A FAULT TOLERANT CONTROL APPROACH TO MAGNETIC LEVITATION

MEng Electronic Engineering

First Supervisor: Prof. R.J. Patton
Second Supervisor: Dr. W. Ahmad
i. Abstract

This paper documents an investigation into fault tolerant design in three dimensional magnetic levitation systems. During the project a levitation system utilising magnetic repulsion was designed, mathematically modelled, simulated in Matlab Simulink, built in real life and then programmed using C language. A strong focus was shown on the safety of the system and therefore three types of fault tolerance were investigated and incorporated in the real life device: analytical (mathematical), dynamic (backup) and hardware (comparison) redundancy. It was found that Matlab Simulink could be used to accurately simulate the non-linear and linear representations of the device. From the simulations a PID controller was designed, tuned and used to control the real life operation of the levitation system.
ii. Acknowledgements

First of all I would like to express my appreciation for my project supervisor, Professor Ron Patton, who has inspired and supported me throughout the year. His valuable guidance has been instrumental in the success of the project.

I would also like to thank other staff members of the University of Hull Engineering Department who have helped to inspire me and provide me with a better insight into my project: Dr. Ming Hou and Dr. Kevin Paulson.

My special thanks go out to my girlfriend, Flavia Torres Horta De Araujo, for keeping me focused and motivated even during the hardest stages of the project. I also express my appreciation to all my friends and family who have encouraged me and helped me keep on track.

I would finally like to dedicate this thesis to my grandparents for their financial support as well as encouragement throughout my university education.
### Contents

1.0 Background ........................................................................................................................................... 12
2.0 Introduction / Project Overview ........................................................................................................ 12
3.0 Review of Previous Work ..................................................................................................................... 13
4.0 Aims and Objectives ............................................................................................................................. 15
5.0 Three Dimensional Magnetic Levitation Product Design Specification .............................................. 17
   5.1 Hardware Specification ...................................................................................................................... 17
   5.2 Software Specification ..................................................................................................................... 17
   5.3 System Performance Criteria ......................................................................................................... 17
6.0 System Planning, Design and Concept Development ............................................................................. 18
   6.1 System Architecture ........................................................................................................................ 19
   6.2 Three Dimensional System Concept Art .......................................................................................... 20
   6.3 Feedback Board Design ................................................................................................................... 21
      6.3.1 Sensor Board Schematic ........................................................................................................... 21
   6.4 Fault Tolerance Design .................................................................................................................... 22
      6.4.1 Analytical Redundancy ............................................................................................................. 22
      6.4.2 Dynamic Redundancy ............................................................................................................... 24
      6.4.3 Hardware Redundancy .............................................................................................................. 24
      6.4.4 Software Techniques ................................................................................................................ 25
   6.5 Control System Concept .................................................................................................................. 25
      6.5.1 Levitation System Control ......................................................................................................... 25
      6.5.2 Position Control ......................................................................................................................... 26
      6.5.3 Feedback Controller Type ......................................................................................................... 26
      6.5.4 Fault Tolerant Control ............................................................................................................... 27
      6.5.5 Measuring the Success of the Project ......................................................................................... 27
   6.6 Hardware Requirements .................................................................................................................... 29
   6.7 Software Requirements ..................................................................................................................... 29
   6.8 Estimated Project Cost ....................................................................................................................... 30
7.0 Mathematical Hypothesis ...................................................................................................................... 31
iv. Table of Figures

Figure 1: Sketch of proposed system........................................................................... 18
Figure 2: System architecture......................................................................................... 19
Figure 3: Levitation device concept art.......................................................................... 20
Figure 4: Power drivers and computer control concept art ........................................... 20
Figure 5: Sensor board schematic .................................................................................. 21
Figure 6: Analytical Redundancy (Patton and Chen, 1998, p.5)...................................... 22
Figure 7: System observer scheme (Patton, Frank and Clark, 1989, p.101).................... 23
Figure 8: Hardware Redundancy (Patton and Chen, 1998, p.5).................................24
Figure 9: Levitation control loop..............................................................................25
Figure 10: Transfer function control loop for each electromagnet pair......................26
Figure 11: Fault tolerant control................................................................................27
Figure 12: Ring Magnet Flux Lines (Users-Phys, 2009)..............................................31
Figure 13: Ring Magnet (Magnet Sales & Manufacturing Company, 2000)...............32
Figure 14: Ideal Levitation Forces ............................................................................33
Figure 15: Actual Levitation Forces .........................................................................34
Figure 16: Change in flux density from wire loop....................................................38
Figure 17: Electromagnet pair force components ....................................................40
Figure 18: X-axis forces model................................................................................43
Figure 19: Y-axis forces model................................................................................45
Figure 20: X-axis realisation in Simulink .................................................................49
Figure 21: Electromagnet force Fc1 in Simulink .......................................................50
Figure 22: Electromagnet force Fc2 in Simulink .......................................................50
Figure 23: Ring magnet horizontal force in Simulink...............................................51
Figure 24: Non-linear model in Simulink .................................................................52
Figure 25: Non-linear simulation with PID control ................................................52
Figure 26: State space block diagram.....................................................................56
Figure 27: Root locus of transfer function ...............................................................58
Figure 28: Open loop response................................................................................60
Figure 29: Proportional gain control ......................................................................61
Figure 30: Pole placement simulation for poles at -8 and -10..................................63
Figure 31: Pole placement at -80 and -100..............................................................64
Figure 32: Block diagram for PID control ...............................................................65
Figure 33: PID control with unstable Kp, Ki and Kd values....................................69
Figure 34: PID control with stable Kp, Ki and Kd values........................................69
Figure 35: Output oscillation of constant amplitude...............................................70
Figure 36: PID control with gain values from Ziegler-Nichols calculations ........................................71
Figure 37: PID controller in Matlab Simulink ...............................................................................................72
Figure 38: PID simulation in Simulink with Kp = 1.8; Ki = 11.25; Kd = 0.072 ........................................73
Figure 39: PID controller simulink simulation with Kp = 2; Ki = 0.22; Kd = 7.69 ........................................73
Figure 40: Levitation plant signal flow ..........................................................................................................74
Figure 41: Observer flow diagram ................................................................................................................74
Figure 42: Observer simulation with poles at -20 and -30 ........................................................................77
Figure 43: Observer simulation with poles at -60 and -80 ........................................................................77
Figure 44: Agile waterfall approach to software design ..............................................................................79
Figure 45: Feedback prototype board .........................................................................................................80
Figure 46: Feedback board PCB design .....................................................................................................80
Figure 47: Three dimensional levitation system prototype V.1 .................................................................81
Figure 48: Prototype v.1 circuit diagram ......................................................................................................82
Figure 49: PID software flow chart ...........................................................................................................83
Figure 50: Validation of predicted unstable system .................................................................................86
Figure 51: Validation of predicted stable system .......................................................................................87
Figure 52: Validation of predicted stable system with nudges .................................................................88
Figure 53: Dynamic redundancy operation ...............................................................................................89
Figure 54: Position tracking when a primary sensor fails .......................................................................90
Figure 55: Three Dimensional Levitation Device Prototype V.3 – Hardware Redundancy ..................92
Figure 56: Prototype v.3 schematic .........................................................................................................93
Figure 57: SPI master Tx version 1.0 ........................................................................................................94
Figure 58: SPI slave testing .....................................................................................................................95
Figure 59: SPI Arduino receiver version 2.0 ...........................................................................................97
Figure 60: Arduino master v.2 flowchart ................................................................................................98
Figure 61: Temperature control circuit diagram ....................................................................................101
Figure 62: Temperature control circuit ...................................................................................................101
Figure 63: Temperature monitoring flowchart .......................................................................................102
Figure 64: Project Gantt Chart version 1.0 ................................................................. 106
Figure 65: Gantt chart version 2.0 ............................................................................. 107
Figure 66: Gantt chart version 3.0 ............................................................................. 109

v. Table of Tables

Table 1: Aims and objectives ......................................................................................... 15
Table 2: Hardware specification ....................................................................................... 17
Table 3: Software specification ....................................................................................... 17
Table 4: System performance criteria ............................................................................... 17
Table 5: Measurements of Success ................................................................................ 27
Table 6: Project cost .......................................................................................................... 30
Table 7: Routh-Hurwitz array ........................................................................................ 67
Table 8: Routh-Hurwitz array with substituted values ................................................... 68
Table 9: Testing of the PID controller ............................................................................. 85
Table 10: Dynamic redundancy software testing .......................................................... 90
Table 11: SPI software version 1.0 testing ................................................................. 96
Table 12: SPI Software Version 2.0 Testing ............................................................... 99
Table 13: Combining hardware and dynamic redundancy ........................................... 99
Table 14: Temperature monitoring circuit testing ......................................................... 102
Table 15: Milestone Deadlines ...................................................................................... 105
Table 16: Prototype hardware specification review ...................................................... 114
Table 17: Prototype software specification review ....................................................... 115
Table 18: Ring magnet parameters ............................................................................... 121
Table 19: Prototype v.1 electromagnet parameters ....................................................... 121
1.0 Background

Magnetic levitation is a process that describes an object being lifted from the ground using repulsive magnetic fields (Online Dictionary, 2013). Many systems around the world currently use magnetic levitation technology. One example would be Maglev trains, which can reach incredibly high speeds. Such a system is responsible for safely transporting hundreds of passengers each day. Because of the large number of passengers and the high speeds, failure of this system would be catastrophic. Therefore safety is of vital importance.

Earnshaw (1842, pp. 97-112) explains that magnetic levitation by repulsion is a process that is naturally unstable. As this technology is often used to levitate extremely heavy loads into the air, if a control system is not used to correct the position of the object under levitation, the system is unsafe and people could be harmed. If a control system is designed correctly, stable levitation can be achieved and safety can be increased.

However, even a perfectly controlled system could still fail if faults are produced that cannot be detected. A fault in either hardware or software will cause an error, and if the error is allowed to manifest then the system will fail. Fault diagnosis, and tolerant design approaches, could be used to stop the system from failing completely in the event of a fault.

Fault diagnosis is a concept that is employed in all major safety-critical systems such as aircraft and nuclear power plants. The outcome of a fault developing in any safety-critical system can be 'extremely serious' for 'human mortality, the environment and the economy' (Patton and Chen, 1998, p.1).

2.0 Introduction / Project Overview

This project details the development of a complex fault tolerant magnetic levitation system. The levitation system itself will be fully designed and produced, then used as an application to demonstrate fault diagnosis techniques, and fault tolerant control.

Levitation will be achieved using a static ring magnet to provide vertical lift in the z plane, two pairs of electromagnets to provide horizontal forces in the x and y planes, and a pair of sensors to provide x and y plane position feedback. Firstly, the levitation system will be designed with redundancy, which is used to detect errors in the sensor readings. The system will then be mathematically modelled and simulated. The design of the system will then be developed further to incorporate several complex fault tolerant approaches, before being built and programmed.
3.0 Review of Previous Work

Almost all magnetic levitation systems are suspension based because they are designed to use magnetic attraction forces. This type of magnetic levitation system can be considered to be a one-dimensional system: the only dimension that must be controlled is the vertical plane. As the levitating magnet drops in height, power is applied to a single electromagnet placed above the levitating magnet which pulls it back up towards a set point.

Chen (2010) and Baranowski (2006) both carried out final year projects working with this type of magnetic levitation system. One-dimensional levitation devices, however, have two major drawbacks: there must always be an electromagnet placed above the levitating magnet, which takes up space, and the device cannot hold a heavy weight without the electromagnet being incredibly large.

Instead of using magnetic attraction, like Chen, Baranowski, the device in this project utilises magnetic repulsion forces and can therefore be considered as a three-dimensional levitation system. The vertical (z), horizontal (x) and horizontal (y) planes must all be controlled separately to achieve stable levitation.

Williams (2005) describes in his thesis how he attempted to design a levitation system using repulsive forces in order to levitate a bar magnet. However, his design consisted only of electromagnets and had no position control. The idea was that the electromagnets were arranged in a formation such that the levitating magnet became ‘trapped’ in space. Because position feedback was not used, and no method was applied to control vertical motion, the conclusion of the project was that ‘stable levitation could not be fully achieved’.

The three-dimensional magnetic levitation system proposed in this project is novel because, by using a permanent ring magnet to control the vertical position of the levitating magnet and four electromagnets to control both horizontal planes, it should overcome all of the problems described by Williams.

The four electromagnets will be split into two pairs, each given a separate horizontal plane to control. The height of the ring magnet will be adjustable so that the levitation height can be controlled for levitating magnets of any weight. Instead of trapping the levitating magnet as chosen by Williams, the levitating magnet will be allowed to freely move away from a centre point. As this happens a feedback controller for each horizontal axis will apply appropriate power to each pair of coils, therefore bringing the levitating magnet back to the centre point.

Zhang et al. (2007) developed a fault tolerant controller for a magnetic suspension system. They developed the conventional PID controller to include fault tolerant techniques. However, if the central processing unit that realises their controller fails then the system will fail immediately.
The fault tolerant design in this project will include multiple central processing units all calculating the output of a controller simultaneously. These calculations will then be compared and logic will decide which processor should be used.

Nazari et al. (2010) developed an observer based control system for a magnetic suspension system. The system included 2 sensors and could detect a faulty reading. However, the magnetic repulsion system in this project will require 2 sensors alone to each of the horizontal axis. Therefore observer based error detection will be designed around 4 sensors instead of just 2.

Because suspension systems only control the levitation position in one dimension, with a single controller, the switch between active sensors will not cause any disturbance to the levitating magnet. However, for a three-dimensional system, it is predicted that the switch over between active sensors will cause a disturbance because of the high levels of positive flux produced by the ring magnet. Therefore the controller designed in this project will also include a routine to adjust the set points of the controllers and boost the gain temporarily in order to compensate for the sudden disturbance caused by the sensor switchover.

The magnetic levitation system proposed in this project is unique because it demonstrates how three complex types of fault tolerant design, all operating simultaneously, can be applied to a three-dimensional levitation system using magnetic repulsion. The design will also include additional software redundancy and stabilisation routines that will aid the fault tolerant techniques.
4.0 Aims and Objectives

The aims for this project describe the overall desired achievements upon completion. The objectives identify a list of smaller accomplishments that need to be made in order to achieve each of the aims.

Table 1 displays the aims of the project, and which objectives need to be completed in order to achieve them.

<table>
<thead>
<tr>
<th>Aims</th>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a.1) Achieve stable magnetic levitation of an object using magnetic repulsion.</td>
<td>(ob.1) Produce a mathematical model of the levitation system</td>
</tr>
<tr>
<td></td>
<td>(ob.2) Produce a software model of the system, and use it to produce simulations</td>
</tr>
<tr>
<td></td>
<td>(ob.3) Design and build a physical structure that can be used to levitate an object</td>
</tr>
<tr>
<td></td>
<td>(ob.4) Design a PID controller in C computer language</td>
</tr>
<tr>
<td></td>
<td>(ob.5) Program the PID controller onto a microprocessor and wire up the system</td>
</tr>
<tr>
<td>Aims</td>
<td>Objectives</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>(a.2) Detect any numerical errors provided by the sensor feedback</td>
<td>(ob.6) Research how observers can be used in control systems to monitor data and detect errors</td>
</tr>
<tr>
<td></td>
<td>(ob.7) Incorporate the observers into the software model and simulate</td>
</tr>
<tr>
<td></td>
<td>(ob.8) Design four observers in C language, and program them onto a microprocessor</td>
</tr>
<tr>
<td>(a.3) Produce a fault tolerant system, that will continue to operate when a variety of different faults occur</td>
<td>(ob.9) Research and implement static redundancy</td>
</tr>
<tr>
<td></td>
<td>(ob.10) Research dynamic redundancy</td>
</tr>
<tr>
<td></td>
<td>(ob.11) Research software error masking</td>
</tr>
<tr>
<td></td>
<td>(ob.12) Implement temperature monitoring</td>
</tr>
</tbody>
</table>

The overall desired achievement at the end of the project is to have produced a digitally controlled fault tolerant system, which can levitate an object using magnetic repulsion undisturbed by system faults.
5.0 Three Dimensional Magnetic Levitation Product Design Specification

5.1 Hardware Specification

The hardware specification of the system is shown in Table 2:

<table>
<thead>
<tr>
<th>Specification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Redundancy</td>
<td>Additional “backup” sensors that will be used as soon as the primary pair of sensors fail.</td>
</tr>
<tr>
<td>Hardware Redundancy</td>
<td>At least one additional microprocessor for calculation comparison and error masking.</td>
</tr>
<tr>
<td>Temperature Control</td>
<td>The system should operate between +5 and +60 Degrees C.</td>
</tr>
</tbody>
</table>

5.2 Software Specification

The software specification of the system is shown in Table 3.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levitation Control</td>
<td>A controller must be designed in software to read the feedback sensors and adjust the power to the electromagnets as appropriate to achieve stable levitation.</td>
</tr>
<tr>
<td>Dynamic Redundancy</td>
<td>Software should be developed that can switch between sensor pair programs.</td>
</tr>
<tr>
<td>Hardware Redundancy</td>
<td>SPI should be integrated into the controller design to compare control values.</td>
</tr>
<tr>
<td>Analytical Redundancy</td>
<td>Accomplished by realising a group of observers within a microprocessing unit.</td>
</tr>
</tbody>
</table>

5.3 System Performance Criteria

The system performance criteria (Table 4) is of great importance for simulation analysis.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steady-state error</td>
<td>$e(\infty) = 0$</td>
</tr>
<tr>
<td>10% Settling time</td>
<td>$T_s \leq 0.2s$</td>
</tr>
<tr>
<td>Overshoot</td>
<td>$O_p \leq 20%$</td>
</tr>
</tbody>
</table>
6.0 System Planning, Design and Concept Development

Figure 1 illustrates a basic bird’s eye view sketch of the proposed levitation system.

Figure 1 shows two electromagnet pairs. One pair will be driven by driver X and the other by driver Y. The sketch shows four Hall Effect sensors located in the centre of all four electromagnets. A magnet will levitate in the centre of the system, just above the circuit board in-between the four Hall Effect sensors.

The permanent ring magnet will be fixed in place and will provide a force in the vertical plane (z-axis) equal to gravity and therefore keep the levitating magnet (LM) at a constant height. The ring magnet will also however produce a horizontal force component which will displace the LM from the centre point. The two electromagnet pairs will provide force in the horizontal plane to counteract the unwanted ring magnet horizontal force component. In both electromagnet pairs, each of the two electromagnets will be wired in parallel and in opposite directions. Therefore the two electromagnets will always have opposite poles. Because of the wiring when current is applied to the pair, each electromagnet will have opposite poles, and therefore one will attract the levitating magnet while the other repels it.

When the system is started, only X+ and Y+ sensors will be used. When an error is detected in the original sensor pair, the computer control system will switch over to using sensors X- and Y-.

The computer control system will consist of a number of microprocessors which will be programmed to perform several control algorithms. These algorithms will include feedback controllers and sensor observers.
6.1 System Architecture

Figure 2 illustrates a system architecture consisting of 3 levels. Level 1 shows the high level design, and the three major sections communicate with each other. Level 2 shows which project aim is being accomplished by having each of the major sections in level 1. Finally level 3 shows a component breakdown of the sections in level 1. The green arrows between the three sections show the direction of data flow.

It is important to include the project aims from section 3.0 in the architecture. This shows which sections of the architecture are expected to achieve which aims at the end of the project.
6.2 Three Dimensional System Concept Art

Concept art is made from essential illustrations of the proposed design. The levitation system is made from three sections: repulsion device, power drivers and computer control. The concept art of the proposed repulsion device is shown in Figure 3:

![Figure 3: Levitation device concept art](image)

Figure 3 illustrates the proposed layout of the device's components and will be used to make the first device prototype. The power drivers and computer control (excluding hardware redundancy) are shown in Figure 4:

![Figure 4: Power drivers and computer control concept art](image)
6.3 Feedback Board Design

The sensor board is responsible for detecting the position of the levitating magnet (LM) and then providing feedback to the computer control system. To achieve dynamic redundancy the board must contain a spare pair of sensors. These must be ready to provide feedback to the computer control at any time during levitation.

6.3.1 Sensor Board Schematic

A schematic for the feedback board was designed using Eagle software (Figure 5). A standard grid of wire pads were used for the connections between the board and the signal wires so that a connector could be used at a later date if necessary.

![Sensor board schematic](image)

Figure 5: Sensor board schematic

The schematic in Figure 5 shows the four Hall Effect sensors being connected to a positive voltage supply (VCC), ground (GND) and a signal out wire pad. A $1\mu$F decoupling capacitor is connected between VCC and GND to smooth the voltage input to the sensors.
6.4 Fault Tolerance Design

A fault diagnosis system is described by Patton and Chen (1998, pp.2-3) as a system that usually executes fault detection, isolation and identification. Fault detection is described by Patton and Chen as 'an absolute must' for practical systems; without fault detection the fault diagnosis would be useless. The fault isolation is described as slightly less important, but still necessary, whereas the fault identification task (i.e. estimating the magnitude, or type of fault) is not essential.

There are two major types of fault detection and isolation (FDI): analytical redundancy, and hardware redundancy. FDI will be implemented using both of these methods to detect errors in different sections of the levitation system.

6.4.1 Analytical Redundancy

Analytical redundancy will be used to monitor and detect errors in the sensor readings. A mathematical model-based FDI algorithm will be implemented in the form of an observer. An observer will be assigned to each sensor, and mathematically predict the desired output. If the measured output is different, then an error has occurred and a fault has been detected.

![Diagram of Analytical Redundancy](image)

Figure 6: Analytical Redundancy (Patton and Chen, 1998, p.5).

Figure 6 shows how analytical redundancy makes use of a model-based FDI algorithm. This algorithm will eventually be programmed in software and loaded onto the Central Processing Unit (CPU).
Patton, Frank and Clark (1989, p.99) describe in their book "Fault Diagnosis in Dynamic Systems" how a group of observers can be used to isolate faults in an FDI system using the "Generalised Observer Scheme (GOS)". The GOS scheme will be applied to the levitation system by using four observers, each driven by a different set of data (provided by each of the four Hall Effect sensors). There will be two sets of two observers: set 1 (X+, Y+) and set 2 (X-, Y-). Figure 7 shows how the bank of observers will be driven by the sensors.

Once a fault is detected by the comparison logic, the fault flag will be set. The computer control will then immediately utilise the system's dynamic redundancy, therefore recovering the system's performance before total failure occurs.
6.4.2 Dynamic Redundancy

Dynamic redundancy utilises additional components which are ready to be used when a faulty component is detected.

Two additional Hall Effect sensors will be integrated into the levitation system, and remain permanently on. When an error is detected in the original pair of sensors, the system will switch over to using the extra pair of sensors to provide feedback and prevent failure. When the system switches sensor pairs, the software will have to boost the gain temporarily to compensate for the switching time delay.

6.4.3 Hardware Redundancy

A microcontroller will be used to implement the controller for the system. The controller will realise a feedback controller, written in the C programming language.

Hardware redundancy combined with a voting system (diagnostic logic) will be used to monitor, detect and mask errors produced by the microprocessor. As it will be directly responsible for controlling the actuators of the system, it is important that the risk of microprocessor failure is considerably reduced.

Figure 8 illustrates how hardware redundancy uses additional microprocessors in order to identify a fault. This method will add large costs to the design, but will significantly reduce the risk of a failure.

![Figure 8: Hardware Redundancy (Patton and Chen, 1998, p.5).](image)

The diagnostic logic block in Figure 8 represents a voting system. All available processors will carry out a control algorithm and produce an 8-bit value which will be used to create a PWM signal for the electromagnets. The diagnostic logic will compare each of the received 8-bit values and determine the majority value.

This method will only mask a faulty processor, not remove it. However, for the system to fail, all additional processors must also fail. It is assumed that each processor will fail individually, therefore the likelihood of system failure is almost completely removed.
6.4.4 Software Techniques

Additional fault tolerant techniques will be implemented in the software. For example, multiple readings of each sensor will be taken and an algorithm will be written to remove any anomalous readings caused by noise or early signs of failure.

6.5 Control System Concept

6.5.1 Levitation System Control

The magnetic levitation system in Figure 47 incorporates two identical control loops, one for the x-axis position of the levitating object and another for the y-axis.

Each control loop (Figure 9) is made from a controller (computer control via the Arduino in Figure 47), a pair of actuators and sensor feedback.

Figure 9: Levitation control loop
The actuators will be an electromagnet pair and the feedback will be provided by Hall Effect sensors. The transfer function block diagram for the control of each electromagnet pair is shown in Figure 10.

![Figure 10: Transfer function control loop for each electromagnet pair](image)

### 6.5.2 Position Control

The idea of the control system in Figure 10 is to control the position of the LM. The desired position set point in Figure 10 can be denoted as \( x_r \). The desired feedback control will converge the current position, \( x \), of the LM to be \( x_r \). This is described as: \( x \to x_r \) as \( t \to \infty \) where \( t \) describes time (Hou, 2010). Three potential control types for this system are: proportional gain; pole placement and PID control.

### 6.5.3 Feedback Controller Type

The proportional gain controller multiplies the feedback by a constant gain. Therefore the increase in feedback is proportional to the position error of the LM. This controller is simple and may not work. Analysis of the system’s poles will reveal the suitability of this controller.

The pole-placement controller uses mathematics to re-position the poles of the closed loop system. This type of controller can be easily calculated using mathematical techniques. Simulations can be used to determine the stability, and response, of the system utilising this type of feedback control.

A PID controller compares the error produced by taking the position of the LM away from the defined position set point. The controller then uses the error to produce three gain values. These values are used to change the magnitude of the output power to go through the electromagnets. The three gain types are: proportional, differential and integral. The transfer function of a PID controller in the s domain is: \( K_p + \frac{K_i}{s} + K_d s \) (Control Tutorials for Matlab, 2012).
6.5.4 Fault Tolerant Control

Figure 11 utilises the fault tolerant design techniques to illustrate the proposed fault tolerant control loop for the system.

The control loop in Figure 11 shows that analytical, hardware and dynamic redundancy will be used in the proposed system.

6.5.5 Measuring the Success of the Project

Upon completion of the project, the success of the system will be measured by how many of the project aims (Table 1) have been achieved. Table 5 describes how the success of each aim will be measured.

<table>
<thead>
<tr>
<th>Aim</th>
<th>How the success will be measured</th>
</tr>
</thead>
<tbody>
<tr>
<td>Achieve stable magnetic levitation of an object using magnetic repulsion</td>
<td>If a magnet can be placed in the centre of the electromagnets and released without shooting away from the device magnetic levitation has been achieved.</td>
</tr>
<tr>
<td>Aim</td>
<td>How the success will be measured</td>
</tr>
<tr>
<td>--------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Achieve stable magnetic levitation of an object using magnetic repulsion (<em>continued</em>)</td>
<td>If the magnet can be nudged gently into an unstable oscillation, and then proceed to stabilise itself then stable levitation has been achieved.</td>
</tr>
<tr>
<td>Detect any numerical errors provided by the sensor feedback</td>
<td>The success of the sensor error detection is determined by how accurately the system can predict the position of the levitating object.</td>
</tr>
<tr>
<td></td>
<td>If the system can accurately predict the location of the levitating object in the x and y-axis, and therefore flag any data error, this aim was a success.</td>
</tr>
<tr>
<td>Produce a fault tolerant system that can continue to operate when faults occur</td>
<td>A testing board will be produced, where buttons can be pressed to introduce a variety of different faults into the system.</td>
</tr>
<tr>
<td></td>
<td>If the system can continue to operate without failing after each fault is introduced then the fault tolerant design was fully successful.</td>
</tr>
<tr>
<td></td>
<td>The overall success of this aim is determined by how many faults can be introduced without the system failing.</td>
</tr>
</tbody>
</table>
6.6 Hardware Requirements

- **Arduino Mega 2560 microprocessor**: This microprocessor will be used as the computer control. This processor has a large range of inputs and outputs (I/O) and has the ability to use serial data transfer, Analogue to digital converter (ADC) and pulse width modulation outputs (PWM). The processor has a clock speed of 16Mhz which should be fast enough to run the control software.

- **Hall effect sensors (ss495A)**: The hall effect sensor detects magnetic field strength. These will be used to locate the levitating magnet. These specific sensors were chosen because they have a relatively large change in output for a small change in magnetic field.

- **Electromagnets**: The electromagnets will be used to repel or attract the levitating magnet in the horizontal plane (x-axis and y-axis).

- **Static Ring magnet**: A permanent ring magnet is required to provide the lift for the levitating magnet in the vertical plane (z-axis). The ring magnet can be either a one-piece magnet or several small magnets arranged in the shape of a ring.

6.7 Software Requirements

- **Matlab Simulink simulation software**: Matlab Simulink allows the user to design control circuits and run simulations.

- **Matlab software**: Complex mathematical functions can be solved and modelled in this software.

- **Arduino compiler**: This compiler allows software to be written and downloaded onto the Arduino microprocessor.
### 6.8 Estimated Project Cost

The project costs, for both hardware and software, are detailed in Table 6.

Table 6: Project cost

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Component Cost (£)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware</strong></td>
<td></td>
</tr>
<tr>
<td>Ring magnets</td>
<td>40.00</td>
</tr>
<tr>
<td>Wire for the electromagnets</td>
<td>10.00</td>
</tr>
<tr>
<td>Hall Effect sensor X4</td>
<td>10.00</td>
</tr>
<tr>
<td>Arduino Mega microprocessor X2</td>
<td>60.00</td>
</tr>
<tr>
<td>Wooden boards</td>
<td>5.00</td>
</tr>
<tr>
<td>Vero board</td>
<td>5.00</td>
</tr>
<tr>
<td>Electromagnet driver X2</td>
<td>10.00</td>
</tr>
<tr>
<td>Levitating Magnet</td>
<td>10.00</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>25.00</td>
</tr>
<tr>
<td><strong>Total Hardware Cost:</strong></td>
<td>£175.00</td>
</tr>
<tr>
<td><strong>Software</strong></td>
<td></td>
</tr>
<tr>
<td>Matlab Simulink</td>
<td>N/A *</td>
</tr>
<tr>
<td>Arduino compiler</td>
<td>N/A **</td>
</tr>
<tr>
<td><strong>Total Software Cost:</strong></td>
<td>0.00</td>
</tr>
<tr>
<td><strong>Total Project Cost:</strong></td>
<td>£175.00</td>
</tr>
</tbody>
</table>

* University of Hull already has access to this software

** This software is free to download
7.0 Mathematical Hypothesis


The three dimensional magnetic levitation device generates a number of forces in the in the x, y and z axis (Earnshaw 1842), which are felt by a levitating magnet (LM). These forces, as well as external forces acting on the LM need to be illustrated and then modeled mathematically. The three sections of the device that generate force are the: ring magnet, electromagnet pair X and electromagnet pair Y.

7.1 Ring Magnet Force Components Model

A ring magnet is used to provide the vertical (z-axis) force, which is equal to the force of gravity under levitating conditions. However, the ring magnet also produces an unwanted horizontal plane force component due to the direction of the change in flux (Earnshaw 1842). The flux lines produced by the ring magnet are shown in Figure 12.

Figure 12 also shows the flux divergence of a ring magnet. This divergence of flux is the reason why a LM with finite dimensions and mass will always feel a horizontal component of force, as described by Earnshaw (1842).
The ring magnet to be used is made from a grade 35 alloy of Neodymium, Iron and Boron (NdFeB) known as a Neodymium magnet (Magnet Sales & Manufacturing Company, 2000). This material has a residual flux density (Br) of 1.23T. The flux density (B) at a distance d from the centre of a Neodymium ring magnet is illustrated in Figure 13.

![Figure 13: Ring Magnet (Magnet Sales & Manufacturing Company, 2000)](image)

According to the Magnet Sales & Manufacturing Company (2000) the flux density at distance d away from the centre of the ring magnet can be calculated using Equation [1]:

\[
B = \frac{B_r}{2} \left( \left( \frac{L + d}{\sqrt{R^2 + (L + d)^2}} \right) - \left( \frac{L + d}{\sqrt{r^2 + (L + d)^2}} \right) \right) - \left( \frac{d}{\sqrt{R^2 + d^2}} \right) - \left( \frac{d}{\sqrt{r^2 + d^2}} \right)
\]

Equation [1]

Substituting the ring magnet parameters in Table 20 (page 121) into Equation [1] derives:

\[
B = 0.615 \left( \left( \frac{30 \times 10^{-3} + d}{\sqrt{0.0169 + (30 \times 10^{-3} + d)^2}} \right) - \left( \frac{30 \times 10^{-3} + d}{\sqrt{0.0121 + (30 \times 10^{-3} + d)^2}} \right) \right) - \left( \frac{d}{\sqrt{0.0169 + d^2}} \right) - \left( \frac{d}{\sqrt{0.0121 + d^2}} \right)
\]

Equation [2]

The force felt by the LM from the ring magnet can be calculated using Equation [3] (François, 2008):

\[
F = \frac{B^2 A}{2 \mu_0}
\]

Equation [3]

Where: 'A' represents the pole area and '\( \mu_0 \)' is the permeability of free space.
In a perfect world the LM would be centralised directly above the ring magnet. In this case the horizontal force components of the ring magnet would cancel each other out (Jayawant, 1981) and therefore the LM would experience purely vertical force components as shown in Figure 14.

The force acting downwards in Figure 14 (Fg) is the result of gravity and the force acting upwards (Fr) is the vertical force component of the ring magnet (Equation [3]).

The force Fr generated from the ring magnet can be derived as a function of distance d by substituting Equation [2] into Equation [3] and using component parameters specified in Table 20 (page 121):

\[
F_r(d) = \frac{\left(\frac{30 \times 10^{-3} + d}{\sqrt{0.0169 + (30 \times 10^{-3} + d)^2}}\right) \left(\frac{30 \times 10^{-3} + d}{\sqrt{0.0121 + (30 \times 10^{-3} + d)^2}}\right)}{1.084 \times 10^{-3}}
\]  
Equation [4]

However, because of the diverging flux illustrated in Figure 12, the LM cannot remain centralised in the horizontal plane. Figure 15 shows the real forces acting on the magnet.
Figure 15: Actual Levitation Forces

where:

\( F_r \) = Ring magnet force

\( F_g \) = Force due to mass multiplied by gravity

\( F_{rv} \) = Ring magnet vertical force component

\( F_{rh} \) = Ring magnet horizontal force component

\( x \) = Displacement from the center

\( h \) = LM height from the center

\( d \) = Distance from center of the ring

Because the LM is not an infinitesimal point there are always horizontal and vertical force components acting on it (Earnshaw 1842). Figure 15 shows how these components are dependent on both the ring magnet force \( F_r \) and the angle \( \theta \) (Jayawant, 1981).

The ring magnet force produced by a change in flux \( F_r \), described by Equation [4], is used to determine the magnitude of the force felt by the LM at displacement \( x \). The distance \( d \) shown in Figure 15 can be described using Equation [5]:

[Diagram of ring magnet forces with labels for \( f_r \), \( f_{rv} \), \( f_{rh} \), and \( f_s \) forces, \( h \), \( d \), \( x \), and \( \theta \) shown as per figure description.]
\[ d = (x^2 + h^2)^{\frac{1}{2}} \quad \text{Equation [5]} \]

Therefore substituting Equation [5] into Equation [4]:

\[ F_r(x) = \frac{1}{1.084 \times 10^{-3}} \left( \left( \frac{30 \times 10^{-3} + (x^2 + h^2)^{\frac{1}{2}}}{0.0169 + (30 \times 10^{-3} + (x^2 + h^2)^{\frac{1}{2}})^2} \right) - \left( \frac{30 \times 10^{-3} + (x^2 + h^2)^{\frac{1}{2}}}{0.0121 + (30 \times 10^{-3} + (x^2 + h^2)^{\frac{1}{2}})^2} \right) \right)^2 \]

Equation [6]

A function for \( \vartheta \) is derived as follows:

\[ \vartheta = \sin^{-1} \left( \frac{h}{d} \right) \quad \text{Equation [7]} \]

where:

\[ d = (x^2 + h^2)^{\frac{1}{2}} \quad \text{Equation [8]} \]

Therefore substituting Equation [7] into Equation [6]:

\[ F_\vartheta(h, x) = \sin^{-1} \left( \frac{h}{(x^2 + h^2)^{\frac{1}{2}}} \right) \quad \text{Equation [9]} \]

where:

\[ F_\vartheta = \text{Function of angle } \vartheta \text{ in degrees.} \]

The vertical force acting upwards on the LM is derived to be equation [11]:

\[ \sin \vartheta = \frac{F_{rv}}{F_r} \quad \text{Equation [10]} \]

\[ F_{rv} = F_r(x) \sin \vartheta \quad \text{Equation [11]} \]

\[ F_{rv}(x) = \frac{1}{1.084 \times 10^{-3}} \left( \left( \frac{30 \times 10^{-3} + (x^2 + h^2)^{\frac{1}{2}}}{0.0169 + (30 \times 10^{-3} + (x^2 + h^2)^{\frac{1}{2}})^2} \right) - \left( \frac{30 \times 10^{-3} + (x^2 + h^2)^{\frac{1}{2}}}{0.0121 + (30 \times 10^{-3} + (x^2 + h^2)^{\frac{1}{2}})^2} \right) \right)^2 \sin \left( \sin^{-1} \left( \frac{h}{(x^2 + h^2)^{\frac{1}{2}}} \right) \right) \]

Equation [12]

The second vertical force in Figure 15 is the force of gravity, where:
\[ F_g = mg \quad \text{Equation [13]} \]

As the LM is displaced further from the centre, the vertical lift produced by the ring will decrease. This will result in the LM obtaining a downwards velocity because the force of gravity will be greater than the upwards lift. As the LM falls the distance \( d \) will decrease, therefore increasing the vertical lift. When the upwards force is once again equal to gravity the LM will stop falling.

The control system for this device will monitor the displacement of the LM thousands of times a second. Therefore the maximum displacement \( x \) from the reference point is expected to only be fractions of a millimetre. Such a small horizontal displacement will result in a minute change in the upwards vertical force and therefore the change in height will be negligible.

The horizontal force component can also be derived to be Equation [16] as follows:

\[
\cos \theta = \frac{F_{rh}}{F_r} \quad \text{Equation [14]}
\]

\[ F_{rh} = F_r(x) \cos \theta \quad \text{Equation [15]} \]

\[
\therefore F_{rh}(x) = \left( \left( \frac{1}{30 \times 10^{-3} + \left( x^2 + h^2 \right)^2} \right) - \left( \frac{1}{0.0169 + \left( 30 \times 10^{-3} + d \right)^2} \right) \right)^2 \left( \frac{1}{0.0121 + d^2} \right) - \left( \frac{1}{0.0169 + d^2} \right) \right) \cos \left( \sin^{-1} \left( \frac{h}{(x^2 + h^2)^{1/2}} \right) \right) \quad \text{Equation [16]} \]

### 7.2 Z-Axis (Vertical Plane) Model

A second order differential equation for the vertical plane (\( z \)-axis) can be derived using Newton’s second law (Plastino and Muzzio, 1992):

\[ \ddot{F} = m\ddot{a} \quad \text{Equation [17]} \]

\[ \ddot{F} = m\ddot{x} \quad \text{Equation [18]} \]

\[ m\ddot{x} = F_{rv}(x) - F_g \quad \text{Equation [19]} \]

\[
m\ddot{\mathbf{x}} = \frac{\left(\frac{30 \times 10^{-3} + (x^2 + h^2)^{\frac{1}{2}}}{\sqrt{0.0169 + (30 \times 10^{-3} + d)^2}} - \frac{30 \times 10^{-3} + (x^2 + h^2)^{\frac{1}{2}}}{\sqrt{0.0121 + (30 \times 10^{-3} + d)^2}}\right)^2 - \left(\frac{x^2 + h^2}{\sqrt{0.0169 + d^2}} - \frac{x^2 + h^2}{\sqrt{0.0121 + d^2}}\right)^2}{1.084 \times 10^{-3}}\times \sin^{-1}\left(\frac{h}{(x^2+h^2)^{\frac{1}{2}}}\right) - mg
\]

Equation [20]

Equation [20] describes how the LM will experience acceleration equal to the vertical force component from the ring magnet minus the force due to gravity. If the result is a positive value the magnet will accelerate upwards. If the result is negative the magnet will accelerate downwards, and if the result is zero then the magnet will have zero velocity in either direction and appear to be floating.

It is predicted that the force due to gravity will be approximately equal to the ring magnet’s vertical component when the user releases the LM. If the LM position is kept within a small variation boundary, then the change in height of the LM will be negligible. Therefore it is predicted that the system will not need a form of height control.

However, for stable levitation to exist, the system will need to generate a horizontal force component equal and opposite to the ring magnet’s horizontal force component. This force will be produced by pairs of electromagnets.

### 7.3 Electromagnet Force Components Model

When current passes through a wire element a magnet field is produced. Moon (1994) describes the Biot-Sovart Law, which describes how the flux density changes at a specific point at distance d from a curved wire element:

\[
\frac{\partial \mathbf{B}}{\partial t} = \frac{\mu_0 I (\mathbf{\nabla} \times \mathbf{E})}{4\pi |d|^2} (T)
\]

Equation [21]

This equation can be illustrated as shown in Figure 16.
Figure 16: Change in flux density from wire loop

Where:

\[ \mu_0 = \text{The permeability of free space} \quad (4\pi \times 10^{-7} \text{ mKgs}^{-2}\text{A}^{-2}) \]

\[ \partial\vec{L} = \text{An infinitely small length of the wire element} \]

\[ \partial\vec{B} = \text{An infinitesimal change in flux density} \]

\[ I = \text{Current flowing in the conductor} \]

\[ \hat{r} = \text{A unit vector to describe the direction of the vector } d \]

\[ d = \text{The distance vector that travels from the current in the wire to a point located anywhere on the point of object line.} \]

Because the electrical conductors that will be used for the levitation system's electromagnets are not of infinitesimal lengths, the flux density is derived from the line integral of the wire element along the path \( L \) (Moon, 1994): as follows

\[ B = \int_L \frac{\mu_0 I (\partial \vec{L} \times \hat{r})}{4\pi |d|^2} (T) \quad \text{Equation [22]} \]

\[ \therefore B = \frac{\mu_0 I L}{4\pi d^2} \quad \text{Equation [23]} \]

Jayawant (1981) describes the force felt on an object, dependant on the electric current, magnetic flux and length of wire to be:
\[ F = BIL \]  \hspace{1cm} \text{Equation [24]}

Equation [23] is then substituted into Equation [24]:

\[ F = \frac{\mu_0 I^2 N^2}{4\pi d^2} \]  \hspace{1cm} \text{Equation [25]}

The length of the wire element is defined by Körner (2007) as:

\[ L = 2\pi r \]  \hspace{1cm} \text{Equation [26]}

Substituting Equation [26] into Equation [25] produces:

\[ F = \frac{\mu_0 I^2 N^2 4\pi^2 r^2}{4\pi d^2} \]  \hspace{1cm} \text{Equation [27]}

\[ \therefore F = \frac{\mu_0 I^2 N^2 \pi r^2}{d^2} \]  \hspace{1cm} \text{Equation [28]}

The area of a circle is defined as:

\[ A = \pi r^2 \]  \hspace{1cm} \text{Equation [29]}

Substituting Equation [29] into Equation [28] derives a function of force in terms of distance from the current source and input current:

\[ F_{cx}(d, I) = \frac{\mu_0 (NI)^2 A}{d^2} \]  \hspace{1cm} \text{Equation [30]}

where \( F_{cx} = \text{Force from coil number } x \)

The force produced by each electromagnet will contain both horizontal and vertical components, similar to the ring magnet. Figure 17 shows the forces produced by either the x-axis or y-axis electromagnet pair:
Where:

- \( F_a \) = The magnitude of the force produced by the A electromagnet
- \( F_{ah} \) = Horizontal component of the force \( F_a \)
- \( F_{av} \) = Vertical component of the force \( F_a \)
- \( F_b \) = The magnitude of the force produced by the B electromagnet
- \( F_{bh} \) = Horizontal component of the force \( F_b \)
- \( F_{bv} \) = Vertical component of the force \( F_b \)
- \( x \) = displacement
- \( h_{lm} \) = LM height from the center
- \( h_e \) = height of electromagnet

The displacement of the LM is denoted as 'x'. Any displacement to the left of the chosen reference point (r) will describe a negative value of x and any displacement to the right will produce a positive value of x.

Force \( F_a \) can be described using Equation [30] and the distance from the electromagnet \( d_a \) to the LM in terms of displacement x using Pythagoras (MathsIsFun, 2013):

\[
d_a = \sqrt{(h_{lm} - h_e)^2 + (\omega_x + x)^2}
\]

Equation [31]
Substituting Equation [31] into Equation [30] produces:

\[ F_a(\omega_x + x, I) = \frac{\mu_0 (NI)^2 A}{(h_{lm} - h_e)^2 + (\omega_x + x)^2} \]  

Equation [32]

Similarly force \( F_b \) is derived in terms of displacement \( x \):

\[ d_b = \sqrt{(h_{lm} - h_e)^2 + (\omega_x - x)^2} \]  

Equation [33]

\[ F_b(\omega_x - x, I) = \frac{\mu_0 (NI)^2 A}{(h_{lm} - h_e)^2 + (\omega_x - x)^2} \]  

Equation [34]

The horizontal force component \( F_{ah} \) is dependent on \( \phi \) and can be calculated using Equation [35] which utilises trigonometry. All trigonometry rules used in this document are described by MathsIsFun (2013).

\[ \phi = \sin^{-1}\left(\frac{(h_{lm} - h_e)}{d_a}\right) \]  

Equation [35]

Substituting Equation [31] into Equation [35]:

\[ \phi = \sin^{-1}\left(\frac{(h_{lm} - h_e)}{\sqrt{(h_{lm} - h_e)^2 + (\omega_x - x)^2}}\right) \]  

Equation [36]

Angle \( \phi \) can now be used to derive the force \( F_{ah} \) using trigonometric rules and substituting Equation [32] into Equation [37]:

\[ F_{ah} = F_a \cos \phi \]  

Equation [37]

\[ F_{ah}(x, I) = \frac{\mu_0 (NI)^2 A}{(h_{lm} - h_e)^2 + (\omega_x + x)^2} \cos \left( \sin^{-1}\left( \frac{(h_{lm} - h_e)}{\sqrt{(h_{lm} - h_e)^2 + (\omega_x - x)^2}} \right) \right) \]  

Equation [38]

The horizontal force component \( F_{bh} \) is dependent on angle \( \zeta \) which can again be calculated using trigonometry and therefore Equation [39]:

\[ \zeta = \sin^{-1}\left(\frac{(h_{lm} - h_e)}{d_b}\right) \]  

Equation [39]

Substituting Equation [31] into Equation [39]:

\[ \zeta = \sin^{-1}\left(\frac{(h_{lm} - h_e)}{\sqrt{(h_{lm} - h_e)^2 + (\omega_x - x)^2}}\right) \]  

Equation [40]
Angle $\zeta$ can now be used to derive the force $F_{bh}$:

$$F_{bh} = F_b \cos \zeta$$  \hspace{1cm} \text{Equation [41]}

$$F_{bh}(x, l) = \frac{\mu_0 (NI)^2 A}{(h_{lm} - h_e)^2 + (\omega x - x)^2} \cos \left( \sin^{-1} \left( \frac{h_{lm} - h_e}{\sqrt{(h_{lm} - h_e)^2 + (\omega x - x)^2}} \right) \right)$$  \hspace{1cm} \text{Equation [42]}

The vertical force component of $F_a$ is calculated using Equation [43]:

$$F_{av} = F_a \sin \phi$$  \hspace{1cm} \text{Equation [43]}

$$F_{av}(x, l) = \frac{\mu_0 (NI)^2 A}{(h_{lm} - h_e)^2 + (\omega x + x)^2} \sin \left( \sin^{-1} \left( \frac{h_{lm} - h_e}{\sqrt{(h_{lm} - h_e)^2 + (\omega x + x)^2}} \right) \right)$$  \hspace{1cm} \text{Equation [44]}

The vertical force component of $F_b$ is calculated using Equation [45]:

$$F_{bv} = F_b \sin \zeta$$  \hspace{1cm} \text{Equation [45]}

$$F_{bv}(x, l) = \frac{\mu_0 (NI)^2 A}{(h_{lm} - h_e)^2 + (\omega x - x)^2} \sin \left( \sin^{-1} \left( \frac{h_{lm} - h_e}{\sqrt{(h_{lm} - h_e)^2 + (\omega x - x)^2}} \right) \right)$$  \hspace{1cm} \text{Equation [46]}

For the proposed levitation system the angles $\phi$ and $\zeta$ will be small and therefore the following approximations can be made:

$$F_{ah}(x, l) \approx F_a(x, l)$$

$$F_{bh}(x, l) \approx F_b(x, l)$$

$$F_{av}(x, l) \approx 0$$

$$F_{bv}(x, l) \approx 0$$

### 7.4 X-Axis (Horizontal Plane) Model

Now that equations for the horizontal force components generated by the ring magnet ($F_{rh}$) and the electromagnet ($F_{cx}$) have been derived, the complete mathematical model for the x-axis can be illustrated using Figure 18.
Figure 18: X-axis forces model

Figure 18 shows that the horizontal forces produced from each of the electromagnets (Fc1 and Fc2), when combined, can overcome the unwanted horizontal force component generated by the ring magnet Frh (Equation [16], page 36).

The horizontal forces felt by the LM from each of the electromagnets can be described using Equation [32], Equation [34] and the approximations described in section 7.4, page 47:

\[ F_{c1}(\omega_x + x, I) = \frac{\mu_0 (NI)^2 A}{(h_{lm}-h_e)^2 + (\omega_x+x)^2} \]  
\[ F_{c2}(\omega_x - x, I) = \frac{\mu_0 (NI)^2 A}{(h_{lm}-h_e)^2 + (\omega_x-x)^2} \]

Equation [47]  
Equation [48]

The electromagnets in each pair will be wired in parallel and therefore the potential difference across each will be identical. The current in each of the coils depends on their resistance. The wire length, diameter and number of turns will be the same for each coil and therefore it can be assumed that they have identical impedance, and resulting in current dividing equally between both electromagnets in each pair.

However, C1 will be wired in reverse to C2 (i.e. electromagnet poles are swapped). This will ensure that when the same current flows through both electromagnets, C1 will repel the LM whereas C2 will attract. This also implies that when the LM is closer to C1, the force acting on the LM from C1 will be greater than that from C2.

A differential equation for the horizontal plane (x-axis) can be derived using Figure 18 and Newton's second law (Equation [17], page 36):
\begin{align*}
\vec{F} &= m\vec{a} \\
\ddot{\vec{F}} &= m\ddot{x} \\
\mathbf{m}\ddot{x} &= F_{c1}(\mathbf{\omega} + x, I) + F_{c2}(\mathbf{\omega} - x, I) - F_{rh}(x) \quad \text{Equation [49]}
\end{align*}

Substituting

Equation [16], Equation [47] and Equation [48] into Equation [49]:

\[
\mathbf{m}\ddot{x} = \left(\frac{\mu_0 (Nt)^2 A}{(h_{lm} - h_e)^2 + (\omega_x - x)^2}\right) + \left(\frac{\mu_0 (Nt)^2 A}{(h_{lm} - h_e)^2 + (\omega_x - x)^2}\right) - \\
\left(\frac{30\times10^{-3} + x^2 + h^2}{0.0169 + \left(30\times10^{-3} + d\right)^2}\right) - \left(\frac{30\times10^{-3} + x^2 + h^2}{0.0121 + \left(30\times10^{-3} + d\right)^2}\right) \cdot \left(\frac{\left(x^2 + h^2\right)^{1/2}}{0.0169 + d^2}\right) - \left(\frac{\left(x^2 + h^2\right)^{1/2}}{0.0121 + d^2}\right) \cdot \cos \left(\sin^{-1}\left(\frac{h}{(x^2 + h^2)^{1/2}}\right)\right) \\
\text{Equation [50]}
\]

Equation [50] describes how the LM will experience acceleration if the horizontal force component produced by the ring magnet is not cancelled out by the combination of electromagnets.
7.5 Y-Axis (Horizontal Plane) Model

The mathematical model for the y-axis is derived in the same fashion as for the x-axis because it is simply the x-axis rotated around the centre of the ring magnet by 90°. The y-axis model is illustrated in Figure 19.

![Figure 19: Y-axis forces model](image)

The horizontal forces produced by the electromagnets C3 and C4 can again be described using Equation [32], Equation [34] and the approximations described in section 7.4 page 47:

\[
F_{c3}(\omega_y + y, I) = \frac{\mu_0 (NI)^2 A}{(h_{im} - h_e)^2 + (\omega_y + y)^2} \quad \text{Equation [51]}
\]

\[
F_{c4}(\omega_y - y, I) = \frac{\mu_0 (NI)^2 A}{(h_{im} - h_e)^2 + (\omega_y - y)^2} \quad \text{Equation [52]}
\]

Similar to the x-axis a differential equation for the horizontal plane (y-axis) can be derived as follows:

\[
\ddot{F} = m\ddot{a}
\]

\[
\ddot{F} = m\dddot{x}
\]

\[
m\dddot{x} = F_{c3}(\omega + x, I) + F_{c4}(\omega - x, I) - F_{rh}(y) \quad \text{Equation [53]}
\]
Substituting Equation [16], Equation [51] and Equation [52] into Equation [53]:

\[
\ddot{m} = \left( \frac{\mu_0 (N)^2 A}{(l_{1m} - h)^2 + (\omega_{y} + y)^2} \right) + \left( \frac{\mu_0 (N)^2 A}{(l_{1m} - h)^2 + (\omega_{y} - y)^2} \right) - \\
\left( \frac{\left( 30 \times 10^{-3} + \left( x^2 + h^2 \right) \frac{1}{\sqrt{0.0169 + (30 \times 10^{-3} + d)^2}} \right)}{1.084 \times 10^{-3}} \right)^2 \cos \left( \sin^{-1} \left( \frac{h}{(x^2 + h^2)^2} \right) \right)
\]

Equation [54]

8.0 Summary of Mathematical Model

Throughout section 6.0 it was derived that the following second order differential equations represent the complete mathematical model of the three dimensional levitation system:

8.1 Z-axis (Vertical Plane)

\[
\ddot{m} = F_{rZ}(x) - F_g
\]

\[
\ddot{m} = \left( \frac{\left( 30 \times 10^{-3} + \left( x^2 + h^2 \right) \frac{1}{\sqrt{0.0169 + (30 \times 10^{-3} + d)^2}} \right)}{1.084 \times 10^{-3}} \right)^2 \cos \left( \sin^{-1} \left( \frac{h}{(x^2 + h^2)^2} \right) \right) - m g
\]

Equation [55]

8.2 X-axis (Horizontal Plane)

\[
\ddot{m} = F_{c1}(\omega_x + x, I) + F_{c2}(\omega_x - x, I) - F_{rH}(x)
\]

\[
\ddot{m} = \left( \frac{\mu_0 (N)^2 A}{(l_{1m} - h)^2 + (\omega_{x} + x)^2} \right) + \left( \frac{\mu_0 (N)^2 A}{(l_{1m} - h)^2 + (\omega_{x} - x)^2} \right) - \\
\left( \frac{\left( 30 \times 10^{-3} + \left( x^2 + h^2 \right) \frac{1}{\sqrt{0.0169 + (30 \times 10^{-3} + d)^2}} \right)}{1.084 \times 10^{-3}} \right)^2 \cos \left( \sin^{-1} \left( \frac{h}{(x^2 + h^2)^2} \right) \right)
\]

Equation [56]
8.3 Y-axis (Horizontal plane)

\[ m\ddot{x} = F_{c2}(\omega_y + y, I) + F_{c4}(\omega_y - y, I) - F_{rh}(x) \]  

\[ m\ddot{x} = \left( \frac{\mu_0(N I)^2 A}{(h_{lm}-h_e)^2+(\omega_y+y)^2} \right) + \left( \frac{\mu_0(N I)^2 A}{(h_{lm}-h_e)^2+(\omega_y-y)^2} \right) - \left( \frac{30 \times 10^{-3} + (y^2 + h_e^2)^{1/2}}{0.0169 + (30 \times 10^{-3} + d)^2} \right) \left( \frac{30 \times 10^{-3} + (y^2 + h_e^2)^{1/2}}{0.0121 + (30 \times 10^{-3} + d)^2} \right)^2 \cos^{-1} \left( \frac{h}{(y^2 + h_e^2)^{1/2}} \right) \]  

Equation [59]

Equation [60]

8.4 Assumptions and Approximations

For the purpose of generating a simplified approximation of the system the following assumptions and approximations have been made:

- The height of the electromagnet when the system is running will remain constant.
- Damping of the LM due to the mass of the magnet is negligible.
- The change in flux caused by each of the electromagnets and ring magnet do not interfere with each other.
- The angle \( \phi \) in Figure 17 is small, and therefore the change in flux produced by the electromagnets and felt by the LM is purely horizontal. The distances \( (\omega_x + x) \) and \( (\omega_x + x) \) are therefore also purely horizontal.
- \( F_{c1h}(x, I) \approx F_{c1}(x, I) \)
- \( F_{c2h}(x, I) \approx F_{c2}(x, I) \)
- \( F_{c1v}(x, I) \approx 0 \)
- \( F_{c2v}(x, I) \approx 0 \)

8.5 System Analysis and Simulation Strategy

For this system it can be seen that the model of the x-axis is identical to that of the y-axis, except the replacement of the x variable with the y variable. For this reason only the x-axis plane will be analysed, simulated and documented. It is assumed that the y-axis documentation would be an identical copy of the x-axis and therefore not necessary to include. Any control system designed for the x-axis is assumed to work with the y-axis.
9.0 Non-linear Matlab Analysis and Simulations

The second order non-linear differential Equation [57]: \( m\ddot{x} = F_{c1}(\omega_x + x, I) + F_{c2}(\omega_x - x, I) - F_{rh}(x) \), derived in section 6.0 to represent x-axis forces on the LM, can be modelled in Matlab Simulink software.

9.1 X-axis Overview

Figure 20 (page 49) shows how the levitating magnet displacement is calculated by adding the forces produced by the electromagnet pair, and taking away the force produced by the ring magnet. It can be clearly seen how the three force-calculating blocks represent the three forces in Equation [57]. The three force calculating blocks can be further expanded to show more detailed calculations.

9.2 Electromagnet C1 Force

Figure 21 (page 50) illustrates the block responsible for simulating the force \( F_{c1} \), which is part of Equation [58]: \( F_{c1}(\omega_x + x, I) = \frac{\mu_0 (NI)^2 A}{(h_{lm} - h_e)^2 + (\omega_x + x)^2} \)

The gain “Em K” represents the electromagnet constant: \( Em K = \mu_0 \times N^2 \times A \)

9.3 Electromagnet C2 Force

The block responsible for simulating the force produced by electromagnet C2 (Figure 22, page 50) has a similar arrangement. The distance between the electromagnet and the LM however is now calculated by taking the displacement x away from \( \omega_x \) instead of adding it.

9.4 Ring Magnet Horizontal Force

The ring magnet block in Figure 20 (page 49) is responsible for simulating the ring magnet horizontal force and can be expanded to show how it is modelled in Simulink (Figure 23, page 51). This Simulink model fully realises the force \( F_{rh}(x) \) found in Equation [57] and Equation [58]. The Simulink representation of the y-axis is identical to the x-axis except it uses y as the displacement instead of x.
Figure 20: X-axis realisation in Simulink
Figure 21: Electromagnet force Fc1 in Simulink

Figure 22: Electromagnet force Fc2 in Simulink
Figure 23: Ring magnet horizontal force in Simulink
9.5 Non-linear Simulation with PID Controller

The x-axis model in Figure 20 (page 49) was duplicated for the non-linear y axis model. A PID controller was connected to the complete non-linear system as shown in Figure 24. Initial conditions were set to zero to simulate the response if the user placed the magnet in the center of the electromagnets. The PID control gains were adjusted until an acceptable response was produced. These values were: Kp = 6, Ki = 2, Kd = 6.

![Figure 24: Non-linear model in Simulink](image)

The model in Figure 24 was simulated and displayed in Figure 25.

![Figure 25: Non-linear simulation with PID control](image)

The simulation in Figure 25 shows the horizontal ring magnet force start to build up from time zero. The electromagnets then work to cancel out the ring magnet force. It can be seen that the force from Fc2 is much stronger than Fc1. This is expected because the distance between the LM and Fc2 is decreasing, whereas the distance between the LM and Fc1 is increasing. Once the LM arrives at the set point the system is in equilibrium. The non-linear simulation was designed to give an overview of the system response with a feedback controller. For a detailed analysis of the system response, however, the non-linear system should be linearised around a stability point so that mathematical analysis can be carried out.
10.0 System Linearisation

The system described in section x contains nonlinear differential equations. To construct a simplified model of the system, Equation [57] and Equation [59] can be linearised. This linearisation finds the linear approximation at a given stability point.

10.1 X-Axis Linearisation

The x-axis nonlinear differential Equation [57] is described by:

\[ m \ddot{x} = F_{c1}(x, I) + F_{c2}(x, I) - F_{rh}(x) \]

Let the linearisation of \( F_{c1} \) and \( F_{c2} \) be:

\[ F_{c1}(x, I) = F_{c1}(x_0, I_0) + a_1(x - x_0) + b_1(I - I_0) \]

\[ F_{c2}(x, I) = F_{c2}(x_0, I_0) + a_2(x - x_0) + b_2(I - I_0) \]

Where:

\[ a_1 = \frac{\partial F_{c1}}{\partial x} \bigg|_{x=x_0} l = I_0 \quad a_2 = \frac{\partial F_{c2}}{\partial x} \bigg|_{x=x_0} l = I_0 \quad b_1 = \frac{\partial F_{c1}}{\partial I} \bigg|_{x=x_0} l = I_0 \quad b_2 = \frac{\partial F_{c2}}{\partial I} \bigg|_{x=x_0} l = I_0 \]

Before the values of \( a_1, a_2, b_1, b_2 \) can be calculated, values for \( x_0 \) (LM position) and \( I_0 \) (electromagnet current) must be derived. These values are chosen so that the following condition is met:

\[ F_{rh}(x) = F_{c1}(x, I) + F_{c2}(x, I) \]

Equation [61]

At this point the two forces cancel out and the system is said to be in equilibrium. Due to the nature of the system it is impossible to hold the levitating magnet exactly at a displacement of 0mm. Therefore the chosen equilibrium displacement is: \( x = x_0 = 0.1 \text{mm} \).

\( x_0 \) can now be substituted into Equation [61] which expands to become Equation [62]:

\[ \frac{\left( -\frac{30 \times 10^{-3} (x_0^2 + h^2)^2}{0.013 + (30 \times 10^{-3} + d)^2} - \frac{30 \times 10^{-3} (x_0^2 + h^2)^2}{0.013 + (30 \times 10^{-3} + d)^2} \right)^2 \left( \frac{1}{0.012 + d^2} - \frac{1}{0.012 + d^2} \right)^2}{\frac{1.084 \times 10^{-4}}{2}} \cos \left( \sin^{-1} \left( \frac{h}{(x_0^2 + h^2)^2} \right) \right) = \left( \frac{\mu_0 (N)^2 A}{(b_{lm} - b_h)^2 + (x_0 - x_h)^2} \right) + \]

Equation [62]
The system parameters from Table 20 and Table 21 (Appendix 1) are substituted into Equation [62] so that the equilibrium current $I_0$ can be found:

$$4.53 = \left( \frac{\mu_0 (Nl_0)^2 A}{(h_{lm} - h_e)^2 + (\omega x - x)^2} \right) + \left( \frac{\mu_0 (Nl_0)^2 A}{(h_{lm} - h_e)^2 + (\omega x - x)^2} \right)$$

∴ $4.53 = \frac{t_0^2 \times 2.73 \times 10^{-3}}{5 \times 10^{-4}} + \frac{t_0^2 \times 2.73 \times 10^{-3}}{4.96 \times 10^{-4}}$

∴ $4.53 = \frac{1.35 \times 10^{-6} I_0^2 + 1.365 \times 10^{-6} I_0^2}{2.48 \times 10^{-7}}$

∴ $1.12 \times 10^{-6} = 1.35 \times 10^{-6} I_0^2 + 1.365 \times 10^{-6} I_0^2$

∴ $1.12 \times 10^{-6} = (1.35 \times 10^{-6} + 1.365 \times 10^{-6}) I_0^2$

∴ $I_0^2 = \frac{1.12 \times 10^{-6}}{(1.35 \times 10^{-6} + 1.365 \times 10^{-6})} = 0.4 A$

∴ $I_0 = 0.64 A$

Therefore the linearisation becomes:

$$m \ddot{x} = F_{c1}(x_0, I_0) + F_{c2}(x_0, I_0) + a_1 (x - x_0) + a_2 (x - x_0) + b_1 (1 - I_0) + b_2 (1 - I_0) - F_{rh}(x)$$

Equation [63]

Where:

$$F_{rh}(x) = F_{c1}(x, I) + F_{c2}(x, I)$$

Therefore after cancellation:

$$m \ddot{x} = a_1 (x - x_0) + a_2 (x - x_0) + b_1 (1 - I_0) + b_2 (1 - I_0)$$

Equation [64]

∴ $\ddot{x} = \left( \frac{a_1 + a_2}{m} \right) (x - x_0) + \left( \frac{b_1 + b_2}{m} \right) (1 - I_0)$

Equation [65]

Let the following be true:

$x_1 = x - x_0$; $x_2 = \dot{x}_1 = \dot{x}$; $u = I - I_0$

Therefore:

$$\ddot{x} = \left( \frac{a_1 + a_2}{m} \right) (x_1) + \left( \frac{b_1 + b_2}{m} \right) (u)$$

Equation [66]
10.2 Linear State Space Model

According to IPSA (2005), the desired state-space realisation is denoted by:

\[ \dot{x} = Ax + Bu, \quad y = Cx + D \quad \text{Equation [67]} \]

The linearisation, equation [66], derived in section 9.1 can now be represented in state space form as follows:

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2
\end{bmatrix} = \begin{bmatrix}
0 & 1 \\
\frac{a_1 + a_2}{m} & 0
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} + \begin{bmatrix}
0 \\
\frac{b_1 + b_2}{m}
\end{bmatrix} u \quad \text{Equation [68]}
\]

\[ y = [1 \ 0] \begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} + Du \quad \text{Equation [69]} \]

Because \( l_0 \) and \( x_0 \) have been assigned values that meet the condition for equation [61], the values of \( a_1, a_2, b_1, b_2 \) can be derived by substitution. Firstly, \( a_1 \) was found:

\[
a_1 = \frac{\partial F_{c1}}{\partial x} \bigg|_I = l_0, x = x_0
\]

\[
a_1 = (h^2 + (w_x + x_0)^2 \times 0) + \frac{(\mu_0(Nl_0)^2A)(2.(w_x+x_0)^3,1)}{h^4 + 2h^2(w_x+x_0)^2 + (x_x+x_0)^2} \quad \text{Equation [70]}
\]

\[ \therefore a_1 = -179.25 \]

Next, \( a_2 \) was found:

\[
a_2 = \frac{\partial F_{c2}}{\partial x} \bigg|_I = l_0, x = x_0
\]

\[
a_2 = (h^2 + (w_x - x_0)^2 \times 0) + \frac{(\mu_0(Nl_0)^2A)(2.(w_x-x_0)^3,1)}{h^4 + 2h^2(w_x-x_0)^2 + (x_x-x_0)^2} \quad \text{Equation [71]}
\]

\[ \therefore a_2 = 183.24 \]

Next, \( b_1 \) was found:

\[
b_1 = \frac{\partial F_{c1}}{\partial l} \bigg|_I = l_0, x = x_0
\]

\[
b_1 = \frac{(h^2 + (w_x+x_0)^2)(2l_0\mu_0AN^2) - (\mu_0(Nl_0)^2A)0}{h^4 + 2h^2w_x^2 + w_x^2} \quad \text{Equation [72]}
\]
\[ b_1 = 7.04 \]

Finally, \( b_2 \) was found:

\[
b_2 = \frac{\partial F_{c2}}{\partial l} \bigg|_{l = l_0, x = x_0}
\]

\[
b_2 = \frac{(h^2 + (w_x + x_0)^2)(2l_0 \mu_0 AN^2) - (\mu_0 N l_0 A) 0}{h^4 + 2h^2 w_x^2 + w_x^4}
\]

\[ \therefore b_2 = 7.04 \]

By substituting values of \( a_1, b_1, a_2, b_2 \) into Equation [68] the final state space representation of the linearisation becomes:

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2
\end{bmatrix} = \begin{bmatrix}
0 & 1 \\
(-179.25 + 183.24) & 0
\end{bmatrix} \begin{bmatrix}
 x_1 \\
 x_2
\end{bmatrix} + \begin{bmatrix}
 0 \\
 7.04 + 7.04
\end{bmatrix} u
\]

\[ \text{Equation [74]} \]

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2
\end{bmatrix} = \begin{bmatrix}
 0 & 1 \\
 40 & 0
\end{bmatrix} \begin{bmatrix}
 x_1 \\
 x_2
\end{bmatrix} + \begin{bmatrix}
 0 \\
 140
\end{bmatrix} u
\]

\[ \text{Equation [75]} \]

\[
y = \begin{bmatrix}
1 & 0
\end{bmatrix} \begin{bmatrix}
 x_1 \\
 x_2
\end{bmatrix} + D u
\]

\[ \text{Equation [76]} \]

According to Hou (2012) the state space representation can also be illustrated by the block diagram in Figure 26:

Figure 26 shows how the A, B and C matrices are related. The arrow box located after the signal \( \dot{x} \) represents an integrator which transforms the signal into \( x \).
10.3 Transfer Function

Hou (2012) describes how the transfer function can be derived from the state space representation of the system. The Laplace transform of the state equation [74] is represented by:

\[ G(s) = \frac{\mathcal{L}[y(t)]}{\mathcal{L}[u(t)]} = \frac{Y(s)}{U(s)} \]

Equation [77]

\[ G(s) = C(sI - A)^{-1}B + D \]

\[ G(s) = C(sI - A)^{-1}B + D \]

\[ G(s) = [1 \quad 0]\left(\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 40 & 0 \end{bmatrix}\right)^{-1}\begin{bmatrix} 0 \\ 140 \end{bmatrix} + D \]

\[ G(s) = [1 \quad 0]\left(\begin{bmatrix} s & -1 \\ -40 & s \end{bmatrix}\right)^{-1}\begin{bmatrix} 0 \\ 140 \end{bmatrix} \]

\[ G(s) = [1 \quad 0]\left(\begin{bmatrix} \frac{1}{s^2 - 40} \\ \frac{s}{140} \end{bmatrix}\right) \]

\[ G(s) = [s \quad 1]\begin{bmatrix} 0 \\ 140 \end{bmatrix}\left(\frac{1}{s^2 - 40}\right) \]

\[ G(s) = [140]\left(\frac{1}{s^2 - 40}\right) \]

Equation [78]

The transfer function \( G(s) \) illustrates two poles (eigenvalues). The eigenvalues of the system are:

\[ s^2 - 40 = 0 \]

\[ s^2 = 40 \]

\( \lambda_1 = +\sqrt{40} = +6.3; \quad \text{Eigenvalue 1} \)

\( \lambda_2 = -\sqrt{40} = -6.3; \quad \text{Eigenvalue 2} \)

One of the poles of the system is a positive number, and is therefore on the right hand side of the root locus. A positive eigenvalue explains a system that is naturally unstable. Such a system cannot become stable without the aid of a controller. To graphically check the position of the systems eigenvalues Matlab was used to plot the root locus.
Figure 27: Root locus of transfer function

The root locus in figure confirms that the system has two poles, placed at -6.3 and +6.3.

10.4 Controllability / Observability

From the root locus in Figure 27 and the transfer function derived in section 9.3 it is proven that the levitation system is naturally unstable without any form of feedback control. Before a controller is designed, the controllability of the system should be checked. Hou (2012) states that a system is said to be fully controllable if the controllability matrix $M_C$ has full rank. The matrix $M_C$ is described by:

$$M_C = \begin{bmatrix} B & AB & \ldots & A^{n-1}B \end{bmatrix}$$

Equation [79]

The number of columns is equal to the number of system inputs. Therefore for the levitation system the controllability matrix is defined by:

$$M_C = \begin{bmatrix} B & AB \end{bmatrix}$$

Equation [80]

Where:

$$A = \begin{bmatrix} 0 & 1 \\ 40 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 140 \end{bmatrix}, \quad AB = \begin{bmatrix} 140 \\ 0 \end{bmatrix}$$

$$\therefore M_C = \begin{bmatrix} 0 & 140 \\ 140 & 0 \end{bmatrix}$$

$$|M_C| = 1900 \neq 0$$
The determinant of matrix $M_C$ is a non-zero value and therefore the matrix has a full rank of 2, implying that the system is fully controllable.

Aim 2 of the project describes how the system at the end of the project should be able to detect numerical errors provided by the feedback. It was decided that an observer would be used for each sensor in order to check the measured position of the levitating magnet with the predicted position. For an observer to work the system must first be observable.

Similarly to the controllability test, the observability of a system can be deduced from the rank of an observability matrix. The observability matrix is defined as follows:

$$
M_o = \begin{bmatrix}
C \\
CA \\
\vdots \\
CA^{n-1}
\end{bmatrix}
$$

Equation [81]

Therefore the observability matrix of the levitation system is defined by:

$$
M_o = \begin{bmatrix}
C \\
CA
\end{bmatrix}
$$

Equation [82]

Where:

$$
A = \begin{bmatrix} 0 & 1 \\ 40 & 0 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad CA = \begin{bmatrix} 0 & 1 \end{bmatrix}
$$

$$
\therefore \quad M_o = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
$$

$$
|M_o| = 1 \neq 0
$$

The determinant of matrix $M_o$ is a non-zero value and therefore the matrix has a full rank of 2, implying that the system is fully observable.
10.5 Open Loop Response of the system

In section 9.3 it was predicted that the system would be naturally unstable without any feedback control. The open loop response of the system described by the state-space equation [74] was simulated in Matlab using the script in Appendix 2 (page 122).

![Open Loop Response to Non-Zero Initial Condition](image)

**Figure 28: Open loop response**

Figure 28 illustrates that the position of the levitating magnet without any feedback is predicted to be approximately 1.5m away from the starting point in just two seconds of time. This system is therefore completely unstable: If it was to be turned on without the necessary control it could cause serious harm. For this system three different methods of feedback control were analysed and simulated.

10.6 Feedback Controller Design

The block diagram representation of the system with feedback control is shown in Figure 9 (page 25). There are many types of controller that could be designed for this system. The controllers tested in this project were: proportional gain controller; pole placement controller and PID controller.
10.6.1 Proportional gain controller

Facstaff (2002) describes how a proportional gain controller consists simply of one gain value $K$. This can be illustrated by the block diagram in Figure 29.

![Block Diagram of Proportional Gain Control](image)

**Figure 29: Proportional gain control**

The controller in Figure 29 simply multiplies the error between the LM’s measured position and desired position by the chosen gain value of $K$. When the gain value of $K$ is used, the transfer function of the system is derived:

\[
G_{cp}(s) = G_{control}(s) \times G_{Plant}(s)
\]  
Equation [83]

\[
G_{cp}(s) = K \times \frac{140}{s^2 - 40}
\]  
Equation [84]

\[
G_{cp}(s) = \frac{140K}{s^2 - 40}
\]  
Equation [85]

Next the feedback loop is eliminated:

\[
G(s) = \frac{140K}{1 + \frac{140K}{s^2 - 40}}
\]  
Equation [86]

The transfer function is simplified by multiplying through by $\frac{s^2 - 40}{s^2 - 40}$:

\[
G(s) = \frac{140K}{s^2 - 40 + 140K}
\]  
Equation [87]

Upon inspecting the denominator of the transfer function it can be seen that when $K$ is a positive value the two eigenvalues of the system will be a complex conjugate pair with no real part. In this case the system is marginally stable. When the gain $K$ is negative the eigenvalues will be made from one positive and one negative value similar to the open-loop system. In this scenario the system is unstable. Therefore this controller cannot be used to stabilise the system.
10.6.2 Pole Placement Control

Currently the open-loop system has two poles: one negative and one positive. The positive pole is the reason why the system is naturally unstable. If a gain controller could be designed such that the positive pole was placed on the real axis, at a negative value, then the system will theoretically drive to stability.

The controller is designed such that \( A + BK \) has the desired eigenvalues:

\[
p_1 = -8, \quad p_2 = -10
\]

These values can be contained in the following vector:

\[
P = [-8 \quad -10]
\]

The gain matrix for the system is denoted by:

\[
K = [k_0 \quad k_1]
\]

The gain constants \( k_0, k_1 \) are calculated by comparing the coefficients of two characteristic equation polynomials. The first polynomial is produced by expanding the following:

\[
Polynomial 1 = \det(sI - (A - BK))
\]

Equation [88]

The matrices \( A, B \) and \( K \) are substituted into equation [88].

\[
Polynomial 1 = \det \left( \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} - \left( \begin{bmatrix} 0 & 1 \\ 40 & 0 \end{bmatrix} - \begin{bmatrix} 0 & k_0 \\ 140 & k_1 \end{bmatrix} \right) \right)
\]

\[
Polynomial 1 = \det \left( \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} - \left( \begin{bmatrix} 0 & 1 \\ 40 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 140k_0 & 140k_1 \end{bmatrix} \right) \right)
\]

\[
Polynomial 1 = \det \left( \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} - \left( \begin{bmatrix} 0 & 0 \\ 40 - 140k_0 & 1 \end{bmatrix} \right) \right)
\]

\[
Polynomial 1 = \det \begin{bmatrix} s & -1 \\ 140k_0 - 40 & s + 140k_1 \end{bmatrix}
\]

\[
\det \begin{bmatrix} s & -1 \\ 140k_0 - 40 & s + 140k_1 \end{bmatrix} = s^2 + 140k_1s + 140k_0 - 40 \quad \text{(Polynomial 1)}
\]

The second polynomial is described by the desired poles characteristic equation:

\[
(s - p_1)(s - p_2) = (s + 8)(s + 10) = s^2 + 18s + 80 \quad \text{(Polynomial 2)}
\]

Therefore by comparing the coefficients of polynomial 1 and polynomial 2 the gain constants \( k_0 - k_3 \) can be calculated:
Comparing $s^1$: 

\[ 140k_1 = 18 \quad k_1 = \frac{18}{140} \quad k_1 = 0.13 \]

Comparing $s^0$: 

\[ 140k_0 - 40 = 80 \quad k_0 = \frac{120}{140} \quad k_0 = 0.86 \]

Therefore the calculated gain matrix is:

\[ K = \begin{bmatrix} 0.86 & 0.13 \end{bmatrix} \]

The controller in the form $U = Kx$ is therefore:

\[ U = \begin{bmatrix} 0.86 & 0.13 \end{bmatrix} x \]

The system was simulated in Matlab using the code in Appendix 3 (page 124) and Figure 30 shows the result.

---

Figure 30: Pole placement simulation for poles at -8 and -10

Figure 30 shows that the system with feedback is now stable. Because additional gain has been added the system the poles have been relocated and both now reside on the left-hand S plan of the S-Domain axis. The simulation in Figure 28, however, illustrates that when the poles are placed at -8 and -10 respectively the time response of the system is approximately 0.8s. This response is much slower than the system requirements specified in Table 4. The poles were chosen to be -80 and -100 respectively and matrix K was recalculated.

The new characteristic polynomial equation is as follows:

\[(s - p_1)(s - p_2) = (s + 80)(s + 100) = s^2 + 180s + 8000\]
Comparing $s^1$:

\[ 140k_1 = 180 \quad \implies \quad k_1 = \frac{180}{140} = 1.3 \]

Comparing $s^0$:

\[ 140k_0 - 40 = 8000 \quad \implies \quad k_0 = \frac{8040}{140} = 57.4 \]

Therefore the calculated gain matrix is:

\[ K = [57.4 \quad 1.3] \]

The controller in the form $U = Kx$ is therefore:

\[ U = [57.4 \quad 1.3] x \]

The system was simulated with the new feedback controller:

![Figure 31: Pole placement at -80 and -100](image)

The time response of the system is now approximately 0.08s which is within the accepted time range. However, in order to produce this response a gain component of 57.4 had to be used. There is a chance that this value is too high to use in an embedded system prototype because the range of output is limited to the number of bits of the processor output. Therefore it was decided that this controller was not suitable for use in an embedded system prototype and that a controller using lower gains must be designed.
10.6.3 PID Controller

National Instruments (2014) described how a PID controller will compare the error of the measured levitating magnet position to a defined position set point. The controller will then use this error to produce three gain values: $K_p$, $K_i$, $K_d$. These gain values will be used to change the magnitude of the output power to the electromagnets. The three gain types stand for: differential, proportional and integral respectively. According to Control Tutorials for Matlab (2012) the transfer function of a PID controller in the s domain is:

$$G_{PID}(s) = K_p + \frac{K_i}{s} + K_d s$$

Equation [89]

The block diagram of the closed loop system with PID control is therefore illustrated by Figure 32:

![Figure 32: Block diagram for PID control](Image)

From the block diagram in Figure 32 the transfer function for the control loop can be derived:

$$Gcp(s) = G_{PID}(s) \times G_{Plant}(s)$$

Equation [90]

$$Gcp(s) = (K_p + \frac{K_i}{s} + K_d s) \times \frac{140}{s^2 - 40}$$

Equation [91]

$$Gcp(s) = \frac{(K_p s + K_i + K_d s^2)(140)}{s^3 - 40 s}$$

Equation [92]

Next the feedback loop is eliminated:

$$G(s) = \frac{(K_p s + K_i + K_d s^2)(140)}{1 + \frac{(K_p s + K_i + K_d s^2)(140)}{s^3 - 40 s}}$$

Equation [93]
The transfer function is simplified by multiplying through by \( \frac{s^3 - 40s}{s^3 - 40s} \):

\[
G(s) = \frac{(K_p s + K_i + K_d s^2)(140)}{s^3 - 40s + (K_p s + K_i + K_d s^2)(140)}
\]

Equation [94]

Multiplying out the brackets derives:

\[
G(s) = \frac{140K_p s + 140K_i + 140K_d s^2}{s^3 + 140K_d s^2 - 40s + 140K_p s + 140K_i}
\]

Equation [95]

Factorising powers of \( s \) derives the following:

\[
G(s) = \frac{140K_d s^2 + 140K_p s + 140K_i}{s^3 + 140K_d s^2 - (40 - 140K_p) s + 140K_i}
\]

Equation [96]

Dividing the transfer function by \( \frac{140}{140} \) identifies gain values of \( K_p, K_i, \) and \( K_d \) without pre-multipliers:

\[
G(s) = \frac{K_d s^2 + K_p s + K_i}{s^3 + (0.3 - K_p) s + K_i}
\]

Equation [97]

The transfer function \( G(s) \) of the system can be used to decide whether the PID controller will produce a stable levitation system. The gains \( K_p, K_i, \) and \( K_d \) could be chosen arbitrarily until the desired results are obtained, however, there are two mathematical methods of calculating them:

- Routh-Hurwitz stability criterion
- Ziegler-Nichols Analysis

Section 9.6.4 and 9.6.5 describe the mathematical process defined by the above methods. Section 9.6.6 then analyses how to tune the PID controller.
10.6.4 Routh-Hurwitz Stability Criterion

The polynomial in the denominator of the PID system transfer function, equation [97], is now used to produce the following Routh-Hurwitz array as described by Takaya (2009):

<table>
<thead>
<tr>
<th>#</th>
<th></th>
<th>( \frac{1}{K_d} )</th>
<th>(- (0.3 - K_p) )</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>( \frac{1}{140} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>( K_d )</td>
<td>( K_i )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>( A )</td>
<td>( B )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>( C )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7: Routh-Hurwitz array

Letters A to C are calculated as follows:

\[
A = - \frac{1}{K_d} \times \begin{vmatrix} \frac{1}{140} & -(0.3 - K_p) \\ K_i & K_i \end{vmatrix} = - \frac{K_i}{140K_d} - \frac{-0.3K_d + K_dK_p}{K_d} \]

\[
\therefore A = - \frac{K_i}{140K_d} + 0.3 - K_p \]

\[
B = - \frac{1}{K_i} \times \begin{vmatrix} -(0.3 - K_p) & 0 \\ 140K_i & 0 \end{vmatrix} = 0
\]

\[
C = - \frac{1}{A} \times \begin{vmatrix} 140K_d + K_p & K_i \\ A & 0 \end{vmatrix} = \frac{K_iA}{A} \]

\[
\therefore C = \frac{K_iA}{A} = K_i
\]
By substitution the new Routh-Hurwitz array becomes Table 8:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\frac{1}{140}$</td>
<td>$-(0.3 - K_p)$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$K_d$</td>
<td>$K_i$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>$-(\frac{K_i}{140K_d} + 0.3 - K_p)$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>$K_i$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

For the system to be stable all values in the far left column must be greater than zero. Therefore:

$$-\frac{K_i}{140K_d} - 0.3 + K_p > 0 \quad \therefore \quad K_p > +\frac{K_i}{140K_d} + 0.3$$  \hspace{1cm} \text{Equation [101]}

By rearranging equation [101] the limits for $K_i$ and $K_d$ can be found:

$$0 < K_d < -\frac{K_i}{140(0.3 - K_p)}$$  \hspace{1cm} \text{Equation [102]}

$$0 < K_i < -42K_d + 140K_dK_p$$  \hspace{1cm} \text{Equation [103]}

To test these rules of stability different gain values were simulated in Matlab. First of all values outside of the calculated range were chosen:

$K_p = 0.3; \quad K_i = 2; \quad K_d = 0.1$

Substituting these gain values into equation [101]:

$$\therefore \left(-\frac{2}{140 \times 0.1} - 0.3 + 0.3\right) = -0.74$$

The result is negative and therefore it is predicted that the system will be unstable with these gains. Matlab was used to simulate the system and the output is illustrated in Figure 33.
Figure 33: PID control with unstable Kp, Ki and Kd values

Figure 33 shows that the output of the system oscillates with increasing amplitude over time and therefore represents the system becoming unstable. Next, values of $K_p$, $K_i$, and $K_d$ that provide a positive solution to equation [101] were chosen and the simulation was run again:

For:

$$\begin{align*}
K_p &= 3 \\
K_i &= 2 \\
K_d &= 1
\end{align*}$$

$$- \frac{K_i}{140K_d} - 0.3 + K_p = 2.69$$

Figure 34: PID control with stable Kp, Ki and Kd values

Figure 34 shows that the calculated PID $K_p$, $K_i$, and $K_d$ produced a stable system.

10.6.5 Ziegler-Nichols Method

According to Ziegler and Nichols (1942, pp.759–768) the PID gain values can be tuned using the impulse response of the closed-loop system with only proportional gain control. The gain is slowly
increased until the output of the system is an oscillation of constant amplitude. At this point measurements of the signal can be taken and used to calculate gain values for $K_p$, $K_i$, and $K_d$ for which the system will become stable.

The system was simulated in Matlab, using the script in Appendix 4 (page 126). $K_p$ was increased to a gain of 3, at which point the output represented an oscillation of constant amplitude. Figure 35 shows the result:

![Figure 35: Output oscillation of constant amplitude](image)

$t_0$ is measured to be the time taken for one complete oscillation and $K_0$ represents the proportional gain that produced the output oscillation. Therefore

$t_0 = 0.3$

$K_0 = 3$
The following equations are used to calculate $K_p$, $K_i$, and $K_d$ using $t_0$ and $K_0$:

Calculated $K_p = 0.6 \times K_0$  

Equation [104]

Calculated $K_i = \frac{2 \times \text{Calculated } K_p}{t_0}$  

Equation [105]

Calculated $K_d = \frac{\text{Calculated } K_p \times t_0}{8}$  

Equation [106]

Therefore substituting the measured values of $t_0$ and $K_0$ into equations [104], [105] and [106]:

Calculated $K_p = 0.6 \times 3 = 1.8$  

Equation [107]

Calculated $K_i = \frac{2 \times 1.8}{0.3} = 11.25$  

Equation [108]

Calculated $K_d = \frac{1.8 \times 0.3}{8} = 0.072$  

Equation [109]

The calculated gains of $K_p$, $K_i$, and $K_d$ were inputted into the Matlab simulation. The resultant simulation output is shown in Figure 36.

![Figure 36: PID control with gain values from Ziegler-Nichols calculations](image)
Figure 36 shows the output of the system becoming stable and settling to a final displacement of 0mm after approximately five seconds. This illustrates that the Ziegler-Nichols method works to find gain values that stabilise the system, however, the time taken to settle is outside of the specified time limit for the system (Table 4, page 17).

### 10.6.6 Tuning the PID Controller

The Routh-Hurwitz array in Table 8 produced a stability criterion, while the Ziegler-Nichols method provided sensible estimates of $K_p$, $K_i$, and $K_d$ such that the system is also stable. After reviewing the simulations in section 67 and 69 it is concluded that the time response of the system is not fast enough to match the desired response specified in Table 4. Therefore the gain values of $K_p$, $K_i$, and $K_d$ needed to be tuned.

### 10.6.7 PID Controller in Simulink

The PID controller designed in section 9.6.3 was built in Matlab Simulink as shown in Figure 37.

![Figure 37: PID controller in Matlab Simulink](image)

The circuit in Figure 37 shows how the block diagram in Figure 32 is represented in Matlab Simulink. The state-space block represents the state-space equation [74] derived in section 9.2.

The constant input block represents the desired position of the levitating magnet and is therefore set to 0.1mm from the centre point. It was assumed that when the system is built in real life and the user attempts to place the magnet at the centre point, that he/she will actually place the magnet approximately 5mm from that point. For this reason the initial condition for the state-space system was set to $x_0 = 5mm$ at time zero ($t_0$).

The first simulation was designed to test the system using the values of $K_p$, $K_i$ and $K_d$ suggested by the Zieger-Nichols method in section 9.6.5. The gain values of the PID control block were therefore assigned to be: $K_p = 1.8$; $K_i = 11.25$ and $K_d = 0.072$ and the simulation was initiated. Figure 38 illustrates the display on the scope.
The simulation result shown by Figure 38 matches the impulse response shown in Figure 36. Therefore the circuit was correctly assembled in Simulink. The gain values of Kp, Ki and Kd were then tuned to be 2, 0.22 and 7.69 respectively, which produced a settling time of approximately 0.08s as shown in Figure 39. This response was less than the settling time requirement of the system in Table 4 (page 17).

This control type produced a suitable system response, while also being simple enough to program onto a CPU. Therefore this method was chosen to be used in the final design.

10.7 Observer Design

The output of the linear state space model derived in section 9.2 can be used to predict the states of the system. Therefore an initial position measurement of the levitating magnet can be used to predict the
future position. Patton (2014) describes how the structure of the levitation plant can be represented as a signal flow diagram as shown in Figure 40.

\[ \dot{x} = \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} \]

**Figure 40: Levitation plant signal flow**

Where:

- \( x \) = states of the plant;
- \( y \) = outputs from the plant;
- \( u \) = inputs to the plant

Patton (2014) also describes how the structure of an observer is almost identical to that of the plant as shown in Figure 41:

**Figure 41: Observer flow diagram**

The observer can therefore be represented by the following state-space expression:

**Equation [110]**

\[ \dot{z} = Dz + LCx + Bu \]

### 10.7.1 Calculating Observer Gain

The observer gain matrix can be described by the following matrix:

**Equation [111]**

\[ L = \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} \]
The generic state estimation matrix $D$ is defined by Patton (2014) to be equation [112]:

$$
D = A - LC
$$

Equation [112]

$$
D = \begin{bmatrix} 0 & 1 \\ 40 & 0 \end{bmatrix} - \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix}
$$

Equation [113]

$$
D = \begin{bmatrix} -l_1 \\ 40 - l_2 \\ 1 \\ 0 \end{bmatrix}
$$

Equation [114]

The characteristic polynomial of the observer is found by solving equation [115]:

$$
\det(\lambda I - D) = 0
$$

Equation [115]

$$
(\lambda I - D) = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} - \begin{bmatrix} -l_1 \\ 40 - l_2 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \lambda + l_1 & -1 \\ l_2 - 40 & \lambda \end{bmatrix}
$$

Equation [116]

$$
\therefore \; \det(\lambda I - D) = (\lambda^2 + l_1 \lambda) - (40 - l_2) = \lambda^2 + l_1 \lambda + l_2 - 40
$$

Equation [117]

The desired poles of the observer were chosen arbitrarily be:

$$
\lambda_{o1} = -20
$$

$$
\lambda_{o2} = -30
$$

Therefore the characteristic equation of the desired poles is:

$$
(\lambda + 20)(\lambda + 30) = \lambda^2 + 50\lambda + 600
$$

Equation [118]

Therefore by comparing the coefficients of the system characteristic equation with the desired characteristic equation of the observer the gain constants $l_1$, $l_2$ can be calculated:
Comparing $\lambda^1$: 
\[ l_1 = 50 \]

Comparing $\lambda^0$: 
\[ l_2 - 40 = 600 \quad l_2 = 640 \]

Therefore the calculated gain matrix $L$ is:
\[ L = \begin{bmatrix} 50 \\ 640 \end{bmatrix} \quad \text{Equation [119]} \]

The mathematical representation of the observer is also defined by Patton (2014) as follows:

\[ \dot{Z} = Az + L(y - Cz) + Bu \quad \text{Equation [120]} \]

\[ \dot{Z} = Az + Ly - LCz + Bu \]

\[ \dot{Z} = (A - LC)z + Bu + Ly \]

Where:

\[ (A - LC) = D = \begin{bmatrix} -l_1 \\ 40 - l_2 \\ 1 \\ 0 \end{bmatrix} \quad \text{Equation [121]} \]

By substituting the calculated gain values into equation [121]:

\[ (A - LC) = \begin{bmatrix} -50 \\ -600 \\ 1 \\ 0 \end{bmatrix} \quad \text{Equation [122]} \]

Patton (notes) describes the transformed state variable system, using the separation principle, as:

\[ \begin{bmatrix} \dot{x} \\ \dot{e} \end{bmatrix} = \begin{bmatrix} A & BK \\ 0 & A - LC \end{bmatrix} \times \begin{bmatrix} x \\ e \end{bmatrix} \quad \text{Equation [123]} \]

Where:

\[ \text{State error} = x - \hat{x} \]

\[ \text{gain matrix } K = [57.4 \quad 1.3] \quad \text{(Derived using pole-placement in section 9.6.2, page 62)} \]

Therefore the transformed matrices of $A$, $B$ and $C$ are:

\[ A_{\text{transformed}} = \begin{bmatrix} A & BK \\ 0 & A_{\text{LC}} \end{bmatrix} \quad \text{Equation [124]} \]

\[ B_{\text{transformed}} = \begin{bmatrix} B \\ 0 \end{bmatrix} \quad \text{Equation [125]} \]

\[ C_{\text{transformed}} = [C \quad 0] \quad \text{Equation [126]} \]

Using the Matlab process described by Control Tutorials for Matlab (2014), the state-estimate feedback response was plotted using the script in Appendix 5 (page 128), and equations [124 – 126], for observer poles at -20 and -30 at non-zero initial conditions. Figure 42 illustrates the result.
Figure 42: Observer simulation with poles at -20 and -30

Figure 42 shows how the estimate of the magnet position approximates the measured magnet position quite fast, whereas the estimate of the velocity takes much longer to settle. Control Tutorials for Matlab (2014) suggests that the observer poles should be at least 5 times further to the left than closed loop poles. Therefore the observer poles were changed to be -40 and -50 and the simulation was run again.

Figure 43: Observer simulation with poles at -60 and -80

Both states in Figure 43 are tracked much better. It is therefore concluded that observer poles at -40 and -50 produce acceptable state-estimates of the system. This observer can be used to detect a faulty sensor reading.
11.0 Discussion of Simulation Results

In section 9.5 Matlab was used to determine that the open-loop system was unstable. Therefore, from the open-loop simulation (Figure 28), it is predicted that if the system prototype is built without a feedback controller then the levitating magnet would be flung from the device. This would be a dangerous situation and therefore validated the requirement to design a feedback controller. The mathematics in section 9.4 determined that the linear system is controllable and therefore allowed the design of a feedback controller to begin.

When a proportional gain controller was designed in section 9.6.1, it was found that there was not a feedback gain value that produced only negative closed-loop eigenvalues. When a positive feedback gain was applied, there existed only a complex conjugate pair of poles with no real part. This would cause the output of the system to oscillate and not settle. When a negative gain was applied it was found that the location of poles was similar to that of the open-loop system: one positive and one negative. If the system was run with these eigenvalues then the output response would be identical to that of the open-loop system: the levitating magnet would become unstable. This analysis justified the need to design a different type of feedback controller.

By realising that the poles of the system were keeping it from reaching stability a pole-placement controller was designed in section 9.6.2. When the desired poles were both chosen to be low negative values, the simulation in Figure 30 illustrated that the system response settles and therefore stabilised the levitation. If this controller was used in the system prototype then it is predicted that the levitation would settle and become stable. However, from further analysis it was noticed that the settling time of this controller would be too slow and therefore the desired closed-loop poles were placed further to the left on the s-domain axis. The new simulation in Figure 31 showed that the system produced a stable response within the required settling time. However, by lowering the settling time, one gain component of the controller was increased to a high value of 57.4. It was decided that this gain value was too high to implement in an embedded control system. Therefore a more complex control system was designed: the PID controller.

By using Ziegler-Nichols and Routh-Hurwitz analysis, initial gain values for Kp, Ki and Kd were derived to be 1.8, 11.25 and 0.072 respectively. These values were further refined, through experimentation, to be 2, 0.22 and 7.69 respectively. By using these values, a desired system response (Figure 39) was produced with an acceptable settling time. The PID gain values were all relatively low and therefore this controller was chosen to be the best for an embedded system prototype.

An observer was designed in section 9.7 that used mathematical techniques to estimate the state response of the system. When the observer poles were too close to the closed-loop poles the tracking of the states was too slow. When the poles were moved further left, the tracking speed became acceptable.
12.0 Hardware and Software Prototyping

The levitation system was built and tested in four stages:

- Prototype v.1 – Achievement of stable levitation
- Prototype v.2 – Addition of dynamic redundancy to eliminate risk of sensor failure
- Prototype v.3 – Addition of hardware redundancy to eliminate risk of microprocessor failure
- Temperature Control – To eliminate risk of components overheating

For each prototype, appropriate software was designed and the system was tested. When designing the software it was important to choose an appropriate development model. Munassar and Govardhan (2010) compared three popular development models: Waterfall Model; Spiral Model and the Prototyping Model.

The Prototyping Model revolves around the idea of prototyping a software concept from an incomplete specification. It is usually assumed that the full specification is not known; several prototypes of the software can be made and then developed further when more specification is available. For this project, the complete software specification was initially developed and there were strict deadlines to be met and therefore this was not the right model to use.

The Spiral Model focuses on high risk projects. If the software development faces high risk then this can be incorporated into the lifecycle and actions taken accordingly. For this project, however, the software development process faced low risk and time limitations and therefore this model was not used.

The Waterfall Model describes a simple development process for a project which is not so heavily focused on the software side of development. The model analyses a full specification and then works in a logical fashion towards a definitive finish when the software is ready for use. Therefore this development model (shown in Figure 44) was chosen to be incorporated into the creation of each version of levitation software.

![Figure 44: Agile waterfall approach to software design](image)
12.1 Feedback Board Prototype

Before the first prototype was built, a prototype feedback board (Figure 45) was first made from Veroboard:

![Feedback prototype board](image)

Figure 45: Feedback prototype board

Figure 45 shows the main sensor pair 1 (X+ and Y+) to be used initially and the spare sensor pair (X- and Y-) to be used as dynamic redundancy.

12.2 Feedback PCB Design

The schematic in Figure 5, combined with the Veroboard prototype layout, was then converted into the feedback PCB design shown in Figure 46.

![Feedback board PCB design](image)

Figure 46: Feedback board PCB design

The tracks between each of the components on the board were routed manually and a ground plane was inserted. Three drill holes were placed in the corners of the board so that screws can later be used to easily attach the board to the device.
13.0 Three Dimensional Levitation Device Prototype V.1

The first levitation prototype was built from the concept art designs (Figure 3 and Figure 4) in section 5.2 (page 20) with the sole purpose of providing stable levitation without fault tolerant techniques. The prototype levitation device (Figure 47) was built with small electromagnets (properties found in Table 21), and therefore utilises relatively small currents of approximately 600mA per electromagnet.

Figure 47 shows the prototype feedback board from Figure 45 (page 80) in the centre of the four electromagnets. The permanent Neodymium ring magnet is located inside of the wooden box. The wooden box provides safety to the user and also protection for the ring magnet. The transparent plastic container which surrounds the electromagnets and feedback board provides essential protection to the Hall Effect sensors.

The rest of the system (computer control and electromagnet drivers), designed in Figure 4, was also assembled. The variable resistors shown in Figure 47 provide the computer control with the following data: x-axis set point; y-axis set point; proportional gain; differential gain and integral gain. The computer control (Arduino) in this setup realises a PID controller to control the levitation. The circuit schematic for prototype v.1 is shown in Figure 48:
Figure 48: Prototype v.1 circuit diagram
13.1.1 PID Control Software in C Language

The PID controller designed in section 9.6.3 was chosen to be the control method used in the levitation prototype. The prototype utilised two identical PID controllers, responsible for controlling the x-axis and y-axis respectively. The $K_p$, $K_i$, and $K_d$ gains tested and simulated in section 9.6.7, Figure 39, will be used in the real life PID controller. By using these values the validity of the mathematical analysis and simulations can be tested. When designing software a flow chart should always be designed to describe the operation of the software. A flowchart describing the software processes is shown in Figure 49.

![Figure 49: PID software flow chart](image)
13.1.2 Flow Chart Analysis

The software flow chart in Figure 49 is analysed as follows:

1. **Declare variables**: Variables must be declared at the start of any piece of software.

2. **Read variable resistors**: The values of the variables resistors are read and mapped to the chosen data range. The mapped ranges are: \( Kp = 0 – 5; Ki = 0 – 0.5 \) and \( Kd = 0 – 12 \).

3. **Random error detection**: It is expected that random errors will occur when taking readings due to noise. Therefore it was decided that an algorithm would be used to average out five readings for each sensor. The average function in the Arduino function library could have been used, however using an algorithm is much faster and therefore more suitable for my system. The algorithm is as follows:
   a. Read sensor and set sensor value to minimum and maximum value.
   b. Read sensor again. If the new value is greater than the previous then assign it to the maximum value. Otherwise the new value must be less than the previous and can be assigned to the minimum value.
   c. Repeat step b 4 more times.
   d. Sum the 5 sensor readings.
   e. Remove the minimum and maximum readings as they could be potential error.
   f. The average reading is therefore: \[
   \text{average} = \frac{\sum \text{readings} - \text{Minimum} - \text{Maximum}}{3}
   \]

4. **Errors and integral** (CSIMN, 2014): The system errors and integral are taken for both the x-axis and the y-axis using the following equations:
   \[
   \text{position error} = \text{desired setpoint} - \text{sensor reading}
   \]
   \[
   \text{differential error} = \text{position error} - \text{previous position error}
   \]
   \[
   \text{integral} = \text{integral value + position error}
   \]

5. **Output power** (CSIMN, 2014): The output power for each axis is determined by:
   \[
   \text{Power} = (\text{position error} \times Kp) + (\text{differential error} \times Kd) + (\text{integral} \times Ki)
   \]
   If the power value is greater than zero it implies that the LM has moved towards the sensor and therefore the output direction must be set to positive thus repelling the LM. If the power value is negative the opposite situation is true and therefore the output direction must be negative thus attracting the LM.

6. **Magnet detection**: It is important that the system knows if the magnet is present. It is assumed that if the measured position of the magnet is too far from the reference point then the output power and integral term should be made equal to zero. This prevents the integral term rising exponentially to infinity and also stops the system drawing too much current.

7. **Update error**: The differential error is dependent upon the previous error and therefore the current error needs to be assigned to the previous error variable.
13.1.3 Primary Sensor Pair Software

The flow chart in Figure 49 was used to write the PID controller software which can be found in Appendix 6 (page 130). The software in Appendix 6 makes use of the primary sensor pair.

13.1.4 Secondary Sensor Pair Software

The software written in Appendix 6 was then duplicated for use with the spare sensor pair and can be found in Appendix 7 (page 134). However, because the backup sensor on each axis is located at the opposite side to the primary sensor the software had to be modified. To adapt the software to work with the secondary sensors the calculated electromagnet power direction was reversed.

14.0 Validation of PID Controller Simulations

The simulation in Figure 39 showed that for Kp = 2, Ki = 0.22 and Kd = 7.69 the system would become stable within the time limit specified by Table 4 in section 4.0 (page 17). Therefore these parameters were chosen as the starting point of operation. The system was tested first using the primary sensor pair and then the secondary pair for both sets of predicted stable and unstable gain values. The results are shown in Table 9.

<table>
<thead>
<tr>
<th>Test No.</th>
<th>Details</th>
<th>Sensor Pair</th>
<th>Simulation Prediction</th>
<th>Actual Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Kp = 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ki = 0.22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kd = 7.69</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Primary</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>System becomes stable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The LM was released and the system stabilised very quickly. When the LM was given a nudge it stabilised efficiently.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Kp = 0.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ki = 2.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kd = 0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Primary</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>System becomes unstable!</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The LM was released and the system oscillated with increasing speed until the LM was thrown from the device.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Kp = 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ki = 0.22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kd = 7.69</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Secondary</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>System becomes stable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The LM was released and slowly moved back and forth on the spot. When the LM was given a nudge it oscillated with increasing speed until the system became unstable and the LM fell. Therefore the gain values were tweaked for the next test.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Kp = 1.58</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ki = 0.18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kd = 9.79</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Secondary</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>System becomes stable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The LM was released and the system stabilised very quickly. When the LM was given a nudge it stabilised efficiently.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 9 shows that the predicted system stability due to both stable and non-stable gain values (Figure 39 and Figure 38) was reflected in real life. This confirmed the simulation results.

The gain values for the secondary sensor pair, however, needed tweaking very slightly in order to produce the same result. It is assumed that the difference in gain values was due to small differences of the secondary sensors in relation to the reference point.

In order to validate the simulations completely the x position of the levitating magnet was tracked using the terminal to display the data on the PC monitor. The information was then imported into Microsoft Excel and analysed.

**14.1 Predicted Unstable Gain Values**

The simulation Figure 38 predicted that the response of the system using $K_p = 0.3$, $K_i = 2$ and $K_d = 0.1$ would be an oscillation of increasing amplitude. This was confirmed during test 2 in Table 9. Figure 50 shows the response of the real system.

![X-axis Position Tracking of Levitating Magnet](image)

*Figure 50: Validation of predicted unstable system*

Figure 50 shows that the magnet was placed into the system after approximately 0.5s. The magnet then goes into instant oscillation of increasing amplitude until approximately 0.9s when it levels out. The levelled out amplitude is equal to that of 0-0.5s where there is no magnet present, therefore the magnet has been flung out of the system. This result validates the Routh-Hurwitz stability criterion derived in section 9.6.4 (page 67).
14.2 Predicted Stable Gain Values

The system was re-calibrated with the stable gain predictions of $K_p = 2$, $K_i = 0.22$ and $K_d = 7.69$ (test 1 conditions) and the tracking was observed Figure 51:

Figure 51: Validation of predicted stable system

Figure 51 shows that the magnet was placed into the system after approximately 0.9s. The position experienced overshoot before settling in approximately 0.2s. After approximately 1.8s it can be seen that the magnet was removed from the system. Figure 51 shows an identical system response to the simulation in Figure 39.
To extensively test the apparent stability of the system the position tracking was observed when the levitating magnet received a number of nudges of varying power. Figure 52 shows the results.

![X-axis Position Tracking of Levitating Magnet With Applied Additional Force](image)

**Figure 52: Validation of predicted stable system with nudges**

Figure 52 shows how the system settles in a similar fashion to Figure 51. It can be seen that over a 1.5s period the levitating magnet receives 3 nudges. After each nudge the magnet responds by settling in an acceptable amount of time. This result completely validates the Matlab Simulink simulations designed in section 9.6.7 (page 72).
15.0 Three Dimensional Levitation Prototype V.2 – Dynamic Redundancy

Prototype v.1 was designed to run using either the primary or the secondary sensor pair. The dynamic redundancy concept discussed in section 5.4.2 was implemented by designing software that could act upon a sensor failure. It is the role of the analytical redundancy (observer) to determine whether a sensor pair has failed, and therefore the role of the dynamic redundancy software is solely to switch the sensor pair software over when a fault is detected.

15.1 Dynamic Redundancy Software

The flowchart representing this software is illustrated by Figure 53 below.

![Figure 53: Dynamic redundancy operation](image)

The flow chart in Figure 53 was used to write software which incorporates dynamic redundancy into the Arduino transmitter and receiver software written in sections 16.2 and 16.6. The dynamic redundancy system software is located in Appendix 8 (page 138).

The gain boost section of Figure 53 was designed to strongly increase the gain of the system temporarily when a sensor fail is detected. When the sensor fail occurs, for the first 500 cycles of the control cycle the integral gain $Ki$ is boosted from 0.13 to 0.5.
15.2 Dynamic Redundancy Software Testing

The dynamic redundancy software was tested as described in Table 10.

Table 10: Dynamic redundancy software testing

<table>
<thead>
<tr>
<th>Test No.</th>
<th>Details</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Levitate the LM using primary sensor pair</td>
<td>The LM was released and the system stabilised very quickly. When the LM was given a nudge it stabilised efficiently.</td>
</tr>
<tr>
<td>2</td>
<td>Activate the primary sensor fail by sending the sensor fail pin high.</td>
<td>The LM instantly wobbled and shook, however remained fixed in both the x-axis and y-axis. After a short while the system was once again completely stable.</td>
</tr>
</tbody>
</table>

Table 10 demonstrates how the system utilised dynamic redundancy to prevent the levitating magnet from flying off the device when the primary sensor pair failed. The same position tracking technique used in section 13.1 was applied and the result is illustrated in Figure 54.

Figure 54: Position tracking when a primary sensor fails

Figure 54 shows that approximately 2.8s an error in the primary sensors was generated (test 2). The software detected the error, temporarily increased the gain, and changed to the secondary pair of sensors. The position of the magnet oscillated violently before settling back to a controlled and stable state.
16.0 Discussion of Levitation Prototype V.1 and V.2 Results

The concept art in section 5.2 was used to build a levitation hardware prototype. The first prototype consisted of primary and backup sensors and one Arduino processor. The mathematical analysis of the linearised system in section 9.6 revealed that the PID controller was the most suitable controller to use in an embedded control system and therefore a PID controller was written for each of the horizontal axes (x and y) in software and programmed onto the Arduino.

In order to compare the prototype’s performance with the predictions made in the mathematical analysis (section 9.0), the position of the levitating magnet was tracked by recording the x-axis sensor readings during operation. Using this technique it was observed that when gain values were used, that did not satisfy the Routh-Hurwitz stability criterion, the magnet position oscillated violently with increasing amplitude (Figure 50). This resulted in the magnet being flung from the system in less than 0.5s. This result validated the corresponding simulation (Figure 33). When appropriate gain values were used the magnet position tracking (Figure 51) showed that stability was achieved in approximately 0.2s. When the identical gain values were simulated (Figure 39) the predicted response was almost a perfect match.

Comparing the results observed by tracking the levitating magnet during operation with the simulated predictions validated not just the PID control simulations, but also the entire mathematical approach used to linearise the system.

When dynamic redundancy was added to the system the tests in section 13.2 (Table 10) showed how the levitating magnet stabilised quickly and efficiently when a sensor fail was detected and the secondary sensor pair was activated. The system used a temporary gain boost because it was predicted that when the primary sensors fail, the magnet would be flung from the device incredibly fast. The gain boost temporarily increased the integral gain (responsible for predicting the future) and was restored after the magnet stabilised.

The relative position of the magnet during the sensor pair fail moved from a value of 200 (stable levitation half way between both electromagnets) to a value of approximately 320. This shows that the magnet had moved away from the reference point before the switchover was complete and the feedback control from the secondary sensor pair started to bring the magnet back. When switching sensor pairs, it was discovered that the instability caused by the switchover could be minimised by temporarily increasing the gain Ki. The position tracking of the sensors failing (Figure 54) shows that the oscillation during the switchover period decreases quickly and therefore it was not required to adjust any of the other gain values.
17.0 Three Dimensional Levitation Prototype V.3 – Hardware Redundancy

The prototype v.2 was improved to incorporate the hardware redundancy specified by the hardware specification (Table 2, page 17). Figure shows prototype V.3:

![Prototype V.3](image)

It can be seen in Figure 55 that a second Arduino was added to the system. The Arduino on the left was used as the master and the Arduino on the right was used as the slave. In this project it was decided that due to time and budget restraints, only two microprocessors would be used to demonstrate hardware redundancy. In a real life system three microprocessors would be used so that a majority vote could be taken and the faulty processor determined. Prototype V.3 was designed to prove only the concept and method behind achieving hardware redundancy and therefore two microprocessors was acceptable.

The master Arduino sent the feedback control data to the slave Arduino using the Serial Peripheral Interface (SPI), which then compared the received data with calculated data. This method proved that data could be sent from one Arduino to another, and then compared. When only two processors are used in this type of system, the question of ‘which processor is at fault?’ could be raised. It was decided that, during this project, making the system work with only two processors communicating via SPI would prove the concept and enable the system to receive a third Arduino, connected to the SPI bus, in the future.
The circuit schematic for prototype v.3 is shown in Figure 56:

Figure 56: Prototype v.3 schematic

Figure 56 illustrates how hardware redundancy is added to the circuit in the form of an additional Arduino processing unit. Pins 50 – 53 on each Arduino represent the SPI communication control ports.
17.1 Development of Hardware Redundancy using SPI Communication

The first version of SPI software needs to test whether the computer control can calculate the controller output power values, send them to an additional processing unit and then output them to the electromagnets fast enough for the system to work. Therefore, in version 1.0 the software will not include the hardware redundancy stage.

17.2 Master Transmitter (Tx) V.1

In version 1.0 the master Tx has the role of determining the control parameters as in section 12.1.1, and then must transmit them via SPI to the receiving Arduino. The flowchart designed to describe this process is shown in Figure 57:

![Flowchart](image-url)  
**Figure 57: SPI master Tx version 1.0**
17.3 Arduino Master Transmitter V.1 Flow Chart Analysis

The flowchart in Figure 57 utilises the following processes:

1. **Enable SPI**: The SPI setup for the Arduino is listed below:
   a) Send slave select high
   b) Initialise SPI mode
   c) Specify SPI clock speed

   The slave select is active low and therefore must remain high until any communications take place.

2. **PID control**: Initiate steps 1 – 5 from section 12.1.2.

3. **Send power values**: To send the x-axis and y-axis power values via SPI the following steps must be taken:
   a) Send the slave select low to select the receiving Arduino
   b) Output a high value to the receiving Arduino that alerts it to the fact that the x-axis power is going to be sent.
   c) Transfer the x-axis power value to the receiving Arduino via SPI
   d) Send the slave select high to de-select the receiving Arduino

4. **Final steps**: Initiate steps 6 and 7 from section 12.1.2.

17.3.1 Arduino Slave Receiver (Rx) V.1

The receiving Arduino must receive the information from the transmitter and output the received power values to the electromagnets. Figure 58 shows the Arduino receiver software flowchart.

Figure 58: SPI slave testing
17.4 Arduino Slave Receiver V.1 Flowchart Analysis

The flowchart in Figure 58 utilises the following processes:

1. **Enable SPI slave mode**: The Arduino receive must be told that it is the slave device and not the master.

2. **Setup interrupt flag and enable interrupts**: The receiver will make use of interrupts. Therefore a variable is assigned to be used as a flag to alert the software to when new data is ready for use. Interrupts must specifically be enabled before they can be used.

3. **Interrupt service routine (ISR)**: The ISR is the routine that will initiate upon receiving an interrupt.
   
   The ISR for this software must therefore do the following things:
   
   a) Disable interrupts: if an interrupt interrupts an ISR the system will crash.
   
   b) Read the xy identifier input
   
   c) Receive the SPI data
   
   d) Re-enable interrupts

4. **Output power to electromagnets**: When both the x-axis data and y-axis power value has been received, output power in the correct direction to each of the electromagnets.

17.5 SPI Software Version 1.0 Testing

The transmitter V1.0 software was written and can be found in Appendix 9 (page 146). The receiver V1.0 software was written and can be found in Appendix 10 (page 151). The system was run and the results are shown in Table 9.

<table>
<thead>
<tr>
<th>Test</th>
<th>Sensor Pair</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculate PID outputs and transmit them using SPI</td>
<td>Primary</td>
<td>The LM was released and the system stabilised very quickly. When the LM was given a nudge it stabilised efficiently.</td>
</tr>
</tbody>
</table>

Table 11 shows that the SPI transfer of the x-axis and y-axis power values was performed correctly and the system ran as if the receiver Arduino was the controller. This result proves that the SPI speed is sufficient for the use of this system. Hardware redundancy can now be incorporated into the system.
17.6 Arduino Slave Receiver (Rx) version 2.0

The hardware redundancy was introduced by adding the PID control to the Arduino receiver. The receiving processor could then compare the power values it had calculated with the data received via SPI from the transmitter. If the received data was incorrect then the receiver would choose to ignore the received data and use the calculated values instead.

The flow chart for the hardware redundancy receiver is illustrated in Figure 59 and contains the following processes:

1. Perform steps 1 – 3 of section 16.4
2. Perform steps 2 – 4 of section 12.1.1
3. Request SPI data from the transmitter
4. Compare received values with calculated values
5. Perform step 4 from 16.4
6. Perform steps 6 and 7 from 12.1.1

Figure 59: SPI Arduino receiver version 2.0
17.7 Arduino Master Transmitter version 2.0

The transmitting Arduino software needs to be modified in order to transmit the PID values when the receiver is ready for them. To achieve this, the transmitter must wait until the control signals ‘x_ready’ and ‘y_ready’ go high before it sends the relevant data. The amended flowchart for this software is shown in Figure 60.

By including the wait until ready sections of code, the master Arduino will only send the calculated PID power values when requested by the slave.
17.8 SPI Software Version 2.0 Testing

The transmitter software V.2 can be found in Appendix 11 (page 153) and the receiver software v2 can be found in Appendix 12 (page 157). The software was downloaded to the system and was once again tested. The result is shown in Table 12.

<table>
<thead>
<tr>
<th>Test</th>
<th>Sensor Pair</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculate PID outputs and transmit them using SPI to a slave device which has also calculated the desired PID outputs.</td>
<td>Primary</td>
<td>The LM was released and the system stabilised very quickly. When the LM was given a nudge it stabilised efficiently.</td>
</tr>
<tr>
<td>Adjusted the master Arduino software to add a value of 20 to the output of the PID power calculations.</td>
<td>Primary</td>
<td>The slave Arduino ignored the data from the master Arduino because it was significantly different to its own calculations. Levitation remained stable.</td>
</tr>
</tbody>
</table>

The result in Table 12 shows that the hardware redundancy was successfully implemented in the levitation system.

18.0 Combining Hardware & Dynamic Redundancy

The transmitter and receiver software versions 2.0 were combined with dynamic redundancy to produce versions 3.0 (Appendix 13, page 163, and Appendix 14, page 170). Testing results are shown in Table 13.

<table>
<thead>
<tr>
<th>Test</th>
<th>Sensor Pair</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>While using SPI communication, send the error pins of both Tx and Rx Arduinos high.</td>
<td>Primary then Secondary</td>
<td>The LM was released and the system stabilised very quickly. When the LM was given a nudge it stabilised efficiently.</td>
</tr>
<tr>
<td>While using the secondary pair of sensors the master Arduino software added a value of 20 to the output of the PID power calculations.</td>
<td>Secondary</td>
<td>The slave Arduino ignored the data from the master Arduino because it was significantly different to its own calculations. Levitation remained stable.</td>
</tr>
</tbody>
</table>
19.0 Temperature Monitoring

Electronic components and systems are designed to work within a specified temperature range. If the temperature becomes too low, it can affect the way a circuit works. For example, amplifier gain may change. If the temperature becomes too high, components may fail early or suddenly without warning, or even catch fire and give out potentially poisonous fumes.

Different companies and component manufacturers produce components and systems for different temperature ranges. Altera (2014), who manufacture Field Programmable Gate Arrays in Silicon Valley, uses the following temperature range classifications for their components:

- Commercial: 0°C to 85°C
- Industrial: −40°C to 100°C
- Automotive: −40°C to 125°C
- Extended: −40°C to 125°C
- Military: −55°C to 125°C

For the prototype, it was decided that the commercial specification components were to be used. Therefore the design temperature range was limited to +5°C to +60°C, which incorporates a safety margin either side of the temperature limits. A larger margin of safety was used for high temperatures than for lower temperatures. When temperatures are detected outside of this range then warnings should be given. It was decided that although the commercial specification was suitable for the prototype, if a larger system is designed for purposes other than small demonstrations, the most appropriate specification should be chosen.

Throughout the prototype design heat has already been reduced by:

- Choosing energy efficient components that operate with low voltages and currents minimising heat production.
- Using good design principles to avoid the unnecessary use of energy when it is not required. For example, powering down the electromagnets when a permanent magnet is not detected close to them.
- Using good software design to ensure that the controlled components are operated at the right time, at the right voltages and at the right frequency.
- Using passive heat sinks to dissipate any heat generated.

During long operation times the small electromagnets became hot. The risk to a big system with large electromagnets is therefore exacerbated. Therefore temperature monitoring was designed and incorporated into the design so that this risk was heavily reduced.
The temperature monitoring circuit diagram is shown in Figure 61.

Figure 61: Temperature control circuit diagram

Figure 61 comprises of a DHT11 Temperature Sensor connected to an Arduino control processor. The DHT11 sends, upon demand, a 16bit reading of temperature and humidity to the Arduino. These readings are used to control a small 12V DC fan using a PWM output and simple transistor circuit. The circuit was built and is shown in Figure 62.

Figure 62: Temperature control circuit

Control software (located in Appendix 15, page 180) was written using the flow chart in Figure 63. The software monitors and controls the air temperature within the system based on the operating temperature range specified in the hardware specification (Table 2, page17).
The temperature control uses pulse width modulation (PWM) to increase the speed of the fan as the temperature exceeds 20°C. When the temperature rises to above 60°C the system displays a warning. This warning can be used to shut the system down in a safe and controlled manner if required. The circuit was tested by putting the sensor near a heat source (desk lamp). The circuit testing is described by Table 14.

Table 14: Temperature monitoring circuit testing

<table>
<thead>
<tr>
<th>Test No.</th>
<th>Details</th>
<th>Expected Result</th>
<th>Actual Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The sensor was kept cool</td>
<td>Fan to stay off</td>
<td>Fan stayed off</td>
</tr>
<tr>
<td>2</td>
<td>The sensor temperature was increased to above 20°C</td>
<td>Fan turns on</td>
<td>Fan turned on</td>
</tr>
<tr>
<td>3</td>
<td>The sensor temperature was increased from 20°C to 30°C</td>
<td>Fan speed should increase proportionally</td>
<td>Fan speed increased proportionally</td>
</tr>
<tr>
<td>4</td>
<td>The sensor temperature was increased to 65°C</td>
<td>Warning message should appear in the terminal</td>
<td>Warning message appeared in the terminal</td>
</tr>
</tbody>
</table>

Table 14 reveals that each of the four tests displayed positive results. The temperature control can therefore be used to successfully prevent system failure caused by the operating temperature being outside its limits.
20.0 Discussion of Levitation Prototype V3 Results

The concept of hardware redundancy was introduced to the system in two stages:

- Introduce SPI technology and interrupt service routines
- Compare the received data with another value

When the SPI technology was initially introduced, the software testing result (Table 11) shows that interrupts were used by the receiver Arduino, in order to successfully to read both x and y axis PID output data. These values were then used it to successfully levitate the magnet. This result proved that the system was fast enough to stabilise the levitating magnet, even when transmitting data between processors. If this test had shown that the system was not fast enough with SPI then hardware redundancy of the embedded system would not have been possible with this hardware.

After proving that the SPI communication was fast enough for the requirements of the system the slave Arduino software was modified to also calculate its own PID output data. It was a concern that the system might still not be fast enough to levitate the magnet because the software contained a short delay during the comparison of calculated and received data. However, the results in Table 12 shows that this concern was not an issue. The magnet remained in stable levitation while the processors communicated with each other, and the slave compared two sets of data. The tests in Table 12 went on to show how the data from the transmitting Arduino was used when it matched the slave Arduinos data. When the data was not a match the slave Arduinos data was used to control the levitation.

This result would be acceptable without further work if it is assumed that the slave Arduino could never fail. In this situation the slave Arduino would always produce the correct data and therefore make the transmitting Arduino redundant. However, in reality the slave Arduino could also fail. Therefore, for complete hardware redundancy at least three processors are required. In this situation if any one of the three Arduino processors should fail, the output from the failed arduino could be isolated using a majority voting system. It was decided at the beginning of section 17.0 that during this project the concept of hardware redundancy could be proved with the use of only two Arduinos and improved in the future to include an additional Arduino if testing was successful. All testing performed on prototype v.3 was successful and therefore proved that extra hardware redundancy can be fully implemented into the system in the future.

The temperature control circuitry designed in section 0 successfully adjusted the speed of a pwm fan according to the temperature of a heat source. The temperature control circuitry was programmed to the commercial component temperature range. It was therefore suitable for the levitation system.
21.0 Prototype V3 User Documentation

To successfully setup, program and run the prototype the following steps should be taken:

21.1 Setup Prototype

1) Place the permanent magnet to one side, at a safe distance from the system.
2) Provide power to both the master and slave Arduino controllers, in any order. Power can be provided either by using a 9V battery for each Arduino, or connecting them to a laptop via USB adapter cables.
3) Provide power to the electromagnet drivers by connecting the 12V switched power supply to the 12V power input.
4) Turn on the electromagnet power drivers by pressing in the white button.
5) Check that, while no magnet is placed near the prototype, the output power of the system is zero. This is the case if the prototype is silent, and no green lights are flashing on the drivers. If the power output of the system is not zero then turn the drivers off immediately, there is a fault with the circuit.
6) The prototype is now safe to program and use.

21.2 Install / Upload Software

1) Install and run the Arduino software.
2) Open the following softwares:
   a. Arduino_Master_Hardware_Dynamic.ino
   b. Arduino_Slave_Hardware_Dynamic.ino
3) Identify which serial port is assigned by the computer to each of the Arduino controllers.
4) Verify each of the softwares using the verify button and then upload them to their respective Arduino controller.
5) Wait until the software confirms a successful upload before running the system.

21.3 Running the System

1) Hold the permanent magnet approximately in the middle of the 4 electromagnets for approximately 5 seconds. During this period the integral term builds and it should be felt how the magnet becomes quickly pulled into the centre.
2) When the magnet feels stable let go and prepare for it to be ejected should stable levitation not have been achieved. Check for a green LED to confirm that there is no sensor fault. To produce a manual sensor fail press the test button by the green LED. The red LED should become lit, whilst the sensor pair changes to the redundant pair. The system should stabilise quickly. To reset the fault press the other button. The green LED should now be lit once more.
22.0 Project Management

22.1 Activities Overview

The project was divided down into the following three main sections:

- **Mathematical model**: This section included the mathematical calculations used to describe the levitating objects position in free space. State variables and differential equations were used to produce a state-space representation.

- **Software model and simulation**: The mathematical equations from the Mathematical Model section was used to model the system in the advanced computer software: Matlab Simulink.

- **System hardware, software design and production**: The hardware was carefully chosen, and circuit diagrams designed. Microprocessors were programmed to carry out the digital control and chosen fault tolerant design approaches.

Each of the three main sections was responsible for a large list of activities. These activities are displayed individually in the Gantt chart shown in Figure 64, page 106.

22.2 Milestone Deadlines

Over the course of this project there were several major deadlines that had to be met in order to show professionalism, efficiency and reliability. Table 15 identifies the major deadlines that were associated with the project. These deadlines are also identified in the Gantt chart (Figure 64, page 106).

<table>
<thead>
<tr>
<th>Table 15: Milestone Deadlines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Task</td>
</tr>
<tr>
<td>Project plan with health and safety analysis</td>
</tr>
<tr>
<td>Progress report</td>
</tr>
<tr>
<td>Poster submission</td>
</tr>
<tr>
<td>Project thesis submission</td>
</tr>
<tr>
<td>Project Presentation</td>
</tr>
<tr>
<td>Project Viva</td>
</tr>
</tbody>
</table>
22.3 Gantt Chart Version 1.0

Gantt project software was used to create the first version of the project Gantt chart (Figure 64). The chart shows the expected duration and completion dates for each task in the project. Milestone deadlines specified in Table 15 are shown as diamonds. The critical path is also shown.

Figure 64: Project Gantt Chart version 1.0
22.4 Gantt Chart (Revised) Version 2.0

At the end of semester 1 the initial Gantt chart in Figure 64 was revised at the end of semester 1. The revision is shown in Figure 65.

Figure 65: Gantt chart version 2.0
22.5 Gantt Chart Version 2.0 Revision Details

In engineering projects it is expected that not all tasks will be carried out in the exact order in which they were planned. However, it was important that the project milestones were not missed or delayed. It was also essential that all tasks located on the critical path were not delayed. This would have resulted in the entire project completion date being delayed. Therefore the state of the project was analysed and the following revisions were appropriately made:

22.5.1 Revisions Made

Upon analysis of the initial Gantt Chart the following revisions were made:

- **Mathematical Model:**
  The mathematical model was changed from a duration of 7 days to a duration of 38 days. This change was made due to the realisation that the mathematical equations being derived were of a much greater complexity than originally expected.

- **Labview Model:**
  It was decided that the task of building a Labview model was originally located in the wrong place in the time line. It was decided that until a simulation of the model had been built there was no justification for starting a Labview model.

- **Simulink Simulation:**
  The start of the simulation task was moved back a day due to the repositioning of the Labview task. This section was removed from the critical path so that a delay in this section would not delay the completion of the project.

- **Basic System Production:**
  This section was moved back to an initial starting of 25/11/2013. Bringing the basic system production section forward stopped it clashing with the new building of the Labview model time allocation.
22.6 Gantt Chart (Final) Version 3.0

After the project had been completed a final version of the Gantt chart (Figure 67) was produced that displayed the true activity durations of the project.

Figure 66: Gantt chart version 3.0
22.7 Project Completion Progress Overview

The final version of the project Gantt chart, (version 3.0, Figure 67), contained only two revisions from Gantt chart version 2.0 (Figure 65).

Firstly, the decision was made to remove the activity labelled “utilise Lab view software”. The reason behind this revision was to concentrate all focus on the Matlab and Simulink simulation activities. It can be seen in the Gantt chart version 2.0 (Figure 65) that the Labview activity did not lie on the critical path. This is because the idea behind the activity was to fit it into the project as an addition to Matlab. Because the activity was not on the critical path it did not delay the project completion date by removing this section from the project.

Secondly, the decision to remove the observer design in C language was due to the realisation that in the time restraints of the project this activity was too complex. The activity was located on the critical path and therefore if delayed it would have delayed the entire project. Therefore the logical decision was to remove it from the list of proposed second semester activities.

Version 3.0, Figure 67, also illustrates the true activity durations. It is shown that 4 activities were completed ahead of schedule in the second semester (all of the project time after the Christmas holidays). All other tasks were completed on time. Because of the four activities completed ahead of schedule the thesis was actually finished 14 days ahead of schedule.

22.8 Review of Project Aims and Objectives

The aims and objects of the project (Table 1, section 3.0) were originally specified in the project plan and should be used to measure the success of the project upon completion.

22.8.1 Aim 1: Achieve stable magnetic levitation of an object using magnetic repulsion

Aim 1 was fully achieved when objectives 1 - 6 were successfully completed as follows:

- The detailed mathematical model derived throughout section 6.0 (pages 31 to 48) fully achieved objective 1: 'Produce a mathematical model of the levitation system'. The revised Gantt Chart (Figure 65) shows how this task was completed ahead of schedule.

- The mathematical model derived throughout section 6.0 was used to produce a non-linear simulation in Simulink. The model was then linearised to produce a linear state-space representation of the system. This state-space representation was modelled in Matlab and Simulink to produce several simulations. The linear system was simulated with no control, proportional control, pole placement control, PID control and variable state feedback control. Therefore objective 2: 'Produce a software model of the system' was fully achieved.
• The concept art (Figure 3 and Figure 4, page 20), together with the levitation device prototype v.1 (Figure 47, page 81) fully achieved objective 3: 'Produce a mathematical model of the levitation system'. The revised Gantt Chart, Figure 67, illustrates how these tasks were completed on and ahead of schedule.

• Section 12.1.1 fully achieved objective 4: 'Design a PID controller in C programming language' and objective 5: 'Program the PID controller onto the system' by designing, coding and extensively testing a PID in the C programming language.

The percentage completion of aim 1 was calculated by adding together the relative completion percentages of objectives 1-5 (Equation [127]):

\[
\text{Aim 1 completion} = \frac{ob.1\%}{5} + \frac{ob.2\%}{5} + \frac{ob.3\%}{5} + \frac{ob.4\%}{5} + \frac{ob.5\%}{5} \tag{Equation [127]}
\]

\[
\therefore \text{Aim 1 completion} = \frac{100\%}{5} + \frac{100\%}{5} + \frac{100\%}{5} + \frac{100\%}{5} + \frac{100\%}{5} = 100\%
\]

22.8.2 Aim 2: Detect any numerical errors provided by the sensor feedback

Aim 2 consisted of objectives 6 – 8. The final progress of aim 2 was as follows:

• Objective 6: 'Research observer' was given 3 days duration in the semester 2 prediction Gantt chart (Figure 65, page 107). Three days were officially spent on the completing this activity, however, as this activity was research based it was logical to unofficially spend much more time reading about observers. Therefore objective 6 was fully achieved.

• The software research from objective 6 was used to produce an observer model in Matlab Simulink. Section 9.7 (page 73) illustrates how the observer was modelled and simulated in Simulink, therefore fully achieving objective 7: 'Simulate an observer design'.

• Objective 8: 'write the observer in c program' was not completed. The reasons for omitting this activity from the project were discussed in detail in section 20.8, page 110.

The percentage completion of aim 2 was calculated by adding together the relative completion percentages of objectives 6-8 (Equation [128]):

\[
\text{Aim 2 completion} = \frac{ob.6\%}{3} + \frac{ob.7\%}{3} + \frac{ob.8\%}{3} \tag{Equation [128]}
\]

\[
\therefore \text{Aim 2 completion} = \frac{100\%}{3} + \frac{100\%}{3} + \frac{0\%}{3} = 66\%
\]
22.8.3 Aim 3: Produce a fault tolerant system

Aim 3 consisted of objectives 8 – 12 which focused strongly on the fault tolerance aspect of the project. The final completion of aim 3 was as follows:

- Objective 9: ‘Research and implement static (hardware) redundancy’ was worked towards and fully achieved on time. Section 14.0, page 89, described how SPI communication was implemented into the system design. Hardware redundancy is usually made using 1 or 2 additional microprocessors and comparison logic. In this project just 1 additional microprocessor was used. This leaves room for future improvement of the system.

- Objective 10: ‘Research dynamic redundancy’ was not only achieved but further developed and utilised within the system. Section 17.0, page 99, discussed in great detail how an extra pair of sensors was utilised to perform extensive dynamic redundancy.

- Objective 11: ‘Research software error masking’ was fully achieved when an algorithm to eliminate random noise affecting sensor readings was developed. Section 12.1.2, page 84, described the algorithm that involves taking the average of 5 sensor readings. This masks any random error affecting a few of the readings.

- Objective 12: ‘Implement temperature monitoring’ was achieved by the development described by section 0, page 99.

The percentage completion of aim 3 was calculated by adding together the relative completion percentages of objectives 9-12 (Equation [129]):

\[
\text{Aim 3 completion} = \frac{\text{ob. 9}\%}{4} + \frac{\text{ob. 10}\%}{4} + \frac{\text{ob. 11}\%}{4} + \frac{\text{ob. 12}\%}{4}
\]

\[
\therefore \text{Aim 2 completion} = \frac{100\%}{4} + \frac{100\%}{4} + \frac{100\%}{4} + \frac{100\%}{4} = 100\%
\]
22.8.4 Overall Project Completion

The total percentage of the project completion can be calculated using Equation [130]:

\[
Project \ completion = \frac{\text{Sum} (\text{objectives 1 to 12 percentage completion})}{12}
\]

\[\therefore Project \ completion = \frac{1100}{12} = 91.7\%
\]

In total 11 out of 12 objectives were 100% completed. Objective 8 was realised to be too ambitious and was therefore not completed due to removing it from the activity list. It can be concluded from the in depth analysis in section 20.9 that this project has substantially achieved the aims and objectives.
23.0 Prototype hardware Review

In section 4.1, page 17, a full hardware specification for the levitation device was described by Table 2. In order to measure the success of the prototype described in section 11.0 the specification must be compared with the attributes of the prototype. Table 18 shows the prototype hardware to specification comparison.

Table 16: Prototype hardware specification review

<table>
<thead>
<tr>
<th>Specification</th>
<th>Requirement</th>
<th>Prototype Analysis</th>
<th>Achieved?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Redundancy</td>
<td>Additional “backup” sensors that will be used as soon as the primary pair of sensors fail.</td>
<td>The prototype initially runs using the primary pair of hall effect sensors. The prototype also contains a secondary pair of sensors that are always connected to the computer control.</td>
<td>✔️</td>
</tr>
<tr>
<td>Hardware Redundancy</td>
<td>At least one additional microprocessor for calculation comparison and error masking.</td>
<td>The prototype was tested using two Arduino microprocessors. The master Arduino sends the calculated PID outputs to the Slave Arduino which then outputs to the electromagnets.</td>
<td>✔️</td>
</tr>
<tr>
<td>Temperature Control</td>
<td>A temperature sensor should be used to control a fan.</td>
<td>Temperature control was achieved using a DHT11 sensor and control software.</td>
<td>✔️</td>
</tr>
</tbody>
</table>

Table 18 reveals that all three hardware requirements of the system were met by the prototype design.
24.0 Prototype Software Review

In section 4.1, page 17, a full software specification for the levitation device was described by Table 3. It was reviewed in section 21.0 that the prototype hardware fully met the specification. For the prototype to be successful, however, it must also meet the software specification. Table 19 shows how the prototype software compares to the specification.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Requirement</th>
<th>Prototype Analysis</th>
<th>Achieved?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levitation Control</td>
<td>A controller must be designed to achieve stable levitation.</td>
<td>Two individual PID controllers were realised in the prototype software in order to control the x-axis and y-axis positions separately. The gain values were used from the simulated PID controller design and tested to achieve successful levitation.</td>
<td>✔️</td>
</tr>
<tr>
<td>Dynamic Redundancy</td>
<td>Software should be developed that can switch between sensor pair programs.</td>
<td>The prototype initially runs using the primary pair of hall effect sensors. When the software detects a fault the system switches to using the secondary pair of sensors. During the transition the gain is boosted temporarily to maintain stable levitation.</td>
<td>✔️</td>
</tr>
<tr>
<td>Hardware Redundancy</td>
<td>SPI should be integrated into the controller design to compare control values.</td>
<td>The SPI protocol was successfully tested in the prototype to transmit and receive both the x-axis and the y-axis PID control values. These values were then successfully used to achieve stable levitation.</td>
<td>✔️</td>
</tr>
<tr>
<td>Analytical Redundancy</td>
<td>Observers should compare the sensor feedback with calculated position values.</td>
<td>Due to time constraints and the complexity of achieving this activity, the observer software in C language was omitted from this project.</td>
<td>❌</td>
</tr>
</tbody>
</table>

Table 19 shows that 3 out of 4 software requirements were met. The analytical redundancy in software proved to take more time than was predicted in the Gantt chart (Figure 67). The prototype fully demonstrates 3 out of 4 fault tolerant techniques, with room for future work on the software observer.
25.0 Conclusions

There are a large number of reasons why a large electrical system might experience temporary or permanent failure. When focusing on electromagnetic levitation technology it is clear that safety is of vital importance. It is concluded that fault tolerant techniques, in the form of both hardware and software, can be successfully implemented to prevent a system from failing. Error masking techniques, such as hardware redundancy and software comparison logic, can be used to identify and ignore any errors produced by faulty processors. Analytical redundancy identifies any errors in sensor readings and alerts the system. Dynamic redundancy, in the form of backup sensors, can be used to immediately disable a faulty pair of sensors and commence operation utilising the secondary pair. This technique mitigates sensor error completely.

From inspection of the design concept in section 6.0, it was recognised that the repulsion force exerted from a ring magnet is produced from diverging flux lines. By deriving a full mathematical hypothesis of the system it was deduced that the natural response of the system, due to the ring magnet, would become extremely unstable over a short amount of time.

The non-linear mathematical model of the system was successfully linearised to produce a linear state-space representation. Matlab was used in co-operation with Simulink to mathematically design and simulate the response of the system with a number of controllers. It was confirmed that the open-loop system is naturally unstable. A proportional feedback controller could not be used to stabilise the system due to the closed-loop poles not being negative. The pole-placement controller worked well at stabilising the system; however, the gain value required by the controller was too high for an embedded processor to use. It was found that the Ziegler-Nichols PID tuning method produced simply a rough estimate of gains that would control the system in a stable fashion, and with low gain values. A Routh-Hurwitz stability criterion was useful in confirming the stability boundaries of the system. The PID controller could be tuned more easily using both of these methods as a starting point. Using C programming and careful design, a levitation prototype was successfully built to demonstrate a high level of fault tolerance. When tested, the use of hardware redundancy (extra computer control) in a levitation prototype completely eliminated the risk of system failure if the primary processor ever failed. Dynamic redundancy was also tested to be a success: the system could safely keep a levitating magnet stable during the process of a sensor failure. The observer design was investigated in Matlab Simulink and found to be a useful tool in identifying position error measured by any sensor.

The outcome of this project could definitely be used to further studies into three dimensional levitation systems utilising magnetic repulsion technology. The ability to program the computer controller in the C language leaves the future open for various other controllers to be designed using Matlab and programmed into the system.
26.0 Future Work

The future development work required to further develop this project contains the following main aspects:

26.1 Further development of the observer design

The observer designed in this project was tested only in Matlab Simulink software. A good indication was given that the observer could be used in the real life system. Observer software in C language could be written in the future in order to control the system. This would not only increase the performance of the system but also the desirability: the fault tolerant design would be complete.

26.2 Improved hardware redundancy

The hardware redundancy utilised by the levitation system built in during this project could be further improved with the addition of a second Arduino slave connected to the SPI bus. The system could then use all three processing units to increase the efficiency of the system response to microprocessor failure.

26.3 User interactivity

The user interactivity with the system could be improved a vast amount. For example, a touch screen could be used to test for faults instead of mechanical buttons. An LCD could also be used to display the current status of the system to user. If the system detects faults then the LCD could display warnings in the form of error reports.
27.0 References

27.1 Books


27.2 Websites


### 27.3 Journals


27.4 Articles

27.5 PDF Documents

27.6 University Notes

27.7 Dissertation and Thesis Papers
APPENDIX 1: System Parameters

The ring magnet parameters are shown in Table 20 and the parameters of each electromagnet are shown in Table 21.

Table 18: Ring magnet parameters

<table>
<thead>
<tr>
<th>Material</th>
<th>Magnet Grade</th>
<th>Residual Flux Density 'B_r' (T)</th>
<th>Outer Diameter '2R' (mm)</th>
<th>Inner Diameter '2r' (mm)</th>
<th>Length 'L' mm</th>
<th>Pole Area (m²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neodymium Alloy (NdFeB)</td>
<td>N35</td>
<td>1.23</td>
<td>130</td>
<td>110</td>
<td>30</td>
<td>3.77x10⁻³</td>
</tr>
</tbody>
</table>

Table 19: Prototype v.1 electromagnet parameters

<table>
<thead>
<tr>
<th>Outer Diameter (mm)</th>
<th>Bolt Width (mm)</th>
<th>Height (mm)</th>
<th>Wire Diameter (mm)</th>
<th>Number of Turns</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>6</td>
<td>10</td>
<td>0.3</td>
<td>759</td>
</tr>
</tbody>
</table>
APPENDIX 2: Open-Loop Simulation in Matlab

% System Parameters
u0 = 1.257*10^-6;
N = 759;
i0 = 644.1*10^-3;
Area = 3.77*10^-3;
Wx = 2*10^-2;
x0 = 0.0001;
h = 10*10^-3;
m = 0.1;

% calculate a1
% a1 = d/dx(Fc1) where x=x0 and i=i0
a1 = -(u0*((N*i0)^2)*Area)*((2*h^2)*(Wx+x0)))/((h^4)+(2*h^2)*(Wx+x0)^2)+(Wx+x0)^4))

% calculate a2
% a1 = d/di(Fc2) where x=x0 and i=i0
a2 = -(u0*((N*i0)^2)*Area)*(-2*(Wx-x0))/((h^4)+((2*h^2)*(Wx-x0)^2)+(Wx-x0)^4))

% calculate b1
% b1 = d/di(Fc1) where x=x0 and i=i0
b1 = ((h^2)+(Wx)^2)*((2*i0*u0*Area*N^2))/((h^4)+((2*h^2)*Wx^2)+(Wx^4))

% calculate b2
% b1 = d/di(Fc2) where x=x0 and i=i0
b2 = ((h^2)+(Wx)^2)*((2*i0*u0*Area*N^2))/((h^4)+((2*h^2)*Wx^2)+(Wx^4))

% calculate lamda 1
l1 = (a1 + a2)/m

% calculate lamda 2
l2 = (b1 + b2)/m

% state space
A = [0 1; l1 0];
B = [0 ; 12];
C = [1 0];
D = 0;

[num, den] = ss2tf(A,B,C,D)
plant = tf(num, den)

% eigen values
eigenvalues = eig(A)

% Observerbility
Mo = [C ; C*A]
M0_Rank = rank(Mc)

% system is observerable
% controllability
Mc = [B A*B]
MC_Rank = rank(Mc)

% uncontrolled system response

t = 0:0.01:2;
u = zeros(size(t));
xt0 = [0.01 0];

sys = ss(A,B,C,0);
[y,t,x] = lsim(sys,u,t,xt0);
plot(t,y)
title('Open-Loop Response to Non-Zero Initial Condition')
xlabel('Time (sec)')
ylabel('Ball Position (m)')

% note that the position of the ball continues to move infinitely away
APPENDIX 3: Pole-Placement Simulation in Matlab

% System Parameters
u0 = 1.257*10^-6;
N = 759;
i0 = 644.1*10^-3;
Area = 3.77*10^-3;
Wx = 2*10^-2;
x0 = 0.0001;
h = 10*10^-3;
m = 0.1;

% calculate a1
% a1 = d/dx(Fc1) where x=x0 and i=i0
a1 = -(u0*((N*i0)^2)*Area)*(2*(Wx+x0))/((h^4)+(2*h^2)*(Wx+x0)^2)+((Wx+x0)^4))

% calculate a2
% a1 = d/di(Fc2) where x=x0 and i=i0
a2 = -(u0*((N*i0)^2)*Area)*(-2*(Wx-x0))/((h^4)+(2*h^2)*(Wx-x0)^2)+((Wx-x0)^4))

% calculate b1
% b1 = d/di(Fc1) where x=x0 and i=i0
b1 = ((h^2)+(Wx)^2)*(2*i0*u0*Area*N^2)/((h^4)+(2*h^2)*Wx^2)+(Wx^4))

% calculate b2
% b1 = d/di(Fc2) where x=x0 and i=i0
b2 = ((h^2)+(Wx)^2)*(2*i0*u0*Area*N^2)/((h^4)+(2*h^2)*Wx^2)+(Wx^4))

% calculate lamda 1
l1 = (a1 + a2)/m

% calculate lamda 2
l2 = (b1 + b2)/m

% state space
A = [0 1; 40 0];
B = [0 ; 140];
C = [1 0];
D = 0;

[num, den] = ss2tf(A,B,C,D)

plant = tf(num, den)

% eigen values
eigenvalues = eig(A)
% Observerbility
Mo = [C ; C*A]
M0_Rank = rank(Mo)
% system is observerble
%controllability
Mc = [B A*B]
MC_Rank = rank(Mc)
% system is controllable
% Pole Placement Technique
sys = ss(A,B,C,0);

p1 = -8; % -80 for part 2
p2 = -10; % -100 for part 3
K = place(A,B,[p1 p2])
sys_cl = ss(A-B*K,B,C,0);

% Simulation

[t, u, xt0] = lsim(sys_cl,u,t,xt0);
xlabel('Time (sec)')
ylabel('Ball Position (m)')
APPENDIX 4: Ziegler-Nichols Method in Matlab

% System Parameters
u0 = 1.257*10^-6;
N = 759;
i0 = 644.1*10^-3;
Area = 3.77*10^-3;
Wx = 2*10^-2;
x0 = 0.0001;
h = 10*10^-3;
m = 0.1;

% calculate a1
% a1 = d/dx(Fc1) where x=x0 and i=i0
a1 = -(u0*((N*i0)^2)*Area)*((2*(Wx+x0))/(h^4)+((2*h^2)*(Wx+x0)^2)+((Wx+x0)^4))

% calculate a2
% a2 = d/di(Fc2) where x=x0 and i=i0
a2 = -(u0*((N*i0)^2)*Area)*(-2*(Wx-x0))/(h^4)+((2*h^2)*(Wx-x0)^2)+((Wx-x0)^4))

% calculate b1
% b1 = d/di(Fc1) where x=x0 and i=i0
b1 = ((h^2)+(Wx)^2)*(2*i0*u0*Area*N^2)/(h^4)+((2*h^2)*Wx^2)+(Wx^4))

% calculate b2
% b2 = d/di(Fc2) where x=x0 and i=i0
b2 = ((h^2)+(Wx)^2)*(2*i0*u0*Area*N^2)/(h^4)+((2*h^2)*Wx^2)+(Wx^4))

% calculate lamda 1
l1 = (a1 + a2)/m

% calculate lamda 2
l2 = (b1 + b2)/m

% state space
A = [0 1; l1 0];
B = [0 ; 12];
C = [1 0];
D = 0;

[num, den] = ss2tf(A,B,C,D)
plant = tf(num, den)
% Ziegler-Nichols Method for PID Tuning

% Impulse response with closed loop gain K

Kp = 3;
Ki = 0;
Kd = 0;

pid_controller = pid(Kp,Ki,Kd);

H = feedback(plant,pid_controller);

t = 0:0.01:10;
impulse(H,t)
axis([0, 0.5, -8, 8]);
title('Response of Magnet Position to an Impulse under PID Control: Kp = 3, Ki = 0, Kd = 0');
xlabel('Time (sec)')
ylabel('Ball Position (mm)')

% Tuning Stage

T0 = 0.32;
K0 = Kp;
Tuned_Kp = 0.6*K0
Tuned_Ki = 2*Tuned_Kp/T0
Tuned_Kd = Tuned_Kp*T0/8
APPENDIX 5: Observer Simulation in Matlab

% System Parameters
u0 = 1.257*10^-6;
N = 759;
i0 = 644.1*10^-3;
Area = 3.77*10^-3;
Wx = 2*10^-2;
x0 = 0.0001;
h = 10*10^-3;
m = 0.1;

% calculate a1
% a1 = d/dx(Fc1) where x=x0 and i=i0
a1 = -(u0*((N*i0)^2)*Area)*((2*(Wx+x0))/((h^4)+(2*h^2)*(Wx+x0)^2)+(Wx+x0)^4));

% calculate a2
% a2 = d/di(Fc2) where x=x0 and i=i0
a2 = -(u0*((N*i0)^2)*Area)*(-2*(Wx-x0))/((h^4)+(2*h^2)*(Wx-x0)^2)+(Wx-x0)^4));

% calculate b1
% b1 = d/di(Fc1) where x=x0 and i=i0
b1 = ((h^2)+(Wx)^2)*((2*i0*u0*Area*N^2)/((h^4)+(2*h^2)*Wx^2)+(Wx^4));

% calculate b2
% b2 = d/di(Fc2) where x=x0 and i=i0
b2 = ((h^2)+(Wx)^2)*((2*i0*u0*Area*N^2)/((h^4)+(2*h^2)*Wx^2)+(Wx^4));

% calculate lamda 1
l1 = (a1 + a2)/m;

% calculate lamda 2
l2 = (b1 + b2)/m;

% state space
A = [0 1; l1 0];
B = [0 ; 12];
C = [1 0];
D = 0;
p = [-8 -10];

% Feedback
K = place(A,B,p);
system = ss(A-B*K,B,C,0);
ClosedloopEigenvalues = eig(A-B*K)
% Calculate gain matrix L
% Specify observer poles
Op = [-40 -50]
L = place(A',C',Op)'
% Define composite system
At = [ A -B*K B*K zeros(size(A)) A-L*C ]
Bt = [ B zeros(size(B)) ];
Ct = [ C zeros(size(C)) ];
sys = ss(At,Bt,Ct,0)
% Simulate observer
n = 2;
e = x(:, n+1:end);
x = x(:, 1:n);
x_est = x - e;
% Save state variables and plot them
x_pos = x(:, 1); x_pos_dot = x(:, 2);
x_pos_est = x_est(:, 1); x_pos_dot_est = x_est(:, 2);
plot(t, x_pos, '-r', t, x_pos_est, ':r', t, x_pos_dot, '-b', t, x_pos_dot_est, ':b')
xlabel('Time (sec)')
ylabel('Amplitude')
title('Real states vs predicted states')
APPENDIX 6: Primary Sensor Pair PID Software

/********************************************************
    Magnetic Leviation PID Controller
    By James Ballard
********************************************************/

// New values are: Kp = 2.00, Ki = 0.22, Kd = 7.69
// Default setPtX = 218  setPtY=163

int pinX=1;
int pinY=0;

int HallX[5], HallY[5];  //Hall effect reading arrays
int MaxX, MaxY, MinX, MinY;  //Error control variables
int AveHallX=0, AveHallY=0;  //Average hall sensor readings

int varsetPtX;
int varsetPtY;

int setPtX=167;  // Hall sensor value for equilibrium state
int setPtY=150;

int in1=11;
int in2=10;
int ena=13;  // PWM pins for y coils

int in3=8;
int in4=7;
int enb=4;  // PWM pins for x coils

int errorX;  // midValue - current X hall sensor reading
int prevErrX=0;  // Previous error used to calc derivative
int dErrorX;  // Derivative of error in X

int errorY;  // midValue - current Y hall sensor reading
int prevErrY=0;  // previous error for calculating derivative
int dErrorY;  // derivative of error in Y

int powerX;  //PWM value driving X coil
int powerY;  //PWM value driving Y coil

float Kp=2;  // Proportional weighting
float Ki=0.22;  // Derivative weighting
float Kd=7.69;  // Integral weighting

float varKp = 0;  // Tuning variables (external)
float varKl = 0;
float varKd = 0;

float SumX = 0;  // Sum of XY readings
float SumY = 0;

float integralX = 0;
float integralY = 0;

void setup()
{
    pinMode(in1, OUTPUT);
    pinMode(in2, OUTPUT);
    pinMode(ena, OUTPUT);
}
pinMode(in3, OUTPUT);
pinMode(in4, OUTPUT);
pinMode(enb, OUTPUT);

//Serial.begin(9600); // Enable when debugging, disable otherwise for speed
}

void loop()
{

// Option to read variable resistors and assign values to variables

varsetPtX = analogRead(A8); // Set midpoint X
varsetPtY = analogRead(A9); // Set midpoint Y

varKp = analogRead(A6);
varKi = analogRead(A5);
varKd = analogRead(A4);

// Kp = varKp/204.6; // Set Kp range 0 - 5.0
// Ki=varKi/2046; // Set Ki range 0 - 0.5
// Kd = varKd/80; // Set Kd range 0 - 12.0

// Error correction, read sensors, remove highest and lowest
// readings, and calculates average of remaining readings.

HallX[0] = analogRead(pinX);
MaxX=HallX[0];
MinX=MaxX;
HallY[0] = analogRead(pinY);
MaxY=HallY[0];
MinY=MaxY;

HallX[1] = analogRead(pinX);
if (HallX[1]>MaxX){
  MaxX=HallX[1];
} else if (HallX[1]<MinX){
  MinX=HallX[1];
} HallY[1] = analogRead(pinY);
if (HallY[1]>MaxY){
  MaxY=HallY[1];
} else if (HallY[1]<MinY){
  MinY=HallY[1];
}

HallX[2]=analogRead(pinX);
if (HallX[2]>MaxX){
  MaxX=HallX[2];
} else if (HallX[2]<MinX){
  MinX=HallX[2];
} HallY[2] = analogRead(pinY);
if (HallY[2]>MaxY){
  MaxY=HallY[2];
} else if (HallY[2]<MinY){
  MinY=HallY[2];
}

HallX[3] = analogRead(pinX);
if (HallX[3]>MaxX){
  MaxX=HallX[3];
} else if (HallX[3]<MinX){
  MinX=HallX[3];
}
HallY[3] = analogRead(pinY); if (HallY[3]>MaxY){
  MaxY=HallY[3];
} else if (HallY[3]<MinY){
  MinY=HallY[3];
} HallX[4] = analogRead(pinX); if (HallX[4]>MaxX){
  MaxX=HallX[4];
} else if (HallX[4]<MinX){
  MinX=HallX[4];
}
HallY[4] = analogRead(pinY); if (HallY[4]>MaxY){
  MaxY=HallY[4];
} else if (HallY[4]<MinY){
  MinY=HallY[4];
AveHallX = (SumX-MinX-MaxX)/3; // Average x reading
AveHallY = (SumY-MinY-MaxY)/3; // Average y reading
errorX = setPtX - AveHallX; // X position error
errorY = setPtY - AveHallY; // Y position error
dErrorX = errorX - prevErrX;
dErrorY = errorY - prevErrY;
integralX = integralX + errorX*0.01; // X integral
integralY = integralY + errorY*0.01; // Y integral
powerX = errorX*Kp + integralX*Ki + dErrorX*Kd; // X power
powerY = errorY*Kp + integralY*Ki + dErrorY*Kd; // Y power

if(powerX>0) // Need to push in -x direction
  {
    digitalWrite(in3, HIGH);
digitalWrite(in4, LOW);
  }
else //Need to push in +x direction
  {
powerX=-powerX;
digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
  }
if(powerX>240) { powerX=240; }
if(AveHallX>350) { powerX=0; //Stops power being used when no magnet
  integralX=0; //Stops integral if no magnet
}
analogWrite(enb, powerX);

if (powerY > 0) // Push in -y direction {
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
} else // Push in +y direction {
    powerY = -powerY;
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
}

if (powerY > 240) {
    powerY = 240;
}

if (AveHallY > 350) {
    powerY = 0; // Stops power being used when no magnet
    integralY = 0; // Stops integral if no magnet
}

analogWrite(ena, powerY);

prevErrX = errorX; // Assign new previous x error
prevErrY = errorY; // Assign new previous y error

// When debugging the software print the values
/*Serial.print("kp = ");
Serial.print(Kp);
Serial.print("\t ki = ");
Serial.print(Ki);
Serial.print("\t kd = ");
Serial.print(Kd);
Serial.print("\t set y = ");
Serial.print(setPtY);
Serial.print("\t set x = ");
Serial.print(setPtX);
Serial.print("\t Hall x = ");
Serial.print(AveHallX);
Serial.print("\t HallY = ");
Serial.println(AveHallY);*/
APPENDIX 7: Secondary Sensor Pair PID Software

/********************************************************
*        Magnetic Leviation PID Controller             *
*                By James Ballard                      *
********************************************************/

int pinX=3;
int pinY=2;

int HallX[5], HallY[5]; //Hall effect reading arrays
int MaxX, MaxY, MinX, MinY; //Error contol variables
int AveHallX=0, AveHallY=0; //Average hall sensor readings

int varsetPtX;
int varsetPtY;

int setPtX=152; // Hall sensor value for equilibrium state
setPtY=160;

int in1=11;
in2=10;
en=13;  // PWM pins for y coils

int in3=8;
in4=7;
enb=4; // PWM pins for x coils

int errorX; // MidValue - current X hall sensor reading
int prevErrX=0; // Previous error used to calc derivative
int dErrorX; // Derivative of error in X

int errorY; // MidValue - current Y hall sensor reading
int prevErrY=0; // Previous error for calculating derivative
int dErrorY; // Derivative of error in Y

int powerX; //PWM value driving X coil
int powerY; //PWM value driving Y coil

float Kp=1.58; // Proportional weighting
float Ki=0.18; // Derivative weighting
float Kd=9.79; // Integral weighting

float varKp = 0; // Tuning variables (external)
float varKi = 0;
float varKd = 0;

float SumX = 0; // Sum of XY readings
float SumY = 0;

float integralX = 0; // Set intial integrals to be zero
float integralY = 0;

void setup()
{
    pinMode(in1, OUTPUT);
pinMode(in2, OUTPUT);
pinMode(en, OUTPUT);

    pinMode(in3, OUTPUT);
void loop()
{
  varsetPtX = analogRead(A8); // Set midpoint X
  // setPtX = (varsetPtX/10) + 150; // 780 to 820

  varsetPtY = analogRead(A9); // Set midpoint Y
  // setPtY = (varsetPtY/10) + 150; // 780 to 820

  varKp = analogRead(A6);
  // Kp = varKp/204.6; // Set Kp range 0 - 5.0

  varKi = analogRead(A5);
  // Ki=varKi/2046; // Set Ki range 0 - 0.5

  varKd = analogRead(A4);
  Kd = varKd/80; // set Kd range 0 - 12.0

  // Error correction, read sensors, remove highest and lowest readings.
  // and calculates average of remaining readings.

  HallX[0] = analogRead(pinX);
  MaxX=HallX[0];
  MinX=MaxX;

  HallY[0] = analogRead(pinY);
  MaxY=HallY[0];
  MinY=MaxY;

  HallX[1] = analogRead(pinX);
  if (HallX[1]>MaxX){
    MaxX=HallX[1];}
  else if (HallX[1]<MinX){
    MinX=HallX[1];}

  HallY[1] = analogRead(pinY);
  if (HallY[1]>MaxY){
    MaxY=HallY[1];}
  else if (HallY[1]<MinY){
    MinY=HallY[1];}

  HallX[2]=analogRead(pinX);
  if (HallX[2]>MaxX){
    MaxX=HallX[2];}
  else if (HallX[2]<MinX){
    MinX=HallX[2];}

  HallY[2] = analogRead(pinY);
  if (HallY[2]>MaxY){
    MaxY=HallY[2];}
  else if (HallY[2]<MinY){
    MinY=HallY[2];}

  HallX[3] = analogRead(pinX);
  if (HallX[3]>MaxX){
    MaxX=HallX[3];}
  else if (HallX[3]<MinX){
    MinX=HallX[3];}

  HallY[3] = analogRead(pinY);
if (HallY[3] > MaxY){
    MaxY = HallY[3];
} else if (HallY[3] < MinY){
    MinY = HallY[3];
}

HallX[4] = analogRead(pinX);
if (HallX[4] > MaxX){
    MaxX = HallX[4];
} else if (HallX[4] < MinX){
    MinX = HallX[4];
}

HallY[4] = analogRead(pinY);
if (HallY[4] > MaxY){
    MaxY = HallY[4];
} else if (HallY[4] < MinY){
    MinY = HallY[4];
}

HallX[4] = analogRead(pinX);
if (HallX[4] > MaxX){
    MaxX = HallX[4];
} else if (HallX[4] < MinX){
    MinX = HallX[4];
}

HallY[4] = analogRead(pinY);
if (HallY[4] > MaxY){
    MaxY = HallY[4];
} else if (HallY[4] < MinY){
    MinY = HallY[4];
}


AveHallX = (SumX - MinX - MaxX) / 3; // Average x reading
AveHallY = (SumY - MinY - MaxY) / 3; // Average y reading

errorX = setPtX - AveHallX; // X position error
errorY = setPtY - AveHallY; // Y position error

dErrorX = errorX - prevErrX;
dErrorY = errorY - prevErrY;

integralX = integralX + errorX * 0.01; // X integral
integralY = integralY + errorY * 0.01; // Y integral

powerX = errorX * Kp + integralX * Ki + dErrorX * Kd; // X power
powerY = errorY * Kp + integralY * Ki + dErrorY * Kd; // Y power

if (powerX > 0) //need to push in -x direction (opposite to sensor pair 1)
{
    digitalWrite(in3, 0);
    digitalWrite(in4, 1);
} else //need to push in +x direction (opposite to sensor pair 1)
{
    powerX = -powerX;
    digitalWrite(in4, 0);
    digitalWrite(in3, 1);
}

if (powerX > 240)
{
    powerX = 240;
}

if (AveHallX > 350)
{
    powerX = 0;  //Stops power being used when no magnet
    integralX = 0;  //Stops integral if no magnet
}

analogWrite(enb, powerX);  //Output power to y-axis

if (powerY > 0)
  //Push in -y direction (opposite to sensor pair 1)
  digitalWrite(in1, 0);
digitalWrite(in2, 1);
}
else
// Push in +y direction (opposite to sensor pair 1)
{
powerY = -powerY;
digitalWrite(in2, 0);
digitalWrite(in1, 1);
}
if(powerY>240)
{
powerY=240;
}
if(AveHallY>350)
{
powerY=0; // Stops power being used when no magnet
integralY=0; // Stops integral if no magnet
}
analogWrite(ena, powerY); // Output power to y-axis
prevErrX=errorX; // Assign new previous x error
prevErrY=errorY; // Assign new previous y error
/* Serial.print("kp = ");
Serial.print(Kp);
Serial.print(" \t ki = ");
Serial.print(Ki);
Serial.print(" \t kd = ");
Serial.print(Kd);
Serial.print(" \t set y = ");
Serial.print(setPtY);
Serial.print(" \t set x = ");
Serial.print(setPtX);
Serial.print(" \t Hall x = ");
Serial.print(AveHallX);
Serial.print(" \t HallY = ");
Serial.println(AveHallY);
*/
}
APPENDIX 8: Dynamic Redundancy Software

/******************
*    Magnetic Leviation PID Controller        *
*         By James Ballard                    *
******************/

int pinX=3;
int pinY=2;
int fail_pin = 22;
int fail_status = 0;
int sensorfail=0;
int working = 0;

int HallX[5], HallY[5];       //Hall effect reading arrays
int MaxX, MaxY, MinX, MinY;   //Error control variables
int AveHallX=0, AveHallY=0;   //Average hall sensor readings

int varsetPtX;
int varsetPtY;

int setPtX=152; // Hall sensor value for equilibrium state
int setPtY=160;

// YY Coil control
int in1=11;       // blue wire
int in2=10;       // yellow wire
int ena=13;       // white wire PWM pins for y coils

// XX Coil control
int in3=8;        // yellow wire
int in4=7;        // blue wire
int enb=4;        // white whire PWM pins for x coils

int errorX;       // midValue - current X hall sensor reading
int prevErrX=0;   // Previous error used to calculate derivative
int dErrorX;      // Derivative of error in X

int errorY;       // midValue - current Y hall sensor reading
int prevErrY=0;   // previous error for calculating derivative
int dErrorY;      // derivative of error in Y

int powerX;       //PWM value driving X coil
int powerY;       //PWM value driving Y coil

float Kp=1.58;     // Proportional weighting
float Ki=0.18;     // Derivative weighting
float Kd=9.79;     // Integral weighting

float varKp = 0;   // Tuning variables (external)
float varKi = 0;
float varKd = 0;

float SumX = 0;    // Sum of XY readings
float SumY = 0;

float integralX = 0; // Set initial integrals to be zero
float integralY = 0;
void setup()
{
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(ena, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  pinMode(enb, OUTPUT);
  pinMode(fail_pin, INPUT);
  //Serial.begin(9600); // Enable when debugging, disable otherwise for speed
}

void loop()
{
  // Define variables for no sensor fail
  pinX=3;
  pinY=2;
  setPtX=157; // Hall sensor value for equilibrium state
  setPtY=157;
  Kp=1.01; // Proportional weighting
  Ki=0.13; // Derivative weighting
  Kd=7.50; // Integral weighting
  prevErrX=0; // Previous error used to calc derivatve
  prevErrY=0; // Previous error for calculating dirivative
  integralX = 0; // Reset integral values
  integralY = 0;

  while(sensorfail==0)
  {
    varsetPtX = analogRead (A8); // Set midpoint X
    //setPtX = (varsetPtX/10) + 150; // 150 to 200
    varsetPtY = analogRead (A9); // Set midpoint Y
    //setPtY = (varsetPtY/10) + 150; //150 to 200
    varKp = analogRead (A6);
    //Kp = varKp/204.6; // Set Kp range 0 - 5.0
    varKi = analogRead (A5);
    //Ki=varKi/2046; // Set Ki range 0 - 0.5
    varKd = analogRead (A4);
    //Kd = varKd/80; // set Kd range 0 - 12.0

    // Error correction, read sensors, remove highest and lowest
    // readings, and calculates average of remaining readings.
    HallX[0] = analogRead(pinX);
    MaxX=HallX[0];
    MinX=MaxX;
HallY[0] = analogRead(pinY);  
MaxY=HallY[0];  
MinY=MaxY;

HallX[1] = analogRead(pinX);  
if (HallX[1]>MaxX){  
  MaxX=HallX[1];  
} else if (HallX[1]<MinX){  
  MinX=HallX[1];  
} 
HallY[1] = analogRead(pinY);  
if (HallY[1]>MaxY){  
  MaxY=HallY[1];  
} else if (HallY[1]<MinY){  
  MinY=HallY[1];  
}

HallX[2]=analogRead(pinX);  
if (HallX[2]>MaxX){  
  MaxX=HallX[2];  
} else if (HallX[2]<MinX){  
  MinX=HallX[2];  
} 
HallY[2] = analogRead(pinY);  
if (HallY[2]>MaxY){  
  MaxY=HallY[2];  
} else if (HallY[2]<MinY){  
  MinY=HallY[2];  
}

HallX[3] = analogRead(pinX);  
if (HallX[3]>MaxX){  
  MaxX=HallX[3];  
} else if (HallX[3]<MinX){  
  MinX=HallX[3];  
} 
HallY[3] = analogRead(pinY);  
if (HallY[3]>MaxY){  
  MaxY=HallY[3];  
} else if (HallY[3]<MinY){  
  MinY=HallY[3];  
}

HallX[4] = analogRead(pinX);  
if (HallX[4]>MaxX){  
  MaxX=HallX[4];  
} else if (HallX[4]<MinX){  
  MinX=HallX[4];  
} 
HallY[4] = analogRead(pinY);  
if (HallY[4]>MaxY){  
  MaxY=HallY[4];  
} else if (HallY[4]<MinY){  
  MinY=HallY[4];  
}


AveHallX = (SumX-MinX-MaxX)/3;  // Average x reading  
AveHallY = (SumY-MinY-MaxY)/3;  // Average y reading  

errorX = setPtX - AveHallX;  // X position error  
errorY = setPtY - AveHallY;  // Y position error  

dErrorX = errorX - prevErrX;  // X position differential error  
dErrorY = errorY - prevErrY;  // Y position differential error  

integralX = integralX + errorX*0.015;  // X integral  
integralY = integralY + errorY*0.015;  // Y integral
powerX = errorX*Kp + integralX*Ki + dErrorX*Kd;  // X power
powerY = errorY*Kp + integralY*Ki + dErrorY*Kd;  // Y power

if(powerX>0) // need to push in -x direction
{
digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
}
else //need to push in +x direction
{
powerX=-powerX;
digitalWrite(in4, LOW);
digitalWrite(in3, HIGH);
}

if(powerX>240)
{
powerX=240;
}
if(AveHallX>350)
{
powerX=0;    //stops power being used when no magnet
integralX=0;  //stops integral if no magnet
}
analogWrite(enb, powerX);  // Output power to x-axis

if(powerY>0) //Push in -y direction
{
digitalWrite(in1, LOW);
digitalWrite(in2, HIGH);
}
else // Push in +y direction
{
powerY = -powerY;
digitalWrite(in2, LOW);
digitalWrite(in1, HIGH);
}

if(powerY>240)
{
powerY=240;
}
if(AveHallY>400)
{
powerY=0;    // Stops power being used when no magnet
integralY=0;  // Stops integral if no magnet
}
analogWrite(enya, powerY);  // Output power to y-axis

prevErrX=errorX; // Assign new previous x error value
prevErrY=errorY;  // Assign new previous y error value

/*
Serial.print("kp = ");
Serial.print(Kp);
Serial.print("\t ki = ");
Serial.print(Ki);
*/
Serial.print("\t kd = ");
Serial.print(Kd);

Serial.print("\t set y = ");
Serial.print(setPtY);

Serial.print("\t set x = ");
Serial.print(setPtX);

Serial.print("\t Hall x = ");
Serial.print(AveHallX);

Serial.print("\t HallY = ");
Serial.println(AveHallY);
*/

// Determine status of primary sensor pair
fail_status = digitalRead(fail_pin);

// Serial.print(fail_status); // Optional message to screen
if (fail_status == 1) // If fail detected switch sensor pair
{
sensorfail=1;
}
}

/******* Do something else *******/

// use secondary sensor pair

/******* disable power *******/

// Define variables for secondary sensors

pinX=1;
pinY=0;

setPtX=167; // Hall sensor value for equilibrium state
setPtY=150;

Kp = 2;
Ki = 0.5;
Kd = 9.74;

int i = 0; // Counting variable

prevErrX=0; // Previous error used to calc derivative

prevErrY=0; // previous error for calculating dirivative

integralX = 0; // Set intial integrals to be zero
integralY = 0;

// Temporary gain boost

while(sensorfail==1)
{

if (i > 500) {  // for 500 cycles increase Ki gain
    Ki = 0.13;
}

//analogWrite(ena, power0);  //Reset power output
//analogWrite(enb, power0);  //Reset power output

// Error correction, read sensors, remove highest and lowest
// readings, and calculates average of remaining readings.

HallX[0] = analogRead(pinX);
MaxX=HallX[0];
MinX=MaxX;
HallY[0] = analogRead(pinY);
MaxY=HallY[0];
MinY=MaxY;

HallX[1] = analogRead(pinX);
if (HallX[1]>MaxX){
    MaxX=HallX[1];
    } else if (HallX[1]<MinX){
    MinX=HallX[1];
    } HallY[1] = analogRead(pinY);
if (HallY[1]>MaxY){
    MaxY=HallY[1];
    } else if (HallY[1]<MinY){
    MinY=HallY[1];
    }

HallX[2]=analogRead(pinX);
if (HallX[2]>MaxX){
    MaxX=HallX[2];
    } else if (HallX[2]<MinX){
    MinX=HallX[2];
    } HallY[2] = analogRead(pinY);
if (HallY[2]>MaxY){
    MaxY=HallY[2];
    } else if (HallY[2]<MinY){
    MinY=HallY[2];
    }

HallX[3] = analogRead(pinX);
if (HallX[3]>MaxX){
    MaxX=HallX[3];
    } else if (HallX[3]<MinX){
    MinX=HallX[3];
    } HallY[3] = analogRead(pinY);
if (HallY[3]>MaxY){
    MaxY=HallY[3];
    } else if (HallY[3]<MinY){
    MinY=HallY[3];
    }

HallX[4] = analogRead(pinX);
if (HallX[4]>MaxX){
    MaxX=HallX[4];
    } else if (HallX[4]<MinX){
    MinX=HallX[4];
    } HallY[4] = analogRead(pinY);
if (HallY[4]>MaxY){
    MaxY=HallY[4];
    } else if (HallY[4]<MinY){
    MinY=HallY[4];
    }

AveHallX = (SumX-MinX-MaxX)/3;  // Average x reading
AveHallY = (SumY-MinY-MaxY)/3;  // Average y reading

errorX = setPtX - AveHallX;  // X position error
errorY = setPtY - AveHallY;  // Y position error

dErrorX = errorX - prevErrX;  // X position differential error
dErrorY = errorY - prevErrY;  // Y position differential error

integralX = integralX + errorX*0.01;  // X integral
integralY = integralY + errorY*0.01;  // Y integral

powerX = errorX*Kp + integralX*Ki + dErrorX*Kd;  // X power
powerY = errorY*Kp + integralY*Ki + dErrorY*Kd;  // Y power

if(powerX>0) // need to push in -x direction (opposite to sensor pair 1)
{
  digitalWrite(in3, HIGH);
digitalWrite(in4, LOW);
}
else //need to push in +x direction (opposite to sensor pair 1)
{
powerX=-powerX;
digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
}

if(powerX>240) //Current limit
{
powerX=240;
}
if(AveHallX>350)
{
powerX=0;  //Stops power being used when no magnet
integralX=0;  //Stops integral if no magnet
}
analogWrite(enb, powerX);  // Output power to x-axis

if(powerY>0) //Push in -y direction (opposite to sensor pair 1)
{
digitalWrite(in1, HIGH);
digitalWrite(in2, LOW);
}
else // Push in +y direction (opposite to sensor pair 1)
{
powerY = -powerY;
digitalWrite(in1, LOW);
digitalWrite(in2, HIGH);
}

if(powerY>240) //Current limit
{
powerY=240;
}
if(AveHallY>350)
{
powerY=0;  //stops power being used when no magnet
integralY=0;  //stops integral if no magnet
}

analogWrite(ena, powerY);  // Output power to y-axis
prevErrX=errorX;  //assign new previous x error
prevErrY=errorY;  // Assign new previous y error

/* Serial.print("kp = ");
  Serial.print(Kp);
  Serial.print("\t ki = ");
  Serial.print(Ki);
  Serial.print("\t kd = ");
  Serial.print(Kd);
  Serial.print("\t set y = ");
  Serial.print(setPtY);
  Serial.print("\t set x = ");
  Serial.print(setPtX);
  Serial.print("\t Hall x = ");
  Serial.print(AveHallX);
  Serial.print("\t HallY = ");
  Serial.println(AveHallY);
*/

// Serial.print(working);

// Read the fail pin of the Arduino in order to decide the status of the
primary sensor pair
fail_status = digitalRead(fail_pin);

if (fail_status == LOW)  //If primary sensor pair is repaired restore
fail status
(sensorfail=0;
  // Serial.print(sensorfail);  //Optional print to screen
}

i++;  //Increase the cycle count for the gain boost
}
APPENDIX 9: Transmitter Arduino v.1.0 Software

/*******************************************************************************/
//----------Magnetic Levitation by Repulsion
//----------SPI Data Transmitter Software (Version 1)
/*******************************************************************************/

#include <SPI.h>
#include "pins_arduino.h"

// SPI
x_ready = 24; // Pin number of x_ready signal
y_ready = 26; // Pin number of y_ready signal

// Other pin numbers
int pinX=3;
int pinY=2;
int fail_pin = 22;
int fail_status = 0;
int sensorfail=0;
int XY_identify = 32;

int HallX[5], HallY[5]; //Hall effect reading arrays
int MaxX, MaxY, MinX, MinY; //Error control variables
int AveHallX=0, AveHallY=0; //Average hall sensor readings

int varsetPtX;
int varsetPtY;

int setPtX=152; // Hall sensor value for equilibrium state
int setPtY=160;

// YY Coil control
int in1=11; // blue wire
int in2=10; // yellow wire
int ena=13; // white wire PWM pins for y coils

// XX Coil control
int in3=8; // yellow wire
int in4=7; // blue wire
int enb=4; // white wire PWM pins for x coils

int errorX; // midValue - current X hall sensor reading
int prevErrX=0; // Previous error used to calc derivative
int dErrorX; // Derivative of error in X

int errorY; // midValue - current Y hall sensor reading
int prevErrY=0; // previous error for calculating derivative
int dErrorY; // derivative of error in Y

int powerX; //PWM value driving X coil
int powerY; //PWM value driving Y coil

float Kp=1.58; // Proportional weighting
float Ki=0.18; // Derivative weighting
float Kd=9.79; // Integral weighting

float varKp = 0; // Tuning variables (external)
float varKi = 0;
float varKd = 0;

float SumX = 0;  // Sum of XY readings
float SumY = 0;

float integralX = 0;  // Reset integral values
float integralY = 0;

void setup()
{
    pinMode(in1, OUTPUT);
    pinMode(in2, OUTPUT);
    pinMode(ena, OUTPUT);
    pinMode(in3, OUTPUT);
    pinMode(in4, OUTPUT);
    pinMode(enb, OUTPUT);
    pinMode(XY_identify, OUTPUT);  // XY identify signal pin

    // Enable SPI Settings
    digitalWrite(SS, HIGH);  // Ensure SS stays high for now
    SPI.begin ();  // Begin the SPI mode
    SPI.setClockDivider(SPI_CLOCK_DIV4);  // Set the SPI clock speed
    digitalWrite(XY_identify, LOW);

    //Serial.begin(9600);
}

void loop()
{

    varsetPtX = analogRead(A8);  // Set midpoint X
    //setPtX = (varsetPtX/10) + 150;  // 150 to 200

    varsetPtY = analogRead(A9);  // Set midpoint Y
    //setPtY = (varsetPtY/10) + 150;  //150 to 200

    varKp = analogRead(A6);
    //Kp = varKp/204.6;  // Set Kp range 0 - 5.0

    varKi = analogRead(A5);
    //Ki=varKi/2046;  // Set Ki range 0 - 0.5

    varKd = analogRead(A4);
    //Kd = varKd/80;  // set Kd range 0 - 12.0
// Error correction, read sensors, remove highest and lowest readings, and calculates average of remaining readings.

HallX[0] = analogRead(pinX);
MaxX=HallX[0];
MinX=MaxX;
HallY[0] = analogRead(pinY);
MaxY=HallY[0];
MinY=MaxY;

HallX[1] = analogRead(pinX);
if (HallX[1]>MaxX){
    MaxX=HallX[1];
} else if (HallX[1]<MinX){
    MinX=HallX[1];
}
HallY[1] = analogRead(pinY);
if (HallY[1]>MaxY){
    MaxY=HallY[1];
} else if (HallY[1]<MinY){
    MinY=HallY[1];
}

HallX[2]=analogRead(pinX);
if (HallX[2]>MaxX){
    MaxX=HallX[2];
} else if (HallX[2]<MinX){
    MinX=HallX[2];
}
HallY[2] = analogRead(pinY);
if (HallY[2]>MaxY){
    MaxY=HallY[2];
} else if (HallY[2]<MinY){
    MinY=HallY[2];
}

HallX[3] = analogRead(pinX);
if (HallX[3]>MaxX){
    MaxX=HallX[3];
} else if (HallX[3]<MinX){
    MinX=HallX[3];
}
HallY[3] = analogRead(pinY);
if (HallY[3]>MaxY){
    MaxY=HallY[3];
} else if (HallY[3]<MinY){
    MinY=HallY[3];
}

HallX[4] = analogRead(pinX);
if (HallX[4]>MaxX){
    MaxX=HallX[4];
} else if (HallX[4]<MinX){
    MinX=HallX[4];
}
HallY[4] = analogRead(pinY);
if (HallY[4]>MaxY){
    MaxY=HallY[4];
} else if (HallY[4]<MinY){
    MinY=HallY[4];
}


AveHallX = (SumX-MinX-MaxX)/3;  // Average x reading

// Calculate PID

//************* Calculate PID **************
*********************************************/
AveHallY = (SumY-MinY-MaxY)/3;  // Average y reading
errorX = setPtX - AveHallX;  // X position error
errorY = setPtY - AveHallY;  // Y position error
dErrorX = errorX - prevErrX;  // X position differential error
dErrorY = errorY - prevErrY;  // Y position differential error
integralX = integralX + errorX*0.015;  // Reset integral values
integralY = integralY + errorY*0.015;
powerX = errorX*Kp + integralX*Ki + dErrorX*Kd;  // Calculate X and Y
powers
powerY = errorY*Kp + integralY*Ki + dErrorY*Kd;

/*********************************************/
/********* Transmit PID Values ***********/
/*********************************************/

if(powerX>0)  // need to push in -x direction
{
digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
}
else  //need to push in +x direction
{
powerX=-powerX;
digitalWrite(in4, LOW);
digitalWrite(in3, HIGH);
}
if(powerX>240)  // Current Limit
{
powerX=240;
}
if(AveHallX>350)
{
powerX=0;  // Stops power being used when no magnet
integralX=0;  // Stops integral if no magnet
}  
analogWrite(enb, powerX);
digitalWrite(XY_identify, HIGH);  // Select X identify
digitalWrite(SS, LOW);  // Select slave
SPI.transfer (powerX);  // Transmit X power
digitalWrite(SS, HIGH);  // de-select slave
digitalWrite(XY_identify, LOW);  // End X identify

if(powerY>0)  //Push in -y direction
{
digitalWrite(in1, LOW);
digitalWrite(in2, HIGH);
}
else  // Push in +y direction
{
powerY = -powerY;
digitalWrite(in2, LOW);
digitalWrite(in1, HIGH);
}
if(powerY>240)  // Current Limit
{
  powerY=240;
}

if(AveHallY>400)
{
  powerY=0;     // stops power being used when no magnet
  integralY=0;  // stops integral if no magnet
}

digitalWrite(XY_identify, LOW);  // Select Y identify
digitalWrite(SS, LOW);           // Select slave
SPI.transfer(powerY);            // Transmit Y power
digitalWrite(SS, HIGH);          // de-select slave
APPENDIX 10: Receiver Arduino v.1.0 Software

/******************************************************/
//----------Magnetic Levitation by Repulsion
//---------SPI Data Receive Software (Version 1)
/******************************************************/

// include SPI header file
#include <SPI.h>

// define global variables
volatile boolean process_it;
byte power; // Data byte
byte powerX;
int enb=4; // white wire
int in3=8; // yellow wire
int in4=7; // blue wire

// Y-axis
byte powerY;
int in1=11; // blue wire
int in2=10; // yellow wire
int ena=13; // white wire PWM pins for y coils

// Read x-axis
int read_in3=30;
int read_in4=32;

// Read y-axis
int read_in1=40;
int read_in2=42;

int XY_identify=34;

void setup (void)
{
  //Serial.begin (9600); // debugging

  // Slave mode
  pinMode(MISO, OUTPUT);

  // Enable X-axis outputs
  pinMode(enb, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);

  // Enable Y-axis outputs
  pinMode(ena, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);

  // Enable Inputs
  pinMode(read_in3, INPUT);
  pinMode(read_in4, INPUT);
  pinMode(XY_identify, INPUT);

  // turn on SPI in slave mode
  SPCR |= _BV(SPE);
// Interrupt Flag
process_it = false;

// Enable SPI interrupts
SPIattachInterrupt();

ISR (SPI_STC_vect)
{
    cli(); // disable interrupts

    if (digitalRead(XY_identify) == HIGH)
    {
        powerX = SPDR; // read SPI data
        process_it = true; // set interrupt flag
        sei(); // re-enable interrupts
    }
    else
    {
        powerY = SPDR; // read SPI data
        process_it = true; // set interrupt flag
        sei(); // re-enable interrupts
    }
}

void loop (void)
{
    if (process_it) // if interrupts flag is set
    {
        // X-axis direction
        digitalWrite(in3, digitalRead(read_in3)); // Read coil direction
        digitalWrite(in4, digitalRead(read_in4)); // Read coil direction

        // Serial.println(powerX);
        analogWrite(enb, powerX); // output the power

        // Y-axis direction
        digitalWrite(in1, digitalRead(read_in1)); // Read coil direction
        digitalWrite(in2, digitalRead(read_in2)); // Read coil direction
        analogWrite(enb, powerX); // output the power

        // End
        process_it = false; // restore interrupt flag
    }
}
APPENDIX 11: Transmitter Arduino v.2.0 Software

/////////////////////////////////////////////////////////////////////////////////////////
// Magnetic Levitation by Repulsion
// SPI Data Transmitter Software (Version 2)
/////////////////////////////////////////////////////////////////////////////////////////

#include <SPI.h>
#include "pins_arduino.h"

// New values are: Kp = 2.00, Ki = 0.22, Kd = 7.69
// Default setPtX = 218 setPtY=163

// SPI
x_ready = 24;
y_ready = 26;

//
int pinX=3;
int pinY=2;
int fail_pin = 22;
int fail_status = 0;
int sensorfail=0;
int working = 0;
int XY_identify = 32;

int HallX[5], HallY[5];  //Hall effect reading arrays
int MaxX, MaxY, MinX, MinY;  //Error control variables
int AveHallX=0, AveHallY=0;  //Average hall sensor readings

int varsetPtX;
int varsetPtY;

int setPtX=152;  // Hall sensor value for equilibrium state
int setPtY=160;

// YY Coil control
int in1=11;  // blue wire
int in2=10;  // yellow wire
int ena=13;  // white wire PWM pins for y coils

// XX Coil control
int in3=8;  // yellow wire
int in4=7;  // blue wire
int enb=4;  // white wire PWM pins for x coils

int errorX;  // midValue - current X hall sensor reading
int prevErrX=0;  // Previous error used to calc derivative
int dErrorX;  // Derivative of error in X

int errorY;  // midValue - current Y hall sensor reading
int prevErrY=0;  // previous error for calculating derivative
int dErrorY;  // derivative of error in Y

int powerX;  // PWM value driving X coil
int powerY;  // PWM value driving Y coil

float Kp=1.58;  // Proportional weighting
float Ki=0.18;  // Derivative weighting
float Kd = 9.79;  // Integral weighting
float varKp = 0;  // Tuning variables (external)
float varKi = 0;
float varKd = 0;

float SumX = 0;  // Sum of XY readings
float SumY = 0;

float integralX = 0;  // Reset integral values
float integralY = 0;

void setup()
{
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(ena, OUTPUT);

  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  pinMode(enb, OUTPUT);

  pinMode(XY_identify, OUTPUT);  // XY identify signal pin

  // pinMode(3, OUTPUT);

  // SPI Settings
  digitalWrite(SS, HIGH);  // ensure SS stays high for now
  SPI.begin();  // Begin the SPI mode
  SPI.setClockDivider(SPI_CLOCK_DIV4);  // Set the SPI clock speed
  digitalWrite(XY_identify, LOW);

  //Serial.begin(9600);
}

void loop()
{

  varsetPtX = analogRead(A8);  // Set midpoint X
  //setPtX = (varsetPtX/10) + 150;  // 150 to 200

  varsetPtY = analogRead(A9);  // Set midpoint Y
  //setPtY = (varsetPtY/10) + 150;  // 150 to 200

  varKp = analogRead(A6);
  //Kp = varKp/204.6;  // Set Kp range 0 - 5.0

  varKi = analogRead(A5);
  //Ki=varKi/2046;  // Set Ki range 0 - 0.5

  varKd = analogRead(A4);
  //Kd = varKd/80;  // set Kd range 0 - 12.0

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

  // Error correction, read sensors, remove highest and lowest
  // readings, and calculates average of remaining readings.
HallX[0] = analogRead(pinX);
MaxX=HallX[0];
MinX=MaxX;
HallY[0] = analogRead(pinY);
MaxY=HallY[0];
MinY=MaxY;

HallX[1] = analogRead(pinX);
if (HallX[1]>MaxX){
    MaxX=HallX[1];
} else if (HallX[1]<MinX){
    MinX=HallX[1];
} HallY[1] = analogRead(pinY);
if (HallY[1]>MaxY){
    MaxY=HallY[1];
} else if (HallY[1]<MinY){
    MinY=HallY[1];
}

HallX[2]=analogRead(pinX);
if (HallX[2]>MaxX){
    MaxX=HallX[2];
} else if (HallX[2]<MinX){
    MinX=HallX[2];
} HallY[2] = analogRead(pinY);
if (HallY[2]>MaxY){
    MaxY=HallY[2];
} else if (HallY[2]<MinY){
    MinY=HallY[2];
}

HallX[3] = analogRead(pinX);
if (HallX[3]>MaxX){
    MaxX=HallX[3];
} else if (HallX[3]<MinX){
    MinX=HallX[3];
} HallY[3] = analogRead(pinY);
if (HallY[3]>MaxY){
    MaxY=HallY[3];
} else if (HallY[3]<MinY){
    MinY=HallY[3];
}

HallX[4] = analogRead(pinX);
if (HallX[4]>MaxX){
    MaxX=HallX[4];
} else if (HallX[4]<MinX){
    MinX=HallX[4];
} HallY[4] = analogRead(pinY);
if (HallY[4]>MaxY){
    MaxY=HallY[4];
} else if (HallY[4]<MinY){
    MinY=HallY[4];
}


AveHallX = (SumX-MinX-MaxX)/3; // Average x reading
AveHallY = (SumY-MinY-MaxY)/3; // Average y reading

errorX = setPtX - AveHallX; // X position error
errorY = setPtY - AveHallY; // Y position error

dErrorX = errorX - prevErrX; // X position differential error
dErrorY = errorY - prevErrY; // Y position differential error
integralX = integralX + errorX * 0.015;  // Reset integral values
integralY = integralY + errorY * 0.015;

powerX = errorX * Kp + integralX * Ki + dErrorX * Kd;  // Calculate X and Y powers
powerY = errorY * Kp + integralY * Ki + dErrorY * Kd;

/*******************************************************
********* Transmit PID Values ***************/
/*******************************************************

while (digitalRead(x_ready) == LOW) {}  // Wait for slave to become ready
digitalWrite(XY_identify, HIGH);  // Select X identify
digitalWrite(SS, LOW);  // Select slave
SPI.transfer(powerX);  // Transmit X power
digitalWrite(SS, HIGH);  // de-select slave
digitalWrite(XY_identify, LOW);  // End X identify

while (digitalRead(y_ready) == LOW) {}  // Wait for slave to become ready
digitalWrite(XY_identify, LOW);  // Select Y identify
digitalWrite(SS, LOW);  // Select slave
SPI.transfer(powerY);  // Transmit Y power
digitalWrite(SS, HIGH);  // End Y identify

}
APPENDIX 12: Receiver Arduino v.2.0 Software

/**************************************************************/
//----------Magnetic Levitation by Repulsion
//----------SPI Data Receive Software (Version 2)
/**************************************************************/

// include SPI header file
#include <SPI.h>

// define global variables
volatile boolean process_it;

// Control signals
int x_ready = 24;
int y_ready = 26;

// sensor pins
int pinX=3;
int pinY=2;
int fail_pin = 22;
int fail_status = 0;
int sensorfail=0;
int working = 0;
byte power; // Data byte

// Sensor reading
int HallX[5], HallY[5]; //Hall effect reading arrays
int MaxX, MaxY, MinX, MinY; //Error control variables
int AveHallX=0, AveHallY=0; //Average hall sensor readings

int varsetPtX;
int varsetPtY;

int setPtX=152; // Hall sensor value for equilibrium state
int setPtY=160;

//Y-axis
int in1=11; // blue wire
int in2=10; // yellow wire
int ena=13; // white wire PWM pins for y coils

// X-axis
byte received_powerX;
int enb=4; // white wire
int in3=8; // yellow wire
int in4=7; // blue wire

// Y-axis
byte received_powerY;
int in1=11; // blue wire
int in2=10; // yellow wire
int ena=13; // white wire PWM pins for y coils

// Read x-axis
int read_in3=30;
int read_in4=32;
// Read y-axis
int read_in1=40;
int read_in2=42;

// axis direction control signal
int XY_identify=34;

// PID
int errorX;  // midValue - current X hall sensor reading
int prevErrX=0;  // Previous error used to calc derivative
int dErrorX;  // Derivative of error in X
int errorY;  // midValue - current Y hall sensor reading
int prevErrY=0;  // previous error for calculating derivative
int dErrorY;  // derivative of error in Y

int powerX;  //PWM value driving X coil
int powerY;  //PWM value driving Y coil

float Kp=1.58;  // Proportional weighting
float Ki=0.18;  // Derivative weighting
float Kd=9.79;  // Integral weighting

float varKp = 0;  // Tuning variables (external)
float varKi = 0;
float varKd = 0;

float SumX = 0;  // Sum of XY readings
float SumY = 0;

float integralX = 0;
float integralY = 0;

void setup (void)
{
  //Serial.begin (9600);  // debugging

  // Slave mode
  pinMode(MISO, OUTPUT);

  // Enable X-axis outputs
  pinMode(enb, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);

  // Enable Y-axis outputs
  pinMode(ena, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);

  // Control signals
  pinMode(x_ready, OUTPUT);
  pinMode(y_ready, OUTPUT);

  // Enable Inputs
  pinMode(read_in3, INPUT);
  pinMode(read_in4, INPUT);
  pinMode(XY_identify, INPUT);
  pinMode(fail_pin, INPUT);

  // turn on SPI in slave mode
SPCR |= _BV(SPE);

// Interrupt Flag
process_it = false;

// Enable SPI interrupts
SPI.attachInterrupt();
}

// SPI interrupt service routine
ISR (SPI_STC_vect)
{
  cli(); // disable interrupts

  if(digitalRead(XY_identify)==HIGH)
  {
    received_powerX = SPDR; // read SPI data
    process_it = true; // set interrupt flag
    sei(); // re-enable interrupts
  }
  else
  {
    received_powerY = SPDR; // read SPI data
    process_it = true; // set interrupt flag
    sei(); // re-enable interrupts
  }

digitalWrite(x_ready,LOW); // Receiver is not yet ready
digitalWrite(y_ready,LOW);
}

void loop (void)
{

  varsetPtX = analogRead(A8); // Set midpoint X
  //setPtX = (varsetPtX/10) + 150; // 150 to 200

  varsetPtY = analogRead(A9); // Set midpoint Y
  //setPtY = (varsetPtY/10) + 150; //150 to 200

  varKp = analogRead(A6);
  //Kp = varKp/204.6; // Set Kp range 0 - 5.0

  varKi = analogRead(A5);
  //Ki=varKi/2046; // Set Ki range 0 - 0.5

  varKd = analogRead(A4);
  //Kd = varKd/80; // set Kd range 0 - 12.0

  /************************************
  /************* Calculate PID ***********/
  /*************************************/

  // Error correction, read sensors, remove highest and lowest
  // readings, and calculates average of remaining readings.

  HallX[0] = analogRead(pinX);
  MaxX=HallX[0];
  MinX=MaxX;
HallY[0] = analogRead(pinY);
MaxY=HallY[0];
MinY=MaxY;

HallX[1] = analogRead(pinX);
if (HallX[1]>MaxX){
  MaxX=HallX[1];}
else if (HallX[1]<MinX){
  MinX=HallX[1];}
HallY[1] = analogRead(pinY);
if (HallY[1]>MaxY){
  MaxY=HallY[1];}
else if (HallY[1]<MinY){
  MinY=HallY[1];}

HallX[2] = analogRead(pinX);
if (HallX[2]>MaxX){
  MaxX=HallX[2];}
else if (HallX[2]<MinX){
  MinX=HallX[2];}
HallY[2] = analogRead(pinY);
if (HallY[2]>MaxY){
  MaxY=HallY[2];}
else if (HallY[2]<MinY){
  MinY=HallY[2];}

HallX[3] = analogRead(pinX);
if (HallX[3]>MaxX){
  MaxX=HallX[3];}
else if (HallX[3]<MinX){
  MinX=HallX[3];}
HallY[3] = analogRead(pinY);
if (HallY[3]>MaxY){
  MaxY=HallY[3];}
else if (HallY[3]<MinY){
  MinY=HallY[3];}

HallX[4] = analogRead(pinX);
if (HallX[4]>MaxX){
  MaxX=HallX[4];}
else if (HallX[4]<MinX){
  MinX=HallX[4];}
HallY[4] = analogRead(pinY);
if (HallY[4]>MaxY){
  MaxY=HallY[4];}
else if (HallY[4]<MinY){
  MinY=HallY[4];}


AveHallX = (SumX-MinX-MaxX)/3;
AveHallY = (SumY-MinY-MaxY)/3;
errorX = setPtX - AveHallX;
errorY = setPtY - AveHallY;

dErrorX = errorX - prevErrX;
dErrorY = errorY - prevErrY;

integralX = integralX + errorX*0.015;
integralY = integralY + errorY*0.015;
powerX = errorX*Kp + integralX*Ki + dErrorX*Kd;
powerY = errorY*Kp + integralY*Ki + dErrorY*Kd;

/****** Receive and Compare PID **********/
/*****************************/

// check X_PID
digitalWrite(x_ready,HIGH); // Ask for X power value
while(process_it!=true){} // Wait until interrupt flag is set
process_it = false;
if(received_powerX == powerX) // Compare data
{
powerX = received_powerX; // Assign data if correct
}

// check Y_PID
digitalWrite(y_ready,HIGH);
while(process_it!=true){} // Wait until interrupt flag is set
process_it = false;
if(received_powerY == powerY) // Compare data
{
powerY = received_powerY; // Assign data if correct
}

/****************************/  
/****** Output PID ***********/
/*****************************/

if(powerX>0) // need to push in -x direction
{
digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
}
else // need to push in +x direction
{
powerX=-powerX;
digitalWrite(in4, LOW);
digitalWrite(in3, HIGH);
}

if(powerX>240) // Current Limit
{
powerX=240;
}

if(AveHallX>350)
{
powerX=0; // stops power being used when no magnet
integralX=0; // stops integral if no magnet
}
analogWrite(enb, powerX);

if(powerY>0) // Push in -y direction
{
digitalWrite(in1, LOW);
digitalWrite(in2, HIGH);
}
else // Push in +y direction
{
    powerY = -powerY;
digitalWrite(in2, LOW);
digitalWrite(in1, HIGH);
}

if(powerY>240) // Current Limit
{
    powerY=240;
}

if(AveHallY>400)
{
    powerY=0; // stops power being used when no magnet
    integralY=0; // stops integral if no magnet
}

prevErrX=errorX; // Assign new previous x error value
prevErrY=errorY; // Assign new previous y error value
APPENDIX 13: Transmitter Arduino v.3.0 Software

#include <SPI.h>
#include "pins_arduino.h"

// New values are: Kp = 2.00, Ki = 0.22, Kd = 7.69
// Default setPtX = 218  setPtY=163

// SPI
x_ready = 24;
y_ready = 26;

//
int pinX=3;
int pinY=2;
int fail_pin = 22;
int fail_status = 0;
int sensorfail=0;
int working = 0;
int XY_identify = 32;

int HallX[5], HallY[5];    //Hall effect reading arrays
int MaxX, MaxY, MinX, MinY;    //Error control variables
int AveHallX=0, AveHallY=0;    //Average hall sensor readings

int setPtX=152;    // Hall sensor value for equilibrium state
int setPtY=160;

// YY Coil control

int in1=11;    // blue wire
int in2=10;    // yellow wire
int ena=13;    // white wire PWM pins for y coils

// XX Coil control

int in3=8;    // yellow wire
int in4=7;    // blue wire
int enb=4;    // white wire PWM pins for x coils

int errorX;    // midValue - current X hall sensor reading
int prevErrX=0;    // Previous error used to calc derivative
int dErrorX;    // Derivative of error in X

int errorY;    // midValue - current Y hall sensor reading
int prevErrY=0;    // previous error for calculating derivative
int dErrorY;    // derivative of error in Y

int powerX;    //PWM value driving X coil
int powerY;    //PWM value driving Y coil

float Kp=1.58;    // Proportional weighting
float Ki = 0.18; // Derivative weighting
float Kd = 9.79; // Integral weighting

float varKp = 0; // Tuning variables (external)
float varKi = 0;
float varKd = 0;

float SumX = 0; // Sum of XY readings
float SumY = 0;

float integralX = 0;
float integralY = 0;

void setup()
{
    pinMode(in1, OUTPUT);
    pinMode(in2, OUTPUT);
    pinMode(ena, OUTPUT);
    pinMode(in3, OUTPUT);
    pinMode(in4, OUTPUT);
    pinMode(enb, OUTPUT);
    pinMode(XY_identify, OUTPUT);

    // pinMode(3, OUTPUT);
    // SPI Settings
    digitalWrite(SS, HIGH); // ensure SS stays high for now
    SPI.begin();
    SPI.setClockDivider(SPI_CLOCK_DIV4);
    digitalWrite(XY_identify, LOW);

    pinMode(fail_pin, INPUT);

    //Serial.begin(9600);
}

void loop()
{
    // set 2
    pinX=3;
    pinY=2;

    setPtX=157;
    setPtY=157;

    Kp = 1.01; // Proportional weighting
    Ki = 0.13; // Derivative weighting
    Kd = 7.50; // Integral weighting

    prevErrX = 0; // Previous error used to calc derivative
    prevErrY = 0; // previous error for calculating dirivative

    integralX = 0;
    integralY = 0;

    while(sensorfail == 0)
{  
  varsetPtX = analogRead(A8); // Set midpoint X  
  // setPtX = (varsetPtX/10) + 150; // 150 to 200  
  varsetPtY = analogRead(A9); // Set midpoint Y  
  // setPtY = (varsetPtY/10) + 150; // 150 to 200  
  varKp = analogRead(A6);  
  // Kp = varKp/204.6; // Set Kp range 0 - 5.0  
  varKi = analogRead(A5);  
  // Ki = varKi/2046; // Set Ki range 0 - 0.5  
  varKd = analogRead(A4);  
  // Kd = varKd/80; // Set Kd range 0 - 12.0  
  
  /*****************************************************************************  
  ************* Calculate PID **************/  
  /*****************************************************************************  
  // Error correction, read sensors, remove highest and lowest // readings, and calculates average of remaining readings.  
  HallX[0] = analogRead(pinX);  
  MaxX=HallX[0];  
  MinX=MaxX;  
  HallY[0] = analogRead(pinY);  
  MaxY=HallY[0];  
  MinY=MaxY;  
  HallX[1] = analogRead(pinX);  
  if (HallX[1]>MaxX){  
    MaxX=HallX[1];  
  } else if (HallX[1]<MinX){  
    MinX=HallX[1];  
  }  
  HallY[1] = analogRead(pinY);  
  if (HallY[1]>MaxY){  
    MaxY=HallY[1];  
  } else if (HallY[1]<MinY){  
    MinY=HallY[1];  
  }  
  HallX[2]=analogRead(pinX);  
  if (HallX[2]>MaxX){  
    MaxX=HallX[2];  
  } else if (HallX[2]<MinX){  
    MinX=HallX[2];  
  }  
  HallY[2] = analogRead(pinY);  
  if (HallY[2]>MaxY){  
    MaxY=HallY[2];  
  } else if (HallY[2]<MinY){  
    MinY=HallY[2];  
  }  
  HallX[3] = analogRead(pinX);  
  if (HallX[3]>MaxX){  
    MaxX=HallX[3];  
  } else if (HallX[3]<MinX){  
    MinX=HallX[3];  
  }  
  HallY[3] = analogRead(pinY);  
  if (HallY[3]>MaxY){  
    MaxY=HallY[3];  
  }
else if (HallY[3]<MinY){
    MinY=HallY[3];}
HallX[4] = analogRead(pinX);
if (HallX[4]>MaxX){
    MaxX=HallX[4];
    else if (HallX[4]<MinX){
    MinX=HallX[4];}
HallY[4] = analogRead(pinY);
if (HallY[4]>MaxY){
    MaxY=HallY[4];
    else if (HallY[4]<MinY){
    MinY=HallY[4];}
HallX[4] = analogRead(pinX);
if (HallX[4]>MaxX){
    MaxX=HallX[4];
    else if (HallX[4]<MinX){
    MinX=HallX[4];}
HallY[4] = analogRead(pinY);
if (HallY[4]>MaxY){
    MaxY=HallY[4];
    else if (HallY[4]<MinY){
    MinY=HallY[4];}
AveHallX = (SumX-MinX-MaxX)/3;
AveHallY = (SumY-MinY-MaxY)/3;
errorX = setPtX - AveHallX;
errorY = setPtY - AveHallY;
dErrorX = errorX - prevErrX;
dErrorY = errorY - prevErrY;
integralX = integralX + errorX*0.015;
integralY = integralY + errorY*0.015;
powerX = errorX*Kp + integralX*Ki + dErrorX*Kd;
powerY = errorY*Kp + integralY*Ki + dErrorY*Kd;
/****************************************************
/******* Transmit PID Values ********************/
/****************************************************/
while(digitalRead(x_ready) == LOW){}
digitalWrite(XY_identify, HIGH);
digitalWrite(SS, LOW);
SPI.transfer (powerX);
digitalWrite(SS, HIGH);
digitalWrite(XY_identify, LOW);

while(digitalRead(y_ready) == LOW){}
digitalWrite(XY_identify, LOW);
digitalWrite(SS, LOW);
SPI.transfer (powerY);
digitalWrite(SS, HIGH);

// Detect Fail
working = 0;
fail_status = digitalRead(fail_pin);
if (fail_status == 1)
{sensorfail=1;
}

/*************** Do something else *************/
// use sensor pair 1

/****************** disable power ******************/

pinX=1;
pinY=0;

setPtX=167; // Hall sensor value for equilibrium state
setPtY=150;

Kp = 2;
Ki = 0.5;
Kd = 9.74;

int i = 0;

prevErrX=0; // Previous error used to calc derivative
prevErrY=0; // previous error for calculating derivative

integralX = 0;
integralY = 0;

digitalWrite(3,HIGH);

while(sensorfail==1)
{
   if (i > 500){
      Ki = 0.13;
   }

   /*********************************************/
   /************* Calculate PID ***************
   /*********************************************/

   HallX[0] = analogRead(pinX);
   MaxX=HallX[0];
   MinX=MaxX;
   HallY[0] = analogRead(pinY);
   MaxY=HallY[0];
   MinY=MaxY;

   HallX[1] = analogRead(pinX);
   if (HallX[1]>MaxX){
      MaxX=HallX[1];
   } else if (HallX[1]<MinX){
      MinX=HallX[1];
   }
   HallY[1] = analogRead(pinY);
   if (HallY[1]>MaxY){
      MaxY=HallY[1];
   } else if (HallY[1]<MinY){
      MinY=HallY[1];
   }

   HallX[2]=analogRead(pinX);
   if (HallX[2]>MaxX){
      MaxX=HallX[2];
   } else if (HallX[2]<MinX){
      MinX=HallX[2];
   }

   /*********************************************/
MinX=HallX[2];
HallY[2] = analogRead(pinY);
if (HallY[2]>MaxY){
    MaxY=HallY[2];
} else if (HallY[2]<MinY){
    MinY=HallY[2];
}
HallX[3] = analogRead(pinX);
if (HallX[3]>MaxX){
    MaxX=HallX[3];
} else if (HallX[3]<MinX){
    MinX=HallX[3];
}
HallY[3] = analogRead(pinY);
if (HallY[3]>MaxY){
    MaxY=HallY[3];
} else if (HallY[3]<MinY){
    MinY=HallY[3];
}
HallX[4] = analogRead(pinX);
if (HallX[4]>MaxX){
    MaxX=HallX[4];
} else if (HallX[4]<MinX){
    MinX=HallX[4];
}
HallY[4] = analogRead(pinY);
if (HallY[4]>MaxY){
    MaxY=HallY[4];
} else if (HallY[4]<MinY){
    MinY=HallY[4];
}


AveHallX = (SumX-MinX-MaxX)/3;
AveHallY = (SumY-MinY-MaxY)/3;

errorX = setPtX - AveHallX;
errorY = setPtY - AveHallY;

dErrorX = errorX - prevErrX;
dErrorY = errorY - prevErrY;

integralX = integralX + errorX*0.01;
integralY = integralY + errorY*0.01;

powerX = errorX*Kp + integralX*Ki + dErrorX*Kd;
powerY = errorY*Kp + integralY*Ki + dErrorY*Kd;

/*****************************/
/********* Transmit PID Values ***********/
/*****************************/

while(digitalRead(x_ready) == LOW){
    digitalWrite(XY_identify, HIGH);
digitalWrite(SS, LOW);
    SPI.transfer(powerX);
digitalWrite(SS, HIGH);
digitalWrite(XY_identify, LOW);
}

while(digitalRead(y_ready) == LOW){
    digitalWrite(XY_identify, LOW);
digitalWrite(SS, LOW);
    SPI.transfer(powerY);
digitalWrite(SS, HIGH);

// End Transmit
prevErrX = errorX;
prevErrY = errorY;
fail_status =digitalRead(fail_pin);
if (fail_status == LOW)
(sensorfail=0;
}

i++;
}
}
APPENDIX 14: Receiver Arduino v.3.0 Software

(/[-----------------------------]/
//----------Magnetic Levitation by Repulsion
//----------SPI Data Receive Software (Slave 1)
/[-----------------------------]/

// include SPI header file
#include <SPI.h>

// define global variables
volatile boolean process_it;

// Control signals
int x_ready = 24;
int y_ready = 26;

// sensor pins
int pinX=3;
int pinY=2;
int fail_pin = 22;
int fail_status = 0;
int sensorfail=0;
int working = 0;
byte power;

// Sensor reading
int HallX[5], HallY[5];       //Hall effect reading arrays
int MaxX, MaxY, MinX, MinY;   //Error control variables
int AveHallX=0, AveHallY=0;   //Average hall sensor readings

int varsetPtX;
int varsetPtY;

int setPtX=152;       //Hall sensor value for equilibrium state
int setPtY=160;

//Y-axis
int in1=11;           //blue wire
int in2=10;           //yellow wire
int ena=13;           //white wire PWM pins for y coils

// X-axis
byte received_powerX;
int enb=4;            //white wire
int in3=9;            //yellow wire
int in4=7;            //blue wire

// Y-axis
byte received_powerY;
int in1=11;           //blue wire
int in2=10;           //yellow wire
int ena=13;           //white wire PWM pins for y coils

// Read x-axis
int read_in3=30;
int read_in4=32;
// Read y-axis
int read_in1=40;
int read_in2=42;

// axis direction control signal
int XY_identify=34;

// PID
int errorX; // midValue - current X hall sensor reading
int prevErrX=0; // Previous error used to calc derivative
int dErrorX; // Derivative of error in X
int errorY; // midValue - current Y hall sensor reading
int prevErrY=0; // previous error for calculating derivative
int dErrorY; // derivative of error in Y
int powerX; //PWM value driving X coil
int powerY; //PWM value driving Y coil

float Kp=1.58; // Proportional weighting
float Ki=0.18; // Derivative weighting
float Kd=9.79; // Integral weighting

float varKp = 0; // Tuning variables (external)
float varKi = 0;
float varKd = 0;

float SumX = 0; // Sum of XY readings
float SumY = 0;

float integralX = 0;
float integralY = 0;

void setup (void)
{
    //Serial.begin (9600);  // debugging

    // Slave mode
    pinMode(MISO, OUTPUT);

    // Enable X-axis outputs
    pinMode(enb, OUTPUT);
pinMode(in3, OUTPUT);
pinMode(in4, OUTPUT);

    // Enable Y-axis outputs
    pinMode(ena, OUTPUT);
pinMode(in1, OUTPUT);
pinMode(in2, OUTPUT);

    // Control signals
    pinMode(x_ready, OUTPUT);
pinMode(y_ready, OUTPUT);

    // Enable Inputs
    pinMode(read_in3, INPUT);
pinMode(read_in4, INPUT);
pinMode(XY_identify, INPUT);
pinMode(fail_pin, INPUT);

    // turn on SPI in slave mode
SPCR |= _BV(SPE);

// Interrupt Flag
process_it = false;

// Enable SPI interrupts
SPI.attachInterrupt();
}

// SPI interrupt service routine
ISR (SPI_STC_vect)
{
  cli(); // disable interrupts

  if(digitalRead(XY_identify)==HIGH)
  {
    received_powerX = SPDR; // read SPI data
    process_it = true; // set interrupt flag
    sei(); // re-enable interrupts
  }
  else
  {
    received_powerY = SPDR; // read SPI data
    process_it = true; // set interrupt flag
    sei(); // re-enable interrupts
  }
  digitalWrite(x_ready, LOW);
  digitalWrite(y_ready, LOW);
}

void loop (void)
{
  pinX = 3;
  pinY = 2;

  setPtX = 157;
  setPtY = 157;

  Kp = 1.01; // Proportional weighting
  Ki = 0.13; // Derivative weighting
  Kd = 7.50; // Integral weighting

  prevErrX = 0; // Previous error used to calc derivative
  prevErrY = 0; // previous error for calculating derivative

  integralX = 0;
  integralY = 0;

  while(sensorfail == 0)
  {
    varsetPtX = analogRead(A8); // Set midpoint X
    //varsetPtX = (varsetPtX/10) + 150; // 150 to 200
    varsetPtY = analogRead(A9); // Set midpoint Y
    //varsetPtY = (varsetPtY/10) + 150; //150 to 200
```plaintext
varKp = analogRead (A6); // Kp = varKp/204.6; // Set Kp range 0 - 5.0

varKi = analogRead (A5); // Ki = varKi/2046; // Set Ki range 0 - 0.5

varKd = analogRead (A4); // Kd = varKd/80; // Set Kd range 0 - 12.0

/********************************************
/************* Calculate PID *************/
/********************************************/

// Error correction, read sensors, remove highest and lowest readings, and calculates average of remaining readings.

HallX[0] = analogRead(pinX);
MaxX=HallX[0];
MinX=MaxX;
HallY[0] = analogRead(pinY);
MaxY=HallY[0];
MinY=MaxY;

HallX[1] = analogRead(pinX);
if (HallX[1]>MaxX){
    MaxX=HallX[1];
} else if (HallX[1]<MinX){
    MinX=HallX[1];
}
HallY[1] = analogRead(pinY);
if (HallY[1]>MaxY){
    MaxY=HallY[1];
} else if (HallY[1]<MinY){
    MinY=HallY[1];
}

HallX[2]=analogRead(pinX);
if (HallX[2]>MaxX){
    MaxX=HallX[2];
} else if (HallX[2]<MinX){
    MinX=HallX[2];
}
HallY[2] = analogRead(pinY);
if (HallY[2]>MaxY){
    MaxY=HallY[2];
} else if (HallY[2]<MinY){
    MinY=HallY[2];
}

HallX[3] = analogRead(pinX);
if (HallX[3]>MaxX){
    MaxX=HallX[3];
} else if (HallX[3]<MinX){
    MinX=HallX[3];
}
HallY[3] = analogRead(pinY);
if (HallY[3]>MaxY){
    MaxY=HallY[3];
} else if (HallY[3]<MinY){
    MinY=HallY[3];
}

HallX[4] = analogRead(pinX);
if (HallX[4]>MaxX){
    MaxX=HallX[4];
} else if (HallX[4]<MinX){
    MinX=HallX[4];
}
```

173
HallY[4] = analogRead(pinY);
if (HallY[4]>MaxY){
    MaxY=HallY[4];
} else if (HallY[4]<MinY){
    MinY=HallY[4];
}
AveHallX = (SumX-MinX-MaxX)/3;
AveHallY = (SumY-MinY-MaxY)/3;
errorX = setPtX - AveHallX;
errorY = setPtY - AveHallY;
dErrorX = errorX - prevErrX;
dErrorY = errorY - prevErrY;
integralX = integralX + errorX*0.015;
integralY = integralY + errorY*0.015;
powerX = errorX*Kp + integralX*Ki + dErrorX*Kd;
powerY = errorY*Kp + integralY*Ki + dErrorY*Kd;

// check X_PID
digitalWrite(x_ready,HIGH);
while(process_it!=true){} // if interrupts flag is set
    process_it = false;
if(received_powerX == powerX){
    powerX = received_powerX;
}

// check Y_PID
digitalWrite(y_ready,HIGH);
while(process_it!=true){} // if interrupts flag is set
    process_it = false;
if(received_powerY == powerY){
    powerY = received_powerY;
}

if(powerX>0) // need to push in -x direction
    {dWrite(in3, LOW);
dWrite(in4, HIGH);
}
else //need to push in +x direction
    {powerX=-powerX;
dWrite(in4, LOW);
dWrite(in3, HIGH);
if (powerX > 240) {
    powerX = 240;
}

if (AveHallX > 350) {
    powerX = 0; // stops power being used when no magnet
    integralX = 0; // stops integral if no magnet
}
    analogWrite(enb, powerX);

if (powerY > 0) // Push in -y direction
{
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
}
else // Push in +y direction
{
    powerY = -powerY;
    digitalWrite(in2, LOW);
    digitalWrite(in1, HIGH);
}

if (powerY > 240) {
    powerY = 240;
}

if (AveHallY > 400) {
    powerY = 0; // stops power being used when no magnet
    integralY = 0; // stops integral if no magnet
}

prevErrX = errorX;
prevErrY = errorY;

// Detect Fail

fail_status = digitalRead(fail_pin);

// Serial.print(fail_status);

if (fail_status == 1) {
sensorfail = 1;
}

//*************** Do something else **************/

// use sensor pair 1 (secondary pair of sensors)

//*************** disable power **************/
pinX=1;
pinY=0;

setPtX=167; // Hall sensor value for equilibrium state
setPtY=150;

Kp = 2;
Ki = 0.5;
Kd = 9.74;

int i = 0;

prevErrX=0; // Previous error used to calculate derivative
prevErrY=0; // previous error for calculating derivative

integralX = 0;
integralY = 0;

digitalWrite(3,HIGH);

while(sensorfail==1)
{
    if (i > 500)
    {
        Ki = 0.13;
    }

        /****************************** Calculate PID ************************************/
        /****************************** Calculate PID ************************************/
        /****************************** Calculate PID ************************************/
        /****************************** Calculate PID ************************************/

        HallX[0] = analogRead(pinX);
        MaxX=HallX[0];
        MinX=MaxX;
        HallY[0] = analogRead(pinY);
        MaxY=HallY[0];
        MinY=MaxY;

        HallX[1] = analogRead(pinX);
        if (HallX[1]>MaxX){
            MaxX=HallX[1];
        } else if (HallX[1]<MinX){
            MinX=HallX[1];
        }
        HallY[1] = analogRead(pinY);
        if (HallY[1]>MaxY){
            MaxY=HallY[1];
        } else if (HallY[1]<MinY){
            MinY=HallY[1];
        }

        HallX[2]=analogRead(pinX);
        if (HallX[2]>MaxX){
            MaxX=HallX[2];
        } else if (HallX[2]<MinX){
            MinX=HallX[2];
        }
        HallY[2] = analogRead(pinY);
        if (HallY[2]>MaxY){
            MaxY=HallY[2];
        } else if (HallY[2]<MinY){
            MinY=HallY[2];
        }

    else
        { }
MinY=HallY[2];

HallX[3] = analogRead(pinX);
if (HallX[3]>MaxX){
    MaxX=HallX[3];
}else if (HallX[3]<MinX){
    MinX=HallX[3];
} HallY[3] = analogRead(pinY);
if (HallY[3]>MaxY){
    MaxY=HallY[3];
}else if (HallY[3]<MinY){
    MinY=HallY[3];
}

HallX[4] = analogRead(pinX);
if (HallX[4]>MaxX){
    MaxX=HallX[4];
}else if (HallX[4]<MinX){
    MinX=HallX[4];
} HallY[4] = analogRead(pinY);
if (HallY[4]>MaxY){
    MaxY=HallY[4];
}else if (HallY[4]<MinY){
    MinY=HallY[4];
}


AveHallX = (SumX-MinX-MaxX)/3;
AveHallY = (SumY-MinY-MaxY)/3;

errorX = setPtX - AveHallX;
errorY = setPtY - AveHallY;

dErrorX = errorX - prevErrX;
dErrorY = errorY - prevErrY;

integralX = integralX + errorX*0.01;
integralY = integralY + errorY*0.01;

powerX = errorX*Kp + integralX*Ki + dErrorX*Kd;
powerY = errorY*Kp + integralY*Ki + dErrorY*Kd;

/*********************************************/
/******* Receive and Compare PID ***********/
/*********************************************/

// check X_PID
digitalWrite(x_ready,HIGH);
while(process_it!=true){}    // if interrupts flag is set
digitalWrite(x_ready,LOW);
process_it = false;

if(received_powerX == powerX)
{
    powerX = received_powerX;
}

// check Y_PID
digitalWrite(y_ready,HIGH);
while(process_it!=true){}    // if interrupts flag is set
digitalWrite(y_ready,LOW);
process_it = false;

if(received_powerY == powerY)
{
    powerY = received_powerY;
}

/****************************************************************************
/******* Calculate PID **********/
****************************************************************************/

if(powerX>0) // need to push in -x direction
{
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
}
else // need to push in +x direction
{
    powerX=-powerX;
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
}

if(powerX>240)
{
    powerX=240;
}
if(AveHallX>350)
{
    powerX=0;       // stops power being used when no magnet
    integralX=0;    // stops integral if no magnet
}
analogWrite(enb, powerX);

if(powerY>0) // Push in -y direction
{
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
}
else // Push in +y direction
{
    powerY = -powerY;
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
}

if(powerY>240)
{
    powerY=240;
}
if(AveHallY>350)
{
    powerY=0;        // stops power being used when no magnet
    integralY=0;     // stops integral if no magnet
}
analogWrite(ena, powerY);

prevErrX=errorX;
prevErrY=errorY;
fail_status = digitalRead(fail_pin);

if (fail_status == LOW)
{
  sensorfail=0;
  Serial.print(sensorfail);
}

i++;

}
APPENDIX 15: Temperature Control Software

/*
Temperature Control by James Ballard
Hull University 2014
*/

#include <dht11.h>.  // DHT11 Temp and Humidity Sensor Library
dht11 DHT;
#define DHT11_PIN 4  // Sensor reading on Arduino pin 4

float td=10; // Dew Point
int checkStatus; // DHT11 Sensor Status
int tmax = 60; // Maximum working temperature
int tmin = 5; // Minimum working temperature
int temp = 20; // Starting temperature
int hum = 50; // Starting Humidity
int pwm = 0; // pwm output off

void setup() {
    pinMode(2, OUTPUT); // pwm out put on pin 2
    Serial.begin(9600);
}

void loop() {
    checkStatus = DHT.read(DHT11_PIN); // Read Temperature and Humidity

    if (checkStatus!=DHTLIB_OK) { // Detects Sensor failure
        // Display warning
        Serial.print("Warning Message");
        // Execute safe shutdown sequence
    }

    temp=DHT.temperature; // Temperature measurement

    if (temp <= 21) {
        analogWrite(2, 0); // Turn off Fan
        pwm = 0; // Fan off
    }

    if (temp > 20){ // If temperature is above 20 Deg C control fan speed
        pwm = map(temp, 21, 30, 75, 255); // Use pwm = 75 to start fan running
    }

    if (temp > 30) pwm = 255; // If temp over 30 Deg C keep fan on full
    analogWrite(2, pwm); // Set fan speed
}

    if (temp >= tmax) {
        // Display warning
        Serial.print("Warning Message");
        // Execute safe shutdown sequence
    }

    if (temp < tmin) {
        // Display warning
    }
Serial.print("Warning Message");
// Execute safe shutdown sequence
}

delay(1000);  // 1 second minimum delay between temperature readings
}