Manipulator Arm
Control System for Characterizing
Large-Area Photosensors

Harish Kugel

Project Sponsor:
Dr. Hirohisa Tanaka

Project Number: 1271

Applied Science 479
Engineering Physics Project Laboratory
The University of British Columbia
April 16, 2013

Executive Summary

This project is a continuation of a previous ENPH 479 project, completed by Aaron Zimmer and Nils Smit. The original project, for Dr. Hirohisa Tanaka of TRIUMF, sought to characterize the optical properties of a new acrylic shielding used on PMTs employed by the Super-K neutrino observatory.

To perform this characterization, a PMT Testing Rig was constructed at TRIUMF. This Testing Rig consists of two robotic manipulator arms, a water tank and a set of Helmholtz coils. It was the task of the previous project team to write control software for the manipulator arms. While they were successful in completing basic control and collision detection, much remained to be finished. Therefore, in order to move the project closer to its objective, this report details improvements to both the Testing Rig and its control software.

The improvements proposed for the Testing Rig were moving it to its final location, wiring the limit switches on the rotational axes of the manipulator arms, reducing vibrations by fixing a free-swinging counterweight and cleaning up the wiring. The improvements proposed for the control software were the addition of scan routines to perform Magnetic Field Surveys.

Furthermore, additional improvements were requested during the project. These include: constructing cable brackets, wiring all four controllers, wiring the remaining limit switches for X, Y and Z axes, running cables through cable trays and rewriting the initialization routine in feMove.

All of the above improvements were successfully completed except for the fixing of the free swinging counter-weight. The methods, equipment used and results for these improvements can be found in the body of the following report. It is the overall recommendation of this report, that work continue as planned.

Table of Contents

Executive Summary 2
Table of Contents 3
1.0 Introduction..... 4
1.1 Background and Motivation..... 4
1.2 Project Scope..... 6
1.3 Sponsor Information..... 7
1.4 Project Objectives..... 7
1.4.1 Testing Rig Improvements..... 7
1.4.2 3D Magnetic Field Survey..... 7
1.5 Organization..... 8
2.0 Discussion..... 8
2.1 Theory..... 8
2.1.1 System Architecture..... 8
2.2 Experimental Equipment..... 11
2.2.1 Motors and Actuators..... 11
2.2.2 Mechanical Improvements..... 12
2.2.3 Wiring Improvements 13
2.3 Testing Protocols and Methods 16
2.3.1 Movement Repeatability Testing..... 16
2.3.2 Hall Probe Verification 16
2.3.3 Initialization and Scan Routines 17
2.3.4 Performing Scans 21
2.4 Results 27
2.4.1 Results from Repeatability Testing 27
2.4.2 Results of Magnetic Surveys 29
2.4.3 Data Analysis 30
2.5 Discussion of Results 42
3.0 Conclusions 44
4.0 Project Deliverables 45
4.1 List of Deliverables 45
4.2 Financial Summary 46
4.3 Ongoing Commitments by Team Members 46
5.0 Recommendations 46
6.0 References 47
7.0 Appendix A: feMotor Settings 48
8.0 Appendix B: Code for initialization Routine 50
9.0 Appendix C: Code for gen_scan_path Routine 52

1.0 – Introduction

1.1 – Background and Motivation

This project is a continuation of a previous ENPH 479 project. The goal of the original project was to test the optical properties of a new acrylic shielding being used on the photomultiplier tubes (PMT) inside the Super Kamiokande (Super-K) neutrino observatory.

The Super-K, located in Japan, was constructed to observe proton decay and to study astronomical events. It consists of a large water tank, lined with 11,146 PMTS and filled with 50,000 tons of ultra pure water. The Super-K is able to find neutrinos by detecting Cherenkov radiation. When neutrinos pass through the detector, they weakly interact with the nuclei or electrons of the water and produce charged particles. These charged particles can move faster than the speed of light in water. As a result, a cone of light is produced inside the detector, similar to how a sonic boom is created when the speed of sound is exceeded in air. This cone, known as Cherenkov radiation, is projected onto the wall of the detector and recorded as a ring. Using the information from the PMTs, it is then possible to reconstruct the interaction events of the neutrinos and determine their properties.

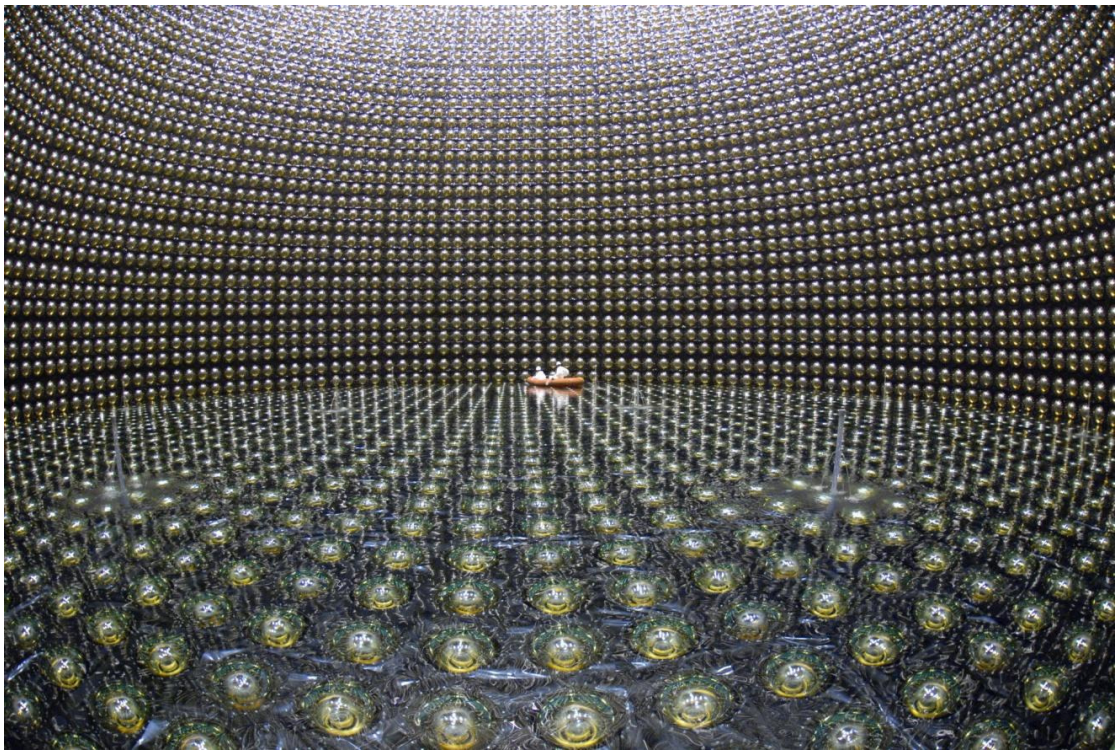
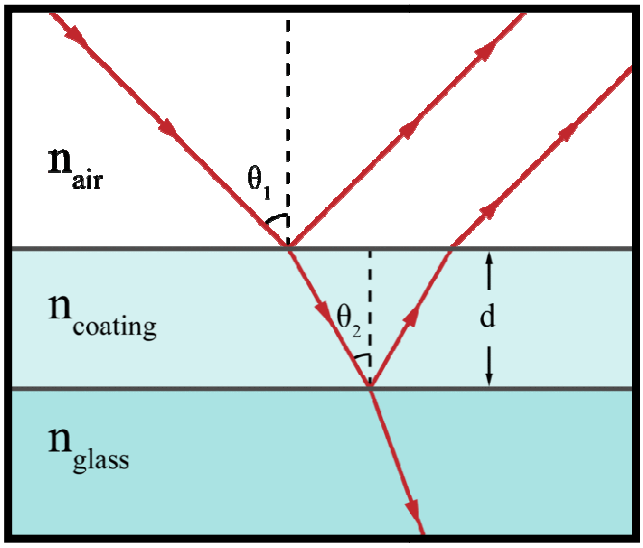


Figure 1: *The interior of the Super-K neutrino observatory.*
http://www.sinet.ad.jp/case/kamioka/PH20-water-withboat-apr23.jpg

In November 2001, a cracked PMT led to a cascading implosion of about 6600 of the PMTs, causing nearly $20 million worth of damage. In order to prevent any future PMT failure from causing such cascades, an acrylic shield was added to each PMT to protect it from shockwaves in the water. (Smit-Anseeuw, Zimmer 2012)

This shielding, however, has changed the optical properties of the PMTs by introducing thin-film effects. To construct an anti-reflective coating, the shielding thickness has been controlled precisely so that normally incident light waves accumulate exactly a quarter-wavelength of phase when they pass through. For this type of coating, the light will be reflected at both the air-coating and coating-PMT interfaces. However, since the light reflected from the second interface will have passed through the coating twice by the time it rejoins the light reflected from the first interface, it will have a half-wavelength phase shift. In other words, the two waves will destructively interfere, rendering the coating transparent. If, however, the angle of incidence is large, the light reflected from the second interface will accumulate more than a half-wavelength in phase. At its worst, this will cause the reflected waves to constructively interfere, greatly reducing the performance of the PMT.



$$n_{air} < n_{coating} < n_{glass}$$

$$d = \lambda/(4n_{coating})$$

For constructive interference:

$$2n_{coating} d \cos(\theta_2) = m\lambda$$

For destructive interference

$$2n_{coating} d \cos(\theta_2) = \left(m - \frac{1}{2}\right)\lambda$$

Figure 2: *Thin-film effects and anti-reflective coatings* http://en.wikipedia.org/wiki/Thin-film_interference

As this angle dependence of the reflectivity will affect the results obtained from the PMTs, it is necessary to characterize this behaviour so that it can be incorporated into the analysis. Similarly, the effects of the wavelength (330-550 nm) and polarization of the light will be studied. To do this, Dr. Hiro Tanaka, our sponsor at TRIUMF, has constructed a PMT Testing Rig. This Testing Rig will place the PMT in an environment similar to that of the Super-K, a water tank. It will then position a light source and detector at different angles to the surface, using

two gantry style manipulator arms, and measure the reflectivity. Furthermore, the Testing Rig will have a set of Helmholtz coils used to compensate for the Earth's magnetic fields and that of the nearby TRIUMF cyclotron. The control of the magnetic fields inside the Testing Rig is important because they can also affect the performance of the PMT by curving electron trajectories and reducing gain.

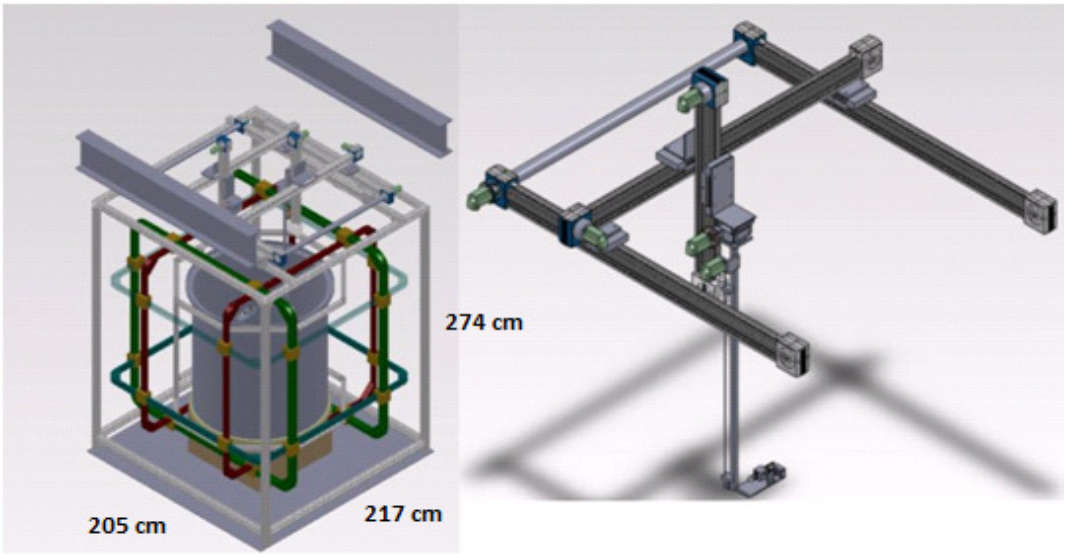


Figure 3: *PMT Testing Rig*

1.2 – Project Scope

As mentioned before, this project is a continuation of a previous ENPH 479. The project team for the original project was Aaron Zimmer and Nils Smit and their final report can be found in the references. The focus of their project was to implement the control of the manipulator arms. Each arm has three positional and two rotational degrees of freedom and can be positioned anywhere within the 3D rectangular prism defined by their limit switches.

The aforementioned team was successful in completing basic control of the arms. That is, simple movement and collision detection, in each degree of freedom, has been implemented for a single arm. Therefore, it was the task of this project to build on this by making improvements in both hardware and software. Specifically, modifications were to be made to the Testing Rig, such as wiring the limit switches for the rotary axes, fixing the free-swinging counter-weight, and cleaning up the general wiring. Furthermore, a magnetic field survey was to be performed to verify the operation of the Helmholtz coils. To do this, a Hall probe was to be placed on the end of one of the manipulator arms and scanned through the tank volume. As such, scan routines which focus on efficiency and accuracy were to be added to the current control software.

1.3 - Sponsor Information

Dr. Hirohisa Tanaka is a particle physicist at UBC with specialization in neutrino physics. He is currently working on the T2K (Tokai-to-Kamioka) project, an experiment searching for the transmutation of muon neutrinos to electron neutrinos. His group is also actively involved in the development of calibration and reconstruction algorithms for the super-k observatory. (Smit-Anseeuw, Zimmer 2012)

1.4 - Project Objectives

The final objective of this project is to characterize the optical properties of a new acrylic shielding used on the PMTs employed by the Super-K neutrino observatory. The following tasks are assuming that the basic control of the manipulator arms has been completed.

1.4.1 - Testing Rig Improvements

The following improvements to the Testing Rig were requested:

- Move Testing Rig to its final location
- Wire the limit switches for the rotary axes
- Fix the free-swinging counter-weight
- Clean up wiring on controller panel and Testing Rig

1.4.2 - 3D Magnetic Field Survey

The first task was to develop scan routines for the volume of the test chamber. These routines would minimize travel time and deflection (due to vibration) of the manipulator arm, to ensure maximum accuracy. Magnetic field measurements would be performed by a Hall probe placed at the end of one of the manipulator arms. Data logging for the Hall sensor would need to be developed. Once these tasks had been completed, an initial magnetic field Survey would be performed at the final location of the Test Rig without the Helmholtz coils in place. This would provide information required to set the coil currents. Once this was completed and the Helmholtz coils were in place, a final magnetic field survey of the test chamber would be completed.

1.5 - Organization

The following section, Discussion, examines the exact methods used in obtaining the project objectives. This is done by, first, giving a theoretical background for the methods. Then, the experimental equipment and methods, themselves, are described. Lastly, this is followed by a presentation and analysis of the results obtained from the methods. The section after Discussion, Conclusions, discusses the conclusions that have been reached from the results. Project Deliverables details the items that will be delivered upon completion of the project, as well as any costs or ongoing commitments for the team, and Recommendations contains the team's suggestions for future work. Finally, the Appendices contain additional data, while the References include source details for information used in this report.

2.0 - Discussion

2.1 - Theory

2.1.1 - System Architecture

The software architecture for the Testing Rig control software consists of a series of frontends which communicate with each other through an online database (ODB). This database is part of a c-based, in-house language used by TRIUMF, called MIDAS. In addition to the ODB, MIDAS contains functionality for data acquisition and the slow control of external equipment. It is these aspects on which the Testing Rig's control software is built.

The hierarchy of the slow control structures in MIDAS is as follows. At the highest level, each subsystem being controlled is labeled as an equipment. At the mid-level, frontends, implemented in c, control each equipment by reading values from the online directories of the ODB. These values are used in routines which, using a class driver, send commands to the external hardware at the lowest level. The status of this external hardware is monitored by the frontends while these commands are being executed. This data is written back into the ODB, in order to provide information to the user. Combined, these three levels enable control of the external hardware by the user. Currently, there are four frontends: feMotor, feMove, feScan and fedvm.

The first frontend, called feMotor, was written before the first project team began work. Therefore, it contains much of the basic functionality for the control software. By itself, it can be used to set the motor velocity and acceleration of each axis, to send the motors to a relative position on each axis and to monitor real-time variables such as the current position, the current state of the limit switches and whether or not the motors are currently moving. However, feMotor has many options that will rarely be changed by the user. Therefore, the second frontend, called feMove, was written by the first project team to hide this functionality.

feMove does this by only communicating to those values which are necessary for movement. These can be separated into three categories: settings, controls and variables. The values for the settings in the ODB are read into feMove once upon initialization. These settings include velocity and acceleration as well as the additional settings of motor scaling, axis channels and limit positions. The control values are used to send commands to the axis motors. Therefore, they are hot-linked to functions in feMove which send the desired commands to feMotor. Hot-linking mean that, if the user changes the control values in the ODB, the function is called. These controls include Start Move and Stop Move as well as the additional control, ReInitialize. Furthermore, an array of destination values for the axes is included in the control values. This, however, is not hot-linked to any particular function and is simply read into feMotor when Start Move is set to true. The variables are used to monitor aspects of the Testing Rig in real-time. Like feMotor, they include the current position, the current state of the limit switches and whether or not the motors are currently moving. However, in addition to these, feMove also monitors whether or not a move has been completed, whether or not the gantry destinations have been swapped and whether or not the axes have been initialized. The initialization of the axes is functionality not found in feMotor and is used to convert the relative position in counts that feMotor uses into absolute positions in meters. This is done by defining an origin using the position of the limit switches. Therefore, all of the above values for feMove are in absolute meters, instead of relative counts. This was done originally by the first project team but was then completely revised in order to be able to disable the initialization of certain axes and to simplify the code. For details of the implementation of this routine, please refer to the following section. Please note that feMove also contains an advanced collision detection routine that was not used during the course of this project. For more information on the feMove frontend, please consult the previous project group's final report, found in the references at the end of this report. The third frontend feScan is an additional layer on feMove.

The purpose of feScan is to string together the individual movements of feMove into a path. It does this by reading in additional scan settings from the ODB. feScan uses these settings to generate a path which it then sends to feMove, one move at a time. If desired, feScan can also take Magnetic Field data by communicating with the fourth frontend, fedvm. fedvm is solely

used to send commands to a digital voltmeter connected to a 3-axis Hall Probe. It is being used "as is" from another project which required similar data. feScan uses the "run" feature in MIDAS to produce Magnetic Field and position data in "banks". The format of these banks can be read by an auxiliary tool “analyzer_example.exe”, which produces the data tables found in the Results section below.

Below is a diagram showing the architecture described above. The Settings or Controls of a layer, set through the User Interface, are overwritten by those of the layer above, if connected by a downward arrow. Only those ODB values which are common to the two levels are overwritten. Note that setting data and control tends to flows down, while variable data tends to flow up. The only two exceptions to this are fedvm which is not directly set or controlled by the other frontends and feScan which uses the settings of feMove while not changing them.

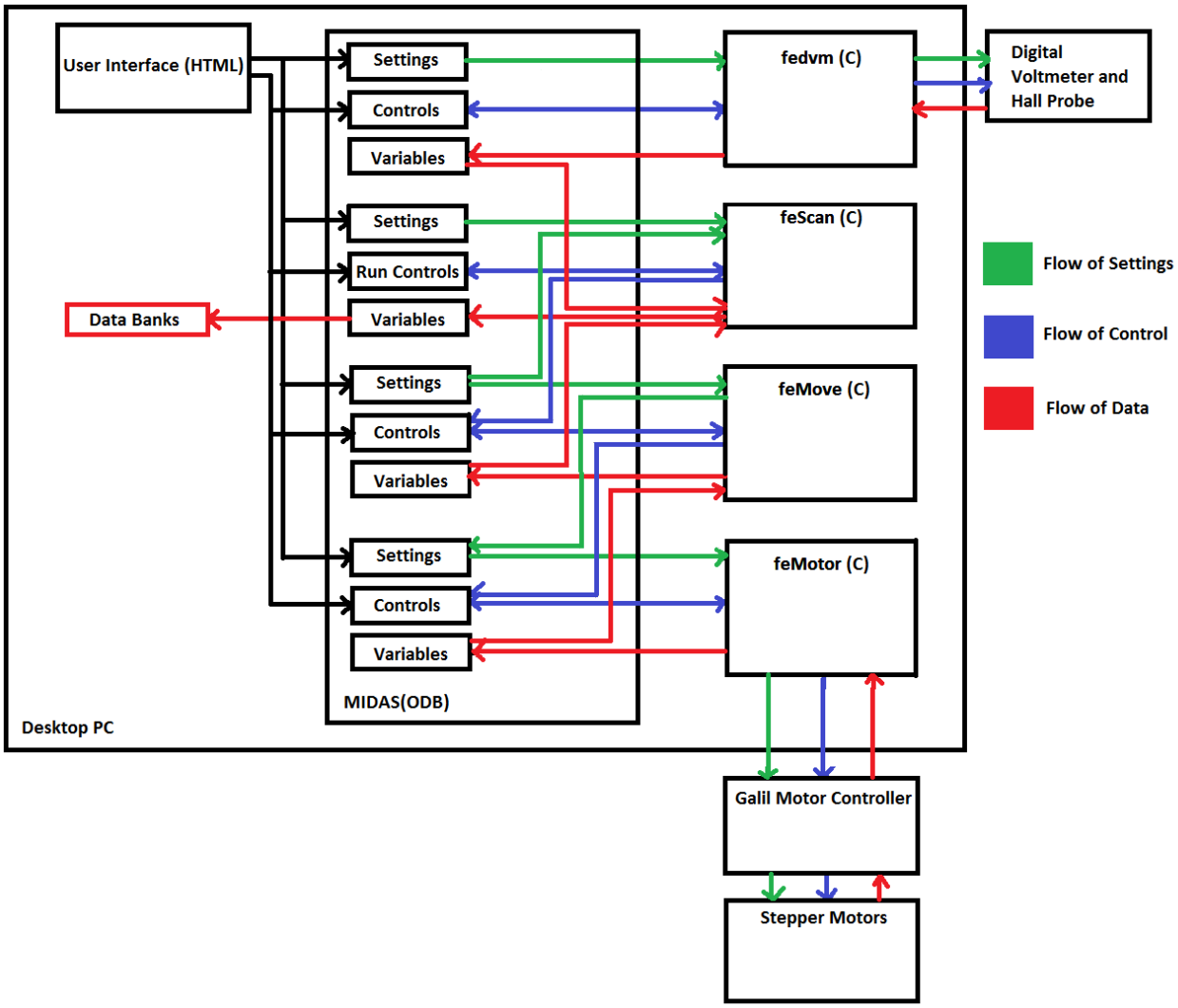


Figure 4: *System Architecture*

2.2 – Experimental Equipment

2.2.1 – Motors and Actuators

The X, Y and Z axes of the Testing Rig are constructed from three main components: a linear stage from Bosch-Rexroth, a single 12 volt stepper motor and limit switches to limit the range of motion (Figure 5).

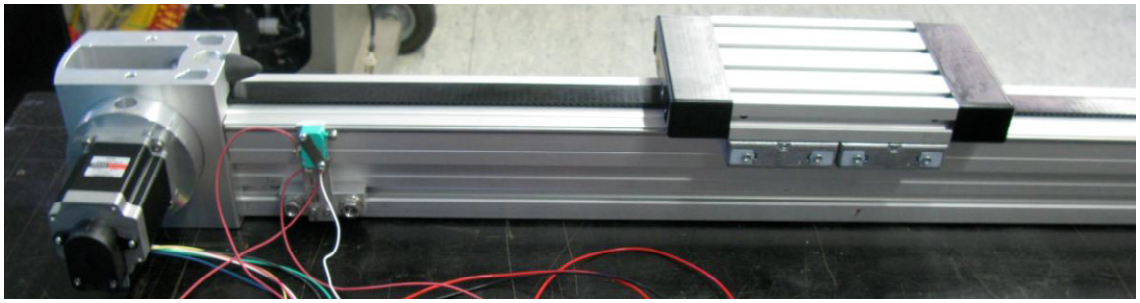


Figure 5: *One of the Bosch-Rexroth linear stages used in the Testing Rig. The limit switch and stepper motor for this axis can also be seen in the picture.*

Rotation about the vertical axis is actuated using a high precision rotary table from Parker. Tilt of the end-affecter is accomplished using a worm screw on a stepper motor. This stepper motor is identical to that above (Figure 6).

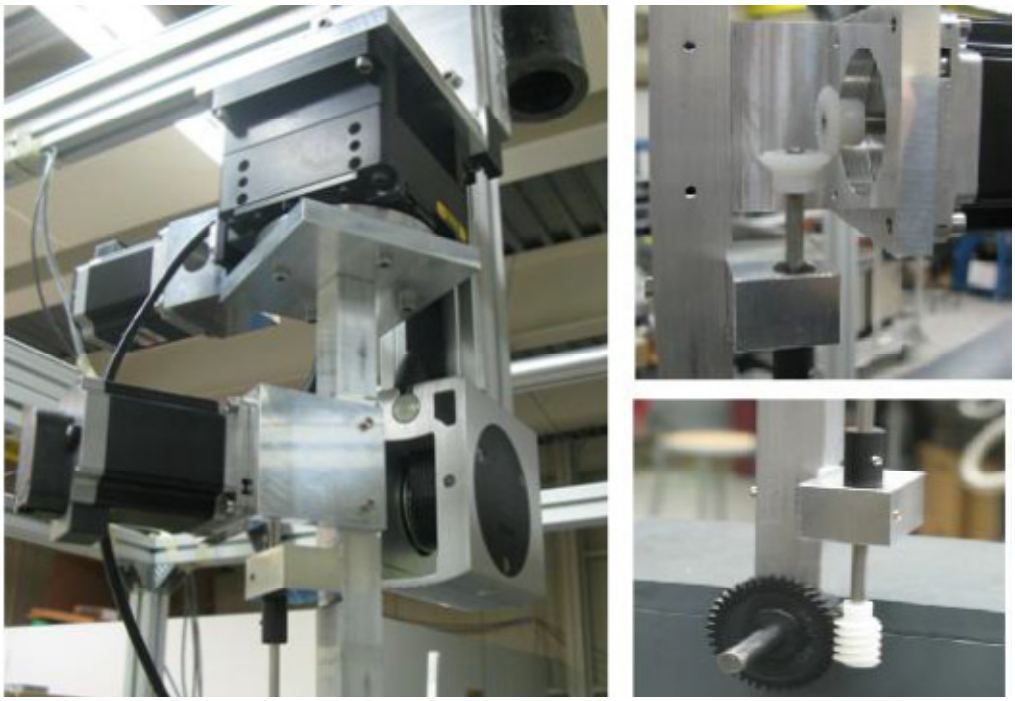


Figure 6: *Parker rotary table (left) and top and bottom of the worm screw (right).*

2.2.2 – Mechanical Improvements

The following hardware improvements were proposed for the Testing Rig:

- Move Testing Rig to its final location
- Fix the free-swinging counter-weight
- Mount Hall Probe
- Install Helmholtz coils

Additionally, the following hardware improvements were requested during the project:

- Construct nine cable brackets

The movement of the Testing Rig to room B002 at TRIUMF was performed with the help of Wayne Faszer. At the same time, most of the mechanical assembly of the second manipulator arm was completed. The fixing of the free-swinging counter-weight was delayed and eventually abandoned in lieu of more urgent tasks. The Hall probe was attached to the manipulator arm using a temporary mounting and the installation of the Helmholtz coils was left incomplete due to time constraints. Lastly, in order to securely fasten cables to the Testing Rig, nine cable brackets were constructed in the TRIUMF machine shop (Figure 7).

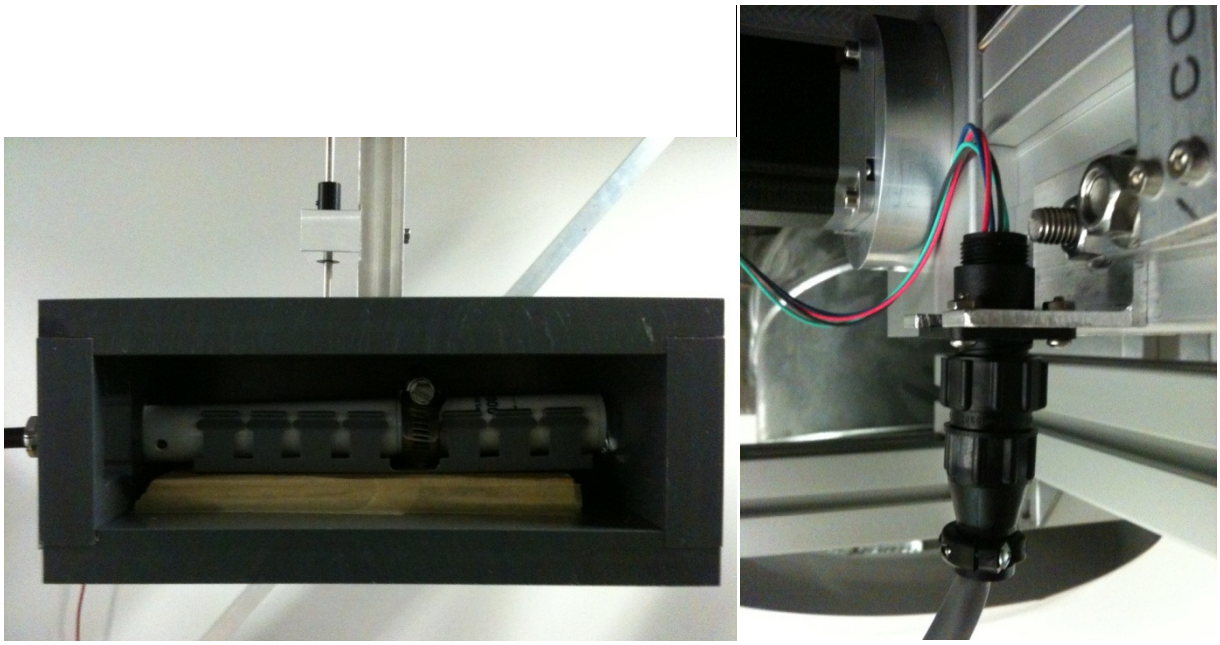


Figure 7: *Hall probe mount (left) and cable bracket (right)*

2.2.3 – Wiring Improvements

To wire one of the motors to the Galil 4183 motor controller, four wires are required, two for each phase. Similarly, to wire a limit switch to the controller, two wires are required, one for the +5 V and one for the floating signal voltage.

To accomplish this, the previous project team used Belden 9368 cables for the motors. These cables contain two shielded wire pairs. The connectors used were standard Molex connectors, six pin on the motor side and four pin on the controller side. For the limit switches, Belden 9504 cables were used. These cables contain four shielded wire pairs. This allowed for the connection of four limit switches with a single cable.

By the end of their project, the previous project team had completed the following:

- Motor connections for the X, Y, and Z axes of a single arm
- Negative limit switches for the X, Y and Z axes of the arm
- Positive limit switch for the X axis of the arm.

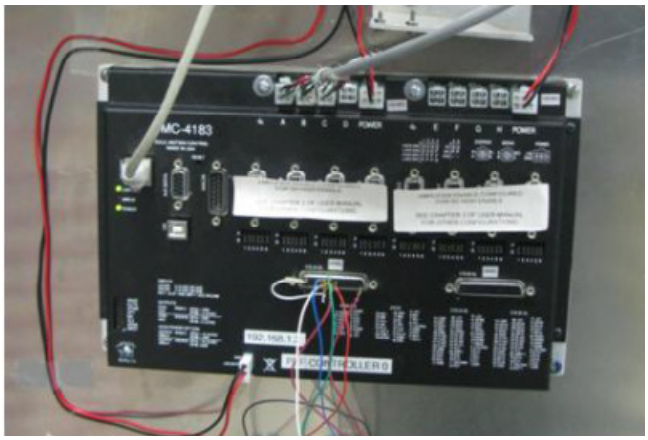
However, this wiring was deemed too messy by the project sponsors and had to be re-done. As such, the following wiring improvements were proposed:

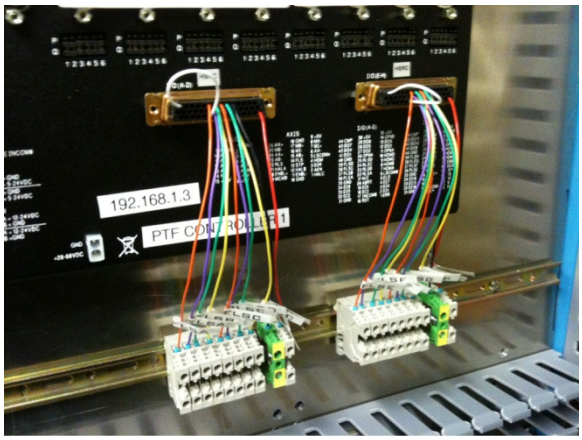
- Replacement of some of the cables (same cable types)
- Combining wires into plugs when interfacing with the controller
- Using easily modified rail terminals
- Labelling all wires using label maker
- Improve cable festooning on gantry rails
- Wire the limit switches for the rotary axis

Additionally, the following wiring improvements were requested during the project:

- Completely wire terminal to controller interface for both arms
- Wire remaining limit switches for X, Y and Z axes
- Run cables through cable trays

All of the above wiring improvements were completed with the help of Thomas Lindner. Figure 8 shows two before and after comparisons to see the effects of these improvements. Figure 9 shows cables running through the cable trays constructed for the Testing Rig and Figure 10 shows the completed manipulator arm inside the Testing Rig.





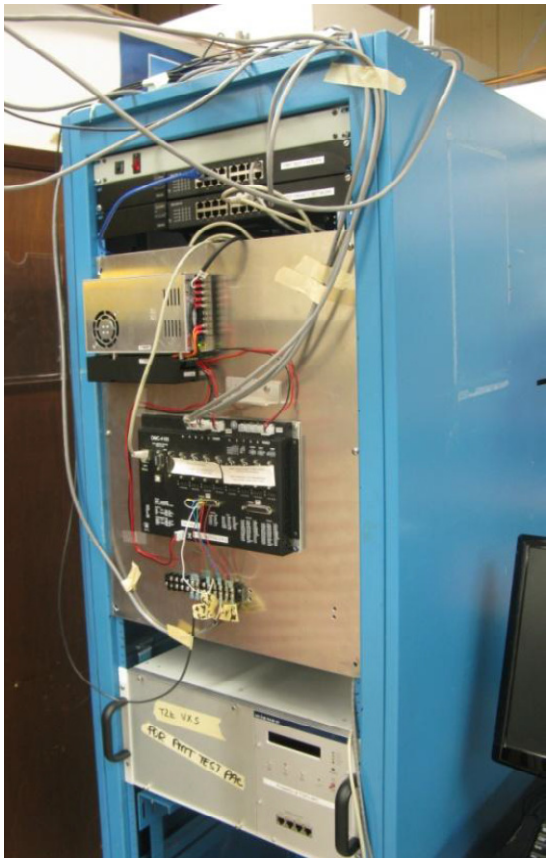


Figure 8: *Before and after wiring improvements*

Figure 9: *Cable trays from Testing Rig to Rack*

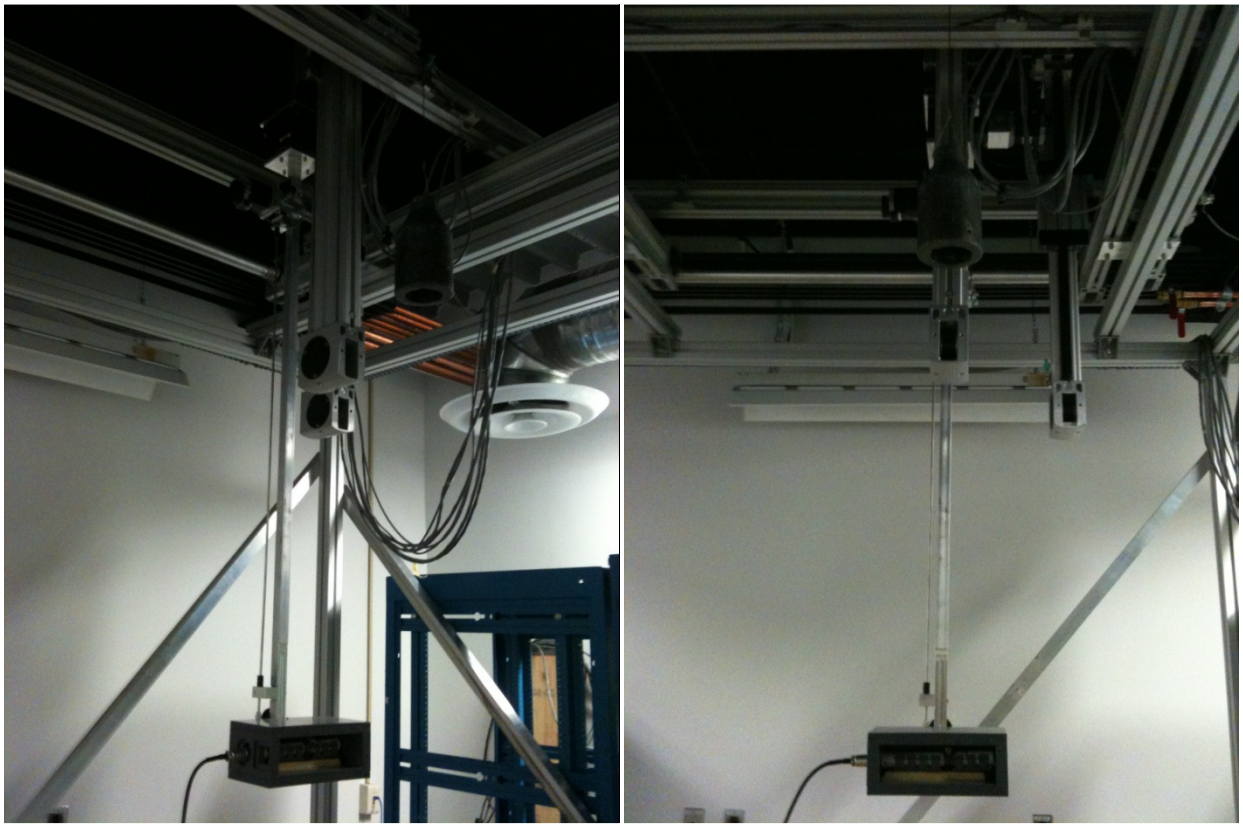


Figure 10: *Completed manipulator arm inside Testing Rig*

2.3 – Testing Protocols and Methods

2.3.1 – Movement Repeatability Testing

Movement repeatability testing had been performed by the previous project group. However, this was done using the setup visible in Figure 5. Therefore, while this testing proved that the accuracy of the stepper motors was good enough to achieve 1 mm tolerances under minimal load, it said nothing about the accuracy of the fully assembled system. As such, movement repeatability testing was continued on the completed manipulator arm. This was done by testing the effects of the settings in feMove. That is, the effects of destination, velocity and acceleration were investigated.

The effects of destination were tested by repeatedly moving each axis to short, middle and long distances. In each test, the axis would be moved into the negative limit switch in order to provide a fixed reference point. Then, the axis would be moved to each position five times. At which point, the position of the carriage on each axis was marked and the variation from an initial benchmark would be measured. In these tests, it was assumed that the relative position of the end-affecter to the carriage remained fixed. This is an appropriate assumption, as long as the vibrations of the manipulator arm are allowed to damp out. For each movement, the variation was measured using a set of digital calipers. If this variation was measured to be less than 0.5 mm, or the width of the benchmark, than the variation was determined as being negligible.

The effects of velocity were tested by, again, moving the axis into the negative limit switch. Then, at progressively increasing speeds, the axis would be moved to the long distance above. This is because overshoot is more likely if the axis is able to reach top speed. Each velocity was tested three times and again, the variation from an initial benchmark was measured using a set of digital calipers.

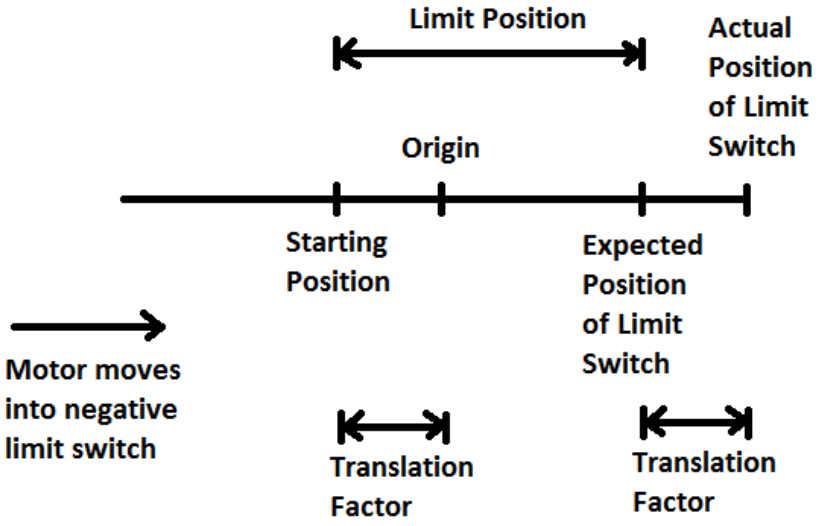
The effects of acceleration were tested in the exact same manner as velocity. The goal was to increase the acceleration to such a high value that the force, due to the momentum of the arm, would cause the motor to skip, causing error in the final position.

2.3.2 – Hall Probe Verification

The Hall Probe was verified in two ways. First, the repeatability of its measurements was tested by doing three consecutive scans of the X-Y plane centered at Z=0 m. This data was compared to see if, indeed, they were identical. Second, the values measured by the Hall Probe were tested by comparing the minimum value of the 3D Magnetic Field Surveys with the expected value of the Earth's magnetic field. This expected value was obtained using the calculator on the National Geophysical Data Center website. Coordinate and elevation data for TRIUMF were obtained from Google Earth.

2.3.3 – Initialization and Scan Routines

During this project, two routines were written. The first, "initialize", in the feMove frontend, was written in order to convert the relative positions in counts, used by feMotor, into absolute positions in meters. This was done by defining an origin to which positions could be referenced. This was accomplished using one of the settings added to feMove, the physical position of the limit switches with respect to the origin. Since the positions of the limit switches on each physical axis are actually fixed, defining these positions ends up defining the position of the origin. The stepper motors used in the Testing Rig are controlled by relative positions in counts. Therefore, physical positions in meters need to be converted via a scaling factor. Furthermore, since the positions are relative, they must be shifted by some translation factor in order for them to line up with our pre-defined origin. Another way of seeing this is if the starting positions of the motors were already on the pre-defined origin. Then, the relative positions used by the motors would be the absolute positions. To find this translation factor for each axis, the difference between the expected position and the actual position of the negative limit switch is found. These factors give the position of the origin with respect to the starting position and are converted so that they are in counts. This conversion is visualized on a number line below (Figure 11).



Translation Factor = Actual Position of L.S. - Expected Position of L.S. (converted to counts)

Figure 11: *Conversion from relative position to absolute position*

The current "initialize" routine is a revised version of the original "initialize" routine written by Nils Smit. The routine was revised in order to be able to disable the initialization of certain axes. Below is the old "initialization" routine converted into pseudo-code side by side with the pseudo-code used in the implementation of the revised routine (Figure 12).

Set all limit flags to true so that disconnected axes will already be at their limits when movement starts (other limits will revert to false)

Set destination for each axis to negative limit switch

Write the destination values to the ODB

Send both arms to their limits on the Z-axis

Wait until one of the two arms hits their limit switch, then stop both

For the arm with the Z limit triggered, move arm to X limit and wait until triggered

For the arm with the Z limit not triggered, move arm to Z limit and then X limit

Send both arms to Y, Phi and Theta limits

Wait for all limit switches to be triggered

Wait for all the motors stop moving

Read in the current position of the motors

For each motor

If the channel is enabled

Set the origin to the current position minus the limit position in counts

Set the current position in meters to the limit position

Write the position values to the ODB

Set initialized variable to true and exit

For each motor

If the position of the negative limit switch is 9999

Set the destination to the current position (0 counts)

Record that the limit is disabled

Else

Set the destination to a value past the position of the negative limit switch

Record that the limit is enabled

Write the destination values to the ODB

For each pair of motors(one on each arm)

Send motors to their destinations

Wait until both, one or none (based on if enabled) of the motors reach their limit switches

If the current position is not changing

Stop motors and exit since limit switch broken

Wait for all the motors stop moving

Read in the current position of the motors

For each motor

If the limit is enabled

Set the origin to the current position minus the limit position in counts

Set the current position in meters to the limit position

Else

Set the origin to the current position (0 counts)

Set the current position in meters to the starting position in meters (0 meters with respect to origin)

Write the position values to the ODB

Set initialized variable to true and exit

Figure 12: *Side by Side comparison of pseudo-code for original and revise Initialization routine*

As can be seen above, the following changes were made:

- Instead of setting all the negative Axis Limits (array AxisLimit) to true in the routine and then reading them back from the ODB to determine which axes are enabled, I have made it so that a Limit Position (array LimPos) setting of 9999 will indicate when a negative axis limit is disabled (array tempNegLimitEnabled). This is because there is no difference between a limit switch being active low or disconnected. Meaning that when the routine reads back the Axis Limits from the ODB, they will all be false, regardless (if not already at a limit). If a negative Axis Limit is disabled, that axis will not be initialized (can still be moved, though). Specifically, during initialization, the arm will not be moved along that axis and the Origin (array mOrigin) will be set to its current position.

- Instead of moving the arms one at a time, I have made it so that both arms move simultaneously. Specifically, they both move in the following order: neg x-axis limit, neg y-axis limit, neg z-axis limit, neg phi limit and neg theta limit. This was done because there seemed to be no reason to move the arms individually. This is because the negative limit switches for each axis are on opposite sides of the testing rig. Furthermore, the arms are physically prohibited from moving past each other. This change significantly simplified the code, reducing it from 153 lines to 122 lines.

- The previous code assumed that the limit switches were operational. As a result, this would cause the Testing Rig to repeatedly ram into the limit switches, if they were broken. To avoid this, a simple check was added to the routine. If the position of the enabled motors does not seem to be changing, all motors are stopped and an error is returned.

The above routine is called in two different ways. Either a move has been started with the Initialization flag set to false or the Reinitialization flag is set to true. In the future, an independent array in the ODB should be created for limit enables.

The second, "gen_scan_path", in the feScan frontend, was written in order to generate a number of different scan paths based on user settings read from the ODB. So far, the "gen_scan_path" routine only contains two scan types: cylindrical and rectangular prism. These two scan types were chosen for a variety of reasons. First, the cylindrical scan path was chosen to accommodate the geometry of the water tank. This will be useful for the magnetic survey with the Helmholtz coils turned on since the water tank will most likely be installed before the Helmholtz coils. Second, the rectangular prism scan path was chosen because it provided the best coverage of the interior volume of the Testing Rig. This was particularly useful for the magnetic survey without Helmholtz coils. Furthermore, the rectangular prism scan path allowed for the additional functionality of linear and plane scans. These scans are essential for quick data collection since otherwise, a full volume scan would need to be performed. Below is pseudo-code used to implement the routine (Figure 13). And below that is a list and description of the user settings referenced in the code.

```
If Cylindrical scan selected
        Check cylindrical scan parameters
        For layers in volume (height/layer_thickness)
                Start at center point
                For loops in layer (radius/loop_seperation)
                        For points in loop (2*PI*loop number*loop_seperation/arc_step)
                                Set X,Y,Z position of point to that of previous point
                                Calculate the new X,Y positions using polar coordinates
                                Record point number
        Add end point

If Rectangular Prism scan selected
        Check Prism scan parameters
        For layer in Z layers (prism_height_z/z_step)
                If Y axis not fixed
                        Reverse direction along Y axis
                For layer in Y layers (prism_width_y/y_step)
                        If X axis not fixed
                                Reverse direction along X axis
                        For layer in X layers (prism_length_x/x_step)
                                Calculate X,Y,Z position of point(initial position + layer number*step)
                                Record point number
        Add end point
Check path size
```

Figure 13: *Pseudo-code for Scan routine*

User Settings:

- scan_type: Selects the scan type (0 = cylindrical, 1 = rectangular prism/plane/linear)
- height: Height of cylinder centered at Z=0 (default value = 0.50 m)
- radius: Radius of cylinder centered at X,Y = 0 (default value = 0.32 m)
- arc_step: Arc length between points on loop (default value = 0.05 m)
- loop_separation: Seperation between loops (default value = 0.04 m)
- layer_thickness: Thickness of each horizontal layer (default value = 0.05 m)
- prism_height_z: Height of prism w.r.t initial z position (default value = 0.50 m)
- prism_length_x: Length of prism w.r.t. initial x position (default value = 0.72 m)
- prism_width_y: Width of prism w.r.t initial y position (default values = 0.64 m)
- z_step: Distance between points along Z axis (default value = 0.05 m)
- x_step: Distance between points along X axis (default value = 0.08 m)
- y_step: Distance between points along Y axis (default value = 0.08 m)
- init_pos_z: Initial position on Z axis (default value = -1.0*(prism_height_z/2.0) m)
- init_pos_x: Initial position on X axis (default value = -1.0*(prism_length_x/2.0) m)
- init_pos_y: Inital position on Y axis (default value = prism_width_y/2.0 m)

To perform plane and linear scans, it is necessary to fix either one or two of the three axes. To fix an axis, set the dimension of the rectangular prism along that axis to zero. This can be seen as squeezing the prism along that axis until it turns into a plane. Furthermore, it is necessary to define the position of the plane by selecting its intercept along the fixed axis. Below are a few examples of plane and linear scans (Figure 14).

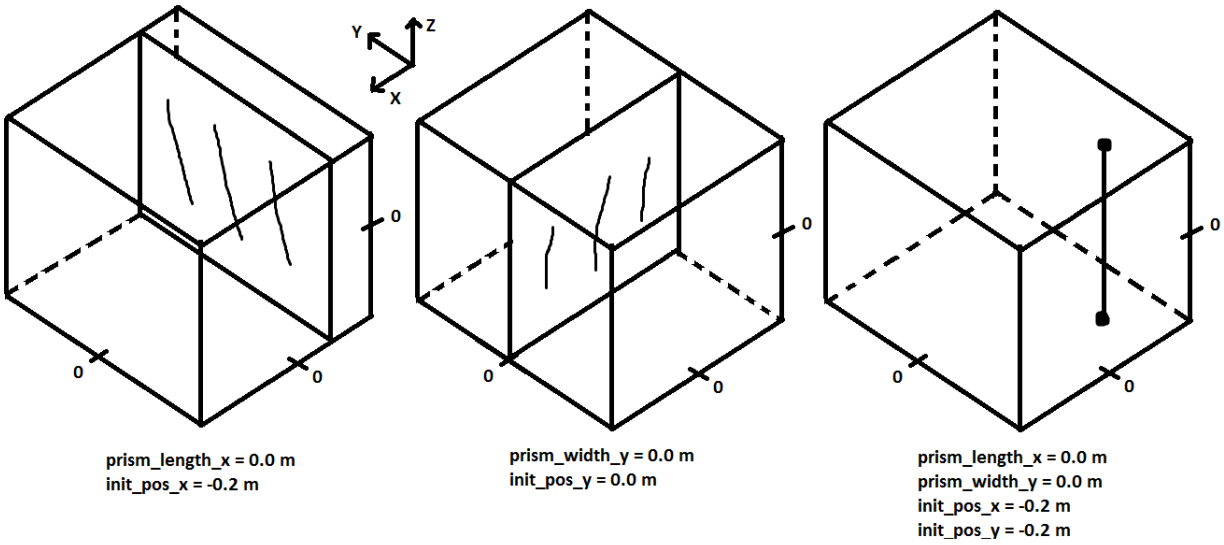


Figure 14: *Plane and linear scans inside full rectangular prism*

The above routine is only called once. This is by the "begin_of_run" routine in feScan. In the future, the scan settings should be read in from the ODB.

2.3.4 – Performing Scans

To perform a Magnetic Field Survey, follow the procedure below:

Pre-Scan Checklist:

- Are the X, Y and Z axis motors connected to the Galil Controller?
- Are the limit switches for the X, Y and Z axis motors connected to the Galil controller?
- Is the Galil Controller connected to the network via the PC as shown below (Figure 15)? Is everything turned on?
- Are the Digital Voltmeter and Hall probe connected as shown below (Figure 15)? Are they turned on?
- Is the Hall probe tightly fastened to the manipulator arm?
- Are the arms able to move freely between their limit switches
- Are all wireless radios turned off (cell phones and laptops)

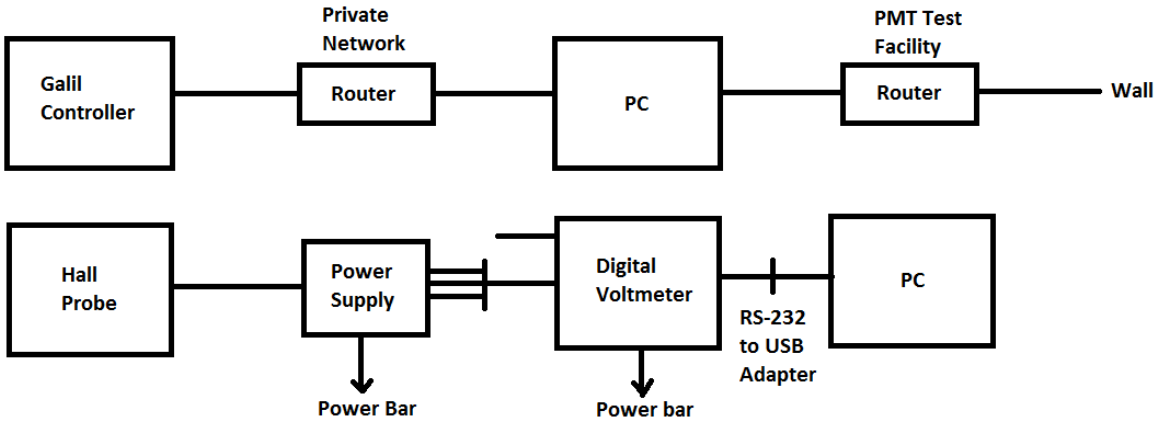


Figure 15: *Equipment connections*

Main Procedure:

1.) Log onto PC

2.) Open up four separate Linux command lines in GNOME Terminal 2.31.3 (Figure 16)

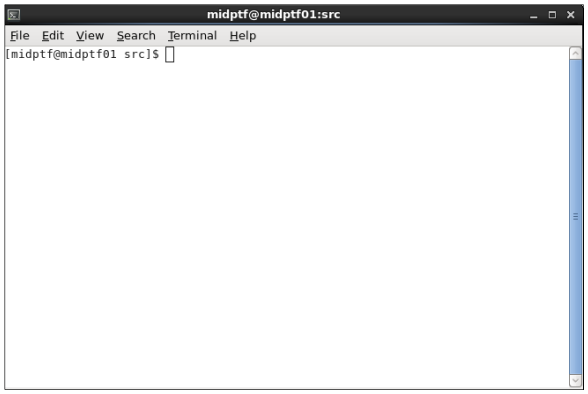
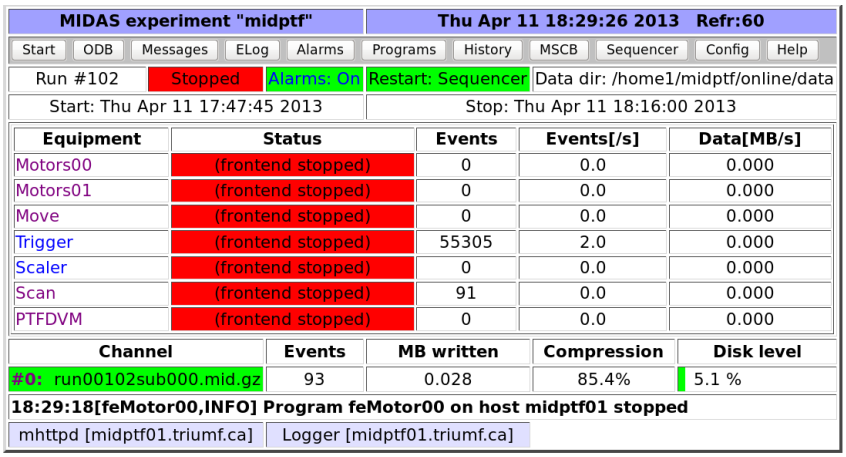


Figure 16: *GNOME Terminal 2.31.3*

3.) For each terminal, cd into the src (source) directory by typing "cd online/src"

4.) Open up Firefox and go to the midptf MIDAS experiment by typing "localhost:8081" in the address bar (Figure 17)



MIDAS experiment "midptf" | Thu Apr 11 18:29:26 2013 Refr:60

Start | ODB | Messages | ELog | Alarms | Programs | History | MSCB | Sequencer | Config | Help

Run #102 | Stopped | Alarms: On | Restart: Sequencer | Data dir: /home1/midptf/online/data

Start: Thu Apr 11 17:47:45 2013 | Stop: Thu Apr 11 18:16:00 2013

Equipment	Status	Events	Events[/s]	Data[MB/s]
Motors00	(frontend stopped)	0	0.0	0.000
Motors01	(frontend stopped)	0	0.0	0.000
Move	(frontend stopped)	0	0.0	0.000
Trigger	(frontend stopped)	55305	2.0	0.000
Scaler	(frontend stopped)	0	0.0	0.000
Scan	(frontend stopped)	91	0.0	0.000
PTFDVM	(frontend stopped)	0	0.0	0.000

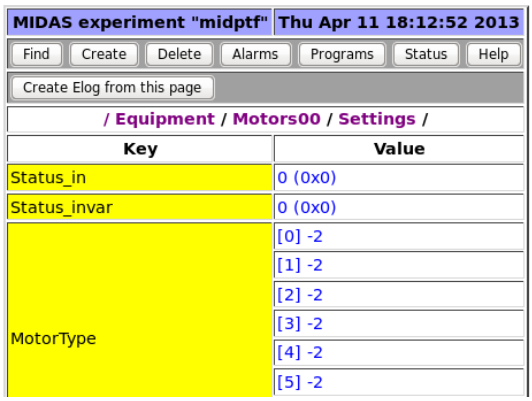
Channel	Events	MB written	Compression	Disk level
#0: run00102sub000.mid.gz	93	0.028	85.4%	5.1 %

18:29:18[feMotor00,INFO] Program feMotor00 on host midptf01 stopped

mhttpd [midptf01.triumf.ca] | Logger [midptf01.triumf.ca]

Figure 17: *midptf experiment page in MIDAS*

5.) On the midptf experiment page, go to ODB->Equipment->Motors00->Settings (Figure 18) and set as follows (or as desired):



MIDAS experiment "midptf" | Thu Apr 11 18:12:52 2013

Find | Create | Delete | Alarms | Programs | Status | Help

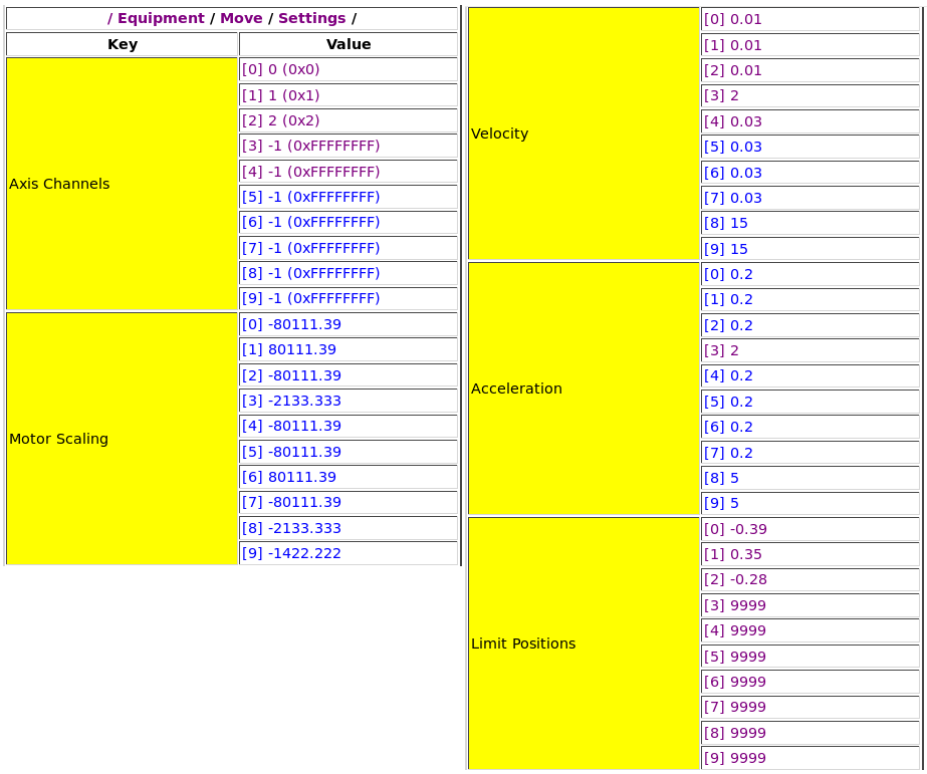
Create Elog from this page

/ Equipment / Motors00 / Settings /

Key	Value
Status_in	0 (0x0)
Status_invar	0 (0x0)
MotorType	[0] -2
	[1] -2
	[2] -2
	[3] -2
	[4] -2
	[5] -2

Figure 18: *feMotor settings (continued in appendices)*

6.) On the midptf experiment page, go to ODB->Equipment->Move->Settings (Figure 19) and set as follows (or as desired):

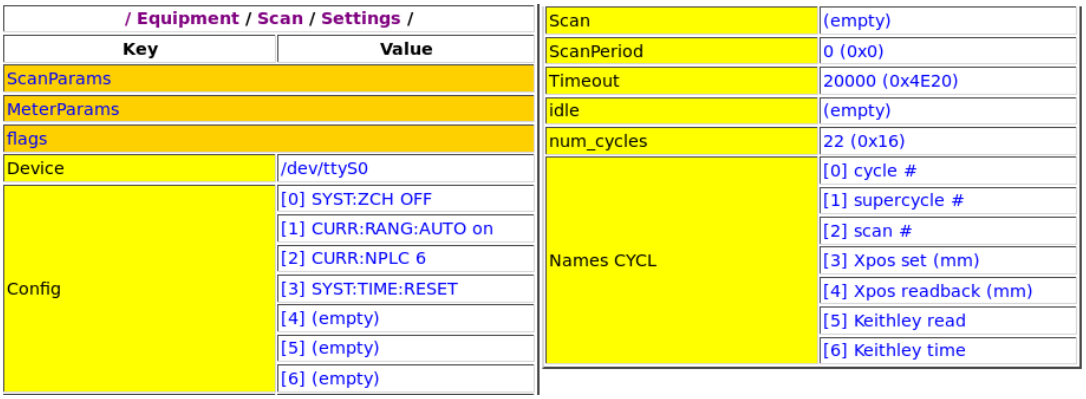


/ Equipment / Move / Settings /

Key	Value
Axis Channels	[0] 0 (0x0)
	[1] 1 (0x1)
	[2] 2 (0x2)
	[3] -1 (0xFFFFFFFF)
	[4] -1 (0xFFFFFFFF)
	[5] -1 (0xFFFFFFFF)
	[6] -1 (0xFFFFFFFF)
	[7] -1 (0xFFFFFFFF)
	[8] -1 (0xFFFFFFFF)
	[9] -1 (0xFFFFFFFF)
Motor Scaling	[0] -80111.39
	[1] 80111.39
	[2] -80111.39
	[3] -2133.333
	[4] -80111.39
	[5] -80111.39
	[6] 80111.39
	[7] -80111.39
	[8] -2133.333
	[9] -1422.222
Velocity	[0] 0.01
	[1] 0.01
	[2] 0.01
	[3] 2
	[4] 0.03
	[5] 0.03
	[6] 0.03
	[7] 0.03
	[8] 15
	[9] 15
Acceleration	[0] 0.2
	[1] 0.2
	[2] 0.2
	[3] 2
	[4] 0.2
	[5] 0.2
	[6] 0.2
	[7] 0.2
	[8] 5
	[9] 5
Limit Positions	[0] -0.39
	[1] 0.35
	[2] -0.28
	[3] 9999
	[4] 9999
	[5] 9999
	[6] 9999
	[7] 9999
	[8] 9999
	[9] 9999

Figure 19: *feMove settings*

7.) On the midptf experiment page, go to ODB->Equipment->Scan->Settings (Figure 20) and set as follows (or as desired)



/ Equipment / Scan / Settings /

Key	Value
ScanParams	
MeterParams	
flags	
Device	/dev/ttyS0
Config	[0] SYST:ZCH OFF
	[1] CURR:RANG:AUTO on
	[2] CURR:NPLC 6
	[3] SYST:TIME:RESET
	[4] (empty)
	[5] (empty)
	[6] (empty)
Scan	(empty)
ScanPeriod	0 (0x0)
Timeout	20000 (0x4E20)
idle	(empty)
num_cycles	22 (0x16)
Names CYCL	[0] cycle #
	[1] supercycle #
	[2] scan #
	[3] Xpos set (mm)
	[4] Xpos readback (mm)
	[5] Keithley read
	[6] Keithley time

Figure 20: *feScan settings*

8.) Since some scan settings have not yet been implemented in the ODB, open up feScan.c (found in the source directory) using GNU Emacs 23.1.1

9.) Starting on line 538, change the scan settings to the desired values

10.) Save and close Emacs and recompile the code by typing "make clean" and "make" in one of the terminals

11.) On the midptf experiment page, go to ODB->Equipment->PTFDVM->Settings (Figure 21) and set as follows (or as desired):

/ Equipment / PTFDVM / Settings /

Key	Value
Device	localhost:3010
Slot 100 type	HEWLETT-PACKARD,34901A,0,2.3
Names slot 100	[0] ch101
	[1] ch102
	[2] ch103
	[3] ch104
	[4] ch105
	[5] ch106
	[6] ch107
	[7] ch108
	[8] ch109
	[9] ch110
	[10] ch111
	[11] ch112
	[12] ch113
	[13] ch114
	[14] ch115
	[15] ch116
	[16] ch117
	[17] ch118
	[18] ch119
	[19] ch120
	[20] ch121
	[21] ch122
Slot 200 type	HEWLETT-PACKARD,0,0,0
Slot 300 type	HEWLETT-PACKARD,0,0,0
Timeout	60000 (0xEA60)
Idle	(empty)
Scan	[0] rout:scan (@101,102,103)
	[1] (empty)
	[2] (empty)
ScanPeriod	[0] 50 (0x32)
	[1] 50 (0x32)
	[2] 50 (0x32)
Config	[0] func "volt:dc",(@101,102,103);volt:dc:range:auto on,(@101,102,103)
	[1] SENS:VOLT:DC:NPLC 10,(@101,102,103)
	[2] (empty)
	[3] (empty)
	[4] (empty)
	[5] (empty)
	[6] (empty)
	[7] (empty)
	[8] (empty)
	[9] (empty)

Figure 21: *fedvm settings*

12.) Start the feMotor frontend by entering the following command in one of the terminals: "./feMotor -i 0"

13.) Optional: Test the limit switches: Manually switch each limit while observing the values of "Limit Pos" and "Limit Neg" on the Motors00 page just off the midptf experiment page (Figure 22)

MIDAS experiment "midptf" — Thu Apr 11 18:28:46 2013 Refr:60

ODB Status Help

Equipment: **Motors00** Motors01 Scan PTFDVM

Groups: **All**

Names	Destination	Position	Encoder	Limit Pos	Limit Neg	Limit Home	Moving	Enabled	LastStop	Analog1	Analog2	DigitalIn1	DigitalIn2	path
X axis	28841	-0	0	n	n	y	n	y	1	2.5824	2.5827	y	y	0
Y axis	25636	-0	0	n	n	y	n	y	1	2.5827	2.5827	y	y	0
Z Axis	0	-0	0	n	n	y	n	y	1	2.583	2.5824	y	y	0
SpareD	2133333	-0	0	n	n	y	n	y	1	2.5827	2.5827	y	y	
X axis	1000	-0	0	n	n	y	n	y	1	2.5827	2.5824	y	y	
unused	50000	-0	0	n	n	y	n	y	1	2.583	2.5824	y	y	
SpareG	2000	-0	0	n	n	y	n	y	1	2.5824	2.5827	y	y	
SpareH	-5500	-0	0	n	n	y	n	y	1	2.5827	2.5827	y	y	

Figure 22: *Status of limit switches*

14.) Start the feMove frontend by entering the following command in the next terminal: "./feMove"

15.) Initialize the axis by going to ODB->Equipment->Move->Control on the midptf experiment page and setting "ReInitialize" to "y" (Figure 23). Note: the axes will move to whatever positions you have entered into the "Destination" array, after initialization is complete

/ Equipment / Move / Control /	
Key	**Value**
Destination	[0] 0
	[1] 0
	[2] 0
	[3] 0
	[4] 0
	[5] 0
	[6] 0
	[7] 0
	[8] 0
	[9] 0
Start Move	n
Stop Move	n
ReInitialize	n

Figure 22: *feMove controls*

16.) Start the fedvm frontend by entering the following command in the next terminal: "./fedvm"
17.) Start the feScan frontend by entering the following command in the last terminal: "./feScan"
18.) If the "Status" of each equipment on the midptf experiment page appears green, run the scan by selecting Start->Start(ignore settings)
19.) Monitor the progress of the scan in each of the frontend terminals
20.) Once the end of the run has been reached, stop each frontend by pressing Ctrl+C
21.) To retrieve the data produced by the scan, open a terminal in the packages/rootana/libAnalyzer directory and enter "./analyzer_example.exe ~/online/data/run#.mid.gz" where your run# can be found on the midptf experiment page
22.) Copy these results into a .txt file and proceed to the data analysis section below

Post-Scan Checklist:

- Return all settings to their defaults as shown in the images above
- Clean up desktop of PC
- Log off PC
- Turn off all devices

2.4– Results

2.4.1 – Results from Repeatability Testing

Table 1 contains variation measurements from the destination tests. Table 2 contains variation measurements from the velocity tests. Figures 24 and 25 show plots of this data. Table 3 contains variation measurements from the acceleration tests. All benchmarks were made using the feMove settings that appear in section 2.3.4. All measurements were made with digital calipers that have an error of +/- 0.1 mm.

Table 1: *Variation measurements from destination tests*

Axis	Destination (m)	Trial 1 (mm)	Trial 2 (mm)	Trials 3 (mm)	Trial 4 (mm)	Trial 5 (mm)
X	-0.36	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
X	0.0	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
X	0.36	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
Y	0.32	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
Y	0.0	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
Y	-0.32	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
Z	-0.25	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
Z	0.0	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
Z	0.25)	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
Z	0.25 (from opposite direction)	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
Z	0.0 (from opposite direction)	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
Z	-0.25 (from opposite direction)	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5

Table 2: *Variation measurements from velocity tests*

Axis	Velocity (m/s)	Trial 1 (mm)	Trial 2 (mm)	Trial 3 (mm)
X	0.02	< 0.5	< 0.5	< 0.5
X	0.04	< 0.5	< 0.5	< 0.5
X	0.08	-2.23	-2.24	-2.20
X	0.07	-1.31	-1.32	-1.31
X	0.06	-0.70	-0.70	-0.70
Y	0.06	-0.72	-0.73	-0.71
Y	0.07	-1.44	-1.44	-1.46
Z	-0.06 (down)	1.65	1.65	1.66
Z	0.06	-1.67	-1.67	-1.67
Z	-0.05	-1.84	-1.86	-1.85
Z	-0.04	-0.77	-0.77	-0.77
Z	0.05	< 0.5	< 0.5	< 0.5

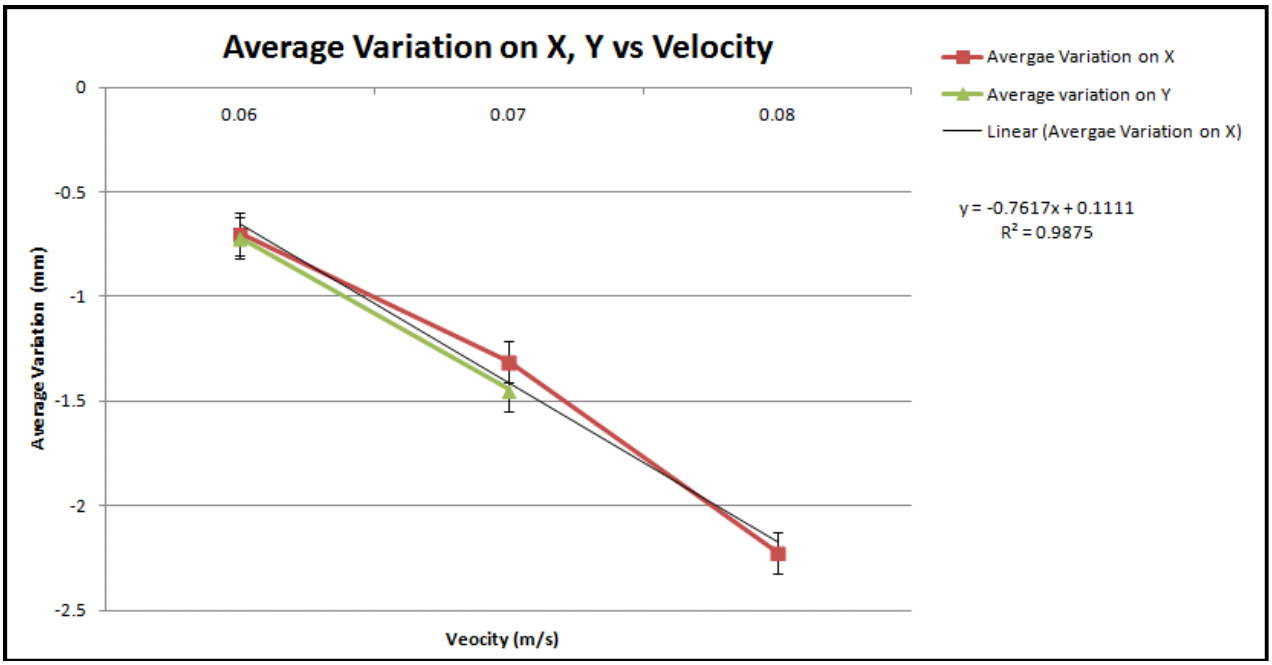


Figure 24: *Average Variation on X,Y versus Velocity*

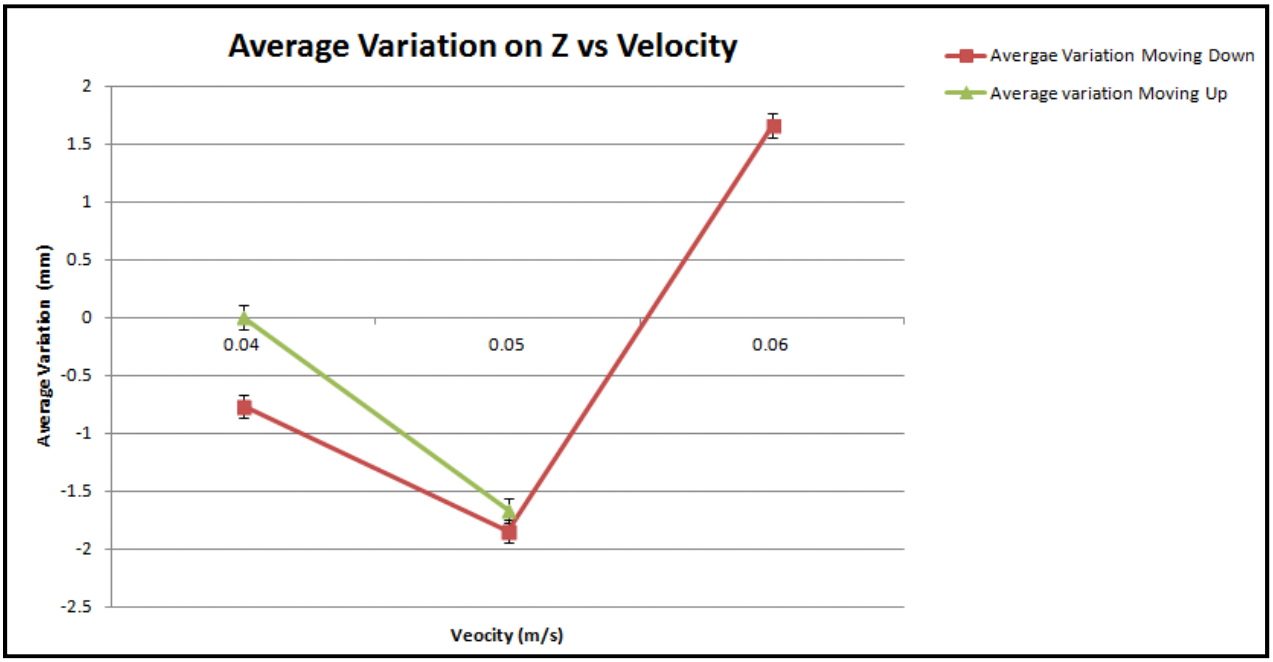


Figure 25: *Average Variation on Z versus Velocity*

Table 3: *Variation measurements from acceleration tests*

Axis	Acceleration (m/s^2)	Trial 1 (mm)	Trial 2 (mm)	Trial 3 (mm)
X	0.4	< 0.5	< 0.5	< 0.5
X	0.8	< 0.5	< 0.5	< 0.5
X	1.6	< 0.5	< 0.5	< 0.5
Y	0.4	< 0.5	< 0.5	< 0.5
Y	0.8	< 0.5	< 0.5	< 0.5
Y	1.6	< 0.5	< 0.5	< 0.5
Z	0.4	< 0.5	< 0.5	< 0.5
Z	0.8	< 0.5	< 0.5	< 0.5
Z	1.6	< 0.5	< 0.5	< 0.5

Testing notes:

- Negative variation is undershoot, positive is overshoot
- Movement more stable along Y axis than X axis
- Vibration of arm is reduced when speed of Z axis increased
- Fully extending the arm during X-Y tests had no effect
- Assumed direction doesn't matter for X-Y

2.4.2 – Results of Magnetic Surveys

In total, seven Magnetic Surveys were performed:

1.) High Resolution Vertical Scan @ X,Y = 0
2.) Three consecutive scans of X-Y plane @ Z=0
3.) Scan of X-Y plane @ Z=0
4.) Scan of Y-Z plane @ X=0
5.) Scan of X-Z plane @ Y=0
6.) Full Cylindrical Scan
7.) Full Rectangular Prism Scan

The table below contains the settings for each scan (Table 4). All other settings found in feMotor, feMove, feScan and fedvm are the same as what appears in the section 2.3.4 above. Table 5 contains additional information about each scan.

Table 4: *Scan Settings (all settings in m)*

#	prism_height_z	prism_length_x	prism_width_y	z_step	x_step	y_step	Init_pos_z	Init_pos_x	Init_pos_y
1	0.50	0.0	0.0	0.005	0.08	0.08	-0.26	0.0	0.0
2	0.0	0.72	0.64	0.05	0.08	0.08	0.0	-0.36	0.32
3	0.0	0.72	0.64	0.05	0.08	0.08	0.0	-0.36	0.32
4	0.50	0.0	0.64	0.05	0.08	0.08	-0.25	0.0	0.32
5	0.50	0.72	0.0	0.05	0.08	0.08	-0.25	-0.36	0.0
7	0.50	0.72	0.64	0.05	0.08	0.08	-0.25	-0.36	0.32
#	height	radius	arc_step	loop_separation		layer_thickness			
6	0.50	0.32	0.05	0.04		0.05			

Table 5: *Additional Scan Information*

#	Date Performed	# of Points	# of Measurements Per Position	Approximate Duration (hrs)	Filename
1	Mar. 25	104	5	-	Vertical Scan(high res).xlsx
2	Apr. 12	3 x 91	3	3 x 0.5	3 x X-Y @ Z=0.xlsx
3	Apr. 11	91	3	0.5	X-Y @ Z=0.xlsx
4	Apr. 11	110	3	0.5	Y-Z @ X=0.xlsx
5	Apr. 11	101	3	0.5	X-Z @ Y=0.xlsx
6	Apr. 4, Apr. 5	2080	3	7.5	Final Cylindrical Scan Data.xlsx
7	Apr. 5	992	3	5	Final Prism Scan Data.xlsx

2.4.3 – Data Analysis

The data produced by the scans appears as follows (Table 6). To get it in this form, you must import the .txt file you saved during the procedure in section 2.3.4 into Excel. To do this, select Data->From Text->Filename of .txt file->Open->Delimited->Next->Space->Next->Finish->Cell to paste data.

Table 6: *Example of Data*

Point #	Measurement #	X pos. (m)	Y pos. (m)	Z pos. (m)	B_x	B_y	B_z	Time
1	0	0	0	0	0.457498	0.957078	-0.271134	0
1	1	0	0	0	0.457474	0.957084	-0.271128	0
1	2	0	0	0	0.457549	0.957035	-0.271102	0
2	0	-0.36	0.32	-0.25	0.349561	0.999212	-0.031188	0
2	1	-0.36	0.32	-0.25	0.34949	0.999216	-0.0311526	0
2	2	-0.36	0.32	-0.25	0.349281	0.999268	-0.0311614	0
3	0	-0.28	0.32	-0.25	0.355708	0.981342	-0.0202075	0
3	1	-0.28	0.32	-0.25	0.355673	0.981283	-0.0201956	0
3	2	-0.28	0.32	-0.25	0.355605	0.981318	-0.0201915	0

To make it easier to plot the data, filter it so that only the last measurement for each point is shown. This measurement should be the most accurate. To do this, highlight the measurement number column and select Data->Filter. From the drop-down menu that appears over the column, select “2” only.

To verify the path generated by the “gen_scan_path” routine, the 3D path can be plotted by selecting only the position values in the Excel sheet. This report will use gnuplot 4.2.3 to make 3D plots. To do this, copy the position values into a .txt file and rename it 3Dpath.tsv. Then, opening gnuplot, enter the following commands:

```
set size ratio 1
set key off
set ticslevel 0
set view 63,45,1,1
set size 1.6, 1.3
set origin -0.3,-0.1
splot
'C:\Users\Harish\Desktop\Archive\School\Tools\Gnuplot\gp423win32\gnuplot\bin\share\ENP
H 479\3Dpath.tsv' using 1:2:(-1*$3) with lines
```

Doing this on the cylindrical, rectangular prism and plane data sets, we get Figures 26, 27, 28, 29 and 30. To get a better idea of the data point coverage of each layer in the volume scans, enter the following commands:

```
set view map
set size 1.6, 1.3
set origin -0.3,-0.15
replot
```

This gives Figures 31 and 32. Finally, if we want to see the layer coverage of the entire volume, a profile view is obtained using the following commands (Figures 33 and 34):

```
set view 90,0
set ticslevel 0.1
set size 1.6, 1.6
set origin -0.3,-0.3
unset ytics
replot
```

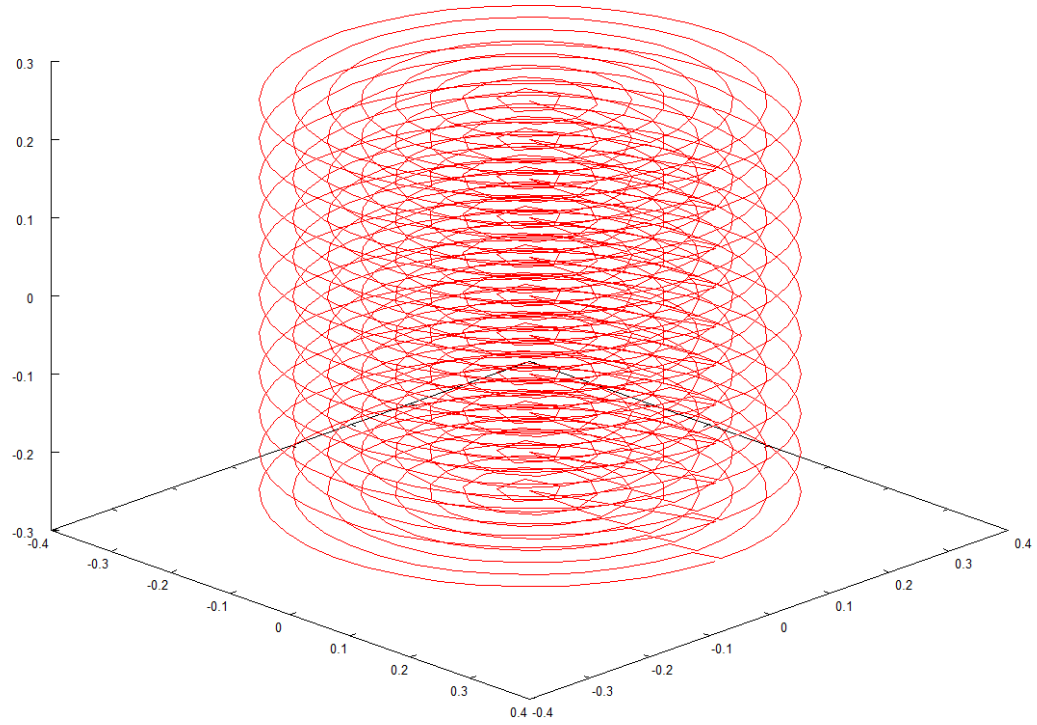


Figure 26: *Plot of 3D path for cylindrical scan*

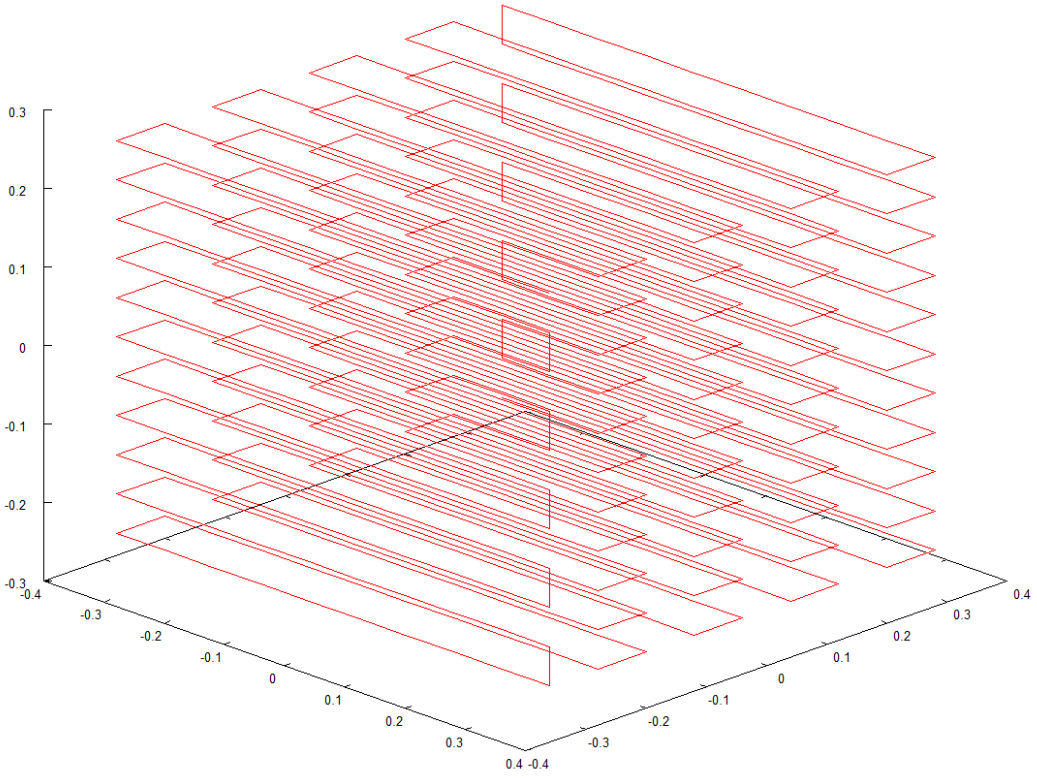


Figure 27: *Plot of 3D path for rectangular prism scan*

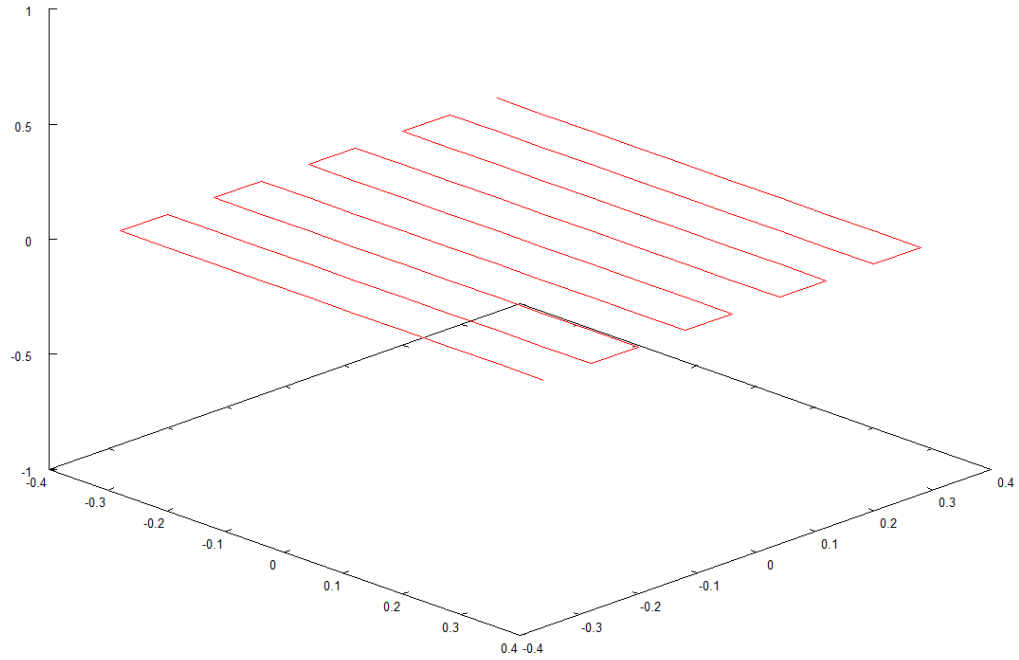


Figure 28: *Plot of 3D path for X-Y plane scan*

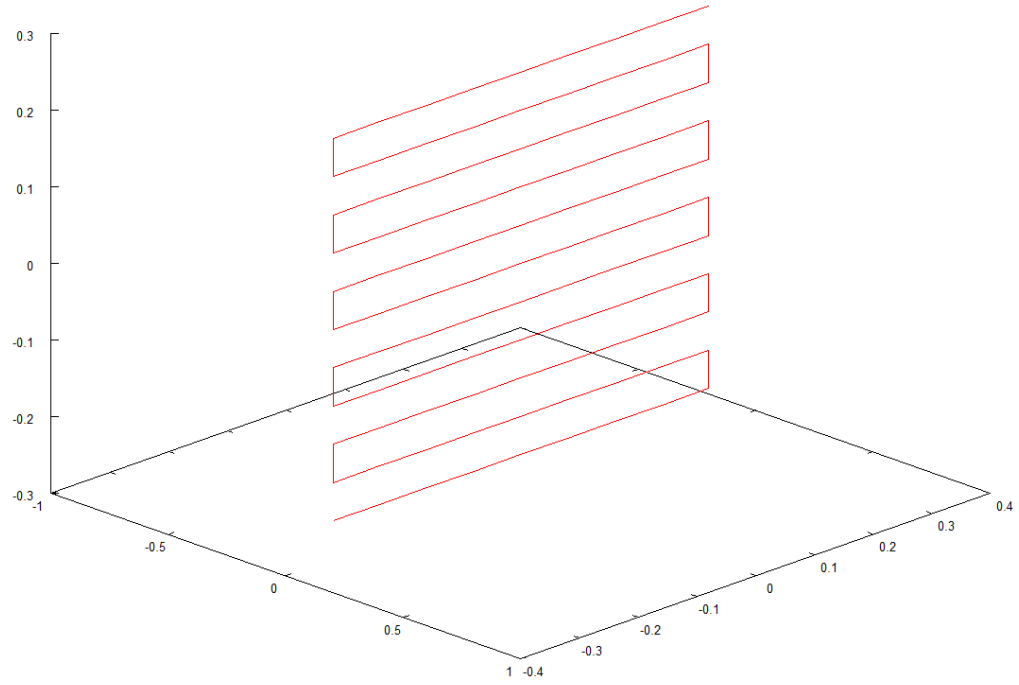


Figure 29: *Plot of 3D path for X-Z plane scan*

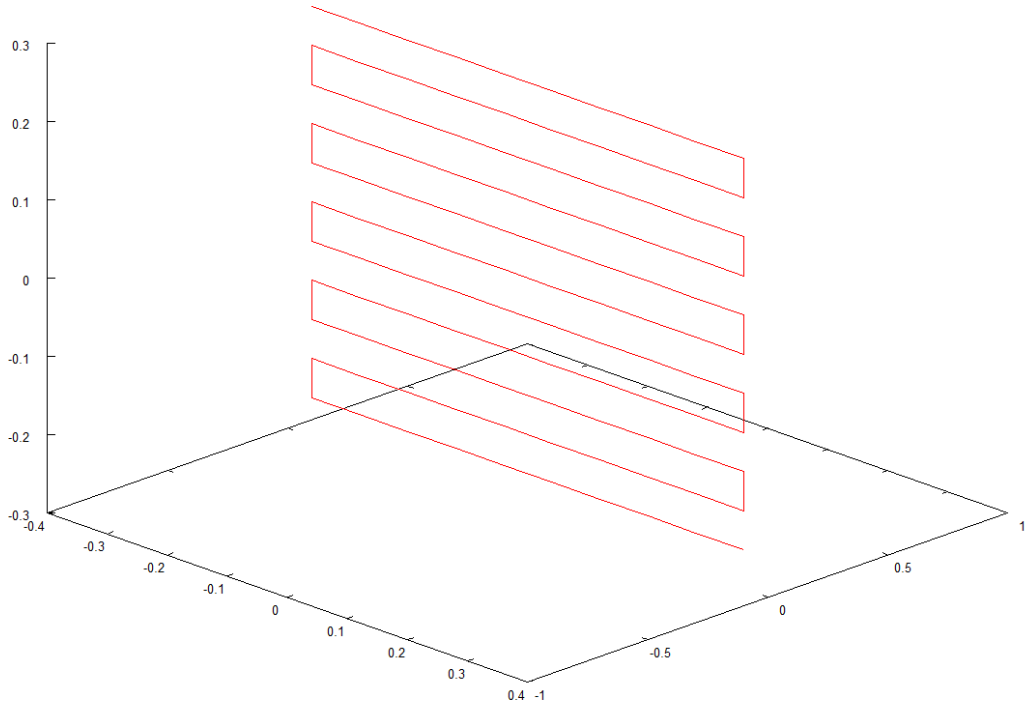


Figure 30: *Plot of 3D path for Y-Z plane scan*

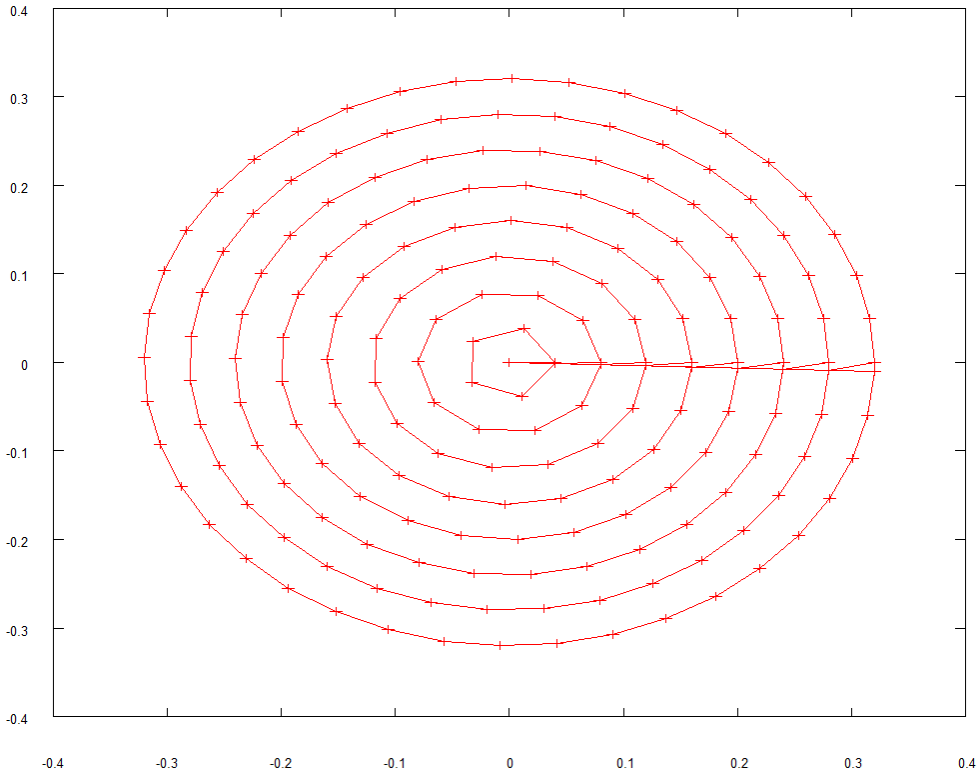


Figure 31: *Plot of layer path for cylindrical scan*

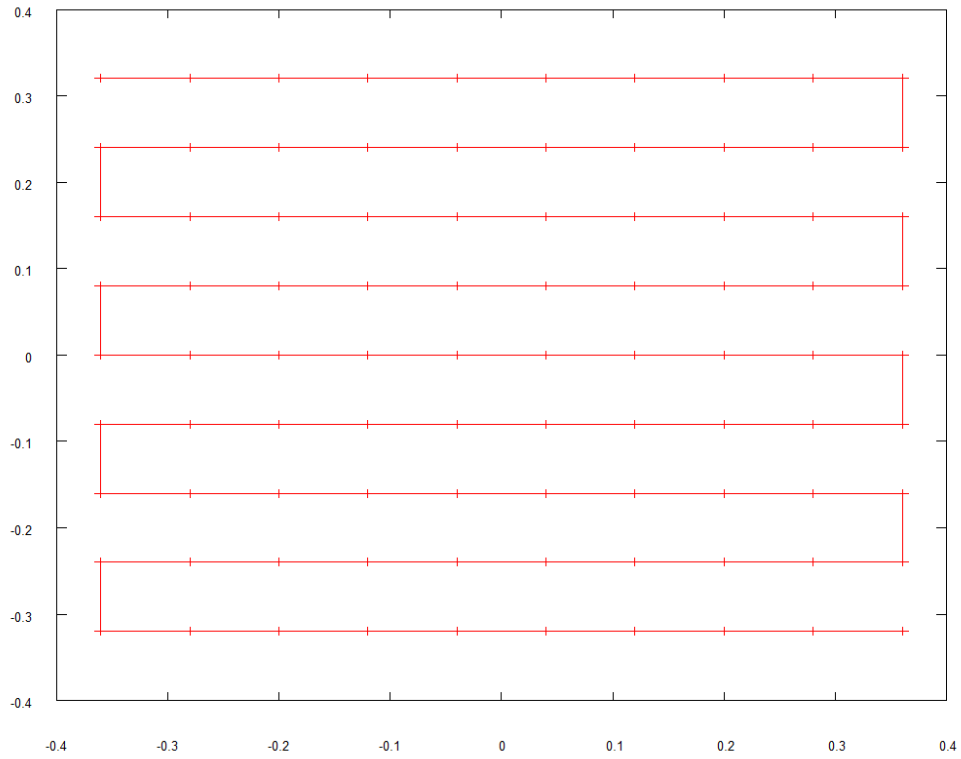


Figure 32: *Plot of layer path for rectangular prism scan*

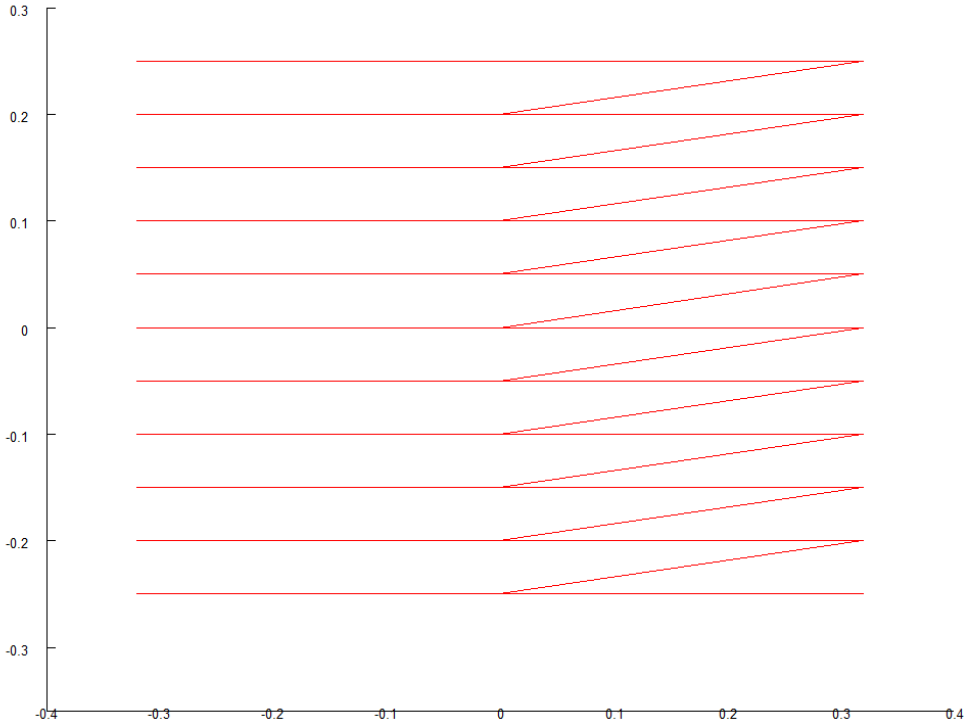


Figure 33: *Plot of vertical path for cylindrical scan*

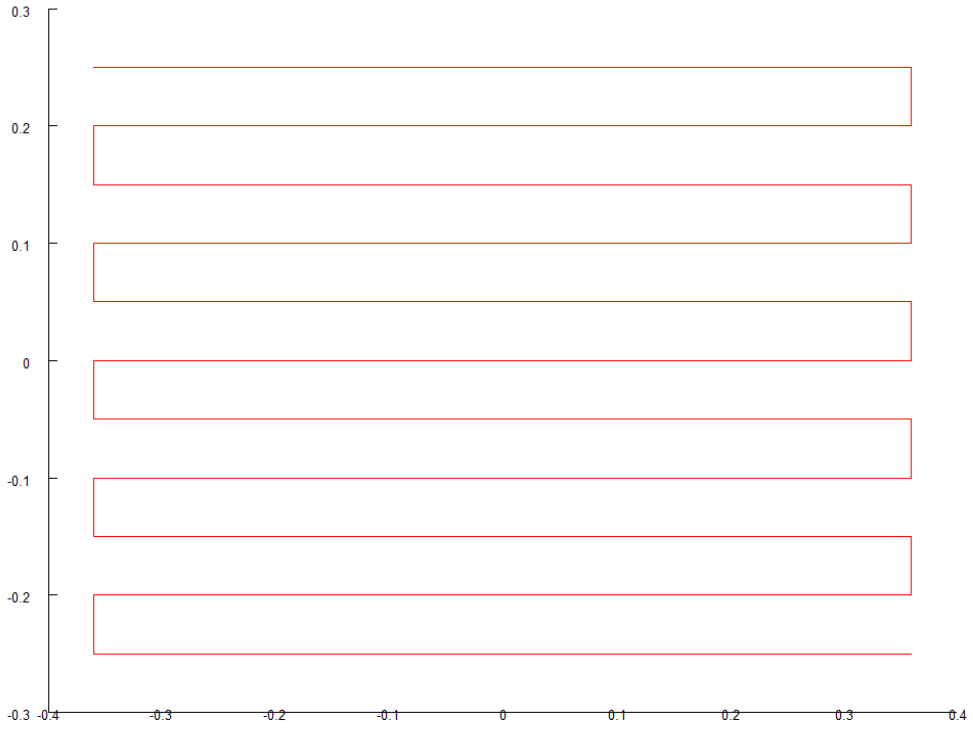


Figure 34: *Plot of vertical path for rectangular prism scan*

To plot the field measurements, we need to calculate the magnitude of the magnetic field at each point. To do this, copy the filtered data into a separate spreadsheet and insert the following formula:

$$B_{mag} = 88.4 * \sqrt{B_x^2 + B_y^2 + B_z^2}$$

For the vertical scan, we can plot this directly in Excel (Figure 35). However, for the other scans, it is necessary to plot $B_{mag}(x, y, z)$ with one of the axes fixed. This is done by only using the data for a single layer. Therefore, select the data for the desired layer and copy it into a .txt file. Similar to before, rename it FeildStrength.tsv and enter the following commands into gnuplot:

set size ratio 1
set key off
set view 63,225,1,1
set size 1.6, 1.3
set origin -0.3,-0.1
splot
'C:\Users\Harish\Desktop\Archive\School\Tools\Gnuplot\gp423win32\gnuplot\bin\share\ENP H 479\FeildStrength.tsv' using 1:2:3 with points pointtype 5 pointsize 1 palette linewidth 10

Using layer Z=0 in the cylindrical scan, this gives Figure 36.

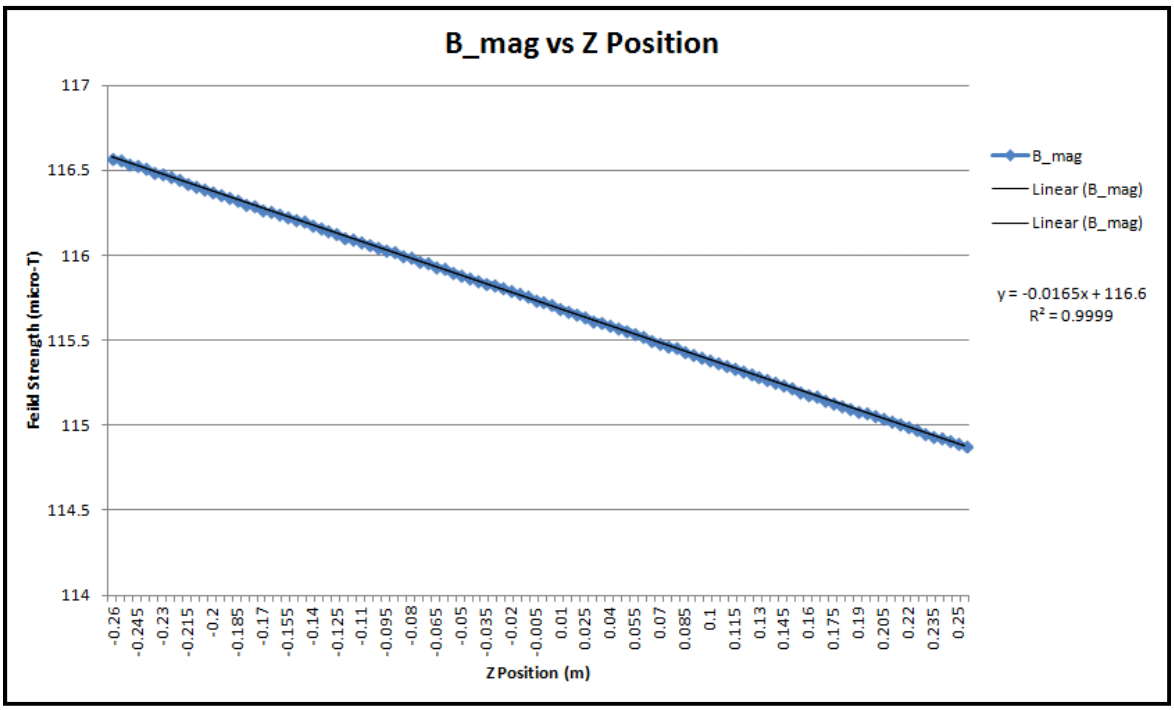


Figure 35: *Plot of Magnetic Field Strength versus Z Position*

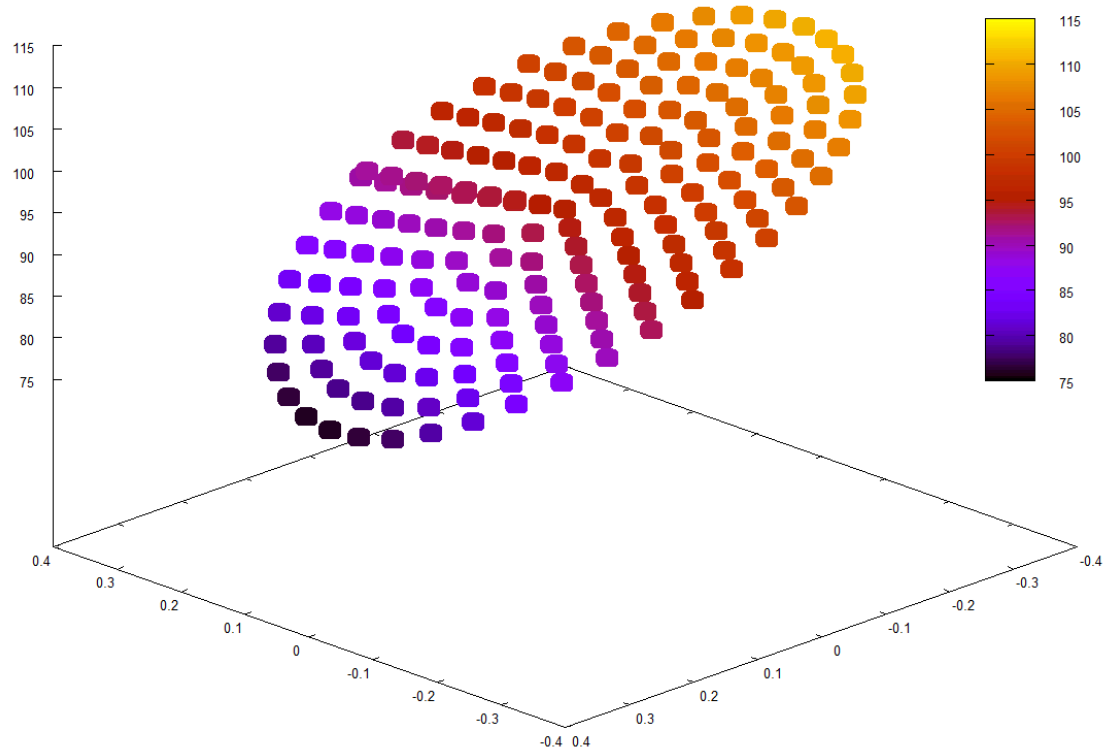


Figure 36: *Plot of B_mag(x,y) @ Z=0 (3D points)*

Interpolation can be used to turn these points into a surface (Figure 37). This is done by entering the following commands:

set dgrid3d
set pm3d at s
unset surface
replot

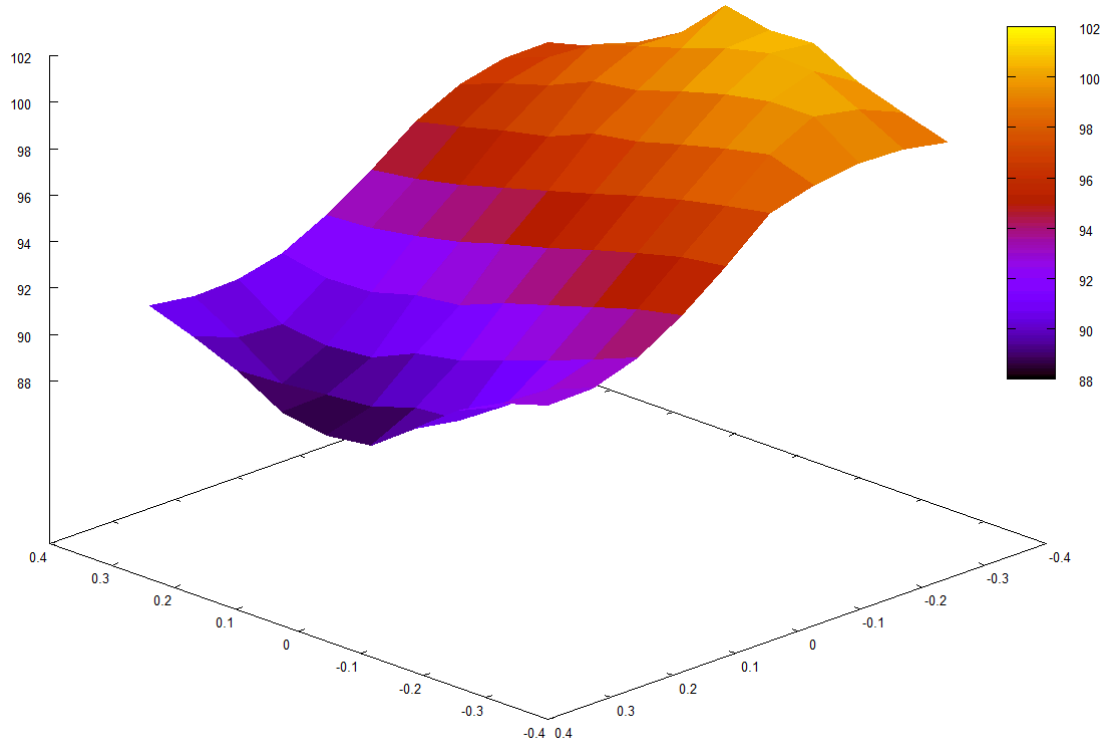


Figure 37: *Plot of B_mag(x,y) @ Z=0 (3D surface)*

To view the gradient better, we can view this surface top-down (Figures 38 and 39) by entering the following commands:

set view map
set size 1.5, 1.2
set origin -0.25,-0.1
set title "B_mag(x,y) at Z=0" -0.3,0.45
set xlabel "X Position (m)" 0,0.5
set ylabel "Y Position (m)" 2, 0
set label 1 "Field Strength (micro-T)" at 0.33, 0.44 l
replot

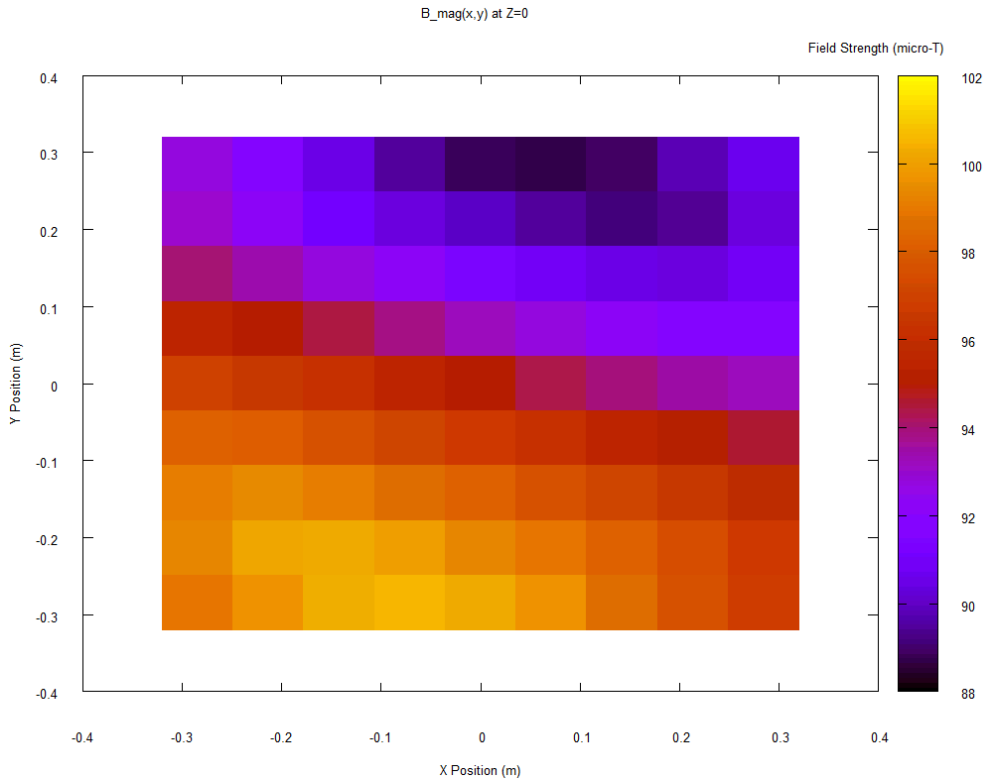


Figure 38: *Plot of B_mag(x,y) @ Z=0 for cylindrical scan*

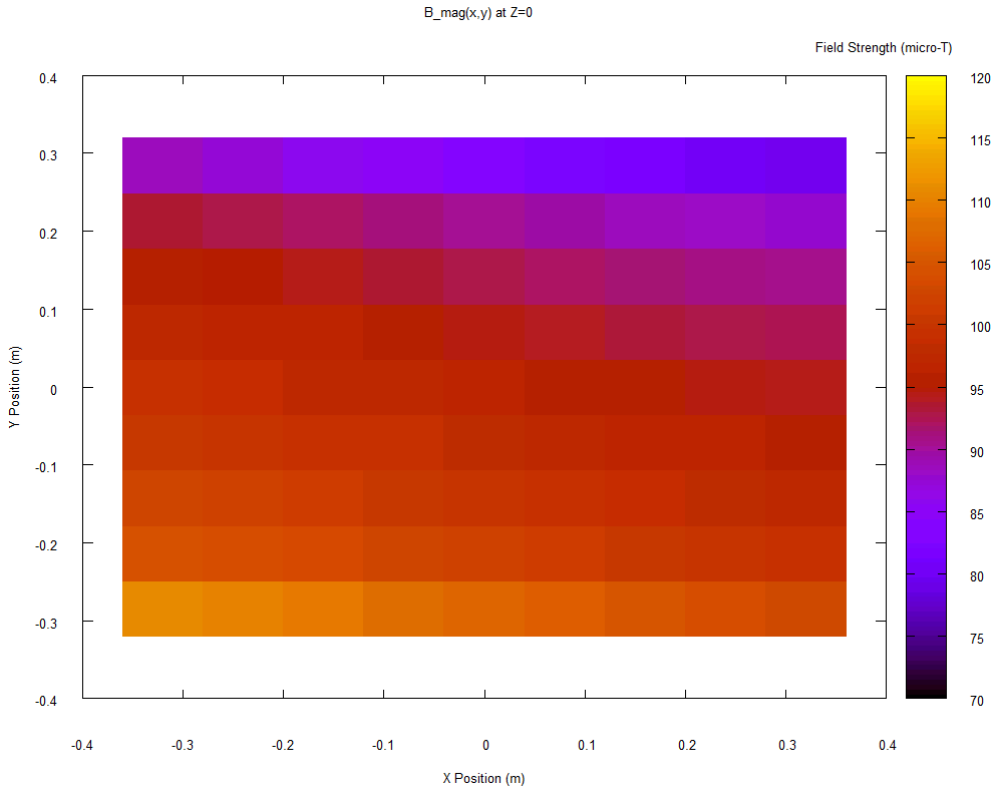


Figure 39: *Plot of B_mag(x,y) @ Z=0 for rectangular prism scan*

If contours are preferred (Figure 40), enter the following commands:

unset pm3d
set contour base
set cntrparam bspline
set cntrparam levels 20
set key 0.47,0.38
splot'C:\Users\Harish\Desktop\Archive\School\Tools\Gnuplot\gp423win32\gnuplot\bin\share\ENPH 479\FeildStrength.tsv' using 1:2:3 with lines

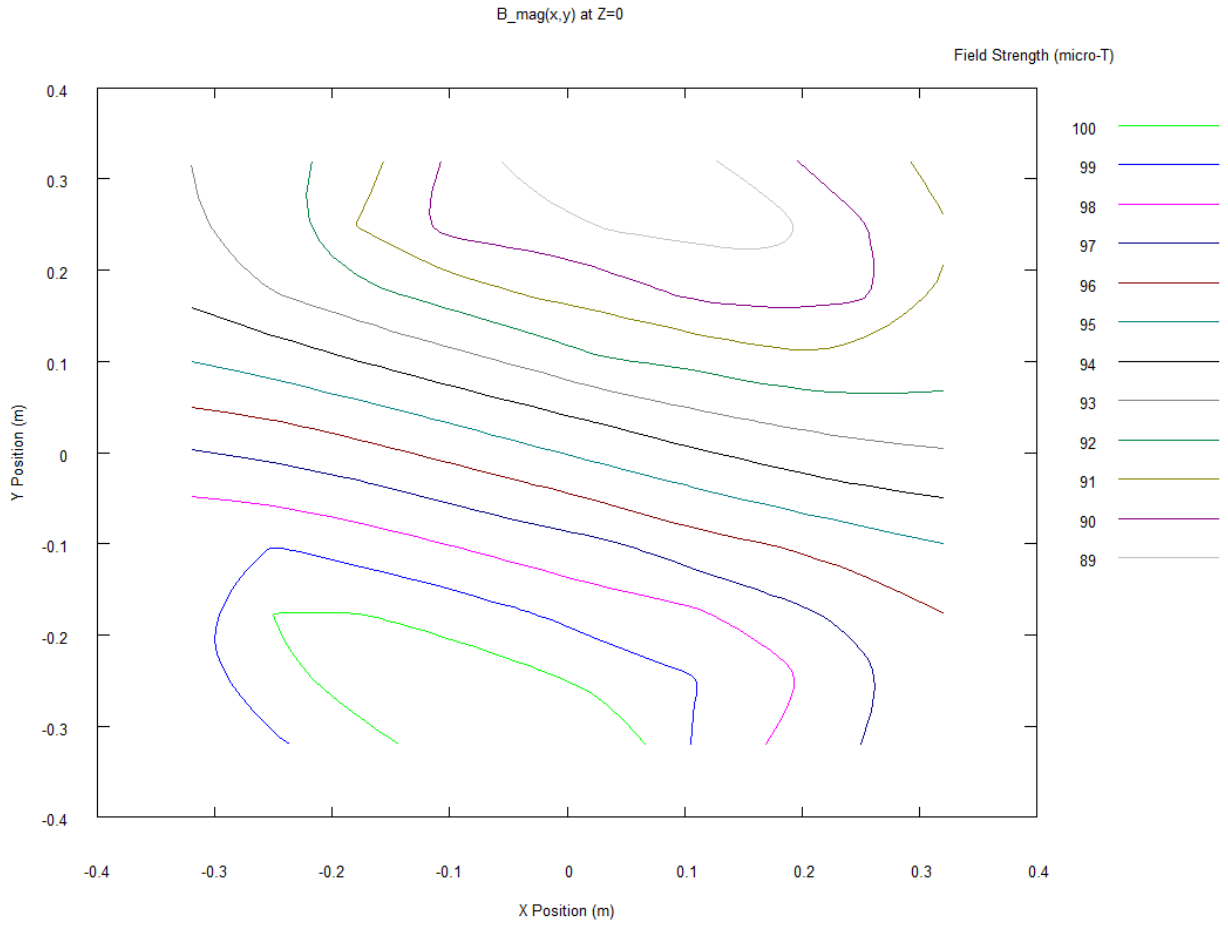


Figure 40: *Plot of B_mag(x,y) @ Z=0 (contours)*

Figure 41 is of the three consecutive scans of the X-Y plane at Z=0. Figure 42 is of the scans of the various planes.

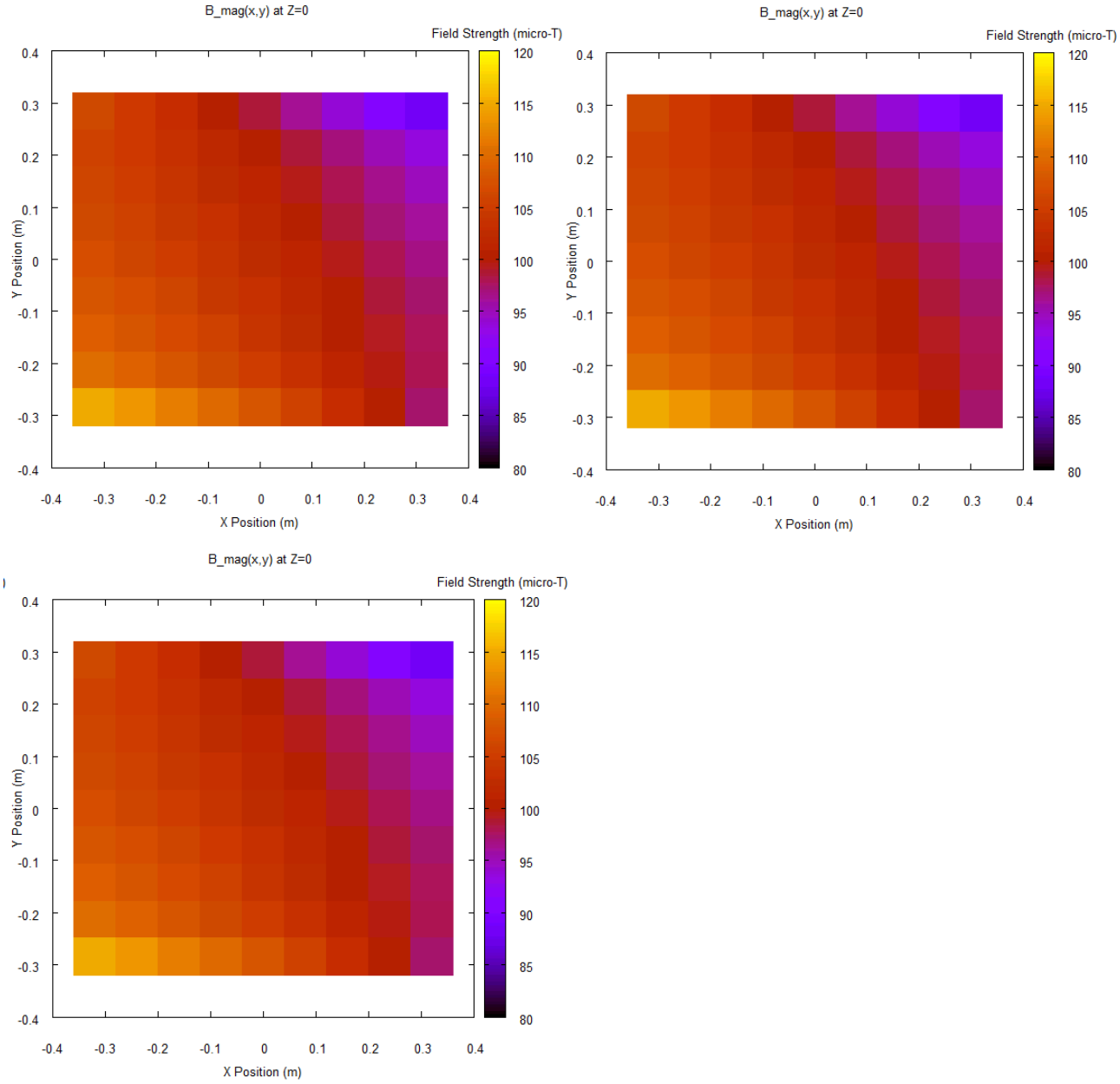


Figure 41: Plots of t*hree consecutive scans of the X-Y plane at Z=0*

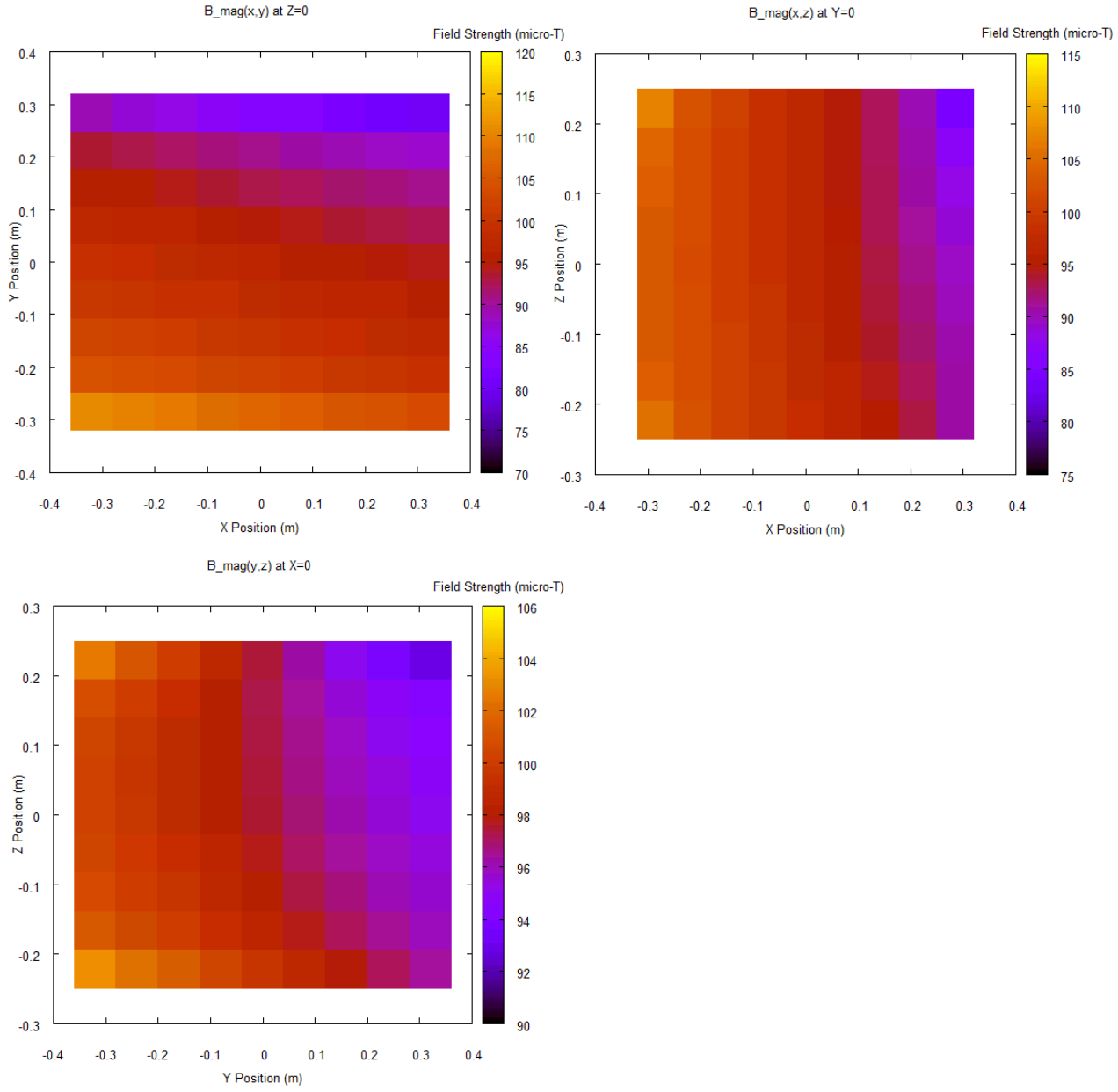


Figure 41: Plots of various planes centered about the midpoint

2.5– Discussion of Results

2.5.1 – Repeatability Testing

The destination trials in the repeatability testing showed that distance travelled has no effect on accuracy. Specifically, all variations for all destinations were less than 0.5 mm (see Table 1). In other words, they were indistinguishable from each other. The velocity trials showed that position error, in the form of under-shoot, increases linearly with speed along the X and Y axes (Figure 24). This under-shoot is thought to be the result of the initialization. Specifically, during initialization, the axes are sent to positions beyond their limit switches. This means that the axes do not decelerate on approach. Therefore, within the finite time the axis limits take to switch, the axis is moving at full speed. As a result, distance is covered in this time which shifts the resulting origin. At low speeds, this shift is small but at high speeds it can become significant. This effect, however, should eventually saturate at the height of the switch lever, approximately 3.11 +/- 0.1 mm. Furthermore, the velocity trials showed that the position error on the Z axis depends on direction. Specifically, in the upward direction, the variation behaves like that of the X and Y axes but in the downward direction it transitions from under-shoot to over-shoot at high speeds (Figure 25). Therefore, the velocity should not exceed 0.04 m/s on the Z axis and 0.06 m/s on the X and Y axes, in order to maintain the +/- 1 mm tolerances of the Testing Rig. The acceleration trials showed no change in position error with increasing speeds. This is most likely due to the acceleration not being set high enough. However, trials were stopped in fear of damaging the system.

2.5.2 – Hall Probe Verification Results

The Hall Probe was successfully verified in the two ways described in section 2.3.2. First, the three consecutive scans of the X-Y plane showed that the measurements taken from the Hall Probe were, indeed, repeatable. Secondly, the values measured by the Hall Probe were tested by comparing the minimum value of the 3D Magnetic Field Surveys to the expected value of the Earth's magnetic field. Out of all recorded data sets, the minimum Field Strength was determined to be 69.3 micro-T while the Earth's magnetic field was determined to be 54.7377 micro-T. The latter value was obtained using 49°14'50.54"N/123°13'47.60"W and 60 m for the coordinates and elevation of TRIUMF, respectively. Therefore, the minimum value is around 15 micro-T off the expected value. This discrepancy can be accounted for by additional magnetic interference present in the lab.

2.5.3 – Scan Routines

As shown by Figures 26-34, the scan paths were generated as expected. Surprisingly, however, the Magnetic Field data (Figures 38-41) showed an increasing gradient away from the cyclotron and towards the North wall of the lab. This might be expected if the cyclotron were turned off during the dates of the scans, as the magnetic fields produced by the lab equipment and shop next door might become significant. This should be investigated further, knowing the dates of operation for the cyclotron and the magnitude of the fields produced. Lastly, one severe limitation of the Hall Probe data is the absence of error values for the measurements.

3.0 - Conclusions

All mechanical improvements requested by the project sponsors were completed successfully. However, the fixing of the free-swinging counter-weight was delayed and eventually abandoned in lieu of more urgent tasks and the installation of the Helmholtz coils was left incomplete due to time constraints. Additionally, all wiring improvements requested by the project sponsors were completed successfully.

The repeatability testing for the fully assembled manipulator arm was completed successfully. The first conclusion reached by this testing was that destination has no effect on the repeatability of movements for the full manipulator arm. That is, varying destinations resulted in variations that were <0.5 mm or indistinguishable by eye. Furthermore, it was determined that increasing the velocity in the +X, +Y and –Z (upward) directions caused under-shoot while increasing the velocity in the +Z (downward) direction caused under-shoot and then over-shoot. To ensure the Testing Rig tolerances of +/-1 mm, the velocity should not exceed 0.04 m/s on the Z axis and 0.06 m/s on the X and Y axes. Lastly, it was determined that acceleration also had no effect on the repeatability of movements.

The Hall Probe data was successfully verified as reasonable by comparing the lowest measured field strength to the expected value for the Earth's magnetic field. These values were 69.3 micro-T and 54.7377 micro-T, respectively. The approximately 15 micro-T discrepancy is thought to be caused by magnetic interference from the lab equipment.

The Initialization routine was determined as being successful. This is evident through its constant use during the repeatability testing and Magnetic Field Surveys. Furthermore, the generating of cylindrical, rectangular prism and plane scans by the “gen_scan_path” routine was determined as successful. This is evident by Figures 26-34 above. Lastly, while the Magnetic Field Surveys were successful, they produced the unexpected result of an increasing field gradient away from the cyclotron and towards the North wall of the lab. This will need to be investigated further with additional information about the cyclotron. The Magnetic Survey, with the Helmholtz coils in place, was left incomplete due to time constraints.

4.0– Project Deliverables

4.1 – List of Deliverables

In addition to the Final Engineering Recommendation Report, the deliverables for the project will be as follows:

Testing Rig Improvements:

1.) The Testing Rig
- counter-weight not fixed
- Helmholtz coils not installed due to time constraints

3D Magnetic Field Survey:

2.) Path Generator C code and documentation
    - Code on lab computer/in appendices
    - Extensive documentation in this report
3.) Results from movement repeatability testing
    - Results tabulated in this report
4.) Results from Hall Probe verification testing, including error analysis
    - Results in this report
    - No quantitative error analysis
5.) Results from Magnetic Field Survey without Helmholtz coils (plot)
    - Found inside this report
6.) Results from Magnetic Field Survey with Helmholtz coils (plot)
    - not completed due to time constraints

4.2 – Financial Summary

No costs were incurred by the project team during this project.

4.3 – Ongoing Commitments by Team Members

Software support for the routines in this report will be provided through email for the next year.

5.0 - Recommendations

1. In addition to the planned continuation of Testing Rig construction, I recommend that fixing the free-swinging counter-weight be revisited. During the performance of the scans, the amplitude of the swings would increase as time went on. In order to maintain the accuracy of the field measurements, I was forced to stabilize the weight by hand.
2. In order to maintain the +/- 1 mm translational tolerances of the Testing Rig, I recommend that the velocity not exceed 0.04 m/s on the Z axis and 0.06 m/s on the X and Y axes.
3. I recommend that a thorough verification of the Hall Probe be performed. Specifically, its readings should be compared with a source of known strength, complete with a full error analysis. It is only after this that the results of the Magnetic Field Surveys can be used to set the currents of the Helmholtz coils.
4. As discussed with the project sponsors, I recommend that ODB settings be added to the feMove and feScan user interfaces. Specifically, I recommend that a limit enable array be added to the feMove settings and that all of the scan parameters in the gen_scan_path routine be added to the feScan settings.
5. Lastly, for the future, I recommend that the cylindrical scan routine be used for the Magnetic Survey with the Helmholtz coils in place. This scan is best suited to the geometry of the water tank, which should be installed by that point.

6.0 - References

1) http://en.wikipedia.org/wiki/Super_Kamiokande

2) Smit-Anseeuw, Zimmer. “Proposal for a Manipulator Arm Control System for Characterizing Large-Area Photosensors.” September 19, 2012.

3) http://en.wikipedia.org/wiki/Thin-film_interference

4) http://en.wikipedia.org/wiki/Anti-reflective_coating

5) http://en.wikipedia.org/wiki/Photomultiplier

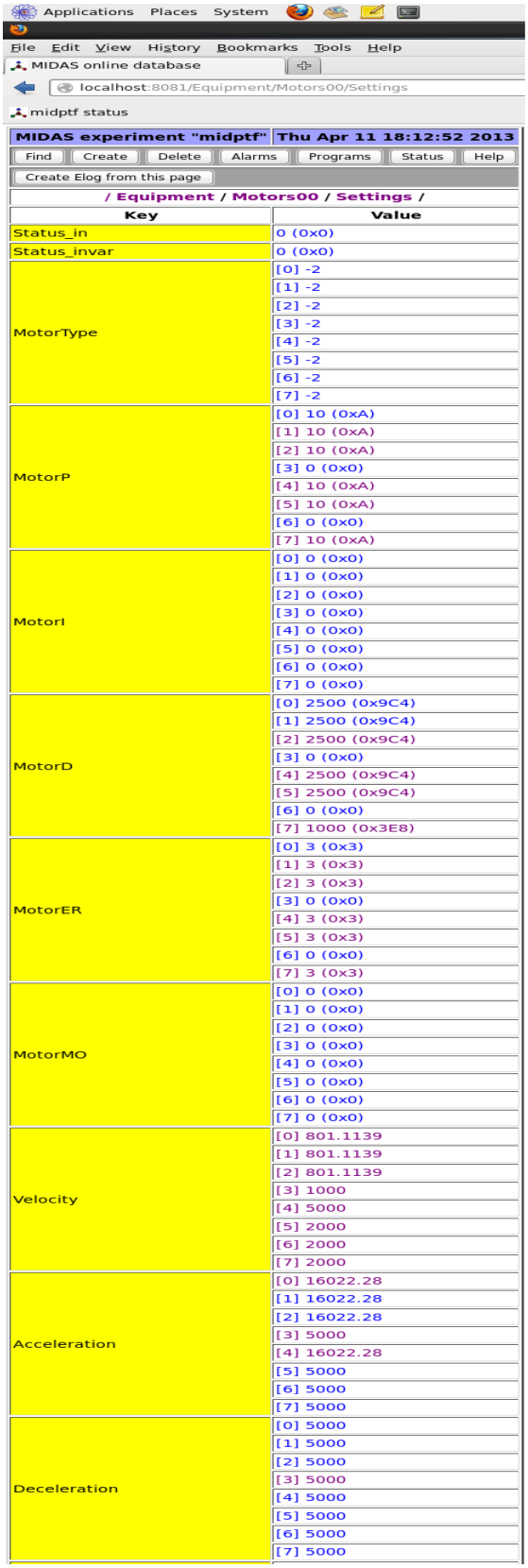
6) Galil motor controller reference. http://www.galilmc.com/products/dmc-41x3.php

7) MIDAS homepage. http://midas.psi.ch/

8) Smit-Anseeuw, Zimmer. “Manipulator Arm Control System for Characterizing Large-Area Photosensors.” February 16, 2013.

9) National Geophysical Data Center (NGDC) Magnetic Field Calculators. http://www.ngdc.noaa.gov/geomag-web/#igrfwmm

7.0 – Appendix A: feMotor Settings



Key	Value
Status_in	0 (0x0)
Status_invar	0 (0x0)
MotorType	[0] -2 [1] -2 [2] -2 [3] -2 [4] -2 [5] -2 [6] -2 [7] -2
MotorP	[0] 10 (0xA) [1] 10 (0xA) [2] 10 (0xA) [3] 0 (0x0) [4] 10 (0xA) [5] 10 (0xA) [6] 0 (0x0) [7] 10 (0xA)
MotorI	[0] 0 (0x0) [1] 0 (0x0) [2] 0 (0x0) [3] 0 (0x0) [4] 0 (0x0) [5] 0 (0x0) [6] 0 (0x0) [7] 0 (0x0)
MotorD	[0] 2500 (0x9C4) [1] 2500 (0x9C4) [2] 2500 (0x9C4) [3] 0 (0x0) [4] 2500 (0x9C4) [5] 2500 (0x9C4) [6] 0 (0x0) [7] 1000 (0x3E8)
MotorER	[0] 3 (0x3) [1] 3 (0x3) [2] 3 (0x3) [3] 0 (0x0) [4] 3 (0x3) [5] 3 (0x3) [6] 0 (0x0) [7] 3 (0x3)
MotorMO	[0] 0 (0x0) [1] 0 (0x0) [2] 0 (0x0) [3] 0 (0x0) [4] 0 (0x0) [5] 0 (0x0) [6] 0 (0x0) [7] 0 (0x0)
Velocity	[0] 801.1139 [1] 801.1139 [2] 801.1139 [3] 1000 [4] 5000 [5] 2000 [6] 2000 [7] 2000
Acceleration	[0] 16022.28 [1] 16022.28 [2] 16022.28 [3] 5000 [4] 16022.28 [5] 5000 [6] 5000 [7] 5000
Deceleration	[0] 5000 [1] 5000 [2] 5000 [3] 5000 [4] 5000 [5] 5000 [6] 5000 [7] 5000
Jog Velocity	[0] 1000 [1] 1000 [2] 1000 [3] 1000 [4] 1000 [5] 1000 [6] 1000 [7] 1000
EncoderPosition	[0] 0 [1] 0 [2] 0 [3] 0 [4] 0 [5] 0 [6] 0 [7] 0
Move	[0] n [1] n [2] n [3] n [4] n [5] n [6] n [7] n
Advance	[0] n [1] n [2] n [3] n [4] n [5] n [6] n [7] n
Jog Pos	[0] n [1] n [2] n [3] n [4] n [5] n [6] n [7] n
Jog Neg	[0] n [1] n [2] n [3] n [4] n [5] n [6] n [7] n
Home	[0] n [1] n [2] n [3] n [4] n [5] n [6] n [7] n
PHome	[0] n [1] n [2] n [3] n [4] n [5] n [6] n [7] n
Bounce	[0] n [1] n [2] n [3] n [4] n [5] n [6] n [7] n
Stop	[0] n [1] n [2] n [3] n [4] n [5] n [6] n [7] n

Index Position	[0] n
	[1] n
	[2] n
	[3] n
	[4] n
	[5] n
	[6] n
	[7] n
Slope	[0] -1
	[1] -1
	[2] -1
	[3] -1
	[4] -1
	[5] -1
	[6] -1
	[7] -1
Offset	[0] 0
	[1] 0
	[2] 0
	[3] 0
	[4] 0
	[5] 0
	[6] 0
	[7] 0
Encoder Slope	[0] 1
	[1] 1
	[2] 1
	[3] 1
	[4] 1
	[5] 1
	[6] 1
	[7] 1
Encoder Offset	[0] 0
	[1] 0
	[2] 0
	[3] 0
	[4] 0
	[5] 0
	[6] 0
	[7] 0
PressThresh	[0] 0
	[1] 0
	[2] 0
	[3] 0
	[4] 0
	[5] 0
	[6] 0
	[7] 0
LimitPolarity	[0] -1 (0xFFFFFFFF)
	[1] -1 (0xFFFFFFFF)
	[2] -1 (0xFFFFFFFF)
	[3] -1 (0xFFFFFFFF)
	[4] -1 (0xFFFFFFFF)
	[5] -1 (0xFFFFFFFF)
	[6] -1 (0xFFFFFFFF)
	[7] -1 (0xFFFFFFFF)
MotorCurrent	[0] 2 (0x2)
	[1] 2 (0x2)
	[2] 2 (0x2)
	[3] 2 (0x2)
	[4] 2 (0x2)
	[5] 2 (0x2)
	[6] 2 (0x2)
	[7] 2 (0x2)

HoldCurrent	[0] 1 (0x1)
	[1] 1 (0x1)
	[2] 1 (0x1)
	[3] 1 (0x1)
	[4] 1 (0x1)
	[5] 1 (0x1)
	[6] 1 (0x1)
	[7] 1 (0x1)
Torque	[0] 0.3
	[1] 0.3
	[2] 0.3
	[3] 0.3
	[4] 0.3
	[5] 0.3
	[6] 0.3
	[7] 0.3
PowerOn	[0] n
	[1] n
	[2] n
	[3] n
	[4] n
	[5] n
	[6] n
	[7] n
DigitalOut1	[0] n
	[1] n
	[2] n
	[3] n
	[4] n
	[5] n
	[6] n
	[7] n
DigitalOut2	[0] n
	[1] n
	[2] n
	[3] n
	[4] n
	[5] n
	[6] n
	[7] n
Names	[0] X axis
	[1] Y axis
	[2] Z Axis
	[3] SpareD
	[4] X axis
	[5] unused
	[6] SpareG
	[7] SpareH
Destination	[0] -6409
	[1] 0
	[2] 0
	[3] 0
	[4] 1000
	[5] 50000
	[6] 0
	[7] -5500
Letter	[0] A
	[1] B
	[2] C
	[3] D
	[4] E
	[5] F
	[6] G
	[7] H
Channels	
Devices	

8.0 – Appendix B: Code for initialization Routine

```
/*-- Initialize ----------------------------------------------------*/
// Move motors to limit switches. Use the motor coordinates at this
// position to determine the motor coordinates at the origin. If the
// negative limit switch is disabled, the current position is set as the
// origin for that axis.
void initialize(INFO *pInfo){
  int i;
  BOOL *tempStart = (BOOL *)  calloc(10,sizeof(BOOL));
  BOOL *tempNegLimitEnabled = (BOOL *) calloc(10,sizeof(BOOL));
  float *tempPos  = (float *)  calloc(10,sizeof(float));
  BOOL tempStop[10] = {1,1,1,1,1,1,1,1,1,1};
  BOOL exitFlag = 0;
  float lastCountPosArm1;
  float lastCountPosArm2;

  printf("Initializing motors:");
  // Set motors to move into their negative limit switches, if they are enabled
  //(disabled is LimPos = 9999)
  for(i=0;i<10;i++){
    if(pInfo->LimPos[i] == 9999){
      tempPos[i] = 0;
      tempNegLimitEnabled[i] = 0;
      printf("\nNegative limit switch for axis %i disabled. Axis will not be initialized.", i);
    }
    else{
      tempPos[i] = 1000*pInfo->LimPos[i]*pInfo->mScale[i];
      tempNegLimitEnabled[i] = 1;
      printf("\nNegative limit switch for axis %i enabled. Axis will be initialized.", i);
    }
  }
  channel_rw(pInfo,pInfo->hKeyMDest,(void *)tempPos,TID_FLOAT,1);

  // Cycle through each pair of motors corresponding to the same axis on each arm
  for(i=0;i<5;i++){
    tempStart[i] = tempNegLimitEnabled[i];
    tempStart[i+5] = tempNegLimitEnabled[i+5];
    channel_rw(pInfo,pInfo->hKeyMStart,(void *)tempStart,TID_BOOL,1);
    // Wait for axes with enabled limit switches to hit their limits
    while(1){
      channel_rw(pInfo,pInfo->hKeyMPos,pInfo->CountPos,TID_FLOAT,0);
      lastCountPosArm1 = pInfo->CountPos[i];
      lastCountPosArm2 = pInfo->CountPos[i+5];
      sleep(600); // Approx. polling period
      if((tempNegLimitEnabled[i] == 1) && (tempNegLimitEnabled[i+5] == 1)){
        channel_rw(pInfo,pInfo->hKeyMLimit,(void *)pInfo->AxisLimit,TID_BOOL,0);
        if(pInfo->AxisLimit[i] && pInfo->AxisLimit[i+5]) break;
        else{
          channel_rw(pInfo,pInfo->hKeyMPos,pInfo->CountPos,TID_FLOAT,0);
          if(pInfo->CountPos[i] == lastCountPosArm1){
            printf("\nAxis %i not moving!!! Stopping initialization since limit switch must be broken...", i);
            channel_rw(pInfo,pInfo->hKeyMStop,(void *)tempStop,TID_BOOL,1);
            exitFlag = 1;
            break;
          }
          if(pInfo->CountPos[i+5] == lastCountPosArm2){
            printf("\nAxis %i not moving!!! Stopping initialization since limit switch must be broken...", i+5);
            channel_rw(pInfo,pInfo->hKeyMStop,(void *)tempStop,TID_BOOL,1);
            exitFlag = 1;
            break;
          }
        }
      }
      else if((tempNegLimitEnabled[i] == 0) && (tempNegLimitEnabled[i] == 0)){
        break;
      }
      else if(tempNegLimitEnabled[i] == 0){
        channel_rw(pInfo,pInfo->hKeyMLimit,(void *)pInfo->AxisLimit,TID_BOOL,0);
        if(pInfo->AxisLimit[i+5]) break;
        else{
          channel_rw(pInfo,pInfo->hKeyMPos,pInfo->CountPos,TID_FLOAT,0);
          if(pInfo->CountPos[i+5] == lastCountPosArm2){
            printf("\nAxis %i not moving!!! Stopping initialization since limit switch must be broken...", i+5);
            channel_rw(pInfo,pInfo->hKeyMStop,(void *)tempStop,TID_BOOL,1);
            exitFlag = 1;
            break;
          }
        }
      }
      else{
        channel_rw(pInfo,pInfo->hKeyMLimit,(void *)pInfo->AxisLimit,TID_BOOL,0);
        if(pInfo->AxisLimit[i]) break;
        else{
          channel_rw(pInfo,pInfo->hKeyMPos,pInfo->CountPos,TID_FLOAT,0);
          if(pInfo->CountPos[i] == lastCountPosArm1){
            printf("\nAxis %i not moving!!! Stopping initialization since limit switch must be broken...", i);
            channel_rw(pInfo,pInfo->hKeyMStop,(void *)tempStop,TID_BOOL,1);
            exitFlag = 1;
            break;
          }
        }
      }
    }
    tempStart[i] = 0;
    tempStart[i+5] = 0;
    if(exitFlag == 1) break;
  }
```

```
  // Wait for all the motors to stop moving (sometimes this
  // occurs slightly after the limit switches are triggered)
  pInfo->Moving = 1;
  while(pInfo->Moving){
    pInfo->Moving = 0;
    channel_rw(pInfo,pInfo->hKeyMMoving,(void *)pInfo->AxisMoving,TID_BOOL,0);
    for(i=0;i<10;i++) pInfo->Moving = pInfo->Moving || pInfo->AxisMoving[i];
  }

  if(exitFlag == 0){
    // Determine the motor positions at the origin and put the results in
    // pInfo->mOrigin
    channel_rw(pInfo,pInfo->hKeyMPos,pInfo->CountPos,TID_FLOAT,0);
    for(i=0;i<10;i++){
      if(tempNegLimitEnabled[i] == 1){
        pInfo->mOrigin[i] = pInfo->CountPos[i] - pInfo->LimPos[i]*pInfo->mScale[i];
        pInfo->Position[i] = pInfo->LimPos[i];
        printf("\nFinished initializing axis %i; Origin = %5.2f counts, Position = %5.2f m.",i,pInfo->mOrigin[i], pInfo->Position[i]);
      }
      else{
        pInfo->mOrigin[i] = 0;
        pInfo->Position[i] = 0;
      }
    }
    db_set_data(pInfo->hDB,pInfo->hKeyPos,&pInfo->Position,10*sizeof(float),10,TID_FLOAT);

    // Set Initialized to 1 and exit

    pInfo->Initialized = 1;
    db_set_data(pInfo->hDB,pInfo->hKeyInit,&pInfo->Initialized,sizeof(BOOL),1,TID_BOOL);
  }

  sleep(100);
  printf("\nInitialization complete\n\n");
}
```

9.0 – Appendix C: Code for gen_scan_path Routine

```
/*-- Generate path for scan-----------------------------------------*/
BOOL gen_scan_path(void)
{
  INT j, i;
  INT point_num = 0;
  float temp_points[10000][3];

  // Scan parameters in m (read in eventually)
  int scan_type = 0;
  // For cylindrical scan
  int layer;
  int loop;
  int waypoint;
  float PI = 3.14159265;
  float height = 0.50;
  float radius = 0.32;
  float theta = 0;
  float arc_step = 0.05;
  float loop_separation = 0.16; //0.04;
  float layer_thickness = 0.05;
  // For linear/plane/rectangular prism scan
  int z_layer;
  int x_layer;
  int y_layer;
  int direction_x = -1.0;
  int direction_y = -1.0;
  float prism_height_z = 0.50;
  float prism_length_x = 0.72;
  float prism_width_y = 0.64;
  float z_step = 0.05;
  float x_step = 0.08;
  float y_step = 0.08;
  float init_pos_z = -1.0*(prism_height_z/2.0);
  float init_pos_x = -1.0*(prism_length_x/2.0);
  float init_pos_y = prism_width_y/2.0;

  if(scan_type > 1){
    printf("Error: Invalid scan type.\n");
    return 0;
  }

  // Generate path based on scan parameters:
  switch(scan_type){
    case 0:
      // Parameter checks
      if(height >= abs(gGantryLimits[2]*2.0)){
        printf("Error: Height outside of limits.\n");
        return 0;
      }
      if(radius >= abs(gGantryLimits[1])){
        printf("Error: Radius outside of limits.\n");
        return 0;
      }
      if(loop_separation <= 0.001){
        printf("Error: Loop separation too small for motors (min resolution = 1 mm).\n");
        return 0;
      }
      if(layer_thickness <= 0.001){
        printf("Error: Layer thickness too small for motors (min resolution = 1 mm).\n");
        return 0;
      }
      if(arc_step <= 0.001){
        printf("Error: Arc step size too small for motors (min resolution = 1 mm).\n");
        return 0;
      }
      printf("Generating cylindrical path...\n");
      // Generate layers
      for(layer=0;layer<=(trunc(height/layer_thickness));layer++){
        point_num = point_num + 1;
        temp_points[point_num][0] = 0;
        temp_points[point_num][1] = 0;
        temp_points[point_num][2] = (-1.0*(height/2.0)) + (layer*layer_thickness);
        // Generate loops
        for(loop=1;loop<=(trunc(radius/loop_separation));loop++){
          for(waypoint=0;waypoint<=trunc((2*PI*loop*loop_separation)/arc_step);waypoint++){
            point_num = point_num + 1;
            for(j=0;j<3;j++){
              temp_points[point_num][j] = temp_points[point_num-1][j];
            }
            theta = (2*PI)*((arc_step*waypoint)/(2*PI*loop*loop_separation));
            temp_points[point_num][0] = (loop*loop_separation)*cos(theta);
            temp_points[point_num][1] = (loop*loop_separation)*sin(theta);
          }
        }
      }
      break;
```

```
    case 1:
      // Parameter checks
      if(prism_height_z >= abs(gGantryLimits[2]*2.0)){
        printf("Error: Height outside of limits.\n");
        return 0;
      }
      if(prism_length_x >= abs(gGantryLimits[0]*2.0)){
        printf("Error: Length outside of limits.\n");
        return 0;
      }
      if(prism_width_y >= abs(gGantryLimits[1]*2.0)){
        printf("Error: Width outside of limits.\n");
        return 0;
      }
      if(init_pos_z >= abs(gGantryLimits[2]*2.0)){
        printf("Error: Initial position of z-axis outside of limits.\n");
        return 0;
      }
      if(init_pos_x >= abs(gGantryLimits[0]*2.0)){
        printf("Error: Initial position of x-axis outside of limits.\n");
        return 0;
      }
      if(init_pos_y >= abs(gGantryLimits[1]*2.0)){
        printf("Error: Initial position of y-axis outside of limits.\n");
        return 0;
      }
      if(z_step <= 0.001){
        printf("Error: Z step size too small for motors (min resolution = 1 mm).\n");
        return 0;
      }
      if(x_step <= 0.001){
        printf("Error: X step size too small for motors (min resolution = 1 mm).\n");
        return 0;
      }
      if(y_step <= 0.001){
        printf("Error: Y step size too small for motors (min resolution = 1 mm).\n");
        return 0;
      }
      printf("Generating linear/plane/rectangular prism path...\n");
      // Generate path
      for(z_layer=0;z_layer<=(trunc(prism_height_z/z_step));z_layer++){
        direction_y = -1.0*direction_y;
        if(prism_width_y == 0.0) direction_y = 1.0;
        for(y_layer=0;y_layer<=(trunc(prism_width_y/y_step));y_layer++){
          direction_x = -1.0*direction_x;
          if(prism_length_x == 0.0) direction_x = 1.0;
          for(x_layer=0;x_layer<=(trunc(prism_length_x/x_step));x_layer++){
            point_num = point_num + 1;
            temp_points[point_num][0] = direction_x*(init_pos_x + x_layer*x_step);
            temp_points[point_num][1] = direction_y*(init_pos_y - y_layer*y_step);
            temp_points[point_num][2] = init_pos_z + (z_layer*z_step);
          }
        }
      }
      break;
  }
  // Insert path into global variable/array
  gbl_total_number_points = point_num + 1;
  points = (float **) calloc(point_num + 1,sizeof(float *));
  for(j=0;j<(point_num + 1);j++){
    points[j] = (float *) calloc(3,sizeof(float));
  }
  for(j=0;j<(point_num + 1);j++){
    for(i=0;i<3;i++){
      points[j][i] = temp_points[j][i];
    }
  }
  // Check path size
  if(gbl_total_number_points > max_event_size){
    printf("Error: Path contains too many points (%i points). Max event size = %i.\n", gbl_total_number_points, max_event_size);
    return 0;
  }
  else{
    printf("Path contains %i points.\n");
    printf("Done.\n");
    return 1;
  }
}
```