Streamlining the Calculation of an Ionization Electron's Path in a GEM Detector

A thesis submitted in partial fulfillment of the requirement
for the degree of Bachelor of Science in Physics
from the College of William and Mary in Virginia

by

Joshua David Evans.

Williamsburg, Virginia
May 2006

Abstract

I have written a series of programs using the Perl and C languages to streamline the calculation of an ionization electron's path in the drift chamber of the Radial Time Projection Chamber (RTPC) used in the BoNuS experiment at the Jefferson National Accelerator Facility. Accurate ionization electron path information is crucial in the reconstruction of an energetic particle's track through detector. The first program is a Perl script that calls two existing programs, Magboltz 2 and calcEB, to perform the calculations necessary to create a drift velocity map for many points within the detector. The user specifies parameters such as the gas mixture in the drift chamber and the local electric and magnetic fields. The output is a data file containing the drift velocities at regular intervals within the chamber. The second main program reads the drift velocity map and interpolates the drift velocity for any point within the detector. Using source code from Nate Baillie, this program then uses the drift velocities to create files that specify the path ionization electrons travel in the detector. The collection of software I have created allows the user to determine the drift velocity at any point within the Radial Time Projection Chamber and calculate the path of an ionization electron in the detector with greatly increased accuracy and efficiency over previous methods.

Acknowledgements

I would first like to thank Keith Griffioen for his continued support through every stage of my project. My work was interesting and rewarding as a direct result of his patient guidance. I would also like to thank Nate Baillie for the source code from which my project became possible as well as for answers to my constant flow of questions. Finally, I would like to thank the entire BoNuS group for taking time from their extremely busy schedules to make me a welcome addition to the team.

Contents

List of Figures

List of Tables

1 Introduction

Although much information has been gathered on the structure of the proton, relatively little is known of the neutron's structure. This lack of knowledge arises from the difficulties encountered in producing a neutron target with a density high enough to be experimentally useful. Since neutrons have no charge and decay in approximately fifteen minutes when free, they are hard to contain in a suitable target. The Bound Nucleon Structure (BoNuS) collaboration proposed an alternative method to study the structure of the neutron, as described below.

1.1 The BoNuS Experiment

The goal of the BoNuS collaboration is to study the structure of the neutron. To overcome the challenges of preparing a suitable free neutron target, the BoNuS experiement scatters electrons from a deuterium target. By studying the recoil 'spectator' proton from the electron-deuterium interaction, we can correct for the nuclear effects on the neutron in the deuterium atom.

The BoNuS experiment [1] is designed to use the CEBAF Large Acceptance Spectrometer (CLAS) in Hall B [2] of the Thomas Jefferson National Accelerator Facility [3]. The CLAS detector has a lower momenta threshold of 250 Mev/c for protons. To detect 'slow' protons from the electron-deuterium interaction, the collaboration has designed a detector that will be able to track protons with momenta as low as 70 MeV/c. This detector, the Radial Time Projection Chamber (RTPC), is designed to be inserted into the CLAS detector. In this way, we can measure the results of the electron-neutron interaction with CLAS and correct for the nuclear effects by measuring the backwards traveling proton with the RTPC.

1.2 The Importance of Ionization Electron Path Information

When an energetic particle such as a proton travels through a particle detector's drift chamber, it ionizes some of the gas molecules along its path. An applied voltage across the drift chamber, also known as the sensitive volume, creates an electric field that accelerates the ionization electrons towards the detector's readout pads. A magnetic field is also present in the sensitive volume, which creates a Lorentz force orthogonal to the magnetic field and the ionization electron's velocity.

By calculating the drift velocity of ionization electrons within the sensitive volume we can find the path these electrons traveled to the readout pad on which they are detected. Using time coincidence techniques, we can project backwards in time to find the location within the detector where the ionization electron was liberated. From here, we can reconstruct the path of the energetic particle that passed through the detector's drift chamber and thus identify the particle's momentum by the curvature of its path. The reconstruction of the energetic particle's path is extremely dependent on accurate calculation of the ionization electron's drift velocity.

1.3 Existing Calculation Methods

The preexisting method used by the BoNuS collaboration to calculate the path of an ionization electron's path through the detector required many steps and was highly inefficient. A user would first need to run the program calcEB to obtain the field information for a specific point within the detector. The program uses a field map of the detector to interpolate the electric and magnetic fields for the input coordinates.

The next step is to calculate the drift velocity at that point within the detector. Magboltz is a program that uses Monte Carlo simulation to determine the drift velocity of an ionization electron within a gas detector chamber [5],[6]. The user must modify Magboltz's input file fpr the specific gas mixture in the drift chamber, the pressure and temperature in the detector, and also the field information at the coordinate point in question. Magboltz takes nearly four minutes to run, on a home PC running Ubuntu Linux and returns approximately 75 lines of output. The user must then scroll through and find the three lines of output that contain the drift velocity values in the s, ϕ, and z directions and their associated statistical errors.

From here the user would then have to project the location of the ionization electron a certain time later. This next coordinate point would then need the above process repeated again to find the drift velocity at that point so the user could then project the electron to the next point along its path.

Needless to say, this method was extremely tedious and inefficient in calculating an ionization electron's path for the many different voltages and gas mixtures used in the BoNuS experiment. In response to this inefficiency in calculation, my project was born. The software developed to efficiently calculate the many necessary path files is discussed in depth in the following sections. But first, we must discuss some symmetries and assumptions that we used in calculating drift velocities for the RTPC.

2 The Radial Time Projection Chamber

The Radial Time Projection Chamber (RTPC) is the detector designed by the BoNuS collaboration to detect the low momenta spectator protons from the electron-deuterium interaction. It is a cylinder 20 cm in length with a radius of approximately 7 cm. The cylinder is split into two identical halves. The drift region lies between 30 mm and 60 mm in the radial direction. The drift region is filled with Helium-dimethyl ether(HeDME) at one atmosphere. Together, the two halves form a sensitive volume 200 mm in length covering over 148^o in ϕ.

Three Gas Electron Multipliers (GEMs) surround the drift chamber at radii (s) of 60 mm, 63 mm, and 66 mm. The GEMs were developed by the Gas Detectors Development Group at CERN [4]. Each GEM has a gain of approximately 100, allowing the initial ionization electrons to create a cascade of electrons that can be measured by the readout pads. The main difference between a conventional time projection chamber (TPC) and the radial TPC is that the radial TPC has its readout pads on the surface of the cylinder, not on the ends of the cylinder. The radial TPC has a shorter drift path than a conventional TPC allowing events to be captured at a higher rate.

The electronic readout pads are located at 69 mm from the center of the RTPC. Each pad is 4.445 mm by 5 mm. In total, there are 3200 readout pads on the detector. There is a preamplifier for every 16 pads connected on the outside of the RTPC. The target is a thin-walled Kapton tube 12 mm in diameter filled with 7 atm of deuterium. Figures 1 and 2 show these dimensions and main features of the RTPC.

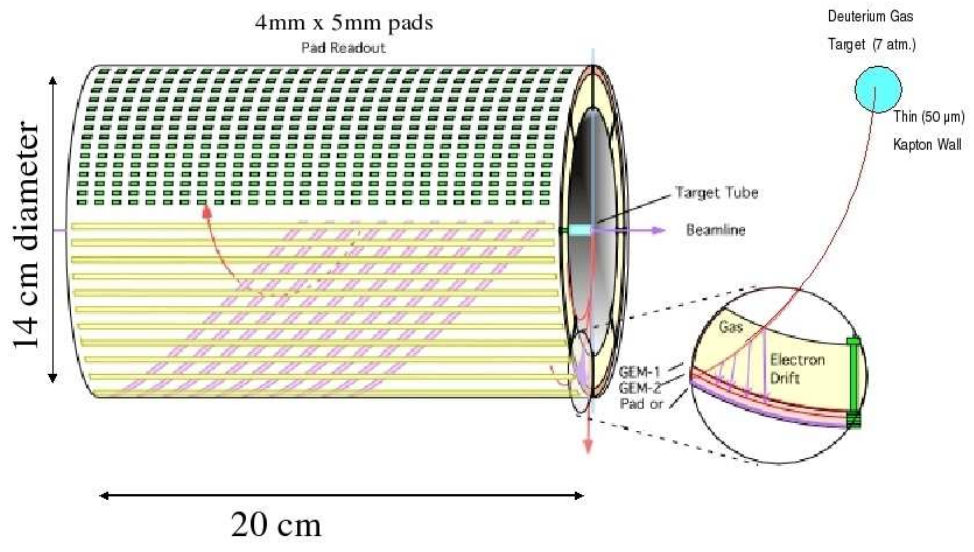


Figure 1: Dimensions and main features of the Radial Time Projection Chamber. The RTPC is 20 cm long and 14 cm in diameter. The target is a thin-walled Kapton tube 12 mm in diameter filled with 7 atm of deuterium. The target tube and sensitive volume (30 mm $< s <$ 69mm) are shown in cross section with a schematic of an ionizing proton track and the corresponding ionization electrons.

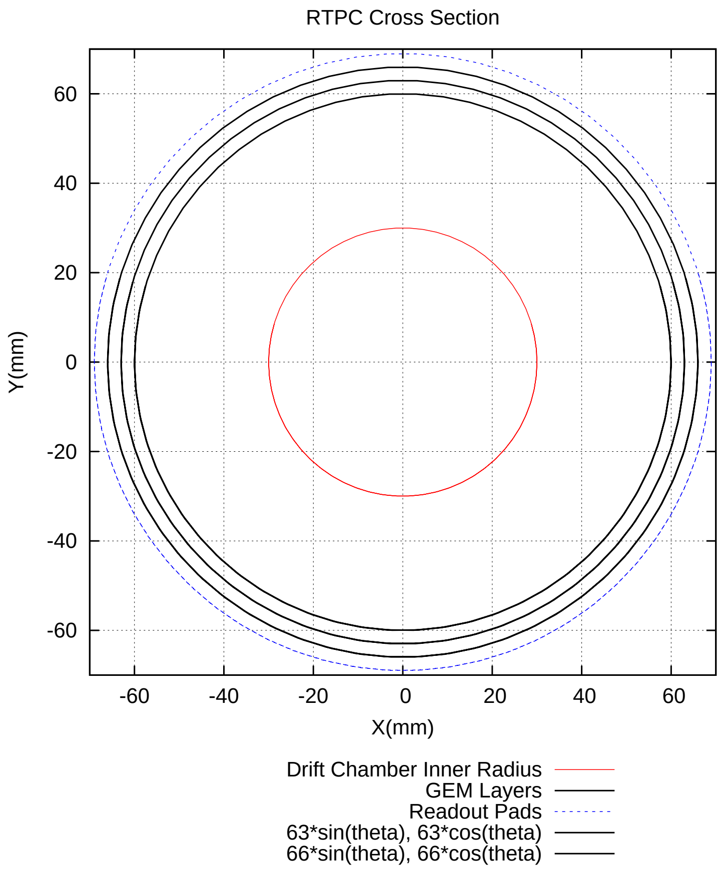


Figure 2: Cross Section of the Radial Time Projection Chamber. The inner radius of the sensitive volume is at s=30 mm, GEM 1 is at s=60 mm, GEM 2 is at s=63 mm, GEM 3 is at s=66 mm, and the readout pads are at s=69 mm.

2.1 Assumptions and Symmetries

The symmetry of the Radial Time Projection Chamber allows us to make a few assumptions when dealing with the drift velocities of the ionization electrons. But first we need to discuss the direction of the electric and magnetic fields present in the drift chamber. Please note we are using a cylindrical coordinate system with z=0 at the center of the detector, not at one end. The electric field points radially inward causing the electrons to accelerate towards the readout pads on the outer radius. This electric field is generated by a potential difference between the inner radius of the drift chamber (s=30 mm) and the first GEM (s=60 mm). There is also a potential difference between each of the GEMs, and the readout pads and the third GEM. Thus we define four separate voltages and their corresponding electric fields:

- Drift Voltage = V_d: 30 mm < s < 60 mm
- Transfer Voltage 1 = V_{t1}: 60 mm < s < 63 mm
- Transfer Voltage 2 = V_{t2}: 63 mm < s < 66 mm
- Induction Voltage = V_i: 66 mm < s < 69 mm

The naming convention for the corresponding electric fields uses the same subscripts. The magnetic field that permeates the drift region of the RTPC is generated by the CLAS DVCS. This magnetic field's direction is coaxial with the beam of electrons that passes through the target. Thus, the magnetic field points in the positive z direction. Since the electrons drift towards the outer radius (the s direction) and the magnetic field points in the z direction, they will experience a Lorentz force in the ϕ direction. Thus, we expect the electrons to have a velocity in the s direction from the electric field and velocity in the ϕ direction as a result of the Lorentz force.

However, the magnetic field does have some radial and angular components near the ends of the detector (z=-100 mm and z=100 mm). These fringe fields create a nonzero Lorentz force in the z direction. However, the force in the z direction is not large enough to shift the electrons trajectory to a neighboring readout pad. Therefore, we consider the z drift velocity to be negligible and assume that the electron will strike a readout pad at the same z value that it was ionized. Refer to Appendix A for the calculation to support this assumption.

The electric and magnetic fields are symmetric around the center of the detector. This allows us to calculate the drift velocity and electron path information for one half of the detector (0 mm < z < 100 mm). The RTPC is symmetric in ϕ given the same drift, transfer, and induction voltages. This means we only need to calculate the drift velocity at ϕ=0, since the drift velocity will be the same for all ϕ values.

It should be noted that the physical halves of the detector are not symmetric. Due to slight differences in electronics, the same voltage set point will yield different drift, transfer and induction voltages in the two halves. However, the programs that calculate the drift velocity and then the ionization electron's path require the specific voltages in the drift chamber and thus do not discriminate which half they are calculating for. Thus, we need only to know the drift, transfer and induction voltages to calculate the drift velocity at any point within the detector.

3 Streamlining the Calculation of Path Information

The goal of my senior thesis was to create an accurate and efficient method of calculating an ionization electron's path through the Radial Time Projection Chamber. The programs created simplified the user input process, automated the data collection, and allowed for any number of data points to be specified. The end result is a universally useful suite of programs that calculate the ionization electron path information.

3.1 Program Overview

The programs developed to calculate the path of an ionization electron through the RTPC are contained within two main programs: drift.pl and electronPath. Drift.pl is a perl script written to streamline the calculation of drift velocity. It simplifies user input and calls calcEB and Magboltz for many points within the detector to create a map of drift velocities. ElectronPath is a program that uses the drift velocity maps created by drift.pl to calculate the path an ionization electron travels from the inner radius to the readout pads. Its output is a set of files specifying the coordinates of the electron's path for a regular time interval. Figure 3 shows the simplified user process to calculate the ionization electron path information.

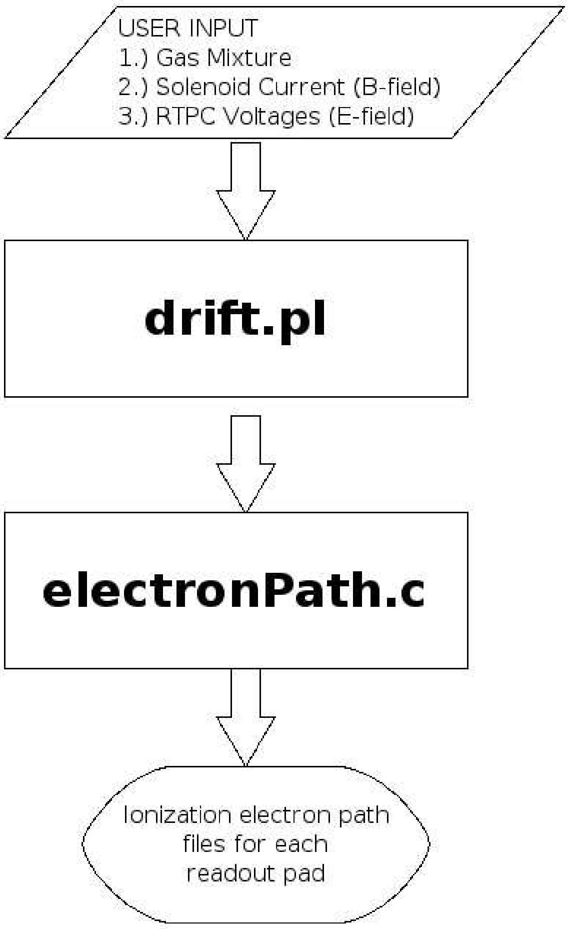

USER INPUT
1.) Gas Mixture
2.) Solenoid Current (B-field)
3.) RTPC Voltages (E-field)
drift.pl
electronPath.c
Ionization electron path files for each readout pad

Figure 3: Program overview flow chart. Drift.pl prompts the user to input the gas mixture, the solenoid current, and the RTPC voltages. The user then runs electronPath and must input the same information. This is necessary for electronPath to find the correct drift velocity map created by drift.pl. ElectronPath then outputs a set of path files that describe the trajectory an ionization electron would travel through the detector.

3.2 Magboltz

Magboltz 2 is a Monte Carlo simulation program written by Dr. Biagi at the University of Liverpool [5]. This program integrates the Boltzmann transport function to calculate the drift velocity of an ionization electron in a gas. Magboltz 2 is a revised version of the original Magboltz code, which helps minimize the amount of user input necessary to run the program. The user must modify an input file, fort.20, to specify the gas mixture, temperature, pressure, the electric and magnetic field magnitudes, and the angle between the E and B fields. Magboltz 2 then outputs approximately

75 lines of data, including the drift velocities in the s, phi, and z directions and their associated statistical errors. Using Magboltz, we can find the drift velocity at any point within the RTPC if we know the gas and field properties.

3.3 calcEB

Nate Baillie wrote the calcEB program to calculate the electric and magnetic field information for any point within the RTPC. Using a field map of the detector, calcEB interpolates the electric and magnetic field magnitudes and the Lorentz angle at any coordinate point that lies within the sensitive volume of the RTPC. The user must input the coordinate point to be calculated, the current in the solenoid, and the four voltages within the detector. The program then uses the exisiting field map to scale the field information to the particular solenoid current and voltage settings. Thus, calcEB gives us the field information necessary to run Magboltz.

3.4 drift.pl

The drift.pl code was created in response to the need for knowing the drift velocity of an ionization electron at many different positions within the detector. The program streamlines the drift velocity calculation by automating the Magboltz and calcEB programs. Most importantly, drift.pl repeats the process for many data points. This creates a map of drift velocities throughout the detector. Upon executing the program, the user specifies the gas mixture; the drift, transfer, and induction voltages; and the solenoid current. In addition, drift.pl lists the possible input files, and allows the user to choose the appropriate coordinate map. This input file specifies for which coordinate points the drift velocity will be calculated. Refer to Appendix B for an example of the coordinate input file. Once the user specifies the drift chamber parameters, drift.pl can run in the background. The core of the program is a while loop that calculates the drift velocity for each coordinate point in the input file.

Drift.pl is written in the Perl scripting language because of its ability to internally execute shell commands. Upon reading in a set ot coordinate points, drift.pl executes the external program calcEB. It saves the output to a local variable, and then searches for the electric field, the magnetic field, and the Lorentz angle. Using this field information, drift.pl then modifies fort.20, Magboltz's input file, and then runs the Magboltz program. The output from Magboltz is saved into a temporary file.

The drift velocity is reported from Magboltz in scientific notation with units of $\frac{\mu m}{ns}$. It should be noted that this is much slower than the speed of light, which is about one foot per nanosecond. Thus we need not consider relativistic effects. Once drift.pl has retrieved the drift velocities and their associated statistical errors from the temporary file, these are saved as string variables as a result of the scientific notation. Drift.pl uses a subroutine to convert the drift velocities from a string variable in scientific notation to a decimal number stored as a floating point variable. The program then opens the output file and appends the three components of drift velocity and their associated errors. Upon completion, this output file forms the drift velocity map used by the electronPath program. Refer to Appendix C for an example of a drift velocity map and Appendix E for the drift.pl code.

The drift velocity map is stored in a directory with the other drift velocity maps. The maps are named according to their initial parameters as follows:

$$[gas\,code]\_[gas\,mixture]\_b[solenoid\,current]A\_V[drift\,voltage].drift$$

The regularized naming convention is used by the electronPath program to search for the correct drift map. Figure 4 illustrates the role drift.pl plays in automating the calcEB and Magboltz programs and iterating the process for many data points.

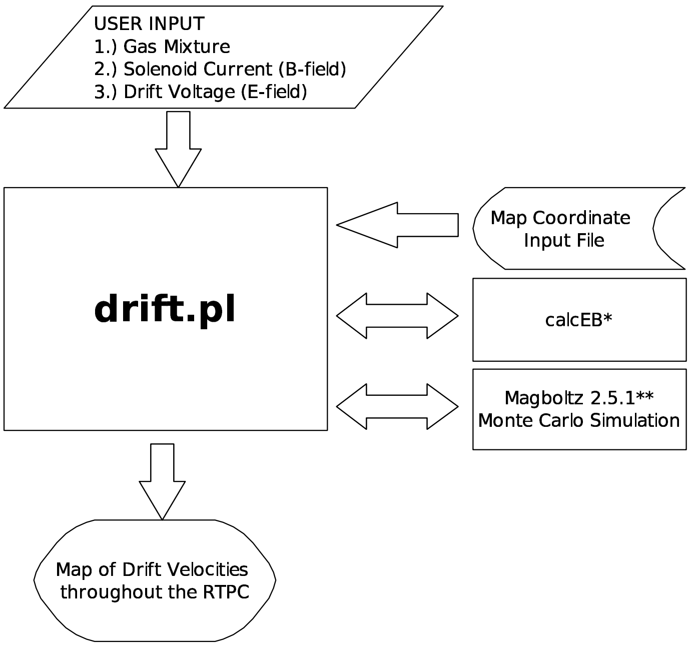

USER INPUT
1.) Gas Mixture
2.) Solenoid Current (B-field)
3.) Drift Voltage (E-field)
drift.pl
Map Coordinate Input File
calcEB*
Magboltz 2.5.1** Monte Carlo Simulation
Map of Drift Velocities throughout the RTPC

Figure 4: Drift.pl flow chart. The user is first prompted for the gas mixture, solenoid current, and the RTPC voltages. The map coordinate input file gives drift.pl the coordinate points for which the drift velocity information will be calculated. The program then runs calcEB and Magboltz 2.5.1 for each coordinate point and captures the output. The drift velocities are then stored in a file named for the input parameters. This output file is the drift velocity map.

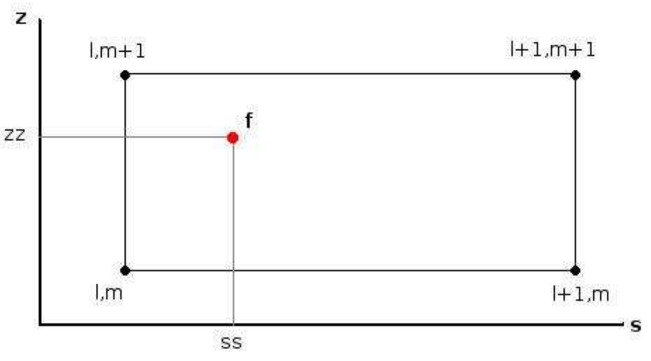
3.5 electronPath

Up to this point all the software we have discussed has given us the ability to calculate the drift velocity information of the ionization electrons moving through the RTPC. ElectronPath uses these drift velocity maps to reconstruct the path the ionization electrons would travel to reach a particular readout pad. Since the electrons do not have a significant drift velocity in the z direction, we assume an ionization electron that strikes a certain readout pad was liberated in the same z slice.

Since the drift velocity is symmetric in ϕ, we need only to calculate the path information for each value of z that a pad is centered on. The readout pads do not have coordinates symmetric around z=0 mm: i.e. a pad centered at z=−42 mm does not necessarily mean that there is one centered at z=+42 mm. Thus, we need to calculate path information for every different z coordinate that a readout pad is located on. The Radial Time Projection Chamber has 3200 individual pads, but only 120 different z coordinates. It is clear that the symmetry of the drift velocity in the ϕ direction saves us a great deal of calculation.

ElectronPath requires the user to input the gas code and mixture, the voltage set points, the solenoid current, and the time step (Δt). The gas and field information is used to look up the appropriate drift velocity map, which is then read into an array. The main loop of the program consists of two primary subroutines. The first uses the drift velocity at s=30 mm (the inner radius of the drift chamber) and Δt to project the next position of the ionization electron. In this way, electronPath calculates the ionization electron's path step by step. Using a small enough time step, we can get an excellent approximation of the ionization electron's path through the RTPC.

The second primary subroutine interpolates the drift velocity if the coordinates needed for the projected point are not found in the drift velocity map. The equations used by the interpolation routine are displayed in Figure 5. Since most pad centers do not lie on the regular z values used to create the drift velocity maps, this subroutine is used almost every time electronPath needs to project the electron's path to the next set of coordinates.



$$t=\frac{(ss-s_l)}{(s_{l+1}-s_l)}$$

$$u=\frac{(zz-z_m)}{(z_{m+1}-z_m)}$$

$$f=(1-t)(1-u)f_{l,m}+t(1-u)f_{l+1,m}+u(1-t)f_{l,m+1}+utf_{l+1,m+1}$$

Figure 5: Interpolation Equation. This interpolation equation solves for the value at coordinate point f if the value at four surrounding points are known. ElectronPath uses this equation to solve for the drift velocity at any point within the detector using the drift velocity map at the indexed points.

By looping this process for each z of a readout pad's center, electronPath calculates the path information for every readout pad in the Radial Time Projection Chamber. Refer to Appendix F for the electronPath code. The path files are stored in a folder named for the parameters with the same convention as the drift velocity map. In this way, path files for different parameters are stored in separate folders. Each individual path file is named for the z coordinate it was calculated for. Path files are stored using the naming convention: $out[z\,coordinate].path$

A short Perl script was written to run the electronPath program. This is advantageous since Perl can execute shell commands more easily than the C language. The Perl program, runEpath.pl, prompts the user to input the standard parameter set: gas mixture, solenoid current, drift voltage, as well as the time step (Δt). RunEpath.pl uses the input values to create the output folder as described in the previous paragraph in the PATH_FILES directory. The program then runs electronPath with the given inputs.

3.6 Verification of the Interpolation Routine

The interpolation subroutine plays a crucial role in the calculation of an ionization electron's path. In fact, the high level of efficiency boasted by this program suite rests in large part on the ability of the interpolation routine to accurately calculate the drift velocity between known points. If the interpolation calculation were not used, electronPath would need to call Magboltz for every single point along the electron's path. The interpolation routine allows the program to use the same drift velocity map for varying time steps. Thus, the accuracy of the ionization path information rests on the accuracy of the interpolation routine.

Magboltz, being a Monte Carlo simulation, has an associated error with each drift velocity calculation it performs. We have set up Magboltz to have a statistical error of approximately 1%, which consequently is why Magboltz takes a few minutes to run a single data point. Hence, we would like to verify that the interpolation routine has an error less than or equal to that of statistical error of Magboltz.

To verify the accuracy of the calculation, I have chosen ten coordinate points within the Radial Time Projection Chamber that are not found in the standard drift velocity map we use. These points span the RTPC from the inner radius to the outer radius, and from the center of the detector to the outer edge of the detector. The coordinates were also chosen such that some were far from all data points in the map and others were very close to one set of coordinate points. This allows us to test the interpolation calculation for the wide range of circumstances it will encounter when run in the electronPath program.

These ten coordinate points were then run manually using Magboltz to gather the drift velocity and the associated error. Since the drift velocity in the z direction is negligible, we need only worry about the s and ϕ components. The interpolation routine was then run for the ten points using the drift velocity map with the same parameters as the Magboltz drift velocities and the standard map increment.

As expected, the interpolated drift velocities fall within the statisical error of Magboltz. Figures 6 and 7 show the ten data points, plotted on the horizontal axis. The vertical axis represents the error in microns per nanosecond. For the Magboltz drift velocities, the error amount is the statistical error. Both the upper and lower limit errors are shown. The red line connects the statistical errors from Magboltz for the ten data points. The interpolated error is simply the difference between the interpolated drift velocity and the drift velocity that Magboltz calculated. Interpolation errors are represented by the blue dots. Notice that the interpolation error falls within the Magboltz statistical error range for almost all of the test data points.

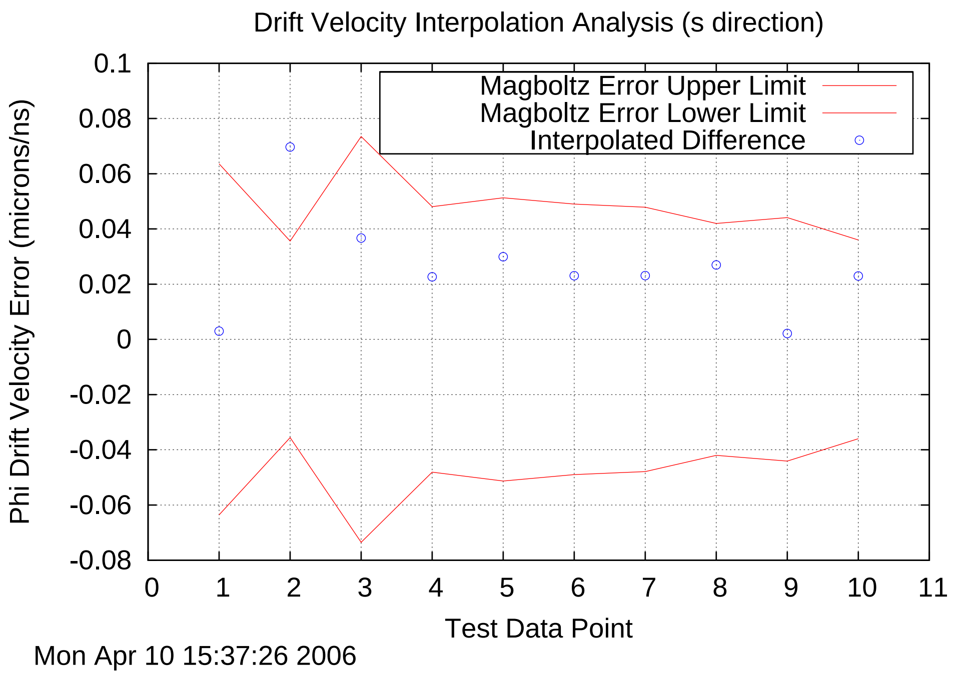


Figure 6: Interpolation Verification: s direction. The mean Magboltz error was $0.0491\frac{\mu m}{ns}$ while the mean interpolated error was $0.0260\frac{\mu m}{ns}$. Since most interpolated errors lie within or very near the Magboltz error range, we can use the interpolation calculation with the current drift velocity map size without contributing significantly to the error.

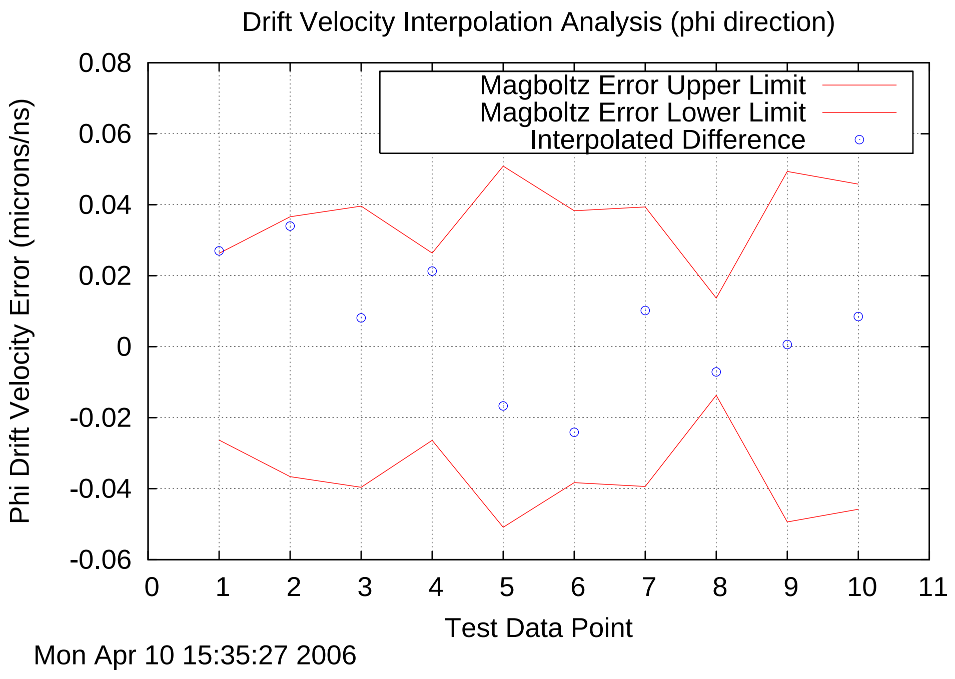


Figure 7: Interpolation Verification: ϕ direction. The mean Magboltz error was $0.0366\frac{\mu m}{ns}$ while the mean interpolated error was $0.0062\frac{\mu m}{ns}$. Since most interpolated errors lie within or very near the Magboltz error range, we can use the interpolation calculation with the current drift velocity map size without contributing significantly to the error.

The mean interpolation error was $0.0260\frac{\mu m}{ns}$ in the s direction and $0.0062\frac{\mu m}{ns}$ in the ϕ direction. The interpolated errors had a standard deviation of $0.0188\frac{\mu m}{ns}$ and $0.0186\frac{\mu m}{ns}$ in the s and ϕ directions, respectively. Magboltz gave a mean error amount of $0.0491\frac{\mu m}{ns}$ in s and $0.0366\frac{\mu m}{ns}$ in ϕ with standards deviations of $0.0118\frac{\mu m}{ns}$ and $0.0116\frac{\mu m}{ns}$, respectively. Refer to Appendix D for the interpolation verification data. Since the interpolation calculation has an error much less than the statistical error inherent in the Magboltz calculation of drift velocity, we can use the interpolation routine to greatly increase the efficiency of electronPath without sacrificing accuracy.

3.7 Ionized Electron Path Files

The end result of this suite of programs is a set of files containing the coordinate points an ionization electron would travel through the RTPC. The following graphs were made with gnuplot to show the path an ionization electron would travel through the Radial Time Projection Chamber. The path files were created for a Helium-DME 80/20 mixture with a 114 ns timestep. The solenoid current was 534 A and the drift voltage was set to 1685 V. Each red point shows the electron's position projected 114 ns from the previous coordinate postion. In this way, we can visualize the electron's path with these plots.

These ionization electron paths show the characteristics we expected. Since the electric field pointed radially inward, we expected the electrons to drift towards the outer radius. The velocity of the ionization electrons in the s direction and the magnetic field in the z direction give rise to a Lorentz force in the ϕ direction. This creates the curvature of the ionization electron paths.

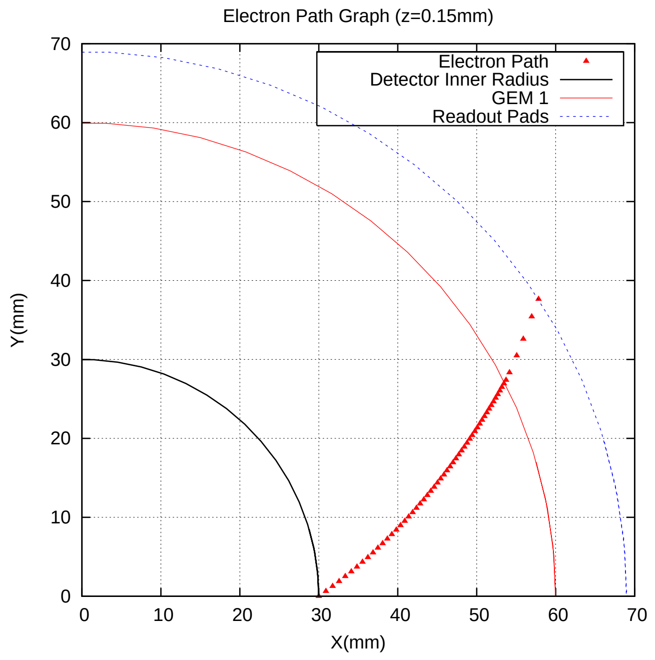


Figure 8: Electron path plot for z=0.15 mm. Note that this path has more curvature than the path at z=98.15 mm because the magnetic field in the z direction is stronger creating a larger Lorentz force.

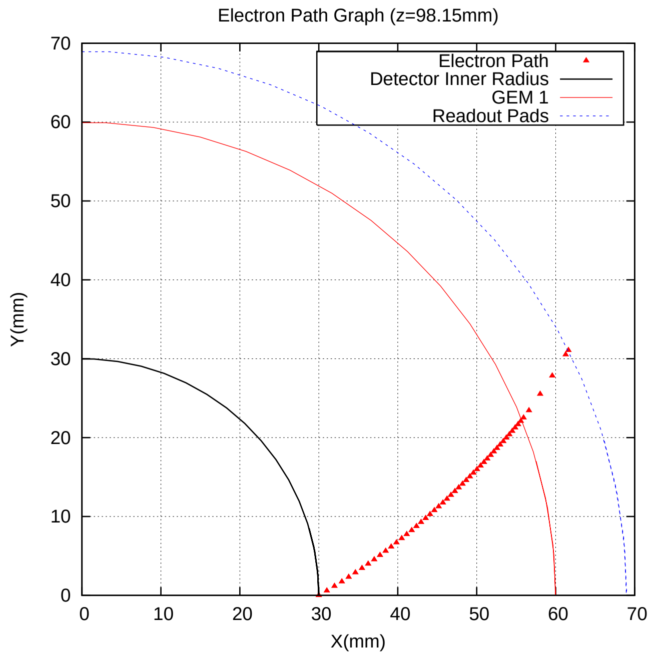


Figure 9: Electron path plot for z=98.15 mm. Note that this path has less curvature than the path at z=0.15 mm since the magnetic field is weaker in the z direction as a result of the fringe effects near the outer edge of the RTPC.

The points get closer together farther from the center of the detector as a result of the cylindrical geometry of the RTPC. Unlike a parallel plate capacitor which has a near constant electric field, the circular cross section of the RTPC causes the electric field to weaken as the radius increases. We also see that the electron travelling near the center of the RTPC (z=0.15 mm) has more curvature than the path at the edge of the RTPC (z=98.15 mm). This is a result of the fringe fields near the end of the detector lessens the magnitude of the magnetic field in the z direction and thus the electrons experience a smaller Lorentz force near the edge of the RTPC. It is also of note that the electron points get very far apart in the GEM region of the detector (60 mm $< s <$ 69 mm). This is a result of the acceleration caused by the GEM foils.

4 Conclusions

The path files created by this program suite tell us the trajectory an ionization electron travels through the RTPC. Each path file also stores how long it takes the electron to travel from one point to the next. This information is used in coincident analysis to project backwards in time where the ionization electron was liberated. Knowing where the electrons detected by the readout pads on the outer radius of the RTPC allows us to reconstruct the path of the ionizing particle traveling through the detector.

By knowing the path a proton from the electron-deuterium interaction traveled through the Radial Time Projection Chamber, we can deduce its momentum and correct for the nuclear effects of the proton and the neutron. Once these nuclear effects are accounted for, we will be able to accurately study the structure of the neutron by analyzing the result of the electron-neutron interaction captured by the CEBAF Large Acceptance Spectrometer.

Without accurate drift velocity information, we cannot deduce the track of the proton through the RTPC detector. Magboltz, by using a Monte Carlo simulation, gives us the ability to calculate the drift velocity of the ionization electrons at any point in the RTPC. The program suite developed in this senior thesis adds efficiency to the path calculation while retaining a high level of accuracy. A user can now use drift.pl and electronPath to efficiently and accurately calculate the drift velocity and path information for any parameters the RTPC can be set for.

A Calculation Supporting Assumption that Drift Velocity in z is Negligible

To support the assumption that the z drift velocity is negligible, I calculated the distance an electron would travel in the z direction using some of the largest drift velocities in the BoNuS experiment. The drift velocity map was set for HeDME with a voltage of 1685 V and a solenoid current of 534 A. These parameters create some of the largest field values in the RTPC used in the BoNuS experiment. I first wrote a program to gather all of the drift velocities at z=100 mm. These are the largest z drift velocities in the detector due to the fringe fields. The following are the gathered z drift velocities.

```
s    z    z_drift  z_error
30  100   0.2251   45.26
32  100   0.1698   42.77
34  100   0.1875   45.50
36  100   0.1797   46.95
38  100   0.1734   42.77
40  100   0.1605   46.79
42  100   0.1747   52.83
44  100   0.1615   61.78
46  100   0.1627   56.33
48  100   0.1399   56.07
50  100   0.1975   37.12
52  100   0.1764   55.60
54  100   0.1699   54.56
56  100   0.1676   44.44
58  100   0.1666   45.87
60  100   0.1378   64.16
61  100   1.0700   16.19
62  100   1.1120   15.86
63  100   1.0890   14.06
64  100   1.3120   12.97
65  100   1.4720   10.81
66  100   1.3730   13.93
67  100   1.3550   20.42
68  100   1.3400   19.47
69  100   1.3890   16.23
```

The next step was to calculate the distance an ionization electron would drift in the z direction. I found that the average z drift velocity was 0.570 $\frac{\mu m}{ns}$. On average, the ionization electrons take 6200 ns to travel through the detector's sensitive region.

Now we can find that the ionization electrons travel 3.534 mm in the z direction.

$$DriftDistance = 0.570\frac{\mu m}{ns} * 6200ns * \frac{1mm}{1000\mu m} = 3.534mm \tag{1}$$

A single readout pad is 5 mm in the z direction, thus the ionization electrons travel less than the length of a single readout pad even for some of the highest z drift velocities in the experiments. Using this, we can assume that the drift velocity in the z direction is negligible for average detector parameters.

B Coordinate Input File

The following is an abbreviated example of the coordinate input file used by drift.pl. This was the input file used to create the drift velocity map for the interpolation verification. Note the spacing is every 5 mm in s and every 25 mm in z. The interpolation verification showed that this spacing does not add unwanted error.

```
s phi z
30 0 0
30 0 25
30 0 50
30 0 75
30 0 100
35 0 0
35 0 25
35 0 50
35 0 75
35 0 100
.
.
.
70 0 100
```

C Drift Velocity Map

This is an example of a drift velocity map for a gas mixture of Helium-DME (gas code 2) at 80% and 20% respectively. The solenoid current was 534 A and the drift voltage was set to 1685 V. The units of the drift velocities (ws, wphi, and wz) are $\frac{\mu m}{ns}$. Note how small the z drift velocity is in comparison to the ϕ and s drift velocities.
$2\_8020\_b534A\_V1685.drift$

```
s   z    ws       wphi     wz        s_err    phi_err z_err    sol    Vdrift
30  0    7.8530   5.5740   0.0000    1.24     1.37    0.00     534    1685
30  25   7.9350   5.5490   0.0736    0.75     0.84    131.10   534    1685
30  50   8.1340   5.4700   0.0884    0.79     1.00    115.74   534    1685
30  75   8.4990   5.3370   0.1486    0.99     1.34    63.88    534    1685
.
.
.
```

D Interpolation Verification Table

The following tables contain the information used for the verification of the interpolation calculation. These abbreviations were used in column titles:

- DP: Data Point
- Mag: Magboltz
- Interp: Interpolation Calculation
- DV: Drift Velocity ($\frac{\mu m}{ns}$)
- Err: Statistical Error (%)
- eAmt: Error Amount ($\frac{\mu m}{ns}$)

Table 1: Interpolation Test Coordinates

DP	s	ϕ	z
1	30	0	20
2	32	0	65
3	35	0	43
4	38	0	3
5	41	0	89
6	45	0	59
7	47	0	12
8	53	0	77
9	56	0	97
10	60	0	34

Table 2: Interpolation Verification Data: *s* direction

DP	Mag DV	Mag Err	Mag eAmt	Interp DV	Interp eAmt
1	6.9930	0.91%	0.0636	6.9960	0.0030
2	6.8380	0.52%	0.0356	6.9077	0.0697
3	6.0240	1.22%	0.0735	6.0607	0.0367
4	5.4070	0.89%	0.0481	5.4297	0.0227
5	5.4580	0.94%	0.0513	5.4879	0.0299
6	4.7090	1.04%	0.0490	4.7320	0.0230
7	4.3150	1.11%	0.0479	4.3381	0.0231
8	4.0780	1.03%	0.0420	4.1050	0.0270
9	4.0060	1.10%	0.0441	4.0081	0.0021
10	3.3690	1.07%	0.0360	3.3920	0.0230

Table 3: Interpolation Verification Data: ϕ direction

DP	Mag DV	Mag Err	Mag eAmt	Interp DV	Interp eAmt
1	4.7740	0.55%	0.0263	4.8010	0.0270
2	4.3030	0.85%	0.0366	4.3370	0.0340
3	3.9230	1.01%	0.0396	3.9311	0.0081
4	3.6180	0.73%	0.0264	3.6393	0.0213
5	3.1010	1.64%	0.0509	3.0842	-0.0167
6	2.9020	1.32%	0.0383	2.8779	-0.0241
7	2.8330	1.39%	0.0394	2.8432	0.0102
8	2.3640	0.58%	0.0137	2.3569	-0.0071
9	2.1280	2.32%	0.0494	2.1286	0.0006
10	2.1420	2.14%	0.0458	2.1505	0.0085

E Drift.pl Code

```
#!/usr/bin/perl -w

#INPUTS: (s, phi, z) specified in the user-defined input file: File must be stored
in the /INPUT folder
#OUTPUTS: (s, z, s_drift & error, phi_drift & error, z-drift & error)
output file specified by user will be stored in the /OUTPUT folder

#MAIN ROUTINE-----------------------------------------------------------------------

user_input (); #run the user_input subroutine.

$totalcount = counter (); #run the counter subroutine to find howmany data points
will be processed.
$loopcount = 0; #initialize the loopcount variable to be used in the
percent_complete subroutine/

open(INFILE, "<$in") || die "ERROR: Can't open $in: $!\n";

while(<INFILE>) { #read a line of input into $_ (the inherent variable)
    chomp; #remove trailing whitespace
    $EBout = `./calcEB $_ $sol $drift $t1 $t2 $ind`; #run ./calcEB for line of input
and capture output
    @input = split; #split the line of input (s, phi, z) and store the values in an
array
    $s = $input[0];
    $z = $input[2];
    $EBin = $_;  #store inherent variable (line of input data)
    $_ = $EBout; #reassign inherent variable to the output of calcEB
    @output = split; #split inherent variable (calcEB output)
    $Es = $output[13]; #assign Electric field magnitude
    $Btot = $output[17]; #assign Magnetic field magnitude
    $Theta = $output[25]; #assign angle between E and B fields

    mod_fort_20 (); #run the mod_fort_20 subroutine.
    magboltz (); #run the magboltz subroutine.
    data_mine (); #run the data_mine subroutine.
    write_output (); #run the write_output subroutine.
    $loopcount += 1; #iterate the loopcount variable
    percent_complete (); #run the percent complete subroutine.
 }#close main input while-loop

close(INFILE) || die "ERROR: Can't close $in: $!\n";
#-----------------------------------------------------------------------------
#SUB ROUTINES----------------------------------------------------------------
sub counter { #Returns the number of data points to be processed
    $count = 0;
    open(INCOUNT, $in) || die "Can't open $in\n";
    while(<INCOUNT>) {
       $count += 1;
    }
    close(INCOUNT);
    print "Data points to be processed: $count\n\n";
    return $count;
}
#-----------------------------------------------------------------------------
sub percent_complete { #Calculates the percent of the input data that has been
processed
    $percent = ($loopcount / $totalcount)*100;
    print "PERCENT COMPLETE: $percent%\n";
}
#-----------------------------------------------------------------------------
sub user_input { #Promtps the user for input and assigns input and output files

    system("clear");
    print "DRIFT VELOCITY PROGRAM\n";
    print "Created June 2005\n";
    print "Contact Josh Evans at William & Mary with any questions.\n\n";
    print "What DVCS Solenoid Current should we simulate?: ";
    $sol = <STDIN>; #read in the solenoid current
    chomp($sol); #remove trailing whitespace
    print "What Drift Voltage should we simulate?: ";
    $drift = <STDIN>; #read in the drift voltage
    chomp($drift);
    print "What Transfer 1 Voltage should we simulate?: ";
    $t1 = <STDIN>; #read in the transfer1 voltage
    chomp($t1);
    print "What Transfer 2 Voltage should we simulate?: ";
    $t2 = <STDIN>; #read in the transfer2 voltage
    chomp($t2);
    print "What Induction Voltage should we simulate?: ";
    $ind = <STDIN>; #read in the induction voltage
    chomp($ind);

    $in_dir = list_files ("/INPUT"); #run list_files subroutine for the /INPUT
directory

    print "\nEnter the name of the input file: "; #prompt user for input file name
    $in_file = <STDIN>; #capture the name of the input file
    chomp ($in_file); #remove the trailing whitespace
    $in = "$in_dir"."/"."$in_file"; #create the entire path for the input file

    if(-e $in) { #check for existence of input file
$out_dir = list_files ("/DRIFT_MAPS"); #assigns the path directory to the
out_dir variable
system("clear"); #clear screen since user does not specify output file name
anymore

print "\n\n   Argon(2) -- CO2(12) -- He4(3) -- DME(25)\n"; #display gas
codes
print "Enter the two gas codes\nGas1: "; #prompt user for gas codes
$gas1 = <STDIN>; # read in first gas code
chomp($gas1); #remove trailing whitespace
print "Gas2: ";
$gas2 = <STDIN>; #read in second gas code
chomp($gas2); #remove trailing whitespace
print "\n";
print "Enter the mixture percentages (0 to 100)\n"; #prompt user for gas
mixture
print "Gas1: ";
$mix1 = <STDIN>; #read in mixture percentage 1
chomp($mix1);
print "Gas2: ";
$mix2 = <STDIN>; #read in mixture percentage 2
chomp($mix2);
print "\n$gas1 - $mix1%\n"; #print gas codes and percentages for
verification purposes
print "$gas2 - $mix2%\n";

#assign output file name
$out_file = "2_"."$mix1"."$mix2"."_b"."$sol"."A_V"."$drift".".drift";
$out = "$out_dir"."/"."$out_file"; #assign output file path name
print "Final data will be stored in $out\n\n";

print "Please be patient while the data is processed.\n";
    }#close if statement
else { #is input file does not exist
    print "ERROR: Input file does not exist.\n";
    die; #exit program
}#close else statement
}
#----------------------------------------------------------------------
sub mod_fort_20 { #Modifies the file fort.20 so that it may be used to run Magboltz
    $std_pres = 760; #standard pressure in Torr
    $std_temp = 20; #standard temp in Celsius

    open(MOD_FORT, ">fort.20") || die "Can't open fort.20"; #open fort.20 for
modifications

#write standard file format to fort.20
    printf MOD_FORT "%10d%10d%9.3f\n", 2, 2, 20;
    printf MOD_FORT "%5d%5d%5d%5d%5d%5d\n", $gas1, $gas2, 52, 52, 52, 52;
    printf MOD_FORT "%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f%10.4f\n",
$mix1, $mix2, 0, 0, 0, 0, $std_temp, $std_pres;
    printf MOD_FORT "%9.2f%9.2f%9.2f\n", $Es, $Btot, $Theta;
    printf MOD_FORT "%10d\n", 0;

    close(MOD_FORT) || die "ERROR: Can't close fort.20"; #close fort.20
}
#-----------------------------------------------------------------------
sub magboltz { #Uses the recently modified fort.20 to run the Magboltz 2 program
    $magin = "fort.20"; #Magboltz INPUT file name
    $magout = "magoutput.dat"; #Magboltz OUTPUT file name

    open(MAG_OUT, ">$magout") || die "Can't open $magout"; #open output file.

    $_ = `./magboltz-2.5.1 $magin`; #run the magboltz program and store the output
in the inherent variable
    print MAG_OUT "$_"; #print the stored output from the magboltz program to a
temporary output file
```

```
    close(MAG_OUT) || die "ERROR: Can't close $magout"; #close the temporary output
file.
}
#-----------------------------------------------------------------------
sub data_mine { #Gathers the drift velocity information from the temporary Magboltz
output file
$input = "magoutput.dat";

open(MINE_IN, $input) || die "Can't open $input"; #open the file that holds the
output from magboltz.

while(<MINE_IN>) { #this while-loop looks at one line of the temp output file at
a time
    if(/DRIFT/) { #if the line contains the string "DRIFT"
@array = split; #split the line around whitespace
if($array[0] eq X) { #if first array element is "X"
    $z_temp = neg_check (); #run the neg_check subroutine
    $z_drift = convert ("$z_temp"); #run the convert subroutine
    $z_error = $error; #set corresponding error value
    }
if($array[0] eq Y) {
    $phi_temp = neg_check ();
    $phi_drift = convert ("$phi_temp");
    $phi_error = $error;
}
if($array[0] eq Z) {
    $s_temp = neg_check ();
    $s_drift = convert ("$s_temp");
    $s_error = $error;
}
    } #close if statement
} #close while loop
close(MINE_IN) || die "Can't close $input";
close(DEC_TEST) || die "Can't close decimal_test.dat";
}
#-----------------------------------------------------------------------
sub neg_check { #Checks the drift velocities and removes any trash
    if($array[3] eq "=") { #positive drift velocities have element "=" as fourth
element in array
$error_temp = "$array[7]"."42";  #append trash (42) onto end of element
after '%'
@array_err = split(/%/,$error_temp); #split the error_temp variable at the
'%'
$error = $array_err[0]; #assign the appropriate error value
return $array[4]; #return the drift velocity value since value stands-alone
    }
    else { #negative drift velocities have the "=" attached to the value: it must be
removed.
$error_temp = "$array[6]"."42";
@array_err = split(/%/,$error_temp);
$error = $array_err[0];
$temp = "42"."$array[3]";
@array2 = split(/=/,$temp); #splits the new string $temp at the previously
bothersome "="
return $array2[1]; #return the new drift velocity value sans '=' but with
the negative sign.
    } #close else statement
}
#---------------------------------------------------------------------------
sub convert { #Drift velocity is reported in scientific notation from Magboltz.
#This string value must be converted to a decimal value.
    $pow = ""; #reinitialize $pow variable
    $string = $_[0]; #the input value is stored as $_[0] and reassigned to 'string'
    @array = split(/E/,$string); #split the line around the exponent
    $num = $array[0]; #assign digits to $num variable
    $pow_string = $array[1]; #assign power to the $pow variable
    @array2 = split(//,$pow_string); #split $pow_string into individual characters

    if($array2[0] eq "-") { #add negative sign to $pow if necessary
$pow = "$pow"."-"; }
    if($array2[1] ne "0") { #add tens digit to $pow if it's not a 0
$pow = "$pow"."$array2[1]"; }
$pow = "$pow"."$array2[2]"; #add the ones digit to $pow
    if($array2[1] eq "0" && $array2[2] eq "0") { #make $pow=0 if both tens and ones
digits are 0.
$pow = "0"; }

    $dec = $num * 10**$pow; #calculate the corresponding decimal value
    return $dec; #return the decimal value
}
#------------------------------------------------------------------------------
sub write_output { #Appends the output to the final drift velocity map
    open(OUTFILE, ">>$out") || die "Can't open $out: $!\n"; #open the final output
file for appending

    if($loopcount == 0) { #write output file header if first iteration
print OUTFILE "s    z    ws   wphi    wz   s_err   phi_err  z_err  sol
Vdrift\n";
    }

    #write the desired values to the final output file
    printf OUTFILE "%-9d%-10d%-10.4f%-10.4f%-10.4f%-9s%-9s%-9s%-9s%-9s", $s, $z,
$s_drift, $phi_drift,
$z_drift, $s_error,$phi_error, $z_error, $sol, $drift;
    print OUTFILE "\n";
    close(OUTFILE) || die "Can't close $out: $!\n"; #close the final output file
}
#-------------------------------------------------------------------------------
sub list_files { #Lists the files in the specified folder
    $directory = $_[0]; #set the value passed into subroutine as the $directory
variable
    $_ = `pwd`; #assign inherent variable to output of the pwd shell command
    chomp; #remove the trailing whitespace
    $cur_dir = $_; #assign $cur_dir to the working directory
    $dir_path = "$cur_dir"."$directory"; #assign the full directory path
    print "\n$dir_path\n"; #print the directory path for the user
    opendir(DIR,"$dir_path") || die "Can't open $dir_path directory"; #open the
directory
```

```
    while(defined ($file = readdir(DIR))) { #prints the name of each file in the
directory
     print "$file\n";
    } #close while-loop
    closedir(DIR); #close the directory
    return $dir_path; #return the directory path
}
```

F ElectronPath.c Code

```
/*ElectronPath.c calculates the path of an electron liberated at proton position
ss,phi,zz

Inputs: Gas code, gas mixture, current in the torus, and the potential across the
drift chamber.

Outputs: -set of files giving the s,phi,z locations of the electron along its path
through the RTPC
  -each output file is for a different z-value
-each z-value matches the center of a readout pad in the detector*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

#define TRUE 1
#define FALSE 0

#define TORADIANS (3.1415926535/180.0) // convert from degrees to radians
#define NUM_Z1 21
#define NUM_S1 15
#define NUM_Z2 5
#define NUM_S2 25
#define NUM_VAL1 315
#define NUM_VAL2 125
#define NUM_VAL_Z 120 //This is the number of Z values for the loop to run
#define LINLMAX  1400 /* maximum number characters per input line */
#define PI 3.1415926535
#define N_CHAN 3200
typedef char linetype[LINLMAX];
linetype textline;
linetype data;
//a single pad length in z (mm)
#define DEL_Z 5.0
//the angular coverage of a single pad (rad)
#define DEL_PHI 0.06442

//////////////////global variables/////////////////////////////////////////
typedef struct {
  float s; //mm
  float phi; //radians!
  float z; //mm
} cylindrical;
typedef struct {
  float phi; //radians
  float z; //mm
} coordinates;

coordinates padLocs[N_CHAN];
```

```
float sVals1[NUM_S1]; //the array of the s values from the field map
float zVals1[NUM_Z1]; //the array of the z values from the field map
float int_Bs[NUM_S1][NUM_Z1]; //the array of the Bs values from the field map
float int_Bz[NUM_S1][NUM_Z1]; //the array of the Bz values from the field map
int calcEB;

//to interpolate the Magboltz sim data
float sVals2[NUM_S2]; //the array of the s values where the sim was performed
float zVals2[NUM_Z2]; //the array of the z values where the sim was performed
float Ws[NUM_S2][NUM_Z2]; //the array of the radial velocity components
float Wphi[NUM_S2][NUM_Z2]; //the array of the angular velocity components
float Wz[NUM_S2][NUM_Z2]; //the array of the axial velocity components
float sVel, phiVel, zVel; //holds the s, phi, z components of velocity from the table

//Define command line input variables
int gas_code; //define the gas code
int gas_mix; //define the gas mixture percentage
int hePer; //percentage of Helium in the gas mixture
int side; //tells which half of the RTPC this is for 0=LEFT, 1=RIGHT
int B_value; //define the Torus current (A)
int E_value; //define the RTPC voltage (V)
float delt; //define the timestep (nanoseconds)
float z_max; //define maximum z value
int finalIndex;
int lastPoint;
int crossedGEM; //tells whether or not the electron has reached the first GEM
int t_gem, t_pad, t_0;//t_gem = time when electron crosses r=60mm
                      //t_pad = time when electron hit readout pad (r=69mm)
                      //t_0 = t_pad - t_gem = time offset
float phi_gem, phi_pad, phi_0;//phi_gem = phi when electron crosses r=60mm
                              //phi_pad = phi when electron hit readout pad (r=69mm)
                              //phi_0 = phi_pad - phi_gem = phi offset
//Define arrays
cylindrical path[10000];
float z_value[NUM_VAL_Z];
char path_file_name[40];
char offset_file_name[40];
char drift_file[40];
char z_file[20];

//Define file handles
FILE *outfile;
FILE *globalfile;
FILE *dvFile;
FILE *Zfile;
FILE *offsetfile;

//Function prototypes
void readDriftVel(); //reads drift velocity map into array
void readZvalues(); //reads pad z-values into array
void getDriftVel(float, float); //interpolates drift velocity
int calcPath(float, float, float, int, int); //calculates electron path

//------------------------------------------------------------------------------
```

```
//MAIN ROUTINE--------------------------------------------------------------------
int main(int argc, char **argv)
{ int i, j, k, n;
  float ss, phi, zz, zMin, zMax, phiMin, phiMax;
  int hitGEM, foundPad, hit_pad_index;
  int time;

   if( argc != 6) //verify correct number of command line inputs
    { fprintf(stderr,"Usage: electronPath [gas code] [gas mix] [B-value] [E-value]
 [delt]\n");
      exit(0);
    }

  //Read inputs from command line
  gas_code = atof(*++argv);
  gas_mix = atof(*++argv);
  B_value = atof(*++argv);
  E_value = atof(*++argv);
  delt = atof(*++argv);

  if(gas_mix == 8515) hePer = 85; //assign standard helium percentages
  else if(gas_mix == 8020) hePer = 80;
  else if(gas_mix == 7525) hePer = 75;
  else
    { printf("this is an odd He Mixture...?");
      hePer = -1000;
    }

//determine which half of the detector we're in
  if(E_value == 1638 || E_value == 1663 || E_value == 1689 || E_value == 1651 ||
E_value == 1676)
    {side = 0;}
  else if(E_value == 1659 || E_value == 1688 || E_value == 1718 || E_value == 1674
|| E_value == 1703)
    {side = 1;}
  else
    { printf("unknown chamber half\n");
      side = -1000;
    }
//assign the starting point for the path files
  ss = 30;
  phi = 0;
  zz = 0;
  z_max = 2;

  phi *= TORADIANS; //convert phi from degrees to radians

  //Define file names
  sprintf(drift_file, "DRIFT_MAPS/%d_%d_b%dA_V%d.drift",gas_code,gas_mix,B_value,
E_value);
  sprintf(z_file, "z.dat");

  //Gather inputs
  readDriftVel(); //Read in the drift velocity file
  readZvalues(); //Read in the z-coordinates to create path files for

  sprintf(offset_file_name,"PATH_FILES/%d_%d_b%dA_V%d/offsets.txt",gas_code,gas_mix,
B_value,E_value);
  printf("opening offset output file\n");
  if ((offsetfile = fopen(offset_file_name,"w"))==NULL)
    { printf(" failed to open offset output file \n");
      exit(1);
    }
  else
    { fprintf(offsetfile, "z_pad(mm)  t_off(ns)   phi_off(deg)  Vdrift\n");
    }

  printf("opening global path output file\n");
  if ((globalfile = fopen("globalPath.txt","a"))==NULL)
    { printf(" failed to open global output file \n");
      exit(1);
    }

  //START LOOP TO GENERATE PATH FILES ALONG SLICES IN Z
  for(n=0;n<NUM_VAL_Z;n++)
    { //Set initial path array elements
      path[0].s = ss;
      path[0].phi = phi;
      path[0].z = z_value[n]; //define a new z-value according to z_value array
      time = 0;
//name the path file for this particular slice
      sprintf(path_file_name,"PATH_FILES/%d_%d_b%dA_V%d/out%.2f.path",
gas_code,gas_mix,B_value,E_value,z_value[n]);

      printf("opening path output file\n");
      if ((outfile = fopen(path_file_name,"w"))==NULL)
{ printf(" failed to open output file \n");
  exit(1);
}
      else{ //print path file header and initial step
fprintf(outfile, "s(mm)   phi(rad)  x(mm)    y(mm)   t(ns)\n");
fprintf(outfile, "%.2f%.4f%.2f%.2f 0\n", ss,phi,ss*cos(phi),ss*sin(phi));
      }

      i=0;
      crossedGEM = FALSE; //reinitialize crossedGEM variable
//invoke calcPath function
      hitGEM = calcPath(path[0].s, path[0].phi, path[0].z, i, time);

      if(hitGEM)
{ //print to screen where electron hit readout pads
  printf("Electron hit GEM at s = %f, phi = %f, z = %f \n", path[finalIndex].s,
path[finalIndex].phi/TORADIANS, path[finalIndex].z);
  //calculate the time and phi offsets, then write them to the offset file.
  t_0 = t_pad - t_gem;
  phi_0 = phi_pad - phi_gem;
  fprintf(offsetfile, "%.2f   %d    %.2f    %d\n", path[0].z,t_0, phi_0/TORADIANS,E_value);
  foundPad = FALSE;
  i = 0;
  while((!foundPad)&&(i<N_CHAN))
    { zMin = padLocs[i].z - DEL_Z/2;
      zMax = padLocs[i].z + DEL_Z/2;
      phiMin = padLocs[i].phi - DEL_PHI/2;
      phiMax = padLocs[i].phi + DEL_PHI/2;
      if (path[finalIndex].phi < 0) path[finalIndex].phi *= 2*PI;
      if ((path[finalIndex].z + 100 > zMin) && (path[finalIndex].z + 100 < zMax)
  &&(path[finalIndex].phi > phiMin) && (path[finalIndex].phi< phiMax))
{ foundPad = TRUE;
  hit_pad_index = i;
}
      else
{ i++;
}
    }//close while loop
  if(foundPad)
    {   printf("hit pad %d z = %f phi = %f \n", hit_pad_index,
padLocs[hit_pad_index].z,padLocs[hit_pad_index].phi/TORADIANS);
    }
}//close if(hitGEM)
      fclose(outfile);
    }//CLOSE MAIN FOR-LOOP
  fclose(globalfile);

  return (0);
}//end main routine----------------------------------------------------
//--------------------------------------------------------------------
//READ DRIFT VELOCITY FUNCTION----------------------------------------------
void readDriftVel()
{ int i, j, k;
  float temp_s[NUM_VAL2];
  float temp_z[NUM_VAL2];
  float temp_ws[NUM_VAL2];
  float temp_wphi[NUM_VAL2];
  float temp_wz[NUM_VAL2];
  float trash;

  if((dvFile = fopen(drift_file,"r"))==NULL)
    { printf("failed to open drift velocity file\n");
      exit(1);
    }
  else
    { fgets(textline, LINLMAX, dvFile); //eat the first line
      printf("read drift velocity file\n");
    }
  for (i=0; i<NUM_VAL2; i++) //read drift velocity file into temp array
    { fscanf(dvFile,"%f %f %f %f %f %f %f %f %f %f",&temp_s[i],&temp_z[i],
     &temp_ws[i], &temp_wphi[i], &temp_wz[i], &trash, &trash, &trash, &trash, &trash);
      printf("read s = %f z = %f Ws = %f Wphi = %f Wz = %f\n", temp_s[i],
temp_z[i], temp_ws[i], temp_wphi[i], temp_wz[i]);
    }
  j = 0;
```

```
//store the appropriate drift velocity info
  for (i=0; i<NUM_VAL2; i+=NUM_Z2)
    { sVals2[j]=temp_s[i];
      j++;
    }
  for (i=0; i<NUM_Z2; i++)
    { zVals2[i] = temp_z[i];
    }
  k=0;
  for (i = 0; i < NUM_S2; i++)
    { for (j = 0; j < NUM_Z2; j++)
{
  Ws[i][j] = temp_ws[k]/1000; //units are microns/ns - change to mm/ns
  Wphi[i][j] = temp_wphi[k]/1000; //units are microns/ns - change to mm/ns
  Wz[i][j] = temp_wz[k]/1000; //units are microns/ns - change to mm/ns
  k++;
}
    }
}
//end read drift velocity function------------------------------------------
//--------------------------------------------------------------------------
//READ Z VALUES FUNCTION----------------------------------------------------
void readZvalues ()
{ int i;
  float temp_z;

  if((Zfile = fopen(z_file,"r"))==NULL)
    { printf("\nfailed to open z-file\n");
      exit(1);
    }
  else
    { for(i=0;i<NUM_VAL_Z;i++) //read z values from file and store in array
{ fgets(data, LINLMAX, Zfile);
  sscanf(data,"%f",&temp_z);
  z_value[i]=temp_z;
  printf ("z=<%f>\n",z_value[i]);
}
printf("\nread z_file\n");
    }
   fclose(Zfile);
}
//end read z values function------------------------------------------------
//--------------------------------------------------------------------------
//PATH CALCULATION FUNCTION-------------------------------------------------
int calcPath(float s, float phi, float z, int i, int time)
{ float s2, phi2, z2, thetaL, drLength1, drLength2;
  float phi_low, phi_high;
    if(s == 69) return (1);
  if(!crossedGEM && s>60) //assign values if electron passed first GEM
    { t_gem = time;
      crossedGEM = TRUE;
      phi_gem = phi;
    }
  if(z < -100 || z > 100) return (0); //verify electron is within detector
  else
    { if (phi >= 2*PI) phi -= 2*PI;
      if (phi < 0) phi = phi + 2*PI;
      getDriftVel(s, z); //call getDriftVel function

      thetaL = atan2(phiVel,sVel);
      drLength1 = sqrt(pow(phiVel,2) + pow(sVel, 2)) * delt;
      s2 = sqrt( pow(s,2) + pow(drLength1,2) + 2*s*drLength1*cos(thetaL) );
      if(s2 > 69)//the next time step takes us outside the detector
{//we need to find where the electron struck the pad plane
  s2 = 69;
  drLength1 = -s*cos(thetaL) + sqrt(s*s*cos(thetaL)*cos(thetaL) - (s*s - 69*69));
}
      phi2 = acos( (pow(drLength1,2) - pow(s2,2) - pow(s,2)) / (-2*s2*s) );
      phi2 = phi + phi2;
      drLength2 = zVel * delt;
      z2 = z; //Added this change so electron doesn't travel out of detector
      i++;
      time = time + delt;
      path[i].s = s2;
      path[i].phi = phi2;
      path[i].z = z2;
    //print next coordinate step to the path file
      fprintf(outfile, "%.2f   %.4f    %.2f    %.2f    %d\n", s2, phi2, s2*
cos(phi2), s2*sin(phi2), time);
      fprintf(globalfile, "%d     %d       %d   %.2f  %d   %.5f    %.5f\n", hePer,
 E_value, side, z, time, s2, phi2);
      finalIndex = i;
      phi_pad = phi2;
      t_pad = time;
      calcPath(s2, phi2, z2, i, time); //calculate next coordinate step
    }
}
//end calculate path function
//----------------------------------------------------------------------------
//GET DRIFT VELOCITY FUNCTION------------------------------------------------
void getDriftVel(float s, float z)
{ int i, j, k, l, m;
  float t, u;
  float pointer;

  if(z<0) z = -z;
//find correct s pointer values for interpolation calculation
  pointer = -1000;
  l = -1000;
  i=0;
  while((s > pointer) && (i<NUM_S2))
    { pointer = sVals2[i];
      if (s == pointer) l = i;
      else if (s <= pointer) l = i-1;
      else i++;
    }
//find correct z pointer values for interpolation calculation
  pointer = -1000;
  m = -1000;
  j=0;
  while((z > pointer) && (j<NUM_Z2))
    { pointer = zVals2[j];
      if (z == pointer) m = j;
      else if (z <= pointer) m = j-1;
      else j++;
    }
  //set interpolation variable
  t = (s - sVals2[l])/(sVals2[l+1] - sVals2[l]);
  u = (z - zVals2[m])/(zVals2[m+1] - zVals2[m]);
//interpolate drift velocity given pointers and variables
  sVel = (1-t)*(1-u)*Ws[l][m] + t*(1-u)*Ws[l+1][m] +
    (1-t)*u*Ws[l][m+1] + t*u*Ws[l+1][m+1];
  phiVel = (1-t)*(1-u)*Wphi[l][m] + t*(1-u)*Wphi[l+1][m] +
    (1-t)*u*Wphi[l][m+1] + t*u*Wphi[l+1][m+1];
  zVel = (1-t)*(1-u)*Wz[l][m] + t*(1-u)*Wz[l+1][m] +
    (1-t)*u*Wz[l][m+1] + t*u*Wz[l+1][m+1];
}
//end get drift velocity function----------------------------------------
```

References

[1] Experiment E03-012: *http://www.jlab.org/˜kuhn/BoNuS_Welcome.html*

[2] Hall B website: *http://www.jlab.org/Hall-B/*

[3] Thomas Jefferson National Accelerator Homepage: *http://www.jlab.org*

[4] GEM information webpage at CERN: *http://gdd.web.cern.ch/GDD/*

[5] Magboltz 2 information webpage: *http://cyclo.mit.edu/drift/www/aboutMagboltz.html*

[6] S.F. Biagi, Nuclear Instruments and Methods in Physics Research Section A, Vol. 421 (1999) 234-240 *Monte Carlo simulation of electron drift and diffusion in counting gases under the influence of electric and magnetic fields*