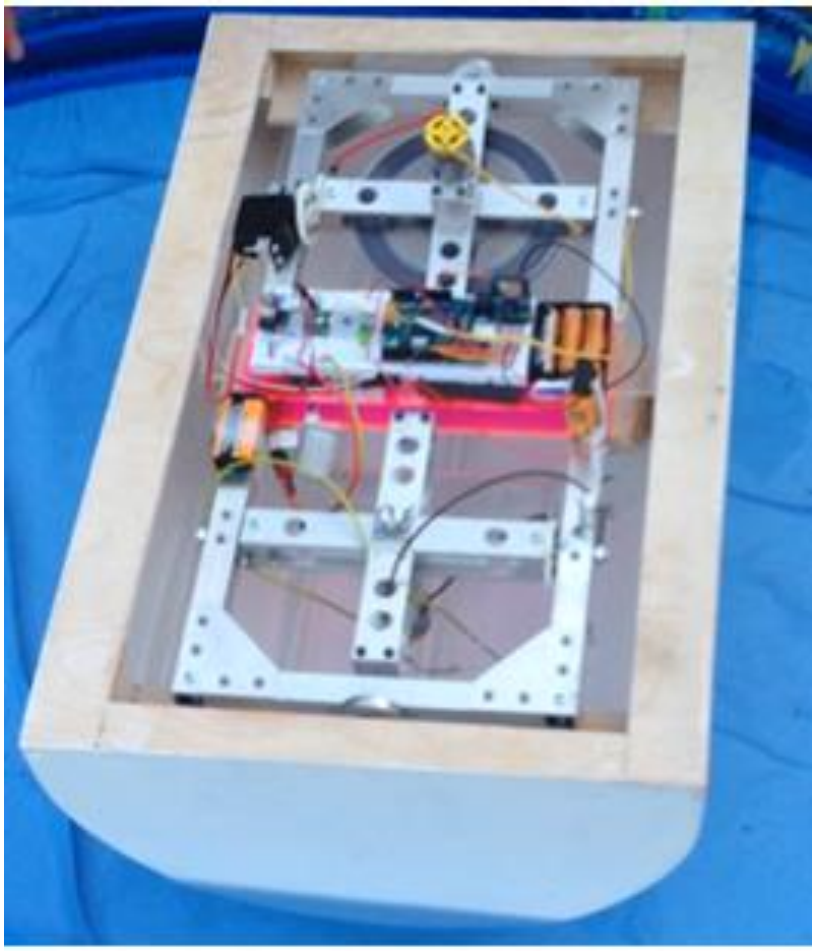
Development of a Gyroscopic Stabilisation System for use on RNLI Rescue Craft

James Kitching

Department of Physics
The University of York
Heslington
York
YO10 5DD

Mphys Project Dissertation
Supervisor: Dr Phil Lightfoot

May 2017



Abstract

A gyroscopic system was developed for use in Royal Naval Lifeboat Institution (RNLI) vessels in order to improve vessel stability in rough seas, thereby supporting safer operation in terms of dealing with casualties in the water or pulling alongside ships in distress.

Gyroscopic stabilisation systems were first developed in the early 20th century. These were passively controlled, relying on the natural precession of a freely rotating flywheel to restore the boat to equilibrium. This project focussed on the development of a faster active system which used a microcontroller equipped with an accelerometer and PID control software to direct a servo motor which rotated a rotating flywheel about its gimbal axis to produce a torque which counteracted the initial roll.

A safety cage was used to test the electronic, mechanical and control systems as they were being developed and the performance characteristics of the PID control system were measured. In parallel 2.5 kg wood and fibreglass boat was designed and constructed. The careful choice of the centre of mass and the centre of buoyancy locations allowed this boat to be inherently unstable, allowing the stabilisation properties of the gyroscopic system to be fully tested.

The stabilisation system was transferred into the boat and its performance was tested in terms of its stability about the roll axis. It was found that a flywheel of moment of inertia (2.03 ± 0.01) x10^{-3} kg m^2, rotating at 2700 RPM, controlled by the PID system with parameters k_p = 0.3, k_i = 0.0, and k_d = 0.4 yielded a stable vessel which could respond to damp an initial 20^o roll angle within 6 seconds.

These results represent superior performance to the passive systems previously used, and could indicate a future application as a vessel stabilisation system for lifeboats.

Contents Page No

1. Introduction

Gyroscopic stabilisation systems were first used on marine vessels at the turn of the last century. Stabilised vessels can offer more comfort and a safer environment for both crew and passengers. A stabilised lifeboat would allow for safer and more controlled rescues as it would be able to keep vertical in the water even in bad weather conditions.

A number of stabilisation systems are currently available, these include;
[1] Bilge keels, these are flat plates that run along the bottom of the boat from front to back. These work by increasing resistance when the boat rolls from side to side. They are very popular due to their low cost but their effectiveness is limited (about 25% roll reduction). Passive tanks, these are a pair of tanks that are filled with water which are tuned to allow the water to pass between them half a cycle out of phase with the ship. This means when the ship is tilting one way the weight of the water will be acting in the opposite direction. These can reduce the roll of a ship by up to 75% but they take up a lot of room and also add weight.

[2] The best way to reduce roll is active roll stabiliser fins which can reduce roll by up to 90%. This system is comprised of fins mounted on rotatable stocks in the middle of the boat. Their angle is continually adjusted to produce a torque which opposes the torque produced by the waves. These fins use the water going past them to produce this torque which means that if the boat is not moving then these fins are ineffective. Another drawback is the additional resistance the fins cause.

A gyroscope usually consists of a spinning flywheel which can be used to apply a torque. In the classic example of a gyroscope this torque can be used to keep the flywheel pointing in a set direction independent of the movement of its mounting. In a gyroscopic stabilisation system the torque produce by the spinning flywheel would be used to counteract any external torques.

A gyroscopic system would offer several advantages over the current available systems. A gyroscopic system would not depend on the movement of the boat to produce a torque so would work equally well for a static boat as it does for a moving one. A gyroscopic system would be entirely internally mounted which means no additional resistance would be caused. The gyroscopic system does however take up room inside the boat but unused areas may be able to be found to reduce the useable room needed.

In this project a gyroscopic stabilisation system was designed and tested to balance an unstable boat in water. The moment of inertia of the boat, flywheel and flywheel frame were be measured. The frame was balanced using a passively controlled system and an actively controlled one using two different flywheels of differing moment of inertia. Finally, the frame and flywheel were mounted into an unstable boat and the ability of the active system to stabilise the boat after a simulated wave was tested.

2. Background

The first experiments done with gyroscopes were conducted by [3] M. Foucault in the 1850s. These were mainly to do with navigation and one of the first devices made was to point towards a pole to give a set point to use for navigation purposes.

As early as the 1880s the torque produced by a rotating disc was known. In his patent [4] C. Benz makes reference to the flywheel of a vehicle or boat being in a horizontal orientation to improve the handling. He goes on to say that this horizontal rotation would not only give better control to the vehicle but also a greater safety against capsizing in a boat.

In 1903 [5] L. Brennan devised a plan for balancing a monorail train using a gyroscope. This was later turned into a working model. The control of the balancing gyroscope on this used a friction plate to slow down one of the two spinning gyroscopes in order to apply a torque to the car. When the car was stabilised the friction surface was removed so no more torque was applied. Although this system was mechanically controlled, it functioned well and in 1910 a full scale car was made and run around an oval track. Due to safety concerns and lack of funding the project was abandoned.

In more recent times there has been a resurgence of interest in gyroscopic stabilisation for automobiles with companies like [6] LIT Motors producing a gyro-stabilised 2-wheeled electric vehicle. And the same concept being applied to an underwater vehicle using three internal rotors [7].

Gyroscopic stabilisation has been successfully used for ships and marine vessels. In 1908 E. Sperry applied for a patent [8] for an actively controlled gyroscopic stabiliser but the control system for this was very complex. In 1915 he applied for another patent for a more effective control system which made use of a smaller gyroscope, which was more responsive to outside factors, to control a larger gyroscope [9].

In 1913 Sperry produced the first gyroscopic stabilisation system for ships and fitted it on the USS Worden. This device worked as expected but the rollout on further vessels came to a stop due to the outbreak of World War One.[10] After the war similar active devices were fitted to yachts and then ultimately the largest ever gyroscopically controlled ship was constructed, the S.S. Conte di Savoia. This was a 41 000 ton displacement Italian liner. The system worked well under normal conditions but struggled to counteract the rolling in Atlantic storms. Sperry's gyroscopic systems were eventually superseded by the Denny-Brown stabiliser which was an activated fin based system.

Since then there has been little interest in gyroscopic stabilisation systems for boats until the 2000s when several companies started producing gyro-stabilisation systems for smaller vessels. Examples include Gearloose Engineering, Inc [11] which is a passive system where the spinning flywheel is kept in a low pressure environment to reduce power consumption. And, [12] Marine Motion Control, llc which has proposed an active system which uses a composite flywheel to minimise weight.

3. Theory

[13]If Newton's third law of motion for a rotational system in the non-isolated case is taken a general particle of mass m_i and position vector $\boldsymbol{r}_i$ will have the equation of motion;

$$m_i \frac{d^2 \boldsymbol{r}_i}{dt^2} = \sum_{j=1,N}^{j \neq i} f_{ij} + \boldsymbol{F}_i \tag{1}$$

where f_{ij} is the internal force exerted by an object j on object i and $\boldsymbol{F}_i$ is the net external force acting on object i.

If the above equation is summed over all objects it becomes;

$$\frac{d\boldsymbol{P}}{dt} = \boldsymbol{F} \tag{2}$$

where $\boldsymbol{P}$ is the total momentum and $\boldsymbol{F}$ is the net external force acting on the system.

In equation 2 the internal forces have cancelled out in pairs, this means that when a net external force is acting on the system it evolves in time according to this equation. Because equation 2 is of a similar form to Newton's second law it can be concluded that the centre of mass of many point objects has dynamics that are analogous to that of a point object.

If equation 1 is crossed with $\boldsymbol{r}_i$ and then summed over all objects it becomes;

$$\frac{d\boldsymbol{L}}{dt} = \boldsymbol{\tau} \tag{3}$$

where $\boldsymbol{L}$ is the total angular momentum and $\boldsymbol{\tau}$ is the net external torque acting on the system.

For the flywheel, the magnitude of the angular momentum,L_0, can be calculated using the equation;

$$L_0 = I_{Fly}\omega_{Fly} \tag{4}$$

where I_{Fly} is the moment of inertia of the flywheel about its rotation axis and ω_{Fly} is the magnitude of the angular velocity of the flywheel.

Assuming the moment of inertia and angular velocity of the flywheel remain constant (which is a good approximation) then the angular momentum of the flywheel, $\boldsymbol{L_{Fly}}$, will be;

$$\boldsymbol{L_{Fly}} = \begin{pmatrix} L_0 \cos\theta(t) \\ L_0 \sin\theta(t) \\ 0 \end{pmatrix} \tag{5}$$

where $\theta(t)$ is the time dependent angle between the frame and the angular momentum vector (i.e. the motor axis).

Differentiating this with respect to time will give the torque produced by the flywheel, $\boldsymbol{\tau_{Fly}}$;

$$\boldsymbol{\tau_{Fly}} = \frac{d\boldsymbol{L_{Fly}}}{dt} = \begin{pmatrix} -L_0\dot{\theta}\sin\theta(t) \\ L_0\dot{\theta}\cos\theta(t) \\ 0 \end{pmatrix} \tag{6}$$

where $\dot{\theta}$ is the rate of change of the angle $\theta(t)$ with respect to time.

The torque produced by the flywheel is the one that will balance the boat. In the active system the $\dot{\theta}$ will come from the forced rotation of they flywheel about the y-direction using the servo. In the passive system $\dot{\theta}$ will come from the flywheel being rotated about the y-direction by the torque of the frame. A labelled photograph of the experimental setup is shown below.

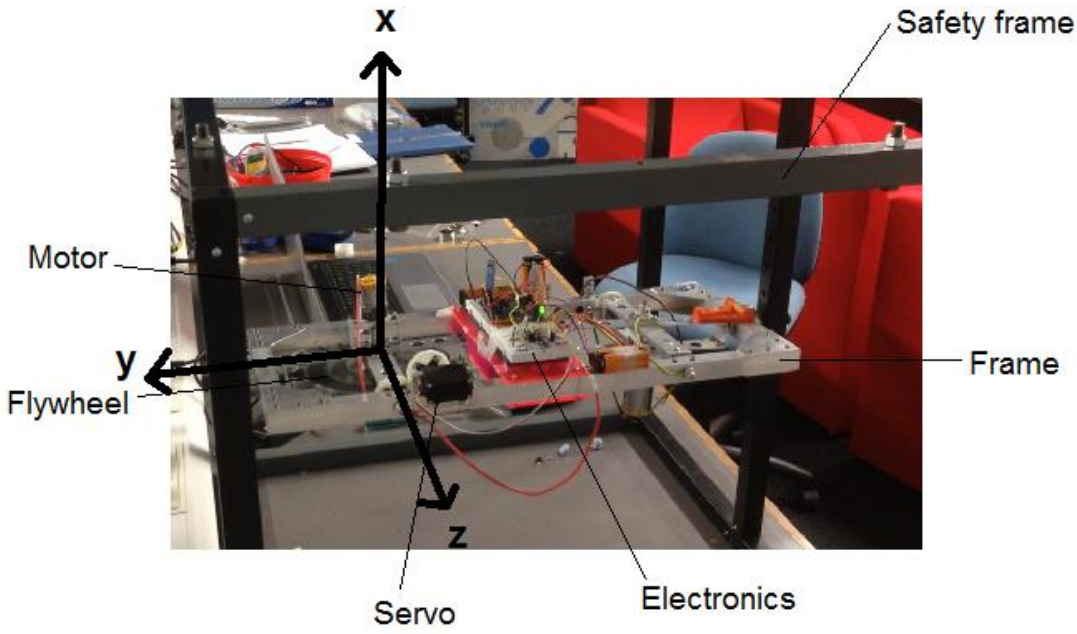


Figure 1: A labelled photograph showing the experimental setup for the laboratory testing of the frame. The defined directional axes are also labelled.

3.1 Measuring the moment of inertia

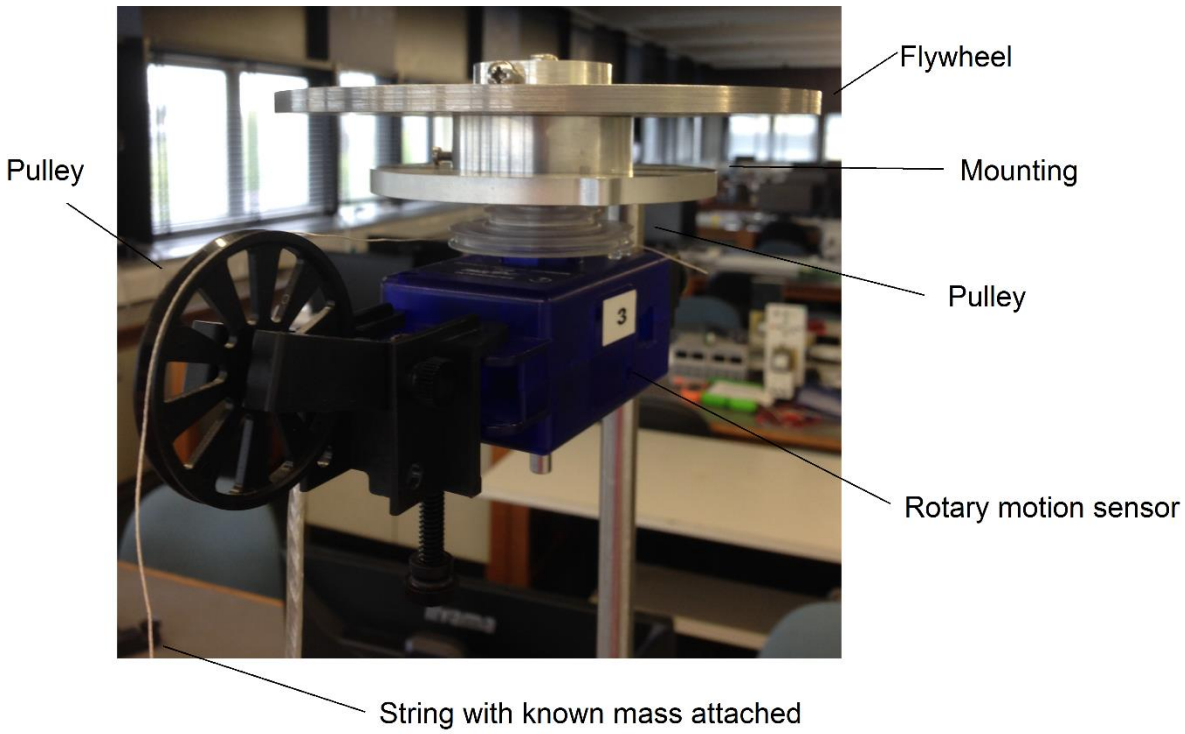


Figure 2: A labelled photograph showing the setup used to measure the moment of inertia of the LMOI flywheel and the HMOI flywheel. The flywheel was mounted on top of a rotary motion sensor using a mounting attachment. String was wrapped around a mounting pulley and then over a directional pulley. A know mass was then attached to the end of this string.

To measure the moment of inertia of the flywheels a rotary motion sensor was used. The flywheel was mounted on top of a pulley which was mounted on top of the sensor. A piece of string was then wound around the pulley and a known mass attached to its end. The mass was then allowed to drop under gravity and the angular velocity was recorded and plotted against time. The gradient of the angular velocity against time graph is the angular acceleration. [14]This can then be used in the equation;

$$\frac{\tau}{\alpha} = I \tag{7}$$

where α is the angular acceleration, τ is the torque applied and I is the scalar moment of inertia of the disc about the axis it is rotated.

To get the torque applied the following equation can be used;

$$\tau = rmg \tag{8}$$

where r is the radius of the pulley, m is the mass of the known mass and g is the acceleration due to gravity.

The moment of inertia of the flywheel and the mounting system was found. Then using the same technique the moment of inertia of the mounting was found. Because moments of inertia are additive, the moment of inertia of the mounting could be subtracted from the moment of inertia of the mounting and flywheel to give the moment of inertia of the flywheel.

To measure the moment of inertia of the frame and boat (including flywheel housing, electronics, batteries etc.) the same principle was applied but it was conducted slightly differently. The batteries and electronics were replaced with masses so they would not get damaged. String was wrapped around a pulley attached to the system and a mass attached to the end of it. Another piece of string was then wrapped around the rotary motion sensor, the end of this was attached to another pulley which was on the other end of the system such that when the system spun the string would be unwound from the rotary motion sensor and onto the pulley.

When the mass was allowed to drop the system started spinning. The string from the rotary motion sensor then unwound onto the pulley and the linear velocity of this was recorded. This linear velocity was plotted against time and the gradient of this was the tangential acceleration. The angular acceleration α could then be found from the tangential acceleration using;

$$\alpha = \frac{a_T}{r} \tag{9}$$

where α is the angular acceleration, a_T is the tangential acceleration and r is the radius of the pulley.

The moment of inertia could then be worked out using equation 7 as torque and angular acceleration were known.

3.2 Flywheel Design

The amount of torque the flywheels need to provide was estimated by considering that the torque needed to right the boat would be no more than the torque provided by a 500g mass one metre away from the rotation axis of the boat. The moment of inertia of the flywheels, I, could then be worked out by estimating the time taken for the servos to move, Δt, and the angular velocity the motors would be able to spin them up to, ω, using the equation;

$$I = \frac{|\vec{\tau}|\Delta t}{\omega} \tag{10}$$

The angular velocity of the flywheels, ω, was estimated to be 5000 rpm. This was based on the limits of the motors from the datasheet. The time taken for the servos was estimated to be 0.5 s. Again this was estimated using information from the datasheet. And the torque $|\vec{\tau}|$ was estimated to be 5 Nm using the assumption above. This gives the total moment of inertia of the two discs to be 4.77 x10^{-3} Kgm^2.

The flywheels were designed to have the lowest mass possible whilst still having the required moment of inertia. This is because the lower the mass the faster the motor will be able to spin the flywheel. The moment of inertia of an object is given by the integral;

$$I = \int_0^m r^2 \, dm \tag{11}$$

where r is the distance from the rotation axis and dm is an infinitesimally small element of mass.

As the moment of inertia is proportional to r^2 to get the maximum moment of inertia for the least mass then most of the mass needs to be put far away from the rotation axis. This means on a disc the edges will need to be thicker than the central part. The radius of the flywheels is constrained by the size of the housing but radius of the flywheel will be as large as possible due to the r^2 dependence. See figure 3 for a detailed schematic of the flywheels.

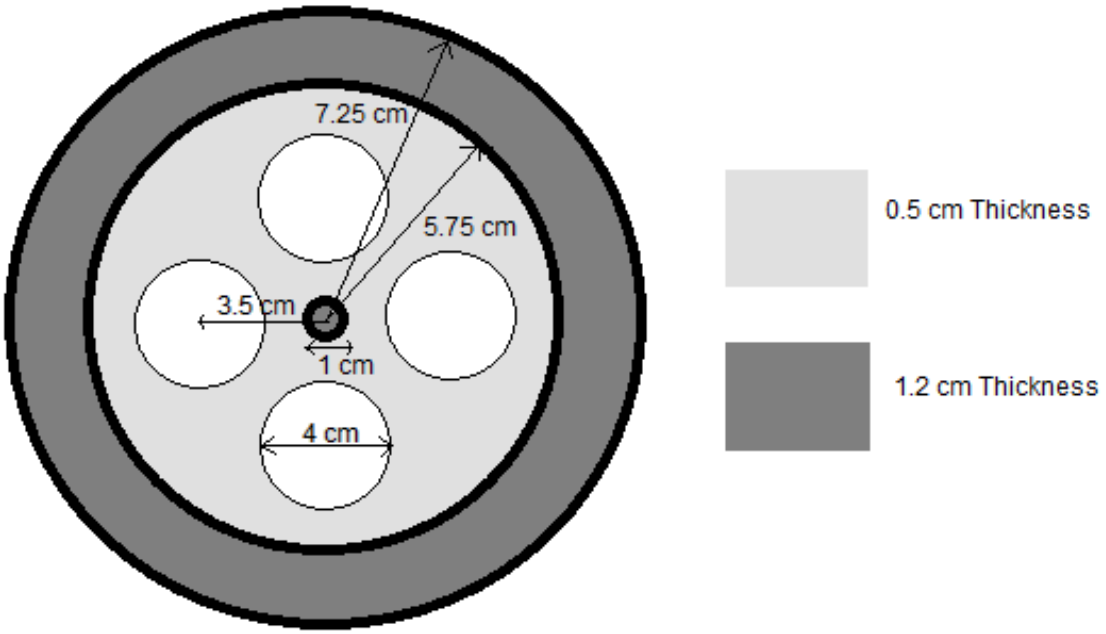


Figure 3: A schematic of the flywheel design. The flywheel was made from steel and had a diameter of 14.5 cm. The four holes reduce the mass of the flywheel but because they are close to the centre they only have a small effect on the moment of inertia. The increased thickness on the outside increases the moment if inertia. The theoretical moment of inertia of this disc was calculated to be 2.45 $x10^{-3}$ kg m^2.

Due to engineering difficulties several adjustments had to be made to the disc designed above. It was impractical to have the disc two different thicknesses because if this was done getting the disc to be balanced would prove a huge challenge and one which could not be solved in the timeframe. Because of this the size of the holes drilled around the centre had to be reduced and their number increased. This was to maintain enough material on the disc to have the desired moment of inertia.

During the early testing stages a smaller and safer low moment of inertia (LMOI) flywheel was used. This was a solid aluminium disc of diameter 76.5 ± 0.1 mm. After the tests had been completed using the LMOI flywheel the high moment of inertia (HMOI) flywheel was fitted.

3.3 Proportional Integral Derivative (PID) control

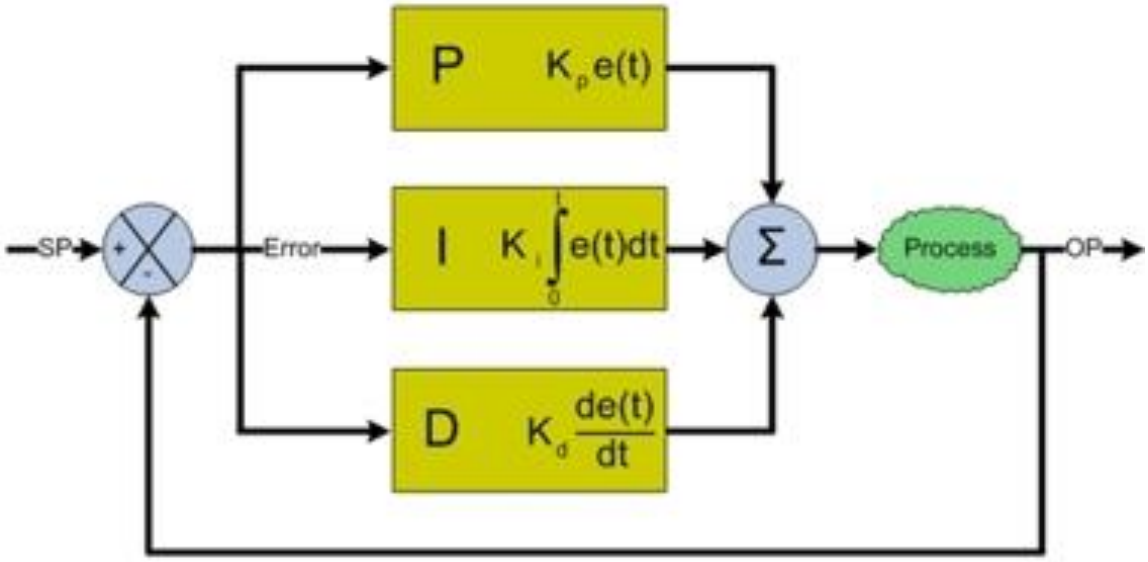


Figure 4: [15] A flow chart showing how the PID movement is calculated. Starting from the left the set point (SP) is compared with the output point (OP), the difference between these is the "error". This error is then operated on by each of the terms in the PID control equation, the value of these terms is then added together to produce a corrective response that is applied to the process.

To control the position of the servo a proportional integral derivative (PID) control program was used. A PID controller outputs the sum of three terms which are functions of an "error" measurement. This "error" measurement is how far away from the set point the system is. In this case the "error" is how far from zero degrees the frame is. Figure 4 shows a flow diagram of equation 12 so it can be seen how the "error" measurement is operated on by the equation.

[16]The PID controller has three parts which are described by the equation below. The first term causes a response which is proportional to the "error" measurement. The second term creates an integral response which takes into account past "error" measurements and accounts for things such as constant forces. The third term creates a derivative response based on the "error". This term takes into account how quickly the system is moving either towards or away from the desired point. This is shown in the equation below;

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)\, d\tau + k_d \frac{de(t)}{dt} \tag{12}$$

where $u(t)$ is the corrective response, k_p, k_i, k_d are all experimentally determined parameters relating to the proportional, integral and derivative terms respectively and $e(t)$ is the "error" term.

The experimentally determined parameters were found using the manual tuning method. Due to time constraints the frame was balanced using only the proportional and derivative terms of the PID control.

3.4 Boat Design

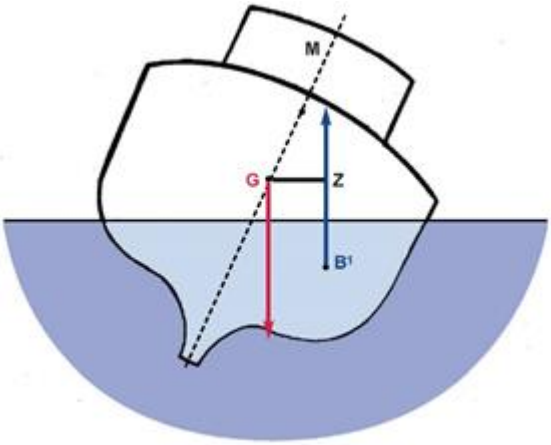


Figure 5: This shows the forces on a normal boat when it starts to roll. M is the metacentre, B is the centre of buoyancy Z is the displacement between the weight and buoyancy forces and G is the centre of mass. [17]

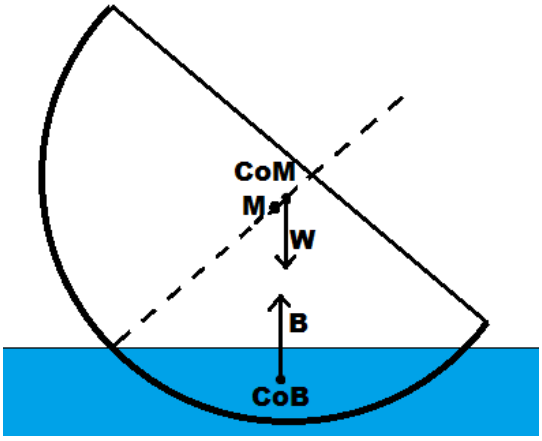


Figure 6: This schematic shows the boat designed for this project. CoM is the centre of mass, CoB is the centre of buoyancy, M is the metacentre, W is the weight force and B is the buoyancy force.

When a stable boat is floating vertically in the water the centre of mass, the point at which the weight force appears to act, and the centre of buoyancy, the point at which the buoyancy force appears to act, are vertically aligned so cancel each other out. When a stable boat rolls to one side the centre of buoyancy is moved in that direction which then creates a horizontal displacement between the weight force (acting downwards) and the buoyancy force (acting upwards). A schematic of this is shown in figure 5. As there are two forces and a horizontal displacement a torque is produced which rotates the boat back towards vertical. [18] The metacentre, M, is the point at which the buoyancy force crosses the centre line of the boat. If the metacentre is above the centre of mass then the boat is stable and a restoring torque will always be produced.

As part of this project a boat was designed which was naturally unstable in order to test the stability of the gyroscopic system in challenging conditions. For a boat to be unstable the metacentre must be below the centre of mass of the boat. To achieve this the hull of the boat was a half cylinder shape so the centre of buoyancy would not move as it rolled. Also the frame containing the flywheels was mounted at the top to raise the centre of mass. A schematic of the boat is shown in figure 6 and a photograph is shown in figure 7.

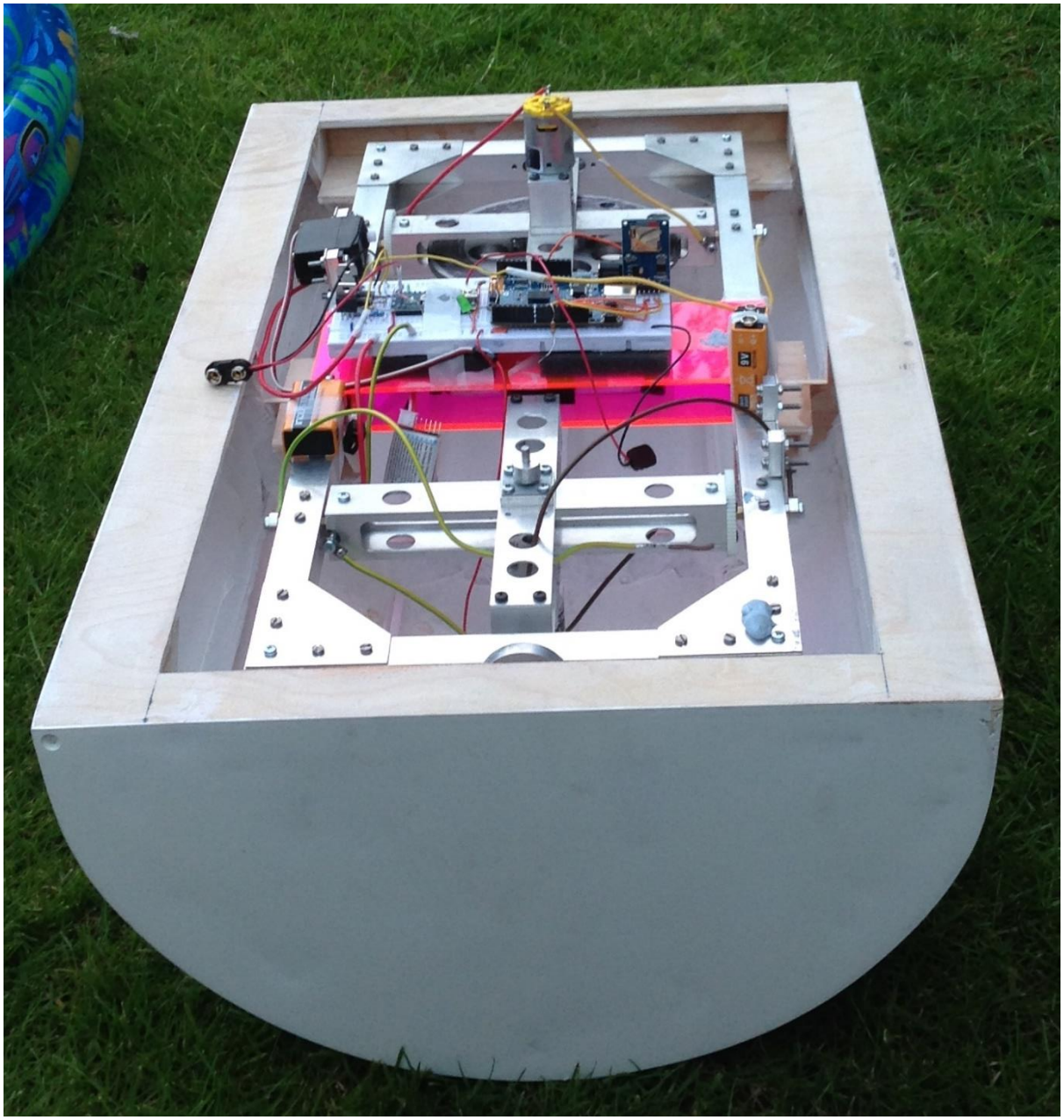


Figure 7: A photograph showing boat used for water based testing. The frame has been mounted inside being fixed in place on mountings built into the boat. The boat was made from wood and fibreglass and the frame was made from aluminium.

4. Experimental details

4.1 Passive system

A system comprising of a flywheel allowed to rotate freely in the z-axis will have a natural gyroscopic response to any net external torque applied about the y-axis. This is defined as the passive system and it relies on the natural movement of the flywheel to apply a torque to return the system to balance.

4.2 Active system

A system comprising of a flywheel forced to rotate about the z-axis will apply an immediate torque about the y-axis when the flywheel is rotated. This is defined as the active system.

4.3 Circuit design

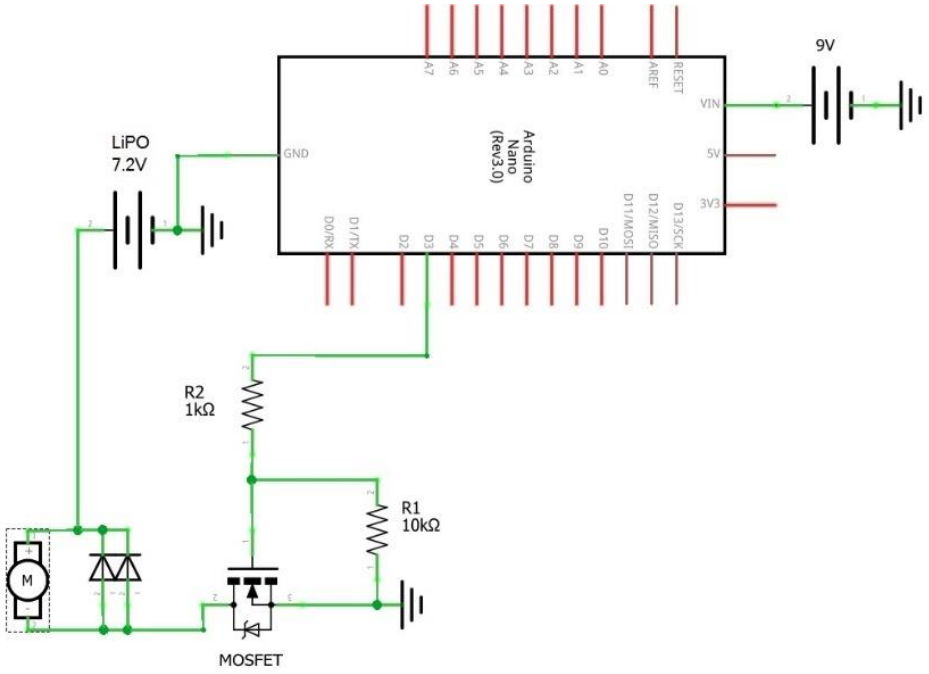


Figure 8: This is a schematic diagram of the circuit used to control the motor used to spin the flywheel. Resistor R1 is a protection resistor used to prevent any power surges from damaging the components. Resistor R2 is a draw down resistor which draws the potential of the MOSFET pin down to zero when the current is turned off. The diodes across the motor prevent back e.m.f produced when the motor is slowing down from damaging any components.

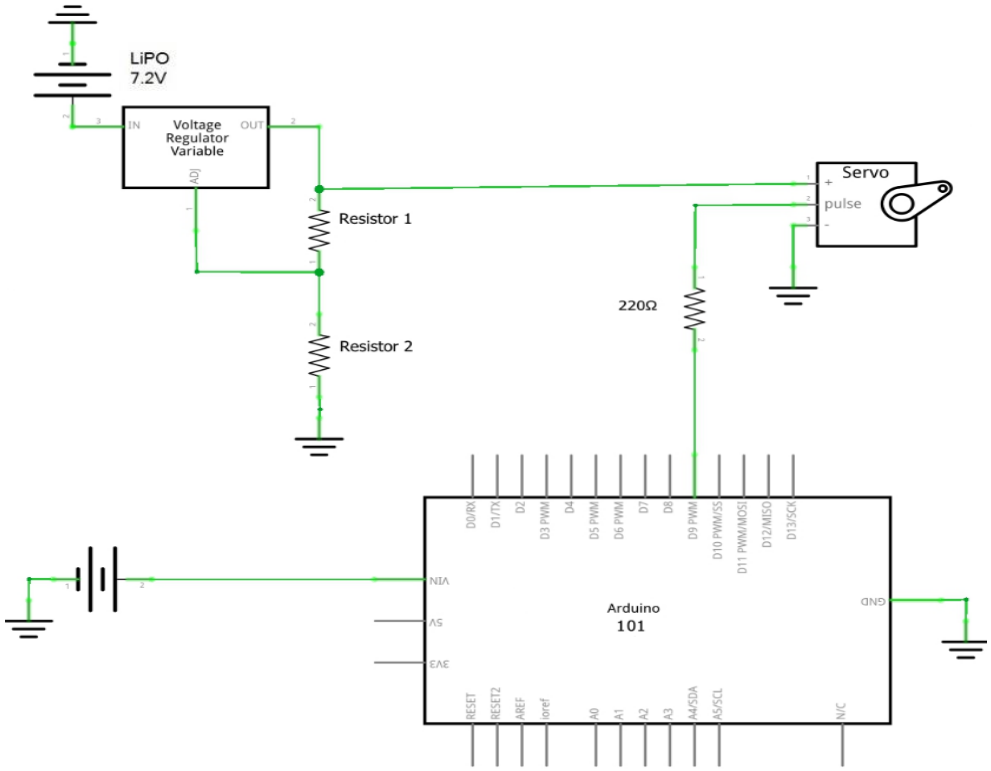


Figure 9: This is a schematic diagram of the circuit used to control the servo. This includes a variable voltage regulator to supply 6V to the servo. As the voltage regular is variable resistors 1 and 2 are needed to set the output voltage. There is a 220Ω protection resistor between the Arduino 101 and the servo. This is to protect the components from a power surge.

4.4 Motor control

To control the motor an Arduino Nano was chosen. The Arduino Nano is a small microcontroller that has programmable input/output pins. Some of these pins are capable of pulse-width-modulation (PWM) outputs and they all output 5 volts.

PWM is a form of voltage control which, instead of delivering a constant voltage, switches the pin on and off very quickly to give a lower voltage. This can be used to reduce the power to the motor without having to use a potentiometer or other analogue device. PWM also has an advantage over analogue devices because very little of the power is wasted through the resistance of the components.

The power demand of the motor was up to 19.68 W with a maximum current draw of 4.41 A. The maximum current draw from the Arduino Nano is 40 mA therefore a transistor was needed to control the power going to the motor. There are various types of transistors, the transistor chosen for this purpose was a Metal Oxide Semiconductor Field Effect Transistor (MOSFET).

MOSFETs have three pins and work by allowing current to flow between two pins, called the source and drain, when a voltage is applied to the third pin, called the gate. The MOSFET chosen is a logic level MOSFET which means that the voltage that needs to be applied to the gate is low enough that it can be used with the microcontroller. The maximum current flow

against the drain to source voltage (motor voltage) for a range of gate voltages is shown in figure 10 which has been taken from the datasheet.

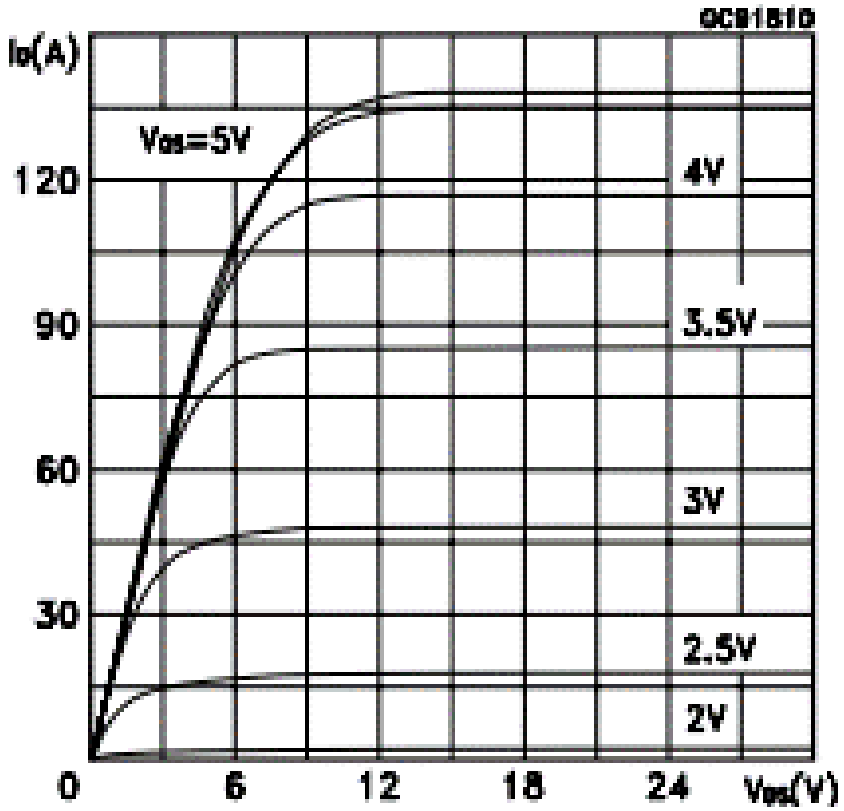


Figure 10: [19] A graph taken from the datasheet of the MOSFET to show the current limits for different gate voltages. The Arduino Nano can supply 4.5V to the gate and the motor uses 7.2V. Therefore the maximum current than can be drawn through the MOSFET exceeds 100A.

The motors that have been used have a maximum voltage of 7.2V supplied from a LiPo battery. The voltage applied to the gate was 4.5V as some of the 5V supplied by the Arduino pin was dropped across the protection resistor. Figure 10 shows that the maximum current allowed through the MOSFET is in excess of 100A which is well above 4.41A which is the motors maximum current draw. This high current rating was one of the main reasons for choosing a MOSFET.

4.5 Voltage regulators

All of the components on the model boat were powered by a LiPo battery which had an output of 7.2V. The servos maximum voltage was 6V therefore a voltage regulator was needed to reduce the voltage for the servo. The voltage regulator that was used was variable and the voltage it outputted was controlled by two resistors, the values of which were worked out using equation 13.

$$V_{out} = 1.25\left(1 + \frac{R_2}{R_1}\right) + I_{ADJ}R_2 \quad (13)$$

where I_{ADJ} is the adjustment current (50μA as standard) and R_2 and R_1 are the resistance of resistors 1 and 2 respectively.

The resistance of resistors 1 and 2 can be chosen to give the required voltage output. This was done by choosing a value for resistor 2 and then re-arrange the equation for resistor 1. It was found that a 1 kΩ resistor provided enough current suppression to be used as resistor 2. Therefore a 266Ω resistor was needed for resistor 1 to reduce the voltage down to 6V for the servos.

4.6 Servo control

An Arduino 101 microcontroller was used to control the servos. The Arduino 101 has a built in Bosch BMI160 chip which contains a 16-bit accelerometer and 16-bit gyro-chip which were used to measure the angle and the rate of rotation of the boat. This then allowed the code to calculate how quickly the flywheel needed to be rotated and in what direction. This information was then used to control the position of the servo.

The servo was controlled via PWM sent through a signal wire and a protection resistor. The position of the servo shaft depends on the duration of time between the signals it receives. The Arduino code has a library built in to deal with controlling the servo which only requires the angle to be written to it. The library then works out and sends the correct PWM signal to the servo.

5. Software

Arduinos were used as the microcontrollers to control the various pieces of hardware. To program the Arduinos code has to be written in the Arduino language and then uploaded to the boards. There were two main software components that needed to be written. These were, a motor control program and an active PID program.

5.1 Motor control program

The motor control program used PWM to gradually increase the power to the flywheel up to a set value and then maintain that value. In PWM the fraction of "time on" is set out of 256. A value of zero means that the power is fully off and a value of 255 means the power is fully on. The motor control program that was written set the motor pin as an output. It then started at a PWM value of zero (power off), wrote this to the motor and then increased this value by 5 and waited 0.5 seconds. It then wrote this new value to the motor, increased it by 5 and then waited 0.5 seconds. This process was repeated until the value written to the motor was the

same as the target value. After this the program wrote the target power to the motor. This is shown in sketch 1.

```
const int MOTOR = 3; // Set motor to pin 3
int i = 0;

void setup() {
  pinMode (MOTOR, OUTPUT); // sets the motor (pin 3) as an output

}
void loop() {
  int motorPower = 50; // use to set the maximum motor speed
  if (i<motorPower) // increases speed up to maximum
  {
    analogWrite( MOTOR, i); // writes PWM speed to motor
    i = i + 5;  // slowly increases the speed
    delay (500);
  }
  else
  {
    analogWrite (MOTOR, motorPower); // writes the motorSpeed to keep the power constant
  }
}
```

Sketch 1: The Arduino code that is used to gradually power up the motor to a set power and then maintain it at this level. At the start of the code a constant variable for the motor is created and then set to pin 3. Pin 3 is then set as an output in the setup. In the main loop the power is increased by 5 every 500 ms and then written to the motor until it reaches the set value, at which point it continuously writes the set power value.

5.2 Active PID program

The program that was written to control the servo in the active system used a PID based control system. To work out the “error” the angle of the frame needed to be measured. This was done using the accelerometer built into the Arduino 101. To use this chip a library called CurieIMU needed to be included in the code. This was included at the start along with the servo and SD libraries. This is shown in sketch 2.

```
#include <Servo.h> // includes the servo library
#include <CurieIMU.h> // includes the IMU library
#include <SD.h> // includes the SD carb library
```

Sketch 2: The Arduino code used to include the necessary libraries in the code. A semicolon terminator is not used at the end of the line when using #include.

In a setup loop the CurieIMU library was then initialised and the range, sample rate, and calibration offset were set for the accelerometer and gyroscope. Also in this loop the servo was attached to a specific pin and the SD card was tested. If the SD card was working an LED was turned on. This is shown in sketch 3.

```
void setup() {
  CurieIMU.begin (); //intialize the library
  CurieIMU.setAccelerometerRange (2); // sets range of acelerometer in units of g
  CurieIMU.setAccelerometerRate (50); // sets sample rate of 50Hz
  CurieIMU.autoCalibrateAccelerometerOffset ( Z_AXIS, 1);
  CurieIMU.autoCalibrateAccelerometerOffset ( Y_AXIS, 0);// calibrates the accelerometer
  CurieIMU.setGyroRate (50); // sets sample rate of 50Hz
  CurieIMU.setGyroRange (250); // sets the range up to 250 degrees per second
  CurieIMU.autoCalibrateGyroOffset (); // sets the offset

  SERVOONE.attach (9); // attaches servo to pin 9
  SERVOONE.write (servoPos); // hold the servo steady while motor spins up

  pinMode (0, OUTPUT);
  if (SD.begin (chipSelect)) // initialise library and check SD card
  {
    digitalWrite (0, HIGH); // turns LED on if card is working
  }
  else
  {
    digitalWrite (0, LOW);
  }

}
```

Sketch 3: The Arduino code used to setup the accelerometer and gyroscope chip, the SD card and the servo. When using an external library the library has to be defined before the instruction is given. For example CurieIMU.begin (), "CurieIMU" defines which library to use and "begin" is the instruction.

To work out the angle of the frame the accelerometer was read ten times and the average acceleration was calculated. As this acceleration was in units of g by taking the cosine of this value the magnitude of the angle was known. The direction of the angle was determined using the acceleration in the z-direction. If this was positive then the angle was positive and if this was negative the angle was negative.

The derivative of the "error" value is the rate of rotation of the frame. As the Arduino 101 contains a gyroscope chip the rate of rotating could be read directly.

The integral of the "error" was found by summing up the past five values of the "error" measurement.

Once all three "error" components of the equation were found they could be combined as in equation 12. The values of k_p, k_i, and k_d were set as global variables so they could be changed at the start of the code. The amount the servo needed to be moved was calculated and this value was then added to the current position of the servo to create a new servo position. This was then written to the servo. This is shown in sketch 4.

```
float servoMove = Kp*angleError + Ki*averageAngleError + Kd*xRotRate; // multiplies each term be the tuning parameter the adds together

if (servoPos > 10 && servoPos < 170 && angleError < 70.00 && angleError > -70.00 && averageAngleError < 70.00 && averageAngleError > -70.00
&& xRotRate < 100.00 && xRotRate > -100.00 && servoMove < 10.0 && servoMove > -10)
// if statement to prevent the servo moving pasts its max and min points
{
  servoPos = servoPos + (int)servoMove; // sets a new value for the servo position
  SERVOONE.write (servoPos); // writes the servo to new position
}
```

Sketch 4: The Arduino code used to determine the amount to move the servo and then write this to the servo. An "if" statement was used to ensure the servo did not go past its allowed maximum point. This if statement also filtered out any results which were outside the expected range.

After the servo had been moved the angle of the frame, the angle of the servo and the time were saved to the SD card. This information was saved in a text file but was delimited using commas which made it easier to import to excel. The code for this is shown in sketch 5.

```
File dataFile = SD.open("angle.txt", FILE_WRITE); // opens a text file called "angle" on SD card

if (dataFile) // checks for a malfunction
{

  dataFile.println (thetaDeg); // print data to text file on SD card
  dataFile.print (","); // seperate the data
  dataFile.print (millis ()); // indicates the time since program started
  dataFile.print (",");
  dataFile.print (servoPos); // print data to SD card
  dataFile.print (","); // serperates the data
  dataFile.close (); // have to close file after writing
}
else
{
  digitalWrite (0, LOW); // indicates if there is an error by turning LED off
}
```

Sketch 5: The Arduino code used to write information to the SD card. The SD card was opened and then a file type and name were given. The code then checked for malfunctions, if there was one then the indicative LED turned off, if not the code printed the servo position, frame position and time to the file before closing it. The file had to be closed and then re-opened on every iteration because other code cannot run if the file is open.

The full Arduino code used for the active system can be found in appendix-A.

6. Results

6.1 Moments of inertia of the LMOI, HMOI, frame and boat

The moment of inertia of both the flywheels and the frame was measured as described in section 3.1. Figure 11 shows an example of the graphs drawn. The gradient of which is the angular acceleration which was then used in equation 7 to find the moment of inertia. A summary of all the angular accelerations is given in table 1.

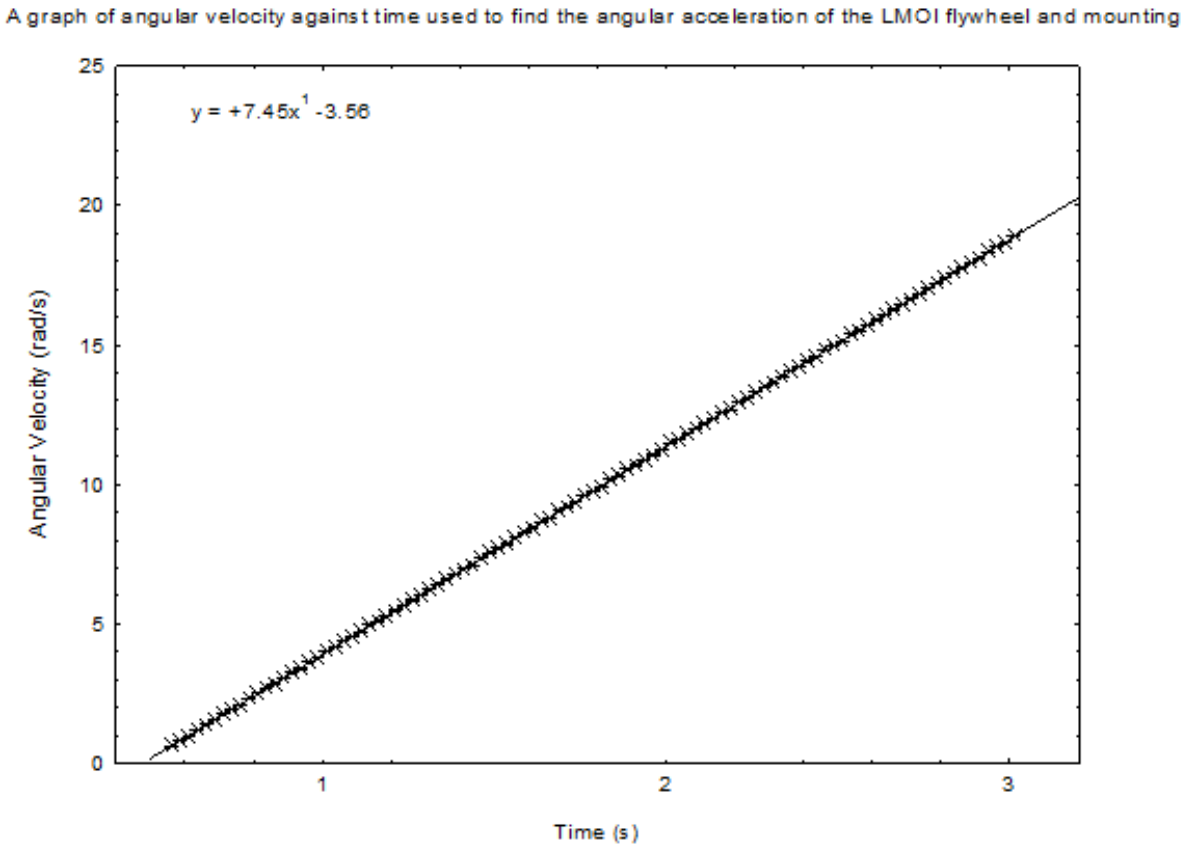


Figure 11: A graph used to find the angular acceleration of the HMOI flywheel and its mount. The flywheel and mount were accelerated using a constant torque and the gradient of this graph is the angular acceleration. The angular acceleration was then used to calculate the moment of inertia of the flywheel and mount. The moment of inertia of the mount was then subtracted to calculate the moment of inertia of the flywheel.

Repeat	Frame linear acceleration (ms^{-2})	LMOI flywheel and mount angular acceleration ($rads^{-2}$)	HMOI flywheel and mount angular acceleration ($rads^{-2}$)	HMOI flywheel mount angular acceleration ($rads^{-2}$)	LMOI flywheel mount angular acceleration ($rads^{-2}$)
1	0.0641	36.6	7.45	65.7	59.2
2	0.0648	36.7	7.46	67.2	59.3
3	0.0645	36.8	7.40	67.1	59.9
4	0.0635	37.1			58.8
5	0.0644	37.0			59.1
Mean	0.06426	36.8	7.44	66.7	59.3
Standard Deviation	4×10^{-4}	0.2	0.03	0.7	0.4

Table 1: A table containing the measurements of angular acceleration of the frame, the HMOI flywheel and the LMOI flywheel. The angular acceleration was used to calculate the moment of inertia. The standard deviation of the repeats was used as the error in the measurement as this exceeded the physical error in the rotary motion sensor.

The standard deviation of the measurements was used as the error. This is because it was found to be much larger than the error of the rotary motion sensor. The error in the torque was then combined in quadrature with the error in the angular acceleration to find the error in the moment of inertia.

The moment of inertia of the HMOI flywheel and its mounting was found to be (2.29 ± 0.01) $x10^{-3}$ kg m^2. The moment of inertia of the mounting was found to be (2.56 ± 0.03) $x10^{-4}$ kg m^2. Therefore the moment of inertia of the HMOI flywheel was found to be (2.03 ± 0.01) $x10^{-3}$ kg m^2.

The moment of inertia of the LMOI flywheel and the mounting was found to be (3.82 ± 0.02) $x10^{-4}$ kg m^2. The moment of inertia of the mounting was found to be (2.37 ± 0.02) $x10^{-4}$ kg m^2. Therefore the moment of inertia of the LMOI flywheel was found to be (1.45 ± 0.02) $x10^{-4}$ kg m^2.

The moment of inertia of the frame was found to be (0.0123 ± 0.0001) kg m^2.

The moment of inertia of the boat and frame used in the water based testing was (0.1225 ± 0.0012) kg m^2. The mass of this was 5.112 ± 0.001 kg.

6.2 Laboratory testing of the passive system using the LMOI flywheel

The passive system was first tested with the LMOI flywheel. It can be seen from figure 12 that using the LMOI flywheel at 2300 RPM was insufficient to achieve passive balancing as it rotated past the recovery point after only 1.5 seconds. Further tests were conducted at higher rotation speeds with the smaller flywheel but due to vibrations from the higher speeds interfering with the accelerometer chip the recorded data was unplottable.

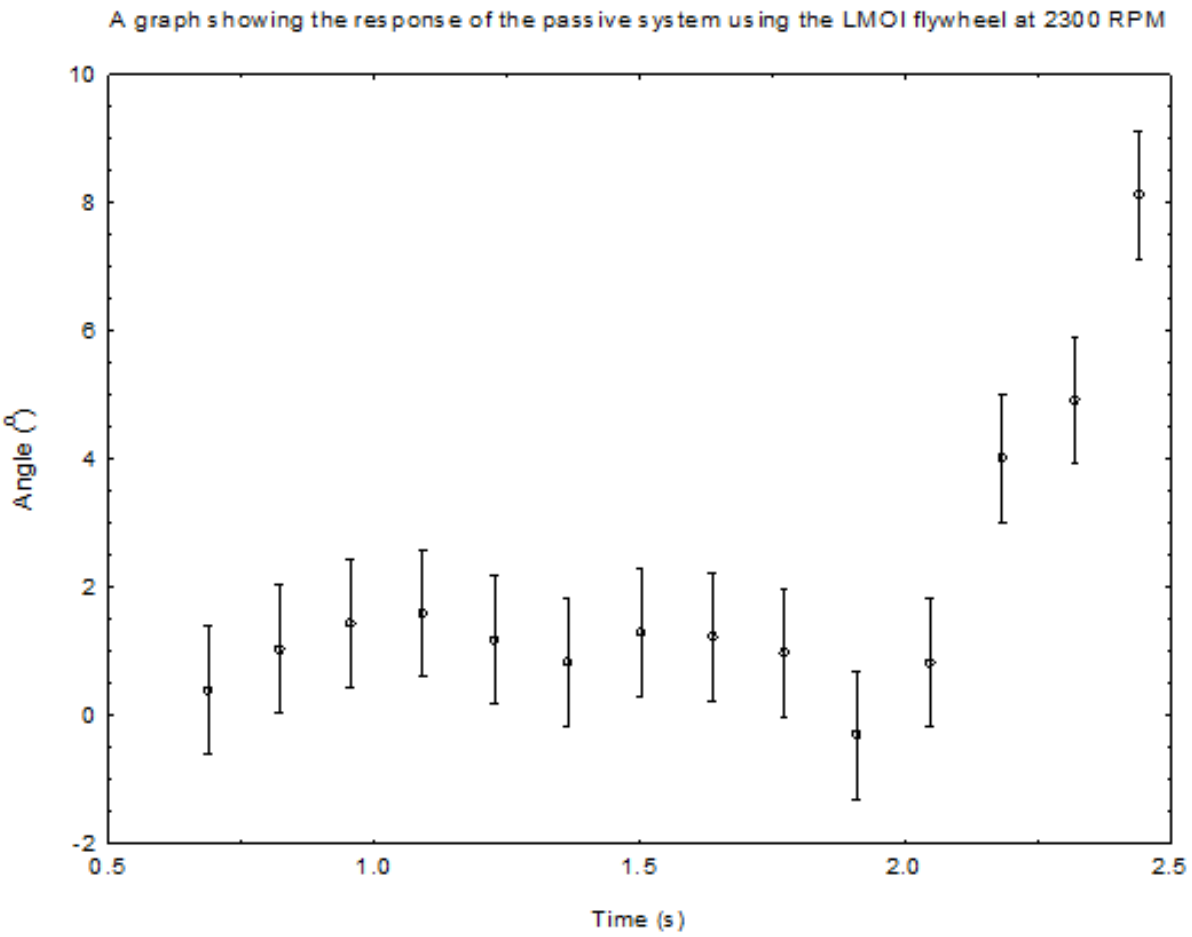


Figure 12: A graph showing the passive response of the frame when the passive system was tested using the LMOI flywheel. It can be seen that the frame has no appreciable response to the rotation of the frame so balance was not achieved.

6.3 Laboratory testing of the passive system using the HMOI flywheel

Five passive tests were conducted using the HMOI flywheel at 700, 1100, 1300, 1500 and 1900 RPM rotational speeds. Because the HMOI flywheel had a moment of inertia that is an order of magnitude bigger than the LMOI flywheel lower rotation speed tests could be conducted.

Because the frame was unstable when it was released it would rotate about the y-axis. When the frame rotated it caused a torque in the z-direction which made the flywheel rotate about the z-axis. Since the motor is mounted above this axis then the weight of the motor caused this torque to be even greater. The rotation of the flywheel about the z-axis then produced a torque about the y-axis that acted in the opposite direction to the initial rotation of the frame. This torque would then bring the frame back up to the zero position (parallel with the ground) but then overshoot, rotating the frame in the other direction. The frame rotating in the other direction would then cause the flywheel to rotate about the z-axis in the opposite direction but this time the weight of the motor would work against the torque of the frame causing the rotation rate to be slower and the frame to achieve a higher angle. Once the flywheel passed the zero point (the angular momentum pointing in the x-direction) the weight of the motor would then add to the torque applied by the frame to increase the rate of rotation of the

flywheel about the z-axis. This increase in the rotation rate of the flywheel would then apply a torque to the frame to return it to the zero position.

This cycle stopped when the flywheel was no longer able to provide enough torque to the frame to return it to the zero position. As the torque produced by the flywheel in the y-direction is dependent on $\cos\theta(t)$ its magnitude diminishes as $\theta(t)$ approaches 90º. Therefore as the angle of the flywheel gets greater the torque it applies to the frame gets smaller until it is unable to rotate the frame back to the zero point. This was when the frame flipped over and measurements were stopped.

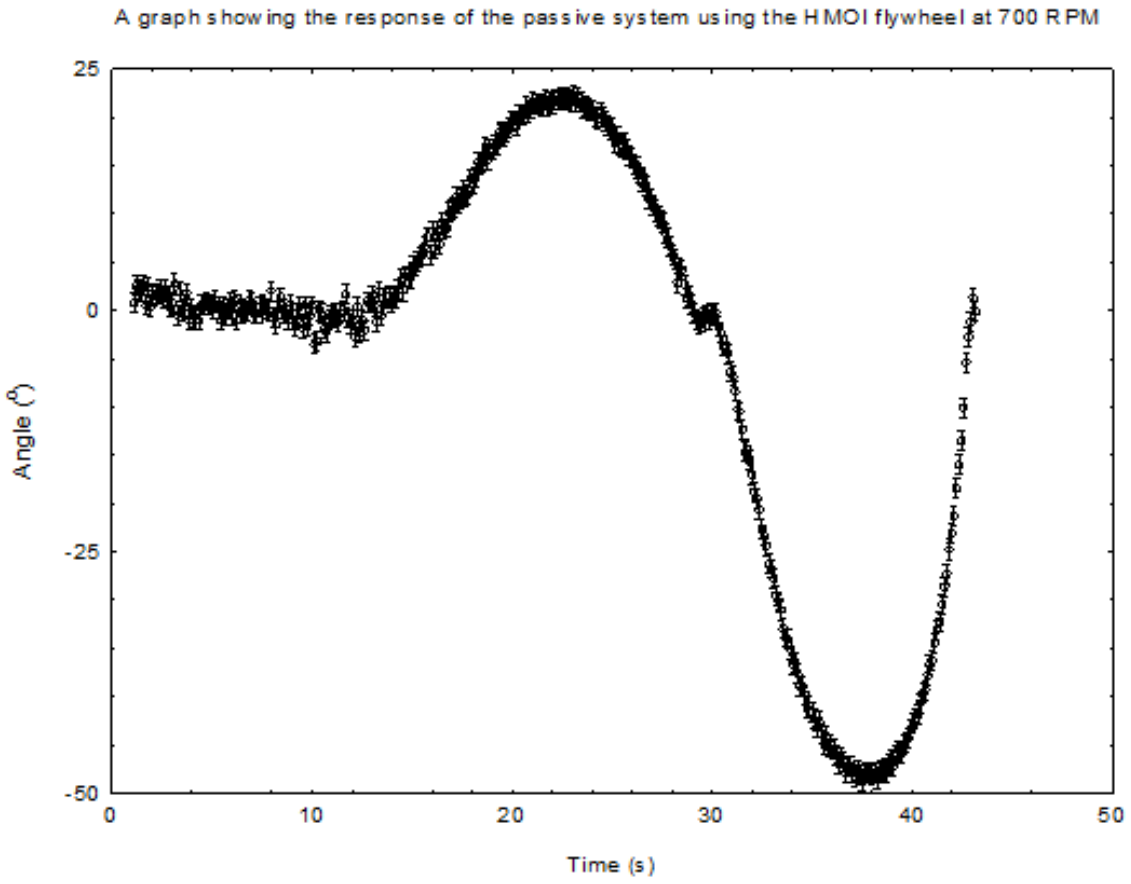


Figure 13: A graph showing the angle of the frame against time when the passive system was tested using the HMOI rotating at 700 RPM. It can be seen that the passive system responded to the free rotation of the frame and started oscillating about the zero point with increasing amplitude.

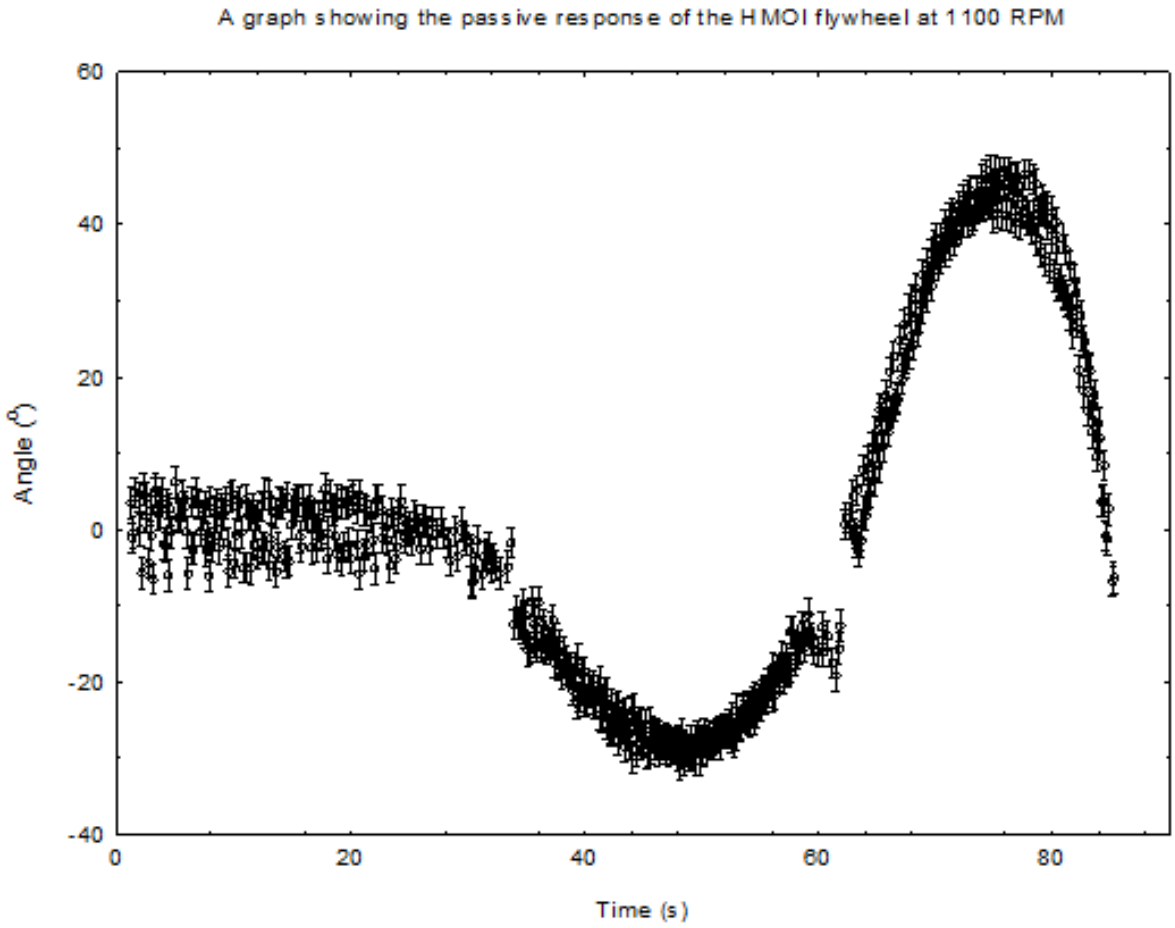


Figure 14: A graph showing the angle of the frame against time when the passive system was tested using the HMOI rotating at 1100 RPM. It can be seen that the passive system responded to the free rotation of the frame and started oscillating about the zero point with increasing amplitude.

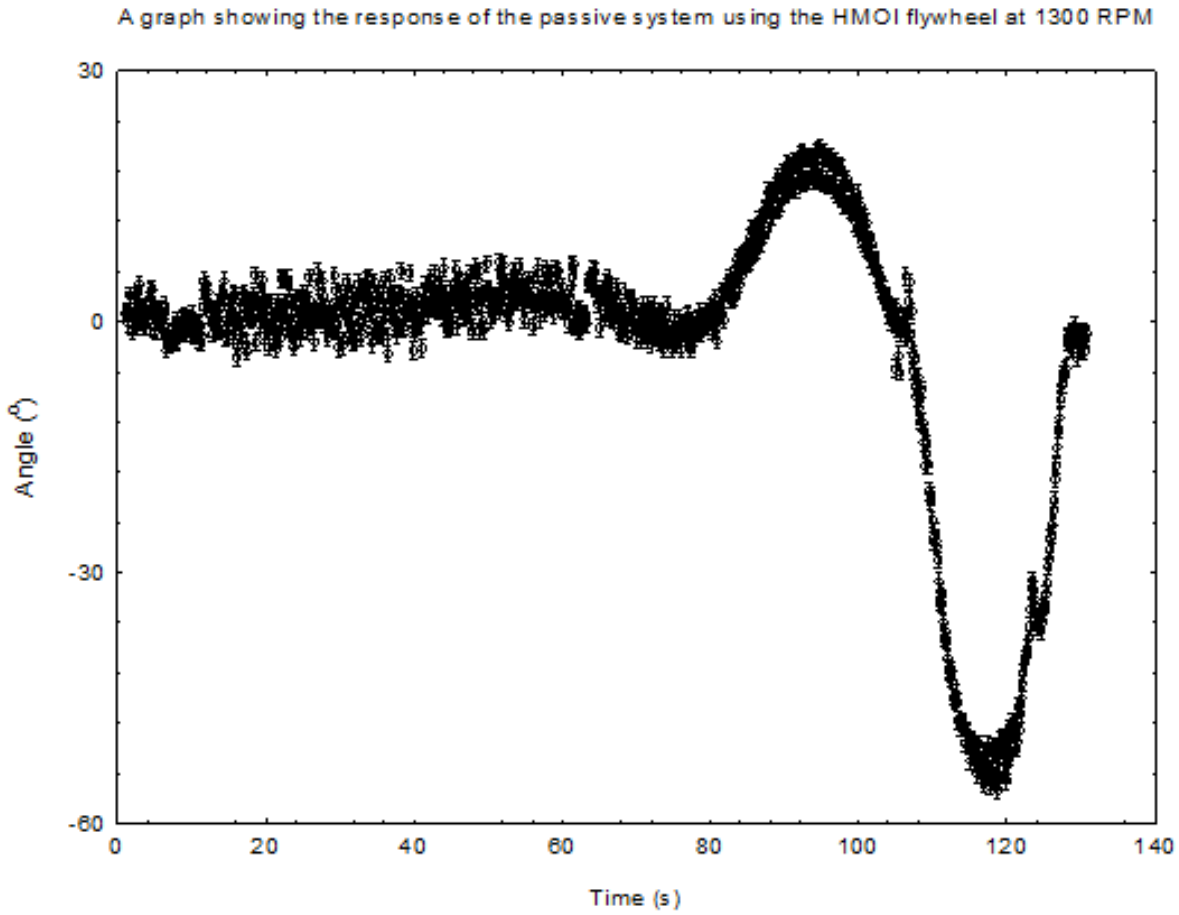


Figure 15: A graph showing the angle of the frame against time when the passive system was tested using the HMOI rotating at 1300 RPM. It can be seen that the passive system responded to the free rotation of the frame and started oscillating about the zero point with increasing amplitude. There is an extended period of time where the frame does not rotate about the y-axis, this may have been due to the bearings holding it steady or other external factors.

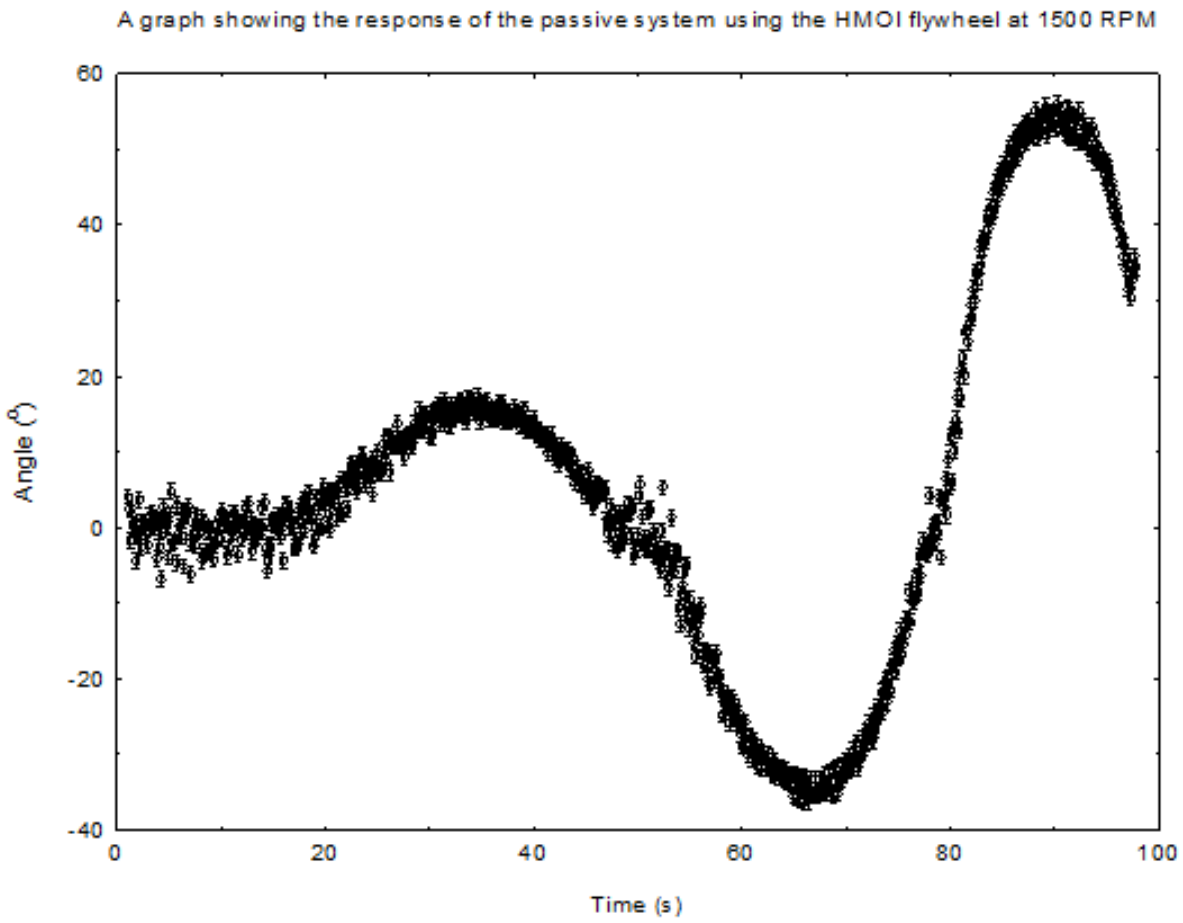


Figure 16: A graph showing the angle of the frame against time when the passive system was tested using the HMOI rotating at 1500 RPM. It can be seen that the passive system responded to the free rotation of the frame and started oscillating about the zero point with increasing amplitude.

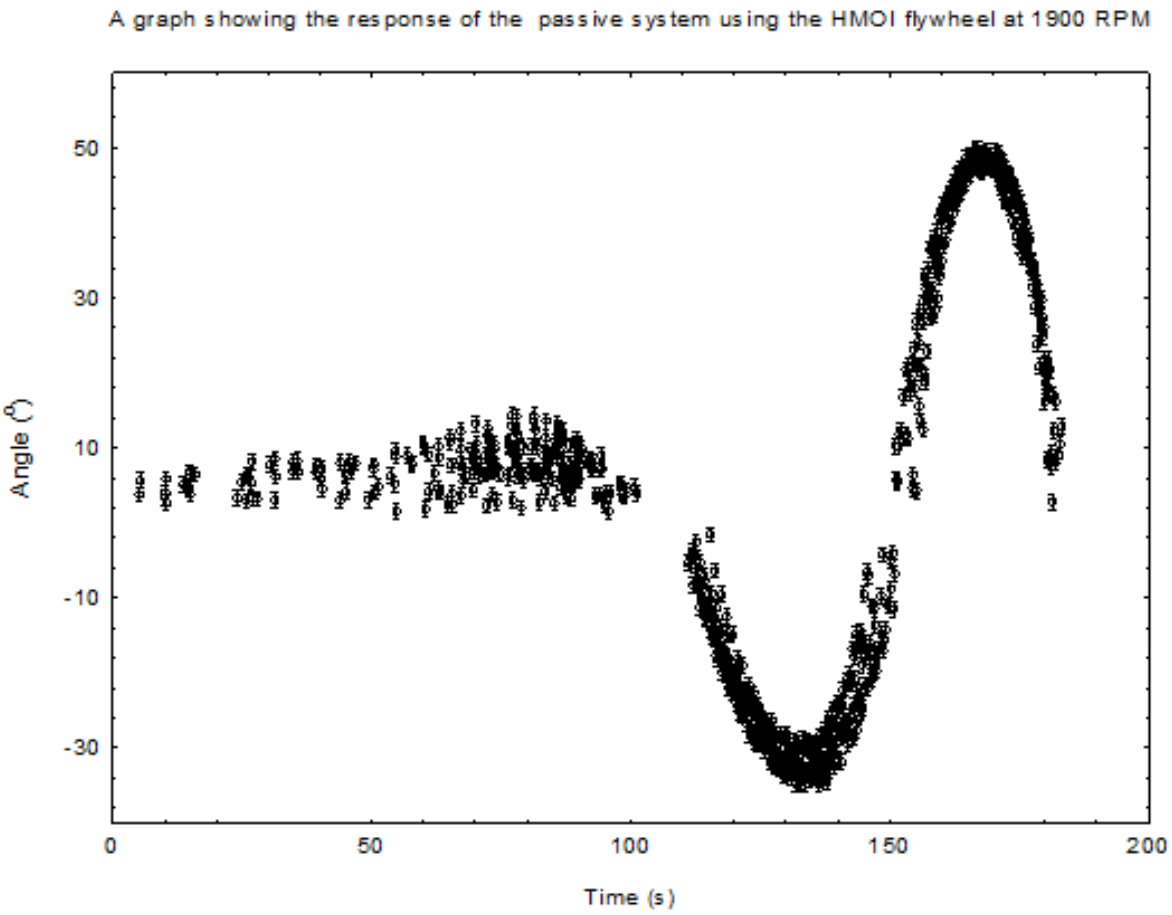


Figure 17: A graph showing the angle of the frame against time when the passive system was tested using the HMOI rotating at 1900 RPM. It can be seen that the passive system responded to the free rotation of the frame and started oscillating about the zero point with increasing amplitude.

The time it takes the frame to go from its maximum angle to its minimum angle is half the time period of the oscillation. As the angular speed of the flywheel was increased the time period also increased. Figure 18 shows a graph of the time period increasing with increasing angular speed.

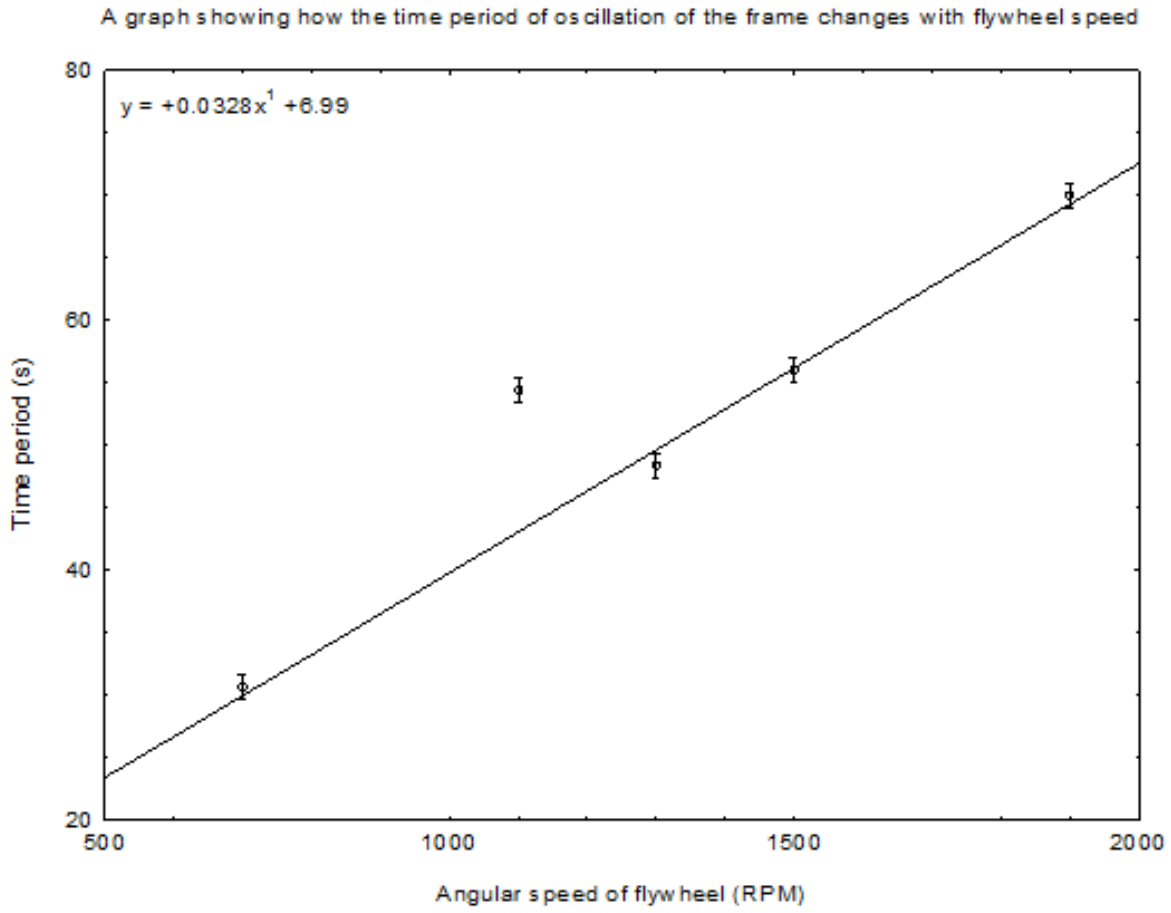


Figure 18: A graph demonstrating the linear relationship between the time period of the frame and the rotational speed of the flywheel. It indicates that as the angular momentum of the flywheel increases the period of oscillation does also.

The above graph indicates that the time period increases linearly with the flywheel speed. As the moment of inertia of the flywheel stays constant and the angular momentum, L_0, is the product of the moment of inertia and the angular velocity, the time period increases with the magnitude of the angular momentum. There is an outlier from the trend at 1100 RPM. This may have been caused by a physical impedance such as friction in the frame bearings. Given more time this would have been repeated.

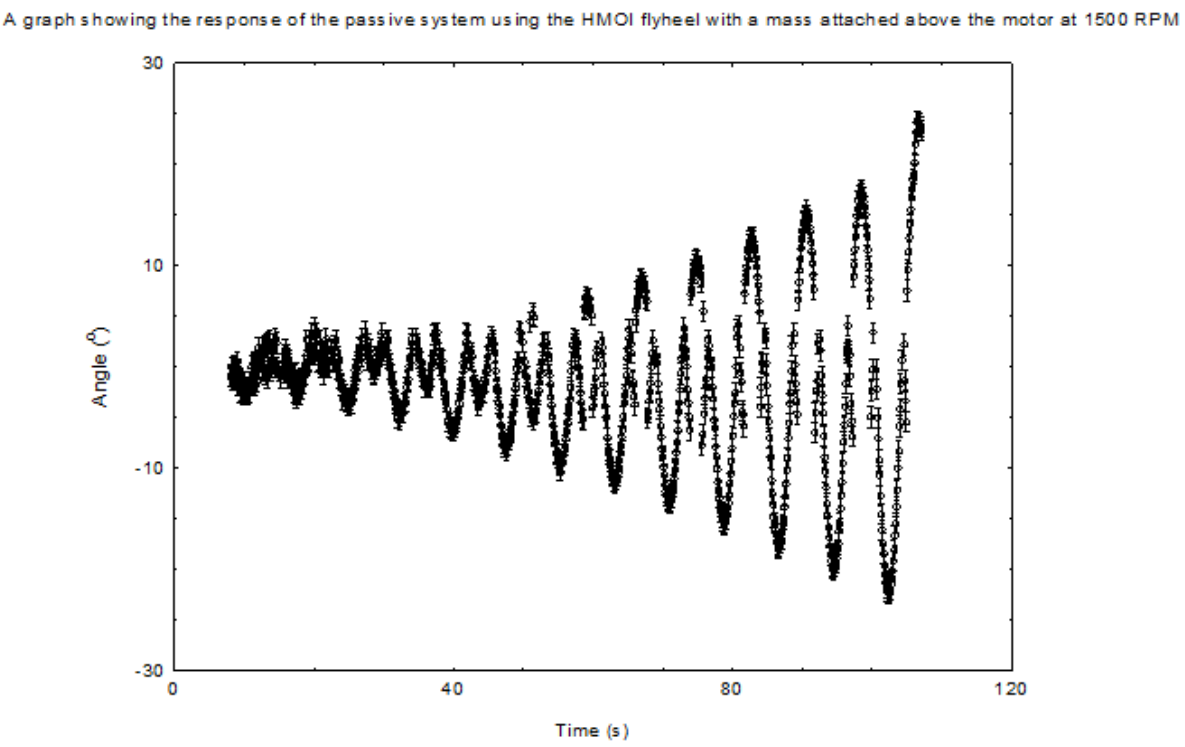


Figure 19: A graph showing the frame oscillations using the passive system when a mass was attached above the motor. It can be seen from this figure that the additional torque the weight of the mass provided caused the time period to decrease.

A mass was attached above the motor to investigate how increasing the weight force would affect the oscillation rate of the frame. The time period for this can be seen in figure 19 and is (8.2 ± 0.5) seconds. This is much smaller than (56 ± 1) second time period from figure 16 which had the same angular speed but did not have the mass attached. This indicated that a higher torque, caused by the increased weight and higher centre of mass, rotated the flywheel about the z-axis faster increases the oscillation speed therefore decreasing the time period.

6.4 Laboratory testing of the active system using the LMOI flywheel

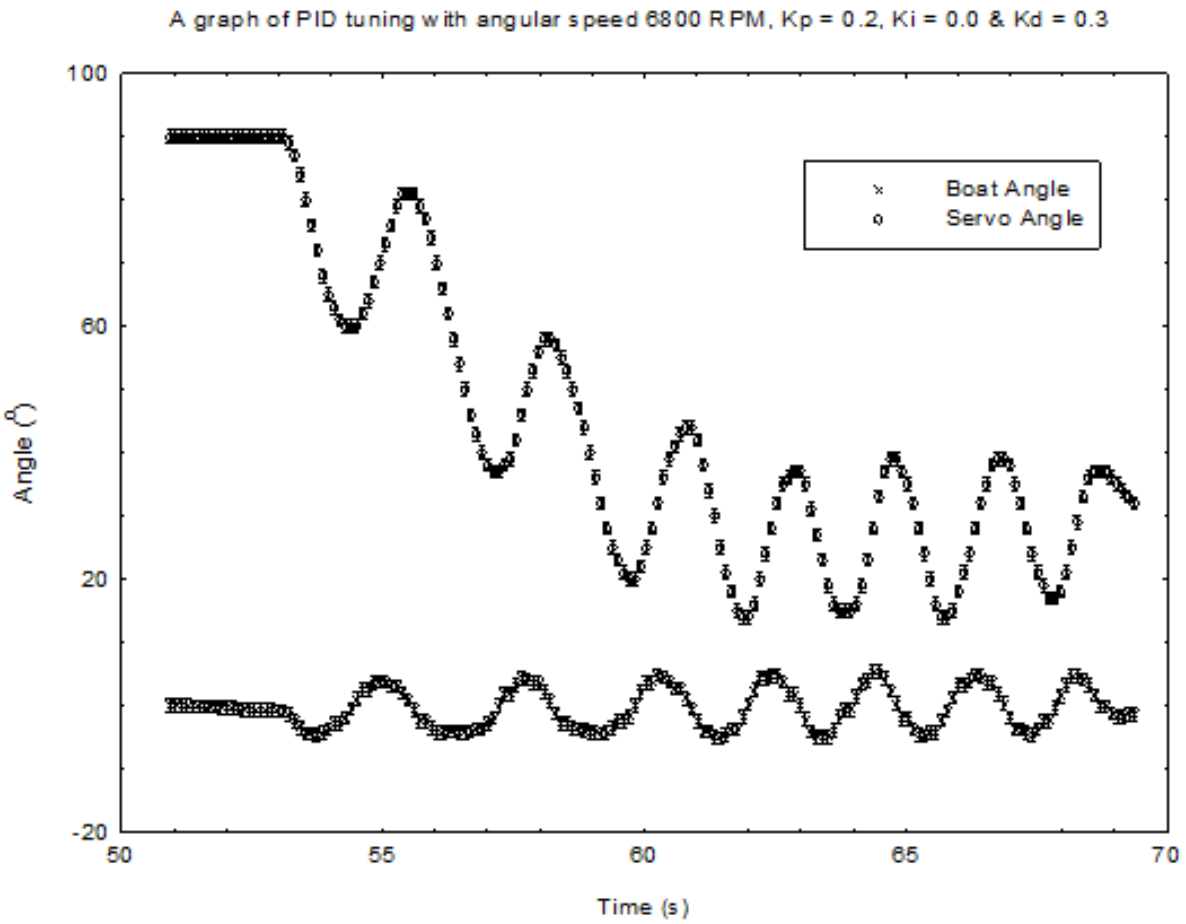


Figure 20: A graph of the frame angle against time for the active system using the LMOI flywheel. This shows that the PID control was responding to the movement of the frame although the system did need to be tuned.

The LMOI flywheel was used first to gauge how the software and the frame would respond so minor adjustments could be made before the more powerful flywheel was fitted. This was done to prevent destructive failure occurring if something was not set up correctly with the HMOI flywheel.

Figure 20 shows that the PID software written was responding to the movement of the frame. As the frame rotated about the y-axis the software recognised this and turned the servo. This turning of the servo rotated the flywheel in the z-axis which then created a torque about the y-axis which rotated the frame the other way. This can be seen by the oscillation of the frame about the zero point. This continued until the servo reached its endpoint and then the frame stopped being balanced because no more torque could be applied.

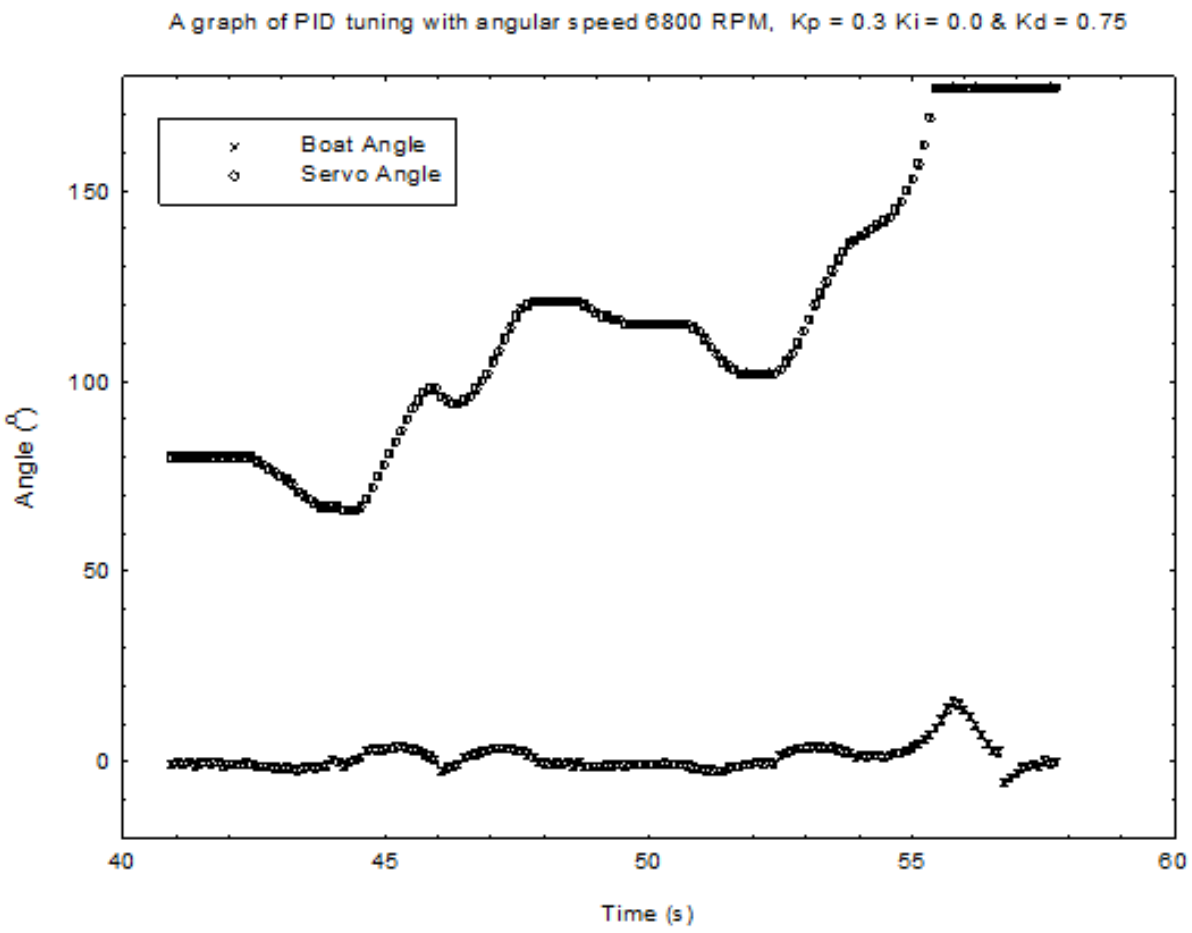


Figure 21: A graph of the frame angle against time for the active system using the LMOI flywheel. The tuning parameters were adjusted to stop the frame overshooting the midpoint. The frame lost balance when the maximum servo angle was reached.

The value of the k_p and k_d constants were changed to reduce the frames oscillation about the midpoint. This is shown in figure 21. Once the servo reaches its maximum angle the frame loses its balance and topples.

6.5 Laboratory testing of the active system using the HMOI flywheel

Figure 21 shows that using the LMOI flywheel the frame can be maintained close to the midpoint for a period of time until the servo reaches its maximum. Because the angular momentum of the spinning LMOI is low large servo angles are needed to produce the required torque to stabilise the frame. Using the HMOI flywheel a larger angular momentum can be produced so the angle the servo had to move was reduced. Therefore the frame could be balanced for a longer period of time before the servo maximum was reached.

It was still necessary to tune the PID control system with the parameters of k_p and k_d so the servo moved the correct amount. The system was tuned by adjusting the parameters using the manual method of tuning. Using this method the value of k_d was set to zero and k_p was

increased until the frame would slightly overshoot the midpoint. Once this point had been reached the value of k_d was increased until the frame could hold itself steady. The value of k_p was then increased until the frame balanced about the midpoint.

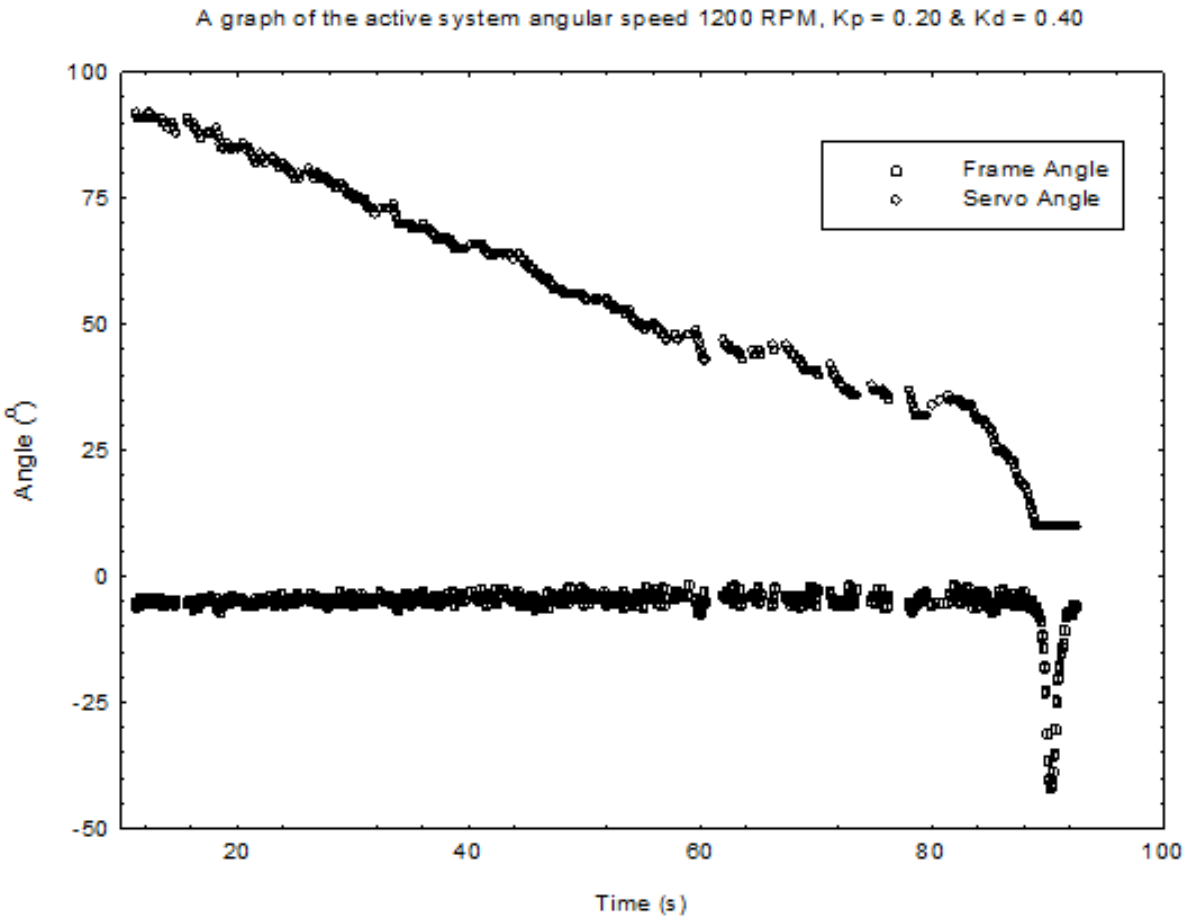


Figure 22: A graph showing the angle of the frame against time using the active system with the HMOI flywheel. The tuning parameters needed some further adjustment as the frame was just being held im position and did not have enough torque to return to the zero point.

Figure 22 shows the frame being held at -5^o by the torque produced by the flywheel. This indicates that the torque is only just balancing the weight of the frame so the frame cannot be rotated back to the midpoint. The flywheel keeps applying this balancing torque until the servo angle reaches its maximum and is unable to rotate the flywheel any further.

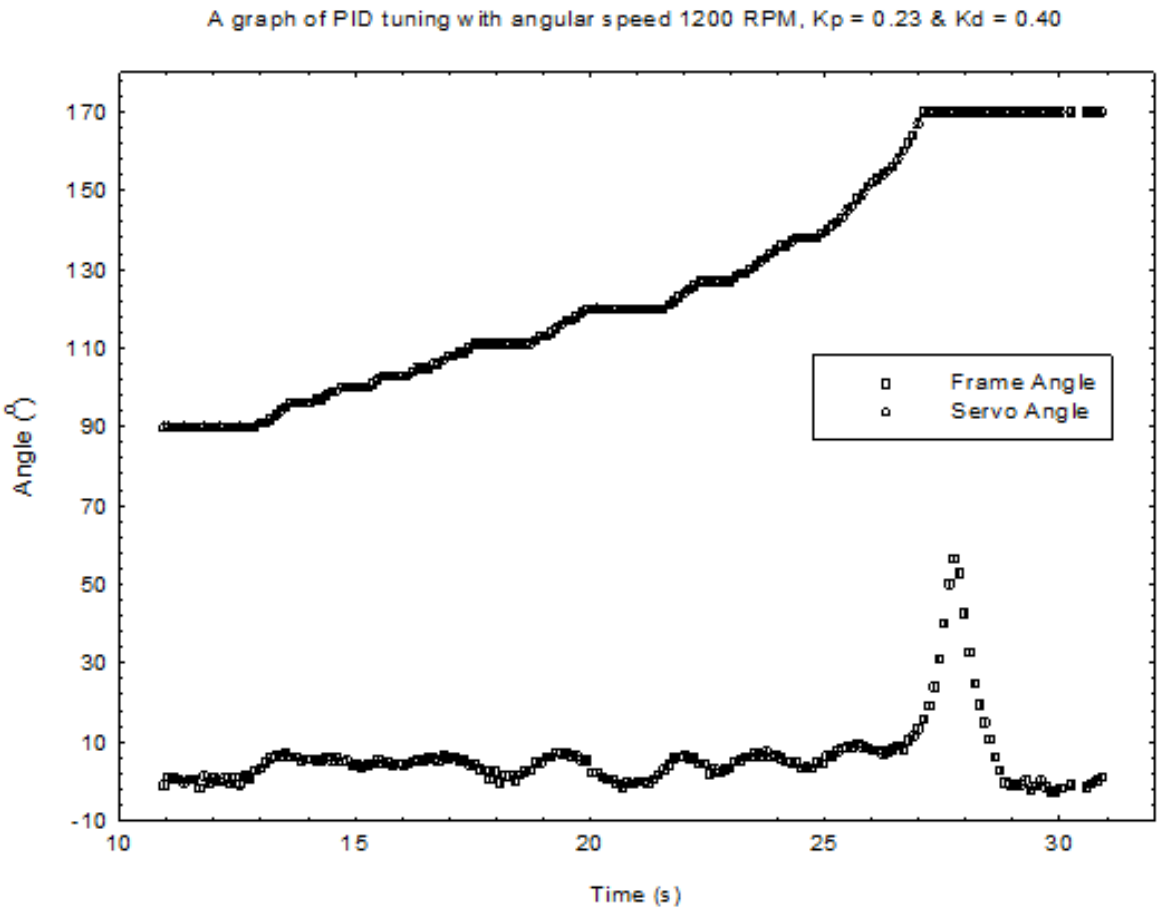


Figure 23: A graph showing the angle of the frame against time using the active system with the HMOI flywheel. The tuning parameters were improving but still needed some more adjustment. It can be seen that the maximum servo angle was reached at 27 seconds which is when the frame lost balance.

The value of k_p was increased to 0.23 so the flywheel would produce more torque to bring the frame past its midpoint to stop the maximum servo angle being reached. This is shown in figure 23. It can be seen from this graph that although the frame went through the midpoint several times it still had an overall bias to one side which meant the servo moved further in one direction than the other so it reached its maximum after 16 seconds.

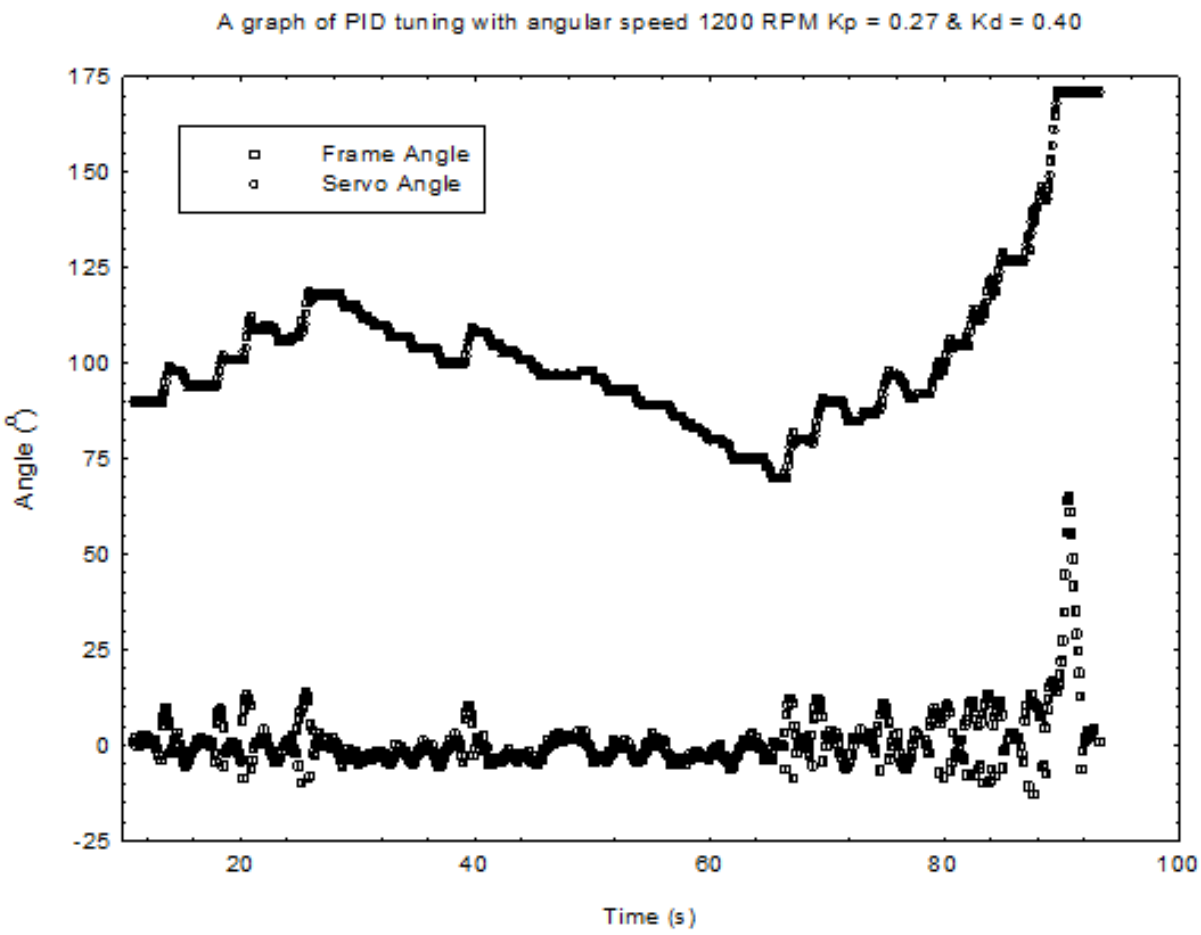


Figure 24: A graph showing the final frame test using the active system. The tuning parameters kept the frame within 15° of the zero point until the endpoint of the servo was reached. As the frame passed through the vertical several times the servo changed direction several times which meant more time passed before the servo maximum point was reached.

The parameter k_p was increased to 0.27 so the flywheel would produce more torque to rotate the frame past the midpoint. The system did this and in figure 24 it can be seen that the frame rotated from 15° to -15° as the servo moved the flywheel to rotate the frame towards the midpoint. The servo reached its maximum point after 80 seconds so could no longer rotate the flywheel to produce a torque. The frame then lost stability.

6.7 Water based testing of the active system using the HMOI flywheel

The active system was tested in a boat designed to be naturally unstable. This boat was unstable to the extent that it would capsize without the stabilisation of the gyroscopic system. To test the active system the boat was manually rolled to an angle of between 15 and 20 degrees using a marker on the hull. It was then allowed to respond either by restoring the boat to equilibrium or capsizing. The ability of the active system to correct the roll of the boat could then be analysed. The initial manual rolling of the boat would simulate a large wave hitting it.

Some initial test were conducted to measure the reaction of the boat using only a proportional response. And then a mainly derivative dominated response. These tests were used to judge the characteristics of the two tuning parameters that were used. Knowing the characteristics of the response of each of the tuning parameters would allow for quicker manual tuning.

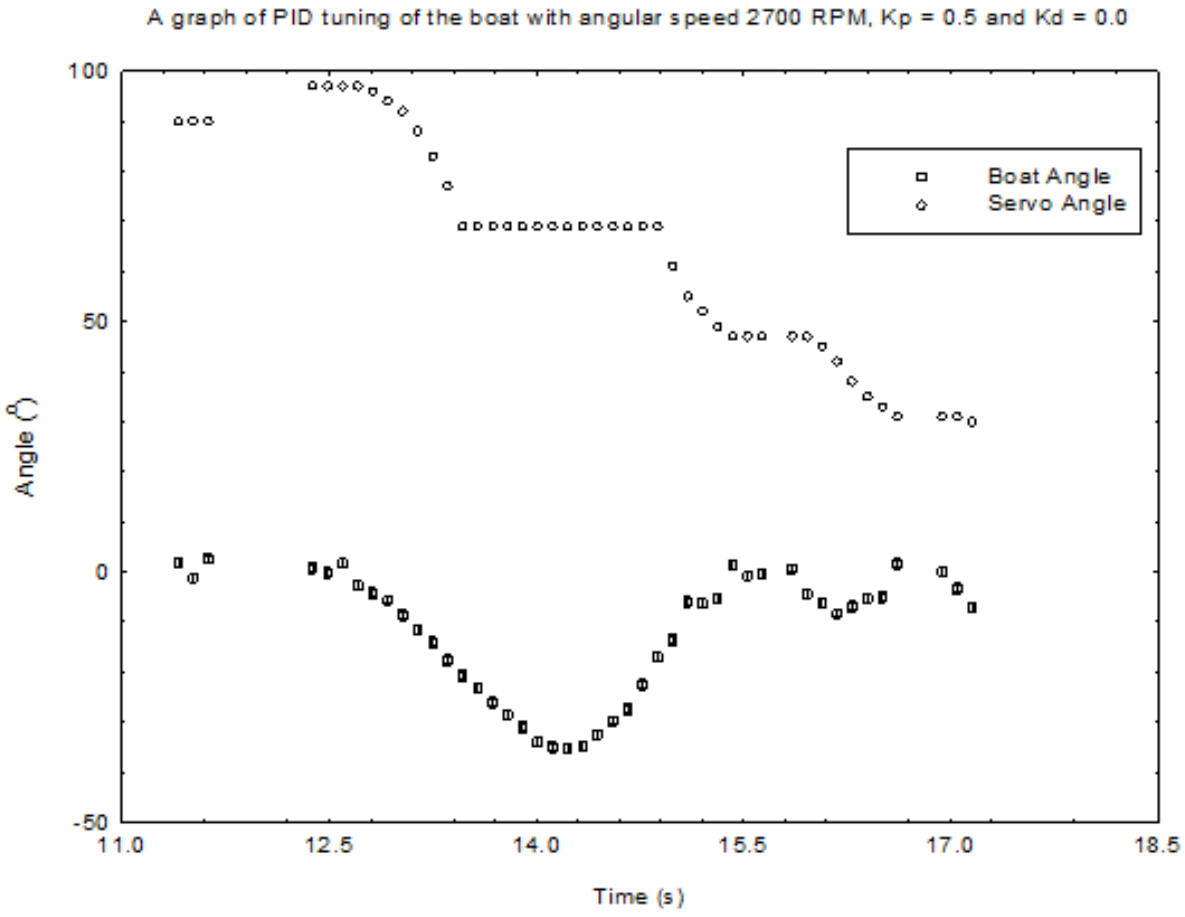


Figure 25: A graph showing the response of the PID control when only the proportional parameter was 0.5. The system responded slowly and forcefully to rotate the boat to nearly capsize. The boat had to be physically stopped from capsizing.

It can be seen from figure 25 that with the value of k_d set to zero and k_p set to 0.5 the system overreacted to the small manual rolling. The boat rotated so far in the negative direction during this test it had to be physically stopped from capsizing at 14.2 seconds.

Another test was then conducted, this time k_d was set to 1.0 and of k_p was set to 0.2. These settings gave the boat a response dominated by the derivative term. The response of the boat can be seen in the graph below.

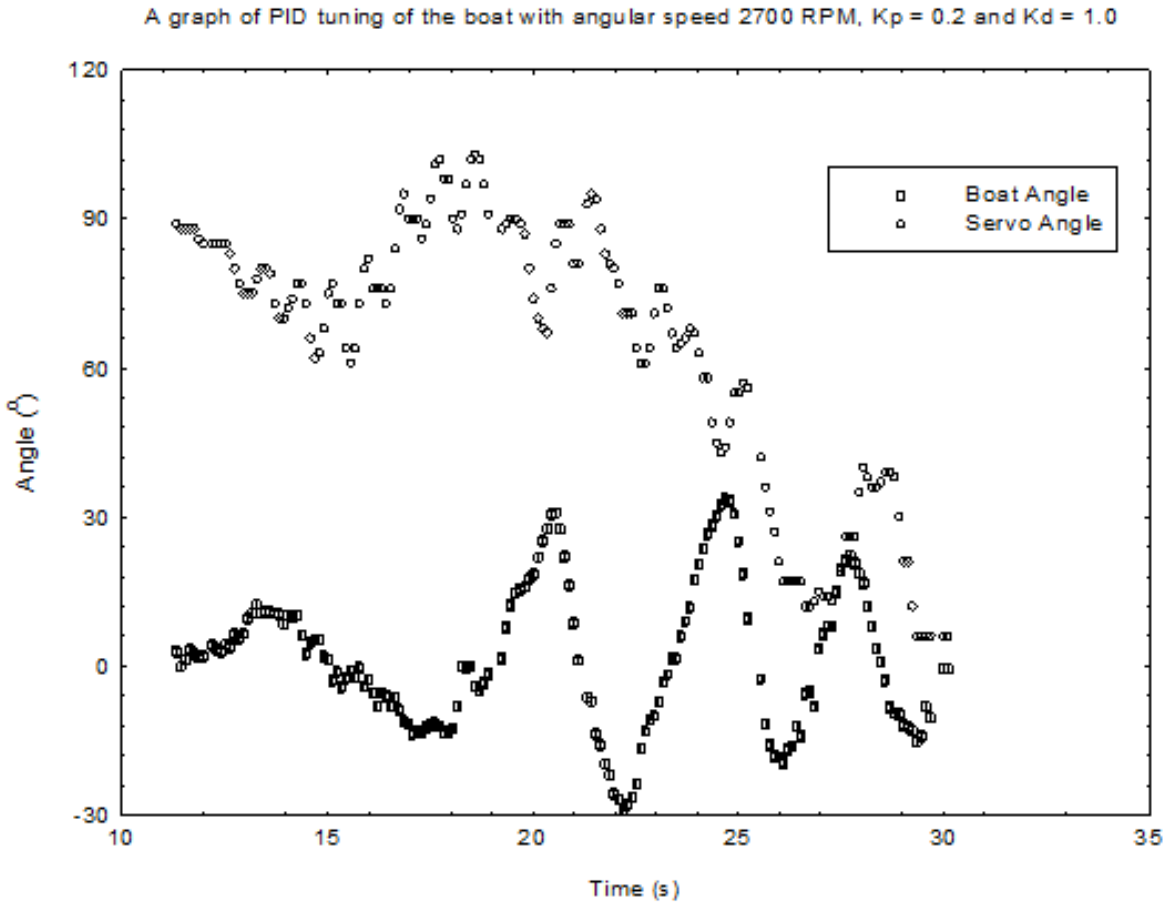


Figure 26: A graph showing the response of the PID control when the proportional parameter is set to a small value and the derivative parameter is set to a large one. The system responded quickly but not smoothly. This made the boat rotate quickly from one place to another.

The dominance of the derivative term made the reaction of the boat very quick. This led to the boat being violently rotated from 30° to -30° about the y-axis. This can be seen in figure 26.

Figures 25 and 26 show how the boat reacts using the PID controlled active system when extreme tuning parameters are used. The proportional term gives a large but slow response and the derivative term is quick to react which can throw the boat from side to side. To tune the boat to recover quickly some combination of k_d and k_p needed to be used.

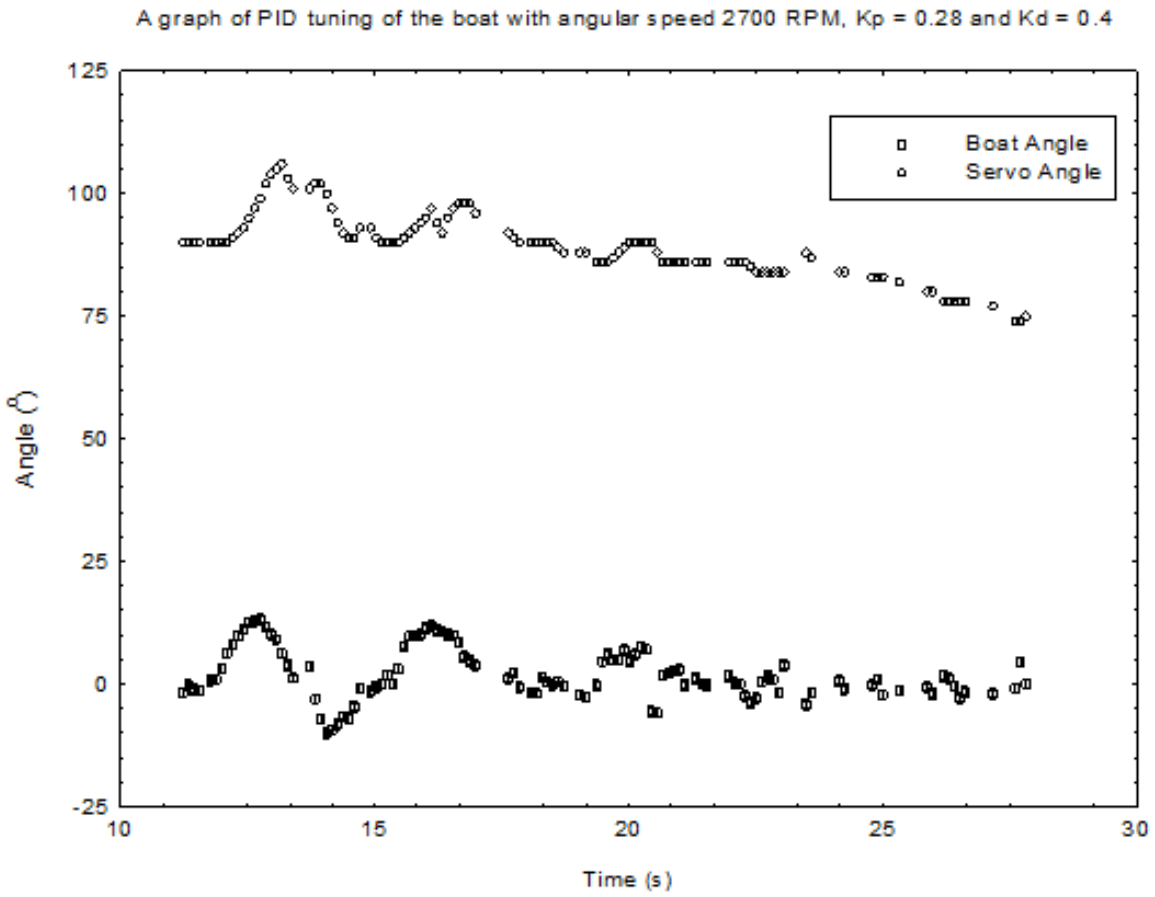


Figure 27: A graph showing the response when k_d was set to 0.4 and k_p was set to 0.28. It can be seen that the boat oscillated several times about the vertical but within 13 seconds reduced the amplitude of these oscillations to below 5° and returned the boat to stability.

In this test k_d was set to 0.4 and k_p was set to 0.28. The boat was manually rotated to 20° and then released. The oscillations of the boat can be seen to reduce over time until it returns to the vertical position after 13 seconds. Figure 27 shows that the PID active system is capable of stopping a boat from oscillating after a simulated wave strike.

The k_p parameter was adjusted to 0.30 and the k_d parameter was kept at 0.4. The boat was then tested again using a simulated wave strike. The response of the boat is shown in figure 28.

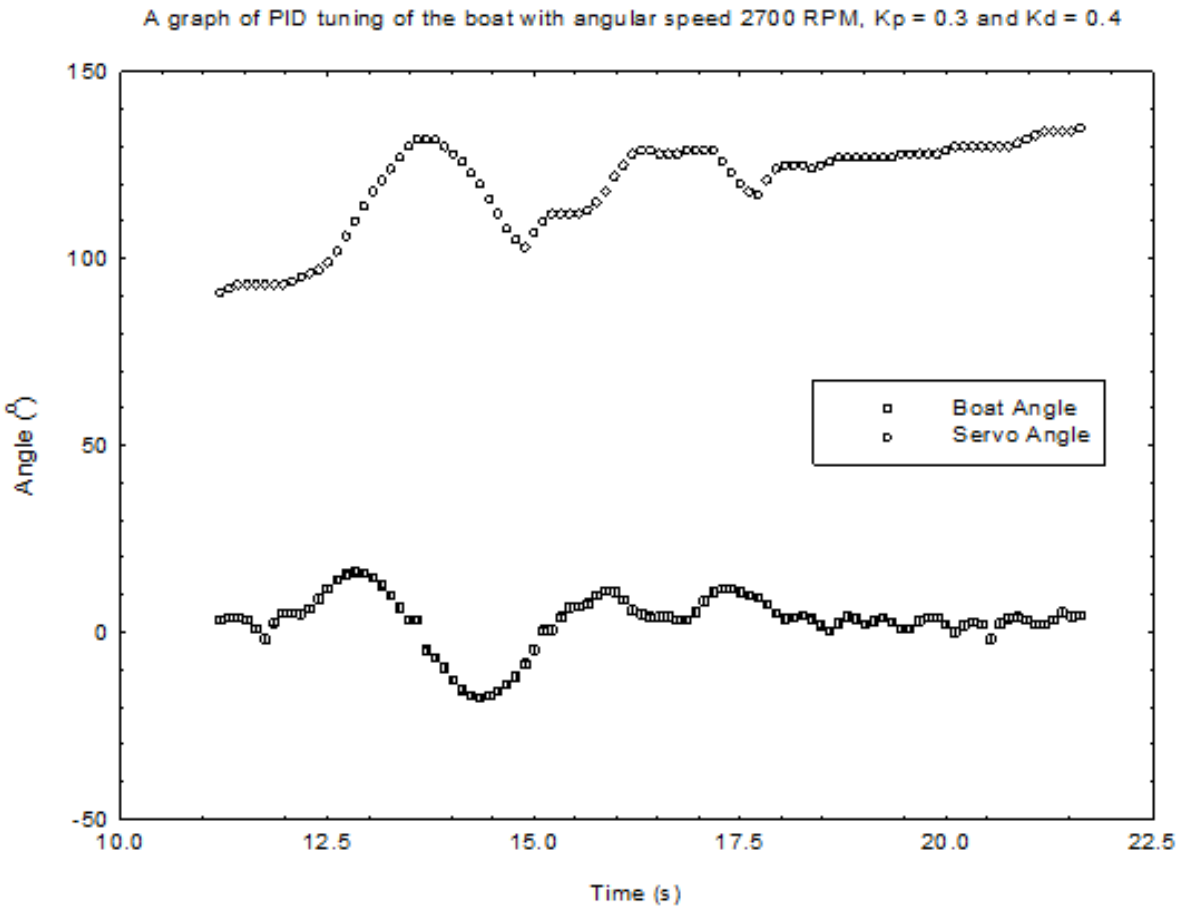


Figure 28: A graph showing the response when k_d was set to 0.4 and k_p was set to 0.3. It can be seen that the boat oscillated several times about the vertical but within 6 seconds reduced the amplitude of these oscillations to below 5° and returned the boat to stability.

The boat was manually rotated to 20° and then released. It can be seen in figure 28 that using these parameters the active system quickly stops the boat oscillating. The active system returns the boat to a stable vertical position after 6 seconds.

Figure 29: A photograph showing the boat recovering from a simulated wave. The servo is moving the flywheel to create a torque to right the boat.

7. Discussion

In the passive system the frame always oscillates about the midpoint because the flywheel always produces more torque than is necessary to balance it. The maximum angle the frame reaches then keeps on increasing until the torque caused by the weight of the motor exceeds the torque the frame can provide and control of the flywheel is lost. It has been shown in figure 18 that the time period of the frame using the passive system increases linearly as the magnitude of the angular momentum increases. Therefore even a flywheel with a large moment of inertia and high angular speed would be expected to make the frame oscillate (albeit at a low frequency). Because of this the passive system would be unsuitable to balance a boat without making it oscillate. Therefore, this system was disregarded as a possible solution.

In the testing of the active system balancing control is lost when the servo reaches the end of its rotation. Therefore to increase the length of time the frame can be balanced for the servo either needs to be changed for a continuous one so it does not have a maximum angle and can rotate continuously or the angle to which the servo moves needs to be kept within its range.

For a continuous servo the software would need to be adapted to take into account the $\cos\theta$ relationship between angular momentum and torque in the y-direction. The servo could then be made to rotate at different speeds to make up for the changing value of θ.

To keep the servo within its range for longer, smaller angles would need to be used. One way smaller angles could be used while still keeping the same amount of torque is to increase the angular momentum of the flywheel. This could be done by increasing the rotational speed of the flywheel, or, increasing its moment of inertia. A second way is to increase the rotational speed of the z-axis rotation. This could be done by gearing the servo.

Whilst using the active system a servo was used to rotate the flywheel about the z-axis. During the testing of the active system on the water the servo was under a high load. After several hours of testing it was noticed that the servo was not responding properly to PID control. After examining the servo it was discovered that the splines that connect the servo to the gears had been sheared off, rendering the servo useless. A solution to this would be to use a more powerful servo which would have stronger splines. Another solution would be to use a stepper motor mounted in line with the bearings to rotate the flywheel about the z-axis. A stepper motor uses a magnetic field to hold a specific angle and the shaft would connect directly to the housing so the splines could not be stripped.

It was observed while testing that the angular velocity of the flywheel fluctuated whilst it was being run. This may have been due to energy being lost from the flywheel to produce the torque and/or the LiPo battery that was powering it not supplying a stable voltage. To solve this problem a feedback loop which monitored the speed of the flywheel and adjusted the PWM signal to the MOSFET was designed. It made use of a light dependant resistor (LDR) in a pair with a fixed resistor attached to an analogue input pin of an Arduino. As the flywheel spun the resistance of the LDR would change (if it was placed in line with the holes) and these fluctuations were then timed and counted to give an angular velocity. This could then be compared with the target angular velocity and the PWM signal adjusted accordingly.

To measure the moment of inertia of the frame string had to be unravelled from the rotary motion sensor and then onto a pulley attached to the frame. This process could have been

improved by mounting the rotary motion sensor directly onto the frame. This way the angular acceleration could be measured directly using the sensor.

Due to engineering difficulties the flywheel that was designed in section 3.2 could not be made. The flywheel used had the same radius but was made of a single thickness of steel (6 mm) and had a moment of inertia of $(2.03 \pm 0.01) \times 10^{-3}$ kg m^2. This was lower than the 2.45×10^{-3} kg m^2 moment of inertia of the designed flywheel. Despite having $(4.2 \pm 0.1) \times 10^{-4}$ kg m^2 less moment of inertia the flywheel that was used performed well. At 2700 RPM it produced enough torque to stabilise the boat using the active system.

The boat that was designed was unstable so the gyroscopic stabilisation could be tested. Due to the mass imbalance of the frame the boat had to be balanced using masses before the testing could begin. Whilst testing it was noticed that the boat had a tendency to rotate about the x-axis. This problem had been predicted and was a consequence of having one flywheel. To stop the boat from rotating in this manner a second flywheel would need to be added and rotated in the opposite direction to the first one about the x-axis. As the flywheels would be rotating in opposite directions (about the x-axis) they would need to be rotated in opposite directions (about the z-axis) so their restoring torques would act in the same direction.

The readings from accelerometer that was used to determine the roll angle of the boat sometimes fluctuated considerably. This may have been because of the high frequency vibrations caused by the motor or due to the instability of the accelerometer chip. These fluctuations meant the accelerometer had to be read a number of times and averaged which used processing power, slowing down the code. These fluctuations also meant the value of the roll angle the PID control system received would sometimes be false, meaning the system would respond to an unphysical stimulus leading to an unnecessary correction. One improvement that could be made is to use an external accelerometer with built in stabilisation and filtering devices. This would allow the PID system to receive a stable value of the frame angle.

The other electrical components in section 4.3 worked as expected. The MOSFET had a maximum operating temperature of 175^{o}C and although it did heat up it only reached 80^o so no additional cooling measures were needed. If the load was increased, by adding another motor for example, additional cooling may need to be considered.

The PID code used to balance the boat worked even though the integral term was ignored. Including the integral term would have allowed the code to properly respond to constant forces such as wind so in the future should be included. The main reason for ignoring the integral term was the unreliability of the accelerometer. As the integral term relies on past data if the accelerometer had given a false reading it would have been carried through for the rest of the test.

PID control uses an “error” measurement to adjust the system to gradually bring it closer to the set point. Due to the ease of implementation a PID control system was chosen for this project. In future work alternative control system may be investigated to find out if there is a more suitable one for this system.

The PID parameters were experimentally determined using the manual tuning method. In this method all of the tuning parameters are set to zero, then k_p is gradually increased until the system oscillates about the set point. k_d is then gradually increased until these oscillations are

damped out. This process is time consuming and may not optimise the system, improvements could be made by using PID tuning software.

Even though the boat was returned to stability after 6 seconds using the tuned PID system it still overshot whilst stabilising, this can be seen if figure 28. This indicates that a more sensitive system, PID or otherwise may be able to further improve the active system.

Tests were conducted in an unstable boat to prove the gyroscopic stabilisation system was capable of balancing and restoring a vessel in challenging conditions. If this system were to be used in a lifeboat it would have to be adapted to work alongside the natural stabilisation of the vessel.

8. Conclusion

The tests that have been conducted have demonstrated that an active system using a flywheel with a moment of inertia of $(2.03 \pm 0.01)\ x10^{-3}$ kg m^2 rotating at 2700 RPM is capable of retuning an unstable boat of mass 5.112 ± 0.001 kg and moment of inertia (0.1225 ± 0.0012) kg m^2 to the vertical position after being struck by a simulated wave. The active system which makes use of a tuned PID control system took 6 seconds to return the boat back to the vertical position from an initial roll angle of 20^o. These test results indicate that it is possible to stabilise an unstable boat, even in the event of an unreasonably large disturbance. However, more work is needed to understand both the secondary effects of the pitching moment and the response of the boat when disturbed by multiple waves. These waves could be in quick succession or from different directions superposing at the hull.

Extensive laboratory attempts were made to balance a frame using the passive system. Although the frame did not immediately fall over, it entered into oscillations about the zero point until the flywheel angle became too large and the frame lost balance. Because of these oscillations the passive system was deemed unsuitable to properly balance a boat.

This project has indicated that it is possible to stabilise an unstable boat using an active PID gyroscopic system. Further tests will need to be conducted to determine the feasibility of fitting a system such as this to a lifeboat, and to further develop the system to be used in real world conditions.

9. Acknowledgments

I would like to thank all of the laboratory technicians for providing the equipment needed to run the tests. I would like to thank Phil Lightfoot for engineering the flywheel, boat, frame and battery holder for use in this project and also for the support given throughout.

10. References

1) F.H. Sellars, J.P. Martim, Selection and evaluation of ship roll stabilization systems Marine Technology, SNAME, 29 (2) (1992),pp. 84–101

2) Lloyd, A. R. J. M., Seakeeping: Ship behaviour in rough weather, (1989), Sussex: Ellis Horwood, 230

3) The Literary gazette, (1852), Issue 1868, pp.827-828

4) Benz, C., (1888), Self-propelling vehicle, Patent US 385,087

5) Crabtree H., An elementary treatment of the theory of Spinning Tops and Gyroscopic Motion, (1909), New York: Longmans, Green, And Co, 70 – 72.

6) Lit Motors Corporation, (2013), Electronic control system for gyroscopic stabilized vehicle, US 8532915 B2

7) Woolsey, C.A., Leonard, N.E., Stabilizing underwater vehicle motion using internal rotors. Automatica 38 (12), (2002), pp. 2053–2062

8) E. A. Sperry, (1908), Ship's gyroscope, US 1150311 A

9) E. A. Sperry, (1915), Ship stabilizing and rolling apparatus, US 1232619 A

10). Arnold, R. N., Maunder, L., Gyrodynamics and its Engineering Applications, (1961), London: Elsevier, 184, 209, 210

11) Gearloose Engineering, Inc., (2003), Gyroscopic roll stabilizer for boats, US6973847 B2

12) Marine Motion Control, LLC., (2007), Gyrostabilizer for small boats, US7240630 B2

13) Fitzpatrick, R., Newtonian Dynamics 4th ed., (2011), [S.l.] : Lulu, 18 & 19

14) Alonso, M & Finn, E. J., Fundamental University Physics, vol 1, (1967), Boston: Addison-Wesley publishing co., 295 & 296

15) http://www.fetcas.com/Services-Control.aspx

16) Dorf, R. C. & Bishop, R. H., Modern control systems, (2008) Upper Saddle River, N.J.: Pearson Prentice Hall, 444

17) http://plato.is/stability_of_fishing_vessels/righting_lever/

18) Molland A. F., The Maritime Engineering Reference Book: A Guide to Ship Design, Construction and Operation, (2008), Oxford: Butterworth-Heinemann, 81

19) STP55NF06L datasheet

11. Appendix A

The full PID code used to control the angle of the flywheels.

```
// PID controller
#include <Servo.h> // includes the servo library
#include <CurieIMU.h> // includes the IMU library
#include <SD.h> // includes the SD card library

const int chipSelect = 10;

Servo SERVOONE; // creates a servo object

int iter = 0; // iteration for array in I section
int servoPos = 90; // starting position for the servo
float Kp = 0.27;// value for proportional tuning parameter
float Ki = 0.00; // value for integral tuning parameter
float Kd = 0.40; // value for differential tuning parameter
float pastAngleErrors [] = { 0.0, 0.0, 0.0, 0.0, 0.0}; // starts an array for 5
sets of data to be stored in
float pastServoMove [] = { 0.0, 0.0, 0.0, 0.0, 0.0}; // starts an array for 5
sets of data to be stored in
float angleError = 0.00;


void setup() {
 CurieIMU.begin (); //intialize the library
 CurieIMU.setAccelerometerRange (2); // sets range of acelerometer in units of g
 CurieIMU.setAccelerometerRate (50); // sets sample rate of 50Hz
 CurieIMU.autoCalibrateAccelerometerOffset ( Z_AXIS, 1);
 CurieIMU.autoCalibrateAccelerometerOffset ( Y_AXIS, 0);// calibrates the
accelerometer
 CurieIMU.setGyroRate (50); // sets sample rate of 50Hz
 CurieIMU.setGyroRange (250); // sets the range up to 250 degrees per second
 CurieIMU.autoCalibrateGyroOffset (); // sets the offset

 SERVOONE.attach (9); // attaches servo to pin 9
 SERVOONE.write (servoPos); // hold the servo steady while motor spins up

 pinMode (0, OUTPUT);
 if (SD.begin (chipSelect)) // initialise library and check SD card
 {
  digitalWrite (0, HIGH); // turns LED on if card is working
 }
 else
 {
  digitalWrite (0, LOW);
 }

}

void loop() {
```

```
  // Proportional code start
    int ax, ay, az; // creates integers for acceleration
  CurieIMU.readAccelerometer (ax, ay, az); //reads raw accelerometer values
  float Accz = (az / 32768.0) * CurieIMU.getAccelerometerRange(); // acceleration
in units of g
  float Accy = (ay / 32768.0) * CurieIMU.getAccelerometerRange(); // y direction
acceleration in units of g
  delay (10);

  int ax1, ay1, az1; // creates integers for acceleration
  CurieIMU.readAccelerometer (ax1, ay1, az1); //reads raw accelerometer values
  float Accz1 = (az1 / 32768.0) * CurieIMU.getAccelerometerRange(); //
acceleration in units of g
  float Accy1 = (ay1 / 32768.0) * CurieIMU.getAccelerometerRange(); // y
direction acceleration in units of g
  delay (10);

  int ax2, ay2, az2; // creates integers for acceleration
  CurieIMU.readAccelerometer (ax2, ay2, az2); //reads raw accelerometer values
  float Accz2 = (az2 / 32768.0) * CurieIMU.getAccelerometerRange(); //
acceleration in units of g
  float Accy2 = (ay2 / 32768.0) * CurieIMU.getAccelerometerRange(); // y
direction acceleration in units of g
  delay (10);

  int ax3, ay3, az3; // creates integers for acceleration
  CurieIMU.readAccelerometer (ax3, ay3, az3); //reads raw accelerometer values
  float Accz3 = (az3 / 32768.0) * CurieIMU.getAccelerometerRange(); //
acceleration in units of g
  float Accy3 = (ay3 / 32768.0) * CurieIMU.getAccelerometerRange(); // y
direction acceleration in units of g
  delay (10);

  int ax4, ay4, az4; // creates integers for acceleration
  CurieIMU.readAccelerometer (ax4, ay4, az4); //reads raw accelerometer values
  float Accz4 = (az4 / 32768.0) * CurieIMU.getAccelerometerRange(); //
acceleration in units of g
  float Accy4 = (ay4 / 32768.0) * CurieIMU.getAccelerometerRange(); // y
direction acceleration in units of g
  delay (10);

  int ax5, ay5, az5; // creates integers for acceleration
  CurieIMU.readAccelerometer (ax5, ay5, az5); //reads raw accelerometer values
  float Accz5 = (az5 / 32768.0) * CurieIMU.getAccelerometerRange(); //
acceleration in units of g
  float Accy5 = (ay5 / 32768.0) * CurieIMU.getAccelerometerRange(); // y
direction acceleration in units of g
  delay (10);
```

```
  int ax6, ay6, az6; // creates integers for acceleration
  CurieIMU.readAccelerometer (ax6, ay6, az6); //reads raw accelerometer values
  float Accz6 = (az6 / 32768.0) * CurieIMU.getAccelerometerRange(); //
acceleration in units of g
  float Accy6 = (ay6 / 32768.0) * CurieIMU.getAccelerometerRange(); // y
direction acceleration in units of g
  delay (10);

  int ax7, ay7, az7; // creates integers for acceleration
  CurieIMU.readAccelerometer (ax7, ay7, az7); //reads raw accelerometer values
  float Accz7 = (az7 / 32768.0) * CurieIMU.getAccelerometerRange(); //
acceleration in units of g
  float Accy7 = (ay7 / 32768.0) * CurieIMU.getAccelerometerRange(); // y
direction acceleration in units of g
  delay (10);

  int ax8, ay8, az8; // creates integers for acceleration
  CurieIMU.readAccelerometer (ax8, ay8, az8); //reads raw accelerometer values
  float Accz8 = (az8 / 32768.0) * CurieIMU.getAccelerometerRange(); //
acceleration in units of g
  float Accy8 = (ay8 / 32768.0) * CurieIMU.getAccelerometerRange(); // y
direction acceleration in units of g
  delay (10);

  int ax9, ay9, az9; // creates integers for acceleration
  CurieIMU.readAccelerometer (ax9, ay9, az9); //reads raw accelerometer values
  float Accz9 = (az9 / 32768.0) * CurieIMU.getAccelerometerRange(); //
acceleration in units of g
  float Accy9 = (ay9 / 32768.0) * CurieIMU.getAccelerometerRange(); // y
direction acceleration in units of g
  // 10 values of the z direction acceleration are taken to smooth out random
fluctuations when averaged

  int signVar = 0; // sets the value of the sign determining variable to zero

  if (Accy < 0.00) signVar++; // if the reading is negative then signVar is
increased by 1
  if (Accy1 < 0.00) signVar++; // if the reading is negative then signVar is
increased by 1
  if (Accy2 < 0.00) signVar++; // if the reading is negative then signVar is
increased by 1
  if (Accy3 < 0.00) signVar++; // if the reading is negative then signVar is
increased by 1
  if (Accy4 < 0.00) signVar++; // if the reading is negative then signVar is
increased by 1
  if (Accy5 < 0.00) signVar++; // if the reading is negative then signVar is
increased by 1
  if (Accy6 < 0.00) signVar++; // if the reading is negative then signVar is
increased by 1
```

```
  if (Accy7 < 0.00) signVar++; // if the reading is negative then signVar is
increased by 1
  if (Accy8 < 0.00) signVar++; // if the reading is negative then signVar is
increased by 1
  if (Accy9 < 0.00) signVar++; // if the reading is negative then signVar is
increased by 1

  float totalAccz = Accz + Accz1 + Accz2 + Accz3 + Accz4 + Accz5 + Accz6 + Accz7
+ Accz8 + Accz9;
  float averageAccz = totalAccz/10.0; // averages over the ten values
  float theta = acos (averageAccz); // determines the angle from the average
acceleration
  float thetaDeg = theta * 57.2958;
  thetaDeg = thetaDeg - 4.5;
  Serial.println (signVar);
  if (signVar > 7) // gives the angle a sign based on which direction it is
tilted in
  {
    thetaDeg = thetaDeg*-1;
  }

  if (thetaDeg < 70.0 && thetaDeg > -70.0)
  {
    angleError = thetaDeg;
  }



  // Proportional code end

  // Integral code starts


  if (iter < 5)
  {
    pastAngleErrors [iter] = angleError; // saves the past 5 angle errors
  }



  float sumAngleError = pastAngleErrors [0] + pastAngleErrors [1] +
pastAngleErrors [2] + pastAngleErrors [3] + pastAngleErrors [4]; // adds the past
errors together
  float averageAngleError = sumAngleError/5.0000; // works out average by
dividing the sum by 5

  // this code should help to overcome a persistent force such as wind

  // Intergral code end
```

```
  // differential code starts

  int gx, gy, gz; // sets integers to read the gyro chip
  CurieIMU.readGyro (gx, gy, gz); // reads gyro chip
  float xRotRate = (gx/32768.9) * CurieIMU.getGyroRange (); // works out the
rotation rate in rad/s
  //if (xRotRate < 0.00) xRotRate = xRotRate * -1; // this takes the modulus of
xRotRate so it doesn't effect the servoMove sign

  // differential code ends



  float servoMove = Kp*angleError + Ki*averageAngleError + Kd*xRotRate; //
multiplies each term be the tuning parameter the adds together

  if (servoPos > 10 && servoPos < 170 && angleError < 70.00 && angleError > -70.
00 && averageAngleError < 70.00 &&
  averageAngleError > -70.00 && xRotRate < 100.00 && xRotRate > -100.00 &&
servoMove < 10.0 && servoMove > -10)
  // if statement to prevent the servo moving pasts its max and min points
  {
    servoPos = servoPos + (int)servoMove; // sets a new value for the servo
position
    SERVOONE.write (servoPos); // writes the servo to new position
  }



  File dataFile = SD.open("angle.txt", FILE_WRITE); // opens a text file called
"angle" on SD card

  if (dataFile) // checks for a malfunction
  {

    dataFile.println (thetaDeg); // print data to text file on SD card
    dataFile.print (","); // seperate the data
    dataFile.print (millis ()); // indicates the time since program started
    dataFile.print (",");
    dataFile.print (servoPos); // print data to SD card
    dataFile.print (","); // serperates the data
    dataFile.close (); // have to close file after writing
  }
  else
  {
    digitalWrite (0, LOW); // indicates if there is an error by turning LED off
  }

  }
```