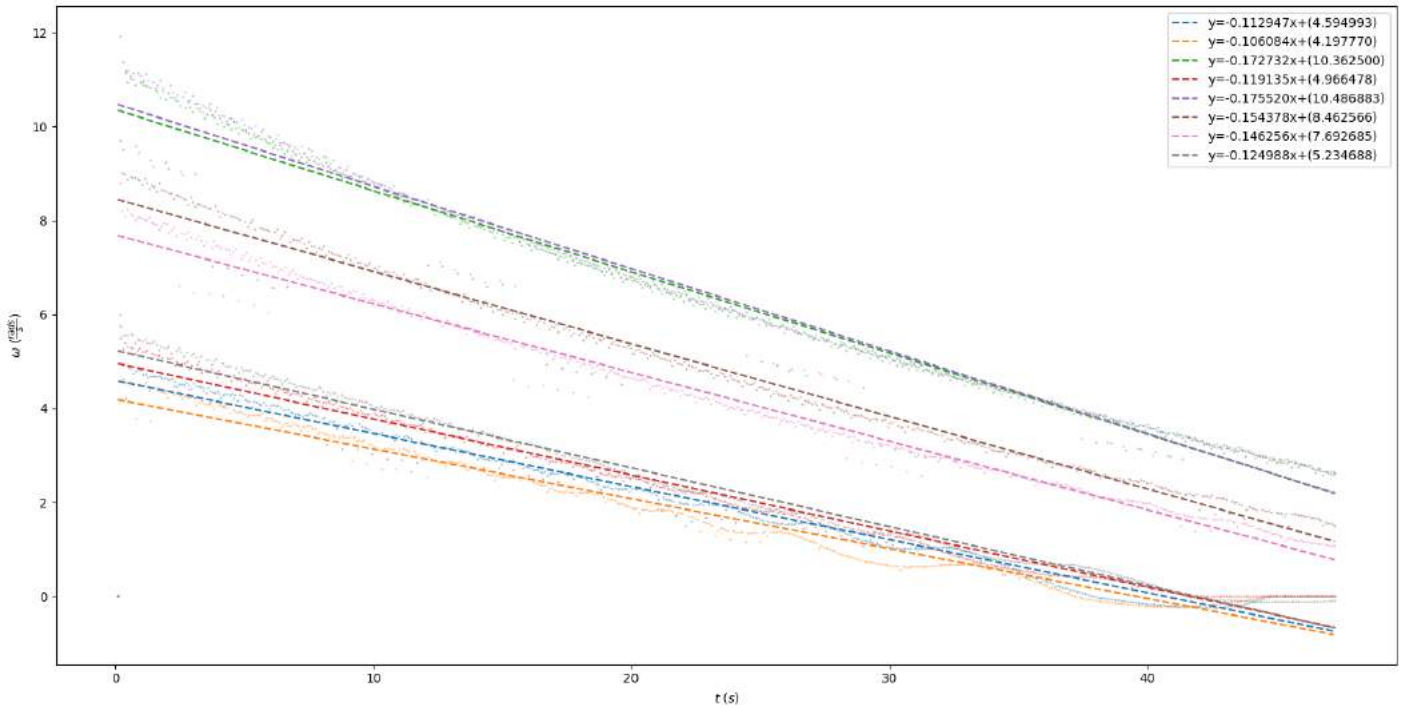


Figure 36: Damped $d \approx 1cm$

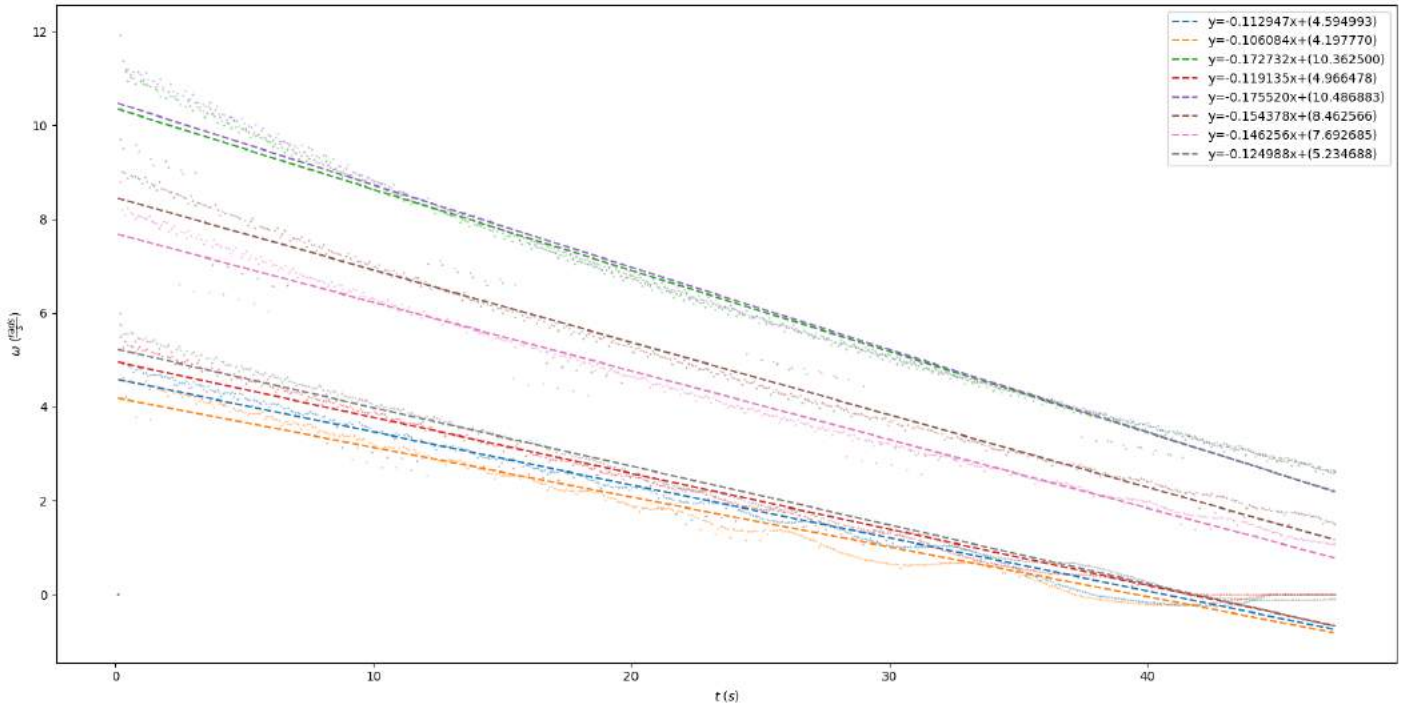
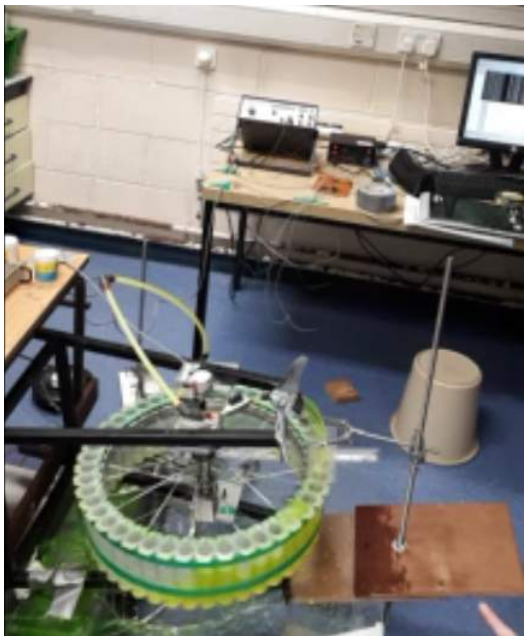


Figure 37: Damped $d \approx 1mm$

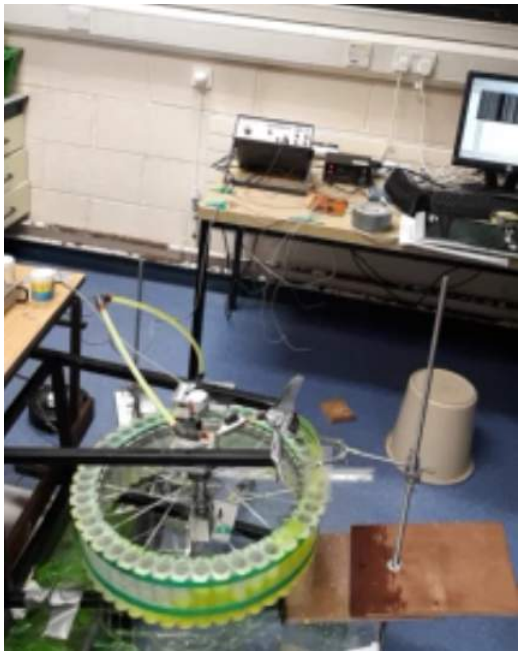
4.4 Determining M_{tot} By Video Processing

In the 45 bucket water wheel, M_{tot} approaches a constant. To estimate this mass, we set up a video recording device for the duration of an experimental run. In order to more easily measure the water content in each syringe, the water was dyed a florescence green colour. Knowing that the the volume of a syringe is 60ml, one can obtain an estimate for M_{tot} visually.

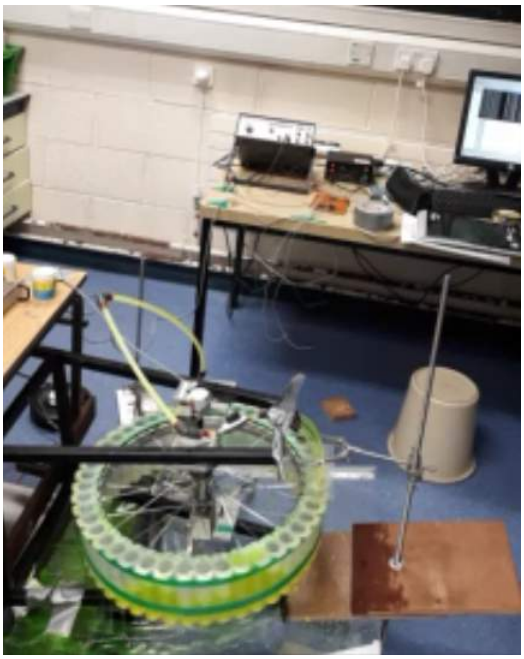
Below are four stills from footage taken $\approx$ 6 minute into the experiment. This particular run corresponds to the periodic motion plots in Figure. In this run only one camera was needed as effectively all of the mass was concentrated in a small number of syringes and thus could be estimated from one angle. In future tests, three or perhaps four cameras will be needed to cover all angles, assuming one wishes to vary Q_{tot} or K.



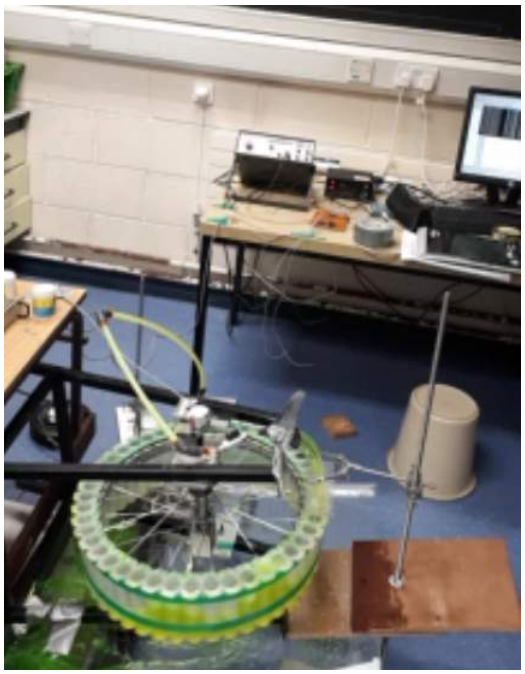
(a) $M_{mot} = 2(60) + 4(30) + 2(20) + 2(10) \approx 300ml \approx 3.0kg$



(b) $M_{mot} = 2(60) + 4(30) + 2(20) + 2(10) \approx 300ml \approx 3.0kg$



(a) $M_{mot} = 2(60) + 3(30) + 3(20) + 2(10) \approx 290ml \approx 2.9kg$



(b) $M_{mot} = 2(60) + 4(30) + 4(10) \approx 280ml \approx 2.8kg$

5 Improvements

The unideal 4 bucket water wheel and the ideal 45 bucket Malkus water wheel had both successes and failures. Certain aspects of the project proved too ambitious for the time frame this project was to be completed by (6 weeks). However a lot was accomplished in the time allocated and hopefully this section in particular can be used as a guide for future enhancements.

Here I will suggest a list of improvements to the apparatus for future users to implement.

5.1 Apparatus

- Implement a **flowmeter** - Attaching a flowmeter before the inflow tubes will give an easy way to accurately measure Q_{tot}.
- Further **balance the wheel** - We assume a symmetrical distribution of mass that around our wheel which results in I_{wheel}. While all efforts were made, the 45 bucket wheel is still slightly unbalanced due to the syringes. Rotating the wheel to certain positions will result in some roll back. Try to fully remove these imbalances by further adding *blue tac*. The 4 bucket water wheel should be balanced already
- Further **align rotary encoder** - The encoder is nearly aligned properly. Fully align it just to be sure.
- The 45 bucket wheel set up will **leak at high tilt**, $\alpha > 45^\circ$ into surroundings. Either correct or avoid such angles.
- As discussed earlier the magnetic braking system did not work. There were too few neodymium magnets to implement any noticeable drag. Implement a **magnetic braking system that does work** by using 4-5 times as many magnets placed on *both* sides of the aluminium disc. To maximise drag insure you cover as large an area of the disc as possible by placing them side by side not stacked together.
- **Fix the inflow tubes** on the 45 bucket wheel to the frame. This insures the tubes don't move such that the inflow is biased to a particular direction.
- With the current set up for the 4 bucket water wheel, the inflow is supplied from a single nozzle that is gravity fed from a basin above. That basin is in turn filled with water by means of the pump. This is in contrast to the 45-bucket wheel where water is supplied by several nozzles which are all fed directly from the pump. The basin water level must be kept constant in order to maintain a constant Q_{tot}. The pumps in turn must be turned on and off. **The float relay switch was not used in this project to achieve this**. Instead it proved more practical to do it by hand. However in either case Q_{tot} would not and was **not constant for the 4 bucket water wheel**. I recommended **directly pumping water into the 4-bucket water wheel** as was done for the 45 bucket water wheel. In the latter case it was very successful in achieving a constant Q_{tot}. Alternately introduce an overflow system into the basin and run the pumps constantly. This should insure a constant pressure head.
- The 4-bucket water wheel and the 45-bucket water wheel both used buckets that often did not fully leak due to air bubbles. Due to the number of buckets in the 45-bucket water wheel, it not matter if, at any given time up to 4 or 5 were not leaking properly as the effect of one bucket was not as severe in the context of 45. However for the 4-bucket water wheel, it does and did make a difference. Try to implement **buckets which fully leak and don't allow air bubbles**.
- The 4-bucket water wheel buckets did not span all angles. The gaps between meant that M_{tot} was not constant. This further complicated the model and made it difficult to make any predictions. Further more the experimental parameters could not be tied into to two nice dimensionless parameters ρ and σ which we could construct a parameter map for. To simplify things, I recommend introducing **pyramidal or conical buckets for the 4-bucket water wheel**. The shape of the buckets is such that the buckets can still remain upright without bumping into each other.

- Florescence dye was introduced so M_{tot} could be more accurately measured by video processing. Being florescences, it glows under UV-light. Try attaching **UV LEDs to the water wheel** or constructing an array of LEDs to place underneath the water wheel and film in the dark.

- The steel rods used to attach the buckets to the wheel were galvanised with a thin layer of lacquer. **Galvanise the wheel itself** and any rods that need redoing.

5.2 Software

- The software that is used to collect data from the MSC USB is written in C++. There is several disadvantages to this, chief among them being the lack of good plotting software. Once the data is collected, it has to be stored in a .txt and transferred to a Python environment before being plotted with *matplotlib.pyplot*. It would be easier to cut out these middle steps by writing this code and collecting the data entirely in Python. This would require a Python equivalent universal library for the MSC USB. Luckily someone has written a library - **PyUniversalLibrary**. The link[17] is provided in References. With this, a Python equivalent program can be written to control all the electronics.

- If one chooses to switch entirely to Python, then one can avail of **real-time plotting software**. Among the different choices out there (including writing your own using Python animation software!), the link[18] in References leads to an excellent dynamic graph program.

- If one wishes to stay in C++, then a **more stable timer function** is needed. As stated above, the *clock()* function is inadequate and (occasionally) slightly inaccurate. If one updates to C++11, one can avail of the *chronos* library, which has time measurement functions accurate down to nanoseconds. It is the most recommended library for time measurement in C++.

5.3 Methods

Here I suggest an organised approach for investigating the 45-bucket Malkus water wheel. The purpose of this project is to evaluate the Malkus-Howard-Lorenz chaotic water wheel. By this we mean - Is our water wheel described by the Malkus water wheel equations? These equations are in turn a subset of the Lorenz equations. We cannot hope (in general) to be able to obtain the exact same plots in our data as our simulations produce since for many settings the motion is chaotic and therefore incredibly sensitive to initial conditions. I propose the following method to determine if our system is indeed described by the Lorenz equations.

- Go to the Parameter Map and select values for ρ and σ. Now we can make a prediction as to what type of motion we will get and the associated Lyapunov value along with it.

- Configure the water wheel's experimental parameters for the selected values of ρ and σ.

- Run the experiment, allowing the hardware to collect data on the water wheel's motion for several hours.

- Compare actual motion with the motion predicted by the Parameter Map. Is it chaotic, periodic or stable?

- Now the difficult part. How do we know for certain that our data is indeed chaotic, periodic or stable? For that we need to estimate the Largest Lyapunov Characteristic Exponent (LLCE) for our data set. Several approaches have been put forward in recent years to do so. I have included[19][20] some suggested methods in References. If calculated accurately, it will confirm if the data collected is indeed chaotic, periodic or stable.

- Compare calculated LLCE with predicted LLCE from Parameter Map.

- Keeping σ fixed, slowly move along a line in the Parameter Map space by varying ρ. Do this by varying the tilt angle α. For each new value of ρ, repeat the above steps.

If all comparisons match, then it is highly likely that our 45-bucket water wheel obeys the Malkus water wheel equations.

6 Conclusion

6.1 Construction

Both apparatuses were successfully constructed. The buckets used for the 4-bucket wheel could be improved but the syringes used for the 45-bucket wheel were successful. The rotary encoder decoder system required to measure the angular position of the wheel was correctly implemented and worked perfectly, as did the MSC USB interface. The pump systems implemented also succeeded but the pump system corresponding to the 45-bucker water wheel worked better. The magnetic braking system was implemented but proved to not be a success. It is hoped that further work can be done to improve this important aspect of the system in the future.

6.2 Evaluation

The 4-bucket water wheel had both successes and failures. The successes were in the simulations and animations. The failures were in being unable to control accurately most of the experimental parameters. The resultant motion was almost always *chaotic*. The 45-bucket water wheel was more of a success but was built too late to preform many tests. It too suffered from being difficult to measure certain experimental parameters accurately, most noticeably Q_{tot} and α. However both *stable* and *periodic* motion was found even if the calculated values of ρ and σ were inaccurate.

It is too early to say if our 45-bucket water wheel truly obeys the Malkus water wheel equations. Further tests must be conducted and accurate measurements of its experimental parameters must be made. However a lot was accomplished in just the 6 weeks allocated for this project. The Malkus-Howard-Lorenz water wheel is an incredibly complex system and the 4-bucket water wheel we built even more so. I hope that some suggestions put forward in Improvements will be implemented and bring us closer to answering the question

Does our 45-bucket water wheel obey the Malkus water wheel equations?

7 Gallery

8 References

References

[1] Julien Clinton Sprott *Honours: A Tribute to Dr. Edward Norton Lorenz* http://sprott.physics.wisc.edu/lorenz.pdf

[2] Edward N. Lorenz *Deterministic Nonperiodic Flow* Journal of The Atmospheric Sciences Volume 20, 130

[3] Steven H. Strogatz *Nonlinear Dynamics and Chaos* Westview Press; 1 edition (January 19, 2001)

[4] Lucas Illing et al *Experiments with a Malkus-Lorenz water wheel: Chaos and Synchronization* Am. J. Phys. 80, 192(2012); DOI: 10.1119/1.3680533

[5] Leslie E. Matson *The Malkus-Lorenz water wheel revisited* Am. J. Phys. 75 (12), December 2007; DOI: 10.1119/1.2785209

[6] James Gleick *Chaos: Making a New Science* Penguin Books 1987

[7] Edward Pode *Modelling the Chaotic Waterwheel* Mathematics 4th Year Dissertation 2014/15 School of Mathematical Sciences University of Nottingham

[8] Macro Sandri *Numerical Calculation of Lyapunov Exponents* The Mathematical Journal 84, 1996

[9] Clyde-Emmanuel Estorninho Meador *Numerical Calculation of Lyapunov Exponents for Three-Dimensional Systems of Ordinary Differential Equations* Theses, Dissertations and Capstones. Paper 107. 2011

[10] *Calculating the entire Lyapunov Spectra of the Lorenz Attractor* http://www.math.tamu.edu/ mpilant/math614/Matlab/Lyapunov/LorenzSpectrum.pdf

[11] *Part H: Quantifying Chaos* http://csc.ucdavis.edu/ chaos/courses/nlp/Software/partH.html

[12] *National Instruments: Encoder Measurements: How-To Guide* http://www.ni.com/tutorial/7109/en/

[13] *Basics of Rotary Encoders: Overview and New Technologies* http://www.machinedesign.com/sensors/basics-rotary-encoders-overview-and-new-technologies-0

[14] *Rotary Encoders* https://www.ia.omron.com/support/guide/34/introduction.html

[15] *Magnetic braking: Simple theory and experiment* H.D Wiederick, N. Gauthier, D. A. Campbell and P. Rochon Am. J. Phys 55,500-503(1987)

[16] *Magnetic braking: Improved theory* M. A. Heald Am. J. Phys 56, 521-522(1988)

[17] The link to the Python MSB Universal Library. https://pypi.python.org/pypi/PyUniversalLibrary

[18] The link to the dynamic plotting software. http://eli.thegreenplace.net/2008/08/01/matplotlib-with-wxpython-guis/

[19] *A practical method for calculating largest Lyapunov exponents from small data sets* Michael T. Rosenstein, James J. Collins and Carlo J. De Luca Elsevier Science Publishers 1993

[20] *Determining Lyapunov exponents from a time series* Alan Wolf, Jack B. Swift, Harry L. Swinney and John A. Vastano Elsevier Science Publishers 1984

9 Appendix

In this section we present the derivation of the water wheel equations of motion. There are two different approaches as outlined by Strogatz[3] and Matson[5]. Here we consider both.

9.1 Strogatz Approach

The Strogatz's derivation models the water distribution as a continuum.

Symbol	Meaning
ω	Angular velocity of the wheel
K	Leakage rate
Q	Water inflow
R	Radius of wheel
I_{tot}	Inertia of wheel and water in cups
$g \sin \alpha$	Effective gravity
a_1	Fourier coefficient - 1st mode
b_1	Fourier coefficient - 1st mode
ν	Viscous damping coefficient
q_1	Inflow Fourier coefficient - 1st mode

Table 1: Notation used in Strogatz derivation

Examining the sector $[\theta_1, \theta_2]$ in the diagram, the mass M in that sector is

$$M(t) = \int_{\theta_1}^{\theta_2} m(\theta, t) d\theta$$

After a time Δt, the change in mass ΔM is given by

$$\frac{\partial m(\theta, t)}{\partial t} = Q(\theta) - km(\theta, t) - \omega(t) \frac{\partial m(\theta, t)}{\partial \theta}$$

where

- $Q(\theta)$ is the flow rate from nozzles. Illing et al modelled their flow by the following function
- $-km(\theta, t)$ is the leakage rate
- $\omega(t) \frac{\partial m(\theta,t)}{\partial \theta}$ represents the change in mass of the sector we are considering due to the rotation of the wheel.

This is just a local conversation of mass equation for our system - a **continuity equation**.

The first equation considers how the mass in the system changes as the wheel rotates. The second equation is a balance of torques that describes how the angular velocity of the wheel changes.

In general the angular velocity of the wheel depends on I_{tot} which depends on the total mass of water in the wheel. However all of the water from the nozzles enters the wheel, so the total mass of the wheel is determined simply by a balance of flows

$$\frac{dM_{tot}}{dt} = -kM_{tot} + Q$$

whose solution is

$$M(t) = (Q/k)(1 - e^{-kt})$$

which shows that as $t \to \infty$, $M_{tot} = Q/k = \text{const}$

So allowing for transients to pass, the total mass is a constant and so the total inertia is a constant so the equation of motion is

$$I\dot{\omega} = \text{magnetic braking torque} + \text{spin-up torque} + \text{gravitational torque}$$

- **magnetic braking torque**: We assume that (due to the presence of ball bearings), the angular rotation between the axle and wheel is frictionless. As discussed in Constructing the Water Wheel, one source of damping that can be added is a magnetic brake, which adds a term

  $$\tau = -\nu\omega$$

- **spin-up torque**: Water enters at zero velocity and is spun up to ω contributing a torque term

  $$\tau = -QR^2\omega(t)$$

- **gravitational torque**: The wheel becomes top heavy when water is pumped in at the top resulting in a infinitesimal torque element

  $$d\tau = (dM)gr\sin\theta = mgr\sin\theta d\theta$$

  .

  By varying the angle of inclination α, we can vary the effective gravity $g = g_0 \sin\alpha$, where $g_0 = 9.81 m/s^2$

  Integrating over all of the water wheel $[0, 2\pi]$, we obtain

  $$\tau = g_0 r \sin\alpha \int_0^{2\pi} m(\theta, t)\sin\theta d\theta$$

Collecting terms, we find that the equation of motion is

$$I\dot{\omega} = -\nu\omega - QR^2\omega + g_0 r\sin\alpha \int_0^{2\pi} m(\theta, t)\sin\theta d\theta$$

The key insight made by Strogatz was to rewrite our mass conservation equation and our torque equation using Fourier Analysis by noting that $m(\theta, t)$ is periodic in θ

$$m(\theta, t) = \sum_{n=0}^{\infty} [a_n(t) \sin n\theta + b_n(t) \cos n\theta]$$

where $a_n(t)$ and $b_n(t)$ are Fourier coefficients or amplitudes corresponding to the different *harmonics* of the system.

Using the key assumption that Q is added *symmetrically*, with respect to the wheel's center line, the inflow may be rewritten as

$$Q(\theta) = \sum_{n=0}^{\infty} q_n \cos n\theta$$

In doing so one obtains an infinite set of amplitude equations. However when one examines the torque balance equation rewritten with the Fourier substitute for $m(\theta, t)$

$$I\dot{\omega} = -\nu\omega - QR^2\omega(t) + g_0 r \sin\alpha \int_0^{2\pi} \left(\sum_{n=0}^{\infty} [a_n(t) \sin n\theta + b_n(t) \cos n\theta] \right) \sin\theta d\theta$$

the integral

$$\int_0^{2\pi} \left(\sum_{n=0}^{\infty} [a_n(t) \sin n\theta + b_n(t) \cos n\theta] \right) \sin\theta d\theta$$

evaluates to

$$\int_0^{2\pi} a_1 \sin^2\theta d\theta = a_1$$

since by orthogonality all other terms evaluate to 0.

This result is rather significant. It implies $a_1(t)$, $b_1(t)$ and $\omega_1(t)$ form a closed system. That is, the angular velocity can be found by solving for $a_1(t)$ and $b_1(t)$, despite requiring to solve all other infinitely many amplitude equations to describe the exact mass distribution at any given t.

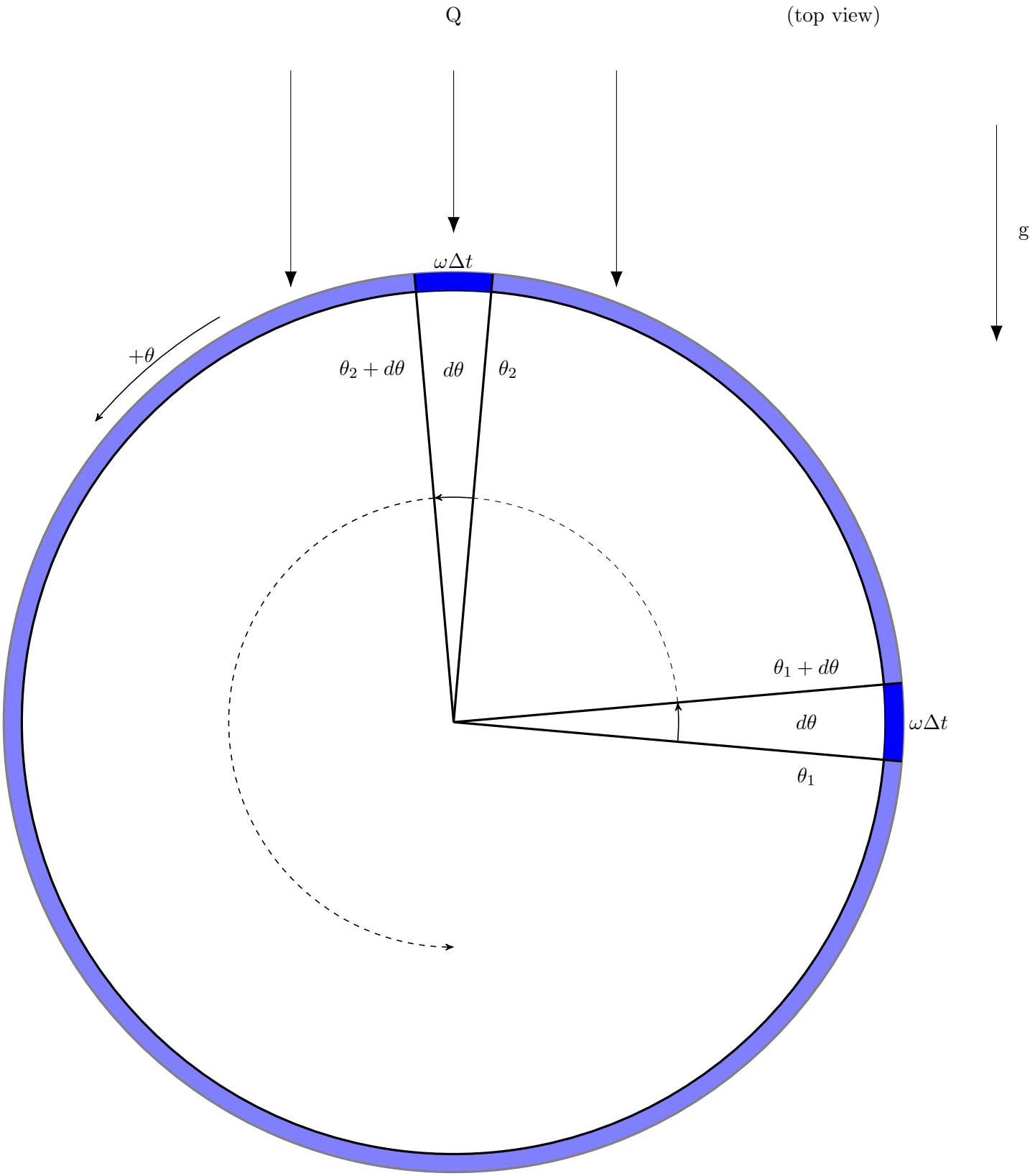
Therefore the infinite dimensional ODE system is reduced to a three dimensional ODE system given as

$$\frac{d\omega}{dt} = -\frac{\nu + Q_{tot}R^2}{I}\omega(t) + \frac{\pi R g \sin\alpha}{I} a_1(t)$$

$$\frac{da_1}{dt} = -Ka_1(t) + \omega(t)b_1(t)$$

$$\frac{db_1}{dt} = q_1 - Kb_1 - \omega(t)a_1(t)$$

9.1.1 Strogatz Derivation Diagram



9.2 Matson Approach

The Matson derivation models the water distribution as a ring of *discrete* cups.

Symbol	Meaning
ω	Angular velocity of the wheel
λ	Leakage rate
Q	Water inflow
$M = Q/\lambda$	Mass of water (after transients)
R	Radius of wheel
I	Inertia of wheel and water in cups
$g \sin \phi$	Effective gravity
I_w	Inertial of water
α	Viscous damping coefficient
y	Position co-ordinate
z	Position co-ordinate

Table 2: Notation used in Matson derivation

The Matson approach begins by deriving the same result shown in the Strogatz derivation. That is, for an ideal water wheel, M approaches $Q/\lambda = constant$.

The motion of the center of mass (COM) of the system is considered by the individual components y and z. The center of mass of of all of the water mass component wise is

$$y = \frac{\sum(M_k y_k)}{\sum M_k} \quad z = \frac{\sum(M_k z_k)}{\sum M_k}$$

where k is the index number of the cup

To derive the equations of motion of the COM, Matson uses a clever trick. At a time t_0, the cups are closed so no more water can pour into them (but can still leak) and at the same time opens an identical set of cups on the wheel to allow water to pour into.

Only the total mass M is necessary to consider so despite the separation of masses into two smaller subsystem, the wheels motion carries on oblivious.

Now we consider the motion of the combined COM of these two subsystems

$$y = \frac{M_a y_a + M_b y_b}{M_a + M_b} \quad z = \frac{M_a z_a + M_a z_b}{M_a + M_b}$$

Taking the derivative of both components

$$\frac{dy}{dt} = \frac{1}{M}\left(M_a \frac{dy_a}{dt} + \frac{dM_a}{dt} y_a + M_b \frac{dy_b}{dt} + \frac{dM_b}{dt} y_b\right) \tag{13a}$$

$$\frac{dz}{dt} = \frac{1}{M}\left(M_a \frac{dz_a}{dt} + \frac{dM_a}{dt} z_a + M_b \frac{dz_b}{dt} + \frac{dM_b}{dt} z_b\right) \tag{13b}$$

We now examine each of our subsystem's COM separately. We define a as the index of cups that were filled prior to $t = t_0$ before having their tops closed. We define b as the index of cups that had their tops closed and were empty prior to $t = t_0$.

At $t = t_0$ we consider the COM (y_a, z_a) of M_a. The COM of M_a will rotate with the wheel so we can equally describe (y_a, z_a) in polar co-ordinates

$$r = \sqrt{y_a^2 + z_a^2}, \quad \theta = \arctan \frac{y_a}{z_a}$$

where r the radius is a constant,$d\theta/dt = \omega$ and velocity v of M_a is $v = \omega r$.

Decomposing v into its y and z components we get

$$\frac{dy_a}{dt} = v \cos\theta = \omega z_a \quad \frac{dz_a}{dt} = -v \sin\theta = -\omega y_a$$

For initial conditions of $M_a = M$ at $t = t_0$, with the COM of M_a at some point $(y_a, z_a) = (y, z)$

$$\frac{dM_a}{dt} = -\lambda M \tag{14a}$$

$$\frac{dy_a}{dt} = \omega z \tag{14b}$$

$$\frac{dz_a}{dt} = -\omega y \tag{14c}$$

The motion of the COM of M_b can be decomposed the same way.

At $t = t_0$, $M_b = 0$, the top bucket is beginning to fill so the COM of M_b is at $(y_b, z_b) = (0, R)$.

However at $t = t_0$, the angular velocity of M_b is zero so the resulting equations for M_b are

$$\frac{dM_b}{dt} = -\lambda M \tag{15a}$$

$$\frac{dy_b}{dt} = (0)R = 0 \tag{15b}$$

$$\frac{dz_b}{dt} = -(0)(0) = 0 \tag{15c}$$

Substituting (15),(16) into (14) gives the COM of the combined system

$$\frac{dy}{dt} = \omega z - \lambda y \tag{16a}$$

$$\frac{dz}{dt} = -\omega y + \lambda(R - z) \tag{16b}$$

Finally since value $t = t_0$ is arbitrary, (17) holds for all t.

Matson's approach to deriving the equation of motion through a balance of torques is essentially the same as Strogatz's except with different notation.

The total moment of inertia I is constant since M is constant after transients. The torque contributions are N_g (gravity acting on water), N_μ (axle friction) and N_w (spin- up incoming water to speed of wheel)

$$N_g = Mgy \sin\phi \tag{17a}$$

$$N_\mu = -\alpha\omega \tag{17b}$$

$$N_w = -\omega\lambda I_w \tag{17c}$$

giving

$$I\frac{d\omega}{dt} = N_g + N_\mu + N_w = Mgy \sin\phi - \alpha\omega - \omega\lambda I_w \tag{18}$$

The final equation is thus

$$\frac{d\omega}{dt} = \frac{Mg \sin\phi}{I} y - \left(\frac{\alpha}{I} + \lambda\frac{I_w}{I}\right)\omega \tag{19}$$

9.3 Programs

9.3.1 Lorenz System

```
# -*- coding: utf-8 -*-
"""
Program for plotting Lorenz equations from a random initial
position
"""

import numpy as np
import random
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy import integrate

#set parameter values here
sigma=10.0
rho=28.0
b=8.0/3.0

def random_float(low, high):
    return random.random()*(high-low) + low

#Lorenz equations in functional form to be accepted by odeint
def dX_dt(X,t=0):
    return [sigma*(X[1]-X[0]),
            rho*X[0]-X[1]-X[0]*X[2],
            X[0]*X[1]-b*X[2],]

#if one wants to iterate over a range of sigma and rho values
iterations=1

for i in range(iterations):

    #sigma=10.0#+0.02*i

    for j in range(iterations):

        #rho=28.0+0.1*j

        X0 = np.array([random_float(-10,10),random_float(-
10,10),random_float(-10,10)])

        t0=0
        tf=100

        N=10000

        t = np.linspace(t0, tf,  N)

        X, infodict = integrate.odeint(dX_dt, X0, t,
full_output=True)
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

X=np.transpose(X)

plt.plot(X[0],X[1],X[2])
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

plt.show()
```

9.3.2 Orbit Separation

```
# -*- coding: utf-8 -*-
"""
Program for calculating Largest Lyapunov Characteristic Exponent (LLCE) using
the method of orbit separation for a range of rho and sigma values.
"""

import numpy as np
import random
from scipy import integrate

#initial co-ordinate displacement
epsilon=1e-9
iterations=10000

#discard as unlikely to be on attractor
discard=2000

rho_iter=2
sigma_iter=2

tot_iter=rho_iter*sigma_iter

iter_count=0

#generates a random float in a given range for initial condition
def random_float(low, high):
    return random.random()*(high-low) + low

#Lorenz system
def dX_dt(X,t=0):
    return [sigma*(X[1]-X[0]),
            rho*X[0]-X[1]-X[0]*X[2],
            X[0]*X[1]-b*X[2],]


#store results
results= np.array([np.zeros(4) for i in range(0,tot_iter)])

for l in range(0,rho_iter):

    rho=28*l

    for j in range(0,sigma_iter):

        sigma=10*j
        b=8.0/3

        for k in range(1):

            t0=0
            tf=1e-3
```

```
    dt=tf-t0

    #initialise two nearby trajectories
    X0 = np.array([random_float(-100,100),random_float(-100,100),random_float(-100,100)])
    Y0 = X0+epsilon

    results_L=[]
    results_t=[]
    results_lyapunov=[]

    for i in range(iterations):

        #time step
        t = np.linspace(t0, tf,  2)

        #integrate both orbits for a time step dt
        X, infodict = integrate.odeint(dX_dt, X0, t, full_output=True)
        Y, infodict = integrate.odeint(dX_dt, Y0, t, full_output=True)

        #displacement vector
        c=Y[-1]-X[-1]

        #magnitude of displacement vector
        d=np.linalg.norm(np.absolute(c))

        L=np.log(np.absolute(d/epsilon))

        results_L.append(L)

        #rescale displacement vector so distance between each
        #co-ordinate is once again epsilon while preserving direction
        Y[-1] = X[-1]+(epsilon/d)*c

        #final position of integration for this time step becomes the
        #initial position for next time step
        X0=X[-1]
        Y0=Y[-1]

        #move forward time
        t0=tf
        tf=tf+dt

#discard first 2000 as these values for L are unlikely to be on the
#attractor
L_filtered=results_L[discard:iterations]

#calculate average LLCE for orbit
lyapunov=(np.sum(L_filtered)/(iterations-discard))/dt

#store results
results[iter_count]=[rho,sigma,b,lyapunov]
iter_count=iter_count+1
```

9.3.3 Gram-Schmidt Re-orthonormalistion

```
# -*- coding: utf-8 -*-
"""
Program for calculating the Lyapunov spectrum. Taken and modified from
LorenzODELCE.py at http://csc.ucdavis.edu/~chaos/courses/nlp/Software/partH.html
This program uses the GramSchmidt process to calculate the entire spectrum for
a range of sigma and rho parameter values.
"""

import numpy as np
import random
import time
from scipy import linalg
from scipy import stats


start_time = time.time()#calculates run time

#function to generate a random float in a chosen range
def random_float(low, high):
    return random.random()*(high-low) + low

#fucntions for the Lorenz equations
def LorenzXDot(sigma,R,b,x,y,z):
        return sigma * (-x + y)

def LorenzYDot(sigma,R,b,x,y,z):
        return R*x - x*z - y

def LorenzZDot(sigma,R,b,x,y,z):
        return -b*z + x*y

#functions for derivatives of Lorenz equations
def LorenzDXDot(sigma,R,b,x,y,z,dx,dy,dz):
        return sigma * (-dx + dy)

def LorenzDYDot(sigma,R,b,x,y,z,dx,dy,dz):
        return (R-z)*dx - dy - x*dz

def LorenzDZDot(sigma,R,b,x,y,z,dx,dy,dz):
        return y*dx + x*dy + -b*dz

#Runge-Kutta 4 method for integrating Lorenz functions
def RKThreeD(a,b,c,x,y,z,f,g,h,dt):
        k1x = dt * f(a,b,c,x,y,z)
        k1y = dt * g(a,b,c,x,y,z)
        k1z = dt * h(a,b,c,x,y,z)
        k2x = dt * f(a,b,c,x + k1x / 2.0,y + k1y / 2.0,z + k1z / 2.0)
        k2y = dt * g(a,b,c,x + k1x / 2.0,y + k1y / 2.0,z + k1z / 2.0)
        k2z = dt * h(a,b,c,x + k1x / 2.0,y + k1y / 2.0,z + k1z / 2.0)
        k3x = dt * f(a,b,c,x + k2x / 2.0,y + k2y / 2.0,z + k2z / 2.0)
        k3y = dt * g(a,b,c,x + k2x / 2.0,y + k2y / 2.0,z + k2z / 2.0)
```

```
        k3z = dt * h(a,b,c,x + k2x / 2.0,y + k2y / 2.0,z + k2z / 2.0)
        k4x = dt * f(a,b,c,x + k3x,y + k3y,z + k3z)
        k4y = dt * g(a,b,c,x + k3x,y + k3y,z + k3z)
        k4z = dt * h(a,b,c,x + k3x,y + k3y,z + k3z)
        x += ( k1x + 2.0 * k2x + 2.0 * k3x + k4x ) / 6.0
        y += ( k1y + 2.0 * k2y + 2.0 * k3y + k4y ) / 6.0
        z += ( k1z + 2.0 * k2z + 2.0 * k3z + k4z ) / 6.0
        return x,y,z

#Runge-Kutta 4 method for integrating Jacobian
def TangentFlowRKThreeD(a,b,c,x,y,z,df,dg,dh,dx,dy,dz,dt):
        k1x = dt * df(a,b,c,x,y,z,dx,dy,dz)
        k1y = dt * dg(a,b,c,x,y,z,dx,dy,dz)
        k1z = dt * dh(a,b,c,x,y,z,dx,dy,dz)
        k2x = dt * df(a,b,c,x,y,z,dx+k1x/2.0,dy+k1y/2.0,dz+k1z/2.0)
        k2y = dt * dg(a,b,c,x,y,z,dx+k1x/2.0,dy+k1y/2.0,dz+k1z/2.0)
        k2z = dt * dh(a,b,c,x,y,z,dx+k1x/2.0,dy+k1y/2.0,dz+k1z/2.0)
        k3x = dt * df(a,b,c,x,y,z,dx+k2x/2.0,dy+k2y/2.0,dz+k2z/2.0)
        k3y = dt * dg(a,b,c,x,y,z,dx+k2x/2.0,dy+k2y/2.0,dz+k2z/2.0)
        k3z = dt * dh(a,b,c,x,y,z,dx+k2x/2.0,dy+k2y/2.0,dz+k2z/2.0)
        k4x = dt * df(a,b,c,x,y,z,dx+k3x,dy+k3y,dz+k3z)
        k4y = dt * dg(a,b,c,x,y,z,dx+k3x,dy+k3y,dz+k3z)
        k4z = dt * dh(a,b,c,x,y,z,dx+k3x,dy+k3y,dz+k3z)
        dx += ( k1x + 2.0 * k2x + 2.0 * k3x + k4x ) / 6.0
        dy += ( k1y + 2.0 * k2y + 2.0 * k3y + k4y ) / 6.0
        dz += ( k1z + 2.0 * k2z + 2.0 * k3z + k4z ) / 6.0
        return dx,dy,dz

#set so solutions describe Malkus water wheel
b=1.0

sigma_iterations=200
rho_iterations=200

#for each Lyapunov spectrum, integrate Lorenz and Jacobian so trajectory is on
#attractor before begin calculating Lyapunov spectrum.
transient_iterations=10
transient_flow_iterations=100

iterations=1000
k_steps_per_iterate=10

#time step
t0=0
tf=0.01

dt=tf-t0

#store sigma,rho,L1,L2,L3
plot_results=[0,0,0,0,0]

for i in range(sigma_iterations):
```

```
sigma=0.1*i

for k in range(rho_iterations):

    rho=2*k

    L1=0.0
    L2=0.0
    L3=0.0

    LCE1=0.0
    LCE2=0.0
    LCE3=0.0

    #basis column vectors to store stretching and shrinking of system
    w=np.identity(3)

    #start integrating from a random initial position
    xState  = random_float(-100,100)
    yState  = random_float(-100,100)
    zState  = random_float(-100,100)

    #store final postion after an integrating cycle so it can be used as
    #the inital position for the next cycle
    x=[]
    y=[]
    z=[]

    #evolve system from random intial position until on attractor
    for i in range(transient_iterations):

        xState,yState,zState = RKThreeD(sigma,rho,b,xState,yState,zState,\
        LorenzXDot,LorenzYDot,LorenzZDot,dt)

        x.append(xState)
        y.append(yState)
        z.append(zState)

    #store Lyapunov spectrum
    L1_results=[]
    L2_results=[]
    L3_results=[]

    #stores total time
    t_total=[]

    #evolve and align co-ordinate vectors under Jacobian until transients
    #is passed
    for i in range(transient_flow_iterations):

        for k in range(k_steps_per_iterate):

            #evolve Lorenz system
```

```
        xState,yState,zState = RKThreeD(sigma,rho,b,xState,yState,zState,\
        LorenzXDot,LorenzYDot,LorenzZDot,dt)

        #store final position for the beginnning of next iteration
        x.append(xState)
        y.append(yState)
        z.append(zState)

        #evolve first column co-ordinate vector
        w[0,0],w[1,0],w[2,0] = TangentFlowRKThreeD(sigma,rho,b,xState,yState,zState,\
        LorenzDXDot,LorenzDYDot,LorenzDZDot,w[0,0],w[1,0],w[2,0],dt)

        #evolve second column co-ordinate vector
        w[0,1],w[1,1],w[2,1] = TangentFlowRKThreeD(sigma,rho,b,xState,yState,zState,\
        LorenzDXDot,LorenzDYDot,LorenzDZDot,w[0,1],w[1,1],w[2,1],dt)

        #evolve third column co-ordinate vector
        w[0,2],w[1,2],w[2,2] = TangentFlowRKThreeD(sigma,rho,b,xState,yState,zState,\
        LorenzDXDot,LorenzDYDot,LorenzDZDot,w[0,2],w[1,2],w[2,2],dt)

        t0=tf

        tf=tf+dt


    w[:,0]=w[:,0]/np.linalg.norm(w[:,0])

    dot_1=np.dot(w[:,1],w[:,0])

    for i in range(len(w)):
        w[i,1]=w[i,1]-dot_1*w[i,0]


    w[:,1]=w[:,1]/np.linalg.norm(w[:,1])

    dot_2=np.dot(w[:,2],w[:,0])
    dot_3=np.dot(w[:,2],w[:,1])

    for i in range(len(w)):
        w[i,2]=w[i,2]-(dot_2*w[i,0]+dot_3*w[i,1])

    w[:,2]=w[:,2]/np.linalg.norm(w[:,2])

#begin estimating Lyapunov spectrum: L1,L2,L3
for i in range(iterations):

    #estimate L1,L2,L3 after (k_steps_per_iterate)*dt
    for k in range(k_steps_per_iterate):

        #evolve Lorenz system
        xState,yState,zState = RKThreeD(sigma,rho,b,xState,yState,zState,\
        LorenzXDot,LorenzYDot,LorenzZDot,dt)
```

```
        x.append(xState)
        y.append(yState)
        z.append(zState)


        w[0,0],w[1,0],w[2,0] = TangentFlowRKThreeD(sigma,rho,b,xState,yState,zState ,\
        LorenzDXDot,LorenzDYDot,LorenzDZDot,w[0,0],w[1,0],w[2,0],dt)

        w[0,1],w[1,1],w[2,1] = TangentFlowRKThreeD(sigma,rho,b,xState,yState,zState ,\
        LorenzDXDot,LorenzDYDot,LorenzDZDot,w[0,1],w[1,1],w[2,1],dt)

        w[0,2],w[1,2],w[2,2] = TangentFlowRKThreeD(sigma,rho,b,xState,yState,zState ,\
        LorenzDXDot,LorenzDYDot,LorenzDZDot,w[0,2],w[1,2],w[2,2],dt)

        t0=tf

        tf=tf+dt

    #GramSchmidt process for orthonormalising vectors
    d1=np.log(np.linalg.norm(w[:,0]))

    w[:,0]=w[:,0]/np.linalg.norm(w[:,0])

    dot_1=np.dot(w[:,1],w[:,0])

    #remove components of column 1 from column 2
    for i in range(len(w)):
        w[i,1]=w[i,1]-dot_1*w[i,0]

    d2=np.log(np.linalg.norm(w[:,1]))

    w[:,1]=w[:,1]/np.linalg.norm(w[:,1])

    dot_2=np.dot(w[:,2],w[:,0])
    dot_3=np.dot(w[:,2],w[:,1])

    #remove components of column 1 and column 2 from column 3
    for i in range(len(w)):
        w[i,2]=w[i,2]-(dot_2*w[i,0]+dot_3*w[i,1])

    d3=np.log(np.linalg.norm(w[:,2]))

    w[:,2]=w[:,2]/np.linalg.norm(w[:,2])

    #estimate Lyapunov spectrum for (k_steps_per_iterate*dt) steps
    L1=d1/(k_steps_per_iterate*dt)
    L2=d2/(k_steps_per_iterate*dt)
    L3=d3/(k_steps_per_iterate*dt)

    L1_results.append(L1)
    L2_results.append(L2)
    L3_results.append(L3)
```

```
            t_total.append(tf)


        #average for final values
        LCE1=np.average(L1_results)
        LCE2=np.average(L2_results)
        LCE3=np.average(L3_results)

        plot_results=np.vstack((plot_results,[sigma,rho,LCE1,LCE2,LCE3]))

        print sigma,rho,LCE1,LCE2,LCE3

#save results
np.save('parameter_space_map.npy', plot_results)

#print run-time
print("--- %s seconds ---" % (time.time() - start_time)
```

9.3.4 LLCE Parameter Map Generator

```
# -*- coding: utf-8 -*-
"""
Program for generating parameter map from data
"""

import numpy as np
import matplotlib.pyplot as plt

#loads in data
plot_results=np.load('parameter_space_map_saved.npy')

#puts all sigma in one column, rho in another etc so matplotlib can plot
plot_results=np.transpose(plot_results)

sigma=plot_results[0]
rho=plot_results[1]
LLCE=plot_results[2]

fig, ax = plt.subplots(tight_layout=True)
#'heat map' generated using hist2d, 200 bins was found to be the best number
hist = ax.hist2d(rho, sigma,weights=LLCE,bins=200)
plt.title(r'Parameter Space Map of $\sigma$ vs $\rho$' )
ax.set_xlabel(r'$\rho$',fontsize=15)
ax.set_ylabel(r'$\sigma$',fontsize=15)
cbar=plt.colorbar(hist[3], ax=ax)
cbar.set_label('Largest Lyapunov Exponent', rotation=90)
plt.show()
```

9.3.5 Plot Data

```
# -*- coding: utf-8 -*-
"""
Program for plotting data collected in .txt from MSB USB
"""

import numpy as np
import matplotlib.pyplot as plt

#replace with location of data to be plotted
#loads values into list. ignores first row corresponding to names of columns
DataIn = np.loadtxt(r'C:\Physics_Project\devcpp_testio_sean\run1_period.txt',skiprows=1)

time=np.zeros(len(DataIn))
angle=np.zeros(len(DataIn))
omega=np.zeros(len(DataIn))

#each column of data is stored in its own array
for i in range(len(DataIn)):

        time[i]=DataIn[i][1]
        angle[i]=DataIn[i][2]
        omega[i]=DataIn[i][3]

#plots omega vs t
plt.figure()
plt.xlabel(r'$t\ (s)$')
plt.ylabel(r'$\omega\ (\frac{rads}{s})$')
plt.plot(time,omega,c="#1f77b4" )
```

9.3.6 Angular Velocity Simulation

```
# -*- coding: utf-8 -*-
"""
Program to simulate angular velocity motion of water wheel
"""

import numpy as np
import random
import matplotlib.pyplot as plt
from scipy import integrate

#periodic motion - values selected by using parameter map

sigma=1.75
rho=120
b=1.0


def random_float(low, high):
    return random.random()*(high-low) + low

def dX_dt(X,t=0):
    return [sigma*(X[1]-X[0]),
            rho*X[0]-X[1]-X[0]*X[2],
            X[0]*X[1]-b*X[2],]


X0 = np.array([random_float(-10,10),random_float(-10,10),random_float(-10,10)])

t0=0
tf=32.5

dt=1e-3

N=10000

t = np.linspace(t0, tf,  N)

X, infodict = integrate.odeint(dX_dt, X0, t, full_output=True)

X=np.transpose(X)

#omega=x*k
k=0.13

X[0]=X[0]*k

#remove transients
X=X[0,950:]

#time=dimsionless time/k
t=t/k
```

```
t=t[950:]

#turn into column for plotting
t=np.transpose(t)

plt.figure()
plt.xlabel(r'$t\ (s)$')
plt.ylabel(r'$\omega\ (\frac{rads}{s})$')
plt.plot(t,X,c="#1f77b4")

plt.show()
```

9.3.7 4-Bucket Simulation and Animation

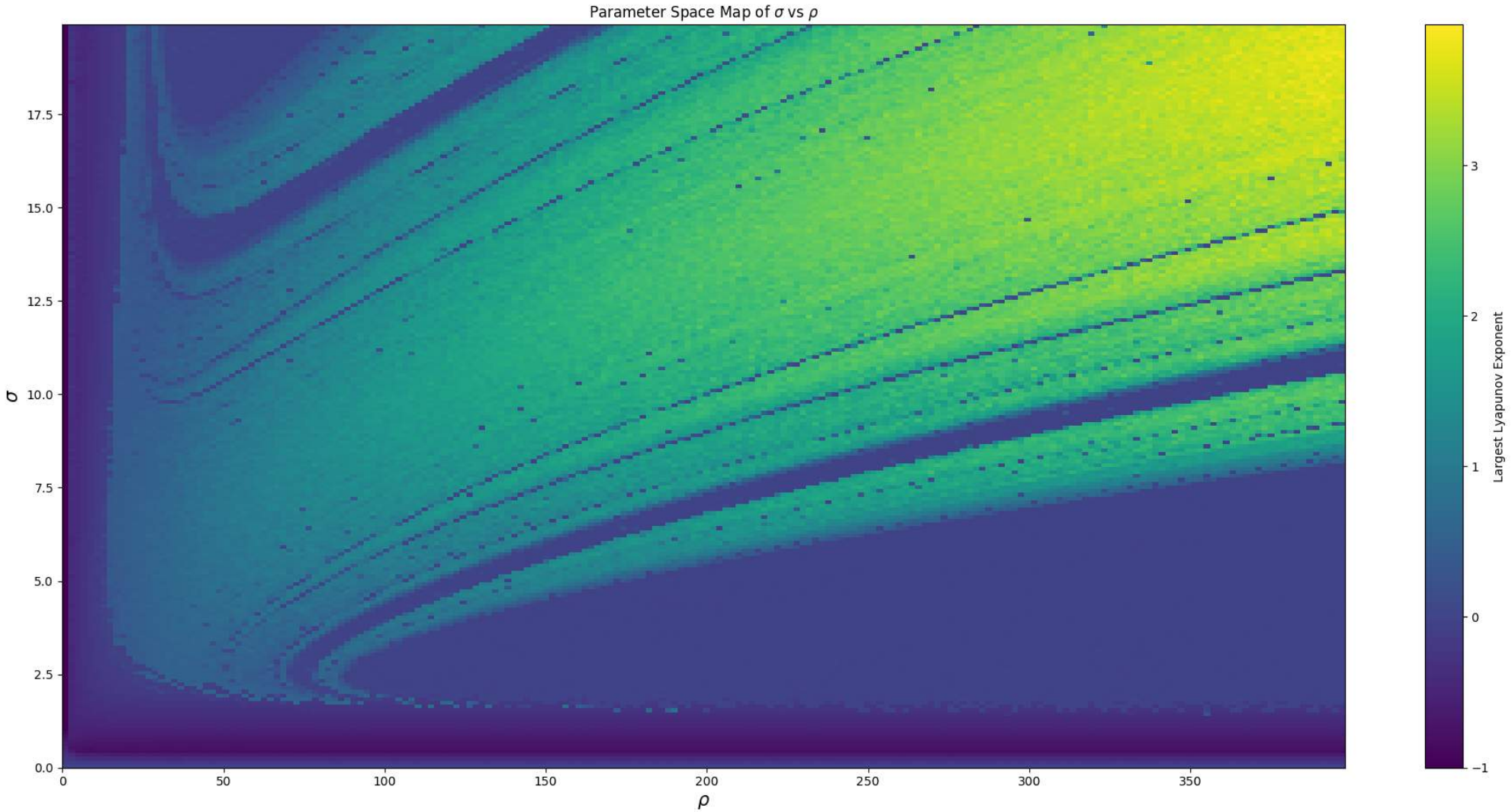
```
# -*- coding: utf-8 -*-
"""
Program to simulate an N bucket water wheel. Note this program breaks down at
high N when buckets start to overlap.
"""
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation

#here you can adjust all of the water wheels parameters
N=4 #number of buckets
ntrial=20000 #time=ntrails*delt

IO=0.1#inertial of wheel

R=0.2
phi=(15.0/360.0)*2.0*np.pi #tilt angle (slightly tilted)
g=9.81

QO=0.039#inflow
L=0.13#leakage
alpha=0.01#damping coefficient

omega=0.0
theta0=0.01

init_pt=5*(np.pi/180)#position of inflow tube

t=0.0
delt=0.01

tn=np.zeros(ntrial)

thetan= np.array([np.zeros(ntrial) for i in range(0,N)])
Mn = np.array([np.zeros(ntrial) for i in range(0,N)])
Qn = np.array([np.zeros(ntrial) for i in range(0,N)])
y_co_ord = np.array([np.zeros(ntrial) for i in range(0,N)])
z_co_ord = np.array([np.zeros(ntrial) for i in range(0,N)])

results_omega=[]
results_rot=[]
results_ycm=[]
results_zcm=[]

for i in range(0,ntrial):


    Tg=0.0
    Iw=0.0
    ycm=0.0
    zcm=0.0
```

```
mtot=0.0

tn[-1]=0.0

tn[i]=tn[i-1]+delt

for k in range(0,N):

    #determine new position of buckets
    thetan[k][i]=(((k)*1.0/N)*2.0*np.pi+theta0);

    #based on position calculate inflow for each bucket. Inflow function
    #assumes only one bucket can be under the spout at a time
    Qn[k][i]=Q0*(np.cos(thetan[k][i]/2.0-init_pt)**1000.0)

    #determone mass in bucket
    Mn[k][i] = delt*(Qn[k][i]-L*Mn[k][i-1])+Mn[k][i-1]

    mtot=mtot+Mn[k][i]

    y_co_ord[k][i]=(np.sin(thetan[k][i])*R)
    z_co_ord[k][i]=(np.cos(thetan[k][i])*R)

    Iw=Iw+Mn[k][i]*R*R

    #calculate total gravitational torque
    Tg=Tg+ Mn[k][i]*g*np.sin(phi)*R*np.sin(thetan[k][i])

for k in range(0,N):

    ycm=ycm+Mn[k][i]*y_co_ord[k][i]
    zcm=zcm+Mn[k][i]*z_co_ord[k][i]


ycm=ycm/mtot
zcm=zcm/mtot

Itot=I0+Iw

#calculate braking torque
Tvis=-omega*alpha

#calculate torque induced in spinning up water to speed of bucket
Tspinup=-omega*L*Iw

Ttot=Tg+Tvis+Tspinup

#calculate angular velocity
omega=delt*(Ttot/Itot) +omega

#determine new position of each bucket
theta0= theta0 +delt*omega
```

```
    rot=theta0/(2.0*np.pi)

    results_omega.append(omega)
    results_rot.append(rot)
    results_ycm.append(ycm)
    results_zcm.append(zcm)


for i in range(0,N):

    plt.plot(tn, Mn[i],label="M%.2g" % i)

plt.plot(tn, results_omega,label=r"$\omega$")
#plt.plot(tn, results_rot,label="rot")
plt.plot(tn, results_ycm,label=r"$y_{cm}$")
plt.plot(tn, results_zcm,label=r"$z_{cm}$")
plt.legend()

plt.figure()
plt.xlabel(r"$y_{cm}$")
plt.ylabel(r"$z_{cm}$")
plt.plot(results_ycm,results_zcm)


xn=np.array([np.zeros(ntrial) for i in range(0,N)])
yn=np.array([np.zeros(ntrial) for i in range(0,N)])

for i in range(0,ntrial):

    for k in range(0,N):
        xn[k][i]=R*np.sin(thetan[k][i])
        yn[k][i]=R*np.cos(thetan[k][i])


fig = plt.figure()
ax = plt.axes(xlim=(-1, 1), ylim=(-1, 1))

time_text = ax.text(0.02, 0.95, '', transform=ax.transAxes)
Mn_text=[]
omega = ax.text(0.02, 0.90, '', transform=ax.transAxes)

circles_water=[]

for i in range (0,N):

    Mn_text.append(ax.text(0.02, 0.85-5*i, '', transform=ax.transAxes))

    circles_water.append(plt.Circle((5, -5), 0.75, fc='r'))
    ax.add_patch(circles_water[i])


circle_cm = plt.Circle((5, -5), 0.05, fc='g')
circle_ycm = plt.Circle((5, -5), 0.05, fc='y')
```

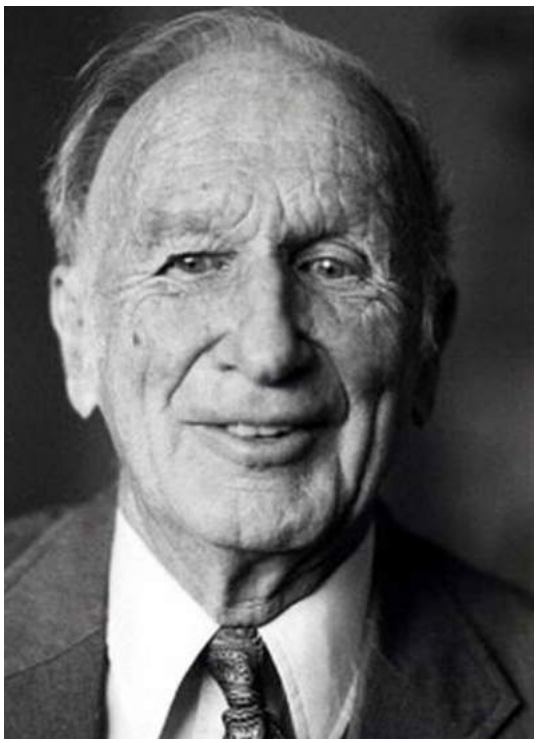
```
circle_zcm = plt.Circle((5, -5), 0.05, fc='b')
ax.add_patch(circle_cm)
ax.add_patch(circle_ycm)
ax.add_patch(circle_zcm)


lines = [plt.plot([], [],c='b')[0] for _ in range(N)]


def init():

    for line in lines:

        line.set_data([], [])

    for i in range (0,N):

        circles_water[i].center = (5, 5)
        ax.add_patch(circles_water[i])

    circle_cm.center = (5, 5)
    circle_ycm.center = (5, 5)
    circle_zcm.center = (5, 5)
    ax.add_patch(circle_cm)
    ax.add_patch(circle_ycm)
    ax.add_patch(circle_zcm)

    return tuple(lines) + tuple(circles_water) + (circle_cm,) + (circle_ycm,) + (circle_zcm,)


def animate(i):

    t_total=i*0.01
    time_text.set_text('Time = %.1f' % t_total)

    omega.set_text(r'$\omega$ = %.1f' % results_omega[i])

    for j,line in enumerate(lines):
        line.set_data([0,R*np.sin(thetan[j][i])], [0,R*np.cos(thetan[j][i])])

    for k in range(0,N):

        x = R*np.sin(thetan[k][i])
        y = R*np.cos(thetan[k][i])


        circles_water[k].center = (x, y)
        circles_water[k].radius = Mn[k][i]/10


    circle_cm.center = (results_ycm[i],results_zcm[i])
    circle_ycm.center = (results_ycm[i],0)
    circle_zcm.center = (0,results_zcm[i])
```

```
    return tuple(lines) + tuple(circles_water) + (time_text,) + (omega,) + \
                (circle_cm,) + (circle_ycm,) + (circle_zcm,)


anim = animation.FuncAnimation(fig, animate, init_func=init,
                               frames=ntrial, interval=10, blit=True)

#save animation.Note animation won't run while being saved and requires ffmpeg
#to be installed

#plt.rcParams['animation.ffmpeg_path'] ='C:\\ffmpeg\\bin\\ffmpeg.exe'
#FFwriter = animation.FFMpegWriter(fps=100)
#anim.save('basic_animation.mp4', writer = FFwriter,dpi=100)
#anim.save('2osc.mp4', writer="ffmpeg")

plt.show()
```


Parameter Space Map of σ vs ρ
σ
ρ
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
50
100
150
200
250
300
350
Largest Lyapunov Exponent
−1
0
1
2
3

10 Biographies

Edward Lorenz (1917-2008)



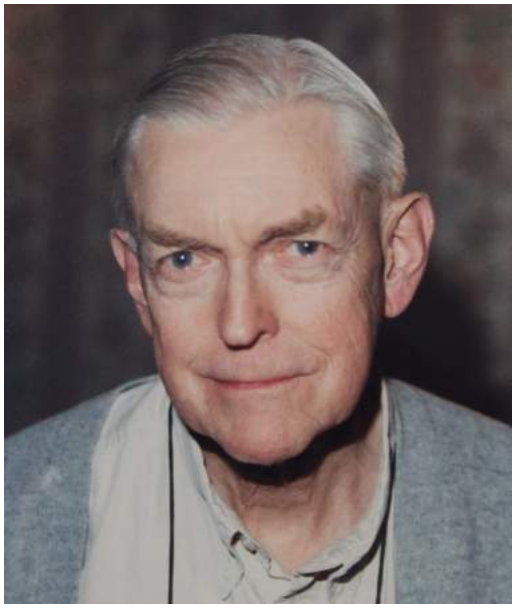
Edward Norten Lorenz was born May 23 1917 in West Hartford, Connecticut. He studied first mathematics at Dartmouth and Harvard and then meteorology at MIT. In the early 1960s his studies into the non-linear field of atmospheric science and weather forecasting lead him to publish his now landmark paper *"Deterministic Non-Periodic Flow"* in 1963. The paper introduces the system of equations now known as the Lorenz system and with it the foundations of modern chaos theory as we know it today. His discoveries however lay dormant and unrecognised until the 1970s. In 1972 he gave a talk at the American Association for the Advancement of Science titled *Predictability: Does the Flap of a Butterflys Wings in Brazil Set Off a Tornado in Texas?*. The title embodied a term coined by Lorenz that has now entered the public conscience - *The Butterfly Effect*. For his contributions to the field of earth science as well as the founding of chaos theory, Lorenz in 1991 was awarded Japan's highest private award for global achievement and contribution to humanity - The Kyoto Prize. Always active, both outdoors and in his research, Lorenz continued working until his death due to cancer on the 16th April 2008, aged 90.

Willem Malkus (1924-2016)

Willem Van Rensselaer Malkus was a professor of applied mathematics at MIT from 1969 until his retirement in 1996. His main areas of research were thermal convection, magnetohydrodynamics, and geophysical fluid dynamics. As a graduate student we worked in the area of particle physics with the great Enrico Fermi before choosing to move to fluid mechanics and in particular its application to geophysics. His contributions range from thermal convection and turbulence to magnetohydrodynamics and elliptical flows. In the 1960's he, along with a fellow applied mathematician Louis Howard and Edward Lorenz invented a simple mechanical water wheel known as Malkus-Howard-Lorenz Waterwheel that mechanically embodied the Lorenz equations. Malkus joked that his toy model seemed to be more correctly predicted by the Lorenz equations then the phenomenon Lorenz wrote down his equations to predict - atmospheric convection! When he arrived at MIT, Malkus founded the Applied Math Laboratory, where he carried out a variety of fluid mechanics experiments. He was also a founding member of the Geophysical Fluid Dynamics (GFD) Program at the Woods Hole Oceanographic Institution in 1959 along side Louis Howard. He was awarded the Excellence in Geophysical Education Award by the American Geophysical Union in 2008 for his contributions to the founding and advancement of the field.

Louis Howard (1929-2015)

Louis Norberg Howard was emeritus professor of mathematics at MIT. He joined MIT in 1955, was promoted to full professor in 1964 and retired in 1984. Howard's main area of interest was fluid dynamics. His work included papers on hydrodynamic stability, geophysical flows, turbulent convection, flows in Hele-Shaw cells, salt-finger zones, rotating flows, and reaction-diffusion equations. Howard's work extended into pure mathematics as well as evidenced by his existence proofs concerning the hydrodynamic equations, and his elegant Semicircle Theorem. In the 1960's himself, along with Willem Malkus and Edward Lorenz built a Malkus-Howard-Lorenz Water Wheel to physically model the Lorenz equations in a practical way. Like Malkus, he was a founding member of the Geophysical Fluid Dynamics (GFD) Program at the Woods Hole Oceanographic Institution and received in 2008 the Excellence in Geophysical Education Award by the American Geophysical Union.



11 Acknowledgements

This project would not have been possible without the help, support and expertise of a number of key individuals, offices and departments. It is here that I wish to convey my deep sense of gratitude.

I would like to extend my heartfelt thanks to the Experiential Learning office and the SPUR program of Maynooth University for giving me this opportunity to participate in this undergraduate research project. The experience was simply wonderful and the entire Experimental Physics department was so welcoming and inclusive during my stay (I only wish I could have stayed longer!).

I would like to thank in particular Dr Aisling Flynn, Experiential Learning Officer for running and co-coordinating the SPUR program.

In terms of the project itself, I would like to acknowledge and thank technician David Watson, for not only building both versions of the water wheel but in helping to fix many other unforeseen problems that arose during the project as well.

As well as David, I would like to thank the entire technical staff for there assistance and help when required: Marie Galligan, Derek Gleeson, John Kelly, Ian McAuley and Patrick Seery.

I would also like to thank fellow SPUR participator Eoghan Ross for graciously assisting me when a third hand was needed.

Finally and most importantly, I would like to thank my mentor Dr Michael Cawley for choosing me for this project and supporting me throughout the project with insight and guidance.