Project Number: RXV AMT3

Magnetoresistive Sensors Applied to the Development of a Three Dimensional Computer Mouse

A Major Qualifying Project

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Bachelor of Science degree

Submitted by:

_____
Andrea L.M. Baker

_____
Wojciech Krajewski

_____
Daniel I. Wallance

Date: October 18, 2003

Sponsoring Agency: AMT Ireland, Limerick Ireland

Approved:

_____
Professor Richard Vaz, MQP Advisor

Submitted to:

Project Advisor: Richard Vaz, WPI Professor
On-Site Liaison: Seamus Clifford, Materials Researcher, AMT Ireland
On-Site Liaison: Essa Jafer, Research Engineer, ECE-University of Limerick

Abstract

AMT Ireland challenged us to envision and create a small range three-dimensional wireless position-tracking prototype for the development of a 3D computer mouse. After surveying existing technology, we chose magnetoresistive sensors as the tracking component and a magnet for the target. Our proof of concept prototype, utilizing real time data acquisition and display, demonstrated repeatable position tracking in two dimensions. This led us to believe that an expanded system using multiple sensor planes is viable towards a three-dimensional mouse.

Authorship Page

This project was completed at the University of Limerick, Ireland during the months of August through October in the year 2003. Professor Richard Vaz of Worcester Polytechnic Institute held the role of advisor. Seamus Clifford and John Harris from AMT Ireland were the project's sponsors. Andrea Baker, Wojciech Krajewski, and Daniel Wallance distributing their resources evenly created both this report and a functional proof of concept prototype. Although their technical backgrounds and experience levels varied, they brought forth an equal and combined effort in producing the deliverables. Without the cooperation of all participants, this project would not have been possible.

Acknowledgements

Upon completing this project, we would like to thank our devoted sponsors, Seamus Clifford and John Harris and the rest of AMT Ireland. In addition, we would like to thank our advisor from Worcester Polytechnic Institute, Professor Richard Vaz, for his utmost support. Leon Cavanagh, Eddy Moore, and Colm Cunniffe, our lab buddies, for their faith, encouragement and humor. Professor Khalil Arshak for his generous support towards our entrepreneurial endeavors. Noel Guinane, James Keane and all the guys in the stores for their benevolent assistance in acquiring parts and equipment. John Clifford and John Harris for always willing to lend a helping hand. Professor Jim Demetry for his exclusive visit to our project site. Finally, this project would not have been possible without the loving and embracing support of Charlotte Tuohy.

Table of Contents

List of Tables

List of Figures

Executive Summary

Over the last decade, the power and functionality of computers has evolved at a phenomenal rate. Their capabilities are skyrocketing, especially in the realm of three-dimensional graphics and gaming. Undoubtedly, this trend will continue bringing about even faster personal computers, more easily emphasizing three-dimensional applications. However, the interesting fact is that this trend in computing technology has not substantially affected human computer interaction. Most computer users still interact through the standard two-dimensional mouse invented in 1968. Although many improvements to the initial design have been made, including replacing the mouse ball with optical sensors, adding scroll wheels and removing the wired connection to the PC, the fundamental design remains. With all this in mind, we ask: Why has no one invented a mouse to track our hand motion in three dimensions, linking our world with that of three-dimensional computer simulation?

AMT Ireland, a manufacturing and technology consultancy service at the University of Limerick, attempted to take today's mouse off the mouse pad and into the third dimension. With this goal in mind, they first asked us to find a technology that could be utilized toward the development of a three-dimensional mouse. If one was available, AMT would then like us to design and build a proof of concept system to test this technology for feasibility. AMT specified technological restraints whereby the tracking system cannot utilize optics, a wireless pointing device would be ideal, the system should be expandable for multiple object tracking, and the technology selected should be novel to the application of a 3D mouse.

From our first project goal, "find a technology that could be utilized toward the development of a 3D mouse", we immediately realized that neither AMT nor ourselves knew of a single product on the market that fulfills the requirements. This led us to focus our initial research on discovering current products that we could apply towards a 3D mouse.

Early on, we considered RF systems that utilize time delay measuring such as GPS. Unfortunately, as our operating area is very small, accurate time delay sensing would be extraordinarily difficult to implement. Another promising technology we discovered but eliminated early on was a Capaciflector. When an object is brought near the sensor, the object affects its dielectric constant, changing the measured capacitance.

We did not consider this device further, as a user's hand would interfere with the sensing system.

Continuing to seek a suitable technology, we came across a product able to track the movement of an electronic pen over a grid of inductors. Wacom Technology developed this system, termed an ArtPad, as a replacement for a computer mouse. A user holds the pen, containing an inductive circuit, above the pad that both powers the oscillating circuit inside the pen and tracks its position. Even though the computer cursor only moves in a two dimensional plane, the ArtPad is able to track the position of the pen up to several inches above the pad. The vertical distance is not very large and Wacom never intended the device for full three-dimensional tracking but we found the technology promising.

Another system using inductive sensing was developed at the Massachusetts Institute of Technology Media Lab under the supervision of Joseph Paradiso. Unlike Wacom, the MIT system developers fully intended their invention to track a target in three-dimensional space. It utilizes six coils mounted on each side of a cube to track the position of a target object in 3D. The Media Lab's design fulfills all of our requirements; however, AMT Ireland requested a novel approach and therefore we refrained from attempting to design a device based on their invention. Nevertheless, the knowledge we gained from Wacom and MIT Media Lab's research was invaluable. The benefits of utilizing inductive and electromagnetic systems to track an object on a small scale became apparent. The major benefit we realized was that the human body's effect on a magnetic field is minimal. Since our application's success rests on the fact that the user will have to touch the mouse in order to move it, this further attracted us to electromagnetics.

Once we shifted our focus towards researching electromagnetic principles, we discovered a type of sensor that produces a voltage proportional to the field strength and orientation of a magnetic field. These sensors are termed magnetoresistive, and their current applications include extremely accurate digital compasses, land mine and metal detectors. As we increased our understanding of these sensors through examining the accuracy required in these applications in conjunction with surrounding sensor literature, we found them extraordinarily promising. This warranted the ordering of one sensor for physical testing. During our experiments, we concluded that moving a magnet near the sensor created a change in output voltage dependent upon distance and orientation of the magnet. Unfortunately, the range of the sensor was rather limited, which we attributed to

the fact that our tests did not include suggested external circuitry to increase sensitivity. We considered our findings promising enough to choose magnetoresistive sensors as our novel approach for the heart of our three-dimensional mouse.

The second goal AMT Ireland set forth was, "to design and build a proof of concept system in order to test feasibility of our chosen technology." The system we designed needs to be flexible and have the ability to characterize the behavior of the magnetic sensors in response to the target object's position. For the final design of a 3D computer mouse, we envisioned a small, three-sided box, where the two adjacent sides connect to the bottom. An object that emits a consistent magnetic field, such as a standard bar magnet would act as the pointing device. This pointing device needs to be small so that it would be comfortable to hold or wear as a ring. The magnetic sensors mounted to each side of the box would detect the magnet's field strength and direction as a relation to its proximity and orientation. This information will then be used to triangulate the magnet's position so that it can be mapped to the computer cursor.

Focusing on creating a proof of concept for our envisioned system, we developed a scaled down version to test the magnetic sensors' feasibility. Our system contains three major components forming a real time data acquisition and display system. These components are a magnetic sensor assembly capable of sensing magnetic field strength in three orthogonal directions, a microcontroller, and real time data acquisition and display software running on a PC. The microcontroller adjusts the sensors' sensitivity, as well as sampling the output before sending it serially to the PC. Once the PC receives the sensor data, our software processes, logs, and generates a display representing the data in real time. We "home built" all portions of the data acquisition system, meaning we did not use any off the shelf software to acquire, transmit, collect, or display sensor data. We did so for low-level control permitting us to quickly add testing algorithms to our design after we deemed the system operational.

The final stage in fulfilling our project goals required determining technological feasibility. Extensive system experimentation is necessary to characterize system response to multiple magnet orientations and to demonstrate multidimensional tracking. Until this point, we were able to accomplish 2D tracking of a magnet within a plane while keeping the magnet's distance and orientation from the sensors constant. We accomplished this by first establishing a square centimeter grid, 24 cm wide by 33 cm long, then characterizing the sensors' reaction to the magnet at a set distance of 10 cm above the grid and magnet orientation with the south pole facing the grid. Utilizing the

characterization data collected, we excited the sensors by placing a magnet at a specific location within the established plane. Then we mapped the sensors' output to the characterization data, cross-referencing that data to the magnet's current position. Our characterization was accurate enough to find the position of the magnet at any location within the test grid. We only incurred occasional errors when we placed the magnet at the edges of the grid.

The two-dimensional system demonstration illustrates the type of system characterization that will need to be preformed in order to add the third dimension to the system. Specifically, this testing will further characterize the system's response to multiple magnet orientations and positions. Our hope is that our system will provide AMT Ireland and the University of Limerick with an excellent platform to facilitate their development of a three-dimensional mouse.

Chapter 1 – Introduction

Prior to 1968, when Doug Engelbart and 17 researchers held the first public demonstration of the computer mouse, the keyboard was the only user-friendly method for interacting with a computer (Stanford University, 2003). Although the keyboard is efficient, many people prefer a more instinctive form of pointing to objects and not having to describe them with tedious textual commands. In our daily lives, we use pointing as a method to request items, such as from a store shelf (Brian, 2000). The computer mouse facilitates this by connecting people's natural gestures to otherwise unintuitive computer commands. A user with a mouse can now position the cursor so that it points to an item on the screen, which the computer interprets as a desire for that item. Moving the mouse has become a natural, everyday task, no different than moving one's hand.

Engelbart's invention of the computer mouse was a large advancement into natural forms of human computer interaction. However, the mouse still misses a critical feature. Even though a computer can display three-dimensional space on a screen, users are not able to interact with all three axes at the same time. This limitation stems directly from the original mouse design. Engelbart's patent describes his invention as an "X-Y position indicator for a display system (Engelbart, 1970)." There is no mention about the third dimension. Therefore, the question remains: Whatever happened to the Z-axis, and can we utilize it?

Over time, institutions and corporations tried and even came close to answering this question. At the Massachusetts Institute of Technology Media Lab, individuals supervised by Joe Paradiso developed an electromagnetic coil based system capable of displaying the position of an inductive circuit in three dimensions (Hsiao, 2001). Wacom Technology developed a system that uses an electronic pen to mimic the functions of a computer mouse. Although the pen controls the cursor only in two dimensions, the system tracks the pen's position in three dimensions, but to a limited range (Wacom Components, 2002). Even with these products, a vacuum remains to be filled through the development of a system focusing on a three-dimensional computer mouse.

AMT Ireland, a manufacturing and technology consultancy service at the University of Limerick, sponsored our project to explore whether it is possible to develop a three-dimensional computer mouse using a novel approach. From their question, we formed our objective:

> To demonstrate the feasibility of a selected technology for tracking and displaying the position of small objects in a given three-dimensional space, even when other non-ferrous bodies are present.

Given that we based our project on expanding the functionality of the existing computer mouse, our product requirements and constraints stemmed from characteristics typical to an everyday mouse as well as developmental requirements imposed by AMT. The initial requirement was with respect to the shape and size of the mouse or target object. With a normal computer mouse, comfort often dictates the shape and size. Our application requires the user to hold the mouse in the air so that he or she can move it in all three dimensions. Consequently, the mouse has to be light and small enough so that the user does not experience unnecessary strain while moving it. To meet this requirement, we settled on a size approximately the same dimensions of a pea enabling the user to wear it on their index finger as a pointing device. Consequently, the mouse itself or the unit that we are tracking must be completely passive and therefore cannot contain any active components such as a battery. Another size related issue was the operating area, or the region in which the user will move the 3D mouse. As with strain relating to the size and weight of the object, a large operating area can also cause excessive muscle pain. Therefore, we decided on a roughly cubic operating area in the range of six liters in volume.

Although alleviating unnecessary stress on the user is essential for successful operation and continued use, ensuring a transparent transition from using a two dimensional mouse to a three dimensional mouse is equally significant. A noticeable aspect of the current two-dimensional mouse that must be transitioned to our three dimensional mouse is the real time coordination between the movement of the mouse and the position of the cursor on the screen. Accordingly, we placed a strong emphasis on the requirement that our system must be able to operate in real time.

The final technological requirement was not derived by criteria based on ease of use, but instead was requested by AMT to ensure the system's expandability beyond the initial application of a three dimensional computer mouse. AMT saw the technology

behind the tracking system and the ability to monitor the independent position of multiple objects as the two main limitations of an expandable system. As a result, our design's tracking system has to be free of optical technologies to ensure that any future development of the system is not hindered by a line of sight requirement. Additionally, the tracking system has to be able to differentiate between multiple target objects so that the individual position of each object can be tracked. Multiple object differentiation would allow future developers to apply our product to applications such as an invisible keyboard in which tracking more than one object is required.

From these requirements, we will complete a proof of concept system. We hope that our system will demonstrate that a three dimensional mouse is possible and practical. Although the computer cursor's position will not move based on our system's output data, we will instead form a plot proving that we can correlate the information from the magnetic sensors to a given position on the screen.

We hope that future developers will continue our work through the construction of a complete system, fully linked to the cursor. Such a system can provide software programmers with the opportunity to utilize the available third dimension, potentially broadening and enhancing human computer interaction.

Chapter 2 – Background Research

Developing a demonstration technology for a three-dimensional mouse, one that satisfies all of the prototype requirements and constraints called for an extensive exploration of small-scale position tracking technologies and their practical applications. Each requirement presented certain needs for its successful development, whether ensuring that the free-floating body remained passive or preventing the interference of non-ferrous materials. The larger, overarching constraint was on the tracking system; optical sensors could not be used. Therefore, we explored other solutions to meet this stringent requirement. The research included in this section depicts these solutions as three major groups of non-optical tracking or sensing technologies, capacitive sensing, inductive sensing and radio frequency delay. We examined each one, depicting the promise and problems along with exploring existing solutions.

2.0 Inductive and Magnetic Field Sensing

We identified and explored three non-optical approaches to position tracking, namely inductive and magnetic field sensing, capacitive based position tracking, and using time delay sensing.

The first of these is based on the principle of detecting the strength of a magnetic field. There are several ways of doing this, including using magnetic field sensors, detecting the induction of eddy currents in metals, and detecting the resonance of a coupled inductor-capacitor (LC) circuit. The magnetic field sensors detect the position and orientation of a magnetic field from a magnet or oscillating LC circuit. Eddy current sensing on the other hand, exploits the properties of induction to cause the flow of eddy currents in the conductive elements of the target object. The eddy currents can then be picked up to determine the location of the object. Detecting LC circuits involves transmitting a high power signal at the resonant frequency of the LC circuit, causing it to oscillate. The oscillations can then be picked up to detect the presence of that circuit. Most applications utilizing inductive sensing are designed to find an object or map an area within a magnetic field. Some existing applications include magnetic anomaly detection, ground surveying, and high accuracy velocity measurements.

2.0.1 Magnetic Field Sensors

Applications using magnetic field sensors typically use sensors that come as prepackaged integrated circuits, and which provide the strength of a magnetic field in one, two or three directions. A three-axis magnetic field sensor outputs three voltages proportional to the magnetic field strength in three orthogonal directions. The sensitivity of the sensor typically allows very small magnetic signatures to be measured. The range over which an anomaly can be measured is determined by using several sensors to record both field magnitude and field gradient (Crossbow Technology, Inc., 2003).

A major application of these sensors is in electronic compasses. The precision and high sensitivity allows for detecting a change in the earth's magnetic field. Once such change is the difference perceived. Another significant application of these sensors has been in detecting the position of teeth in a rotating gear. When a magnet is placed in front of the tooth of a gear, its magnetic field shifts towards the gear. The shift in this field can be detected by using a magnetic sensor.

2.0.2 Detecting Eddy Currents

Although magnetic field sensors played a significant role in developed products, other inductive sensing methods have used techniques not related to prepackaged integrated circuits. One such technique is based on the principle of detecting the presence of eddy currents induced in a metal object.

First, eddy currents must be induced in a metal object by varying the magnetic field around it. These small currents then produce an inverse magnetic field which can be picked up by a sensor coil. Eddy currents are electric currents formed in metallic objects when the object moves through a magnetic field (Case Western Reserve University, 2000). Therefore, a time varying magnetic field is only required if the object is stationary. Inductor sensor coils, used to induce eddy currents and to pick up any response are commonly made out of copper or nonferrous metals. The material of the target object must be a conductive metal to allow the formation of a magnetic field in the target object. It is important to note that as the current induced in the inductor coil intensifies the result is an increase in the magnetic field between the inductor and the target.

The effect allows the generation of a magnetic field large enough to cover a desired region simply by increasing the current. This field is a combination of induced

current through the inductor and the magnetic field in the target (Welsby, 2003). Increasing the current to a preferred level is also important in position tracking since the magnetic field decreases with distance. The current that forms in the target from the initial inductor are eddy currents limited to the regions in which the magnetic field is present (Welsby, 2003). However, this magnetic field propagates in all directions. The magnetic fields from the eddy currents are counteractive to the original magnetic field produced by the inductor coil resulting in a detectable variation. As a result, the impedance and voltage of the coil change proportional to the magnetic field strength based on the distance between the coil and the target (Welsby, 2003). Therefore, to determine the position of the target, a simple measurement of the coil's voltage is taken (Suffi, 2001).

One highly common use of eddy current detection is in metal detectors. Metal detectors contain a coil, which transmits a time varying magnetic field. Any metallic object buried beneath the surface picks up the field forming eddy currents. A coil, either the same one or another, on the metal detector, then retrieves the variation caused by the eddy currents. The result is a reading of whether a metallic object is present. Figure 1 presents a diagram of a similar application in detecting the presence of a metallic reinforcement underneath a concrete structure.

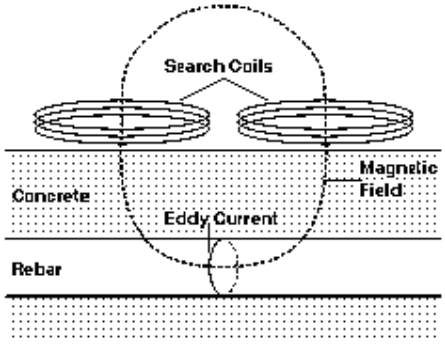


Figure 1 – Eddy Current Metal Detection Configuration (Protovale Oxford Ltd., 2002)

Another application using the principles of eddy currents is measuring the velocity of a high-speed object such as a bullet. In this application, two inductive rings are separated by a specified distance. The object or bullet sequentially passes through the center of each ring. The first ring in the sequence emits a changing magnetic field as the object passes through the center of the ring and therefore the field causes eddy currents to form in the bullet.

Because of the eddy currents, a counteracting voltage develops in the ring proportional to the bullet's distance. When the bullet is in the center of each ring, the magnitude is greatest. As a result, the bullet's velocity can be calculated by measuring the time between the voltage peaks of each ring (Spohn, 2003).

Figure 2 – Eddy Current Sensing for Velocity Measurement (Spohn, 2003)

An important application of inductive sensing, although not directly related to detecting eddy currents, is worth noting. The significance exists because the detection method uses a uniquely produced magnetic field but still follows the techniques behind inductive sensing. Instead of producing an electric current through a coil to form the necessary background field for inductive sensing, this method uses the earth's magnetic field. The target object causes a disturbance or an anomaly in the earth's magnetic field detectable by a device called a gradiometer. Figure 3 below displays one such gradiometer.

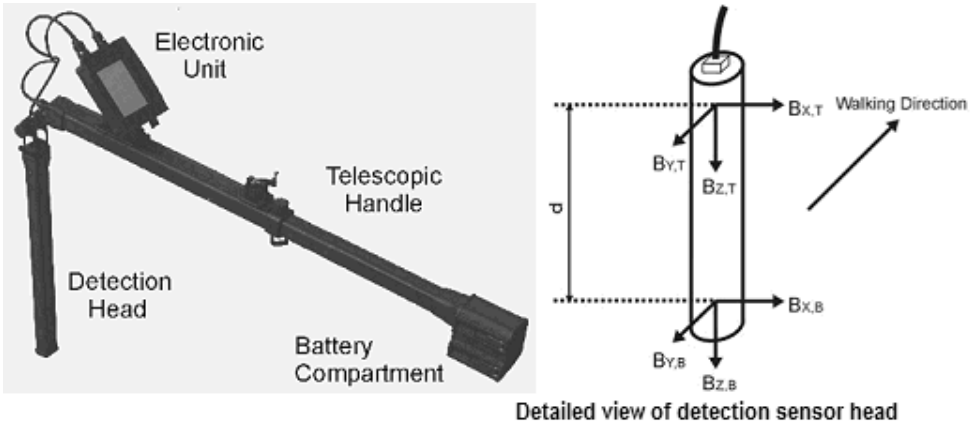


Figure 3 – Magnetic Anomaly Detection System (Hochreiter, Schrottmayer, 2003)

A gradiometer operates on the principle that in a uniform magnetic field, such as the earth's magnetic field, two identical and perfectly aligned sensors will give identical outputs. These outputs when subtracted will be zero. If a small field change occurs within the uniform field, then the output will change as a function of the magnitude and direction of gradient or disturbance. These gradients arise from the disturbance of a magnetic anomaly caused by the change in position of a metallic object within the gradiometer (Fat Quarters Software & Electronics, 2000). A significant aspect of these gradiometers is the sensors used. The sensors are the same three-axis magnetic field sensors previously discussed.

2.0.3 LC Resonant Response

The principles behind inductive and magnetic field sensing all involve listening for an object or a change in a magnetic field. The previously discussed three axis magnetic field sensors pick up the presence of a magnetic field. Similar to this is eddy current detection, on which the principle is to detect a change in a magnetic field caused by currents in a metallic object. A third method also exists using the same technique of listening for a change. However, this time the target object cannot be a simple magnet or a metallic object, but instead is a circuit consisting of two components. These components are an inductor and a capacitor connected together to form an LC circuit. This circuit has an interesting property whereby when the inductor is placed in a changing magnetic field, whose frequency is the resonant frequency of the LC circuit, the energy formed in the inductor resonates or shifts back and forth between the inductor and capacitor. The result is a change in the voltage of the original coil transmitting the AC signal, proportional to the distance of the LC circuit.

A unique feature of the LC circuit is the circuit will only resonate and provide a response when the transmitting signal is at its resonant frequency. This feature permits multiple objects to be detected within the same area simply by varying the frequency of the transmitting signal and listening for a response when the resonant frequency of an object is transmitted.

The principle in detecting multiple objects and being able to distinguish between each object using LC circuits was the underlying foundation for a project worked on by individuals at the Massachusetts Institute of Technology Media Lab under the supervision

of Joe Paradiso (from now on to be referred to as the MIT system). In their project, they are able to track the movement of multiple objects, each containing a unique LC circuit, within an area occupied by a changing magnetic field. The result is the object's position displayed in real time on a computer screen.

In developing this system, the Media Lab went through the development and testing of multiple theories behind picking up a response from an LC circuit. The first stage was their investigation of a ringdown tag reader.

2.0.3.1 MIT Applied Ringdown LC Tag Readers (Hsiao, 2001, pp. 13-22)

The ringdown detection mechanism utilizes the physical response properties of common EAS tags, magnetostrictors and LC tags. EAS or Electronic Article Surveillance, are electronic tags typically used in the prevention of shoplifting. These devices either contain a magnetically coupled strip of metal as in a magnetostrictor or an LC circuit. A sales representative, such as in a clothing store, would attach one of these tags to an item for sale. If he or she did not deactivate the tag, then upon exit a reader alerts employees of a shoplifter. These EAS tags are available in both active and passive forms.

The tags are also typically designed to have a high-Q property to create a strong and consistent resonance response. The response from the tag can be described as "ringdown" due to its characteristic exponentially decaying resonance response.

Three major steps comprise the detection mechanism of a ringdown tag reader as detailed by MIT:

1. "A magnetic pulse is transmitted by the reader coil at the resonant frequency of the desired tag."
2. "The tag, upon exposure to this pulse, produces its own mechanical or electronic resonance response, which will continue for a short while as the tag damps out after the transmitted pulse stops."
3. "Since the tag produces its own weak resonant signal, the magnetic drive at the reader can be turned off once the tag rings up, and the reader "listens" for this small but detectable response."

An effective reader must have the ability to transmit the energizing magnetic pulse, which "pings" each tag and then stops and listens for each tags' response. In order to differentiate between multiple tags the reader has to transmit each tag's individual frequency.

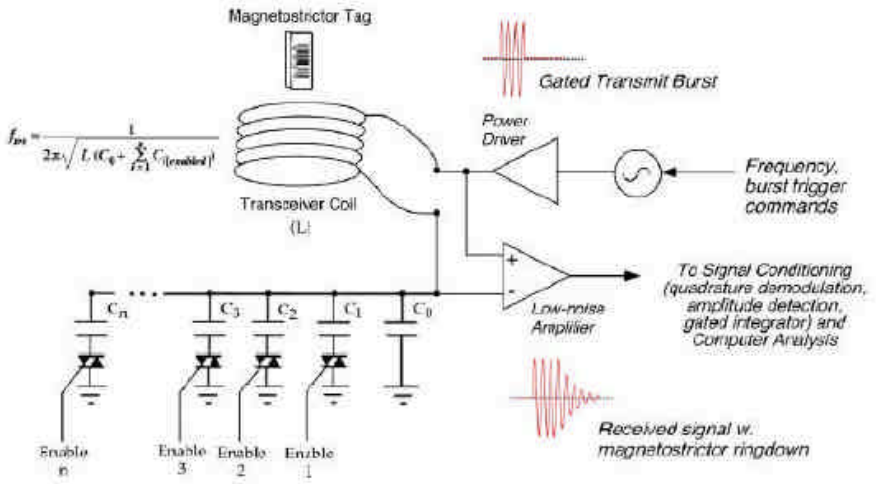


Figure 4 – MIT's Ringdown Tag Reader(Hsiao, 2001, p. 13)

The first step in the detection of a ringdown tag reader involves transmitting a magnetic pulse or "ping" via the reader coil. A problem arises when determining how to drive the same amount of current through the reader coil at each of the tags resonance frequencies. The reader coil has its own distinct resonance curve that affects the power of the transmitted signal. Frequencies transmitted further from the reader coil's resonance frequency produce weaker transmitted signals. MIT solved this problem by physically shifting the resonance curve of the reader coil to the resonance frequency of each tag before pinging. They used a tuned capacitor array to create this effect by switching in a different capacitor before a ping was sent at each frequency of interest.

Once the tag has been pinged, the transmitter switches to reader mode. This means damping the ping quickly and then connecting the reader circuitry. The ringdown response of any tags at the ping frequency near the reader will be detected. The resonance capacitor is then disconnected from the reading coil so the received signal is effectively broadband. A differential amplifier first amplifies this broadband signal, and then the received signal is identified to determine if it equals the pinged frequency. Finally, in order to determine proximity to the coil, as in distance measurement, the total power of the ringdown response is determined.

MIT accomplished this event sequence by first by filtering the received signal and then sending it through a quadrature demodulator designed to bring the signal down to baseband. Using this method, they determined the magnitude of the signal that was in

phase and the magnitude out of phase by ninety degrees with the original transmitted signal. They had to look at both the in-phase and the ninety-degree phase component due to the physical construction of the tags. "The tags ringdown responses could be subject to dynamic and unpredictable phase shifts due to variations in the tags' resonance peaks from small mechanical differences between tag packages." In the end the demodulated signal was significantly narrowbanded ("it contained mainly the components of the received signal that corresponded to the expected transmit frequency.") Finally, they collected the power of the received signal by squaring each quadrature component and then adding them. A gated integrator was used to accumulate the power signal. This produced a single voltage whose magnitude corresponded to the net magnetic coupling and hence proximity and inclination of a single tag relative to the coil.

The main limitation associated with MIT's use of ringdown detection was the problems associated in detecting multiple objects. Although the system was able to detect multiple LC circuits, the number of hardware components and the complexity increased with each additional tracked target. Therefore, they looked into other methods with a set hardware complexity regardless of the number of targets. The technology they stumbled upon was swept frequency detection, sweeping through a continuous range of frequencies instead of a discrete range.

2.0.3.2 MIT Applied Swept-Frequency LC Tag Readers (Hsiao, 2001, pp. 25-40)

In contrast to the ringdown tag reader, swept frequency detection sweeps a range of frequencies instead of pinging select, discrete frequencies. When the field produced by the reader passes through the tag's resonance frequency, the tag draws energy from the field causing a small but detectable change in the inductance of the coil, which can be read by a change in the voltage or current.

A key statement made by MIT explains the basic theory behind why you must sweep the frequency not just hold it constant to detect a tag:

> A coil transmitting at a constant frequency will only see a change in the coil impedance as a tag with a resonance at that frequency is brought into the field; the further away the tag, the smaller the tag's response will be. Tags resonant at other frequencies will have little or no impact on the coil impedance. By sweeping the frequency instead of leaving it constant, however, any tags in the field produce sharp alterations to the coil properties when the sweep reaches each of their resonances. These changes can be easily emphasized and detected via simple analog signal processing (Hsiao, 2001, p. 25).

MIT designed their system so that they can measure the relationship between the proximity and orientation of a tag and the magnitude of the change in the received signal. The magnetic coupling between the tag and the sensor coil at the tag's resonance frequency is dependent upon how fast the sweep occurs, how much energy the tag draws from the field and the proximity and orientation of the tag.

In order for the system to update the target's position in real time, MIT determined that a sweep rate of 30 Hz was adequate. They selected a sweep range from 40 kHz to 400 kHz allowing for detection of lower resonating magnetostrictors and higher resonating home built LC tags. The previous information along with the swept frequency range and rate only comprise a small portion of the whole LC system. For a more complete understanding, the larger units of the swept frequency system must be examined individually. These appear in the block diagram of the tag reader system presented in Figure 5. The first of these units is the transmission circuitry closely followed by the receiving unit.

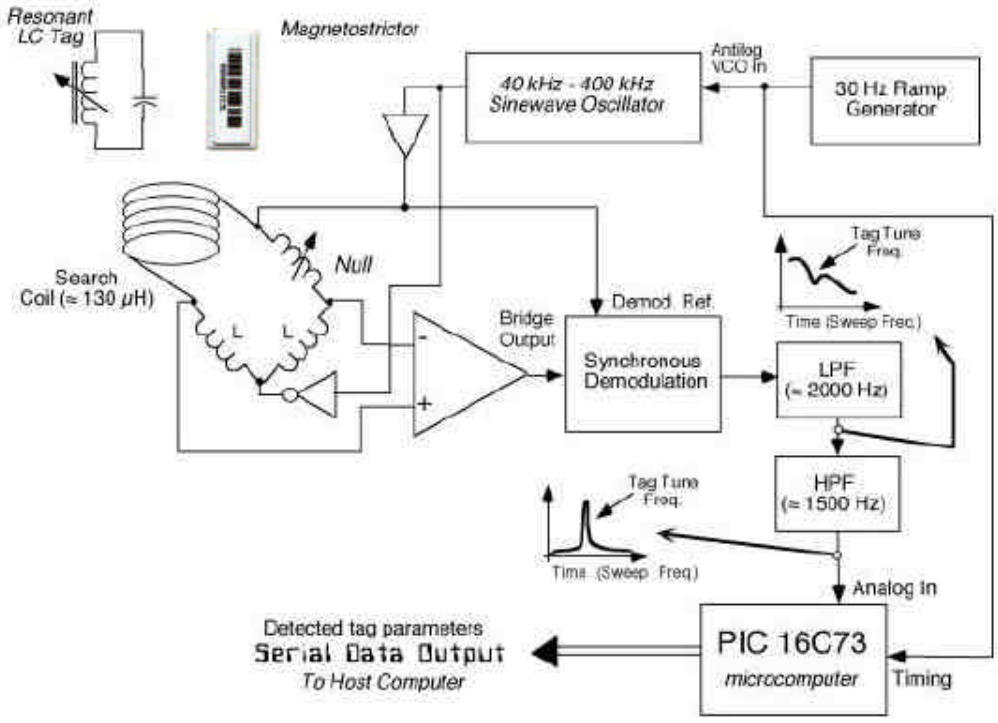


Figure 5 – MIT's Swept Frequency Tag Reader Block Diagram (Hsiao, 2001, p. 28)

The transmission circuitry of the tag reader system in Figure 5 consists of the sine wave oscillator, the 30 Hz ramp generator and the search coil. MIT used the voltage-

controlled sine wave oscillator to implement the frequency sweep signal. The input to the oscillator was originally a linear ramp function but they found that "the responses of tags at lower frequencies tended to be weaker than the responses of tags at higher frequencies". Therefore, they replaced the linear sweep with an exponential one. This allowed the frequency sweep to spend more time at lower frequencies than at higher frequencies providing the LC tag with adequate and consistent time to react to the search at all frequencies. The final transmit circuitry was comprised of an exponential sine wave sweep from high to low frequencies across the search coil. After the transmission circuitry, the next major component of the swept frequency system is the receiver circuitry. The receiver circuitry is used to listen for a response from the LC circuit once a signal transmission occurred. In the block diagram in Figure 5, the receiver circuitry is comprised of the demodulator, the low pass filter, the high pass filter and the microcomputer.

The receiver circuit's goal was to correlate the resonant frequency and proximity to the sensor coil of each tag in the system. This was accomplished by first detecting the slight shift in the sensor coil's properties caused by the presence of a tag at its resonance frequency. Then by synchronously demodulating the received signal to collect pure tag response data. Finally, a microcomputer transmitted each tag's proximity information to a PC for processing.

The slight shift in the senor coil was actually a change in inductance. MIT detected this by using an inductive Wheatstone bridge with a differential amplifier as shown in Figure 5. The bridge was balanced when there was no disturbance in the search coil's magnetic field. When the LC tag entered the field, the differential amplifier detected the bridge imbalance with an output as an absolute voltage referenced to ground. At this point, the synchronous demodulator removed the carrier from the signal resulting in an output brought to DC. A bandpass filter was used to attenuate background noise to "reduce the slow baseline shift" and enhance proximity information. The circuit then sent the filter's output to an analog to digital converter for processing by a computer.

Although MIT's development of a swept frequency tag reader proved more useful than their development of an applied ringdown system, there were still limitations. The main limitation had to do with their use of a single coil configuration.

The single coil swept frequency reader is capable of producing very useful results; however, there are still associated problems. The main problem is that the magnetic field around such a coil is not uniform. In the center of the coil, the field is perpendicular to

the coil's plane, but not very strong. As the distance from the coil's center to rim decreases, the field increases until a peak in field strength is reached upon contact with the coiled wire.

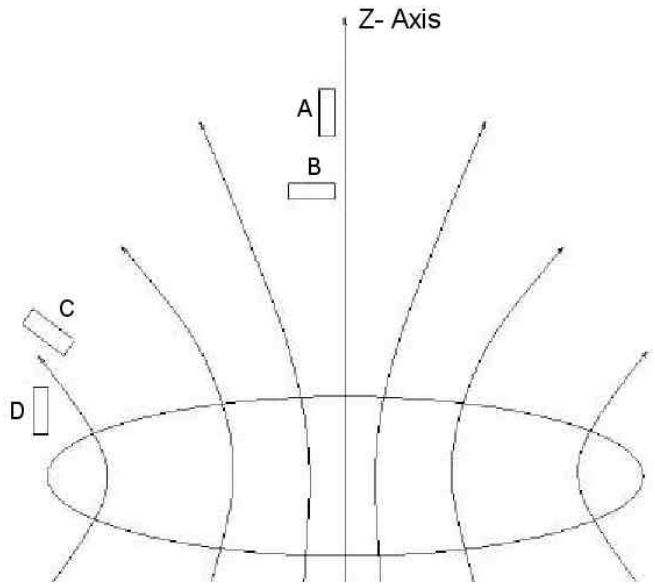


Figure 6 – Field Curvature Around a Circular Magnetic Coil (Hsiao, 2001, p. 56)

A second issue also exists, whereby the field curves as the distance from the plane of the coil increases, similar to that of a toroid as seen above in Figure 6. LC tags only respond to fields that are parallel to their orientation. Therefore, if a tag is near a coil with a strong field, but positioned perpendicular to the field, such as tag D in Figure 6 above, the reader will see a week response from the tag and believe it is far away, or maybe not at all. Similarly, if the tag is moved far above the coil such as tag A, and held parallel to the field, its disturbance will be read with the same strength. The uneven field makes it impossible to detect whether a tag is perpendicular and close to the field or a great distance away and parallel to it. As a result, it is not possible to gain any understanding of the correlation between orientation and position of the tag at any time. To make this possible, MIT added multiple coils to the assembly, expanding the system's capability.

Although MIT's system used inductive sensing to determine the position of an object, this is not the result of a limitation on the methods applicable to position tracking. In fact, many physical principles exist that can be applied to positioning systems. One of these is with capacitance.

2.1 **Capacitive Sensing**

Capacitance and in particular capacitive plates can be used for sensing the position of an object in several ways. For this, two key methods exist. One uses two parallel plates separated from each other in the same configuration as an air capacitor. The other, originally developed by John Vranish for use in NASA projects, allows a capacitive field to extend outwards from two overlapping plates. As a result, the need for a second plate at a distance from the first plate is eliminated. A capaciflector uses this method in the development of a capacitive sensor (Samby, 2003).

The parallel plate method is simple and is based on the theory of parallel plate capacitor design. Two conductive elements parallel to each other with a non-conductive material in between form a typical parallel plate capacitor. Several changes can be made to the configuration altering the amount of charge the plates can hold. To increase the capacitance between the two plates, their size can be increased, the distance between the two plates can be decreased, or the dielectric material can be changed. One of these principles is applicable for use in tracking the location of an object. For tracking an object in a fixed space, the size and position of the parallel plates forming the boundary of the tracking region cannot be changed. Therefore, the only applicable method in changing the capacitance is to cause a disturbance in the dielectric material.

By inserting a foreign object, such as a person's hand between the two plates of a large air capacitor the properties of the dielectric can change. The result is a variation in capacitance. Measuring the capacitance change provides a reading relative to the presence of a foreign body, its size, and type of material or position. Due to the effects from the presence of different materials in the dielectric, the object's position can only be determined if the material is known.

The second method to build a capacitive position sensor is the use of a capaciflector. This method allows a flat surface to detect the proximity of objects without the need for a second capacitive plate. Its design is comparable to that of a parallel plate capacitor, but the plates do not directly interact. Instead, the two plates lay flat, one on top of the other, with the bottom plate larger then the top. A reflective material separates the two plates preventing the creation of a field directly between, resulting in a configuration similar to a parallel plate capacitor. As a result, the field bevels outward from the top plate and curls back to the bottom plate. Placing an object in the capacitor's field modifies the dielectric constant in the region. The change to the dielectric constant

causes a change in capacitance similar to the parallel plate configuration. As such, the change can be read to determine the presence of a foreign body, its size, and type of material or position (Samby, 2003).

A significant limitation of position sensing technology using capacitance, whether it is the parallel plate or capaciflector method is the detectable region. These techniques can only be implemented into systems with a relatively small active search area such as in a moderately sized container. The same limitations also apply to inductive sensing in which the need for a strong magnetic field restricts the operating area. Although these two technologies are not scalable for position tracking over a geographical region or on a global scale, other suitable methods have been developed.

2.2 **Time Delay Sensing**

Position sensing and location tracking technologies developed for use over a geographical region or on a global scale use the principle of time delay sensing. Two of the position sensing technologies based on time delay are ultrasound and GPS.

Ultrasound and sonar are two technologies each used for different applications; however, the reflection of high frequency sound waves provides the same basis for both. Sonar is commonly used on oceangoing vessels, either tracking the location of underwater fish or trying to determine the proximity to a reef or to the seabed (Elert, Yusufov, 2001). Ultrasound has also been used in location tracking. AT&T with Cambridge University developed a position tracking system based on ultrasound whereby ultrasonic pulses are transmitted from a handheld unit to mounted receivers. Although the system is not able to operate over a geographic region, it is able to provide a position accurate to three centimeters within a 10,000 square feet working area (Ward, Steggles, Curwen, Webster, 2001). Regardless of the use or operating range, the same fundamental principles governing ultrasonic waves remain the same.

To be considered an ultrasonic wave, the frequency of the wave must be above 20 kHz. Ultrasound and other acoustic waves travel at a speed of 343 meters per second in air. However, the speed dramatically increases in water to 1,482 meters per second (Lynnworth, 2000, p. 11). To detect proximity to an object, as described in the previous applications, ultrasound and sonar emit a pulse and then listen as it reflects back from the target object. Any object placed within range of the sensor distorts the propagating wave

producing an echo. For ultrasound to penetrate objects, the object must have a very homogeneous structure (Freudenrich, 2003).

Although ultrasound and sonar have provided a significant technological achievement in detecting the position of an object or position, their operating range is limited to a geographic region and does not operate on a global scale. For global operation, the transmitting signal has to be a radio frequency or electromagnetic radiation and not sound.

Electromagnetic radiation, such as radio frequency waves, travel approximately at the speed of light. This makes them very useful for communication in which delays are unwanted. However, transmission over larger distances allows for calculating the propagation of radio frequency waves. Knowing the precise time when a signal left one location and arrived at another permits calculating the distance between both locations. The Global Positioning System (GPS) uses this basic theory as the underlying foundation for its operation.

GPS operates on two frequencies, which are referred to as L1 (1575.42MHz) and L2 (1227.6 MHz) (Department of Defense and Department of Transportation, 2001b, p. 3-3). Twenty-four satellites orbit the earth, each transmitting a satellite clock, their position in reference to the earth, an almanac of their and other satellites' future positions, GPS to UTC time offset, and the current propagation delay caused by the ionosphere. A receiver uses the transmitted information to calculate its position on earth. The satellites are arranged so that at least six are always visible from any point on earth. Even with this level of complexity and provided information, a receiver can only detect its position to within thirteen meters horizontally, and twenty-two meters vertically (Department of Defense and Department of Transportation, 2001, p. 3-3). However, there are ways in which the accuracy of GPS can be increased. The technology behind GPS however, cannot be applied for small-scale position tracking in a container around six-liters in size because the time it takes for an RF signal to travel from one end of the container to another is extremely small and difficult to measure.

Chapter 3– Tracking Technology Selection

Our research into different physical principles used to sense the position or existence of an object provided us with more than one approach in building a small-scale position tracking system. All of these methods, inductive and magnetic field sensing, capacitive sensing, and signal time delay measurement, have applicable characteristics. However, many of the principles contain limitations preventing them from satisfying our prototype requirements. As a result, we had to examine which of these technologies would be most applicable for use in a three-dimensional mouse, while fulfilling our prototype requirements and those set forth by AMT.

3.0.1 Prototype Requirement Matching

As previously stated, the target application set forth by AMT Ireland is a three-dimensional, wireless, passive mouse with the possibility to track multiple objects in future system revisions. In order to choose the technology with the most potential we designed a set of criteria to which it must adhere. These conditions helped us narrow down the list of possible solutions. We developed the criteria from the prototype requirements along with other requests by AMT.

The prototype selection criteria in order of importance includes:

- The utilization of a non optical solution
- The sensors must be located outside of the vessel
- The user's hand must not interfere with the sensing device
- The target object must operate passively
- A real time display must provide the target's position
- The prototype must approach the problem in a novel fashion
- Software complexity must be kept to a minimum
- Hardware complexity must be kept to a minimum
- The physical size of the target must be "small" relative to the vessel
- The vessel's physical size must be comparable with a current mouse pad

From the three styles of position sensing we examined, time delay sensing, capacitive sensing and inductive and magnetic field sensing, we decided to choose a promising technology based on the outlined selection criteria. At first glance, time delay sensing when applied to our application was unsound, resulting in the first technology we

eliminated. Originally, we thought to either use ultrasound transponders or radio frequency (RF) receivers as a method to measure a time delay between sending and receiving a signal to and from the pea-sized object. Measuring time delay using RF successfully works in systems such as GPS and loran. These systems can easily measure a delay in time since transmitters send the signal over great distances. In the case of GPS, satellites send the signal from space and in the case of loran, shore mounted radio beacons send signals to offshore ships. Ideally, we would have liked to use the same principles, but in a much shorter range, that of six liters as specified in the prototype requirements. However, we determined that measuring the time delay of an RF signal in a six-liter container would have been impossible for our project since the delay is almost negligible. The extremely close proximity of the pea-sized object to the RF transmitters removed any possibility of using an RF based system.

Had we still desired a similar time delay technique, the only other option would have been to use ultrasound transceivers. As previously discussed, AT&T laboratories in conjunction with the University of Cambridge successfully developed an ultrasound tracking system. Their system can track a person in three-dimensional space to an accuracy of three centimeters. Although this would have been an ideal solution, their operating area is 10,000 square feet using 720 receivers (Ward, et al., 2001). As with RF, if we were to scale this system for use in a six-liter container, problems would have arisen in measuring time delay due to the close proximity of the receivers. Additionally, if the container size were not a problem, adapting the AT&T system still would not have been possible. This is because their target object contains a power source and active elements sending the ultrasound signal. For our project, the goal is to track a passive element without power. Therefore, we eliminated ultrasound and any techniques using time delay measurements.

Our remaining options were to either use capacitive or inductive sensing. One problem we discovered with capacitive sensing is its susceptibility to interference, making it vulnerable to dirt, oils or any other substance on the sensor, resulting in inaccurate measurements (Welsby, 2003). However, the main problem arises when anything else but the desired target enters the sensitive field. In the application of a 3D mouse, a user will manipulate the pea-sized object with their hand. Therefore, the person's hand will be in the capacitive field resulting in inaccurate position measurements. As a result, we eliminated any type of capacitive based sensors. Our lingering option was to use a sensing technique based on electromagnetism and inductors.

The first option we eliminated involved detecting a disturbance, caused by the pea-sized object, in the earth's magnetic field from two vertically offset magnetic sensors. These sensors would be located on the top of the container in a vertically aligned position. The proposed task was to measure the magnetic field inside the box where the target object is located and compare it to the earth's magnetic field measured between the two sensors where no object is located. Since every metallic material causes a disturbance in a magnetic field, it is possible to detect its presence by measuring the disturbance in the earth's magnetic field. Simply subtracting the measured field from the disturbed field would have allowed us to detect the target object. However, after researching this idea, we concluded that the change in the earth's magnetic field would be very small, and difficult to measure. Therefore, we excluded this option, remaining with the possibilities of placing coils on each side of the container, using magnetic field sensors or a grid of inductors.

3.0.2 Remaining Viable Solutions

The first remaining possibility included placing a coil on each side of the six-liter vessel to detect an object configured as an inductor capacitor (LC) oscillator. One coil at a time transmits an oscillation that charges a coupled inductor capacitor circuit. The LC circuit then transmits a signal back to the coil and the system calculates its proximity as a function of the signal strength. This proven technique implements the concepts of the swept frequency tag reader used by the Massachusetts Institute of Technology Media Lab as outlined earlier (Hsiao, Paradiso, 2000).

The second remaining possibility was the use of magnetic sensors, placing one sensor on three orthogonal planes of the box. These magnetic sensors are integrated circuits developed by companies such as Honeywell and Sentron. They are able to detect magnet field strength from a magnet or an LC circuit. Most of the magnetic sensors only have the ability to detect one axis of the magnetic field while others can detect two and three axes. One limitation in using multiple sensors is their effectiveness as a set is in part a function of physical alignment.

The final option was to form a grid of coils on a single or on multiple surfaces of the container. In this technique, the system multiplexes each coil to read the received magnetic field strength produced by the LC target circuit. The result is the coordinate of the coil with the highest field strength indicating the LC circuit is located at that point.

The system can use the same coil or a separate charging coil to charge the LC circuit. From the output, a computer program will plot the changing position of the target object. One similar application is the Wacom Artpad whereby the movement of a pen tip containing an LC circuit is tracked over a two dimensional surface (Wacom Components, 2002).

All of the three final solutions filled the outlined prototype criteria relatively well. However, we had to pick one method for the development of our three dimensional mouse. As a result, we discussed if there was a way to rule out two of the concepts without further research or experimentation. Our team discussions led us to the conclusion that we did not have sufficient working knowledge to rule out any of the possibilities. Therefore, we began focused hardware experiments and research to gain more insight into the benefits and limitations of each of the three technologies in the hope of choosing one.

3.1 **Hardware Experimentation**

As the theoretical research progressed, we came across key hardware systems that we felt were extremely important to understand in order to make informed design decisions as to which technology to choose. Since we lacked a working understanding of these major components, we used further research along with experimentation to bridge this gap and increase our chances of choosing the most viable technology for the prototype design.

3.1.1 LC Circuit Detection Using Transmit/Receive Coils

The first hardware system we investigated to increase our working knowledge was related to detecting the presence of an LC circuit, or even simpler, the presence of a ferrous object using a transmitting / search coil. All of the three remaining solutions except magnetic field sensors are designed to detect the presence and movement of an LC circuit using inductors as the detection mechanism. Magnetic field sensors instead use a magnetically sensitive Wheatstone bridge.

We first combined an inductor and a capacitor to form an LC circuit with a resonant frequency in the range of a few hundred kiloHertz. This LC circuit formed the target object whose position we aimed to track with a larger coil. The larger coil served

two purposes, to energize magnetically the LC circuit and to pick up any return signal when the transmitting signal was set to the resonant frequency of the LC coil. We formed the coil by tightly wrapping insulated wire around a cardboard tube section forty-seven times, covering an area of 11mm. This provided us with an inductor whose diameter was approximately 5cm resulting in an approximate inductance of 163µH. Although we would have used a larger coil for the actual design, this coil proved sufficient for testing.

In the first test phase, we transmitted the resonant frequency of the LC circuit through the larger hand wound coil. Unfortunately we were not able to detect a resulting change in the transmitting coil's inductance from any fields resonated by the LC oscillator. We expected to see a change in the form of spiking at the peaks of the sinusoid from the transmitting signal. Because we could not detect a change, we devised a method to remove the original signal from any desired feedback produced by the LC circuit, thus providing a clearer view of the isolated response.

3.1.2 Inductive Wheatstone Bridge Development

Our solution was to use a Wheatstone bridge. Although our Wheatstone bridge contained an inductive coil balanced against other elements, the concept is best explained with a purely resistive bridge. The resistive Wheatstone bridge contains two pairs of closely matched resistors. Although the resistors in each pair must match, the pairs themselves do not have to match. A sample schematic for a resistive Wheatstone bridge appears in Figure 7 below.

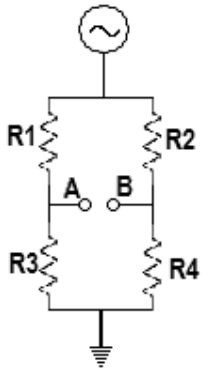


Figure 7 – Resistive Wheatstone Bridge

In Figure 7, resistors R1 and R2 match as well as resistors R3 and R4. As a result, the voltage from point A to B is zero. However, if there is a change in resistance, even to

one resistor, the bridge becomes unbalanced and a voltage drop forms. The key in the design of a Wheatstone bridge such as the one in Figure 7 is that the voltages at points A and B with respect to ground must be equal. Since finding a perfect resistor is impossible, the bridge must have a variable element such as a potentiometer to replace one of the resistors. By varying the resistance, the bridge is balanced when the voltage drop from A to B is zero. One method used to balance a bridge, regardless of the type, suggested by Phywe Systems, is to place a speaker between points A and B, while applying a frequency within human hearing range across the bridge (Phywe Systems, 2003, p. 2). When the bridge is balanced, the speaker no longer emits a tone.

Identifying a change in the transmitting/receiving inductor is necessary for sensing the position of an LC circuit. It is possible to use a similar Wheatstone bridge to the one in Figure 7 to detect the change of a transmitting/receiving inductor by using the inductor as a replacement for one of the resistors and then adjusting the variable resistor to balance the bridge. Therefore, when the AC used to pulse the LC circuit is connected to the bridge, a reading taken across A to B is the isolated response from the presence of the LC circuit. Although the basic concept is valid, since an inductor contains a magnitude and phase required us to balance both instead of only worrying about the magnitude as in a resistive bridge (MCA, 2003).

Three modified Wheatstone bridge circuits exist that enable balancing both the magnitude and phase. Figure 8 presents schematics for these three bridges under their respective headings (Integrated Publishing, 2003), (MCA, 2003). These three, the Hay bridge, Maxwell's bridge and Maxwell's inductive bridge, all are able to detect the change in an inductor forming one branch of the circuit.

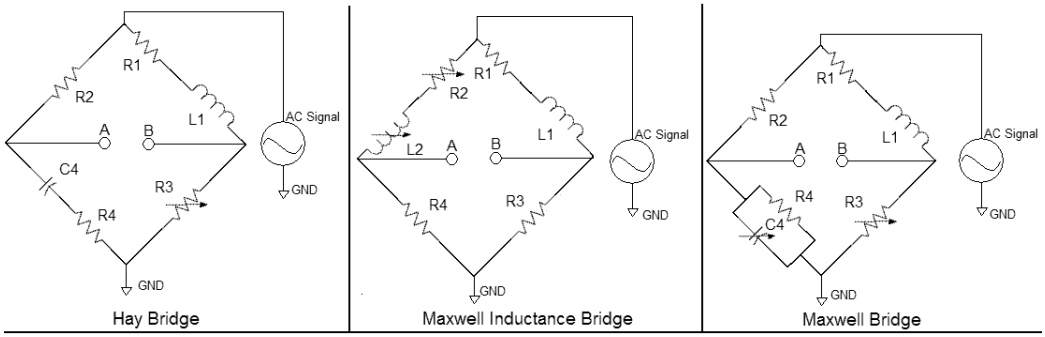


Figure 8 – Inductive Wheatstone Bridge Implementations (Integrated Publishing, 2003)

The first bridge depicted in Figure 8 is the Hay bridge, which is based on a standard resistive Wheatstone bridge. The changes permit detecting a variation in the inductance of L1 with a resistance of R1. Also added is capacitor C4 in parallel with resistor R4. The purpose of adjusting the capacitor is to enable balancing both the magnitude and phase of the bridge when connected to an AC signal. The resistor R3 is a variable resistor, which as previously described, allows further balancing of the circuit. Resistor R2, derived from the original resistive Wheatstone bridge, completes the circuit (Integrated Publishing, 2003).

To balance the Hay bridge or any inductive bridge from Figure 8 connected to an AC signal, we had to satisfy both the magnitude and phase equations. Equation 1 equates the magnitude of an inductive bridge by combining the impedance of the components on each branch and then equating the product of opposite branches.

$$Z_1 Z_4 = Z_2 Z_3,$$

Equation 1

Equation 2 on the other hand adds the combined phase of each side of the branch and then equates the sum of opposite branches (MCA, 2003).

$$q_1 + q_4 = q_2 + q_3,$$

Equation 2

We define a branch of a Wheatstone bridge seen in Figure 8 based on the subscript of the components listed. For example, branch 4 of the Hay bridge contains components R4 and C4. Branch 3 of the Maxwell bridge contains the variable resistor R3. We also consider branch 2 to be opposite branch 3 and branch 1 to be opposite branch 4 for all of the bridges.

The basis for these equations comes from the laws of complex numbers and multiplying phasors. When multiplying phasors, the magnitudes are multiplied and the angles added. Applying Equation 1 and Equation 2 to the Hay bridge resulted in a balanced expression for the bridge. This expression is

$$(R_1 + j\omega L_1)\ (R_4 - j/\omega C_4) = R_2 R_3,$$

Equation 3

Since the equation utilizes rectangular notation instead of polar coordinates we only used one equation to balance both the magnitude and phase components. After

multiplying out the equation, the resulting individual expressions for the real and imaginary parts are

$$R_1=\omega^2R_2R_3R_4C_4^2/(1+\omega^2R_4^2C_4^2) \text{ and } L_1=R_2R_3C_4/(1+\omega^2R_4^2C_4^2) \text{ (MCA, 2003),}$$

Equation 4

This shows that the equations are dependent on the frequency applied to the bridge. As a result, we did not use the Hay bridge because we wanted a bridge that we could balance regardless of the applied frequency. We also eliminated using Maxwell inductance bridge because of the difficulty in acquiring a variable inductor with a range large enough to cause a substantial change in the circuit.

We therefore chose to use the remaining option, the Maxwell bridge from Figure 8. When we applied Equation 1 and Equation 2 to the Maxwell Bridge, the resulting balanced equation in rectangular form was

$$(R_1+j\omega L_1)\,(R_4/(1+j\omega C_4R_4)) = R_2R_3,$$

Equation 5

From this equation, we equated the real and imaginary terms resulting in the two expressions to balance the bridge. These expressions are:

$$R_1=R_2R_3/R_4,$$

Equation 6

$$L_1=R_2R_3C_4,$$

Equation 7

To balance the Maxwell bridge, we had to satisfy the two previous equations by adjusting capacitor C4 and resistor R4 until the voltage drop from A to B became as close to zero as possible. We chose the component values for the Maxwell bridge in Figure 8 as follows. The handmade inductor, L1 was approximately 163µH with an internal resistance, R1, of 4.54? . We chose resistor R2 to be 8k? in order to provide a balance against the capacitor's value in microFarads. We chose R4 to be 820k? in order to offset the kiloOhm range of capacitor R2. For the variable resistor, R3, we chose a potentiometer of zero to 10k? in order to provide a decent range for balancing the bridge. After some experimentation, we found that setting the variable capacitor, C4 to 10nF was desirable. However, to fine-tune the bridge we adjusted the variable resistor, R3, until the output across A, B was approximately 10mV peak to peak with R3 at approximately 2? . This effectively balanced the bridge since the input signal's voltage was 12V. The next

major task became detecting a change we placed a magnet or an LC circuit in the large search/transmitting inductor, L1 from Figure 8.

Moving a magnet or a piece of ferrous metal in the inductor inducted a small voltage across the coil. We were able to view the result on an oscilloscope as a change in the DC offset from the output signal across A, B. The induced voltage was the result of moving a magnetic object inside the coil similar to the way in which an electricity generator works. When we placed an inductor inside the larger inductor connected to the Wheatstone bridge, the result was not a change in the DC offset but instead a change in the phase of the output signal. Unlike the change in the DC offset, the phase change was independent of motion, but instead dependent on the presence of the inductor. When we held the inductor in range of the coil, we viewed a different output phase compared to its absence.

Regardless of whether we monitored a change in the output signal by placing a magnetic object or an inductor in the coil, the change was too small for a desirable reading since it was approximately 10mV. Therefore, we amplified the signal in order to acquire an attractive reading. Even if we were to use the magnetic sensors, we still would have to attach an amplifier to the output. As a result, we wanted to use an amplifier that would work with both systems, inductive search coils and with magnetic sensors. Our solution was to use an instrumentation amplifier.

3.1.3 Instrumentation Amplifier

Regardless of the sensing system we chose for developing our three-dimensional mouse prototype, the instrumentation amplifier proved to be a significant component in each. The concept of the instrumentation amplifier is very similar to a standard operational amplifier (op-amp). One can actually be built from four op-amps and several passive components. Similar to the standard operational amplifier, it has high input impedance, selectable gain and a well-buffered output stage (Analog Devices, 2000). However, instead of amplifying a signal in reference to ground, the instrumentation amplifier has two inputs and amplifies the difference between the two. This is exceptionally useful in applications where only the difference between the two inputs is desired, while the common signal is rejected.

One example is the transmission of a differential signal over two cables. In this case, every time the signal on one wire is brought high, the other is brought low and vice

versa. If the wires are tightly wound together and passed through an electric field, for instance near a power line, both of the cables will acquire the same interference. By attaching these two wires to the inputs of an instrumentation amplifier, the differential signal is amplified which eliminates the common noise, allowing only the intended signal to pass.

Another example is the Wheatstone bridge discussed in Section 3.1.2. As we mentioned in reference to Figure 7 and Figure 8, when a Wheatstone bridge becomes unbalanced because of a change to one branch, the disturbance is detectable as the difference across A, B. However, when a change does occur, the response from A to B is considerably small. Therefore, we used the instrumentation amplifier to subtract the common signal and to amplify the remaining difference producing a usable result.

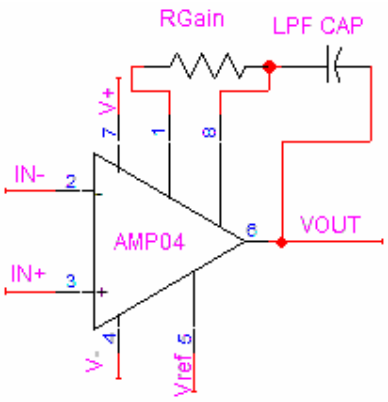


Figure 9 – Circuit Diagram of AMP04 (Analog Devices, 2000, p. 8)

Figure 9 above presents a circuit diagram of the AMP04 instrumentation amplifier. A noteworthy feature of this particular amplifier is the ability to set its gain via a single resistor. The gain can be set anywhere from one to one thousand (Analog Devices, 2000, p. 1). Equation 8 is used to calculate the gain of the amplifier based on the resistor, RGain, from Figure 9.

$$Gain = 100\,k\Omega / RGain$$ (Analog Devices, 2000, p. 7),

Equation 8

Another unique feature of the AMP04 instrumentation amplifier is its ability to act also as a low pass filter. The cutoff frequency of the low pass filter is easily set by changing the capacitor, LPF CAP, from Figure 9 based on Equation 9.

$$f_{LP} = \frac{1}{2p(100k\Omega)C}$$ (Analog Devices, 2000, p. 8),

Equation 9

Although we were able to acquire a great deal of information about the instrumentation amplifier from its datasheet, we wanted to learn more through physical experimentation before applying it to the magnetic sensor. As such, we formed a resistive Wheatstone bridge for our test platform before we expanded to using the magnetic sensor. We were able to power the AMP04 using either a single positive five-volt supply or a dual supply up to +/- 15 Volts as the datasheet specified. Although in our final design we used a single supply, for our initial testing purposes, we used a dual supply, eliminating the need for a reference voltage.

We constructed a resistive Wheatstone bridge using a 100k? resistor and a 22k? potentiometers in each leg. We then trimmed the resistors to ensure a negligible difference between the outputs. This resulted in the initial setup of the balanced Wheatstone bridge, which we then connected to the instrumentation amplifier. After some small adjustments, we were able to stabilize the amplifier's output voltage near zero demonstrating a successfully balanced bridge.

For comparison purposes, we first set the gain of the instrumentation amplifier to one and viewed the output on an oscilloscope. When we adjusted one potentiometer at a time, the output voltage moved in both a positive and negative direction based on whether we increased or decreased the resistance. Since this test was successful, we increased the gain of the amplifier to ten and repeated the same procedure.

With the amplifier's gain set to ten, we had to adjust more precisely the potentiometers to cause the output to remain at zero. When we made a small change to one potentiometer, we saw a larger response on the oscilloscope than when we made the same change with a gain of one. Continuing to increase the gain, for our last test experiment, we chose a gain over 160. Again, we had to fine tune the potentiometers for the output to stay as close to zero as possible. By merely touching one of the resistors causing its temperature to rise, and therefore resistance, we were able to view a change in the output. We determined this was not an error because it was very repeatable. By alternating which resistor we touched, we were able to swing the voltage in both a positive and negative direction.

Having successfully applied the instrumentation amplifier to a resistive Wheatstone bridge, we then connected it to the magnetic sensor. As described in Section

3.0.2, using a magnetic sensor was one of our remaining technologies for detecting the position of a magnet or LC circuit. Therefore, it was important for us to implement a few preliminary trials to gain a better understanding of the device. Even before we engaged in developing a testing system with the magnetic sensor, we acquired background information on the sensors to familiarize ourselves further with their operation.

3.1.4 Magnetoresistive Sensor Testing

A magnetoresistive sensor is an electrical device that acts in a similar fashion to a Hall Effect sensor. Both devices detect the presence of a magnetic field in a single direction and have a varying output based on the strength of the magnetic field. Hall Effect sensors work on a principle described by Lorentz Force, whereby the passage of current through a wire placed in a magnetic field causes the deflection of electrons to one side. This change in electron concentration is measurable across the wire (Chabay, Sherwood, 1995, p. 487). On the other hand, magnetoresistive sensors consist of a thin film of nickel-iron placed on a silicon wafer. These sensors act as a magnetically sensitive resistor, and when a magnetic field is applied in its presence, the result is a change in resistance. Consequently, manufacturers such as Honeywell commonly refer to these as magnetoresistive sensors. A typical magnetoresistive IC contains four sensors in a Wheatstone bridge configuration. When a voltage is connected, the result is a differential output based on an applied magnetic field (Honeywell, 2000, p. 6).

Simply examining the construction of these sensors does not reveal their full power. Instead, their sensitivity ratings only paint the picture of the potential behind magnetoresistive sensors. Most of the sensors we were able to find, and in particular those made by Honeywell, a leading manufacturer of magnetoresistive sensors, can sense fields in a range from positive to negative six Gauss. Their sensitivity stretches to sensing fields as small as 27μGauss. For comparison, earth's magnetic field has a value of 0.5 Gauss (Honeywell, 2000, p. 1). As a result, the high sensitivity common to magnetoresistive sensors enhances our range of possible implementation methods. One such method we identified was to use an oscillating inductor capacitor (LC) circuit as the target object. As a result, any magnetic response received from the LC circuit will be substantially small; however, still large enough to be detected.

Not only is the magnetoresistive sensitivity impressive, but the bandwidth is also of a considerable range. In almost every case, the bandwidth spans from zero to 5MHz,

providing us with the ability to detect a magnetic field at a varying rate (Honeywell, 2000, p. 3). An LC circuit oscillating at a particular frequency produces one such magnetic field. Therefore, we can differentiate between multiple LC circuits, each with a different resonant frequency. Finally, the wide bandwidth removes the high frequency bottleneck resulting from real time object tracking in our application.

Numerous applications have evolved because of these substantial specifications and a very fast response rate to magnetic fields. These applications include encoder wheels, electronic compasses, sensing current flow in wires, and detecting cars, trains and other ferrous objects that affect the earth's magnetic field (Caruso, & Withanawasam, 1999, p. 3). However, applications for magnetoresistive sensors seem almost endless.

Even with the large variety applications, mapping the position of an object by measuring its magnetic field intensity has not been accomplished using magnetic sensors. Most Hall Effect sensors have a much lower sensitivity to magnetic fields, and are usually used simply for indicating the presence or absence of an adjacent magnet. As such, these will not be desirable for detecting magnetic field intensity as required by our target application. Therefore, we focused our attention on magnetoresistive sensors because their output is proportional to magnetic field strength as stated on most magnetoresistive sensor datasheets.

To gain a better understanding of the functionality and capabilities of these sensors, we ordered a single sensor and devised an experiment to enhance our knowledge further. After finding several manufacturers of magnetoresistive sensors including Phillips, Sentron, Honeywell and NVE, we purchased a Honeywell HMC1022 two-axis sensor. Our main reason for choosing a Honeywell sensor was their large selection of sensors with different numbers of axes. However, almost all of Honeywell's sensors operate identically along with a large collection of application notes demonstrating their use. Since we only purchased the sensor for experimentation, we felt that a two-axis sensor was sufficient. However, the difference between a one, two, and three axis sensor is that the two and three axis sensors respectively package two and three one-axis sensors into a single integrated circuit. Multiple axis sensors permit detecting the magnetic field intensity at the sensor in three dimensions, X, Y, and Z. Therefore, to acquire the best results for a three-dimensional mouse, we would have to use a three-axis sensor or a combination of orthogonal mounted one and two axis sensors.

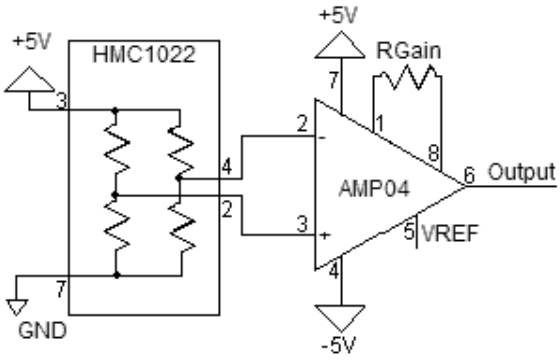


Figure 10 – Initial Magnetoresistive Sensor Test Circuit (Honeywell, 2000, p. 12), (Analog Devices, 2000, p. 8)

We designed our experiment with the HMC1022 magnetoresistive to gain an understanding of how the sensor will react to an adjacent magnetic field produced by a magnet. The magnetic sensor has two modes of operation. In the first mode, the sensor is only able to detect strong fields subsequently causing a small output change. The second mode of operation adjusts the sensitivity by constantly setting and resetting the sensor with a current pulse, permitting the detection of fields in the microGauss range. To accomplish this we would have had to implement a good deal of surrounding circuitry. As this was only a preliminary test, we felt that developing this circuitry was only worthwhile at a later state as we believed operating in the less sensitive mode would provide sufficient results. Therefore, we constructed a test circuit for low sensitivity operation; the schematic can be seen above in Figure 10.

To test the sensor, we connected positive five volts to the magnetoresistive bridge and +/- 5 Volts to the instrumentation amplifier. We then connected the output of the instrumentation amplifier to an oscilloscope for viewing. As explained in Section 3.1.3, the instrumentation amplifier is required since the magnetoresistive sensor's output is differential. Additionally it is important to note that although the HMC1022 contains both an X and a Y sensor, for our experiment we only connected the X sensor. The only difference is that the X sensor is sensitive in a horizontal direction relative to the integrated circuit while the Y sensor is sensitive in a vertical direction. Otherwise, the sensors are identical including output and input pins.

Equally important as the hardware configuration in our experiment was the magnetic object that we will detect using the sensor. Ideally, we would have liked to use a good quality magnet; however, a lack of available parts forced us to build a small

electromagnet from a screw and magnetic wire. The electromagnet we developed was weak and had a small field, but we were still able to detect its presence with the sensor. We saw an increase in voltage when we brought the electromagnet closer to sensor and a decrease in voltage when we moved it away. While monitoring the output of the magnetoresistive sensor on an oscilloscope, we were able to view the magnet's response up to approximately 5 to 7 centimeters away from the sensor with an amplification gain of 160.

Our first experiment with the HMC1022 was a success. Based on these results we believe detecting a magnet or an LC oscillator producing a magnetic field for a three-dimensional mouse is possible using magnetoresistive sensors such as the HMC1022. We also found the accuracy of the sensor to be very good as the shaking of one's hand while holding the electromagnet was detectable. From our promising results, we determined that it would be possible to use magnetoresistive sensors as the basis for our three-dimensional mouse permitting very fine accuracy and a range of at least several centimeters.

3.2 Selection of a Feasible Solution

Prior to the start of the hardware experiments, our research provided us with enough information to narrow our initial list of sensing technologies down to three viable options. These options were using inductive search coils, a grid of inductors on a single plane and magnetic field sensors as described in Sections 3.0.1 and 3.0.2. However, research without experimentation did not permit us to narrow the list to one remaining solution. After carrying out the experiments described in Section 3.1 and analyzing our research, we were able to settle on one sensing technology for the basis of our three-dimensional mouse.

We believe that using magnetoresistive sensors will best fulfill the prototype requirements set earlier. As we demonstrated in our hardware experiments, these sensors meet all of the prototype requirements including sensing magnetic fields instead of utilizing optics, which allows them not to be influenced by the presence of a user's hand. We also discovered that the sensor could detect the slight movement caused by a person's hand as they did their best to hold the electromagnet or target steady. This demonstrated that we could use the sensors in high sensitivity applications. The sensors are also relatively simple to connect and use, alleviating a potential time loss spent in extensive

debugging. Additionally we were not able to find anyone who applied magnetoresistive sensors to small-scale position tracking, fulfilling AMT's request for a novel approach to the three-dimensional mouse.

The other technologies we studied in detail showed great potential, however, each fell short of fulfilling one of our prototype requirements. A perfect technological solution would have been using inductive search coils combined with a swept frequency tag reader as designed and built in the Media Laboratory at MIT. However, AMT asked us to use a novel approach. We also discovered a secondary concern in our experimentation with inductors and detecting signals, as discussed in Sections 3.1.1 and 3.1.2. We realized that using inductors and a Wheatstone bridge to pick up a magnetic field from an LC circuit was much more complicated than it first appeared. The theory behind creating a well-balanced inductive bridge is simple, but difficult in practice.

One final method remained that we were not able to eliminate from our preliminary research. This approach was to use an array of coils to track an LC oscillator in a similar fashion as the ArtPad made by Wacom Technology. Wacom's product encloses an LC oscillator in an electronic pen used to control the computer cursor. This method fulfills most of our prototype requirements; however, after we outlined the basic system architecture we found that it required a significant number of components in comparison to the other approaches. The system's hardware complexity increases with desired precision and therefore resolution since a coil is required for every pixel on the screen. The other option would have been to use fewer coils and instead develop a complex software-tracking algorithm. Again, we also decided against this technology, as it is not a novel idea. Wacom Technology dealt with this style of motion tracking for years and developed many products utilizing the technology (Wacom Components, 2002). However, even with their extensive development they have not applied an array of coils to three-dimensional position tracking.

Through our extensive research and testing, not only were we able to settle on one sensing technology, but also we were able to build a strong argument for applying magnetic sensors towards our three-dimensional mouse prototype. From our experiment with the Honeywell HMC1022 magnetoresistive sensor we believe that further expansion will meet our prototype requirements with a novel solution. As such, we looked back to our initial experiments with the magnetic sensor to determine how to expand the system into a three- dimensional mouse.

Chapter 4– Magnetoresistive Sensor Tracking System Development

Deciding on one technology as the basis for our three-dimensional mouse was a challenging task that evolved from our research and experimentation. Our research brought us knowledge on a wide range of position tracking systems. We discovered that radio frequency based systems were not feasible due to our small operating space requirement. Exploration into alternative methods led us to the conclusion that magnetic and inductive based systems were most suitable for our application. Our experiments in electromagnetism further provided sufficient evidence to eliminate unfeasible methods discovered during our research, leading us on the road to a magnetoresistive sensor tracking system.

Looking back, we concluded that the experiments with Honeywell's HMC1022 magnetic sensor, described in Section 3.1.4, presented successful results in the form of an analog output signal correlated to the proximity of a magnet in close range to the sensor. According to the sensor datasheet, setting and resetting the magnetic sensor increases the sensitivity to a range beyond what we encountered in our experiments. Although this increased sensitivity was not essential for our experiment, it is crucial for the three dimensional mouse as the operating range is extended.

Another major discrepancy was that our initial experiments used only one sensor, while the actual system will have three planes of sensors, each requiring three orthogonally placed sensors. This will allow for greater accuracy by detecting the magnet or coil's response in all three dimensions on each of the operating vessel's planes.

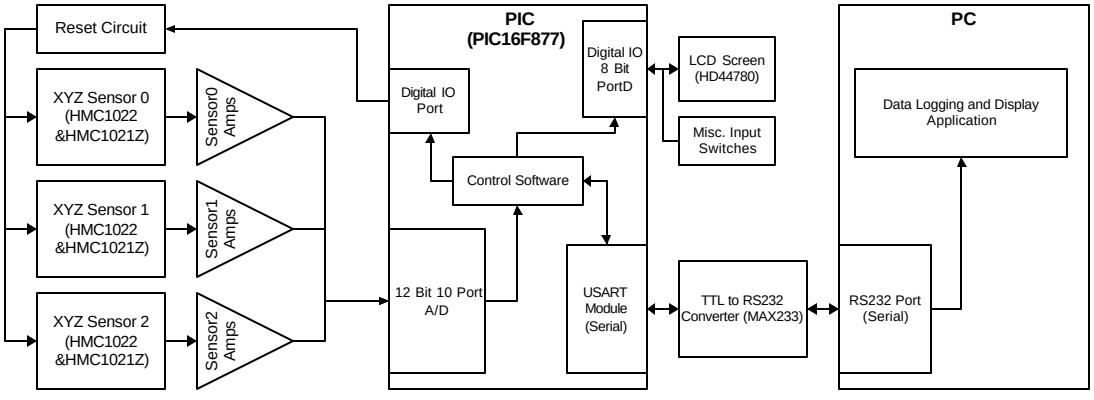


Figure 11 – Three-Dimensional Mouse System Diagram

Figure 11 depicts a block diagram providing an overview of both the hardware and software systems for the three dimensional mouse. The first segment in the system is three sets of magnetic sensors, XYZ Sensor 0, XYZ Sensor 1, and XYZ Sensor 2. Each set represents a board containing three magnetic sensors individually sensitive in the X, Y, and Z direction for a grand total of nine. Since the magnetoresistive sensors have a differential output, we connected an instrumentation amplifier to each sensor, giving us a single voltage output. We connected the set/reset circuitry to each sensor in the hardware system in order to increase the sensitivity range by constantly setting and resetting the sensors as described in Section 3.1.4. The hardware system leaves off at this point with a control line for the set/reset circuitry and data lines extending from each of the nine amplifiers, all connected to the PIC microcontroller.

The sensor board containing the magnetic sensors at the heart of our system is not of much use without a control mechanism. Using a PIC16F877 microcontroller manufactured by Microchip, allows us to interface easily with the magnetic sensor circuitry. We used a microcontroller to allow us to speed up our development time by implementing a great deal of features in software. The PIC contains an analog to digital converter that we used to acquire voltage levels directly from the magnetic sensors. At the same time, there are enough other outputs on the microcontroller for controlling the set/reset circuitry. Another feature included in this microcontroller is a serial port, which is very useful for transmitting data to a computer. We included several other components surrounding the microcontroller, such as an LCD module and a series of dip switches to aid in debugging.

Once the PIC collects data it communicates over the serial port on a PC with our own data collection software. First, some preprocessing of the data is necessary since each piece is transmitted inside a packet carrying an identification and a checksum. From there the program logs the incoming data to a comma delimited text file, which can be imported into a spreadsheet application. Once this is finished, the application again processes the information and then creates graphs of the incoming data displaying it on the monitor in real time.

4.0 Magnetoresistive Sensor Hardware Development

It is apparent from Figure 11 that the hardware and software systems are tightly integrated. The software utilizes the data from the magnetic sensors for plotting a real time graph and the hardware system requires the software to trigger its reset circuitry. The design is almost a closed loop relationship; however, the start to the system lies on the hardware side, in particular with the magnetic sensors.

4.0.1 Magnetoresistive Sensor Circuitry

The magnetic sensors are the heart of our three dimensional mouse since we designed the hardware and software systems around them. As described in Section 3.1.4, we chose the HMC1022 magnetoresistive sensor to measure the X and Y-axis of a magnetic field from a magnet. Using one IC to measure two axes is possible since the HMC1022 contains two individual sensors offset by ninety-degrees. Additionally, we used the HMC1021Z to acquire the Z-axis, as it is vertically mountable, allowing detection in the third dimension. Aside from the packaging style and number of sensitive axes, there is no difference between the HMC1022 and the HMC1021Z. Figure 12 presents the internal circuitry for an individual sensor from the Honeywell HMC sensor line.

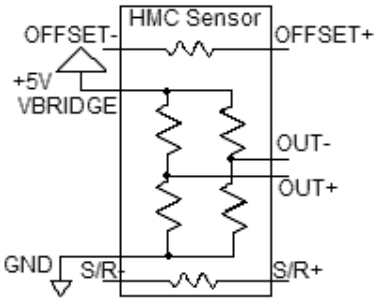


Figure 12 – HMC Magnetoresistive Sensor Diagram

The diagram in Figure 12 contains the differential output, OUT- and OUT+, as initially described in Section 3.1.3. The source voltage for the magnetic sensor, VBRIDGE, connects to the top of the magnetoresistive sensor bridge. We used the recommended 5 Volts as specified in the datasheet and connected the lower half of the bridge to ground. The remaining connections are for the offset strap and the set/reset strap, both of which are resistive elements. The purpose of the offset strap is to cancel out

any background magnetic field by driving a current from OFFSET+ to OFFSET-. The set/reset strap on the other hand is designed to resensitize the sensor into a high sensitivity state by pulsing a large current from S/R+ to S/R-.

Our development commenced by using the basic sensor diagram from Figure 12, derived from Honeywell's datasheet, to configure the X and Y-axis HMC1022 magnetoresistive sensor and the Z-axis HMC1021Z sensor (Honeywell, 2000, p. 5). The first step in configuring the sensors for our application was to develop the amplification circuitry as shown in Figure 13. The schematic represents the two Honeywell sensors with the attached circuitry and instrumentation amplifiers. We decided to use three of the same instrumentation amplifiers from our initial test experiments in Section 3.1.3, all identically configured to ensure uniformity. Each amplifier's gain is set to approximately 147 by resistors R1 through R3 with a value of 680Ω. The resistance is inversely proportional to the resulting amplifier gain as demonstrated by Equation 8. We chose the indicated amount as it was close to the suggested value from Honeywell's datasheet for the HMC1022 and HMC1021Z (Honeywell, 2000, p. 13).

The other component affecting the amplifiers is the size of capacitors C1 through C3. These capacitors determine the amplifier's low frequency cutoff, demonstrated by Equation 9. The indicated value of 1nF results in a cutoff frequency of approximately 1.59 kHz. As with the amplifier gain, our chosen capacitor was also close to the suggested value given in the Honeywell datasheet. In developing the circuit we wanted to minimize any unknowns as much as possible, which is why we initially used the datasheet's suggested values. However, in an expanded system it is possible that the amplifiers will need a smaller capacitor for an increased cutoff frequency.

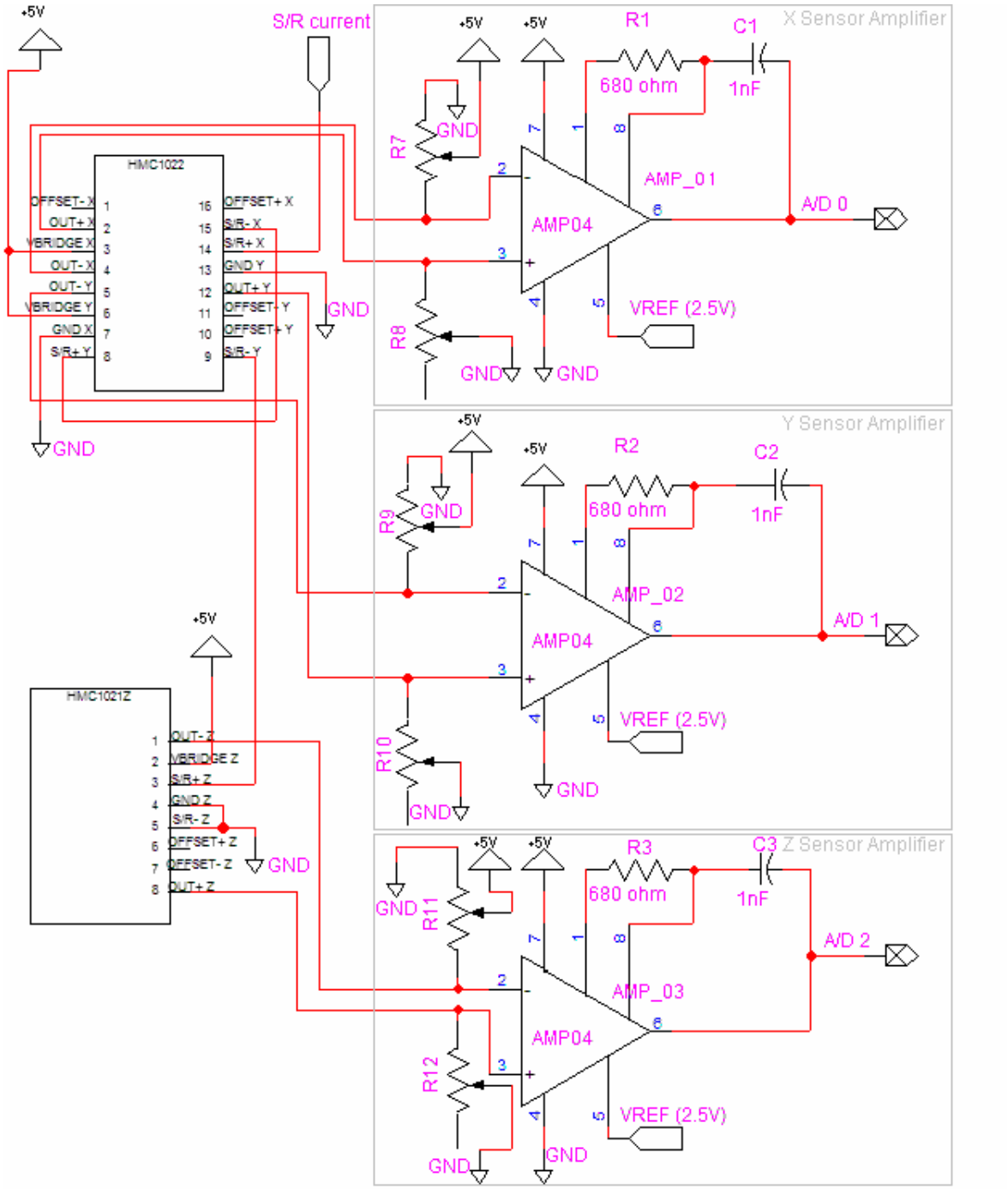


Figure 13 – Magnetoresistive Sensor and Amplifier Schematics

In addition to using a resistor to set the gain and a capacitor to set the cutoff frequency, each amplifier connects to a 2.5 Volt reference. The purpose of the 2.5 Volt reference is to allow the amplified signal to move in a positive and negative direction. This need arises because the output of the magnetic sensor can swing in either direction based upon an increasing or decreasing magnetic field. We would not need the reference

had we attached a negative Voltage instead of ground to the amplifier. However, we decided not to do so in order to avoid using a negative supply. To form the reference we used a 2.5 Volt reference IC along with a 27k? resistor for a load; the schematic appears in Figure 14. We then connected the 2.5 Volt output Voltage to pin five on each of the instrumentation amplifiers in Figure 13.

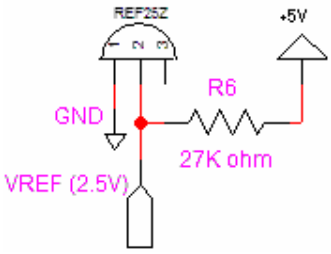


Figure 14 – 2.5 Volt Reference Voltage Schematic

The final circuitry attached to the instrumentation amplifiers consists of resistors R7 through R12 respectively connected to either 5 Volts or ground shown in Figure 13. The six resistors are potentiometers, variable from zero to 20k? . We used them to remove any DC offset in the amplified magnetoresistive sensor signals when a magnet was not present. Although there are six potentiometers attached, two per bridge, we only adjusted those connected to the negative input on the instrumentation amplifiers. The existence of such a DC offset indicates the sensors are detecting a background magnetic field or the interference of a nearby object, limiting the system's accuracy. Another method is to drive a current through the offset straps onboard the magnetoresistive sensors. However, we felt this method was more complicated and we would not need it in our prototype but is worthy of exploration for a future system.

At this point in the circuit, it is possible to pick up magnetic fields around the sensors. However, the sensor output would be extremely low as we found in our early experiments. In meeting the prototype requirements, we increased the sensitivity of the sensors to be able to detect magnetic fields inside a six-liter container. To accomplish this we ran a high current pulse through the reset straps as described in Section 4.0.2

4.0.2 Set/Reset Circuitry

Honeywell produces the magnetoresistive sensors from a nickel-alloy (Permalloy), thin film, deposited on a silicon wafer (Honeywell, 2000, p. 6). The resistance of the material changes as a magnetic field is applied to the system. Originally, when the device is manufactured, the sensors are polarized in a certain direction; however, a very strong magnetic field can weaken, or even flip the polarization of the system. When this happens, the change in resistance, proportional to magnetic field strength of the sensor, drops significantly. To restore sensitivity of the sensor, it needs to be repolarized. This is accomplished by applying a very strong, but brief, magnetic field along the sensitive axis of the sensor through a set/reset strap. By pulsing a current between 0.5 and 4 Amps through the reset strap, we can restore the sensitivity of the sensors (Honeywell, 2000, p. 3). The more often, and the stronger the current through the offset straps, the more sensitive the sensors become.

A stringent requirement specified in the sensor datasheet is that if a positive current pulse is sent to the strap, the current cannot drop below zero, and if a negative reset pulse is sent to the strap, the current cannot rise above zero. Any change in polarity upon setting or resetting the sensor will undo the polarizing effect of the pulse, desensitizing the sensor (Honeywell, 2000, p. 8). Additionally, by pulsing a positive current though the straps, the sensors will be polarized in one direction, while a negative pulse will inversely polarize the sensor. We will from now on refer to flipping of the polarity of the sensor as pulsing the set/reset strap.

Our construction of the set/reset circuit began by analyzing methods to provide an adequate current pulse across all three sensor straps. In designing this system, we looked at suggested methods and circuits from Honeywell's datasheet for the HMC1022 and 1021Z. The two main subsystems are the switching circuitry to control the current flow and the circuitry required to produce the needed voltage. The Honeywell datasheet suggested using a MAX662A, 5 to 12 Volt DC to DC power supply, to produce the reset voltage and an IRF7105 power MOSFET for controlling the current flow (Honeywell, 2000, p. 9).

Figure 15 shows a schematic diagram of the switching system that is responsible for generating a high current pulse through the magnetic sensor straps. The main component of this circuit is the IRF7105 containing an N and P channel MOSFET pair. The S/R tab connects to a microcontroller to initiate a set or reset pulse. Figure 16

presents the same schematic but with a detailed view of the two MOSFETS contained within the IRF7105 chip.

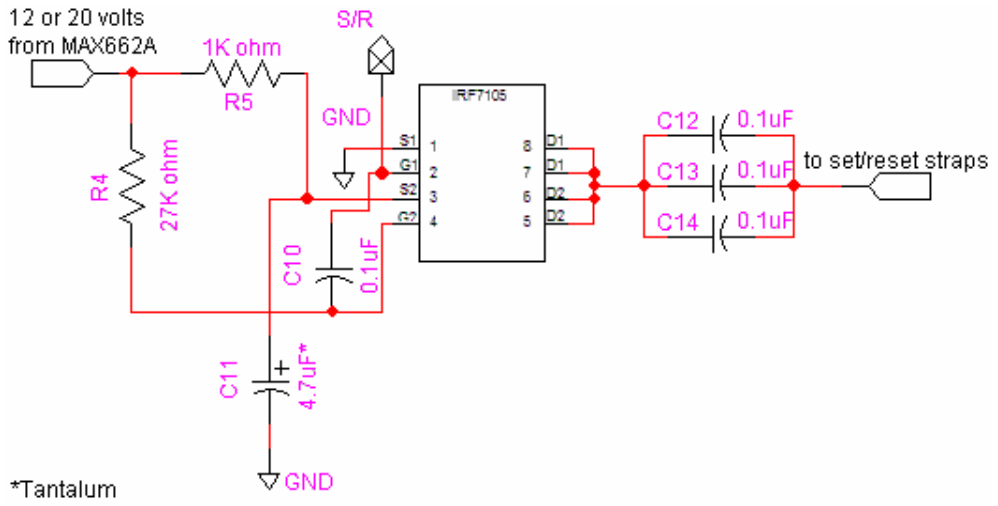


Figure 15 – IRF7105 Power MOSFET Switching Set/Reset S chematic

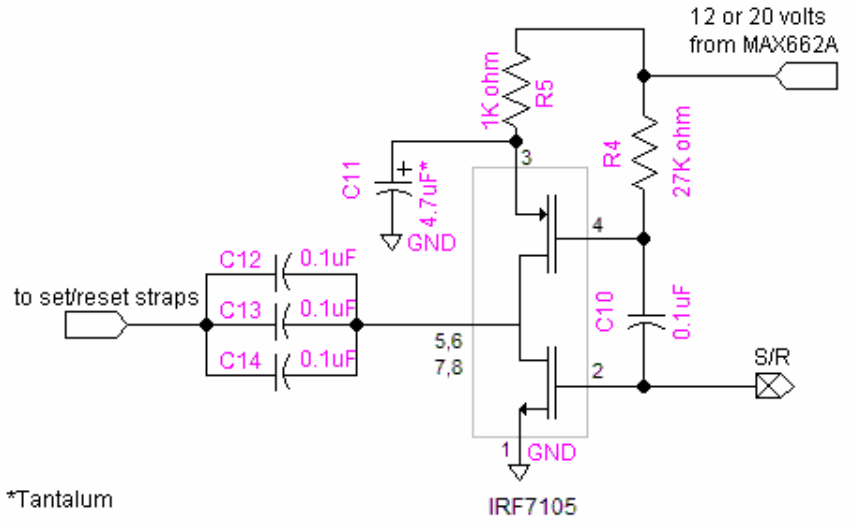


Figure 16 – Detailed View of The IRF7105 Power MOSFET and Connected Circuit (Honeywell, 2000, p. 13)

The purpose of this circuit is to create a very brief current pulse through the magnetic sensor straps. We designed the circuit in a way that ensures that DC current cannot flow though the reset straps. We accomplished this by including an RC combination in the circuit to turn the MOSFET off after a short time period.

The operation of this circuit is relatively simple; however, selecting the components required strict care. The MOSFET pair has a very low ON resistance.

Capacitor C11 must be a quick discharging, high frequency, tantalum capacitor because the MOSFET changes state during a set or reset pulse. Every time there is a change in the state of the MOSFET, the current to the set the sensors flows out of the high frequency 4.7uF tantalum capacitor and quickly charges the set of parallel 0.1uF capacitors. The output of these capacitors connects to the set/reset straps of the sensors. Since these are capacitors, only high frequency pulses can reach the magnetic sensors and a DC current can never run through them. For every high to low or low to high change on the output of the microcontroller, the set/reset circuitry produces alternating high and low pulses.

The second segment of the set/reset circuitry is the MAX662A DC to DC converter, used to deliver enough voltage to reset the magnetic sensors adequately. The output of this circuit links to the IRF7105 switching circuit at the points in Figure 15 and Figure 16 labeled 12 or 20 Volts from MAX662A. The MAX662A datasheet suggested a configuration to enable an output of 12 and 20 Volts. We adapted this configuration for the set/reset circuitry depicted in the schematic in Figure 17.

Figure 17 shows the DC to DC voltage source segment of the set/reset circuitry that we placed on each of the three sensor boards. Having a decentralized power booster on each board allows us to transmit commands from the microcontroller only to initiate a set/reset pulse. This also protects the microcontroller from high voltages.

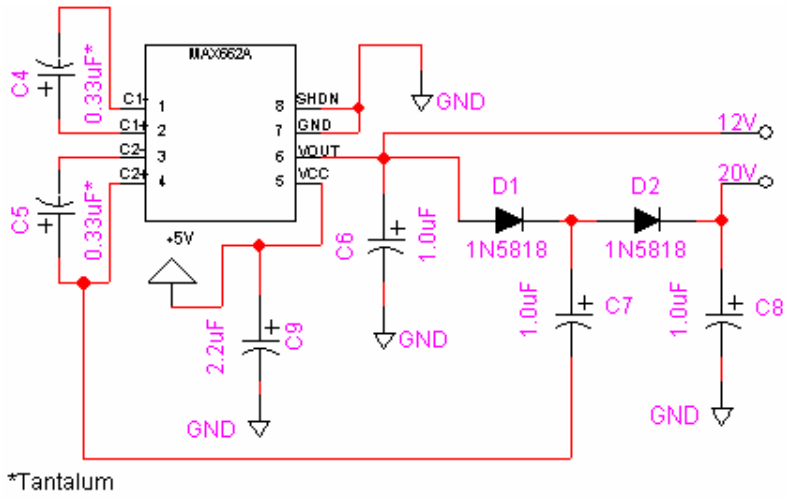


Figure 17 – MAX662A 5V DC to 12 and 20V DC Converter Schematic

The MAX662A IC is a standard DC to DC power converter using two charge pumps. The chip runs off a standard 5 Volt power supply and generates 12 Volts on the output. Adding some extra circuitry allowed us to raise the output voltage to 20 Volts. We

generated the 20 Volts by adding two diodes on the output of the circuit, D1 and D2. The first diode only allows current to flow out of the chip to be stored in a holding capacitor, C7 seen in Figure 17. The output from one of the capacitors in the charge pump system connects to the nega tive end of the same capacitor, C7. This raises the potential across the capacitor past 20 Volts. Diode D1 does not allow this voltage to flow back into the chip that is delivering 12 Volts. Instead, it flows through diode D2 and is stored in capacitor C8.

Our primary concern when using the MAX662A DC to DC power supply was whether to use the standard 12 Volt output or add circuitry to generate 20 Volts. Almost all of the sample circuits in the datasheet with one or two sensors use 12 Volts; however, our system has to reset three sensors with adequate current. Additionally, the system could either pulse them in parallel or in series. We had four wiring configurations to choose from, either using 12 Volts with the straps in parallel, 12 Volts with the straps connected in series, 20 Volts with straps in parallel, or 20 Volts with the straps connected in series.

From the HMC1022 and HMC1021Z datasheet, Honeywell specifies that the resistance of the set/reset straps is between 5.5 and 9? , but typically around 7.7? when measured between S/R+ to S/R- (Honeywell, 2000, p. 3). We also had to take into consideration the IRF7105 power MOSFET with an on resistance of 0.25? that will always be in series with the set/reset straps.

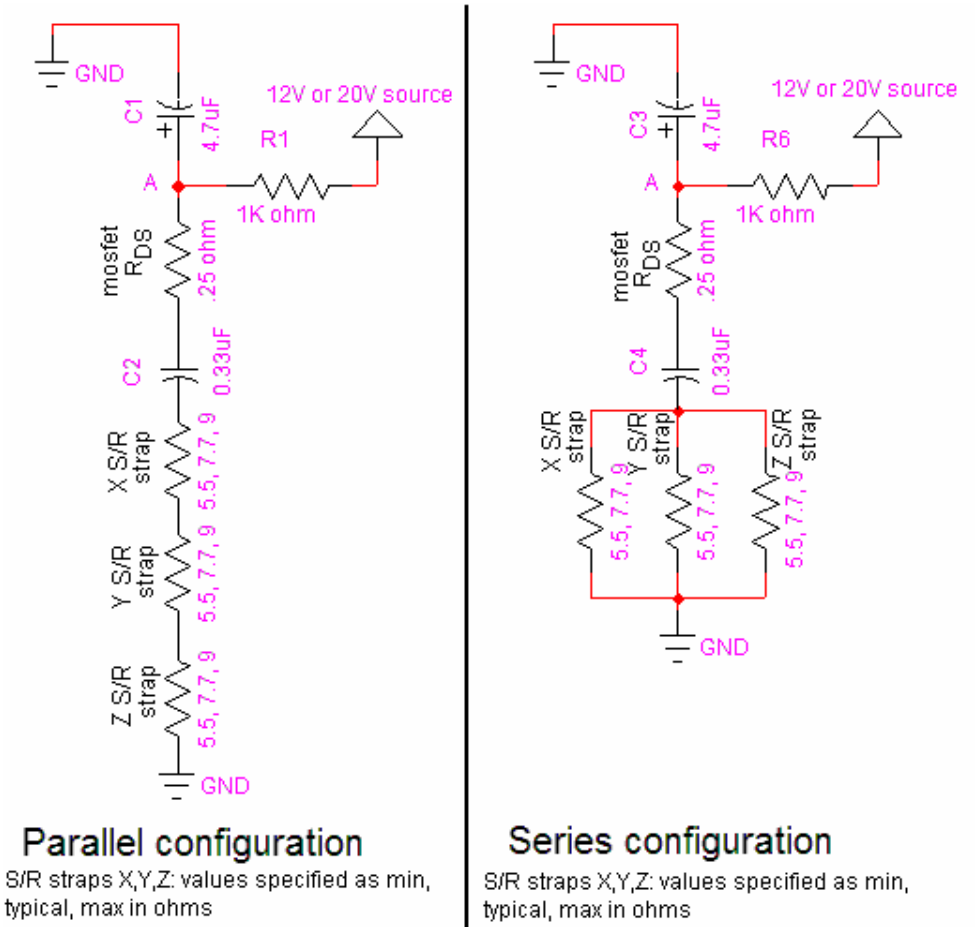


Figure 18 – Series and Parallel Set/Reset Strap Configurations

Figure 18 is a simplified portion of our set/reset schematics designed to show the flow of current through each of the three set/reset straps in either a series or parallel configuration. When the microcontroller triggers the IRF7105 MOSFET in Figure 15, the MOSFET turns on allowing the 4.7uF capacitor, C1 or C3 in Figure 18, to discharge through the straps. When the MOSFET turns off, the 12 or 20 Volt supply subsequently charges the capacitor. To determine which configuration is optimal, we calculated the current that the series and parallel setups would deliver with the results in Table 1.

Table 1 – Calculated Set/Reset Currents

		Series[2]	Parallel[2]
12 Volts	Min resistance (5.5O)[1]	0.72 amps	1.92 amps
	Typical resistance (7.7O)[1]	0.51 amps	1.42 amps
	Maximum resistance (9O)[1]	0.44 amps	1.23 amps
20 Volts	Min resistance (5.5O)[1]	1.19 amps	3.2 amps
	Typical resistance (7.7O)[1]	0.86 amps	2.37 amps
	Maximum resistance (9O)[1]	0.73 amps	2.05 amps

[1]per S/R strap as specified by the HMC1022 & HMC1021Z datasheet
[2]current per strap

Table 1 makes it apparent that a parallel configuration will deliver a much larger current averaging 2 Amps compared to a series configuration averaging 0.7 Amps. Deciding to use a parallel configuration distributing 2 Amps or a series configuration distributing 0.7 Amps was a matter of whether the straps could handle the larger current and whether the set/reset circuitry could supply the power. Even though Honeywell specifies in their datasheet a maximum of 4 Amps for the set/reset strap, we were still concerned that such a large current might damage the sensors. As such, we felt that a series configuration would be safer, delivering an average of 0.5 Amps as the typical value specified on the datasheet. In later system development a parallel configuration could be tested, possibly increasing the sensitivity over a larger operating area because of the stronger current pulse.

Developing the set/reset circuitry was the last task in forming the sensor boards. Having finished the design, we constructed a printed circuit board (PCB) for mounting the components. A PCB was necessary in this design for two reasons. The first is that both sensors are surface mount components, making it close to impossible to use them in any other way. The second was that we need all the sensors mounted orthogonal to each other. If they were not, we would need more calculations to compensate for the misalignment.

4.0.3 Magnetic Sensor Board PCB Layout

Our approach to the PCB layout of the magnetic sensor board was to separate each section of the board, both physically and electrically. We wanted to be able to populate and test each segment of the board without affecting any other section. By incorporating these features in the design, the resulting board is modular with each section able to operate independently.

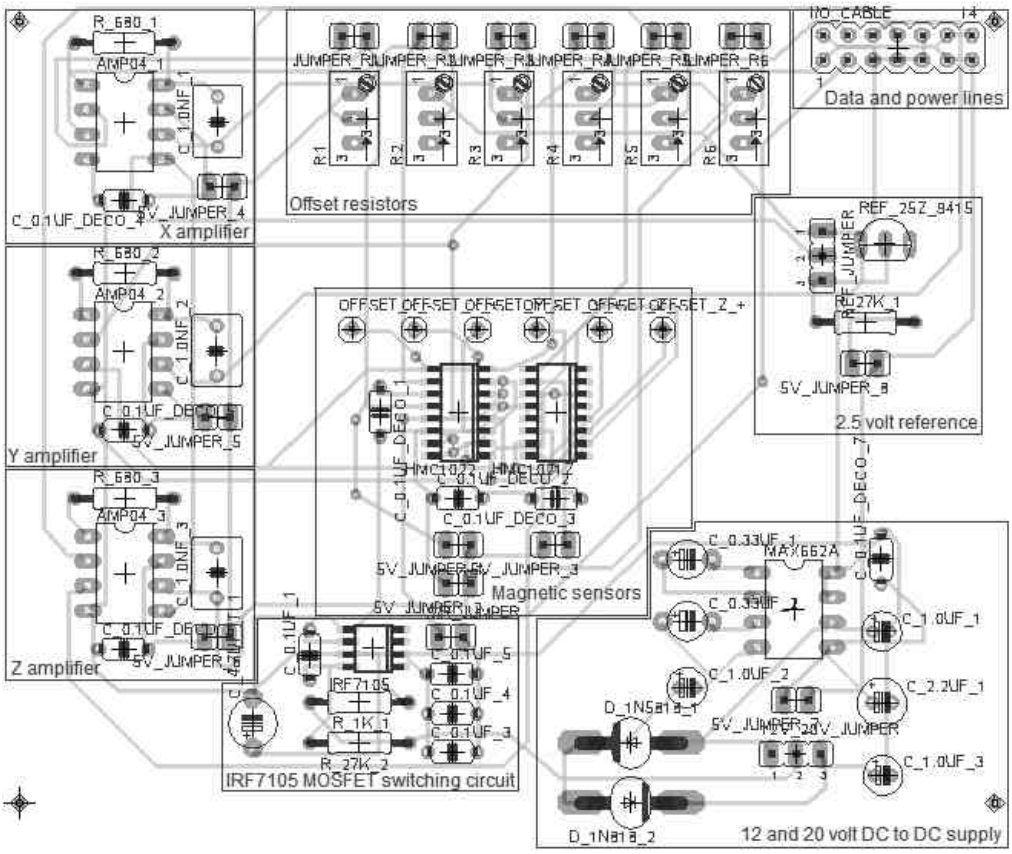


Figure 19 – PCB Layout for The Sensor Boards

The main sections of the printed circuit board in Figure 19 are the amplifiers, X, Y, and Z on the left side, the offset resistors on the top, the 2.5 Volt reference on the right, the magnetic sensors precisely located in the center, the DC to DC supply and the IRF7105 MOSFET circuit on the bottom. Each integrated circuit, including the 2.5 Volt reference IC, has a jumper attached between 5 Volts and the VCC pin. This setup permits disconnecting any device from power, simply by removing a jumper. Additionally, since the X, Y, and Z sensors receive power individually, we can independently activate and deactivate them. This feature proved to be extremely useful with initial software testing. Additionally, every IC has an adjacent 0.1uF decoupling capacitor.

We also placed jumpers between the set/reset circuitry and the set/reset straps. This allows us to view the voltage from the pulsing current on an oscilloscope before connecting the segment to the magnetic sensors. A jumper also exists between the DC to DC supply and the IRF7105 MOSFETs. This jumper permits switching between 20 and 12 Volts for the voltage sent across the set/reset straps. We also used this jumper to check

for the desired output of 20 and 12 Volts from the DC to DC supply before connecting it to the MOSFET circuit.

Another jumper used in a similar manner connects the 2.5 Volt reference to the reference pin on each of the instrumentation amplifiers. This jumper provides the option to connect either ground or 2.5 Volts to the reference pin or to fully disconnect any reference and leave the pins tied to each other. For both the reference jumper and the set/reset voltage jumper we could have connected any voltage by simply attaching the output pin to an external power supply. Although we did not need to do this, we implemented it for both debugging purposes, and in case the DC to DC supply or the reference IC failed.

Jumpers also connect each of the offset resistors to the amplifier inputs. This facilitated populating and testing the board without having to worry about the effects of these resistors. We also connected single pins to the terminals of the offset straps on the magnetic sensors. These pins are visible above the magnetic sensors on the PCB in Figure 19. Although we did not use these connections in our design, they provide for future expandability of the circuit board.

Another portion of the board worth noting is the pin out for the ribbon cable connector on the top right corner. This cable supplies the sensor board with five Volts and ground from the control board that containing the microprocessor. Additionally, the cable delivers the amplified magnetic sensor signals to the control board and retrieves the set/reset signal. Table 2 presents a bird's eye view of the pin configuration of the header.

Table 2 – Pin Configuration for Ribbon Cable Connector on Sensor Board

2	4	6	8	10	12	14
GND	X	Y	Z	GND	+5V	+5V
1	3	5	7	9	11	13
+5V	S/R	GND	GND	GND	+5V	+5V

The reason for multiple ground pins was to reduce interference by alternating between ground and the sensor signals on the ribbon cable. To keep the system relatively balanced, we used the same number of wires for power as for ground.

We populated and tested the board starting with the DC to DC supply. In populating this section, we ran into a problem with the IC socket that we used. After building an identical replica on a proto board and checking the circuit board, we realized the fault lay with the socket. After removing the socket and mounting the MAX662A

directly to the board, we had no problem and received the desired 12 and 20 Volts. From there we went on and populated the IRF7105 MOSFET and its supporting circuitry. In testing the set/reset pulsing system, we needed to connect the output across a 27? resister so that we could represent the load of the magnetic sensors. For this test, we connected a frequency generator to the input of the circuit to trigger the MOSFET allowing us to test the reset circuit at different frequencies.

Next, we populated the instrumentation amplifiers, testing each before populating the subsequent. For testing, we connected the amplifier to the magnetic sensor that we used in our initial experiments, Section 3.1.4. Each amplifier worked just as expected, both with and without the 2.5 Volt reference.

Lastly, once every part of the board was ready, we soldered the magnetic sensors in place. We soldered the HMC1022, containing the X and Y sensors to the board since it comes in a standard SOIC package. The HMC1021Z on the other hand comes in a SIP package allowing the sensor to stand upright on the board, measuring the Z component of a magnetic field.

After fully populating and testing each section of the board, we were able to test the system as a whole. By connecting a frequency generator to the input of the reset circuitry, we were able to set and reset the magnetic sensors, while watching the output of the instrumentation amplifiers on an oscilloscope. We were able to observe distinctly on an oscilloscope the movement of a magnet more than thirty centimeters away.

Successfully developing the magnetic sensor board was critical for the three-dimensional mouse. Equally as important is the subsequent control and display system utilizing the microcontroller and PC. Without these systems, it would be extremely difficult to interpret the output of the sensors, while properly resetting them. Subsequently, the microprocessor control board was the next system to take shape.

4.1 Microcontroller Hardware Development

The magnetic sensors may be the heart of our system, but without a brain to control them they are of little use. For the control system, we decided to use a programmable microcontroller. We chose the PIC16F877 microcontroller made by Microchip. We chose this microcontroller for several reasons. First, everyone in our group had some PIC programming skills, and could help with debugging the software. The second reason was that the chip we chose has an abundance of features, which we

decided, might become useful in debugging the system. There are two primary functions the chosen microcontroller must have. These are a built in analog to digital converter, and a serial port. We needed the first to collect voltage levels from the magnetic sensors. We needed the serial port to easily transmit the data to a computer. We could have chosen a smaller microcontroller, but the extra ports allowed us to add many features during debugging. One of the extra features, which we removed from the final version, was a 16-character LCD display. This was extremely useful in our initial serial communication development. Another feature was adding 8 dip switches. These allowed us to change values internally on the PIC without the need to reprogram it. For additional indicators, we added two LED's to indicate whether the PIC was off, fully booted, or busy.

There are only a few necessary pins on the PIC that must be connected in order for it to be ready for operation. These are ground and two power leads, and the pin labeled MCLR, an indicator to the PIC letting it know whether it is in programming mode or normal operation. During normal operation mode, the pin is brought to 5 Volts. The MCLR pin can also be used to reset the PIC by grounding the pin momentarily. During programming, 15 Volts is connected to the pin. The final part needed is a crystal oscillator connected between pins OSC1 and OSC2, which are also coupled to ground. A schematic of the PIC and supporting circuitry are shown in Figure 20 below.

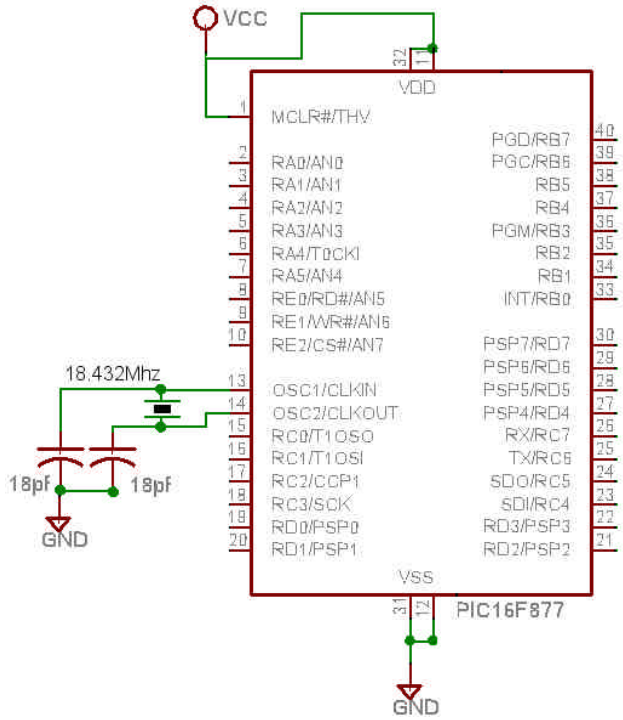


Figure 20 – Necessary PIC Connections (CADsoft, 2003)

To generate a standard serial port speed of 38.4 Kbps, we used a clock speed of 18.432 MHz. The PIC generates the serial baud rate based on a divider of the clock speed. The maximum speed at which the PIC can run is 20 MHz, so this was the closest clock speed available to generate a serial speed easily recognizable by a computer

The main function of the PIC in this system is to collect data from the magnetic sensors. For this to be possible, we implemented an analog to digital converter. Fortunately, the PIC contains an ADC that it multiplexes on eight of the pins. For our prototype, we only needed three of the pins, one for each sensor. Since the output of the instrumentation amplifiers is rather clean and partly filtered, we decided we did not need additional circuitry between them and the ADC port. The analog inputs from the magnetic sensors are connected to pins RA0, RA1 and RA2, correlating to the X, Y and Z sensor, respectively.

As mentioned earlier, to increase the sensitivity of the magnetic sensors, we needed the PIC to control setting and resetting them. For this, we decided to use one output on Port C for each board. The board that we built connects to pin 15 on the PIC, also known as RC0. Since the PIC can only source and sink 25mA per IO pin, we built a simple buffering circuit that could provide us with more current. Initially, we looked into buffers prepackaged in ICs. These however could not sink nor source much more current than the PIC. Figure 21 below shows the simple transistor circuit we built as a buffer. Since we are dealing with only digital signals, all the transistors are always either off or in saturation.

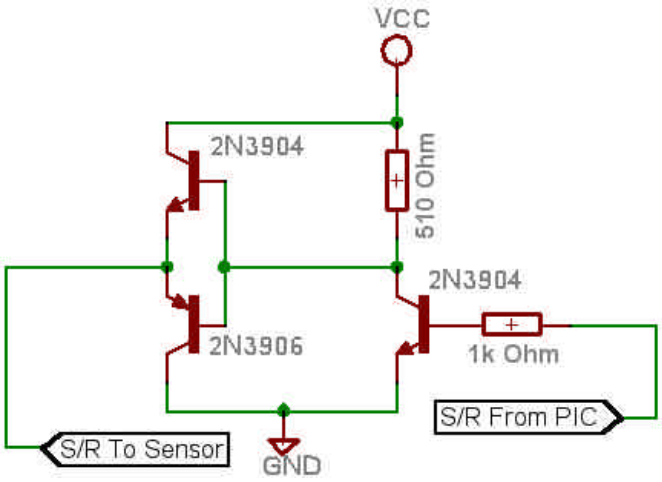


Figure 21 – Reset Circuitry Buffer

Even though the PIC has many great features, we cannot do much with the collected data unless we can somehow view it. To make this possible, the PIC must digitally transmit the collected analog data to a computer. The computer initiates communications between the two systems, so the PIC must be able to both receive and transmit signals over a serial port. For tunately, the PIC16F877 has a built in serial port, also known as a Universal Synchronous Asynchronous Receiver Transmitter module. This module, internal to the PIC, lets the hardware handle the transmission and reception of serial data from the computer. However, standard serial communications follow the RS232 standard, which defines voltage levels between positive and negative 15 Volts, while the PIC only runs on 5 Volts. We therefore added a buffer between the PIC and the computer's serial port to raise the voltage to appropriate levels. For this, we used a MAX233 chip, designed especially for this task. It has internal charge pumps to raise the outgoing voltage to appropriate levels, and step down circuitry to bring RS232 levels down to TTL. The MAX233 buffer can be seen below in Figure 22. The datasheet also suggests a 0.1µF decoupling capacitor between ground and power (not shown). The chip connects directly to a standard 9 pin serial port, shown on the right, and connects to the transmit and receive pins on the PIC.

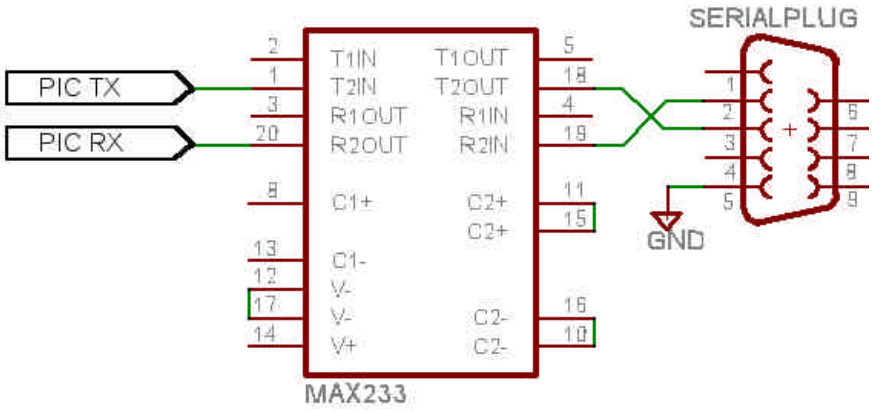


Figure 22 – MAX233 TTL to RS232 Buffer (CADsoft, 2003)

With many ports still free on the PIC microcontroller, we added some features we used as indicators. When the PIC is in the middle of a program, there is no way to tell what it is actually doing. To remove this obstacle we added two LEDs on one of the outputs. The PIC can configure an output pin in three modes. The first is by disconnecting the pin internally, which is the default state on power up. After the PIC boots, it is

possible to leave the pin internally disconnected, or configure it as a low or high output pin. Since we have three modes, we decided to implement a circuit that could show us these different states. The circuit for this can be seen in Figure 23. Pin RC3 on the PIC connects to the center point between the two LEDs and resistors. When nothing connects to the center node, current flows from VCC through the series 400? resistance and the two LEDs, lighting both dimly. The PIC sits in this state before it is booted. When the PIC connects 5 Volts to the center node, the top LED in essence disconnects since current takes the path of least resistance. When the center node is brought to ground, the LED closer to ground turns off, since the internal resistance of the PIC is lower than that of the 200? resistor. By using this simple circuit, we are able to display three different modes on the output. Another benefit of using two LEDs instead of a single on/off is that we always know the current circuit mode. If one LED burns out for instance, we will know that the LED is broken, and not that our microprocessor is not working properly.

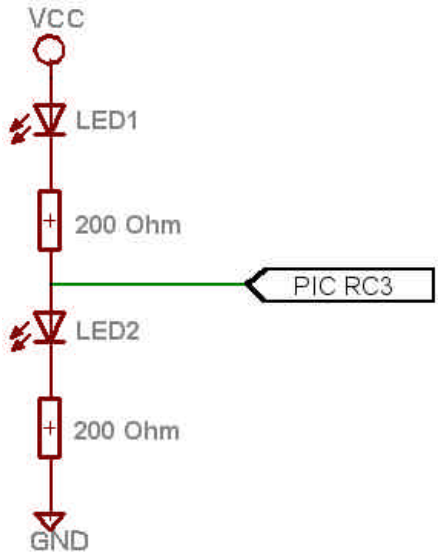


Figure 23 – Indicator LED Circuit

Our motivation behind most of the design in this section was expandability and simplicity. We chose the microcontroller for its robustness, having every feature we needed and room for expansion. Unfortunately, our development timeframe was rather short, and because of this, we needed a simple control system that would get the job done without too many extra components. This prevented us from implementing many

additional features to provide protection throughout the control system, such as buffers to protect our microprocessor.

4.1.1 Control Board PCB Layout

To allow for a portable system, we built an essential PCB for the control hardware. Similarly, to the design of the board containing the magnetic sensors, we wanted the control board to be modular, allowing for expandability and extra debugging features. The layout of the control board, containing the microcontroller, reset circuitry buffer, RS232 to TTL buffer, and other supporting circuitry can be seen in Figure 24 below.

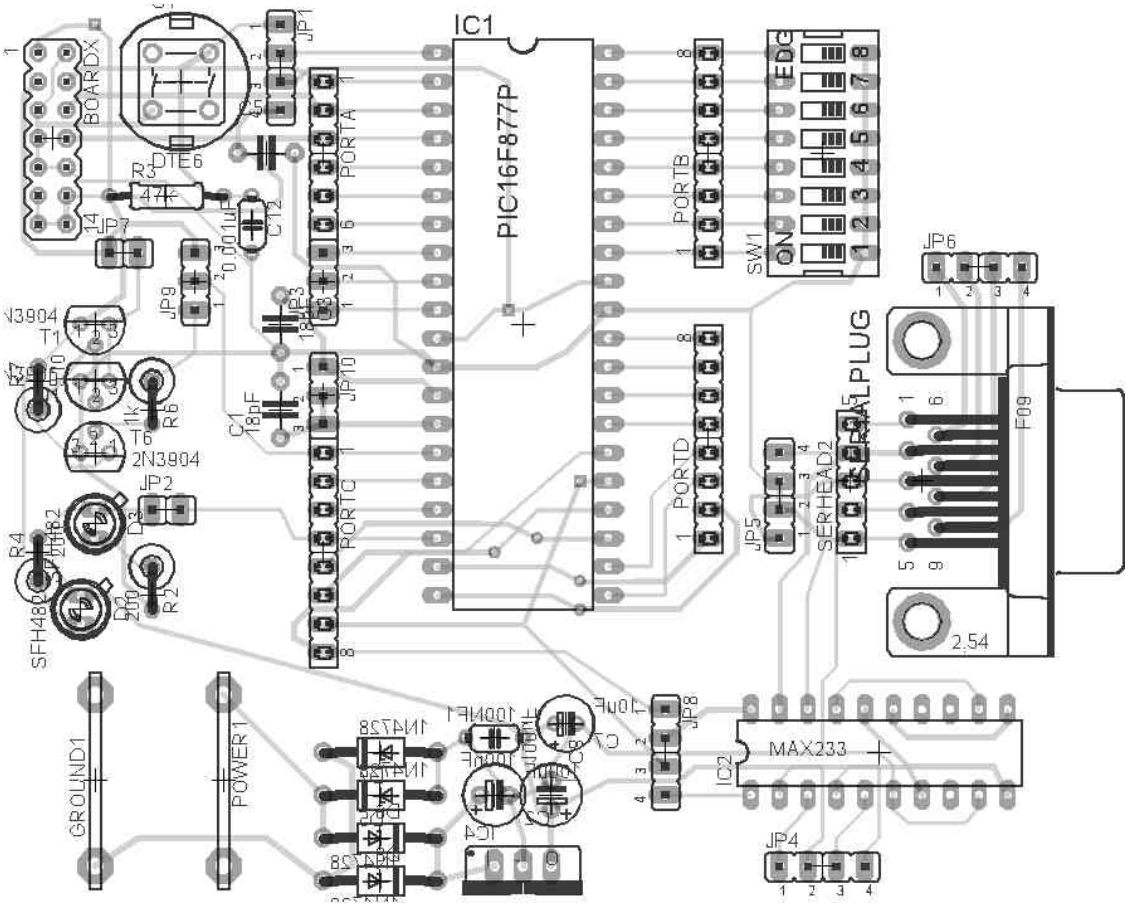


Figure 24 – PCB Design of Control Board

To make the PCB for the PIC microcontroller simple to work with and understand, we grouped portions of the board in a logical manner. In the center is the PIC connected to every system on the board. Starting at the top left corner, we have a 14 pin connector for a ribbon cable to link to the magnetic sensor board. To the right of that is a push button to reboot the PIC. Below this is the buffer for the magnetic sensor reset circuit. Directly below this is a green and blue LED we used for indicating the processor's state. The bars labeled GROUND1 and POWER1 are standard sized banana power plug sockets for easy connection to a power supply. To the right of this we have a full wave rectifier and a 5 Volt power supply. Moving counterclockwise along the board, we come across the MAX233 IC and the DB9 connector used for a serial plug. Right above that is a set of 8 dip switches.

We modeled the PIC reboot circuit containing a pushbutton switch directly after specifications in the Microchip datasheet (Microchip Technology, Inc., 2001). In short, we use a 47k? resistor and a 1nF capacitor to debounce the button. To the right of the pushbutton, is a set of 4 header pins that we can use to disconnect the reboot circuit from the PIC and instead connect it directly to 5 Volts.

To provide for expandability and allow for better testing results, we implemented some header pins and jumpers that allow us to disconnect the buffer circuitry from both the PIC and the magnetic sensor board. By changing jumper JP9, we can directly connect the PIC output to the magnetic sensor board.

The LED circuit also can be disconnected from the microprocessor using a jumper. Leaving the header pins open also allowed us to connect wires to the jumpers checking if other pins on the PIC are working properly.

After having trouble with power supplies found in our laboratory, and destroying two PICs, we implemented a 5 Volt power supply on the PCB. This uses a standard 7805 Voltage regulator, and several decoupling capacitors. To make the power supply even more robust, we implemented a full wave rectifier to prevent any damage if the circuit was incorrectly connected.

The MAX232 chip contains four buffers; however, we are using asynchronous data transfers and therefore only need two of these. Because of this, we added header pins on the connections to the two unused buffers in case we needed them later. To the left of the MAX232 chip is another set of four header pins. These allow us to disconnect both the transmit and receive outputs on the PIC. If the MAX chip becomes damaged, we can connect these pins to the two unused buffers.

Above the MAX chip is the serial port connector. This is a standard 9 pin male connector. As with all the other components on this board, we connected header pins to each of the input pins of the port in case we needed to modify anything quickly.

Lastly, on the very top right of the PCB is a set of 8 toggle switches. These can connect the pins on Port B of the PIC to ground. This port has internal pull up resistor that we can enable in software allowing us to read the Voltage at the pin. If the Voltage is high, the toggle switch is off, and if it is low, the toggle switch is on.

To sum things up, we designed the PCB to be expandable with connections to every input and output on the PIC. We can easily disconnect and reconnect each portion of the circuit by simply swapping jumpers. We created the schematic in a software application from which we generated a PCB layout. This proved to be a very quick and efficient method for creating both a schematic and PCB layout. However, we came across one problem. The software's pin numbering on the serial connector provided to be an inverse of the real part, forcing us to move several traces. Except for this problem, we found the board to be expandable as we had hoped, allowing us to add external components such as a pushbutton for discrete data collection upon request.

4.2 PIC Microcontroller Software Development

To expedite the development time of our three dimensional mouse prototype, we implemented most of our processing and data collection in software. After looking into ready to use data acquisition software, we found that none of the features provided the expandability we needed. Because of this, we wrote our own data acquisition and logging system.

On the acquisition end of the system is the PIC microcontroller, containing software we wrote to control the analog to digital converter, serial transmission, reset circuitry, and miscellaneous debugging features. As such, the PC and PIC are very closely tied in our system. To collect data, the user must first run our application on the computer. This application queries the serial port to see whether the PIC is connected and responding to commands. If it is, the computer sends another command to the PIC, indicating to "start sending data." The PIC then sends a continuous stream of sensor data to the computer until a stop command is received. While sending the data stream, the PIC configures the analog to digital converter, controls the reset circuitry, transmits collected

data to the PC, and displays status messages. The following section describes the software implemented on the microcontroller.

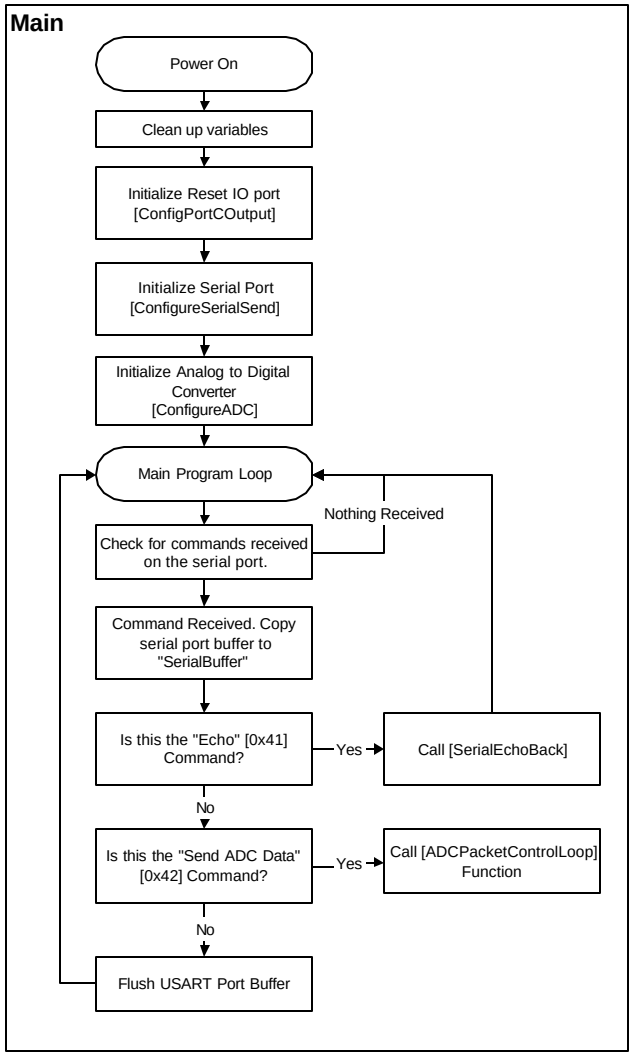


Figure 25 – PIC Software Flow Chart: Main Function

Figure 25 above is the block diagram of the main function that runs on the PIC microcontroller. This section of code runs from the time the PIC receives power to the time power is disconnected. Once the power is turned on, all the variables the PIC needs are defined and initialized to zero or preconfigured values.

First, Port C is initialized by calling ConfigPortCOutput to prepare the magnetic sensor reset circuitry. The PIC then initializes the serial port by calling the ConfigureSerialSend function, described later in detail. After this, the Analog to Digital Converter is configured by calling the ConfigureADC function. After everything is

initialized and configured, the main control loop of the program starts. First, the PIC checks whether it received any commands from the computer. If none, it sits in an endless loop checking for any commands. Once the PIC receives an instruction, the PIC's software decides what needs to be done. Either the PIC executes a command we have termed as "echo", or the microcontroller enters a state in which it continually collects data from the ADCs. The PIC then transmits the information to the computer, and only occasionally checks to see if new commands were received via the serial port. If a new command is received, the software goes back into the main loop to decide what needs to be done.

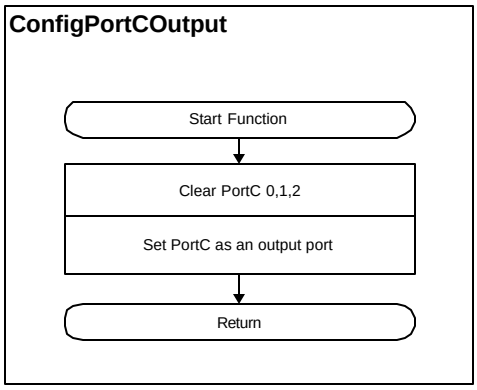


Figure 26 – PIC Software Flow Chart: ConfigPortCOutput Function

Figure 26 above is the flow diagram for the ConfigPortCOutput function. This function simply configures pins RC0, RC1, and RC2 as outputs on the microcontroller and sets them to low. These pins connect to the reset circuitry on the magnetic sensors so that we can control setting and resetting the sensors through the microcontroller. This function is only run once at power up, and returns to the main function after it finishes.

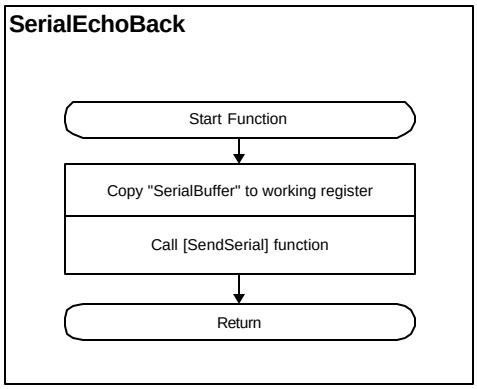


Figure 27 – PIC Software Flow Chart: SerialEchoBack Function

Figure 27 above is the block diagram of the SerialEchoBack function. The job of this function is to take any data received on the serial port and transmit it back to the computer. Our system needs this function so that the computer can query the microcontroller to make sure it is plugged in and ready to send data. After the function executes, the software returns to the main program loop.

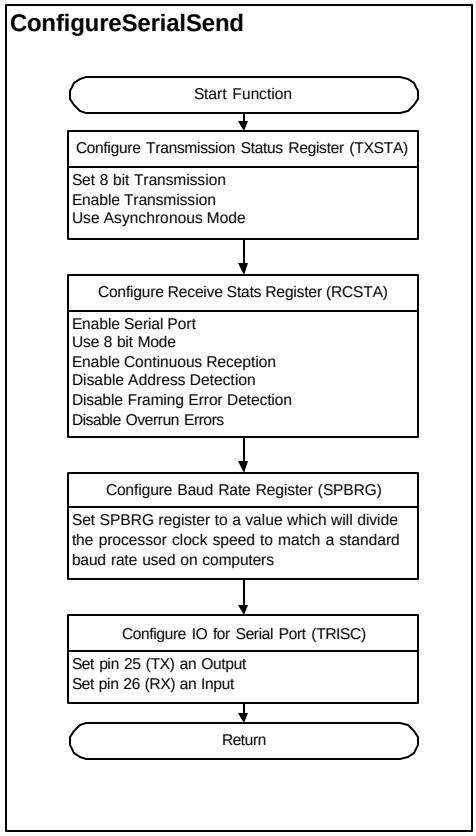


Figure 28 – PIC Software Flow Chart: ConfigureSerialSend Function

Figure 28 above is the flow diagram for the function ConfigureSerialSend, which initializes the serial port on the microcontroller. Four registers control the way the serial port works on the PIC. The TXSTA controls the way in which the PIC microcontroller receives data via the serial port. This register is used to enable the serial port as an output, and configure it for 8 bit asynchronous mode. The RCSTA register is used to enable the serial port module on the PIC, enable transmission in 8 bit asynchronous mode, and disable error warnings. The SPBRG register controls the baud rate used for transmitting data through the microprocessor's built in serial port. After all the registers are loaded with the proper values and the port is configured, the function returns to the startup routine in the Main function.

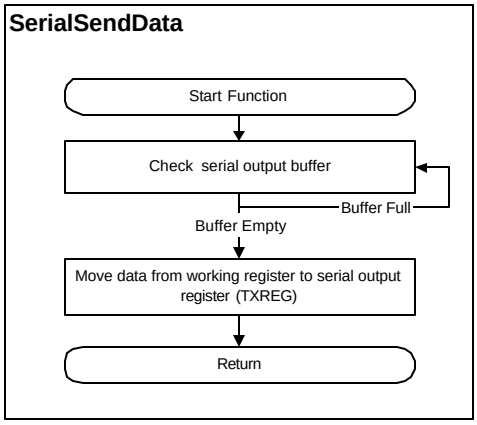


Figure 29 – PIC Software Flow Chart: SerialSendData Function

Figure 29 above takes care of transmitting information from the microcontroller via the serial port to the computer. Any data that needs to be sent using the serial port must be placed in the TXREG. However, before this can be done, a bit needs to be queried to see whether the last 8 bits placed in this register have been successfully sent via the serial port. Once the register is free, the next 8 bits can be placed in the TXREG register and the software can move on to other things. The PIC hardware takes care of transmitting the data at the proper baud rate to the computer.

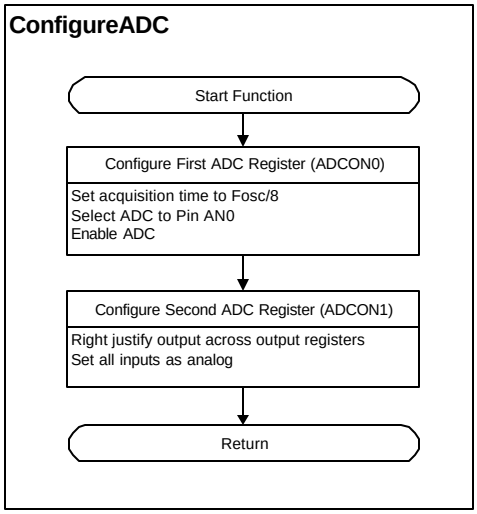


Figure 30 – PIC Software Flow Chart: Main Function ConfigureADC

Figure 30 above is the flow diagram for the ConfigureADC function, whose job is to initialize the Analog to Digital Converter inside the PIC microprocessor. Two registers effect the ADC on the PIC. The PIC uses the ADCON0 register to configure the acquisition time, select the first input pin to the ADC, and enable the ADC module. The input pin must be selected because each analog pin of the microcontroller is multiplexed into the same ADC. The output of the ADC is 12 bits in length. Resultantly, it will not fit in a single 8 bit register, and instead is spread over two registers. Register ADCON1 allows us to specify if the output should be right or left justified over register ADRESH and ADRESL, which contain the most significant and least significant bits.

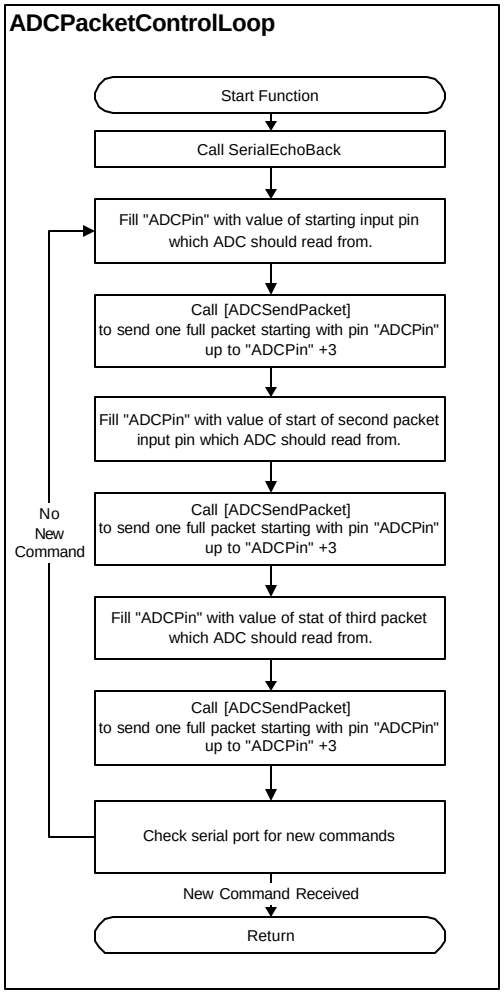


Figure 31 – PIC Software Flow Chart: ADCPacketControlLoop Function

The ADCPacketControlLoop function, found in Figure 31 above, controls the flow of the program once the computer asks for data from the magnetic sensors. This function controls the transmission of three packets, each containing the information from one X,Y,Z sensor assembly. First, the function echoes back the value 0x42 to the computer, meaning, "start data transmission." The variable ADCPin is then loaded with a value of the pin on the microcontroller from which analog data is to be acquired. The ADCSendPacket function is called to convert and transmit to the computer, the analog values found on ports ADCPin through ADCPin+2. Once the PIC sends a full "Picture" of the system's data to the computer, the PIC again checks to see whether the computer has sent any new commands. If none, it prepares to send another "Picture."

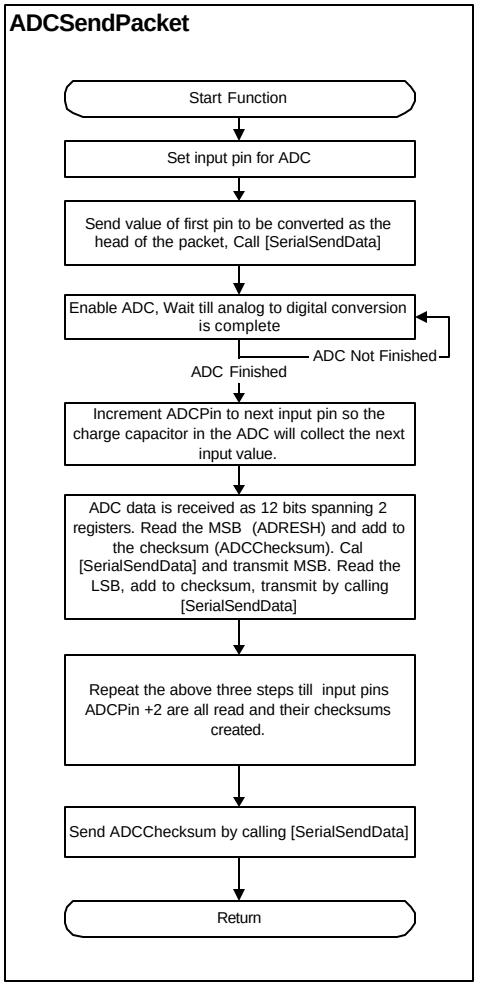


Figure 32 – PIC Software Flow Chart: ADCSendPacket Function

Figure 32 above shows the flow of the ADCSendPacket function. The control loop previously described executes this function. This function reads the value of ADCPin, sets the multiplexer in the ADC module to read that pin, and creates a head packet, which is merely the value of ADCPin. The head packet is then sent via the serial port, after which conversion of the ADC is started. Once the ADC is finished loading ADCRESH and ADCRESL registers, ADCPin is incremented, and the ADC multiplexer is set to the next input pin. While the sample and hold capacitor of the ADC charges to the multiplexed input pin, a checksum of the previously acquired analog data is calculated and the PIC transmits everything to the computer. Once this is completed, the ADC is again enabled to convert an analog value to a digital value. This process repeats until three input pins are converted, a checksum of all the data is calculated, and all values are transmitted to the computer. Once this is finished, the function returns.

4.3 PC Software Development

The hardware system described earlier required a real time data acquisition and logging system in order to manage the large amount of data acquired each second. However, early on we realized that the addition of a real time display would be advantageous for testing and presenting the system. After looking at our group's skill base and trying industry standard data acquisition software, such as LabView, we decided that creating our own program in the object-oriented language of Visual C++, a skill that our group has, would work well for our application. The language is flexible and established well enough to not only allow us to collect and log data in real time over a serial interface, but also to display this data graphically on any PC running a Windows operating system.

We built the final program in multiple stages, first basic serial communication, then data acquisition and logging and finally the real time display. The major hurdle in implementation was not any single section but the overall structure of the program. Our initial conception of this structure lacked a good understanding of how system time delays would affect PIC to PC synchronization, serial communication speed (baud rate), and data acquisition to display time. We restructured the program three times, each time we learned more about the location of these time delays and how they affected our ability to reliably collect and display data in real time. In the end, we were able to meet our initial real time data acquisition and logging requirement along with an extremely accurate real time display of the sensor data. For a better understanding of our final program, we first included an explanation of the style of program architecture we utilized called multithreading, and then an explanation of the second stage of our program development.

Every program currently running on a Windows PC is called a process. Holding down the Alt, Ctrl, and Delete key simultaneously provides a listing of all processing on a machine. A thread resides within a process and can be considered a linear program running in the global program space of its parent process. A multithreaded program is a process that contains multiple linear programs each devoted to a specific task. They share global program resources, such as variables and functions, CPU cycles, and communicate with each other to accomplish the program's main task. The operating system (OS) utilizes the concept of multithreading to manage processor use, allowing multiple applications to be run at the same time. The OS accomplishes this by switching CPU

cycles between each process and the threads within each. Many software languages include classes or functions to manage the large variety of issues that arise when a developer cannot guarantee the flow of a program since the OS controls switching threads in and out of the CPU. Visual C++ can utilize Microsoft Foundation Classes (MFC), which contains many resources specially designed for multithreaded applications; these include thread synchronization, communication and "ThreadSafe" resources. It is the programmer's job to control communication and synchronization between threads and write code in such a way that it is "ThreadSafe". A non-TreadSafe resource is also non-reentrant and is best explained with an example. Say a process contains two threads, one thread is currently utilizing a function to format a character string and the other is waiting to enter the same function. Midway through conversion, the OS suspends the active thread and the inactive thread is activated and allowed to enter the same function, corrupting the shared data. "A ThreadSafe function or resource protects the data by taking care of the details of thread synchronization internally, allowing threads with a need to access the resource to do so as if they were in a single-threaded process" (Weisz, 1996). As seen, it is a complex task to coordinate a multithreaded application but we found it was required in order to provide our system with real time capabilities.

Our first cut application, Revision A, was not multithreaded and accumulated delays causing the program eventually to loose sink with the serial data stream when the PC serial port buffer overflowed. This was unacceptable so we looked for a resolution. The second application, Revision B, consisted of two threads, the Main Application Thread and the Read Write Thread. The Main Application Thread controlled all interactions with the graphical user interface (GUI) and calculated the display position. The Read Write Thread controlled serial communication between the PIC, parsing the incoming sensor data and writing data files. In addition, we implemented system timers to allow the display to update at a consistent rate; the OS managed the signaling of timers but ran them in the Main Application Thread. The major drawback of this application was that the display lagged behind the data acquisition. We hypothesized that it was due to the large delay incurred when writing to the hard drive between each serial port read. Our last option was to spilt reading and writing data into two threads. The final program, Revision C consists of three threads, the Main Application Thread, Read Thread, and Write Thread. Similarly, to Revision B, the main application thread controls all interaction with the GUI, calculates the display position and implements the update display timers. The Read Thread controls serial communication between the

microcontroller and parses incoming sensor data. The Write Thread simply writes the raw data to the file. Below is a detailed description of Revision B and Revision C in order to describe the major problems incurred during development causing us to increase the complexity of our program.

4.3.1 PC Software Architecture Revision B

This revision worked relatively well and consisted of two threads, the Main Application Thread and the Read Write Thread; it communicated with the microcontroller, collected the serial data, wrote the data to a file and displayed the data. The major problem was that the display lagged behind the acquired data. Moving a magnet above the sensor did not result in a displayed change until approximately two seconds later. The promising fact was that the displayed movements were extremely precise; the resolution of the display was excellent. It was just that the data fed to the display was approximately two seconds old. Since we designed and implemented all major functionality through the completion of this program revision, we provided a detailed description of Revision B below and only described the final program, Revision C in relation.

Main Application Thread

The main application thread is the "Parent" to all other application resources. It controls creating and closing the GUI, Read Write Thread and OS controlled timers along with overall program initialization and clean up. The GUI is a dialog window where the user can enter the Com Port number, start and stop the program, and view the output magnitude of each sensor in real time. Each timer is an OS registered interrupt that occurs at a user-defined frequency. The code linked with the timers controls the display update at a consistent interval allowing very close to real time display of the sensor data without complex display timing.

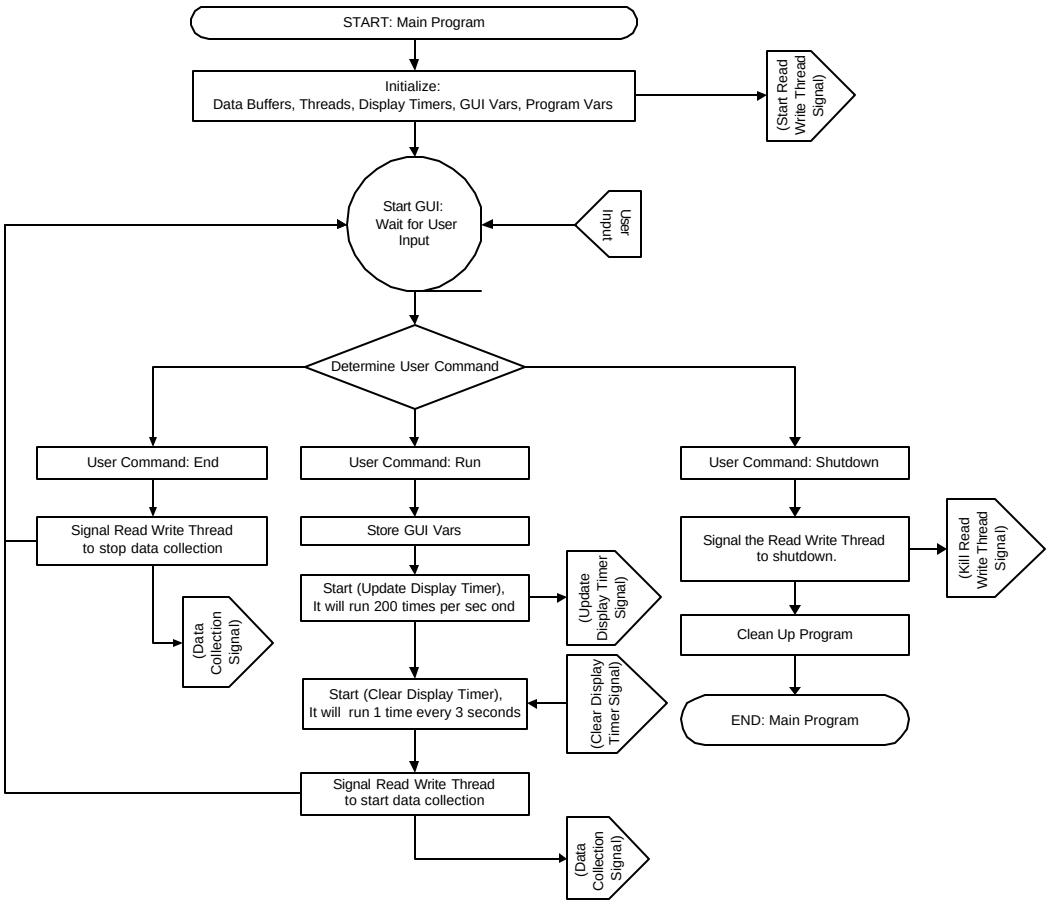


Figure 33 – Revision B: Application Initialization and GUI Management Diagram

The application initialization and GUI management block diagram in Figure 33 above begins by initializing all system wide resources. The Read Write Thread is created here but in a suspended state. This means that it does not begin executing until signaled. Next, the main application thread displays the GUI and then goes into an efficient wait state until the user clicks a button. The OS manages the button click and routes it to the proper message handler. The established handlers include Run, End, and Shutdown.

The user clicks the "Run" program button routing program execution to the Run Program function. This function stores the GUI variables, which includes the Com Port number. Then it starts the Update Display Timer and Clear Display Timer, which controls the display of the sensor magnitude bars in the GUI. Finally, it sets the thread communication signal, which tells the Read Write Thread to begin its data collection and processing. When this function exits, the GUI returns to a wait state until further user input.

The user clicks the "End" program button routing program execution to the End Program function. This function signals the Read Write Thread to stop data collection and processing, and then it suspends the thread so that it minimizes the use of unnecessary system resources. Finally, it cleans up any per program initialization variables so that the program can restart successfully without having to be shutdown. When this function exits, the GUI returns to a wait state until further user input.

The user clicks the "Shutdown" program button routing program execution to the Shutdown Program function. This function signals the Read Write Thread to stop data collection and processing and then it signals the thread to kill itself. Next, it cleans up all memory allocated during run time. Finally, it signals the main application to close and clean itself up.

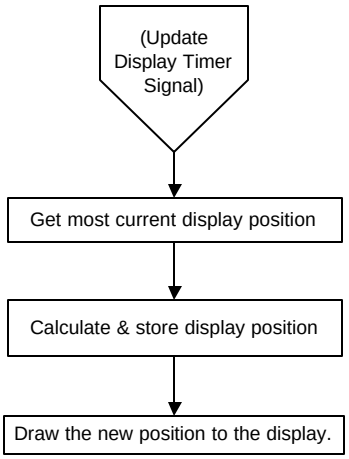


Figure 34 – Revision B: Update Display Timer Block Diagram

The display timer in Figure 34 is responsible for synchronizing with the Read Write Thread to get the most current raw data for display. Once it obtains the data point, the current display position in pixels is calculated and stored. Finally, the application draws the new position to the screen. The function exits and is called again by the OS when the next timer interval elapses.

Read Write Thread

The Read Write Thread controls all serial port access, memory access, PIC communication and raw data integration. It is responsible for initializing the serial port, handshaking with the PIC, checking the serial data for errors, storing the data in memory

and writing the raw data to a file. Upon creating the thread, it is in suspend mode, meaning that the thread will not execute until signaled. When the user selects the Run command on the GUI, the Run function resumes thread operation allowing the thread to enter its main function loop.

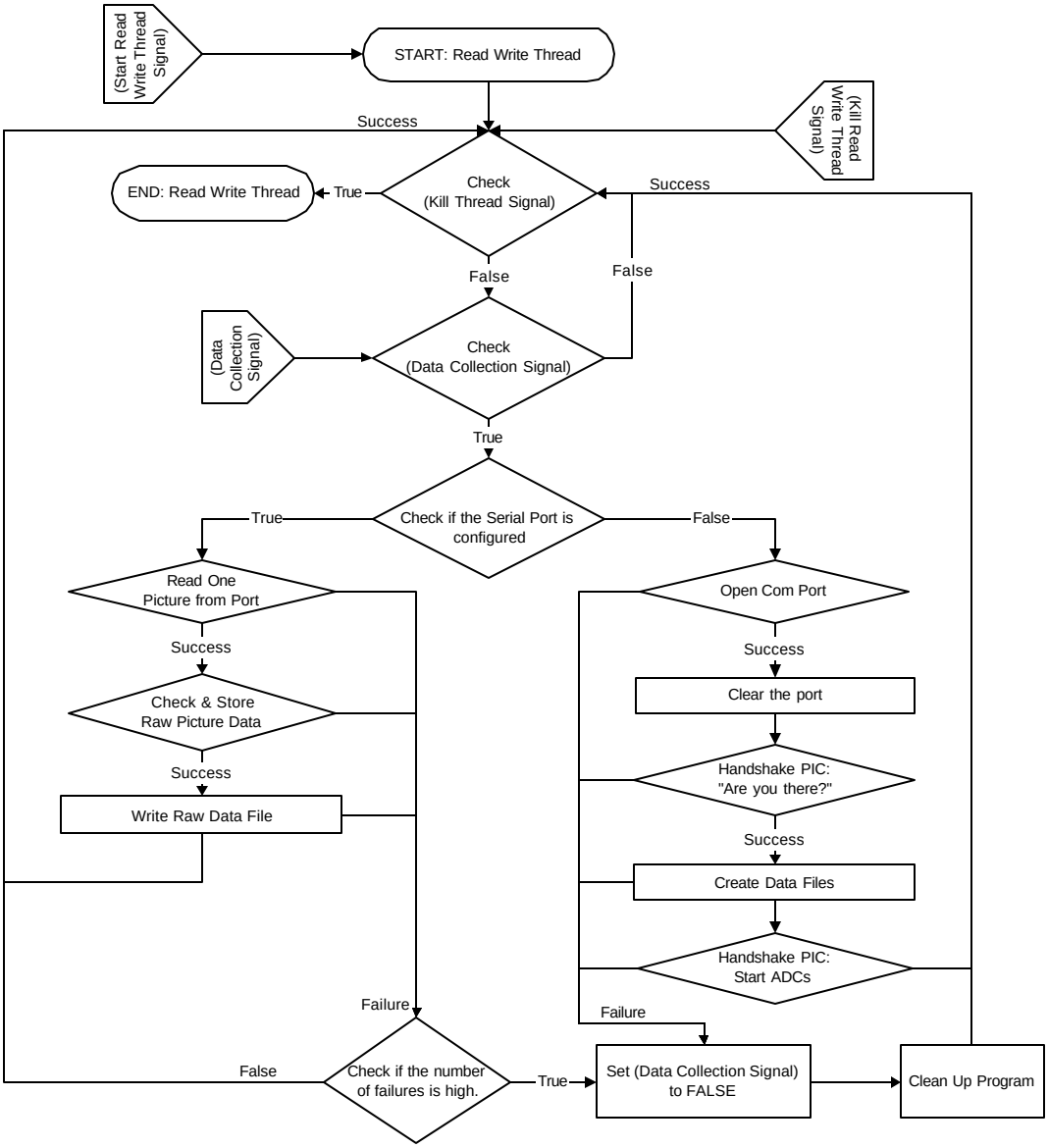


Figure 35 – Revision B: Read Write Thread Main Loop Block Diagram

The main loop in Figure 35 begins by checking whether a terminate message has been sent to the thread, if so it exits here otherwise it checks if it should be running and collecting data. The Main Application Thread sends the Start Data Collection command

when the user clicks the Run button. If the thread is in data collection mode, it begins by configuring and clearing the serial port to get ready for PIC serial communication. Then it will attempt to confirm that the PIC is connected. If the PIC responds, the thread continues with program initialization by creating a raw data file on the user's system to hold the raw sensor magnitude data. Finally, it tells the PIC to start its analog to digital converters (ADCs) and send the data to the PC.

Once the thread configured the port and the PIC is sending serial data, the thread returns to the start of the main loop to check its activation status before moving on to reading from the port. The quantity of data read from the port is always 4,000 packets since the serial hardware buffer is 4096 Bytes. We tried to limit the number of read cycles to increase the speed of the program since each read operation is extremely time consuming. Once the serial read is complete, the thread sorts the serial data, removing the head and tail Bytes while checking for serial data transfer errors utilizing the transmitted checksum. Finally, the thread converts the sorted raw magnitude data to decimal and writes it to a comma delimited file. This process of checking thread activation status and then reading, sorting, checking and writing data continues until the program encounters a hardware or data error and aborts the transmission or the user stops or shuts down the program.

The Read Write Thread caused the major bottleneck in our system. As we described earlier a thread is a linear program running in global application space. First, the thread reads from the port, then it processes the data read and finally writes the data to disk. We misunderstood where the delays in this chain occurred so we could not properly compensate. Initially we believed that accessing the serial port hardware to collect the data was "slow", so we increased the amount of data read from the port, limiting the number of times we had to access the hardware. However, we were mistaken, and this mistake was costly, it actually created a display lag of one second at this baud rate. We also knew that writing files to the hard drive incurred a large delay but we did not realize how large. Since we accessed the hard drive for a write operation every read cycle, 4000 bytes, we made the reading operation wait for writing to complete every read cycle. In short, the delay between data acquisition and display was upward of 1.5 seconds. For Revision C, we hypothesized that if we split the Read Write Thread into two threads, one devoted to reading from the serial port and processing the data and another for just writing to disk we could remove the predetermined delay. In addition, if we changed the frequency of reading and processing of serial data so that it occurred every packet we

could guarantee the display the most current data. The final improvement is that the write operation will occur at a small interval, so that its large delay interrupts the system infrequently.

4.3.2 PC Software Architecture Revision C

This revision contains the program architecture that allowed us to create a real time data acquisition and display system. Revision B was close to fully functional, its flaws taught us a great deal about how to make the final jump to the true real time application. This system as stated earlier contains three threads, the Main Application Thread still in control of all GUI operations, the Read Thread that controls serial communications and raw data processing, and finally the newest addition, the Write Thread in control of writing the raw data files. Since a lot of the code in this application overlaps that of Revision B, what follows is only a detailed description of the additions.

Main Application Thread

This thread is almost identical to that of Revision B. The major difference is it now manages two threads, the Read Thread and the Write Thread. Since both threads begin and end at the same time and by the same communication signals, the code changes here were minimal. They included initializing the Write Thread at the same time as the Read Thread and closing the Write Thread at the same time as the Read Thread.

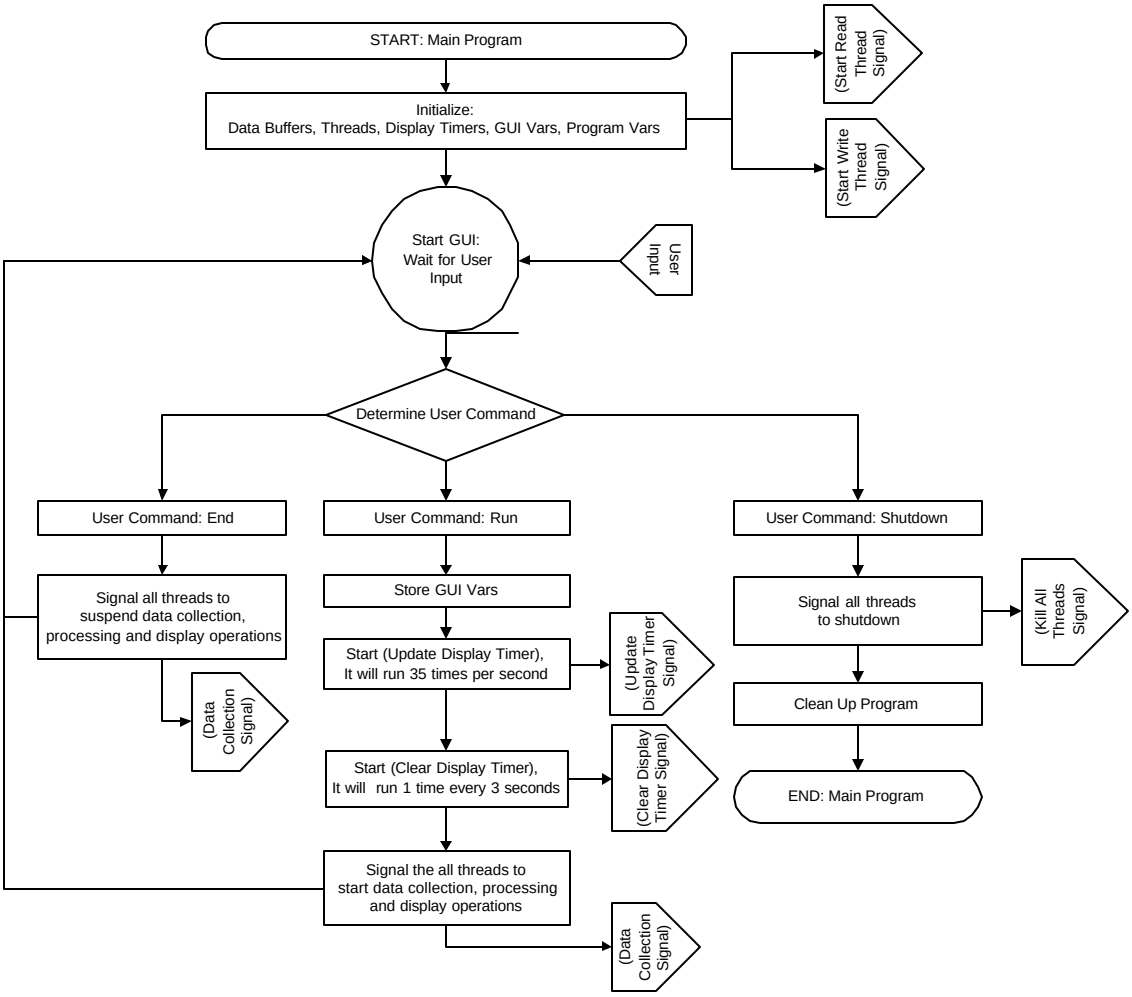


Figure 36 – Revision C: Application Initialization and GUI Management Diagram

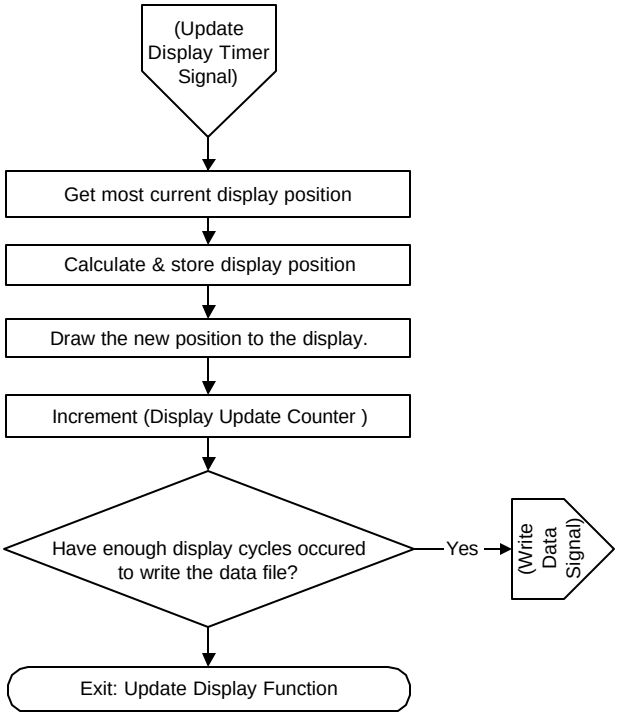


Figure 37 – Revision C: Update Display Timer Diagram

The display timer is also very similar to Revision B. It obtains all prior responsibilities along with the additional responsibility of signaling the Write Thread to write data. Every time this function is called it increments a variable. When that variable is equal to the number of display cycles between a write to disk, this function sends a signal to the Write Thread and the Write Thread writes all new data to disk.

Read Thread

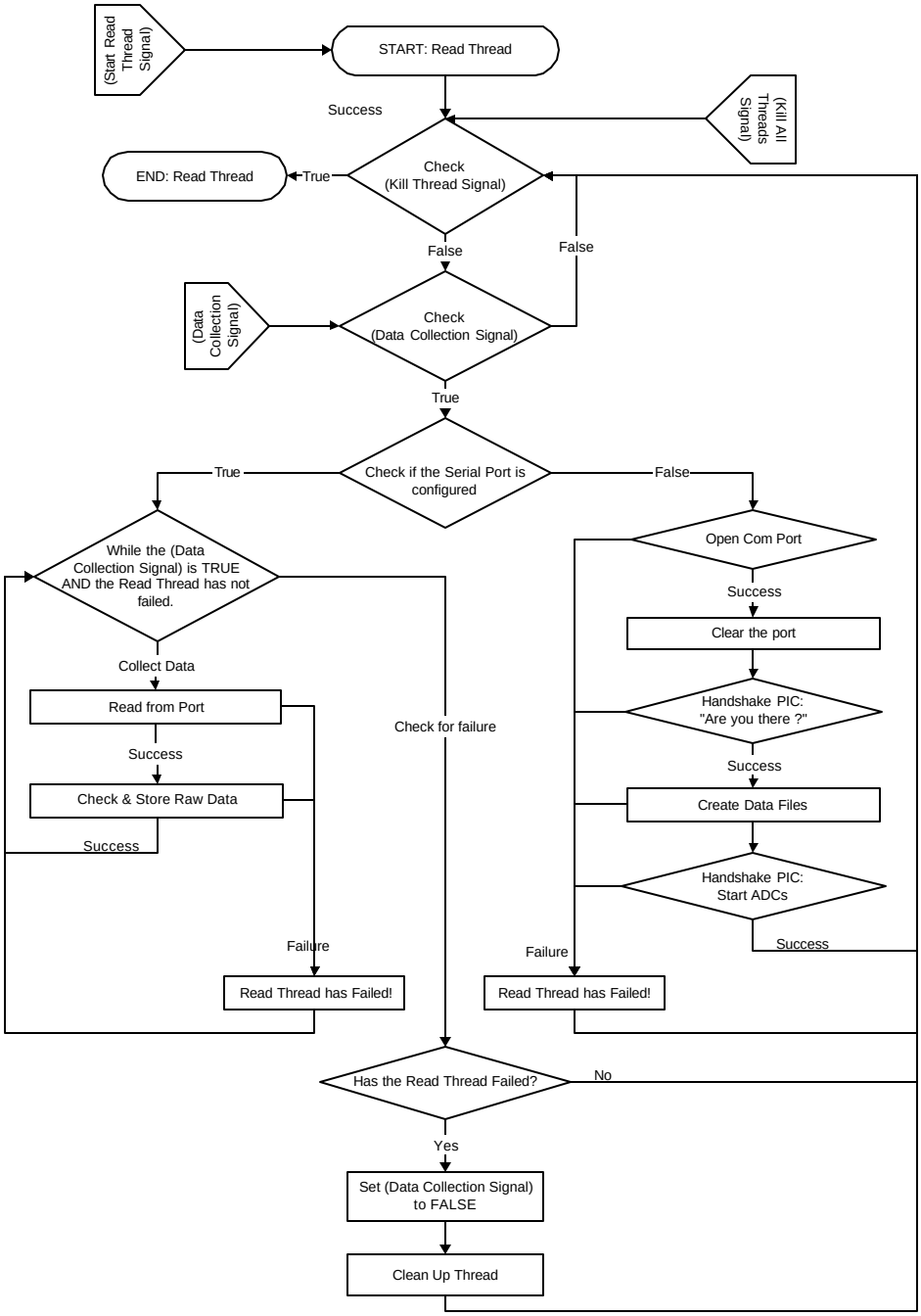


Figure 38 – Revision C: Read Thread Block Diagram

In comparison to Revision B, this thread looks extremely similar to its Read Write Thread. Most of the functionality of that thread became the Read Thread of Revision C. The major changes to the overall structure occurred when we moved the function that

wrote data files to its own thread and changed the Read From Port function to read one packet and then return, versus 500 packets. In addition, we slightly changed the structure of this thread. Once the serial port is configured, this thread checks if data should be collected (Data Collection Signal) and checks its self diagnostic flag to see if it is still functioning properly. If so, it enters the main function loop. This loop contains two functions, Read From Port and Check and Store Raw Data. By tightening the read loop and only checking the required variables each iteration we hoped to limit the delays between successive reads.

Write Thread

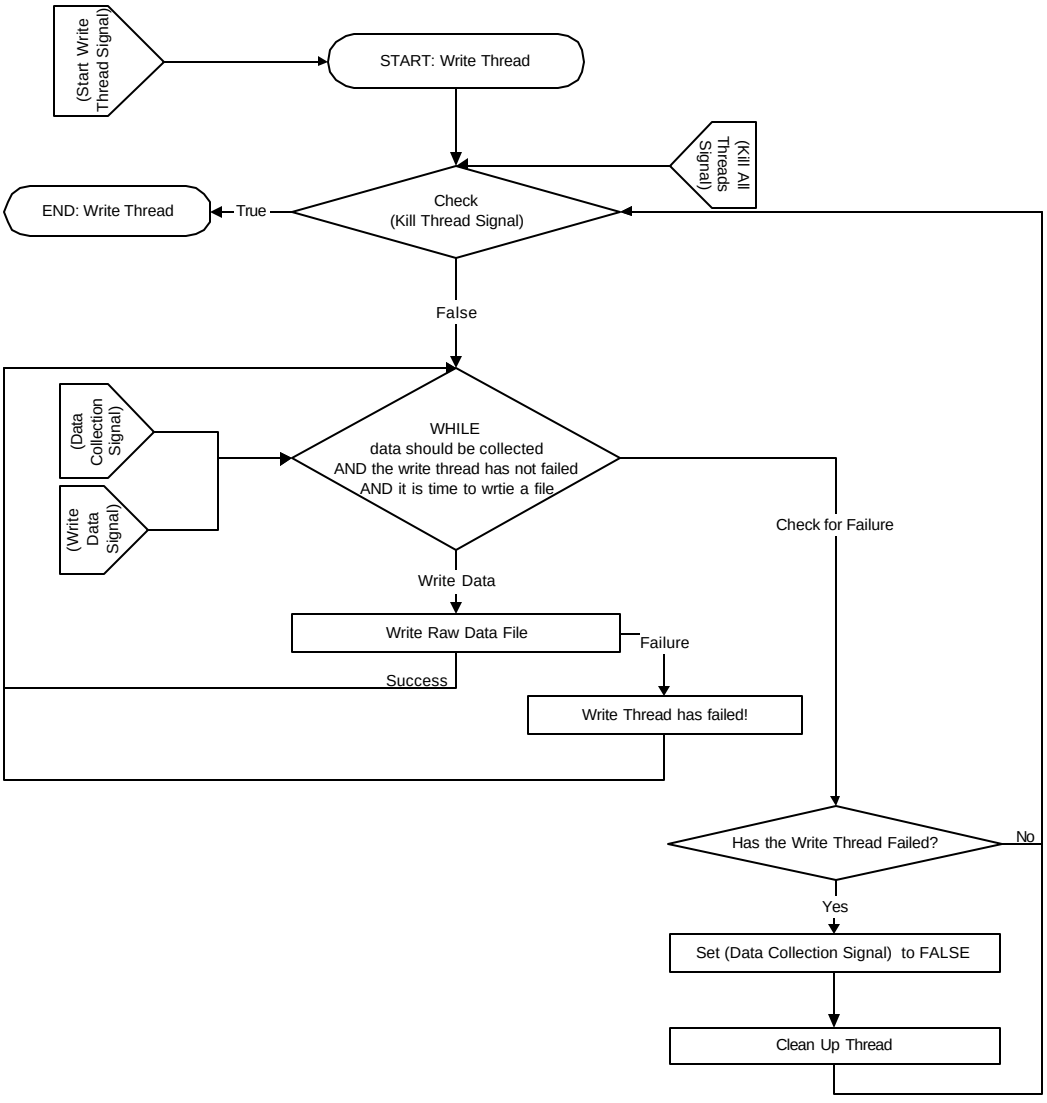


Figure 39 – Revision C: Write Thread Block Diagram

The Write Thread is the newest addition to this program. Its functionality is the same as the function located in the Read Write Thread of Revision B that was responsible for writing raw data to a file. When the Main Application Thread creates this thread it begins by checking if it should be active (Kill Thread Signal). Then it checks the state of three software signals for TRUE; one, if data should be collected, two, if the Write Thread has not failed on a prior loop, and, three, if it is time to write a file. If all conditions are TRUE, it writes all new data to a file. If not, it proceeds to check for program failure and if the thread should still be active (Kill Thread Signal). If neither case is TRUE, it checks all three conditions again. This pattern continues until either the user presses the stop or abort button or the program fails.

Chapter 5– Results & Conclusions

From the start, we divided designing and developing a three-dimensional mouse proof of concept into two stages. The first stage focused on a conceptual design for a fully functional 3D mouse and the second stage focused on designing a proof of concept system that would demonstrate the viability of our conceptual design. The first breakthrough occurred after much research and experimentation when we chose magnetoresistive sensors as the sensing technology for our system. Now we could continue with the second stage of the project, which was to build a prototype to demonstrate the feasibility of 3D object tracking. After extensive debugging, testing and development, we finalized a working system capable of displaying the magnetic sensor data in real time.

However, merely displaying magnetic field strength as we moved a magnet did not provide any proof of the workings of our system. Through a series of experiments, we would need to show that our system is capable of tracking the movement of objects in some number of dimensions, and that this is expandable to a fully functional three-dimensional mouse. The burden of proof lay on us to demonstrate a reproducible correlation between the physical position of a magnet and its representative location on a computer To meet this burden, we took our constructed single dimensional system and devised trials to show a correlation in representing our single axis system in an X, Y, and Z plane.

5.0 Experimentation

To determine the functionality of our system, we developed and ran several experiments using a bar magnet, aligned in different orientations, as the target. For these experiments, we placed our magnetic sensor board and control board in the center of a plastic box measuring 26cm wide by 35cm long by 4cm deep. On the top of the box, we drew a grid of square centimeter cells totaling 24 columns and 33 rows resulting in a resolution of 792 squares. The bar magnet we used for testing is 7cm long, by 1.5cm wide, by 1cm deep.

For acquiring data from our experiments, we modified our real time data logging software, as well as added several extra features to the PIC software. We connected a button to an auxiliary port on the PIC to facilitate data acquisition. By setting the system

in “button experiment mode,” the PIC sends packets as it normally would, but they are filled with zeros. Once the button is pressed, the PIC sends 254 packets of data to the PC, and then sends zeros again. During this time, the PC listens to the incoming data and discards all packets containing zeros. Once real data is received, the PC checks to see if it received 254 packets. If they were not received, the frame is dropped. If the data is correct, the system averages all values and a single value for X, Y and Z is saved to the log file. To make the system more user-friendly, every time the PC detects 254 successful packets it beeps letting the user know the PC successfully received the data from the single button click. This alleviates the need to check constantly the display while moving the magnet.

With the additional software data logging modes, we performed three experiments to characterize our system. We designed these experiments to portray our system as if we implemented multiple sensor boards. Instead of moving the plane into three separate orthogonal positions, we rotated the magnet in different orientations. The purpose of the experiments was to take a reading of the X, Y, and Z sensor data with the magnet over each of the 792 squares. From this data, we were able to generate three-dimensional plots showing the magnet’s field across the sensor board from each of the three magnetoresistive sensors.

Unfortunately, before running these experiments we had a very limited understanding of what we expected to see. From our research, we knew that we should be able to acquire a graph of the magnetic field magnitude at every point in the X, Y and Z direction. Beyond this, we did not have a firm grasp on what our data would show.

5.0.1 Experiment 1: Vertically Orientated Magnet, North Pole Up

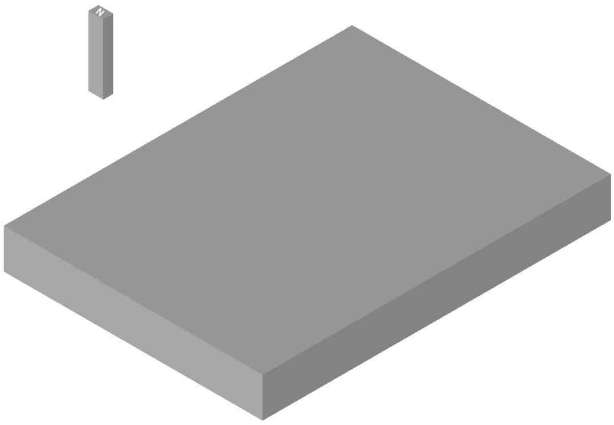


Figure 40 – Experiment 1: Magnet Orientation

In our first experiment, we held the bar magnet perpendicular to, and 10cm above the sensing board as seen in Figure 40. Additionally, the south pole of the magnet faced towards the board. As with each experiment, we started in the top left hand corner and collected 254 samples of data for an average reading at every square on the experiment box. Once we reached the last square in the bottom right hand corner, we had enough information to determine the magnetic field strength in the X, Y, and Z directions across the test platform. The graphs depicted below show a representation of this data that we entered into DADiSP, a software application for 3D graphing.

An additional aspect of the magnetoresistive sensors crucial to our experiments is the orientation of each sensor's axis in relation to our two-dimensional and three-dimensional graphs. The X sensor is situated in a horizontal orientation, Y is vertical, and Z is viewed as coming up out of the page, as seen in Figure 41.

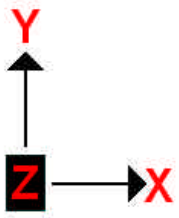


Figure 41 – Sensor Orientation

Figure 42, Figure 43 and Figure 44 found below contain 3D contour plots created from the X, Y, and Z sensor data recorded in our experiment. The plots represent the data recorded above each of the cells in the test box grid. The color scales we used move from red to blue with red being the most positive voltage on each graph and blue representing the most negative. It is important for the reader to note that the output of the sensor when no magnetic field is present rests at 2.5 Volts. A proportional swing in the positive or negative direction from this DC offset denotes the same change in field strength. However, the direction of the change states the direction of the applied magnetic field. The following three images present, at the top, a three dimensional view of the plane from the side, and at the bottom, a density plot of the same data.

Figure 42 shows the plots generated from data collected by the X sensor. In reference to the sensor, we orientated the magnet with the south pole facing the board, and orientated the sensor horizontally along the narrow portion of the image. The large hump, indicated in dark red, was produced by the magnetic field as it moved out the

bottom of the magnet, curled towards the right, moving horizontally, and passing through the sensitive axis of the sensor, before curling higher up towards the north pole. The magnetic field was weakest at the very center of the image, represented by a light blue/green color. This is because at the center, the magnet's south pole was directly over the sensor with no field component inline. When we moved the magnet to the right side of the test board, we saw the inverse effect of the left side. As the field exits the magnet, it curls left along the sensitive axis of the magnet, before curling towards the north pole. Since the magnetic field moved from left to right across the magnetic sensor, opposed to from right to left, the sensor picked up the inverse field strength causing a dip. The field strength is the same in both the dip and the hill; however, the direction of flow is reversed. Figure 43 is a graph of the data acquired from the Y sensor. The data pattern resembles that of the X sensor since the X and Y sensors are offset by 90 degrees. We expected this pattern as the sensors only sense the components of a magnetic field that pass directly through the sensitive axis from tip to tail.

The last graph from our first experiment, seen in Figure 44, is a display of the data collected by the Z sensor. In this case, the sensor is orientated inline with the magnet and its south pole faces the board. When the magnet is directly over the sensor, we saw the strongest field. The sensor's measured field strength weakens as the angle by which the magnetic field interacts with the sensor increases from zero to 90 degrees. As this angle approaches 90 degrees, the field effect approaches zero. When the magnet is located on the fringes of the board, the majority of the field lines no longer run through the sensor from tip to tail, lessening their effect.

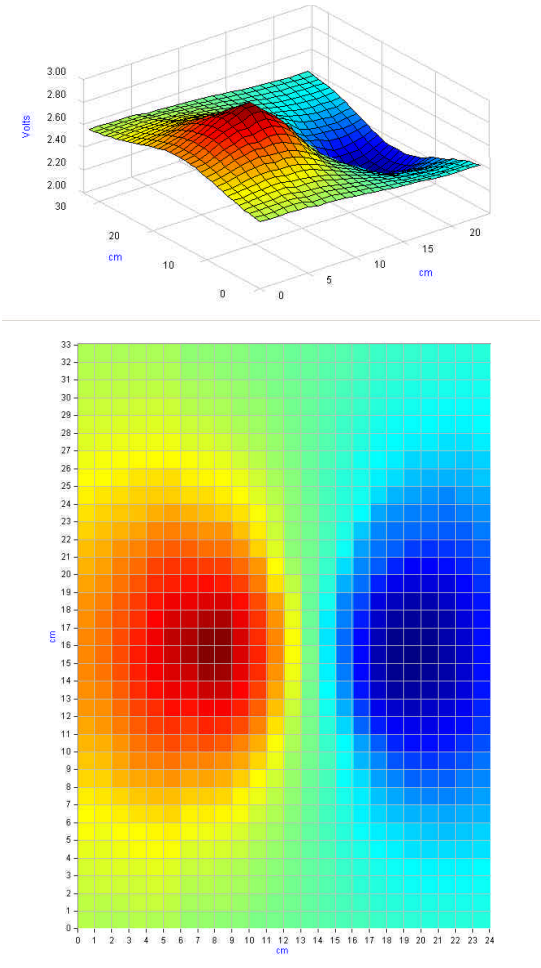


Figure 42 – Experiment 1: X Sensor

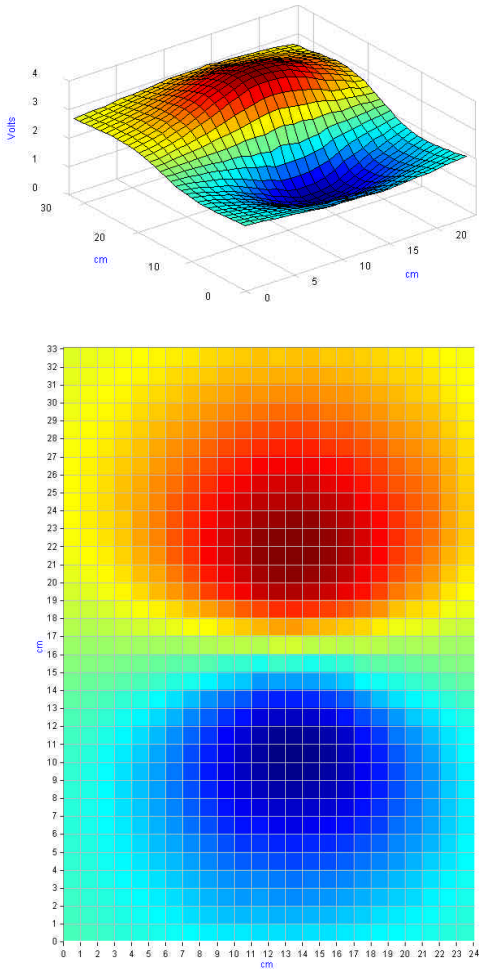


Figure 43 – Experiment 1: Y Sensor

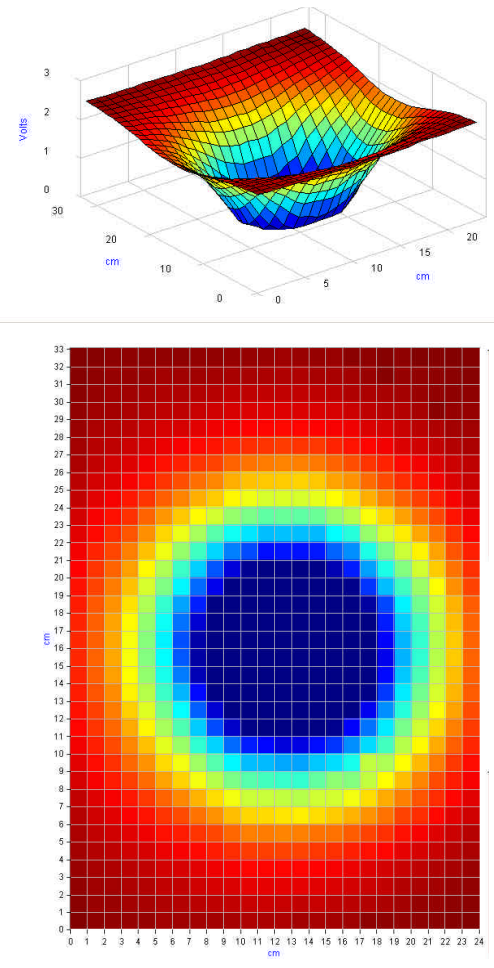


Figure 44 – Experiment 1: Z Sensor

5.0.2 Experiment 2: Horizontally Orientated Magnet, South Pole Facing Right

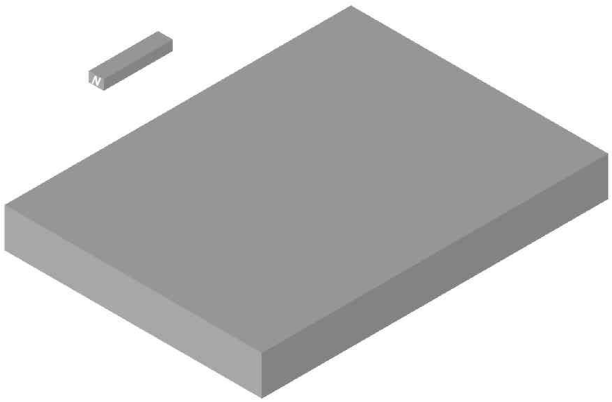


Figure 45 – Experiment 2: Magnet Orientation

In the second experiment, we kept all variables the same, except for orientation of the magnet. This allowed us to represent the picture of the magnet's field from a plane orthogonal to the one in Experiment 1. We placed the magnet horizontally over the plane at a distance of 10cm with the south pole pointing to the right of the board. Again, we measured the magnetic field strength over each square centimeter.

Quickly comparing the information to Experiment 1, we saw a great number of similarities. Figure 47 representing the Y plane is very similar to the picture of the Z plane in the first experiment. Since we rotated the magnet by 90 degrees to match the orientation of the Y sensor, it is understandably a similar view to the Z plane. However, there is a difference between the two. In the first experiment, we held the magnet so that it pointed towards the board with its south pole creating an enormous response. In this experiment, the magnet is parallel to the board and therefore neither pole points directly at the board. As a result, the sensor is only able to detect the field as it loops around the magnet from tip to tail, detecting a much weaker response.

The pattern seen in Figure 48 from the Z sensor is also recognizable as the pattern delivered by the X sensor from Experiment 1. However, in reference to the orientation of the magnet's field, the Z sensor is not positioned in the same fashion as the X sensor. The response we received from the Y sensor in Experiment 2 was much smaller compared to the Z sensor of Experiment 1 because the fields through the sensor came from a parallel-orientated magnet and not from a vertical-orientated magnet.

Figure 46 is unlike any other graph we have seen so far. In this graph, there are four corners where the magnetic field is very strong, with null points between them shaped as a crude plus sign in light blue. This graph is best explained by examining the magnet's effect at the edges of the box. The first situation is when the magnet is at the bottom left corner with the north pole facing upwards. The field exits the magnet at the top, curls to the right and intersects the magnetic sensor in its sensitive direction. Once we moved the magnet upwards, the angle of the field decreases until the magnet is in the center and perpendicular to the sensor. When this happens, there is a null point where the sensor does not detect a field. As we moved the magnet to the upper half of the image, the fields flowing into the north pole become orientated in the same fashion as in the lower left hand corner. However, the field is now seen in an opposite direction causing an inverted output. The same logic behind this effect on the left hand side applies to the right hand side of the box.

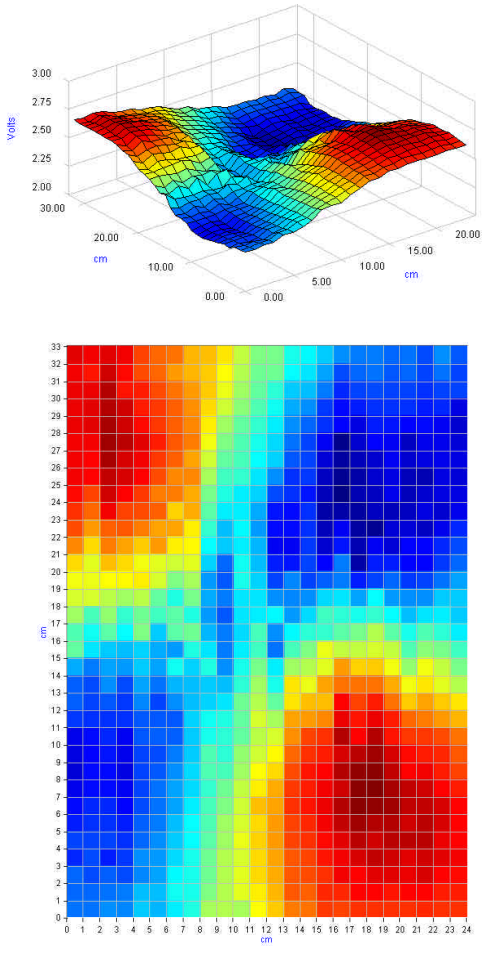


Figure 46 – Experiment 2: X Sensor

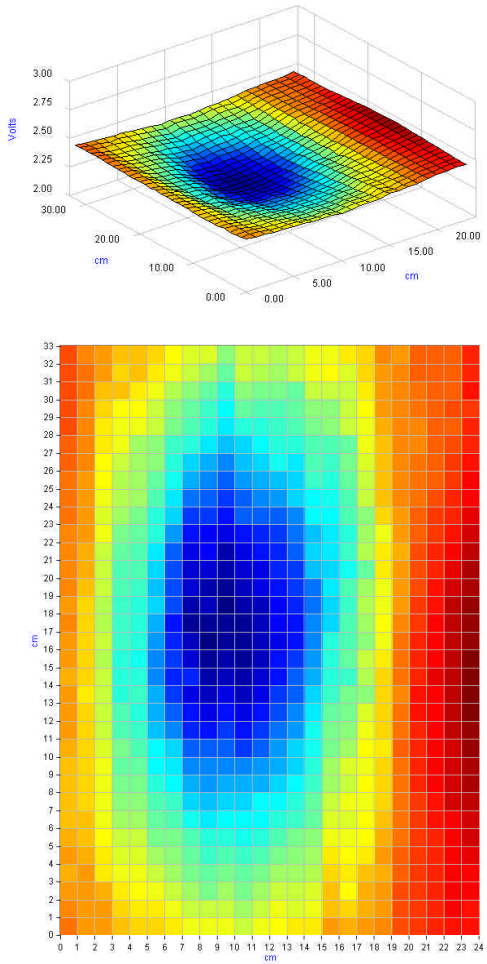


Figure 47 – Experiment 2: Y Sensor

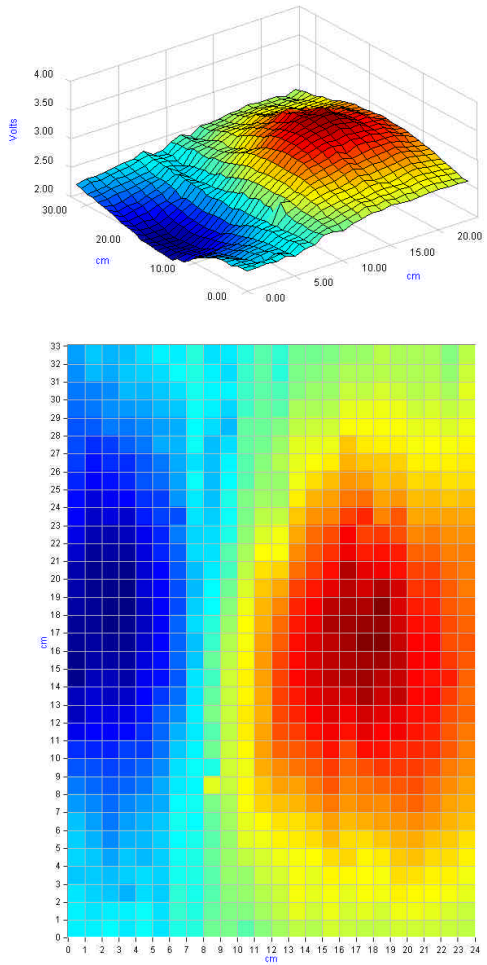


Figure 48 – Experiment 2: Z Sensor

5.0.3 Experiment 3: Horizontally Orientated Magnet, South Pole Facing User

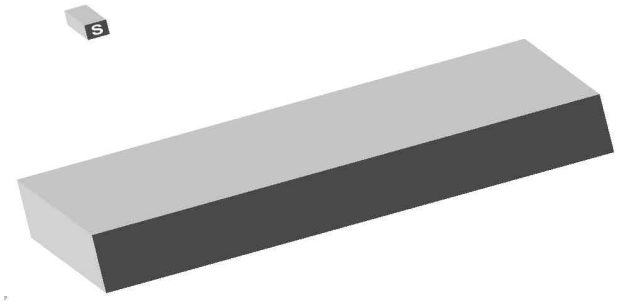


Figure 49 – Experiment 3: Magnet Orientation

We ran the last experiment to characterize the view of a third plane in an orthogonal system. Again, we held the magnet 10cm above the board, but this time the south pole faced the user.

As a result of this experiment we discovered many similarities with the prior two experiments. Each of the graphs below presents an inverted and 90 degree shifted view of the graphs from Experiment 2. This was a result of rotating the magnet by 90 degrees and flipping its polar orientation.

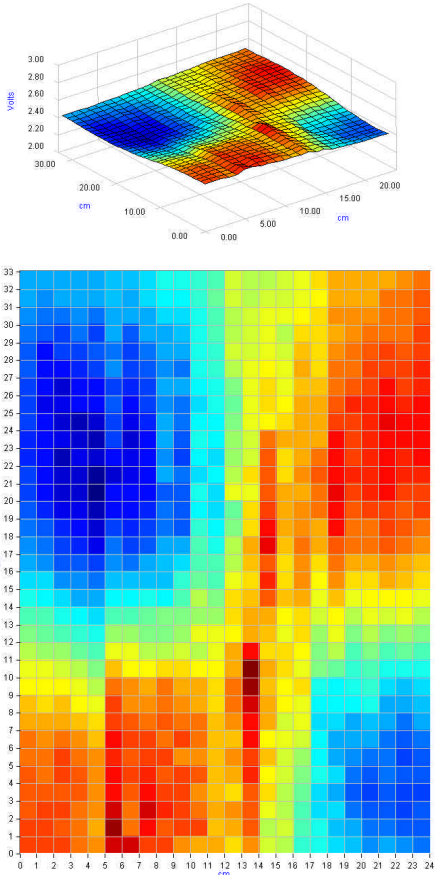


Figure 51 – Experiment 3: Y Sensor

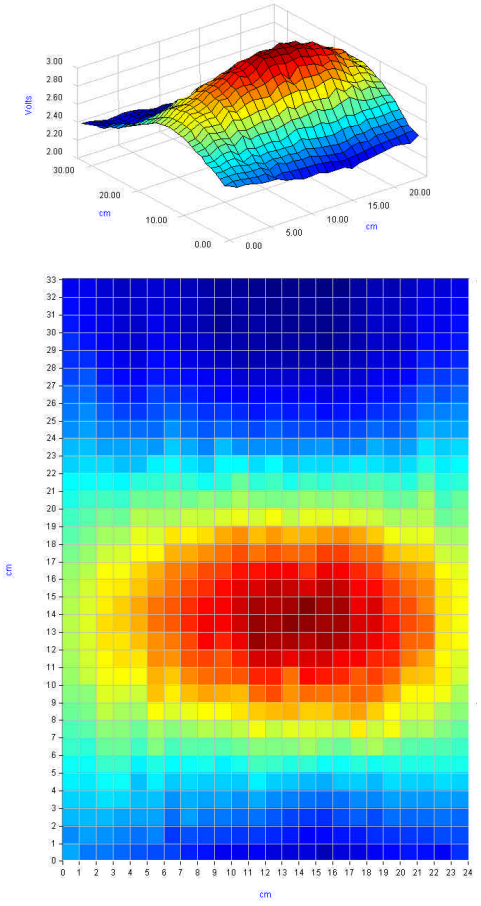


Figure 50 – Experiment 3: X Sensor

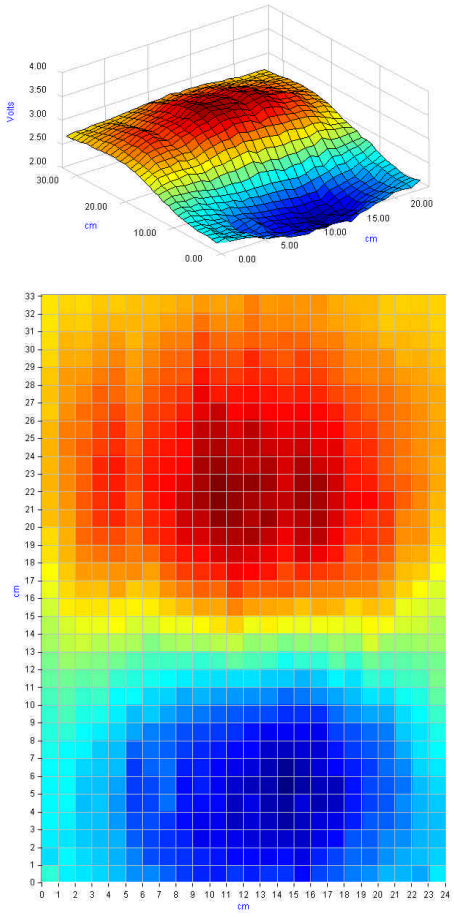


Figure 52 – Experiment 3: Z Sensor

After running these three experiments, we gained a better understanding of our system and its operation. Our results surpassed the quality of information that we expected from our prototype. Only after viewing the three-dimensional graphs of each plane are we definitively able to say that a pattern exists in relation to the information we are collecting. The patterns found in our graphs potentially contain enough information to form best-fit equations and model the magnetic fields in relation to a magnet's position.

5.1 2D Demonstration

Once we collected and interpreted the data, our understanding of the system increased dramatically. We hypothesized that if we performed numerous experiments following the same rules we would receive the same results. With this in mind, we decided to design an experiment that should determine both whether the data is predictable and reproducible and if we currently have the ability to plot an object's position in two dimensions while holding the third dimension constant. The setup required that we first store the data collected in Experiment 1 within an SQL database. We hypothesized that by placing the magnet at a certain location on our grid, and collecting new data for the X, Y and Z sensors, we should be able to find the same point in the database.

We created a database access application, in the scripting language PHP. This application's front end allows the user to view a table representation of the one square centimeter grid on our physical testing box. We gave each square centimeter an ID number within the database and printed the ID number on our testing grid and on the table representation. When running our data collection software in "button test mode," 254 data samples of X, Y, and Z sensor data are collected and the average of each is written to a file, accessible by the PHP script. The script uses these values to query the SQL database and see if a match is found between the data collected in Experiment 1, and the current test. When a point match is found, the square correlating to the physical location of the magnet is shaded in red.

Since even the slightest movement of the magnet changes the output of the sensors, we implemented some tolerance in the search. To allow for this, the script searches the database once for the exact value, which was received from the current button test. If that value is not found, then the search range is expanded by searching for

X values between X+1 and X-1, Y values between Y+1 and Y-1, and Z values between Z+1 and Z-1. The script keeps querying the database until at least one point is returned. An example of a search can be seen in Figure 53, where the point 376 has been found. At the top is the text "Tolerance: 11," which means a query was run between X+11 and X-11, Y+11 and Y+11, and Z+11 and Z-11. The smaller the added and subtracted number, the less searches needed to be performed in order to find that specific point. The fewer searches, the more certain we are of the magnet's position.

Tolerance: 11

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66
67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132
133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165
166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198
199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231
232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264
265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297
298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330
331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363
364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396
397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429
430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462
463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495
496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528
529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561
562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594
595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627
628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660
661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693
694	695	696	697	698	699	700	701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726
727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759
760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792

Figure 53 – Web Page Table Representing Physical Grid

Once this system was operational, we began experimentation. The system was able to locate repeatedly the position of the object within the grid as the specified distance and orientation to the sensors. Within the rectangle defined by grid ID number 69, 696, 724, and 97 the accuracy at this distance was approximately 95%. In the fringes around this rectangle, we found it difficult to lock on to only one point. The database would return multiple hits because the data on the fringes had a higher variability. Nevertheless, one must note that the actual position of the magnet was always returned in the set of possible hits.

5.2 Will The System Be Able to Handle 3D Tracking?

We believe that it is possible to track an object in three dimensions utilizing a system based on our proof of concept. In order to do so, an algorithm needs to be created, possibly taking many variables in to perspective. For this to be possible, the output of our system must be very predictable and repeatable. In addition, it is required by any mathematically based system that a sufficient amount of information be supplied to cover the number of system unknowns. Our system unknowns will include a minimum of the X, Y, and Z location and may include magnet orientation or tilt.

The accuracy of our two dimensional tracking system demonstrated that the output of each sensor, in relation to the magnet's position is predictable and reproducible. In addition, the information to be supplied to the tracking algorithm can be obtained by carefully choosing the position of more sensors and or looking at the data obtained by each sensor not as an independent quantity, but in relation to multiple sensors placed in the same orthogonal direction. We feel confident that our system provides for the major factors required in a 3D position tracking system.

5.3 Limitations and Future Direction

Our system showed remarkable promise towards the application of a three-dimensional computer mouse. We were able to demonstrate a correlation between the position of the target magnet and readings acquired by the magnetic sensors. Our database in the form of a lookup table demonstrated accuracy in the repeatability of locating the magnet in two dimensions when holding the third dimension constant. However, future developers must identify and address limitations if a fully functional 3D mouse is to be developed based our proof of concept.

The first limitation stems from the need for calibrated sensor readings. Regardless of the number of sensors used in a system, they need to be adjusted, so that in the absence of a target object their output sits at 2.5V, the zero level for these sensors. Calibration becomes even more stringent in a system with multiple sensors as they must be at a relative zero level. Without tuning the sensors there would be no method to achieve an accurate position reading. For our proof of concept system, we connected a voltage divider to each of the instrumentation amplifier's negative inputs. By slowly adjusting the potentiometer forming the voltage divider, we were able to adjust each sensor so that

its ambient reading was at the appropriate level. Although this method worked fairly well for our proof of concept, it is unsuitable for an actual consumer level system. Neither a future developer nor we can expect a consumer to have the ability or desire to adjust any number of sensors. Instead, an expanded product should have automatic calibration whereby the system can calibrate itself in the absence of any target object.

A second limitation also arises from a problem associated with user operation. In our proof of concept system, we were able to ensure that no foreign objects beyond the operator's hand and the target entered the system's working area. However, the average consumer might wear metal jewelry or place metallic objects inside the tracking vessel. These effects could potentially be problematic resulting in undesirable readings. Unless the object is stationary, our current design is unable to handle this type of interference. The user must be asked to remove any worn metallic items before operation.

Not all of the possibilities for an expanded system lay in the limitations of our current design. However, many of the features not required by our proof of concept are the next logical stage in the system's future direction. One such feature is the ability to track multiple objects independently in three dimensions. Although the current focus is a three-dimensional mouse requiring only one target, multiple target tracking broadens the potential focus to accompany a wider variety of applications such as an invisible keyboard. As system expansion become promising, the hardware and software complexity increases. Especially, when the tracking system must have the ability to not only locate, but also differentiate between each target. The tracking algorithm developed to track one object's position must be expanded to encompass multiple objects, or the designers have to invent a method to null the effects of each object independently so that the system can function as a single tracking system. In addition, as the system expands a focus should be placed on hardware miniaturization. This becomes crucial when transitioning our prototype into three dimensions causing an increase in hardware components.

Not only is physical size important but the location where the bulk of the processing takes place is significant. In our current system, the three-dimensional graphing and data processing all takes place in our custom-made PC software. Although this is adequate for our proof of concept, the processing did consume valuable system resources and could have been completed on the hardware side. One potential solution would be to implement digital signal processing (DSP), moving the processing from the PC to hardware. Not only will this free up system resources but it could also result in a

much faster and more efficient system allowing designers the ability to link the three-dimensional mouse directly to the computer cursor through the PS/2 port. Moving the entire processing component off the computer completes the last phase in the system development of a three-dimensional computer mouse.

References

Analog Devices, Inc. (2000). Datasheet for Precision Single Supply Instrumentation Amplifier: AMP04. Norwood, MA. pp. 1-16.

Brian, Marshall. (2000). How Computer Mice Work. Retrieved September 24, 2003, from the World Wide Web: http://computer.howstuffworks.com/mouse1.htm

CADsoft. (2003). EAGLE (Version 4.11) [Computer software].

Caruso, J. M., and Withanawasam, S. L. (1999, May). Vehicle Detection and Compass Applications using AMR Magnetic Sensors. Honeywell, SSEC., Plymouth, MN. p. 3.

Case Western Reserve University. (2000). Eddy Current Demo. Retrieved August 15, 2003, from the World Wide Web: http://www.cwru.edu/artsci/phys/courses/demos/eddy.htm

Chabay, R., and Sherwood, B. (1995). Electric & Magnetic Interactions. John Wiley & Sons, Inc., New York: pp. 487-490.

Crossbow Technology, Inc. (2003). Magnetometers. San Jose, CA. pp. 42-43.

Department of Defense and Department of Transportation. (2001). 2001 Federal Radionavigation Plan. pp. 3-3.

----. (2001b). 2001 Federal Radionavigation Systems. pp. 3-6.

Elert, G., and Yusufov, E. (2001). Frequency Used in Navigational Sonar. Retrieved August 24, 2003, from the World Wide Web: http://hypertextbook.com/facts/2001/EmranYusufov.shtml

Energen, Inc. (2003). Magnetic Smart Materials: Magnetostrictor Technology. Billerica, MA. p. 1.

Engelbart, D.C. (1970, November). X-Y Position Indicator for a Display System. United States Patent Office, number 3541541.

Fat Quarters Software & Electronics. (2000). High Sensitivity Gradiometer – Magnetic Anomaly Detection. Retrieved August 18, 2003, from the World Wide Web: http://www.fatquarterssoftware.com/Download/scl007.htm

Freudenrich, C.C. (2003). How Ultrasound Works. Retrieved August 30, 2003, from the World Wide Web: http://electronics.howstuffworks.com/ultrasound1.htm

Hochreiter, J., and Schrottmayer, D. (2003). Digital Magnetic Anomaly Detection System. Wien, Austria. Retrieved August 16, 2003, from the World Wide Web: http://www.iemw.tuwien.ac.at/publication/workshop0600/Hochreiter.html

Honeywell. (2000, April). Datasheet for 1- and 2-Axis Magnetic Sensors: HMC1001/1002 HMC1021/1022. Plymouth, MN. pp. 1-15.

Honeywell. (2002, January). Application Note: Applications of Magnetic Position Sensors. pp. 1-8.

Hsiao, Kai-yuh. (2001, February). Fast Multi-Axis Tracking of Magnetically-Resonant Passive Tags: Methods and Applications. Cambridge, MA. pp. 13-73.

Hsiao, K., and Paradiso, J. (2000, April). Swept RF Tagging Project. Cambridge, MA. Retrieved August 20, 2003, from the World Wide Web: http://www.media.mit.edu/resenv/tags.html

Integrated Publishing. (2003). Quantitative Measurements. Retrieved September 3, 2003, from the World Wide Web: http://www.tpub.com/neets/book21/88.htm

Lynnworth, C.L. (2000). Measurements at the Speed of Ultrasound. The Bent Of Tau Beta Pi Summer 200: 10-13.

Mayfield, Kendra. (2002, May). Radio ID Tags: Beyond Bar Codes. Wired News. p. 1 Retrieved October 16, 2003, from the World Wide Web: http://www.wired.com/news/technology/0,1282,52343,00.html

MCA, Sem Vi. (2003). A.C. and D.C. Bridges. Devi Ahilya Vishwavidyalaya, India. Retrieved September 2, 2003, from the World Wide Web: http://www.davv.ac.in/lectures/elex6/Bridge.htm

Microchip Technology, Inc. (2001). Datasheet for PIC16F87X: 28/40-Pin 8-Bit CMOS Flash. San Jose, CA. pp. 1-218.

Mitchell, Darryl. (February, 2003). NASA Goddard Space Flight Center: Capaciflector Technology. Retrieved October 16, 2003, from the World Wide Web: http://capaciflector.gsfc.nasa.gov/description.html

Phywe Systems. (2003). RCL Measuring Bridge. Göttingen, Germany. pp. 1-3.

Protovale Oxford Ltd. (2002). Pulse Induction Metal Detectors. Retrieved August 18, 2003, from the World Wide Web: http://users.argonet.co.uk/users/protovale/technical-papers/TechnicalNotes/Pulse_Induction_Metal_Detectors.htm

Smaby, Niels. (2003). The "Capaciflector" Proximity Sensor. Retrieved August 20, 2003, from the World Wide Web: http://www-cdr.stanford.edu/touch/previous_projects/capaciflector/capaciflector.html

Spohn, Daniel. (2003). Inductive Sensing for Velocity Measurement at a U.S. Air Force Laboratory. Retrieved August 18, 2003, from the World Wide Web: http://www.sensorsmag.com/articles/0898/in0898/main.shtml

Stanford University. (2003). Doug Engelbart 1968 Demo. Retrieved September 24, 2003, from the World Wide Web: http://sloan.stanford.edu/mousesite/1968Demo.html

Suffi, Guerrino. (2001, February). Proximity Sensor Implementation. Retrieved August 15, 2003, from the World Wide Web: http://www.manufacturing.net/ctl/index.asp?layout=article&articleid=CA188295

Wacom Components. (2002). Wacom Components – Technology. Retrieved August 22, 2003, from the World Wide Web: http://www.wacom-components.com/english/tech.asp

Ward, A., Steggles, P., Curwen, R., & Webster, P. (2001). Sentient Computing Project. Cambridge University, Cambridge England. Retrieved, August 28, 2003, from the World Wide Web: http://www.uk.research.att.com/spirit/

Weisz, R. (December, 1996). First Aid for the Thread-Impaired: Using Multiple Threads with MFC. Microsoft Systems Journal. Los Gatos, CA.

Welsby, Scott, D. (2003, March). Capacitive and Inductive Noncontact Measurement. Sensor Technology and Design.

Appendix A – Microcontroller PIC Code

Contained below is the PIC microcontroller code written for the controller board as discussed in prior sections.

PIC Microcontroller Source Code

```
;/*****************************************************************
;This code cannot be copied, modified, nor used without giving credit to the original authors.
;The authors give no warranty as to the workings of this program.
;Also the text included here must be left with every complete or partial code replication.
;
;Authors:
;     Andrea L.M. Baker, Wojciech Krajewski and Daniel I. Wallance
;Completion Date:
;     Saturday October 18, 2003.
;Designed For:
;     This code was written for AMT Ireland at the University of Limerick, Ireland during the fulfillment
;     of a Major Qualifying Project for Worcester Polytechnic Institute, Worcester, MA, United States.
;*****************************************************************/



      list p=16f877
      include p16f877.inc

      ; below are the confuguration bits for this program.
      __CONFIG _HS_OSC & _CP_OFF & _WDT_OFF & _PWRTE_ON & _BODEN_OFF & _LVP_OFF & _CPD_OFF & _WRT_ENABLE_OFF &
_DEBUG_OFF


; ********************************************************
; equates
Bank0Ram    equ   0x20  ; start of Bank 0 RAM area

; ********************************************************
```

```
; variables

      cblock      Bank0Ram
      Temp

      SerialTestData_length
      SerialBuffer

      ADCDefaults
      ADCPin
      ADCChecksum

      ResetSensorInterrupt

      DebounceTime
      DebounceTemp
      PacketCounter

      endc

; ************************************************************
; start at the reset vector

      org   0x000
      nop                         ; required for debugger

Start
      ; Clear up Variables:
      CLRF  ADCChecksum
      CLRF  Temp
      CLRF  SerialBuffer
      CLRF  RCREG
      CLRF  ADCPin
      CLRF  ResetSensorInterrupt
      CLRF  DebounceTemp
      MOVLW 0x07
      MOVWF DebounceTime
```

```
      ; Initialize hardware:
      Call  ConfigureResetPins
      Call  ConfigureSerialSend
      Call  ConfigureADC
      Call  ConfigurePortBInput

Main
      btfss       PIR1, RCIF                    ; Has recieve flag been set?
      goto  Main                                ; No commands recieved
      movf  RCREG,W                             ; If we got serial data, move it to SerialBuffer
      movwf SerialBuffer

      movlw 0x41                                ; Echo back letter A
      XORWF SerialBuffer,W
      BTFSC STATUS,Z
      CALL  SerialEchoBack

      movlw 0x42                                ; Echo "B" then send data
      XORWF SerialBuffer,W
      BTFSC STATUS,Z
      CALL  ADCPacketControlLoop


      movlw 0x43                                ; Echo "C" then send single data
      XORWF SerialBuffer,W
      BTFSC STATUS,Z
      CALL  ButtonPressControlLoop


      goto  Main
; ********************************************************
; subroutines
; ********************************************************
```

```
ButtonPressControlLoop
      BSF         PORTC,3                     ; Indicator LED goes GREEN.
      MOVF  SerialBuffer,W
      CALL  SerialEchoBack
      ButtonPressControlLoop_REPEAT
      BTFSc       PIR1, RCIF                  ; No new serial data waiting
      GOTO  ENDButtonPressControlLoop
      BTFSc PORTB,0                           ; Check button press
      Call  SendFakeADCData                   ; No button press, Send fake data
      BTFSc PORTB,0                           ; Check button press
      goto  ButtonPressControlLoop_REPEAT     ; No button press
      ; Button Pressed, Debounce

      ; Read debounce time from DIP Switches
      MOVLW 0xF0              ; MASK OUT MSB
      ANDWF PORTB,W           ; PLACE MASK IN W
      IORWF DebounceTime,W    ; PUT BACK IN PACKETCOUNTER
      MOVWF DebounceTemp

      Debouncing                                    ;run a bunch of times
      Call  SendFakeADCData
      DECFSZ      DebounceTemp
      goto  Debouncing

      BTFSc PORTB,0                               ; Check button press
      goto  ButtonPressControlLoop_REPEAT         ; No button press
      ; Button Still Pressed, Send 255 Packets

      BCF         PORTC,3                         ; INDICATOR LED goes RED
      MOVLW d'254'
      MOVWF PacketCounter
      Send255Packets
      MOVLW 0X00
      MOVWF ADCPin
      Call  ResetMagneticSensors
      CALL  ADCSendPacket
      DECFSZ      PacketCounter
      GOTO  Send255Packets
```

```
      ; Wait for button to be released.
      ReleaseButton
      Call  SendFakeADCData
      BTFSs PORTB,0                                 ; Check button press
      goto  ReleaseButton                           ; Button pressed
      ; Button Released, Wait for another.

      BSF         PORTC,3                           ; Indicator LED goes GREEN
      BTFSS       PIR1, RCIF                        ; No new serial data waiting
      GOTO  ButtonPressControlLoop_REPEAT
      ENDButtonPressControlLoop
      BCF         PORTC,3                           ; Indicator LED goes RED.
      return

      ; PortB [0,1,2,3] is the switches we can use
      ; [0] is the button to be read,
      ; [1] cannot be used, button plugs in here
      ; [2] Unimplemented
      ; [3] Unimplemented
      ; PortB [4,5,6,7] is MSB of Debounce counter


ADCPacketControlLoop
      Call  ResetMagneticSensors
      BSF         PORTC,3                           ; Indicator LED goes green.
      MOVF  SerialBuffer,W
      CALL  SerialEchoBack
      ADCPacketControlLoop_REPEAT
      MOVLW 0X00
      MOVWF ADCPin
      CALL  ADCSendPacket
      ; add code for more sensors here.
      Call  ResetMagneticSensors
      BTFSS       PIR1, RCIF                        ; Has recieve flag been set?
      GOTO  ADCPacketControlLoop_REPEAT
      BCF         PORTC,3
      return
```

```
SendFakeADCData
      ;BCF        PORTC,3                          ; INDICATOR LED
      MOVLW 0X00              ; HEAD
      Call  SerialSendData
      MOVLW 0X00              ;X
      Call  SerialSendData
      MOVLW 0X00              ;X
      Call  SerialSendData
      MOVLW 0X00              ;Y
      Call  SerialSendData
      MOVLW 0X00              ;Y
      Call  SerialSendData
      MOVLW 0X00              ;Z
      Call  SerialSendData
      MOVLW 0X00              ;Z
      Call  SerialSendData
      MOVLW 0X00              ; TAIL
      Call  SerialSendData
      ;BSF        PORTC,3                          ; INDICATOR LED
      return

ADCSendPacket
      ;BTFSC      PORTB,0
      ;GOTO SendFakeADCData
      ; Go instead of call so that we return to the
      ; ADC control program, not back here.

      ; Congigure which pins are inputs
      Call  ADCSetInputPin
      ; Clean up Checksum
      CLRF  ADCChecksum
      ; Send head packet
      MOVF  ADCPin,W

      Call  SerialSendData                           ; Send the head packet
```

```
; FIRST Pin in Packet
BSF         ADCON0,GO                       ; start conversion
ADCWait0
BTFSC ADCON0,GO
GOTO  ADCWait0                              ; ADC Not done
; Conversion done, prepare the next pin
INCF  ADCPin,F
;MOVF ADCPin,W
Call  ADCSetInputPin
; Send current pin
MOVF  ADRESH,W
ADDWF ADCChecksum,F                         ; Start checksum
Call  SerialSendData                        ; Send MSB
BSF         STATUS,RP0                      ; LSB is in Bank1
MOVF  ADRESL,W
BCF         STATUS,RP0                      ; Back to Bank0
ADDWF ADCChecksum,F                         ; Add up Checksum
Call  SerialSendData                        ; Send LSB

; SECOND Pin in Packet
BSF         ADCON0,GO                       ; start conversion
ADCWait1
BTFSC ADCON0,GO
GOTO  ADCWait1                              ; ADC Not done
; Conversion done, prepare the next pin
INCF  ADCPin,F
;MOVF ADCPin,W
Call  ADCSetInputPin
; Send current pin
MOVF  ADRESH,W
ADDWF ADCChecksum,F                         ; Start checksum
Call  SerialSendData                        ; Send MSB
BSF         STATUS,RP0                      ; MSB is in Bank1
MOVF  ADRESL,W
BCF         STATUS,RP0                      ; Back to Bank0
ADDWF ADCChecksum,F                         ; Add up Checksum
Call  SerialSendData                        ; Send LSB
```

```
      ; THIRD Pin in Packet
      BSF         ADCON0,GO                         ; start conversion
      ADCWait2
      BTFSC ADCON0,GO
      GOTO  ADCWait2                                ; ADC Not done
      ; Conversion done, prepare the next pin
      INCF  ADCPin,F
      ;MOVF ADCPin,W
      Call  ADCSetInputPin
      ; Send current pin
      MOVF  ADRESH,W
      ADDWF ADCChecksum,F                           ; Start checksum
      Call  SerialSendData                          ; Send MSB
      BSF         STATUS,RP0                        ; MSB is in Bank1
      MOVF  ADRESL,W
      BCF         STATUS,RP0                        ; Back to Bank0
      Call  SerialSendData                          ; send last bit on it's meerry sleepy way.
      ADDWF ADCChecksum,W                           ; Add up Checksum
      Call  SerialSendData                          ; checksum
      return

ResetMagneticSensors
      ; set sensors
      BSF         PORTC,0
      BSF         PORTC,1
      BSF         PORTC,2
      ; wait....
      MOVLW 0xFF                                    ; @ 18.432Mhz, 0x46=15uSec
      MOVWF Temp
      ResetMagneticSensors_DELAY
      DECFSZ      Temp,F
      GOTO  ResetMagneticSensors_DELAY
      BCF         PORTC,0
      BCF         PORTC,1
      BCF         PORTC,2
      return

SerialEchoBack
```

```
      MOVF  SerialBuffer,W
      Call  SerialSendData
      return

ConfigureADC
      ;Startup Code to Enable AD
      MOVLW       b'01000001'                   ; FOSC/32=10, Channel000, GO=0, unimp=0, ADC on=1
      MOVWF ADCON0
      MOVWF ADCDefaults                         ; Later used for changing input pins.
      BSF     STATUS,RP0
      MOVLW b'10000000'                         ; RightJustified=1, Unimp=000, All Inputs Analog=0000
      MOVWF ADCON1
      bcf     STATUS,RP0
      return

ADCSetInputPin
      MOVF  ADCPin,W
      MOVWF Temp
      RLF         Temp
      RLF         Temp
      RLF         Temp
      MOVF  Temp,W
      IORWF ADCDefaults,W
      MOVWF ADCON0
      return

SerialSendData
      SerialSendData_NotClear
      BTFSS       PIR1,4
      GOTO  SerialSendData_NotClear
      MOVWF       TXREG
      return

ConfigureSerialSend
      MOVLW b'10010000'                         ;enable1, 8bit0, 0, enable1, Disable0, No0, No0, 0
      MOVWF RCSTA
      BSF   STATUS,RP0                          ; Bank 1
      BCF   TRISC,6                             ; SET PORTC IN/OUTS
```

```
      BSF TRISC,7
      ; Configure Serial Port Speeds
      MOVLW d'29'                                ; 38.4kKbps @18.432mhz
      MOVWF SPBRG
      MOVLW       b'00100100'                    ; 0,8bit,xmit on, asynch, 0, high speed, 0, 0
      MOVWF       TXSTA
      BCF   STATUS,RP0                           ; Bank 00
      BCF   STATUS,RP1
      return


 ConfigureResetPins
      ; Reset pins on PortC[0,1,2]
      ; Indicator LED on Portc[3]
      BCF         PORTC,0
      BCF         PORTC,1
      BCF         PORTC,2
      BCF         PORTC,3                        ; For indicator LED
      BSF         STATUS,RP0                     ; Go to bank 1
      BCF         TRISC,0
      BCF         TRISC,1
      BCF         TRISC,2
      BCF         TRISC,3                        ; For indicator LED
      BCF         STATUS,RP0
      RETURN

ConfigurePortBInput
      MOVLW 0XFF
      MOVWF PORTB                                ; PORTB PULLUPS HIGH
      BSF   STATUS,RP0                           ; BANK1

      MOVWF TRISB                                ; PORTB INPUT
      BCF   OPTION_REG,NOT_RBPU                  ; PORTB PULLUPS ON
      BCF   STATUS,RP0                           ; BANK0
      return

      End
```

Appendix B – Visual C++ PC Software

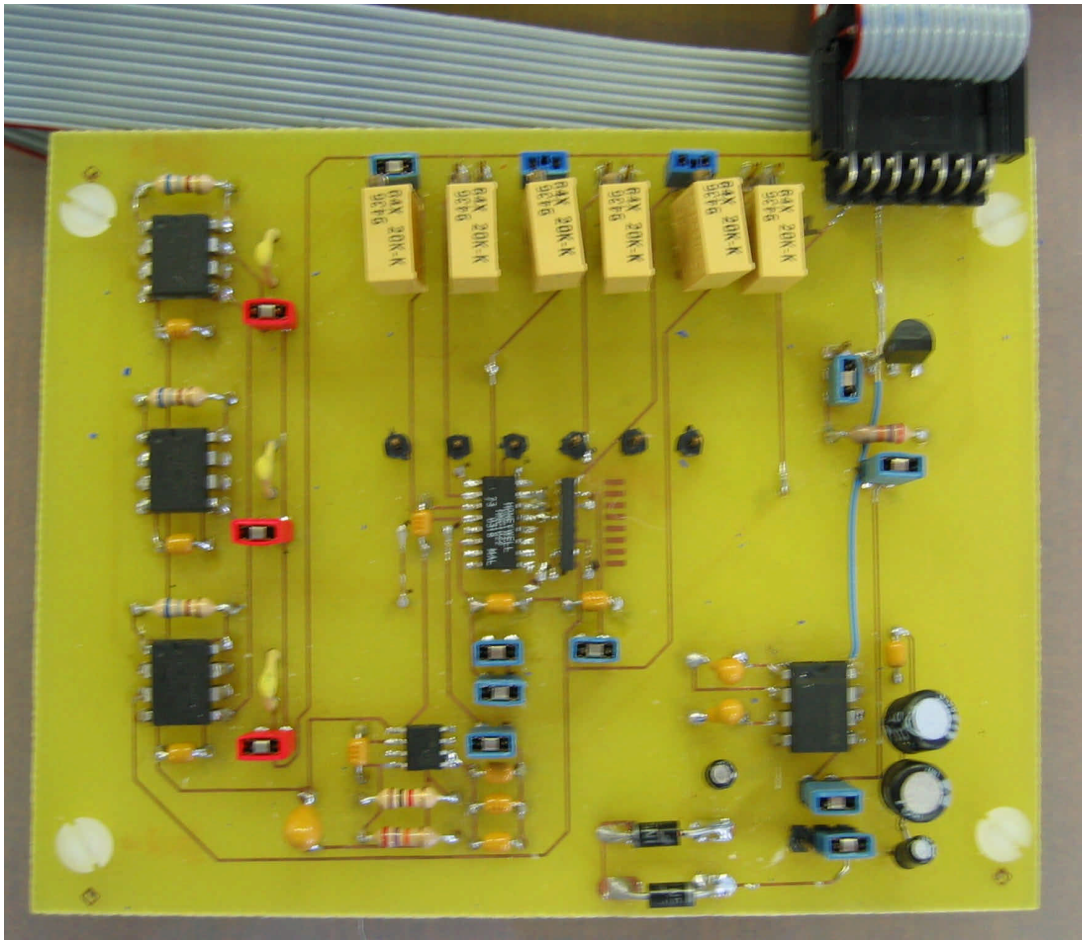
The PC code used for the system's data collection and real time display is contained in this Appendix. However, due to the code length, it can be found at the end of this document with separate page numbering.

Appendix C – System Images

What follows are digital images showing the two circuit boards for our prototype along with the experimental setup discussed in prior sections.

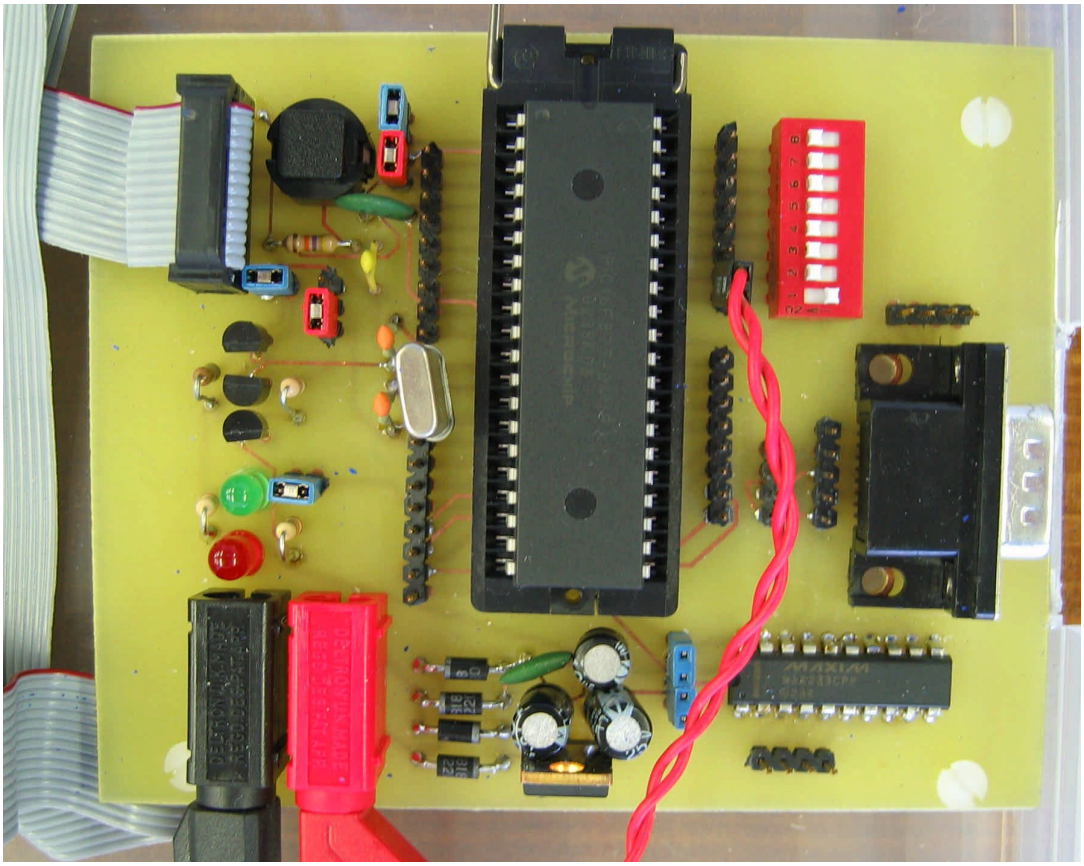
Magnetic Sensor Board

Below is a photograph of the magnetic sensor board. The X, Y and Z sensors are the surface mount components in the center.



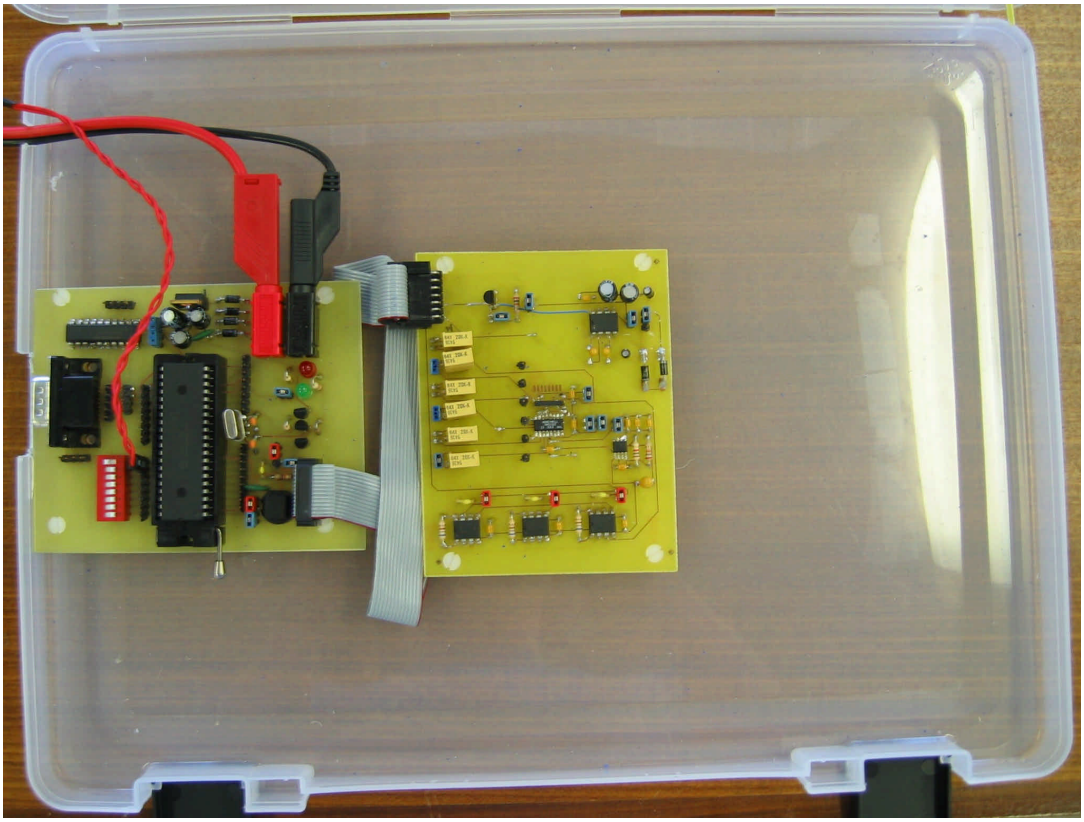
PIC Microcontroller Board

Below is a photograph of the board containing the PIC microcontroller. The cable on the right is for a pushbutton used during “button test mode.”



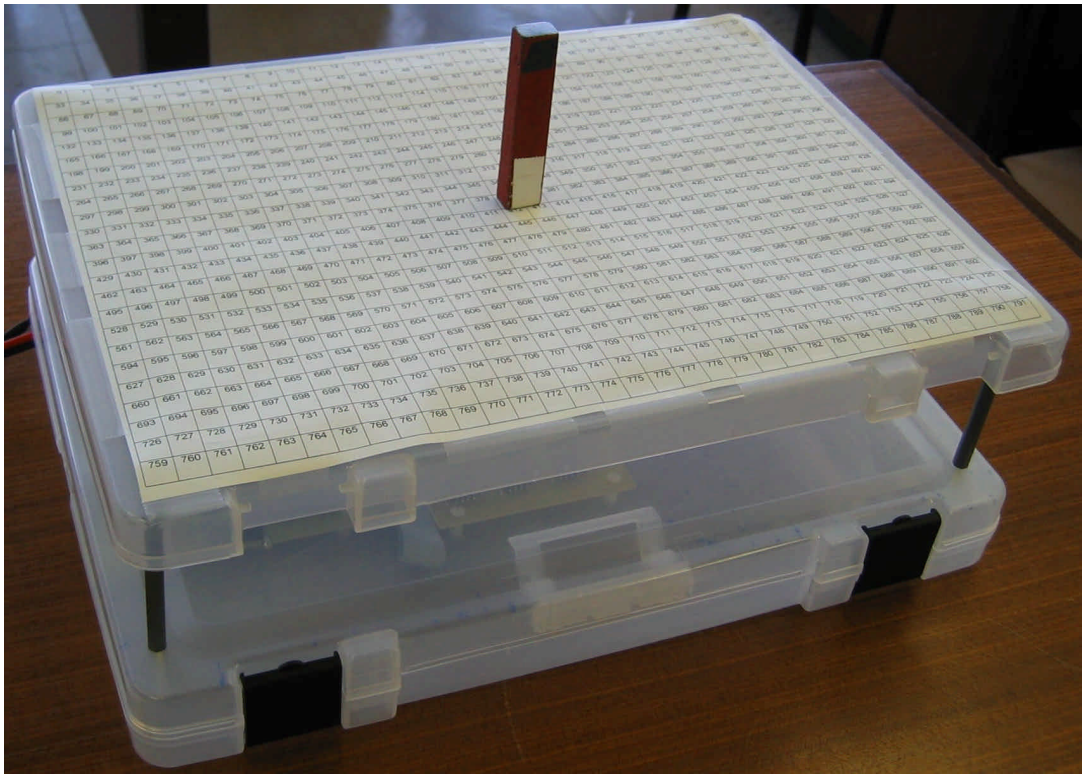
Inside of Experiment Box

Below is a photograph of the sensor and control boards mounted in the box used in our experiment setup.



Experimental Setup

Below is a photograph of the assembly we used for our experiments to map the magnetic field of the displayed magnet using the magnetoresistive sensors.



Appendix D – Project Focus Proposals

From our initial background research in small-scale position tracking technology, we settled on three possible methods for our proof of concept system. We had the option of spreading our resources across all three or focusing our efforts on one. What follows are twoproposals we developed around September 1, 2003 to establish our project's focus.

Exploring a Single Promising Technology for 3D Tracking

AMT Ireland presented us with the task to investigate the possibility of developing a three dimensional, small space, position tracking system, which will cumulate in the eventual development of a commercially viable 3D mouse. Our role is the preliminary development stage of the design, whereby we will scrutinize possible implementation methods and adapt one method to demonstrate application feasibility.

Our MQP will be a thorough design review of one solution that we will select from a list of promising solutions complied over the past three weeks of preliminary research. This single technology will have significant potential in meeting AMT's requirements for a three dimensional mouse. Since our project is entirely based on this solution, we have identified several advantages and disadvantages to choosing this design path along with the final deliverables.

Pro:

- Focusing on one solution allows us to proceed to a greater depth with research and design, permitting for a more complete characterization of this solution.
- We will be able to deliver a demonstration of feasibility (through a partially functional prototype).
- Since we can work toward one common goal we will have the opportunity and time to mix both research and prototype design (we find this to be a very exciting prospect).

Con:

- Since we are in the earliest stage in the development of a 3D mouse, it may turn out that the chosen solution is not well suited for this application.

Deliverables:

- A significant amount of research in the field of electromagnetics, since currently all viable solutions lie in this realm.
- A report detailing both the designed and implemented systems.
- Test circuits used for understanding and developing the solution.
- A partial or completed prototype for tracking movement in one or more dimensions.

Comparing Two Technologies for 3D Tracking

AMT Ireland presented us with the task to investigate the possibility of developing a three dimensional, small space, position tracking system, which will cumulate in the eventual development of a commercially viable 3D mouse. Our role is the preliminary development stage of the design, whereby we will scrutinize possible implementation methods and then choose the most promising two and attempt to demonstrate the application feasibility of each.

Our MQP will be an introductory design review of two solutions we will select from a list of promising technologies complied over the past three weeks of preliminary research. These technologies will have a significant potential in meeting AMT's requirements for a three dimensional mouse. We identified several advantages and disadvantages to choosing this design path along with the final deliverables.

Pro:

- There is a good chance that at least one technology will be suited for the development of a 3D mouse.
- Our research will be greatly overlapped since all feasible solutions at the moment are based on electromagnetism.

Con:

- We do not feel that a strong hardware design component will result from this proposal because our resources will be spread over a large field of study.
- We will only scratch the surface of development by working on two solutions.
- Concentrating mostly on theoretical research for the duration of the project makes it difficult for us to stay on track. (We would like hardware design and implementation to be a key component of our MQP.)

Deliverables:

- A significant amount of superficial research on both technologies.
- Extensive comparisons along with positive and negative benefits of each.
- A detailed description of each technology's suitability for future product development
- Proof of concept circuits and small test rigs.

Appendix E – Weekly Progress Reports

For this project, we were required to submit progress reports on a weekly basis to our advisor. Each report provides a comparison of what we set out to achieve and what tasks we actually performed in a given week. What follows are the progress reports submitted on the Friday of each week between August 10, 2003 and October 11, 2003.

Week 1 Sunday August 10, 2003 – Saturday August 16, 2003

Monday	Learned about the problem.
Tuesday	Discovered the possibility of using EM sensors.
Wednesday	Discovered prepackaged EM sensors.
Thursday	Discovered existing problem solution.
Friday	Summarized and focused this week's research upon a simplified list of possibilities.

Week 2 Sunday August 17, 2003 – Saturday August 23, 2003

Week's goals:

1) Come up with a list of magnetic sensors to order.
2) Develop a list of possible applications for a magnetic pea.
3) Compile a list of pros and cons for developing 3D sensing based upon magnetic sensors opposed to LC circuits.
4) Present results of the above findings to John Harris and make a final decision on which method to implement.
5) Deliver a detailed goal statement based on the chosen application.

Planned schedule:

Monday	Begin detailed pro/con analysis of simplified possibility list. Order test components.
Tuesday	Continued.
Wednesday	Summarize pro/con analysis. Begin detailed goal statement and schedule
Thursday	Eliminate non-feasible solutions. Begin basic block diagram of viable solution(s).
Friday	Deliver detailed goal statement, basic block diagram of chosen solution(s), detailed schedule for the next 3 weeks and basic schedule highlighting key events for the entire project.

Actual schedule:

Monday	Researched components to order.
Tuesday	Compiled a list of desired components and submitted it to John.
Wednesday	Developed a sensor coil and target coil for proof of concept demonstration. Began report outline.
Thursday	Researched electromagnetic fields and continued work on coils.
Friday	Narrowed implementation list and developed block diagrams for three remaining possibilities.

Week 3 Sunday August 24, 2003 – Saturday August 30, 2003

Planned schedule:

Monday	Design and build magnetic sensor test rig.
Tuesday	Continue.
Wednesday	Describe experiments performed with magnetic sensors and what can and

cannot be done with this technology.

Thursday Generate a final analysis of technologies available, and choose the most feasible option. Develop a detailed block diagram of this option. Specify the major design and implementation milestones.

Friday Hand in the detailed prototype requirements, a decision on which technology will be used to solve the requirements, block diagrams of the chosen system and further pro/con documentation.
In relation to our implementation decision, we will hand in a description of the major milestones that need to be completed to deliver a fully functional prototype. We will also submit a first cut decision of what milestones will be completed for our MQP.

Actual schedule:

Monday Tried to fit SOIC package to adapter. Reviewed sensor's datasheet for possible test circuit setups and for a better understanding of the chip. Researched current technologies.

Tuesday Developed printed circuit board to mount SOIC magnetic sensor. Created an LC circuit to detect change in transmitting coil at the LC's resonant frequency. Researched current technologies.

Wednesday Created a wheatstone bridge in an attempt to detect successfully an LC circuit. Mounted magnetic sensor to PCB. Researched current technologies.

Thursday Further researched existing technology to determine the problem with our wheatstone bridge. Began building test circuitry for the magnetic sensor. Found MIT Thesis and read for comparison.

Friday Discussed with John Harris the future direction of our project. We continued to work on testing the magnetic field sensor and documented our week's progress. Continued building test circuitry for the magnetic sensor.

Week 4 Sunday August 31, 2003 – Saturday September 6, 2003

Week's goals:

1) Determine direction for MQP. Summarize current research thru documentation.

Planned schedule:

Monday Complete MQP proposals. Continue building magnetic sensor test circuitry. Write up research to date.

Tuesday Complete magnetic sensor test circuitry. Continue writing up research.

Wednesday Compare findings from magnetic test circuit, if promising we will brain storm possible solutions using the sensors.

Thursday Choose a proposal for the direction of our MQP. Use this day to re-group on the direction of the project depending upon outcome of the magnetic sensor tests. Also, work further on design and calibration of an inductive wheatstone bridge, which includes LC detection circuitry.

Friday Document our findings thus far. (We want to have a clean slate to start from since we will have just determined our MQP focus).

Actual schedule:

Mon-Fri This week went exactly as scheduled day by day. We are still working on summarizing our finding from the week. Additionally, we designed the inductive Wheatstone bridge on Monday and Tuesday. We used

Wednesday to build the system with its completion on Thursday.

Week 5 Sunday September 7, 2003 – Saturday September 13, 2003

Planned schedule:

Monday Continue background writing. Start developing block diagrams for the magnetic sensor solution. Order parts needed to run a sensitive and complete X,Y,Z magnetic sensor. Continue editing the full report.

Tuesday Complete background writing to date. Continue system diagram, brainstorming and testing. Begin dissemination of hardware, software and research tasks so that we can work further in parallel.

Wednesday Design hardware tests needed for further understanding of our sensor interface. Begin building a complete X,Y,Z test rig. Continue editing the full report.

Thursday Continue building the X,Y,Z test rig. Continue editing the report. Research system architecture such as PC interface and display.

Friday Deliver "flowing" first draft report, including a first draft introduction. Organize our thoughts upon the system block diagram. Complete a timeline for the project. (If progress upon the X,Y,Z sensor continues to go smoothly then we can decide the major hardware implementation milestones.)

Actual schedule:

Monday Report: Editing.
System Implementation: Completed system block diagram and schematic.
Parts: Began compiling parts list.
PC Software: Began researching how to access the serial port and how to display the pea's position.

Tuesday Report: Editing.
Parts: Submitted parts list.
PC Software: Began looking into LabView versus Visual C++ for PC software implementation. Also continued serial port and display research.
PIC Software: Began research into how to send serial data and setup the ADCs.
PIC to PC Communication: Decided upon the data packet structure.

Wednesday (Visited with Professor Demetry)
Report: Editing & additions.
Parts: (Waiting for word on availability of 3axis magnetic sensors).
PC Software: Obtained LabView and determined that it is possible to collect serial data, process it and then display entirely it in LabView. Started Visual C++ Serial Port Code to determine ease of programming as a comparison to LabView. Continued serial port and display research.
PIC Software: Began coding the serial and the ADC system while continuing to research serial data and setup the ADCs research.

Thursday Report: Editing & additions.
Parts: (Waiting for word on availability of 3-axis magnetic sensors).
PC Software: Completed functional serial port testing program in Visual C++. Began block diagram of program structure.
PIC Software: Continued coding the serial system and the ADC system. Completed block diagram of program layout.

Friday Report: Editing & additions.
Parts: (Received word that the 3-axis sensors will take 14 weeks to arrive

so we ordered a 2-axis and a 1-axis sensor that would be available within the next week.)
PC Software: Completed block diagram of program layout.
PIC Software: Completed block diagram of program layout.
PIC hardware: Started mounting the PIC and RS232 chip to a board so that we could test our ability to transfer serially data between the PIC and the PC.

Week 6 Sunday September 14, 2003 – Saturday September 20, 2003

Week's goals:

1) We are still waiting for components…so in the mean time we want to complete the analog to digital conversion, data transmission, data receiving portion of our hardware and software so that when our magnetic sensors come in we can fully concentrate on them. In addition, we want to have a solid first draft of our introduction section.

Planned schedule:

Monday Meet to assign tasks and solidify week's goals.
Report: Continue editing and revisions.
PC Software: Begin laying out and coding program. Continue research into how to display the position of the target.
PIC Software: Continue serial and ADC code.
PIC hardware: Complete hardware build.

Tuesday Report: Begin Introduction. Continue editing and additions.
PC Software: Continue coding serial data acquisition and GUI portion of program. Continue research into how to display the position of the target.
PIC Software: Continue serial and ADC development and begin testing.
PIC hardware: Complete hardware build.

Wednesday Report: Continue Introduction. Completion of section integration.
PC Software: Complete coding serial data acquisition and GUI portion of program.
PIC Software: Begin coding prototype LCD and button control.
PIC to PC Communication: Begin testing communication ability

Thursday Report: Write up system findings so far.
PC Software: Fine-tune the serial data portion. Begin coding display portion of the program.
PIC Software: Continue coding the prototype LCD and button control.
PIC to PC Communication: Continue testing communication ability.

Friday Report: Write up system findings to date.
PIC to PC Communication: Deliver a fully functional communication system.
PIC Software: Deliver a fully function PIC code.
PC Software: Deliver fully functional serial communication system (read/write capabilities) and Data storage system. The display portion will be outlined and partially complete.

Actual schedule:

Monday Project Overview: Discussed and assigned week's goals.
Report: Some editing was done.
PC Software: Research into how to graph data using Visual C++.
PIC Software: Started writing code for everything. There was no computer

	available to connect a programmer, so code could not be tested.
	PIC hardware**:** PIC, LCD & serial chip were put on a board. Ready to run.
Tuesday	Report: Minor editing in free time.
	PC Software: Continued researching 3D graphing in C++, start coding.
	PIC Software: Continued developing transmission of serial data.
	PCB Layout: PCB layout was started.
Wednesday	Report: Minor editing in free time.
	PC Software: All night coding of serial data acquisition functions.
	PIC Software: Continued all night coding of serial transmission.
	PCB Layout: Continue PCB & Circuit layout.
Thursday	Report: Minor editing in free time.
	PC Software: Fine-tune the serial data portion.
	PIC Software: Success in transmitting and receiving serial data. Test packets created, LCD debugging code in place.
	PIC to PC Communication: Tested handshaking between systems.
	PCB Layout: Continue PCB & Circuit layout.
Friday	Report: Minor editing in free time.
	PIC to PC Communication: Debugging data reception on PC end.
	PIC Software: Writing software for ADC control.
	PC Software: Tweaked receiving data from the PIC with partially successful data logging.

Week 7 Sunday September 21, 2003 – Saturday September 27, 2003

Week's goals:

1) Have an unpopulated printed circuit board for the sensors and associate d circuitry.
2) Have a completely functional PIC including necessary A/D conversion.
3) Enable PC software to communicate with the PIC in real time.
4) Have a completed second draft of the introduction.
5) Ensure that our sponsors are up to date with our development and progress beyond our preliminary conversation.

Planned schedule:

Monday	PCB: Finalize PCB layout in design application.
	Report: Continue work on the Introduction and edit report sections to date.
Tuesday	PCB: Complete a final check of the PCB.
	PC & PIC Software: Debug timing issues between the PC and PIC.
	Report: Start work on documenting the software system.
Wednesday	PCB: Print the board.
	PIC Software: Start A/D development.
	PC Software: Continue to work on PC code.
	Report: Continue to edit report and document new developments.
Thursday	PCB: Begin to populate the PCB and test populated sections.
	PIC Software: Finalize A/D conversion system.
	PC Software: Continue to work on PC code.
	Report: Complete the second draft of the Introduction.
	Sponsor Update: Give a presentation to sponsors of our work to date.
Friday	PCB: Continue to populate the PCB and test populated sections.
	PC & PIC Software: Finalize PC and PIC communication systems.
	Report: Finish editing all existing report sections from the start of the week.

Actual schedule:

Monday PCB: Finalized PCB layout in design application.
Report: Continue work on the Introduction and edit report sections to date.

Tuesday PCB: We conducted a final design review to double check the schematic and board layout. In addition, we created final schematic.
PC Software: Began reformatting the entire program since data collection in real time was unreliable using the current architecture. Further researched position display possibilities and coded a basic algorithm to solidify how we will be displaying the position in real time on the PC.
PIC Software: Continued writing and debugging the A/D code.

Wednesday PCB: Spun the board.
PIC & PC Software: Completed the PC program reformatting. Completed A/D code development. Solidified communications between the PIC and the PC in Real Time.
Report: Continue working on the Introduction.
Sponsor update: Informally spoke with John and Seamus about the status of the product, they seem extremely excited.

Thursday PCB: Drilled vias on the board and solidified a testing plan.
PC Software: Began cleaning up the functional program.
Report: Completed the second draft of the Introduction.

Friday PCB: Began to populate and test sections.
PIC & PC Software: Documented the program flow with flowcharts and basic explanations.
Report: Completed the Introduction, and edited existing portions of the report.

Week 8 Sunday September 28, 2003 – Saturday October 4, 2003

Week's goals:

1) Algorithm: Begin developing the algorithm that maps the data received from the three sensors to the actual position of the pea.
2) PCB: Have a completely populated PCB.
3) PC Software: Create and test the display portion of the code.
4) System Integration: Begin to design tests to be performed to determine feasibility.
5) Presentation: Begin creating the presentation.
6) Report: Finalize introduction and first draft of the Executive Summary. Document hardware, and software to-date.

Planned schedule:

Sunday PC Software: Continue cleaning up the existing program.
Report: Begin documentation of the hardware, and software completed to-date.

Monday PCB: Continue populating and testing each section.
PC Software: Finish cleaning up the existing program.
Report: Continue documentation.

Tuesday PCB: Continue populating and testing each section.
PC Software: Begin writing the final display code.
Report: Begin documentation of the hardware, and software completed to-date.
Presentation: Begin brainstorming basic presentation layout.

Wednesday PCB: Continue populating and testing each section.
PC Software: Continue writing and testing the final display code.

Report: Continue documentation.

Thursday PCB: Finish populating the board.
System Integration: Connect the output of the PCB to the PIC's and PC to write raw data files. Debug this connection.

Friday PCB: Continue populating and testing each section.
PC Software: Complete the PC software.
Report: Continue documentation.
System Integration: Continue debugging the complete system.
Algorithm: Begin looking at the raw data to select an approach to the algorithm.

Saturday PCB: Continue populating and testing each section.
PC Software & Algorithm: Begin work together to select an algorithm, write code to implement it.
System Integration: Continue debugging the complete system.

Actual schedule:

Sunday Report: Finished editing all sections to date. Began writing up PC software.

Monday PCB: Began populating, and testing the DC to DC power supply.
Report: Continued editing the Introduction. Continued documenting the PC software.
PC Software: Continued working on real-time display code.

Tuesday Report: Began brainstorming the Executive Summary, made changes to the Introduction based on Prof. Vaz's suggestions.
PCB: Continued testing and building and soldered sections.
PC Software: Continued working on real-time display code.

Wednesday Report: Revised the Introduction once again for better flow, began documenting the PCB section; placed all references in their proper format.
PC Software: Major breakthrough with the display code.

Thursday Report: Started second revision of MQP report, continued to document PCB.
PC Software: Reformatted data collection algorithm.
PIC Software: Began sensor reset code
PCB Testing: A major problem with the lab equipment came down to a defective power supply…but in the meantime it caused the destruction of 3 PICs and caused a completely unproductive day.

Friday Report: Continued to document PCB.
Presentation: Began presentation outline.
PCB: Started to develop PIC board due to power supply problems; yesterday, added voltage regulator to PIC board to prevent future problems.
PIC Software: Completed reset code.
PC Software: Completed 90% of the software, just cleanup and tweaking remains.
System Integration: First complete system test. Bar Magnet—Sensor PCB—PIC—Serial link—PC Data Collection—PC Display.

Saturday Report: Continued hardware documentation, PC documentation, and Executive Summary.
Presentation: Continued design.

Week 9 Sunday October 5, 2003 – Saturday October 11, 2003

Week's goals:

1) PC Software: Finalize PC Data Processing / Display Software
2) Sensor PCB: Calibrate sensors, fix sensor offsets
3) PIC PCB: Finish PCB design, add buffers before reset circuitry, build and populate PIC PCB
4) System Testing: Design and Conduct Scientific Experiments to characterize behavior of system with the goal of determining feasibility of this application toward a 3D mouse
5) Presentation: Continue creating presentation
6) Report: Have a completed first draft or subsequent draft of every section in the report

---

Planned schedule:

Sunday General report and system development.

Monday Presentation: Continue outline.
PC Software: Continue cleanup.
System Testing: "Play" with the system to design scientific experiments.

Tuesday Presentation: Continue outline.
PIC PCB: Spin, populate and test board
System Testing: Continue to design scientific experiments.

Wednesday Presentation: Complete rough outline.
System Testing: Begin conducting experiments.

Thursday System Testing: Continue conducting experiments and tweaking system.

Friday System Testing: Complete experiments.

Saturday System Testing: Complete tweaking system.

Actual schedule:

Sunday PC Software: One more complete redesign. Nevertheless, it paid off in the end. The software works SWEET.

Monday PIC PCB: Schematic design.
Presentation: Continued outline.
System Testing: "Played" with the system and designed scientific experiments.

Tuesday PIC PCB: PCB design.
Presentation: Continued

Wednesday Presentation: Continued
PIC PCB: Spun, populated, partially tested.

Thursday Presentation: Completed all background, overview and hardware slides.
System Testing: Began to design scientific experiments.

Friday PIC PCB: Completed Testing
Hardware Test Rig: Constructed the test rig
PIC Software: Wrote more code to aid in system testing
PC Software: Began writing more code to aid in system testing
Dadisp Software: Researched a quick way to acquire a 3D plot of our data so that we can quickly run tests and get feedback (also it will be nice to have for our presentations).
System Testing: Played with the system with some PhD students. They made suggestions as to tests we could run.

Appendix F – Weekly Summaries

In addition to the weekly progress reports, we submitted to our advisor a detailed summary of what we accomplished in the prior week. These summaries provide insight into the main tasks we performed. What follows are the summaries submitted on the Friday of each week between August 10, 2003 and October 11, 2003.

Week 2 Sunday August 17, 2003 – Saturday August 23, 2003

At the start of this week, our preliminary research led us to a list of possible technologies, which we could use to implement a 3D positioning system. Our list included RF, ultrasound, magnetic field sensors, capacitor arrays, eddy current sensors, magnetic coils along with LC circuits and an array of inductors.

Although each method requires specific components, the ones most difficult to acquire are the magnetic field sensors. Therefore, we decided to order a few magnetic field sensors, which could be useful in a possible design.

Our first step in researching magnetic sensors was identifying a list of manufacturers. Our list included Analog Devices, Sentron, Honeywell, Phillips, and NVE. The following day, on Tuesday, we examined datasheets for each of the magnetic field sensors manufactured by the companies we identified. The datasheets provided us with useful information such as whether the sensors detect one, two or three axis of a magnetic field. We were also able to obtain the sensitivity of each sensor, the output voltage range and their linearity. However, we still did not know how they worked and what their output would be based on different proximities of a magnet. Therefore, we compiled a list of parts we hoped, when tested, could answer our remaining questions and submitted the list to John Harris.

On Wednesday, we continued to examine the possibility of using an LC circuit to create a resonant frequency detectable by a larger receiver coil. To form a basic proof of concept we created two coils, a large one with a diameter around ten inches, and a second, much smaller coil to represent the LC pea component. When inducing a current through the larger coil, the smaller coil was highly successful at picking up the field. The output on an oscilloscope showed a change in amplitude of the signal when we moved the coil. The detectable range was over a foot and a half above the coil.

On Thursday, we continued investigating the use of coils while we waited for the magnetic sensor to arrive. In our second experiment, we switched the transmitting and receiving coils so that the larger coil would be the receiver and the smaller coil would be the transmitter. The result was equally as appealing even over a significant distance. However, to acquire these results, we needed to increase the frequency of the transmitter.

That afternoon, we received the magnetic sensor; however, we needed an adequate surface mount adapter, which we did not have. Therefore, at this time we are not able to exclude using magnetic sensors. Since an LC circuit and a sensor coil proved to be a viable option we kept it on our list. Instead, we continued to examine our initial list of possibilities looking to exclude invalid options.

We decided to exclude RF as a possible option because the desired container size is too small to measure signal delay. In addition, ultrasound was not an option primarily because of the container size, but also because of interference generated by the walls. We excluded using capacitor arrays because the placement of a hand inside the box will be detected along with the target object. We also found eddy current sensors not appropriate for this project since we do not have enough knowledge in physics to be able to understand the phenomenon. Therefore, our shortened list of possible implementations exclusively contains magnetic coils with an LC circuit, an array of inductors and magnetic field sensors.

Week 3 Sunday August 24, 2003 – Saturday August 30, 2003

SOIC magnetic field sensor development:

- After acquiring the two-axis magnetic field sensor, we started to develop a test platform for the chip.
  - The datasheet for the Honeywell magnetic sensor provided test circuits to obtain a reasonable platform on which to test the chip.
  - The Magnetic field sensor only comes in an SOIC package, so a SOIC to DIP adapter had to be constructed. In order to implement the sensor in a circuit, we constructed a printed circuit board to mount the chip without damaging it.
  - To gain an understanding of the Magnetic Sensor IC we started developing several of the recommended test circuits.
  - Currently we are still constructing the test circuits so no data has been collected in reference to the sensor.

LC circuit detection:

- We developed an LC circuit for testing its detection with a search coil at the resonant frequency of the circuit.
  - We created an inductive wheatstone bridge to view the small change in amplitude and phase of the sensing coil when the LC circuit was present.
  - Our wheatstone bridge failed to isolate any changes when the LC circuit was present in the magnetic field.
  - We continued to researched methods to improve our wheatstone bridge design.
  - We believe that the current inductor ratios used in our inductive wheatstone bridge and our methods for bridge nullification may be a large source of our problems.
  - We came across an instrumentation amplifier that we believe will provide potential improvements to our wheatstone bridge and will be applicable in the development of the magnetic sensor's test circuit as it has an internal wheatstone bridge.

Project's direction:

- We looked back on the research and development we completed until this point to determine our project's future direction.
  - We met with John Harris to discuss whether we should continue to look at our three remaining development methods, hoping to have good reason to eliminate

two or to immediately chose one method and peruse development along those lines.
  - John advised us to continue looking at all three possibilities, using six coils, an array of coils or using a magnetic field sensor, until one solution stands out as the best choice.
  - John appeared to be most pleased with us pursuing the greatest single viable solution. He seemed to prefer not following the MIT approach if we could find another technology to be feasible.

Background Research:

- We further researched existing technologies and applicable electromagnetic methods
  - We came across the MIT six coil thesis and read it for comparison purposes.
  - We continued to examine the principles behind wheatstone bridges.
  - We investigated the patent acquired by Wacom for their LC art pad.
  - We researched magnetostrictors and other LC anti-shoplifting technology.

Week 4 Sunday August 31, 2003 – Saturday September 6, 2003

SOIC magnetic field sensor development:

- Completed initial development of a test rig for the magnetic sensor.
  - This required that we construct an instrumentation amplifier circuit to read the output of one axis of the magnetic sensor.
  - The rig worked reasonably well and allowed us to see a change in voltage that was proportional to orientation and position of an object creating a magnetic field.
  - The findings were extremely promising and led us to choose these sensors as a basis for our 3d mouse.

LC circuit detection:

- We successfully developed an inductive Wheatstone bridge that we used to isolate inductance changes caused by an object affecting the field of the search coil.
  - A significant amount of research and calculations went into creating an effective bridge.
  - The instrumentation amplifier, which we also interfaced with the magnetic field sensor, was utilized in this system. It allowed us to see the small changes in the search coil's inductance when an object was present in the field.

Project's direction:

- We decided on one technology!!! Since the magnetic sensors seem promising and have never been placed toward the development of a small 3d tracking system, we decided that it would be a great design project.
  - John is gone for the week so we have not told him the news yet but we believe he will be excited to hear that we have chosen to explore a solution that is far from the MIT system.

Background Research & Report Additions:

- We further researched existing technologies and applicable electromagnetic methods.
  - We started a detailed summary of the MIT solution for comparison purposes.
  - We continued to examine the theoretical principles behind inductive Wheatstone bridges.

- We continued to research magnetostrictors and other LC anti-shoplifting technology.

Repo rt Revisions:

- From your comments on our report, we began reformatting and revising.
  - Outline: The basic structure has been revised allowing fewer chapters. Please look specifically at the substructure, do you like the numbering system. We have not yet revised everything. The major editing changes took place in the majority of the background section.
  - Headings: We are still working on designing more concise and interesting subject headings.
  - Introduction: We are holding off on re-writing the introduction until we have a better idea of exactly what we will be building for this project. We noted your comments and completely agree with you suggestions. We expect that the first cut introduction should be complete by the end of next week.
  - Editing: We started to work on the flow of the report…we completely agree that it does not flow well or tell a story at this point. We already went through the start of the report and formed transitions between thoughts and sections.
  - However, once we complete writing up why we chose to use magnetic sensors and finalize the research section, we will have provided a much better flow.

Week 5 Sunday September 7, 2003 – Saturday September 13, 2003

Report:

- As you requested we are including a copy of the table of contents. The section of the report that we also attached has been edited and we worked hard to integrate the sections as one continuous document as you suggested last time. Therefore, along with our other tasks, we did not have the time to work on the introduction. We thought it would be better to focus on the continuity of the report before focusing on the introduction. Therefore, the introduction will be a major focus of the report editing next week along with our other outlined tasks.

Parts:

- Ordering parts has been extremely frustrating this week! We submitted our order this Tuesday and still the order has not been placed. But we do understand that it could not be any other way. The order required calling two separate companies to determine how long it would take to get in our magnetic sensors. We found out today that they would not be in for 14 weeks! Due to that fact, we implemented our backup plan; order 2axis and 1axis magnetic field sensors and create our own 3axis magnetic field sensor. Hopefully all our parts will be in by the end of next week

PC Software:

- After "playing" with LabView and building a Visual C++ serial port program that resembled AOL Instant Messenger we determined that it would be easier to program in C++ due to our prior experience using the language. LabView would require that we learn how to use the package along with how to implement the code. We felt that it would take longer to learn a new language verses coding in Visual C++.

PIC Software / Hardware:

- We have spent this week becoming familiar our chosen PIC. This PIC comes equip with a 10 to 1 ADC and a serial port. The ADC will individually convert one of the 10 inputs at a time with 12bit resolution. The on-board serial port allows us to manage the ADC while simultaneously transmitting serial data to the PC. The code for ADC and serial communication is well on the way and will be complete next week.

Week 6 Sunday September 14, 2003 – Saturday September 20, 2003

Report:

- This week we decided to put our report to the side for a little while. Some minor changes have been done in our *free* time, but not enough to show. This week instead of the report we will give you a copy of the code we have written, and a PCB layout.

Parts:

- Monday of this week our parts were finally ordered. Fortunately the company UL deals with has a turn around time of 1 day, so we had all the parts in our hands on Tuesday after our weekly meeting. Almost everything was exactly what we expected. Unfortunately we missed a major detail about the microprocessors we ordered. They are one time programmable. Fortunately Voytek brought some with him which are identical in all ways except that the analog to digital converter has only 8 inputs, as opposed to 10. Since this order was rather rushed because it needed to be signed off by Arshak who was leaving on business trips. Because of this we placed another order Wednesday for minor components such as capacitors and a few driver MOSFETs.

PC & PIC Software:

- Both PC and PIC software are currently going excellent. On the computer side, we have come a long way since the AOL Instant Messenger like application. There are a few bugs, but currently the software can connect to the serial port, detect if the microprocessor is connected by sending handshaking commands. The data received from the microcontroller is then written to a data file which can be imported into excel for graphing. The PIC software is right in line with the PC software. The PIC is able to receive commands from the computer and respond to them. The handshaking protocol has been implemented, and once this is done another command makes the PIC start transmitting hard coded test data.

PIC Hardware:

- As stated before, we unfortunately chose PIC microcontrollers which are one time programmable and accidentally flashed two of three of them. Fortunately we have three more almost identical ones which are almost exact and use flash memory. The PIC is now on a breadboard, with a TTL to Serial chip, and an LCD attached. Any command the computer sends are displayed on the LCD for easy debugging. Currently three packets are hard coded into the PIC which can be transmitted repeatedly to the computer.

Circuit Design:

- The circuitry surrounding the magnetic sensors has been designed thoroughly. There are only a few deta ils which need to be checked. Since a few of the components are only available in surface mount, we had to design a PCB board to mount everything on. The design of the board is almost completely done. It is only waiting for the last few components to be fitted to make sure the pads are the right size. A few pins will also be added for debugging purposes.

Week 7 Sunday September 21, 2003 – Saturday September 27, 2003

Report:
- The main focus this week has been the Introduction. We hope you enjoy!

PC & PIC Software:
- This week was extremely successful! The PIC and the PC communicate beautifully. The ADC on the PIC has been configured and the PC software has been reformatted to better respond to the demands of real time data collection. Complete Flowcharts for both the PIC and PC code have been attached to help you better understand the capabilities of the software. Also we are currently working to up the baudrate to 115kb/s, we have attached a selection table we created to help determine baudrate compatibility between the PIC and PC.

PCB Design & Fabrication:
- We spun the board and began populating it. We have attached a basic testing plan describe the order by which we plan to populate and test the PCB functionality.

Week 8 Sunday September 28, 2003 – Saturday October 4, 2003

Report:
- Everyone has been working a bit on many sections. We would love your help with feedback as soon as possible so that the editing turnaround can be expedited.

PC & PIC Software:
- Again, this week was extremely successful. The PIC now controls the sensor reset circuitry allowing increased accuracy and sensitivity of our data. The PC now has the ability to display the incoming data from each sensor graphically in real time (sadly there is approximately a 1.5 second delay from a change in the sensor magnitude to display on monitor, we believe that this is an acceptable delay for the prototype since we do not have the time shorten the delay.)

PCB Population & Testing:
- We completed population and testing of the sensor PCB, it went relatively smoothly and no need for redesign. Calibration of the sensors is still required to obtain accurate data.

Week 9 Sunday October 5, 2003 – Saturday October 11, 2003

Report:

- Our focus on the report this week was to complete all of the minor sections such as the Abstract, Executive Summary, Title Page, along with finishing the hardware and software documentation for Chapter 4. We continued to edit existing sections of the report and made the changes requested by our advisor. The main section that remains to be completed is the Results, Conclusions and Limitations. However, to begin this section we must complete the hardware system testing!

PC Software:

- We decided to try one more time at modifying the PC code. We were not happy with the 1.5-second lag on the display so on Saturday and Sunday we completely reworked the program. The PC code now collects data in real time from the PIC, processes and then buffers it to a linked list, writes the data to a file and displays it at a refresh rate of 35 frames per second without any noticeable lag or delay. We are psyched!

PCB Population & Testing:

- We completed populating and testing of the PIC PCB; it went relatively smoothly. However, it did take longer than expected and ran into some valuable testing time.

Appendix G – Terms and Definitions

Capaciflector – A device that changes capacitance when an object is placed near its sensitive plane (Mitchell, 2003).

DB9 – Specifies the physical connector size and number of pins used for serial communication.

Eddy Currents – Circular currents generating their own magnet field, induced in a metal by an external changing magnetic field.

Electronic Article Surveillance Tag (EAS) – A sticker containing an inductor capacitor (LC) circuit commonly attached to retail merchandise for protection against shoplifting.

Hall Effect Sensor – A device that detects a magnetic field based on Lorentz force.

Inductive Wheatstone Bridge – A Wheatstone bridge designed from resistors, capacitors and inductors, that permits a balanced bridge with respect to both magnitude and phase.

Jumper – A connection between two points on a printed circuit board that can either be closed or open. A typical use is the ability to isolate different segments of the circuit allowing each portion to be individually debugged.

Magnetoresistive Sensor – An integrated circuit containing a resistive Wheatstone bridge sensitive to the intensity and orientation of magnetic fields as low as 30 microGauss.

Magnetostrictor – The most common type of EAS shoplifting tag containing a magnetically coupled strip of metallic glass that changes shape and produces a ringdown response when pulsed with a magnetic field at its resonant frequency (Hsiao, 2001, p. 10) (Energen, Inc. 2003, p. 1).

Multithreaded – A process that contains multiple linear programs each devoted to a specific task.

Offset Strap – A metal strip in a magnetoresistive sensor used to offset any ambient magnetic field through applying a DC current.

PIC – (Programmable Interrupt Controller) A microprocessor designed by MicroChip Corporation.

RFID – (Radio Frequency Identification) A tag containing a silicon microchip and an antenna capable of transmitting data such as an identifying code to a wireless receiver (Mayfield, 2002, p. 1).

RS232 – The most common used standard for serial communication.

Set/Reset Strap – A metal strip with a resistance under 10O, magnetically coupled to a magnetoresistive sensor that when pulsed raises the sensitivity.

ThreadSafe – A programming technique that requires any function to take care of protecting their internal variables, so a thread can call that function and not worry about accessing resources if another thread was interrupted (Weisz, 1996).

USART – (Universal Synchronous/Asynchronous Receiver/Transmitter) A port used for serial transmission of data.

Appendix B Supplemental

// MQP_3d.cpp

```
/************************************************************************
This code cannot be copied, modified, nor used without giving credit to the original authors.
The authors give no warranty as to the workings of this program.
Also the text included here must be left with every complete or partial code replication.

Authors:
        Andrea L.M. Baker, Wojciech Krajewski and Daniel I. Wallance
Completion Date:
        Saturday October 18, 2003.
Designed For:
        This code was written for AMT Ireland at the University of Limerick, Ireland during the fulfillment
        of a Major Qualifying Project for Worcester Polytechnic Institute, Worcester, MA, United States.
************************************************************************/
#include "stdafx.h"
#include "Serial_Comm.h"
#include "MQP_3d.h"
#include "Data_Processing.h"
#include "MQP_3dDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

HANDLE SendPortNotInUseEvent;
HANDLE SendWriteThreadWaiting;
```

```
HANDLE SendWriteDataObject;
bool PortConfigured;
bool Program_Active;
bool Display_Position;
int Display_Count;
int Clear_Count;
void Write_To_Log();

/////////////////////////////////////////////////////////////////////////////
// CMQP_3dApp

BEGIN_MESSAGE_MAP(CMQP_3dApp, CWinApp)
	//{{AFX_MSG_MAP(CMQP_3dApp)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMQP_3dApp construction

CMQP_3dApp::CMQP_3dApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CMQP_3dApp object
```

```
CMQP_3dApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CMQP_3dApp initialization

BOOL CMQP_3dApp::InitInstance()
{
	AfxEnableControlContainer();

	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC statically
#endif

	CMQP_3dDlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
```

```
            //  dismissed with Cancel
        }

        // Since the dialog has been closed, return FALSE so that we exit the
        //  application, rather than start the application's message pump.
        return FALSE;
}
```

//MQP_3d.rc

```
/************************************************************************
This code cannot be copied, modified, nor used without giving credit to the original authors.
The authors give no warranty as to the workings of this program.
Also the text included here must be left with every complete or partial code replication.

Authors:
        Andrea L.M. Baker, Wojciech Krajewski and Daniel I. Wallance
Completion Date:
        Friday October 17, 2003.
Designed For:
        This code was written for AMT Ireland at the University of Limerick, Ireland during the fulfillment
        of a Major Qualifying Project for Worcester Polytechnic Institute, Worcester, MA USA.
************************************************************************/
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
//////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

//////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

//////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
```

```
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

#ifdef APSTUDIO_INVOKED
//////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
   "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
   "#include ""afxres.h""\r\n"
   "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
   "#define _AFX_NO_SPLITTER_RESOURCES\r\n"
   "#define _AFX_NO_OLE_RESOURCES\r\n"
   "#define _AFX_NO_TRACKER_RESOURCES\r\n"
   "#define _AFX_NO_PROPERTY_RESOURCES\r\n"
   "\r\n"
   "#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)\r\n"
   "#ifdef _WIN32\r\n"
```

```
    "LANGUAGE 9, 1\r\n"
    "#pragma code_page(1252)\r\n"
    "#endif //_WIN32\r\n"
    "#include ""res\\MQP_3d.rc2""  // non-Microsoft Visual C++ edited resources\r\n"
    "#include ""afxres.rc""         // Standard components\r\n"
    "#endif\r\n"
    "\0"
END

#endif    // APSTUDIO_INVOKED


////////////////////////////////////////////////////////////////////////////
//
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME           ICON    DISCARDABLE     "res\\MQP_3d.ico"

////////////////////////////////////////////////////////////////////////////
//
// Dialog
//

IDD_ABOUTBOX DIALOG DISCARDABLE  0, 0, 235, 55
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About MQP_3d"
FONT 8, "MS Sans Serif"
BEGIN
```

```
    ICON            IDR_MAINFRAME,IDC_STATIC,11,17,20,20
    LTEXT           "MQP_3d Version 1.0",IDC_STATIC,40,10,119,8,SS_NOPREFIX
    LTEXT           "Copyright (C) 2003",IDC_STATIC,40,25,119,8
    DEFPUSHBUTTON   "OK",IDOK,178,7,50,14,WS_GROUP
END

IDD_MQP_3D_DIALOG DIALOGEX 0, 0, 470, 356
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
EXSTYLE WS_EX_APPWINDOW
CAPTION "AMT-WPI 2003"
FONT 8, "MS Sans Serif"
BEGIN
    GROUPBOX        "",IDC_OUTLINE,7,7,149,345,NOT WS_VISIBLE
    GROUPBOX        "Program Control",IDC_STATIC,13,18,132,88
    LTEXT           "Com Port:",IDC_STATIC,36,31,32,8
    COMBOBOX        IDC_COMPORT,77,29,48,95,CBS_DROPDOWN | WS_VSCROLL |
                    WS_TABSTOP
    GROUPBOX        "Mode Select",IDC_STATIC,25,44,114,53,WS_GROUP
    CONTROL         "Standard",IDC_STANDARD_MODE,"Button",BS_AUTORADIOBUTTON |
                    WS_GROUP,41,54,45,10
    CONTROL         "Button Testing",IDC_BUTTON_MODE,"Button",
                    BS_AUTORADIOBUTTON,41,82,62,10
    CONTROL         "Write Raw Data",IDC_DATAFILES,"Button",BS_AUTOCHECKBOX |
                    WS_GROUP | WS_TABSTOP,63,65,67,10
    LTEXT           "Program Log",IDC_STATIC,15,124,41,8
    LISTBOX         IDC_LOG_LIST,14,135,131,157,LBS_NOINTEGRALHEIGHT |
                    WS_VSCROLL | WS_HSCROLL
    CTEXT           "X",IDC_STATIC,196,328,10,9,0,WS_EX_STATICEDGE
    CTEXT           "Y",IDC_STATIC,258,328,10,9,0,WS_EX_STATICEDGE
    CTEXT           "Z",IDC_STATIC,320,328,10,9,0,WS_EX_STATICEDGE
    EDITTEXT        IDC_X,189,340,24,14,ES_AUTOHSCROLL | ES_READONLY | NOT
```

```
                WS_BORDER | NOT WS_TABSTOP,WS_EX_CLIENTEDGE |
                WS_EX_STATICEDGE
    EDITTEXT        IDC_Y,251,340,24,14,ES_AUTOHSCROLL | ES_READONLY | NOT
                WS_BORDER | NOT WS_TABSTOP,WS_EX_CLIENTEDGE |
                WS_EX_STATICEDGE
    EDITTEXT        IDC_Z,313,340,24,14,ES_AUTOHSCROLL | ES_READONLY | NOT
                WS_BORDER | NOT WS_TABSTOP,WS_EX_CLIENTEDGE |
                WS_EX_STATICEDGE
    PUSHBUTTON      "Run",IDC_RUN,66,335,18,14,BS_FLAT
    PUSHBUTTON      "Stop",IDC_STOP,90,335,20,14,BS_FLAT
    PUSHBUTTON      "Shutdown",IDOK,112,335,37,14,BS_FLAT
    PUSHBUTTON      "Clear Log",IDC_CLEAR_LIST,113,123,34,11,BS_FLAT
    EDITTEXT        IDC_X_AVE,38,314,24,12,ES_AUTOHSCROLL | ES_READONLY |
                NOT WS_VISIBLE | NOT WS_BORDER | NOT WS_TABSTOP
    EDITTEXT        IDC_Y_AVE,80,314,24,12,ES_AUTOHSCROLL | ES_READONLY |
                NOT WS_VISIBLE | NOT WS_BORDER | NOT WS_TABSTOP
    EDITTEXT        IDC_Z_AVE,122,314,24,12,ES_AUTOHSCROLL | ES_READONLY |
                NOT WS_VISIBLE | NOT WS_BORDER | NOT WS_TABSTOP
    GROUPBOX        "",IDC_XYZ_AVE_BOX,16,303,134,26,NOT WS_VISIBLE
    CTEXT           "X",IDC_X_TEXT,25,312,10,9,NOT WS_VISIBLE,
                WS_EX_STATICEDGE
    CTEXT           "Y",IDC_Y_TEXT,67,312,10,9,NOT WS_VISIBLE,
                WS_EX_STATICEDGE
    CTEXT           "Z",IDC_Z_TEXT,110,312,10,9,NOT WS_VISIBLE,
                WS_EX_STATICEDGE
    EDITTEXT        IDC_AVE_BUTTON_ROW,17,295,58,12,ES_AUTOHSCROLL |
                ES_READONLY | NOT WS_VISIBLE | NOT WS_BORDER
END


#ifndef _MAC
```

```
//////////////////////////////////////////////////////////////////////////////
//
// Version
//

VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904B0"
        BEGIN
            VALUE "CompanyName", "\0"
            VALUE "FileDescription", "3D Mouse Prototype\0"
            VALUE "FileVersion", "0, 0, 0, 4\0"
            VALUE "InternalName", "MQP_3d\0"
            VALUE "LegalCopyright", "Copyright (C) 2003\0"
            VALUE "LegalTrademarks", "\0"
            VALUE "OriginalFilename", "MQP_3d.EXE\0"
            VALUE "ProductName", "3D Mouse Prototype\0"
            VALUE "ProductVersion", "0, 0, 0, 4\0"
```

```
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END

#endif    // !_MAC


/////////////////////////////////////////////////////////////////////////////
//
// DESIGNINFO
//

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_ABOUTBOX, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 228
        TOPMARGIN, 7
        BOTTOMMARGIN, 48
    END

    IDD_MQP_3D_DIALOG, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 464
```

```
        TOPMARGIN, 7
    END
END
#endif    // APSTUDIO_INVOKED


////////////////////////////////////////////////////////////////////////////
//
// Bitmap
//

IDB_BITTest             BITMAP  DISCARDABLE     "res\\bitmap1.bmp"

////////////////////////////////////////////////////////////////////////////
//
// Dialog Info
//

IDD_MQP_3D_DIALOG DLGINIT
BEGIN
    IDC_COMPORT, 0x403, 5, 0
0x6f43, 0x316d, "\000"
    IDC_COMPORT, 0x403, 5, 0
0x6f43, 0x326d, "\000"
    IDC_COMPORT, 0x403, 5, 0
0x6f43, 0x336d, "\000"
    IDC_COMPORT, 0x403, 5, 0
0x6f43, 0x346d, "\000"
    IDC_COMPORT, 0x403, 5, 0
0x6f43, 0x356d, "\000"
    IDC_COMPORT, 0x403, 5, 0
```

```
0x6f43, 0x366d, "\000"
    IDC_COMPORT, 0x403, 5,  0
0x6f43, 0x376d, "\000"
    IDC_COMPORT, 0x403, 5, 0
0x6f43, 0x386d, "\000"
    0
END


////////////////////////////////////////////////////////////////////////////
//
// String Table
//

STRINGTABLE DISCARDABLE
BEGIN
    IDS_ABOUTBOX            "&About MQP_3d..."
END

#endif    // English (U.S.) resources
////////////////////////////////////////////////////////////////////////////



#ifndef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
```

```
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE 9, 1
#pragma code_page(1252)
#endif //_WIN32
#include "res\MQP_3d.rc2"  // non-Microsoft Visual C++ edited resources
#include "afxres.rc"         // Standard components
#endif

/////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

// MQP_3dDlg.cpp

```
/*************************************************************************
This code cannot be copied, modified, nor used without giving credit to the original authors.
The authors give no warranty as to the workings of this program.
Also the text included here must be left with every complete or partial code replication.

Authors:
        Andrea L.M. Baker, Wojciech Krajewski and Daniel I. Wallance
Completion Date:
        Friday October 17, 2003.
Designed For:
        This code was written for AMT Ireland at the University of Limerick, Ireland during the fulfillment
        of a Major Qualifying Project for Worcester Polytechnic Institute, Worcester, MA USA.
*************************************************************************/
#include "stdafx.h"
#include "ReadThread.h"
#include "WriteThread.h"
#include "Serial_Comm.h"
#include "MQP_3d.h"
#include "Data_Processing.h"
#include "MQP_3dDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About
```

```
class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
```

```
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMQP_3dDlg dialog

CMQP_3dDlg::CMQP_3dDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CMQP_3dDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CMQP_3dDlg)
	m_comport = "Com1";
	m_log_list = _T("");
	m_displayx = 0;
	m_displayy = 0;
	m_displayz = 0;
	m_datafiles = TRUE;
	m_mode_select = 0;
	m_x_ave = _T("");
	m_y_ave = _T("");
	m_z_ave = _T("");
	m_ave_button_row = _T("");
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
```

```
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);


}

void CMQP_3dDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CMQP_3dDlg)
	DDX_Control(pDX, IDC_Z_AVE, m_z_ave_control);
	DDX_Control(pDX, IDC_Y_AVE, m_y_ave_control);
	DDX_Control(pDX, IDC_X_AVE, m_x_ave_control);
	DDX_Control(pDX, IDC_AVE_BUTTON_ROW, m_ave_button_row_control);
	DDX_Control(pDX, IDC_LOG_LIST, m_log_list_control);
	DDX_CBString(pDX, IDC_COMPORT, m_comport);
	DDX_LBString(pDX, IDC_LOG_LIST, m_log_list);
	DDX_Text(pDX, IDC_X, m_displayx);
	DDX_Text(pDX, IDC_Y, m_displayy);
	DDX_Text(pDX, IDC_Z, m_displayz);
	DDX_Check(pDX, IDC_DATAFILES, m_datafiles);
	DDX_Radio(pDX, IDC_STANDARD_MODE, m_mode_select);
	DDX_Text(pDX, IDC_X_AVE, m_x_ave);
	DDX_Text(pDX, IDC_Y_AVE, m_y_ave);
	DDX_Text(pDX, IDC_Z_AVE, m_z_ave);
	DDX_Text(pDX, IDC_AVE_BUTTON_ROW, m_ave_button_row);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CMQP_3dDlg, CDialog)
	//{{AFX_MSG_MAP(CMQP_3dDlg)
	ON_WM_SYSCOMMAND()
```

```
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_WM_LBUTTONDOWN()
	ON_WM_RBUTTONDOWN()
	ON_WM_TIMER()
	ON_BN_CLICKED(IDC_RUN, OnRun)
	ON_BN_CLICKED(IDC_STOP, OnStop)
	ON_BN_CLICKED(IDC_STANDARD_MODE, OnStandardMode)
	ON_BN_CLICKED(IDC_BUTTON_MODE, OnButtonMode)
	ON_BN_CLICKED(IDC_CLEAR_LIST, OnClearList)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMQP_3dDlg message handlers

BOOL CMQP_3dDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
```

```
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

//------------------Init Globals----------------------
	SendPortNotInUseEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
	SendWriteDataObject = CreateEvent(NULL, FALSE, FALSE, NULL);
	Program_Active = FALSE;
	Display_Position = FALSE;
	PortConfigured = FALSE;
	Display_Count = 0;
	Clear_Count =0;

	//Dlg item init
	m_Update_Display = FALSE;

	//Init the CDataProcessing object members
	m_MyData.hRaw_Data_File = NULL;
	m_MyData.hTest_Data_File = NULL;
	sprintf(m_MyData.m_foldername,"%s","c:\\MQP_3d_Data"); // use '\\' to represent the character '\'
	m_MyData.Create_Raw_Data_Linked_List();
	memset(&m_MyData.m_past_pea_pixel,-1,sizeof(m_MyData.m_past_pea_pixel));
	memset(&m_MyData.m_current_pea_pixel,-1,sizeof(m_MyData.m_past_pea_pixel));
```

```
	//Init the CSerialPort object members
	m_MyPort.m_overflow_error = 0;
	m_MyPort.m_hComm = NULL;

	//Init the CMQP_3dDlg object members
	m_hReadThread = NULL;
	m_hWriteThread = NULL;



//----------Draw The Window and save Drawing Specs----------
	CRgn  m_crgn;
	CWnd *m_outline;
	CRect my_rect;
	GetWindowRect(&my_rect);

	//Set the Size and style of the Dialog Window
	::SetWindowPos(this->GetSafeHwnd(), HWND_TOP, 0, 0,
		my_rect.Height(),my_rect.Height(),
		SWP_SHOWWINDOW);

	m_current_wnd_level = (int)TOP;
	m_last_wnd_level = (int) TOP;


	//Get the Clipping Region Specifications
	CDC * pDC = GetDC();

	GetClientRect(&my_rect);
	m_outline = this->GetDescendantWindow(IDC_OUTLINE);
```

```
	m_outline->GetClientRect(&m_MyData.m_outline_rect);

	pDC->OffsetWindowOrg( -(m_MyData.m_outline_rect.Width()+10),0);

	//Store the current clipping region specifications
	m_MyData.m_clipping_rect.left = m_MyData.m_outline_rect.right;
	m_MyData.m_clipping_rect.top = m_MyData.m_outline_rect.top;
	m_MyData.m_clipping_rect.right =my_rect.right;
	m_MyData.m_clipping_rect.bottom =my_rect.bottom;

	pDC->FillSolidRect(my_rect,RGB(255,255,255));

	this->ReleaseDC(pDC);


//---------Create The Threads--------
	if(m_hReadThread == NULL)
	{
		m_hReadThread = (ReadThread*) ::AfxBeginThread( RUNTIME_CLASS( ReadThread ),
THREAD_PRIORITY_ABOVE_NORMAL,0,CREATE_SUSPENDED,NULL );
		VERIFY( m_hReadThread );
		m_hReadThread->RegisterMyPortHandle(&m_MyPort);
		m_hReadThread->RegisterMyDataHandle(&m_MyData);
	}

	if(m_hWriteThread == NULL)
	{
		m_hWriteThread = (WriteThread*) ::AfxBeginThread( RUNTIME_CLASS( WriteThread ),
THREAD_PRIORITY_ABOVE_NORMAL,0,CREATE_SUSPENDED,NULL );
		VERIFY( m_hReadThread );
		m_hWriteThread->RegisterMyPortHandle(&m_MyPort);
```

```
		m_hWriteThread->RegisterMyDataHandle(&m_MyData);
	}

	//Register The Dialog Handle so that we can write to the log list
		m_hReadThread->RegisterDialogHandle(this);
		m_hWriteThread->RegisterDialogHandle(this);
		m_MyData.RegisterDialogHandle(this);
		m_MyPort.RegisterDialogHandle(this);


	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CMQP_3dDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CMQP_3dDlg::OnPaint()
```

```
{


	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
//----------Draw The Window and save Drawing Specs----------

		//Invalidate(TRUE);

		CDialog::OnPaint();

		CRect my_rect;
		GetClientRect(&my_rect);
```

```
		//Get the Clipping Region Specifications
		CDC * pDC = GetDC();
		pDC->OffsetWindowOrg( -(m_MyData.m_outline_rect.Width()+10),0);

		pDC->FillSolidRect(my_rect,RGB(255,255,255));

		this->ReleaseDC(pDC);

	HWND  mywnd;
	HWND topwnd;
	mywnd =	::FindWindow(NULL, TEXT("AMT-WPI 2003"));
	topwnd = ::GetForegroundWindow();
	if(mywnd == topwnd)
	{
		m_current_wnd_level = (int)TOP;

	}
	else
	{
		m_current_wnd_level = (int)BOTTOM;
	}


	if((m_Update_Display == TRUE||m_last_wnd_level == (int)BOTTOM && m_current_wnd_level == (int)TOP)|| (m_last_wnd_level ==
(int)TOP && m_current_wnd_level == (int)BOTTOM))
	{
		m_Update_Display = FALSE;
		RECT notvalid = {-1,-1,-1,-1};
		m_MyData.m_past_pea_pixel[0].m_rect = notvalid;
		m_MyData.m_past_pea_pixel[1].m_rect = notvalid;
		m_MyData.m_past_pea_pixel[2].m_rect = notvalid;
```

```
        }

        m_last_wnd_level = m_current_wnd_level;


        }

}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CMQP_3dDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}

void CMQP_3dDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
        CRgn  m_crgn;
        CWnd *m_outline;
        CRect m_outline_rect;
        RECT   my_rect;
        CDC * pDC = GetDC();
        GetClientRect(&my_rect);
        m_outline = this->GetDescendantWindow(IDC_OUTLINE);
        m_outline->GetClientRect(&m_outline_rect);
        m_crgn.CreateRectRgn(m_outline_rect.left+ m_outline_rect.Width(), m_outline_rect.top - m_outline_rect.Height(),my_rect.right,
my_rect.bottom );
        pDC->SelectClipRgn(&m_crgn);
```

```
	pDC->Ellipse(point.x,point.y, point.x + 30, point.y+30);
	pDC->Rectangle(point.x+30, point.y+30, point.x+60,point.y+60 );
	ReleaseDC(pDC);

	CDialog::OnLButtonDown(nFlags, point);
}

void CMQP_3dDlg::OnRButtonDown(UINT nFlags, CPoint point)
{
	Invalidate(TRUE);
	m_Update_Display = TRUE;

	CDialog::OnRButtonDown(nFlags, point);
}

void CMQP_3dDlg::OnTimer(UINT nIDEvent)
{
	switch(nIDEvent)
	{
		case IDT_UPDATE:

			if(Program_Active == TRUE && Display_Position == TRUE)
			{
				m_MyData.Determine_Pea_XYZ_Location();
				m_MyData.Determine_Pea_Pixel_Location();

				CRgn  m_crgn;
				CPen penBlack;
				CDC * pDC = GetDC();

				pDC->OffsetWindowOrg( -(m_MyData.m_outline_rect.Width()+10),0);
```

```
#if DEBUG_MODE == TRUE
                        pDC->SetROP2(R2_NOT);//Switch the drawing colors

                        for(int rect_num =0; rect_num < 3; rect_num++)
                        {
                                pDC->SelectObject(GetStockObject(NULL_BRUSH));
                                pDC->Rectangle(m_MyData.m_past_pea_pixel[rect_num].m_rect);//draw over the last rect with the
opposet color
                                pDC->Rectangle(m_MyData.m_current_pea_pixel[rect_num].m_rect); //draw the new rect
                                m_MyData.m_past_pea_pixel[rect_num].m_rect = m_MyData.m_current_pea_pixel[rect_num].m_rect;
//save the old rect
                        }

                        //Update the edit boxes with the most current raw data
                        m_displayx = m_MyData.m_current_pea_xyz[0].y;
                        m_displayy = m_MyData.m_current_pea_xyz[1].y;
                        m_displayz = m_MyData.m_current_pea_xyz[2].y;
                        UpdateData(FALSE);

#endif
                        ReleaseDC(pDC);

                        //Update the display
                        UpdateWindow();

                        if(Display_Count == (int)NUM_TO_DISPLAY_BEFORE_WRITE)
                        {
                                SetEvent(SendWriteDataObject);
                                Display_Count = 0;
```

```
		}
		else
		{
			Display_Count++;
		}

		if(m_MyData.m_mode ==BUTTON_MODE && m_Update_Gui_Vars == TRUE)
		{
			UpdateData(FALSE);
			m_Update_Gui_Vars = FALSE;
		}

	}

	if(Clear_Count == (int)NUM_T0_DISPLAY_BEFORE_CLEAR)
		{
			m_Update_Display = TRUE;
			Invalidate(TRUE);
			UpdateWindow();
			Clear_Count=0;
		}
		else
		{
			Clear_Count++;
		}
	break;

case IDT_CLEAR:
	//Invalidate the background of the window so that it will clear the background.
	//c
	//UpdateWindow();
```

```
                        break;
        }

        CDialog::OnTimer(nIDEvent);
}

void CMQP_3dDlg::OnRun()
{
        //Get the Update the GUI Status to the GUI Vars
        UpdateData(TRUE);

        if(m_MyPort.m_hComm == NULL && Program_Active == FALSE)
        {

                GetDlgItem(IDC_RUN)->EnableWindow(FALSE);
                //Save GUI Vars
                m_MyPort.m_gszPort = m_comport;
                m_MyData.m_Write_File_Flag = m_datafiles;
                m_MyData.m_mode = m_mode_select;

                //Set Display Timers
                SetTimer(IDT_UPDATE,5,NULL);//Set it to update the display 20times per second
                SetTimer(IDT_CLEAR,500,NULL);//Set it to clear the display every 0.5 seconds

                //Set up the button Mode vars
                m_MyData.m_write_button.button_pressed = FALSE;
                m_MyData.m_write_button.data_count = 0;
                m_MyData.m_write_button.num_of_button_clicks = 0;
                m_MyData.m_write_button.row_num = 0;
                m_MyData.m_write_button.data_sum[0] = 0;
                m_MyData.m_write_button.data_sum[1] = 0;
```

```
		m_MyData.m_write_button.data_sum[2] = 0;

		if((m_MyData.m_mode == STANDARD_MODE && m_MyData.m_Write_File_Flag == TRUE )||(m_MyData.m_mode
==BUTTON_MODE))
		{//Only write datafiles if the user wants them
			ResetEvent(SendWriteThreadWaiting);
			m_hWriteThread->ResumeThread();
			WaitForSingleObject(SendWriteThreadWaiting,INFINITE);
		}
		m_hReadThread->ResumeThread();
		WaitForSingleObject(SendPortNotInUseEvent,INFINITE );

		if(m_MyData.m_mode ==BUTTON_MODE)
		{
			//Enable the Row and column Average boxes
			GetDlgItem(IDC_XYZ_AVE_BOX)->ShowWindow(SW_SHOW);
			CMQP_3dDlg::m_ave_button_row_control.ShowWindow(SW_SHOW);
			CMQP_3dDlg::m_x_ave_control.ShowWindow(SW_SHOW);
			CMQP_3dDlg::m_y_ave_control.ShowWindow(SW_SHOW);
			CMQP_3dDlg::m_z_ave_control.ShowWindow(SW_SHOW);
			GetDlgItem(IDC_Z_TEXT)->ShowWindow(SW_SHOW);
			GetDlgItem(IDC_Y_TEXT)->ShowWindow(SW_SHOW);
			GetDlgItem(IDC_X_TEXT)->ShowWindow(SW_SHOW);

			CMQP_3dDlg::m_ave_button_row = "Button: – , Row: --";
			CMQP_3dDlg::m_x_ave = "0";
			CMQP_3dDlg::m_y_ave = "0";
			CMQP_3dDlg::m_z_ave = "0";
			m_Update_Display = TRUE;
			Invalidate(TRUE);
			UpdateData(FALSE);
```

```
		}

		//Start Program
		Program_Active = TRUE;

		WriteToLog("*****START*****");
	}
	else
	{
		WriteToLog("Error: Program Currently Active");
	}
}

void CMQP_3dDlg::OnOK()
{//This Closes The Applcation, all handles, all threads

	WriteToLog("*****END*****");

	if(Program_Active == TRUE)
	{
		//Tell the Program to suspend operations
		Program_Active = FALSE;
		Display_Position = FALSE;

		m_hReadThread->ResumeThread();//Sent this just incase the thread was never started
		ResetEvent(SendWriteThreadWaiting);
		m_hWriteThread->ResumeThread();
		WaitForSingleObject(SendWriteThreadWaiting,INFINITE);

		//Wait here until the ReadThread signals that it is not using the port.
		WaitForSingleObject(SendPortNotInUseEvent,INFINITE );
```

```
	//KillTimers
	KillTimer(IDT_UPDATE);
	KillTimer(IDT_CLEAR);

	//This will close the port if it was open.
	m_MyPort.Close_Serial_Port();

	//Close all file handles
	m_MyData.Close_Data_Files();
}

//Sent this just incase the thread was never started
m_hReadThread->ResumeThread();
m_hWriteThread->ResumeThread();

//Destroy the ReadThread
SetEvent(m_hReadThread->m_hEventKill);
SetEvent(m_hWriteThread->m_hEventKill);
m_hReadThread->Quit();
m_hWriteThread->Quit();

Sleep(1500); //Wait for the thread to close up everything and then destroy its self.

//Close all event handles
CloseHandle(SendPortNotInUseEvent);
CloseHandle(SendWriteDataObject);

//Clean up the linked list
m_MyData.Free_Raw_Data_Linked_List();
```

```
	//Close the dialog
	CDialog::OnOK();
}

void CMQP_3dDlg::OnStop()
{//Clean up the program but do not shutdown

	if(Program_Active == TRUE)
	{
		GetDlgItem(IDC_RUN)->EnableWindow(TRUE);
		WriteToLog("*****END*****");
		//Tell the Program to suspend operations
		Program_Active = FALSE;
		Display_Position = FALSE;

		m_hReadThread->ResumeThread();//Sent this just incase the thread was never started

		ResetEvent(SendWriteThreadWaiting);
		m_hWriteThread->ResumeThread();
		WaitForSingleObject(SendWriteThreadWaiting,INFINITE);

		//Wait here until the ReadThread signals that it is not using the port.
		WaitForSingleObject(SendPortNotInUseEvent,INFINITE );

		//KillTimers
		KillTimer(IDT_UPDATE);
		KillTimer(IDT_CLEAR);

		//This will close the port if it was open.
		m_MyPort.Close_Serial_Port();
```

```
			//Close all file handles
			m_MyData.Close_Data_Files();

			m_hReadThread->SuspendThread();
			m_hWriteThread->SuspendThread();
		}

		if(m_MyData.m_mode ==BUTTON_MODE)
			{
				//Disable the Row and column Average boxes
				GetDlgItem(IDC_XYZ_AVE_BOX)->ShowWindow(SW_HIDE);
				CMQP_3dDlg::m_ave_button_row_control.ShowWindow(SW_HIDE);
				CMQP_3dDlg::m_x_ave_control.ShowWindow(SW_HIDE);
				CMQP_3dDlg::m_y_ave_control.ShowWindow(SW_HIDE);
				CMQP_3dDlg::m_z_ave_control.ShowWindow(SW_HIDE);
				GetDlgItem(IDC_Z_TEXT)->ShowWindow(SW_HIDE);
				GetDlgItem(IDC_Y_TEXT)->ShowWindow(SW_HIDE);
				GetDlgItem(IDC_X_TEXT)->ShowWindow(SW_HIDE);
				m_Update_Display = TRUE;
				Invalidate(TRUE);

			}
}

void CMQP_3dDlg::WriteToLog(CString text)
{
		CString temp;
		m_log_list = text;
		m_log_list_control.AddString(m_log_list);
}
```

```
void CMQP_3dDlg::OnStandardMode()
{
	GetDlgItem(IDC_DATAFILES)->EnableWindow(TRUE);

}

void CMQP_3dDlg::OnButtonMode()
{
	GetDlgItem(IDC_DATAFILES)->EnableWindow(FALSE);

}

void CMQP_3dDlg::OnClearList()
{
	CMQP_3dDlg::m_log_list_control.ResetContent();
}
```

// ReadThread.cpp

```
/*********************************************************************
This code cannot be copied, modified, nor used without giving credit to the original authors.
The authors give no warranty as to the workings of this program.
Also the text included here must be left with every complete or partial code replication.

Authors:
        Andrea L.M. Baker, Wojciech Krajewski and Daniel I. Wallance
Completion Date:
        Friday October 17, 2003.
Designed For:
        This code was written for AMT Ireland at the University of Limerick, Ireland during the fulfillment
        of a Major Qualifying Project for Worcester Polytechnic Institute, Worcester, MA USA.
*********************************************************************/
#include "stdafx.h"
#include "MQP_3d.h"
#include "Serial_Comm.h"
#include "Data_Processing.h"
#include "MQP_3dDlg.h"
#include "ReadThread.h"



#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////////////////////////////////////
// ReadThread
```

```
IMPLEMENT_DYNCREATE(ReadThread, CWinThread)

ReadThread::ReadThread()
{
	// kill event starts out in the nonsignaled state
	m_hEventKill = CreateEvent(NULL, TRUE, FALSE, NULL);

}

ReadThread::~ReadThread()
{


}

void ReadThread::KillThread()
{
	//Signal the thread to shutdown!
	VERIFY(SetEvent(m_hEventKill));
}



BOOL ReadThread::InitInstance()
{
	bool Close_Program = FALSE;
	char cmd;

//-------------Read Thread Main Loop---------------
	while((WaitForSingleObject(m_hEventKill, 1) == WAIT_TIMEOUT))
```

```
	{
		if(Program_Active == TRUE)
		{
			//Tell the rest of the program that we are now using the port...do not close it
			ResetEvent(SendPortNotInUseEvent);
//			m_hMyDlg ->WriteToLog("ResetEvent:PortNotInUse");

			if(PortConfigured == FALSE)
			{//Initilize the Program

				if(m_hMyDlg->m_mode_select == STANDARD_MODE)
				{
					cmd = (char)PIC_ECHO_ADCSTART_CMD;
				}
				else if(m_hMyDlg->m_mode_select == BUTTON_MODE)
				{
					cmd = (char)PIC_ECHO_BUTTONSTART_CMD;
				}

				//Configure the Serial Port
				if(m_hMyPort->Configure_Serial_Port() != SUCCESS)
				{//Could not configure Port
					Close_Program = TRUE;
				}
				else if(m_hMyPort->Handshake_Pic((char) PIC_ECHO_CMD) != SUCCESS)
				{//Handshake was not successful!
					Close_Program = TRUE;
				}
				else if(m_hMyData->Init_Folder_Files() != SUCCESS)
				{//Could not Init Folders or Files
					Close_Program = TRUE;
```

```
		}
		else if(m_hMyPort->Handshake_Pic(cmd) != SUCCESS)
		{//Handshake was not successful!
			Close_Program = TRUE;
		}
		else
		{//Successful Inilization!
			PortConfigured = TRUE;
			Display_Position = TRUE;
		}
}
else
{//The port is configured properly

		//This is the main thread loop..it just reads and writes to linked list
		while(Close_Program == FALSE && Program_Active ==TRUE)
		{
			//Read Many Pictures From the Port
			if(m_hMyPort->Read_From_Port(m_hMyData) != SUCCESS)
			{
				Close_Program = TRUE;
			}
			else if(m_hMyData->Sort_Check_Raw_Data(m_hMyPort) != SUCCESS)
			{
				Close_Program = TRUE;
			}
		}
}
```

```
//------------------Check for Program Close------------------
                //Compare program faliures to the acceptable level
                if(Close_Program == TRUE)
                {
                        m_hMyDlg->GetDlgItem(IDC_RUN)->EnableWindow(TRUE);
                        //Send the close program message
                        Program_Active = FALSE;
                        Display_Position = FALSE;

                        //Serial Port Clean Up
                        m_hMyPort->Close_Serial_Port();

                        //Data File Clean up
                        m_hMyData->Close_Data_Files();

                        //KillTimers
                        m_hMyDlg->KillTimer(IDT_UPDATE);
                        m_hMyDlg->KillTimer(IDT_CLEAR);

                        m_hMyDlg->WriteToLog("Error: Read Thread Abort");

                        //reset system vars
                        Close_Program = FALSE;

                        m_hMyDlg->WriteToLog("*****END*****");
                }
        }
        else
        {
                //Tell the rest of the program that we are not using the port.
                Display_Position = FALSE;
```

```
                        PortConfigured = FALSE;
                        SetEvent(SendPortNotInUseEvent);
                //      m_hMyDlg->WriteToLog("SetEvent:PortNotInUse");

                }
        }

return TRUE;
}

int ReadThread::ExitInstance()
{
        // TODO:  perform any per-thread cleanup here
        return CWinThread::ExitInstance();
}

BEGIN_MESSAGE_MAP(ReadThread, CWinThread)
        //{{AFX_MSG_MAP(ReadThread)
                // NOTE - the ClassWizard will add and remove mapping macros here.
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// ReadThread message handlers
```

// WriteThread.cpp

```
/*************************************************************************
This code cannot be copied, modified, nor used without giving credit to the original authors.
The authors give no warranty as to the workings of this program.
Also the text included here must be left with every complete or partial code replication.

Authors:
        Andrea L.M. Baker, Wojciech Krajewski and Daniel I. Wallance
Completion Date:
        Friday October 17, 2003.
Designed For:
        This code was written for AMT Ireland at the University of Limerick, Ireland during the fulfillment
        of a Major Qualifying Project for Worcester Polytechnic Institute, Worcester, MA USA.
*************************************************************************/

#include "stdafx.h"
#include "mqp_3d.h"
#include "Serial_Comm.h"
#include "Data_Processing.h"
#include "WriteThread.h"
#include "MQP_3dDlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// WriteThread

IMPLEMENT_DYNCREATE(WriteThread, CWinThread)
```

```
WriteThread::WriteThread()
{
	m_hEventKill = CreateEvent(NULL, TRUE, FALSE, NULL);
}

WriteThread::~WriteThread()
{
}

void WriteThread::KillThread()
{
	//Signal the thread to shutdown!
	VERIFY(SetEvent(m_hEventKill));
}

BOOL WriteThread::InitInstance()
{
	bool Close_Program = FALSE;
	bool Write_Configured = FALSE;

	while((WaitForSingleObject(m_hEventKill, 1) == WAIT_TIMEOUT))
	{
		if(Program_Active == TRUE)
		{
			ResetEvent(SendWriteThreadWaiting);

			if(Write_Configured == FALSE)
			{	//m_hMyDlg->WriteToLog("Write: Configure Stuff");
				Sleep(10);
				//Sinc the write thread to the read thread when the program starts
```

```
                    m_hMyData->m_Data_Write_End = m_hMyData->m_Data_Sort_Current;
                    m_hMyData->m_Data_Write_Current = m_hMyData ->m_Data_Sort_Current;
                    Write_Configured = TRUE;


            }
            else
            {
                    while (Program_Active == TRUE && Close_Program == FALSE && PortConfigured == TRUE
&&(WaitForSingleObject(SendWriteDataObject,3000) != WAIT_TIMEOUT)  )
                    {
                            if(m_hMyData->m_mode ==STANDARD_MODE && m_hMyData->m_Write_File_Flag == TRUE)
                            {

                                    //m_hMyDlg->WriteToLog("Write: Time to write raw");
                                    if(m_hMyData->Write_Data_File((int)RAW_DATA_FILE) != SUCCESS)
                                    {
                                            m_hMyDlg->WriteToLog("Write: Write raw Failed");
                                            Close_Program = TRUE;
                                    }
                            }
                            else if(m_hMyData->m_mode == BUTTON_MODE)
                            {
                            //      m_hMyDlg->WriteToLog("Write: Time to write button");
                                    if(m_hMyData->Write_Data_File((int)BUTTON_DATA_FILE) != SUCCESS)
                                    {
                                    //      m_hMyDlg->WriteToLog("Write: Write button Failed");
                                            Close_Program = TRUE;
                                    }
                            }
```

```
                        }
                    }
                    if(Close_Program == TRUE)
                    {
                        m_hMyDlg->WriteToLog("Error: Write Thread Abort");
                        m_hMyDlg->GetDlgItem(IDC_RUN)->EnableWindow(TRUE);
                        //Clean up this thread and signal all other threads to terminate this instance of the program
                        Program_Active = FALSE;
                        Close_Program = FALSE;
                        Display_Position = FALSE;
                        Write_Configured = FALSE;
                        m_hMyDlg->WriteToLog("*****END*****");
                    }

                }
                else
                {//m_hMyDlg->WriteToLog("Write:Thread Waiting");
                    Write_Configured = FALSE;
                    Close_Program = FALSE;
                    SetEvent(SendWriteThreadWaiting);
                }


        }
        return TRUE;
}

int WriteThread::ExitInstance()
{
        // TODO:  perform any per-thread cleanup here
        return CWinThread::ExitInstance();
```

```
}

BEGIN_MESSAGE_MAP(WriteThread, CWinThread)
	//{{AFX_MSG_MAP(WriteThread)
		// NOTE - the ClassWizard will add and remove mapping macros here.
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// WriteThread message handlers
```

//Data_Processing.cpp

```
/************************************************************************
This code cannot be copied, modified, nor used without giving credit to the original authors.
The authors give no warranty as to the workings of this program.
Also the text included here must be left with every complete or partial code replication.

Authors:
        Andrea L.M. Baker, Wojciech Krajewski and Daniel I. Wallance
Completion Date:
        Friday October 17, 2003.
Designed For:
        This code was written for AMT Ireland at the University of Limerick, Ireland during the fulfillment
        of a Major Qualifying Project for Worcester Polytechnic Institute, Worcester, MA USA.
************************************************************************/
#include "stdafx.h"
#include "MQP_3d.h"
#include "Data_Processing.h"
#include "Serial_Comm.h"
#include "MQP_3dDlg.h"

//-------------------------------------------------------------------------
//Create_Raw_Data_Linked_List:
//        Creates a doubly linked list to hold raw picture data
void CDataProcessing::Create_Raw_Data_Linked_List()
{
        CRaw_Data * first;
        CRaw_Data * current;
        size_t sizeofCRaw_Data;

        //Get the size in bytes of a CRaw_Data object
        sizeofCRaw_Data = sizeof(CRaw_Data);
```

```
	//Create the first and second link
	first = (CRaw_Data *)calloc(sizeofCRaw_Data,1);
	current= (CRaw_Data *)calloc(sizeofCRaw_Data,1);

int count =0;
	//Point the first and second link to each other
	first->m_next = current;
	first->m_location = count;
	count++;
	current->m_last = first;
current->m_location = count;
	for(int i = 0; i<((int)NUM_OF_RAW_DATA_LINKS_IN_LIST); i++)
	{
		//Create the next link in memory
		current->m_next= (CRaw_Data *)calloc(sizeofCRaw_Data,1);

		//Set the last location pointer to the past link
		current->m_next->m_last = current;
		current->m_location = count;
		//Increment to the next location
		current = current->m_next;
		count++;
	}

	//link the list front and end
	first->m_last = current;
	current->m_next = first;

	//point the CDataProcessing object to the beginning of the list
	m_Data_Sort_Current = first;
```

```
	m_Data_Write_Current = first;
	m_Data_Write_End = first;

}

//-------------------------------------------------------------------------
//Free_Raw_Data_Linked_List:
//Cleans up the raw data list from memory
void CDataProcessing::Free_Raw_Data_Linked_List()
{
	CRaw_Data * current = m_Data_Sort_Current;
	CRaw_Data * next;

	//Break the linked list
	current->m_last->m_next = NULL;

	while(current->m_next != NULL)
	{//free one link in the list until all links have been deleted;
		next = current->m_next;
		free(current);
		current = next;
	}
}


//-------------------------------------------------------------------------
//Create_Folder
int CDataProcessing::Init_Folder_Files()
{
	int returncode;
	CString file_header;
```

```
CString temp;
unsigned long wordswritten;

//Check if the user wants to write data files
if((m_mode == STANDARD_MODE && m_Write_File_Flag == TRUE )||(m_mode ==BUTTON_MODE))
{
	//Check for the existance of the main folder, if it is not there then create it
	if((returncode = Create_Folder())!= SUCCESS)
	{
		return returncode;//ToDo:Error create folder did not create the directory
	}

	//Create the timestamp that will go on the file name and in the file
	Create_Time_Stamp();

	if(m_mode == STANDARD_MODE && m_Write_File_Flag == TRUE )
	{
		sprintf(m_filename,"%s\\%s__%s.txt",m_foldername,m_timestamp,"RawData");

		file_header.Format("File Name: %s\r\nTime Stamp: %s\r\nNumber of Active Planes:%u\r\n",m_filename,m_timestamp,
(int)NUM_OF_SENSORS);

		file_header += "Sensor ID:";
		for(int i = 0; i<((int)NUM_OF_SENSORS); i++)
		{
			temp.Format(",%u,%u,%u",i,i,i);
			file_header+= temp;
		}
		file_header += "\r\nAxis:";

		for(i = 0; i<((int)NUM_OF_SENSORS); i++)
```

```
                {
                        file_header+= ",x,y,z";
                }

                file_header += "\r\n,";

                hRaw_Data_File = Create_File(hRaw_Data_File);
                if(hRaw_Data_File == NULL)
                {
                        return returncode;
                }

//              sprintf(m_filename,"%s\\%s__%s.txt",m_foldername,m_timestamp,"LOG");
//              hLog = Create_File(hLog);
//              if(hLog == NULL)
//              {
//                      return returncode;
//              }
                if(!WriteFile(hRaw_Data_File,file_header.GetBuffer(1),file_header.GetLength(),&wordswritten,NULL))
                {
                        return COULD_NOT_WRITE_TO_FILE;
                }
        }
        else if(m_mode ==BUTTON_MODE)
        {//We are using a button to test the program
                sprintf(m_filename,"%s\\%s__%s.txt",m_foldername,m_timestamp,"ButtonData");

                hTest_Data_File = Create_File(hTest_Data_File);
                if(hTest_Data_File == NULL)
                {
                        return returncode;
```

```
                }

                if(!WriteFile(hTest_Data_File,file_header.GetBuffer(1),file_header.GetLength(),&wordswritten,NULL))
                {
                        return COULD_NOT_WRITE_TO_FILE;
                }


                //Create the PHP file handle
                sprintf(m_filename,"D:\\Web\\MQP\\%s.txt","Find");
                hPHP_Data_File = Create_File(hPHP_Data_File);
                if(hPHP_Data_File == NULL)
                {
                        return returncode;
                }

                if(!WriteFile(hPHP_Data_File,file_header.GetBuffer(1),file_header.GetLength(),&wordswritten,NULL))
                {
                        return COULD_NOT_WRITE_TO_FILE;
                }


        }
    }

    return SUCCESS;

}

//---------------------------------------------------------------------------
//Create_Folder
```

```
int CDataProcessing::Create_Folder()
{
	HANDLE hfolder;

	hfolder = CreateFile(m_foldername,	// open directory if it is present
	   GENERIC_WRITE | GENERIC_READ,// open for write and read
	   FILE_SHARE_READ,			// share for reading
	   NULL,		// no security
	   OPEN_EXISTING,		// existing file only
	   FILE_FLAG_BACKUP_SEMANTICS,
	   NULL);


	if(hfolder == INVALID_HANDLE_VALUE)
	{
		//function does not exist
		if(!CreateDirectory(m_foldername,NULL))
		{
			//ToDo: Error cannot create directory
			return COULD_NOT_CREATE_DIRECTORY;
		}

	}
	else
	{//directory exhists
		CloseHandle(hfolder);

	}

	return SUCCESS;
}
```

```
//---------------------------------------------------------------------------
//Create_Folder
HANDLE CDataProcessing::Create_File(HANDLE hFile)
{

	if(hFile == NULL)
	{
		hFile= CreateFile(m_filename,
					GENERIC_WRITE,		// open for write
					FILE_SHARE_READ,	// share for reading
					NULL,			// no security
					CREATE_ALWAYS,		// existing file only
					FILE_ATTRIBUTE_NORMAL,
					NULL);


		if (hFile == INVALID_HANDLE_VALUE)
		{
			hFile = NULL;

		}
	}
	return hFile;

}

//---------------------------------------------------------------------------
//Create_TimeStamp
void CDataProcessing::Create_Time_Stamp()
{//year,month,day,hour,minute,second
```

```
	int year;
	int month;
	int day;
	int hour;
	int minute;
	int second;

	time_t the_time;
	struct tm *the_time_converted;

	time(&the_time);
	the_time_converted = localtime(&the_time);
	year = the_time_converted->tm_year + 1900;
	month = the_time_converted->tm_mon + 1;
	day = the_time_converted->tm_mday;
	hour = the_time_converted->tm_hour;
	minute = the_time_converted->tm_min;
	second = the_time_converted->tm_sec;

	sprintf(&m_timestamp[0],"%02d%02d%02d%02d%02d%02d",year%100,month%100,day%100,hour%100,minute%100,second%100);
}

//-------------------------------------------------------------------------
//Sort_Raw_Data:
//	Sort the data from my_port.Serial_Read_Buffer
//  Check the Checksum for validity,
//	Store the data in m_Raw_Data_Buffer
int CDataProcessing::Sort_Check_Raw_Data(CSerialPort * my_port)
{
	int buffer_index = 0; //serial read buffer index
```

```
	BYTE plane_head[MAX_NUM_OF_SENSORS] = {HEAD_PLANE0,HEAD_PLANE1,HEAD_PLANE1};
	int x_y_z; //x=0, y=1, z=2
	int byte_MSB_LSB; //msb=1, lsb=0
	BYTE calculated_check_sum;
	BYTE transmitted_check_sum;
	int failures =0;
	int returncode;
	CString temp;
		temp.Format("ReadCur: %u", m_Data_Sort_Current->m_location);
//		m_hMyDlg->WriteToLog(temp);


	//Sort all the data collected during one read from the port
	for(int picture_index = 0;
	((picture_index * (int)(NUM_OF_SENSORS*8)) < my_port->m_bytes_read_from_port) //When we have sorted all the data collected
end
		&& (failures < (int)MAX_FAILURES); picture_index++)//End if we have failed more than the defined amount
			{

		//----Sort the Serial Data into the Raw data buffer-----
		for(int sensor_index = 1; sensor_index <= (int)NUM_OF_SENSORS; sensor_index++)
		{
			//Check the head char
			if(my_port->m_Serial_Read_Buffer[buffer_index] == (char) plane_head[sensor_index-1])
			{//Good Head

				buffer_index++;//increment past the 8-bit header
				byte_MSB_LSB = 1;//Byte number
				x_y_z = 0; //Axis number
				calculated_check_sum = 0;//PC calculated Check sum
```

```
//Parce the data in to the linked list
for(int loopend = buffer_index + 6; buffer_index < loopend; buffer_index++)
{	//Fill each picture buffer with all sensor data to complete one picture
	m_Data_Sort_Current->m_Raw_Data_Picture[sensor_index-1][x_y_z][byte_MSB_LSB] = my_port->m_Serial_Read_Buffer[buffer_index];
	calculated_check_sum += my_port->m_Serial_Read_Buffer[buffer_index]; //Continue creating the checksum

	//Set loop counters
	if(byte_MSB_LSB == 0)
	{
		byte_MSB_LSB = 1;
		x_y_z++;
	}
	else
	{
		byte_MSB_LSB--;
	}
}

//Check the transmitted check sum verses the calculated check sum
transmitted_check_sum = my_port->m_Serial_Read_Buffer[buffer_index];
if(transmitted_check_sum != calculated_check_sum)
{//BAD_CHECKSUM: Error they are not equal so dump packet
	returncode = BAD_CHECKSUM;
	failures++;
	m_Data_Sort_Current = m_Data_Sort_Current->m_last;
	m_hMyDlg->WriteToLog("BAD Checksum");
}

buffer_index++; //Increment to the next head
```

```
                }
                else
                {//BAD_HEAD:Error: Did not Find Head Char
                        returncode = BAD_HEAD;
                        failures++;
                        m_Data_Sort_Current = m_Data_Sort_Current->m_last;
                        m_hMyDlg->WriteToLog("BAD Head");
                }

        }
        //Successful Picture: point to the next location to fill a picture
        m_Data_Sort_Current = m_Data_Sort_Current->m_next;
    }

    if(failures >= (int)MAX_FAILURES)
    {
        return returncode;
    }

    return SUCCESS;
}

//---------------------------------------------------------------------------
//Write_Data_File:
//      Write any Data to A file
int CDataProcessing::Write_Data_File(int file_type)
{//Write a file with the sorted raw data

//      m_hMyDlg ->WriteToLog("Function: Write_Data_File");
        CString data_to_write;
        CString temp;
```

```
HANDLE hFile;
unsigned long wordswritten;
unsigned __int16 MSB_int16;
unsigned __int16 LSB_int16;
unsigned __int16 MSB_LSB_int16;
bool Write_To_File   = FALSE;
DWORD current_position;

if(file_type == RAW_DATA_FILE)
{       int     sensor_num;
        int x_y_z;

        //Set the last Writing pointer so that we get all of the data that is in the list currently
        m_Data_Write_End = m_Data_Sort_Current;

        temp.Format("WriteStart: %u, WriteEnd: %u", m_Data_Write_Current->m_location,m_Data_Write_End->m_location);
//      m_hMyDlg->WriteToLog(temp);


        while(m_Data_Write_Current != m_Data_Write_End)
        {//Get the data from the Raw_Data_Buffer and format it in a CString for printing
                sensor_num = 0;
                x_y_z = 0;

                while(sensor_num < ((int)NUM_OF_SENSORS))
                {
                        //Convert the data to Decimal
                        MSB_int16 = m_Data_Write_Current->m_Raw_Data_Picture[sensor_num][x_y_z][MSB];
                        MSB_int16 <<= 8;
                        LSB_int16 = m_Data_Write_Current->m_Raw_Data_Picture[sensor_num][x_y_z][LSB];
                        LSB_int16 &= 0x00FF; ;
```

```
                    MSB_LSB_int16 = MSB_int16 | LSB_int16;

                    //Format it as a string
                    temp.Format("%u,",MSB_LSB_int16);
                    data_to_write +=temp;

                    //Setup loop counters
                    if(x_y_z == 2)
                    {
                        sensor_num++;
                        x_y_z = 0;
                    }
                    else
                    {
                        x_y_z ++;
                    }

                }
                data_to_write +="\r\n,";

                //Increment to the next link in the list
                m_Data_Write_Current = m_Data_Write_Current->m_next;
                Write_To_File = TRUE;
            }

            //Set the current file handle
            hFile = hRaw_Data_File;

}

else if(file_type == BUTTON_DATA_FILE)
```

```
{
	int	sensor_num;
	int x_y_z;
	bool datapacket_flag = FALSE;
	CString xyz_gui_data[3];

	int zerobyte_count = 0;
	CString php_data_to_write;

	//Set the last Writing pointer so that we get all of the data that is in the list currently
	m_Data_Write_End = m_Data_Sort_Current;

	while((m_Data_Write_Current != m_Data_Write_End))
	{//Get the data from the Raw_Data_Buffer and format it in a CString for printing
		sensor_num = 0;
		x_y_z = 0;

		//Get the data status
		while(sensor_num < ((int)NUM_OF_SENSORS))
		{
			if(m_Data_Write_Current->m_Raw_Data_Picture[sensor_num][x_y_z][MSB]== 0)
			{
				zerobyte_count++;
			}
			if(m_Data_Write_Current->m_Raw_Data_Picture[sensor_num][x_y_z][LSB] == 0)
			{
				zerobyte_count++;
			}

			//Setup loop counters
			if(x_y_z == 2)
```

```
		{
			sensor_num++;
			x_y_z = 0;
		}
		else
		{
			x_y_z ++;
		}

}

if(zerobyte_count != (int)NUM_OF_SENSORS * 6)
{//Found Data
		 datapacket_flag = TRUE;
}
else
{//Found zeros
		datapacket_flag = FALSE;
}
 zerobyte_count = 0;



//Check Data and Button Conditions

if(m_write_button.button_pressed==FALSE && datapacket_flag == TRUE )
{
		//m_hMyDlg->WriteToLog("BP==F, dF==t");
		//found transition
		m_write_button.button_pressed=TRUE;
		m_write_button.data_count++;
```

```
}
else if(m_write_button.button_pressed == TRUE && datapacket_flag == FALSE )
{//m_hMyDlg->WriteToLog("BP==t, dF==f");
	//check to see if we are under 255 packets if so then dump
	//else this is the end of the packet and we need to write to disk
	if(m_write_button.data_count < (int)BUTTON_DATA_COUNT)
	{//m_hMyDlg->WriteToLog("Less than 255 packets");

	temp.Format("data_count: %u",m_write_button.data_count );
		//CLEAR ALL VARS
		m_write_button.button_pressed = FALSE;
		m_write_button.data_count = 0;


		//Clear the sum location
		for(x_y_z = 0; x_y_z < 3; x_y_z++)
		{
			//Clear the memory location for the next time around
			m_write_button.data_sum[x_y_z] = 0;
		}

	}
	else
	{	//m_hMyDlg->WriteToLog("Create Sum");
		Write_To_File = TRUE;

		//get the data point to write
		for(x_y_z = 0; x_y_z < 3; x_y_z++)
		{
			//divide by the number of data points collected
```

```
                m_write_button.data_sum[x_y_z]  = (m_write_button.data_sum[x_y_z] /
((int)(BUTTON_DATA_COUNT - 1)));

                //Format the mean of the data as a string
                temp.Format("%.0f,",m_write_button.data_sum[x_y_z]);
                data_to_write +=temp;
                xyz_gui_data[x_y_z] = temp;
                php_data_to_write+= temp;
                //Clear the memory location for the next time around
                m_write_button.data_sum[x_y_z] = 0;
        }

        temp.Format("Button:%u, Row: %u",m_write_button.num_of_button_clicks,m_write_button.row_num);
        //m_hMyDlg->WriteToLog(temp);
        m_hMyDlg->m_ave_button_row = temp;
        m_hMyDlg->m_x_ave = xyz_gui_data[0];
        m_hMyDlg->m_y_ave = xyz_gui_data[1];
        m_hMyDlg->m_z_ave = xyz_gui_data[2];
        m_hMyDlg->m_Update_Gui_Vars = TRUE;

        //Write Button file
        current_position = SetFilePointer(hPHP_Data_File,0,NULL,FILE_CURRENT);
        current_position = ((-1)* current_position);
        SetFilePointer(hPHP_Data_File,(long)current_position,NULL,FILE_CURRENT);

    if(!WriteFile(hPHP_Data_File,php_data_to_write.GetBuffer(1),php_data_to_write.GetLength(),&wordswritten,NULL))
        {
                //return COULD_NOT_WRITE_TO_FILE;
        }

        php_data_to_write.Empty();
```

```
                Beep((DWORD) 0x1000,300);

                m_write_button.num_of_button_clicks++;

                if(m_write_button.num_of_button_clicks == 33)
                {       //End of row beep
                        data_to_write +="\r\n";
                        m_write_button.num_of_button_clicks = 0;
                        m_write_button.row_num++;
                        Beep((DWORD) 0x500,600);
                }

                //CLEAR ALL VARS
                m_write_button.button_pressed = FALSE;
                m_write_button.data_count = 0;

        }
}
else if(m_write_button.button_pressed==TRUE && datapacket_flag == TRUE )
{//m_hMyDlg->WriteToLog("BP==t, dF==t");
        //Found a data point

        //increment data count
        m_write_button.data_count++;

        //get the data point to write
                sensor_num = 0;
                x_y_z = 0;
```

```
		while(sensor_num < ((int)NUM_OF_SENSORS))
		{
			//Convert the data to Decimal
			MSB_int16 = m_Data_Write_Current->m_Raw_Data_Picture [sensor_num][x_y_z][MSB];
			MSB_int16 <<= 8;
			LSB_int16 = m_Data_Write_Current->m_Raw_Data_Picture [sensor_num][x_y_z][LSB];
			LSB_int16 &= 0x00FF; ;
			MSB_LSB_int16 = MSB_int16 | LSB_int16;

			//Add the data to the running sum for each sensor;
			m_write_button.data_sum[x_y_z] += MSB_LSB_int16;

			//Setup loop counters
			if(x_y_z == 2)
			{
				sensor_num++;
				x_y_z = 0;
			}
			else
			{
				x_y_z ++;
			}

		}
}


//Increment to the next link in the list
m_Data_Write_Current =  m_Data_Write_Current->m_next;
```

```
			}//Close While

			//Set the current file handle
			hFile = hTest_Data_File;
		}//close else if

		//Clear the data collection event so that it can be signaled again
		ResetEvent(SendWriteDataObject);

		if(Write_To_File == TRUE)
		{
			if(!WriteFile(hFile,data_to_write.GetBuffer(1),data_to_write.GetLength(),&wordswritten,NULL))
			{
				return COULD_NOT_WRITE_TO_FILE;
			}

		}
		return SUCCESS;
}

//-------------------------------------------------------------------------------
//Determine_Pea_Output_Location:
void CDataProcessing::Determine_Pea_XYZ_Location()
{
		CRaw_Data * current_raw_data;
		int MSB_int;
		int LSB_int;
		int MSB_LSB_int;
		int sensor_num = 0;
		int x_y_z = 0;
		CString logwriting;
```

```
	//Get the most current raw data for display
	current_raw_data = m_Data_Sort_Current->m_last;

	//Converte the data to decimal so that it can be displayed properly
	while(sensor_num <((int) NUM_OF_SENSORS))
	{
		//Convert the data to Decimal
		MSB_int = current_raw_data->m_Raw_Data_Picture[sensor_num][x_y_z][MSB];
		MSB_int <<= 8;
		LSB_int = current_raw_data->m_Raw_Data_Picture[sensor_num][x_y_z][LSB];
		LSB_int &= 0x000000FF;
		MSB_LSB_int = MSB_int | LSB_int;


#if DEBUG_MODE == TRUE
		ASSERT(DEBUG_MODE == TRUE);
		//We will plot 3 pea for only one sensor, one for each axis of one plane
		//set up for DEBUG Mode
		//this will always plot the last sensor hooked up if mulitple sensors are attached
		m_current_pea_xyz[x_y_z].y = MSB_LSB_int;
#endif

	//Setup loop counters
		if(x_y_z == 2)
		{
			sensor_num++;
			x_y_z = 0;
		}
		else
		{
```

```
                        x_y_z ++;
                }
        }
}

//-------------------------------------------------------------------------
//Determine_Pea_Pixel_Location:
void CDataProcessing::Determine_Pea_Pixel_Location()
{

#if DEBUG_MODE == TRUE

        int pixel_y;
        double normalized_y;
        double y_copy;
        double divisor = (double)ANALOG_SCALE;
        //Init the initial positon in the x direction to zero
        int init_left_offset = m_clipping_rect.Width()/7;

        for(int rect_num =0; rect_num < 3; rect_num++)
        {       y_copy = m_current_pea_xyz[rect_num].y;

                normalized_y = y_copy /divisor;
                pixel_y = normalized_y * m_clipping_rect.Height();
                if(pixel_y == 0)
                {//the minimum rect hight must be 2 pixels or it will not be drawn
                        pixel_y = 2;
                }

                m_current_pea_pixel[rect_num].m_rect.left = init_left_offset +((( m_clipping_rect.Width()/7) *2 )*rect_num);
                m_current_pea_pixel[rect_num].m_rect.top = (m_clipping_rect.Height()-pixel_y);
```

```
			m_current_pea_pixel[rect_num].m_rect.right = ((m_clipping_rect.Width()/7)*2) + (((m_clipping_rect.Width()/7)*2)* rect_num);
			m_current_pea_pixel[rect_num].m_rect.bottom = m_clipping_rect.bottom;
		}
#endif

}


void CDataProcessing::Close_Data_Files()
{

		//--------Data File Clean Up------------
		if(hRaw_Data_File != NULL)
		{
			CloseHandle(hRaw_Data_File);
			hRaw_Data_File = NULL;
		}

		//--------Data File Clean Up------------
		if(hTest_Data_File != NULL)
		{
			CloseHandle(hTest_Data_File);
			hTest_Data_File = NULL;
		}
		//--------Data File Clean Up------------
		if(hPHP_Data_File != NULL)
		{
			CloseHandle(hPHP_Data_File);
			hPHP_Data_File = NULL;
		}
```

```
/*
if(hLog != NULL)
	{
		CloseHandle(hLog);
		hLog = NULL;
	}
*/
}
```

//Serial_Comm.cpp

```
/************************************************************************
This code cannot be copied, modified, nor used without giving credit to the original authors.
The authors give no warranty as to the workings of this program.
Also the text included here must be left with every complete or partial code replication.

Authors:
        Andrea L.M. Baker, Wojciech Krajewski and Daniel I. Wallance
Completion Date:
        Friday October 17, 2003.
Designed For:
        This code was written for AMT Ireland at the University of Limerick, Ireland during the fulfillment
        of a Major Qualifying Project for Worcester Polytechnic Institute, Worcester, MA USA.
************************************************************************/

#include "stdafx.h"
#include "MQP_3d.h"
#include "Serial_Comm.h"
#include "Data_Processing.h"
#include "MQP_3dDlg.h"

//#include "Display.h"

#include <time.h>

//---------------------------------------------------------------------------
//Configure_Serial_Port:
int CSerialPort::Configure_Serial_Port()
{
        int returncode;
```

```
if(m_hComm == NULL)
{
	//Get a handle to the port
	m_hComm = CreateFile(m_gszPort.GetBuffer(1),
				GENERIC_READ | GENERIC_WRITE,
				0,
				0,
				OPEN_EXISTING,
				0,
				0);

	//Check if the handle recived is valid
	if (m_hComm == INVALID_HANDLE_VALUE)
	{
		m_hComm = NULL;

		return INVALID_HANDLE;
	}
	else
	{//The Handle Was valid so Configure the Port for our uses

		//ToDo: Set input and Output buffer size //SetupComm(hComm,

		//Get the current Comm Configureation
		returncode = GetCommState(m_hComm,&m_dcb);//ToDo:Check returncode for errors(Nonzero ==Success)

		// Fill in the DCB: 8 data bits, no parity, and 1 stop bit.
		m_dcb.BaudRate = BAUDRATE;     // set the baud rate
		m_dcb.ByteSize = 8;            // data size, xmit, and rcv
		m_dcb.Parity = NOPARITY;       // no parity bit
		m_dcb.StopBits = ONESTOPBIT;   // one stop bit
```

```
                        //Turn off all handshaking parameters, always allow data in!
                        m_dcb.fDsrSensitivity = FALSE;
                        m_dcb.fOutxCtsFlow = FALSE;
                        m_dcb.fOutxDsrFlow = FALSE;
                        m_dcb.fDtrControl = DTR_CONTROL_DISABLE;
                        m_dcb.fOutX = FALSE;
                        m_dcb.fInX = FALSE;
                        m_dcb.fRtsControl = RTS_CONTROL_DISABLE;

                        //Load our new Comm Configureation
                        returncode = SetCommState(m_hComm,&m_dcb);//ToDo:Check returncode for errors(Nonzero ==Success)

                        //Set the comm timeout for reading and writing
                        GetCommTimeouts(m_hComm,&m_commtimeouts);
                        m_commtimeouts.ReadIntervalTimeout = (int)READ_INTERVAL_TIMEOUT;
                        m_commtimeouts.WriteTotalTimeoutMultiplier = (int)WRITE_TOTAL_TIMEOUT_MULTIPLIER;
                        SetCommTimeouts(m_hComm,&m_commtimeouts);

                        //Configuration Complete
                        return SUCCESS;
                }
        }
        else
        {//ERROR
                return PORT_ALREADY_OPEN;
        }
}

//---------------------------------------------------------------------------
//Handshake_Pic:
```

```
int CSerialPort::Handshake_Pic(char cmd)
{
	DWORD bytes_tobe_read = 0;
	DWORD error_code;//ToDo: Check comm Errors returned
	DWORD wordswritten;
	DWORD byteswritten;
	bool Handshake = TRUE;
	int Total_Bytes_tobe_Read =0;
	int bytes_in_buffer = 0;
	int bytes_needed_to_complete_picture = 0;

	//----------------Clear_Serial_Port--------------
	Clear_Serial_Port();


	//----------------Send the Echo & ADC Command----------

		//Clear the Read Buffer for one char
		m_Serial_Read_Buffer[0] = (char) 0x00;
		WriteFile(m_hComm,&cmd,sizeof(cmd),&wordswritten,NULL);
		Sleep(100);
		//ToDo: This needs to have a time out otherwise it makes no sense!
		//WaitCommEvent(m_hComm,&m_event_type,NULL);

		ClearCommError(m_hComm,&error_code,&m_comm_stats);
		bytes_tobe_read = 1;
		//bytes_tobe_read = m_hMyPort->m_comm_stats.cbInQue;
		ReadFile(m_hComm, &m_Serial_Read_Buffer, bytes_tobe_read, &byteswritten, NULL);

		//****Set up the Serial Port event mask to wait for any char in the buffer.
		//SetCommMask(m_hComm,EV_RXCHAR);
```

```
		if(!(m_Serial_Read_Buf fer[0] == cmd))
		{	//Echo Unsuccessful
			return UNSUCCESSFUL_HANDSHAKE;
			//The handle to the serial port needs to be cleaned up
		}

		return SUCCESS;
}


//---------------------------------------------------------------------------
//Clear_Serial_Port:
int CSerialPort::Clear_Serial_Port()
{

	DWORD bytes_tobe_read = 0;
	DWORD error_code;//ToDo: Check comm Errors returned
	int Total_Bytes_tobe_Read =0;
	int bytes_in_buffer = 0;
	DWORD byteswritten;


	//-----------------Clean The Port--------------
	char cmd =(char) PIC_ECHO_CMD; //Echo

	//Send the command 2 times
	for(int i = 0; i<=(int)TIMES_TO_SEND_PIC_CLEAR_CMD; i++)
	{
		Send_Serial_Command(cmd);
	}
```

```
	//Wait for the PIC to respond
	Sleep(1000);

	//Clear the port
	do{
		//Clear the Errors and get the current num of chars in the buffer
		ClearCommError(m_hComm,&error_code,&m_comm_stats);
		bytes_tobe_read = m_comm_stats.cbInQue;

		//Read the serial port put result into my_port.Serial_Read_Buffer
		ReadFile(m_hComm, &m_Serial_Read_Buffer, bytes_tobe_read, &byteswritten, NULL);

		//****Set up the Serial Port event mask to wait for any char in the buffer.
		SetCommMask(m_hComm,EV_RXCHAR);

		m_bytes_read_from_port = byteswritten;

	}while(bytes_tobe_read >0);


	return SUCCESS;
}


//---------------------------------------------------------------------------
//Read_From_Port:
int CSerialPort::Read_From_Port(CDataProcessing * hMyData)
{

	//double double_temp;
```

```
	int num_of_full_pictures_in_buffer;
	double d_cbInQue;
	DWORD dwErrors;
	DWORD bytes_tobe_read = 0;
	DWORD byteswritten;
	bool Read_Done = FALSE;
	CString temp;
	CString write;
//	m_hMyDlg->WriteToLog("Read From Port");
	do{
//		m_hMyDlg->WriteToLog("Read From Port:Enter do");
		//Wait for one character
		WaitCommEvent(m_hComm,&m_event_type,NULL);//This will hang if data stops being transmitted!!

		//Get the num of chars in the buffer
		ClearCommError(m_hComm,&dwErrors,&m_comm_stats);

		if((dwErrors & CE_IOE) || (dwErrors & CE_OOP) || (dwErrors & CE_PTO) || dwErrors & CE_MODE || dwErrors &
CE_BREAK || ( dwErrors & CE_FRAME) || (dwErrors & CE_RXOVER) || (dwErrors & CE_TXFULL) || (dwErrors & CE_OVERRUN) ||
(dwErrors & CE_RXPARITY) || (dwErrors & CE_DNS))
		{
			m_hMyDlg->WriteToLog("Comm Error on Read");

		}
		//If their are enough chars to read then read if not than wait.
		if(m_comm_stats.cbInQue > ((int)TOTAL_BYTES_IN_PICTURE))
		{
			//calculate the number of bytes to read depending upon the number of bytes in one picture.
			d_cbInQue = m_comm_stats.cbInQue;
			num_of_full_pictures_in_buffer = d_cbInQue / ((double)TOTAL_BYTES_IN_PICTURE);//cast as an int to just get the
whole number of pictures
```

```
                        bytes_tobe_read = ((int)TOTAL_BYTES_IN_PICTURE )* num_of_full_pictures_in_buffer;

                        //Read the serial port put result into m_ Serial_Read_Buffer
                        //m_hMyDlg->WriteToLog("Function: ReadFile");
                        ReadFile(m_hComm, &m_Serial_Read_Buffer, bytes_tobe_read, &byteswritten, NULL);
                        //m_hMyDlg->WriteToLog("FunctionEND: ReadFile");
                        m_bytes_read_from_port = byteswritten;
                        Read_Done = TRUE;
                        temp.Format("BytesRead: %d",byteswritten);
//                      m_hMyDlg->WriteToLog(temp);
                }
                //Set an event tobe triggered on a char entering the buffer
                SetCommMask(m_hComm,EV_RXCHAR);

        }while(Read_Done == FALSE);

        return SUCCESS;
}


//-------------------------------------------------------------------------
//Send_Serial_Command:
int CSerialPort::Send_Serial_Command(char cmd)
{
                DWORD wordswritten;

        if(m_hComm != NULL)
        {
                //Write To The Port
                WriteFile(m_hComm,&cmd,sizeof(cmd),&wordswritten,NULL);
                //ToDo: Check the number of words written vrs size of writebuf
```

```
	}
	else
	{//ERROR
		return PORT_NOT_OPEN;
	}
	return SUCCESS;

}

//---------------------------------------------------------------------------
//Close_Serial_Port:
int CSerialPort::Close_Serial_Port()
{
	if(m_hComm !=NULL)
	{
		DWORD error_code;//ToDo: Check comm Errors returned
		//Clear errors on the port to free it
		ClearCommError(m_hComm,&error_code,&m_comm_stats);

		//Tell the PIC to stop transmitting
		Send_Serial_Command('A');
		Sleep(500);

		//Close the handle to the port
		CloseHandle(m_hComm);
		m_hComm = NULL;
		PortConfigured = FALSE;


		//ToDo: Note to the user that the port has been closed
	}
```

```
        return TRUE;
}
```

// MQP_3d.h :

```
/*********************************************************************
This code cannot be copied, modified, nor used without giving credit to the original authors.
The authors give no warranty as to the workings of this program.
Also the text included here must be left with every complete or partial code replication.

Authors:
        Andrea L.M. Baker, Wojciech Krajewski and Daniel I. Wallance
Completion Date:
        Friday October 17, 2003.
Designed For:
        This code was written for AMT Ireland at the University of Limerick, Ireland during the fulfillment
        of a Major Qualifying Project for Worcester Polytechnic Institute, Worcester, MA USA.
*********************************************************************/

//

#if !defined(AFX_MQP_3D_H__F60E51F8_D16D_41F6_9E27_4FF025FF6E54__INCLUDED_)
#define AFX_MQP_3D_H__F60E51F8_D16D_41F6_9E27_4FF025FF6E54__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"           // main symbols

/////////////////////////////////////////////////////////////////////////////
```

```
// CMQP_3dApp:
// See MQP_3d.cpp for the implementation of this class
//
class CSerialPort;
class CMQP_3dDlg;
class CMQP_3dApp : public CWinApp
{
public:
	CMQP_3dApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CMQP_3dApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CMQP_3dApp)
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};



//----------GLOBAL DEFINES----------
```

```
//****Software & Hardware Configuration Vars***
#define DEBUG_MODE                          TRUE
#define MAX_NUM_OF_SENSORS                  3
#define NUM_OF_SENSORS                      1
#define READ_INTERVAL_TIMEOUT               500
#define WRITE_TOTAL_TIMEOUT_MULTIPLIER      30
#define BAUDRATE                            CBR_38400
#define MAX_FAILURES                        4
#define TIMES_TO_SEND_PIC_CLEAR_CMD         1
#define TOTAL_BYTES_IN_PICTURE              (NUM_OF_SENSORS * 8 )
#define ANALOG_SCALE                        1024
#define MSB                                 1
#define LSB                                 0
#define GOOD_DATA_COUNT                     20
#define BUTTON_DATA_COUNT                   255
#define NUM_T0_DISPLAY_BEFORE_CLEAR         300

//Window Levels
#define TOP                         1
#define BOTTOM                      0


//Head Packet Identifiers
#define HEAD_PLANE0                         0x00
#define HEAD_PLANE1                         0x03
#define HEAD_PLANE2                         0x06

//PC to PIC Commands
#define PIC_ECHO_CMD                'A'
#define PIC_ECHO_ADCSTART_CMD       'B'
#define PIC_ECHO_BUTTONSTART_CMD    'C'
```

```
//Number of pictures to collect per read and write
#if NUM_OF_SENSORS == MAX_NUM_OF_SENSORS
	#define NUM_OF_PICTURES_TO_COLLECT		150 //The serial read hardware buffer is 4096 bytes
#endif

#if NUM_OF_SENSORS == (MAX_NUM_OF_SENSORS -1)
	#define NUM_OF_PICTURES_TO_COLLECT		250 //The serial read hardware buffer is 4096 bytes
#endif

#if NUM_OF_SENSORS == (MAX_NUM_OF_SENSORS -2)
	#define NUM_OF_PICTURES_TO_COLLECT		500 //The serial read hardware buffer is 4096 bytes
#endif

#define NUM_OF_RAW_DATA_LINKS_IN_LIST	4000
#define NUM_TO_DISPLAY_BEFORE_WRITE	50



//Timer IDs and Delays(ms)
#define IDT_UPDATE			0
#define IDT_CLEAR			1
#define DISPLAY_TIMER_DELAY		30
#define CLEAR_TIMER_DELAY		1000


//****Return Codes****
#define SUCCESS				0
#define PORT_NOT_OPEN			1
#define INVALID_HANDLE			2
#define PORT_ALREADY_OPEN		3
```

```
#define UNSUCCESSFUL_HANDSHAKE          4
#define BAD_HEAD                        5
#define COULD_NOT_CREATE_DIRECTORY      6
#define COULD_NOT_CREATE_FILE           7
#define COULD_NOT_WRITE_TO_FILE         8
#define BAD_CHECKSUM                    9
#define UNSUCCESSFUL_READ_FROM_PORT     10
#define NO_DATA_TO_READ                 11
#define NO_FULL_FRAME_FROM_PIC          12
#define BREAK_ON_INPUT                  13
#define ERROR_ON_PORT                   14
#define PORT_OVERFLOW_ERROR             15

//****Valid File Types****
#define RAW_DATA_FILE                   0
#define BUTTON_DATA_FILE                1

//****GUI Program Mode Defines*****
#define STANDARD_MODE                   0
#define BUTTON_MODE                     1

//****System Events****
extern HANDLE SendPortNotInUseEvent;
extern HANDLE SendWriteThreadWaiting;
extern HANDLE SendWriteDataObject;


//extern HANDLE hLog;
//****SIGNAL FLAGS****
extern bool PortConfigured;
extern bool Program_Active;
```

```
extern bool Display_Position;
extern int Display_Count;
extern int Clear_Count;



////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MQP_3D_H__F60E51F8_D16D_41F6_9E27_4FF025FF6E54__INCLUDED_)
```

// MQP_3dDlg.h

```
/************************************************************************
This code cannot be copied, modified, nor used without giving credit to the original authors.
The authors give no warranty as to the workings of this program.
Also the text included here must be left with every complete or partial code replication.

Authors:
        Andrea L.M. Baker, Wojciech Krajewski and Daniel I. Wallance
Completion Date:
        Friday October 17, 2003.
Designed For:
        This code was written for AMT Ireland at the University of Limerick, Ireland during the fulfillment
        of a Major Qualifying Project for Worcester Polytechnic Institute, Worcester, MA USA.
************************************************************************/


#if !defined(AFX_MQP_3DDLG_H__6B26D63A_5E47_4644_96A4_F008EF0A3A07__INCLUDED_)
#define AFX_MQP_3DDLG_H__6B26D63A_5E47_4644_96A4_F008EF0A3A07__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////////
// CMQP_3dDlg dialog
class WriteThread;
class ReadThread;
class CSerialPort;
class CDataProcessing;
class CMQP_3dDlg : public CDialog
{
```

```
// Construction
public:
	CMQP_3dDlg(CWnd* pParent = NULL);	// standard constructor
	CSerialPort m_MyPort;
	CDataProcessing m_MyData;
	void WriteToLog(CString text);

	int m_current_wnd_level;
	int m_last_wnd_level;
	int m_Update_Gui_Vars;
	bool m_Update_Display;

// Dialog Data
	//{{AFX_DATA(CMQP_3dDlg)
	enum { IDD = IDD_MQP_3D_DIALOG };
	CEdit	m_z_ave_control;
	CEdit	m_y_ave_control;
	CEdit	m_x_ave_control;
	CEdit	m_ave_button_row_control;
	CListBox	m_log_list_control;
	CString	m_comport;
	CString	m_log_list;
	UINT	m_displayx;
	UINT	m_displayy;
	UINT	m_displayz;
	BOOL	m_datafiles;
	int		m_mode_select;
	CString	m_x_ave;
	CString	m_y_ave;
	CString	m_z_ave;
	CString	m_ave_button_row;
```

```
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CMQP_3dDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	HICON m_hIcon;

	ReadThread * m_hReadThread;
	WriteThread *m_hWriteThread;

	// Generated message map functions
	//{{AFX_MSG(CMQP_3dDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
	afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
	afx_msg void OnTimer(UINT nIDEvent);
	afx_msg void OnRun();
	virtual void OnOK();
	afx_msg void OnStop();
	afx_msg void OnStandardMode();
	afx_msg void OnButtonMode();
	afx_msg void OnClearList();
	//}}AFX_MSG
```

```
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MQP_3DDLG_H__6B26D63A_5E47_4644_96A4_F008EF0A3A07__INCLUDED_)
```

// ReadThread.h

```
/*********************************************************************
This code cannot be copied, modified, nor used without giving credit to the original authors.
The authors give no warranty as to the workings of this program.
Also the text included here must be left with every complete or partial code replication.

Authors:
        Andrea L.M. Baker, Wojciech Krajewski and Daniel I. Wallance
Completion Date:
        Friday October 17, 2003.
Designed For:
        This code was written for AMT Ireland at the University of Limerick, Ireland during the fulfillment
        of a Major Qualifying Project for Worcester Polytechnic Institute, Worcester, MA USA.
*********************************************************************/

#if !defined(AFX_READTHREAD_H__B3412564_8EA7_45BF_BC8A_D890EB87304A__INCLUDED_)
#define AFX_READTHREAD_H__B3412564_8EA7_45BF_BC8A_D890EB87304A__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////////
// ReadThread
class CSerialPort;
class CDataProcessing;
class CMQP_3dDlg;
class ReadThread : public CWinThread
{
        DECLARE_DYNCREATE(ReadThread)
protected:
```

```
        ReadThread();           // protected constructor used by dynamic creation
        void KillThread();

// Attributes
public:
        HANDLE m_hEventKill; //InitInstance member fcns stop when m_hEventKill is signalled
        bool    Quit( void ) { return(CWinThread::PostThreadMessage( WM_QUIT, NULL, NULL ) ? true : false ); }

// Operations
public:
        void RegisterMyPortHandle(CSerialPort *my_port) {m_hMyPort = my_port;}
        void RegisterMyDataHandle(CDataProcessing *my_data) {m_hMyData = my_data;}
        void RegisterDialogHandle(CMQP_3dDlg *my_dlg) {m_hMyDlg = my_dlg;}
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(ReadThread)
        public:
        virtual BOOL InitInstance();
        virtual int ExitInstance();
        //}}AFX_VIRTUAL

// Implementation
protected:
        virtual ~ReadThread();
        CSerialPort * m_hMyPort;
        CDataProcessing * m_hMyData;
        CMQP_3dDlg *m_hMyDlg;

        // Generated message map functions
        //{{AFX_MSG(ReadThread)
                // NOTE - the ClassWizard will add and remove member functions here.
```

```
	//}}AFX_MSG

	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_READTHREAD_H__B3412564_8EA7_45BF_BC8A_D890EB87304A__INCLUDED_)
```

// WriteThread.h

```
/*********************************************************************
This code cannot be copied, modified, nor used without giving credit to the original authors.
The authors give no warranty as to the workings of this program.
Also the text included here must be left with every complete or partial code replication.

Authors:
        Andrea L.M. Baker, Wojciech Krajewski and Daniel I. Wallance
Completion Date:
        Friday October 17, 2003.
Designed For:
        This code was written for AMT Ireland at the University of Limerick, Ireland during the fulfillment
        of a Major Qualifying Project for Worcester Polytechnic Institute, Worcester, MA USA.
*********************************************************************/

#if !defined(AFX_WRITETHREAD_H__2D74348C_64E7_4421_BC54_63EBA76B0271__INCLUDED_)
#define AFX_WRITETHREAD_H__2D74348C_64E7_4421_BC54_63EBA76B0271__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////////
// WriteThread thread
class CDataProcessing;
class WriteThread : public CWinThread
{
        DECLARE_DYNCREATE(WriteThread)
protected:
        WriteThread();          // protected constructor used by dynamic creation
        void KillThread();
```

```
// Attributes
public:
	HANDLE m_hEventKill; //InitInstance member fcns stop when m_hEventKill is signalled
	bool	Quit( void ) { return(CWinThread::PostThreadMessage( WM_QUIT, NULL, NULL ) ? true : false ); }

// Operations
public:
	void RegisterMyPortHandle(CSerialPort *my_port) {m_hMyPort = my_port;}
	void RegisterMyDataHandle(CDataProcessing *my_data) {m_hMyData = my_data;}
	void RegisterDialogHandle(CMQP_3dDlg *my_dlg) {m_hMyDlg = my_dlg;}

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(WriteThread)
	public:
	virtual BOOL InitInstance();
	virtual int ExitInstance();
	//}}AFX_VIRTUAL

// Implementation
protected:
	virtual ~WriteThread();
	CDataProcessing * m_hMyData;
	CMQP_3dDlg *m_hMyDlg;
	CSerialPort * m_hMyPort;

	// Generated message map functions
	//{{AFX_MSG(WriteThread)
		// NOTE - the ClassWizard will add and remove member functions here.
	//}}AFX_MSG
```

```
	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_WRITETHREAD_H__2D74348C_64E7_4421_BC54_63EBA76B0271__INCLUDED_)
```

//Data_Processing.h

```
/*************************************************************************
This code cannot be copied, modified, nor used without giving credit to the original authors.
The authors give no warranty as to the workings of this program.
Also the text included here must be left with every complete or partial code replication.

Authors:
        Andrea L.M. Baker, Wojciech Krajewski and Daniel I. Wallance
Completion Date:
        Friday October 17, 2003.
Designed For:
        This code was written for AMT Ireland at the University of Limerick, Ireland during the fulfillment
        of a Major Qualifying Project for Worcester Polytechnic Institute, Worcester, MA USA.
*************************************************************************/

class CPea_Location_Pixel
{
public:
        CRect m_rect;
        CPea_Location_Pixel * m_next;
};

class CPea_Location_XYZ
{
public:
        int x;
        int y;
        int z;

};
```

```
class CRaw_Data
{
public:
	char m_Raw_Data_Picture[NUM_OF_SENSORS][3][2];
	CRaw_Data * m_next;
	CRaw_Data * m_last;
	int m_location;
};

class CButton_Data
{
public:
	bool button_pressed;
	int data_count;
	int num_of_button_clicks;
	int row_num;
	double data_sum[3];
};


class CDataProcessing
{
public:

	//CRaw_Data Linked list pointers
	CRaw_Data * m_Data_Sort_Current;
	CRaw_Data * m_Data_Write_End;
	CRaw_Data * m_Data_Write_Current;

	//Display rectangles
	CRect m_clipping_rect;
```

```
        CRect m_outline_rect;

        //GUI VARS
        BOOL m_Write_File_Flag;
        int m_mode;

        CButton_Data m_write_button;

#if DEBUG_MODE == TRUE
        //designed to test one plane
        CPea_Location_XYZ m_current_pea_xyz[3];
        CPea_Location_Pixel m_current_pea_pixel[3];
        CPea_Location_Pixel m_past_pea_pixel[3];
#else
        //designed for multiple sensors
        CPea_Location_XYZ m_current_pea_xyz;
        CPea_Location_Pixel m_current_pea_pixel;
#endif

        char m_foldername[20];
        char m_filename[50];
        char m_timestamp[13];

        //File Handles
        HANDLE hRaw_Data_File;
        HANDLE hTest_Data_File;
        HANDLE hPHP_Data_File;

        //Main Functions
        void Create_Raw_Data_Linked_List();
        void Free_Raw_Data_Linked_List();
```

```
	int Init_Folder_Files();
	int Sort_Check_Raw_Data(CSerialPort * my_port);//Sort the data from my_port.Serial_Read_Buffer,Check the Checksum for validity,
Store the data in my_port.m_Raw_Data_Buffer
	int Write_Data_File(int file_type);//Write a file with the sorted raw data
	void Determine_Pea_Pixel_Location();
	void Determine_Pea_XYZ_Location();

	//Support Functions
	int Create_Folder();
	HANDLE Create_File(HANDLE hFile);
	void Create_Time_Stamp();
	void Close_Data_Files();
	void RegisterDialogHandle(CMQP_3dDlg *my_dlg) {m_hMyDlg = my_dlg;}

private:
CMQP_3dDlg * m_hMyDlg;
};
```

//Serial_Comm.h

```
/********************************************************************
This code cannot be copied, modified, nor used without giving credit to the original authors.
The authors give no warranty as to the workings of this program.
Also the text included here must be left with every complete or partial code replication.

Authors:
        Andrea L.M. Baker, Wojciech Krajewski and Daniel I. Wallance
Completion Date:
        Friday October 17, 2003.
Designed For:
        This code was written for AMT Irela nd at the University of Limerick, Ireland during the fulfillment
        of a Major Qualifying Project for Worcester Polytechnic Institute, Worcester, MA USA.
********************************************************************/

class CMQP_3dView;
class CMQP_3dDlg;
class COrganizedRawData;
class CDataProcessing;
class CSerialPort
{
public:
        HANDLE m_hComm;
        DCB m_dcb;
        CString m_gszPort;
        COMSTAT m_comm_stats;
        COMMTIMEOUTS m_commtimeouts;
        DWORD m_event_type;
        char m_Serial_Read_Buffer[4097];
        int m_bytes_read_from_port;
        //OVERLAPPED m_overlapped;
```

```
        //Main Functions
        int Configure_Serial_Port();
        int Handshake_Pic(char cmd);
        int Send_Serial_Command(char cmd);
        int Read_From_Port(CDataProcessing *mydata);
        int Close_Serial_Port();

        //Support Functions
        int C lear_Serial_Port();
        void RegisterDialogHandle(CMQP_3dDlg *my_dlg) {m_hMyDlg = my_dlg;}

        //Suport Vars
        LONG m_overflow_error;

private:
CMQP_3dDlg * m_hMyDlg;


};
```

//Resource.h

```
//{{NO_DEPENDENCIES}}
/************************************************************************
This code cannot be copied, modified, nor used without giving credit to the original authors.
The authors give no warranty as to the workings of this program.
Also the text included here must be left with every complete or partial code replication.

Authors:
        Andrea L.M. Baker, Wojciech Krajewski and Daniel I. Wallance
Completion Date:
        Friday October 17, 2003.
Designed For:
        This code was written for AMT Ireland at the University of Limerick, Ireland during the fulfillment
        of a Major Qualifying Project for Worcester Polytechnic Institute, Worcester, MA USA.
************************************************************************/


// Used by MQP_3d.rc
//
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDS_ABOUTBOX                    101
#define IDD_MQP_3D_DIALOG               102
#define IDR_MAINFRAME                   128
#define IDB_BITTest                     129
#define IDC_RUN                         1000
#define IDC_COMPORT                     1001
#define IDC_STOP                        1002
#define IDC_LIST1                       1003
#define IDC_LOG_LIST                    1003
#define IDC_BUTTON1                     1004
```

```
#define IDC_STOP2                       1004
#define IDC_CLEAR_LIST                  1004
#define IDC_X                           1006
#define IDC_Y                           1007
#define IDC_Z                           1008
#define IDC_OUTLINE                     1009
#define IDC_SHUTDOWN2                   1010
#define IDC_X_AVE                       1010
#define IDC_DATAFILES                   1011
#define IDC_CLEAN_ZERO_MODE             1012
#define IDC_Y_AVE                       1012
#define IDC_Z_AVE                       1013
#define IDC_STANDARD_MODE               1015
#define IDC_BUTTON_MODE                 1016
#define IDC_XYZ_AVE_BOX                 1018
#define IDC_Z_TEXT                      1019
#define IDC_Y_TEXT                      1020
#define IDC_X_TEXT                      1021
#define IDC_AVE_BUTTON_ROW              1022

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        131
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1023
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```