Study and Construction of a Thick-Gas Electron Multiplier

A *project report* submitted by

Danush S
Roll Number: 1711048

Supervised by

Prof. Bedangadas Mohanty

to the

School of Physical Sciences
National Institute of Science Education and Research

June 7, 2020

Acknowledgements

I would start by thanking Prof. Bedangadas Mohanty for providing me the opportunity to work under his guidance for this project, which has been a tremendous factor into shaping my further interest in research. I would thank Dr. Varchaswi K S Kashyap who has been a source of constant guidance throughout this project, and one who has made me appreciate the amount of work that goes into setting up an experiment from the ground up. I would like to thank Dr. Abhik Jash, Vijay Iyer, Soumik Chandra, Tanmay Pani, Aman Upadhyay and Subasha Rout for their help and motivation they have provided me. I finally thank my parents, friends and all my lab-colleagues for their support.

Abstract

This project involves studying the Thick-Gaseous Electron Multiplying (Thick-GEM) detector, a gaseous detector used in High Energy experiments. We then constructed and assembled a 10×10 cm^2 Thick-GEM detector at NISER. Simulation of the same detector was also carried out, on Garfield++ using field map data generated on COMSOL Multiphysics®. Owing to long run time durations, we looked at other data structures to store the data in, to help accelerate the process of simulation.

Contents

List of Figures

List of Tables

Chapter 1

Gaseous Detectors

A crucial objective in most nuclear or particle physics experiments is the detection of the radiation/particles that are emitted. We will first study how different particles and radiation interact with matter, which is used as a basis for the concept of a detector. The variety of these processes is quite extensive and as a consequence, a large number of detection devices for particles and radiation exist. Depending on the energy of the incident radiation/particle, there are various methods for detection. Particle detection can be done through detectors like the Geiger-Muller Counter and the Scintillation Counter where the output is in an electrical form, or through (now rarely used) detectors like bubble chambers and spark chambers where the output has to be photographed. Further advanced techniques of detection have been developed since then, and we will briefly go through some relevant detectors, studying their structural design and the physics behind them.

1.1 Interaction of Charged and Neutral Particles in a Gas

Charged particles are usually involved in electromagnetic interactions with the gas particles. Upon interaction with a charged particle, gas particles can undergo radiation-less rearrangements, dissociate or get excited or ionized, with the emission of photons or the appearance of free ion-electron pairs. At very high particle energy, other mechanisms like Bremsstrahlung, Cherenkov radiation and transition radiation can occur. Electrons and photons created by the primary encounters can further interact with the gas molecules, causing further ionizations called secondary ionizations. Charged particles can also undergo mechanical elastic collisions, and the slowing down in gas is mainly due to multiple inelastic processes of excitation and ionization.

Neutrons are particles with no charge and hence are not involved in any electromagnetic interaction with charged particles. But they can interact with other nuclei through processes like radioactive capture and nuclear reactions with the emission of particles (like protons, alpha particles, etc.) take place. Neutrons also interact with matter through elastic and inelastic scattering.

1.2 Interaction of Photons in a Gas

Interaction of photons with gas particles depends on the incident energy of the photons, density and other physical properties of the medium. Apart from photons exciting or ionizing gas molecules, the interaction of photons with gas molecules happen majorly through the Photoelectric effect, the Compton Effect or the Pair-Production Effect.

The photoelectric effect refers to the emission of electrons when a photon interacts with it. It occurs when the energy of the incident photon is higher but around the magnitude of the ionization energy of the atom. Sometimes the photon can eject an inner shell electron followed by rearrangement of electrons in the atom producing another photon or electron during the process.

Compton effect occurs when the energy of the incident photon is to some extent greater than the ionization energy of the atom. It is the process of transferring some of the energy and momentum to an electron the photon collides with.

The Pair-Production effect is observed when the incident photon is of energy greater than twice the mass of an electron. Pair production often refers to a photon creating an electron-positron pair near a nucleus.

The above interactions should cover a majority of possible interactions, but things get complicated when our gas does not comprise of just single atoms, but a mixture of atomic and molecular gases. In molecular gases, energy can also be spent in rotational, vibrational energy, etc. An account of more than 20 processes that can follow the inelastic interaction of electrons and molecules are provided in [1].

1.3 Science behind Signal Generation

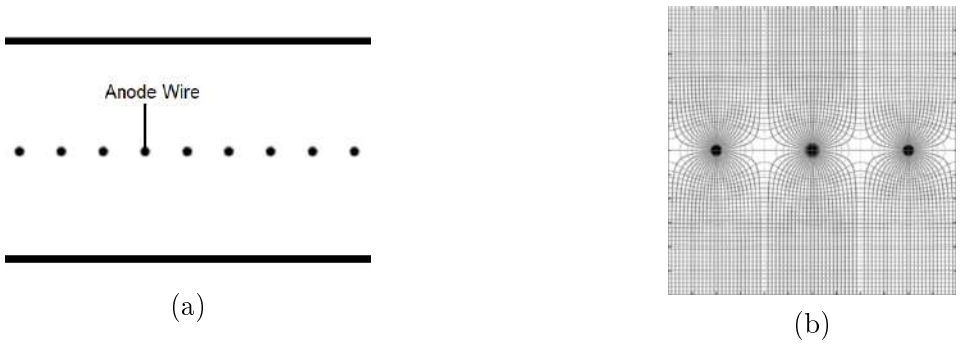
The working principle behind gaseous detectors is to detect incoming particles by producing a readable electric current through ionization of the gas particles. Once an ionizing particle enters the gaseous detector, it ionizes and produces one or more primary ion-electron pairs depending on factors like the energy of the incident particle. Due to the external electric field, usually, these electrons accelerate and gain enough energy to create secondary interactions and produce more ion-electron pairs. Under these conditions, the number of electrons grows rapidly forming an avalanche multiplication, thus producing a readable electric signal on the electrodes [2].

1.4 Multi-Wire Proportional Chamber

The Multi-Wire Proportional Chamber (MWPC) consists of a set of thin, parallel and equally spaced anode wires, symmetrically placed between two cathode planes as shown in the schematic Figure 1.1a. An electric field as seen in Figure 1.1b develops when symmetric

negative potentials are applied to the cathodes, and the anodes are grounded. The chamber is filled with certain gases which get ionized when an ionizing particle passes through.

When an ionizing particle passes through the chamber, electrons and ions created in the gas volume drift along the field lines approaching the high field region close to the anode wires, where a localised avalanche multiplication can occur. The nearest wire collects these charges and the magnitude of the charge depend on the ionizing ability of the incident particle. By computing pulses from all the wires, the particle trajectory can be determined.



(a) (b)

Figure 1.1: (a) Cross-sectional schematic of an MWPC and (b) Field lines and Equi-potentials near the anode wires of an MWPC [3]

Typical values for the anode wire spacing range between 1 and 5 mm, the anode to cathode distance is 5 to 10 mm. The operation gets increasingly difficult at smaller wire spacings, which prevented taking this direction for obtaining higher spatial resolution [4]. Localization accuracies (of the ionizing event) of 50–100 µm can be achieved with a measurement of the drift time, or of the cathode induced charge profiles. The rate capability of MWPCs is also limited to a few kHz/mm^2 by the build-up of a positive ion space charge, dynamically modifying the electric fields.

1.5 Micro-Strip Gas Counter

A micro-strip gas chamber consists of thin parallel metal strips, alternating wide (anodes) and narrow (cathodes) laid on an insulating support usually at a pitch of a few hundred microns. An upper drift electrode at a negative potential, delimits the sensitive gas volume where electrons are released by the ionizing particle. Applying proper potentials to the electrodes, an electric field builds up (See Figure 1.3) such that electrons released in the drift space upon interaction by incident photons or charged particles are collected and multiplied when reaching the anodes [3].

With appropriate potentials applied to the electrodes, when an incident ionizing particle passes, electrons are released in the drift gap. These electrons move towards the strips and multiply in the high field region close to the anodes. For the convenience of readout, the

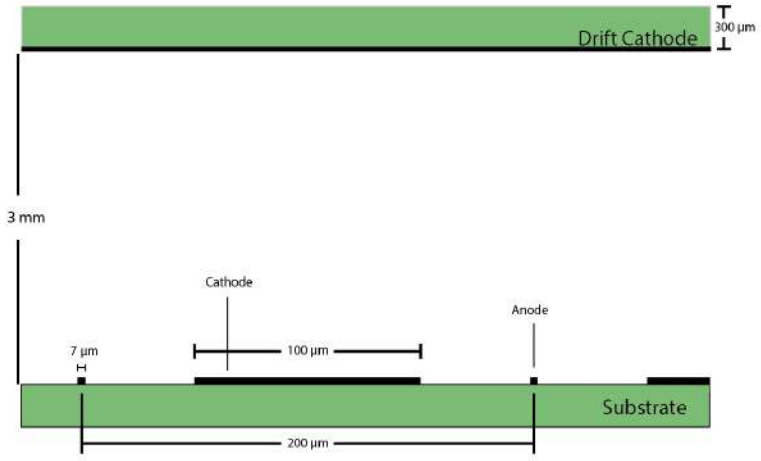


Figure 1.2: Schematic diagram of the MSGC

anode strips are at ground potential, with the cathodes connected individually or in groups to the negative potential through high-value protection resistors. All field lines from the drift volume terminate on the anodes, providing full electron collection efficiency.

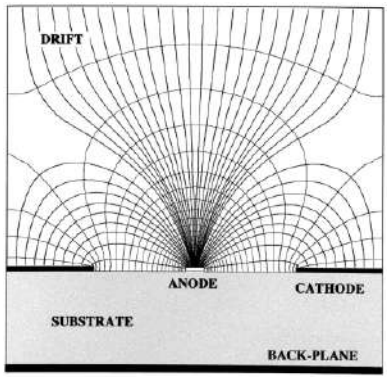


Figure 1.3: Electric field in the MSGC near the strips [3]

MSGCs are used as soft X-ray detectors, such as in crystal diffraction experiments and X-ray astronomy. Although the resolution of the MSGC is higher than that of the MWPC, due to the high electric fields near the strips, the MSGC detectors were prone to damages due to discharges being induced frequently. Thus MSGC-based detectors were eventually discontinued for most applications.

1.6 Gaseous Electron Multipliers

Gas Electron Multipliers (GEM) are electron multipliers which consist of a copper-clad polymer foil perforated by a high density of holes. The GEM electrode is pierced by a regular array of hourglass-shaped holes, typically 100 per mm^2, produced by certain etching techniques. High voltages are applied on both copper surfaces to provide for a high voltage gradient. The applied voltage difference across the GEM electrode is generally around 200-500V. The shape of the hole ensures a high dipole field and a more focused electron path at the centre. The GEM foil acts as a charge pre-amplifier, to a large extent preserving the original ionization pattern. There are readout pads under the GEM electrode to collect all charges that come through the GEM electrode.

When an ionizing particle passes through, it produces primary charges by ionization above the electrode, and by means of drift and diffusion processes, the charges are transported through the gas volume to the amplification region close to the electrodes and these charges will attain high velocities due to the high electric field and cause further ionizations leading to an avalanche effect. These charges are then collected by the readout pads under the GEM electrode. Separated from the multiplying electrode, the charge collection and readout plane can be patterned at will with strips or pads; usually, they are a set of perpendicular strips to serve as a 2-dimensional projective readout.

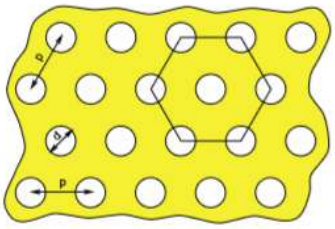


Figure 1.4: Hole pattern in the GEM electrode [3]

The advantages of a GEM detector [3] are:

- A sturdy construction with a separation of the multiplication electrode from the signal pickup electrodes, thus minimizing the likelihood of damage due to discharges.
- The possibility of cascading several electrodes, in what is called a multi- GEM chamber, permits one to reach very high overall gains.
- The readout electrode can be patterned at will, a common choice being two sets of perpendicular strips, to perform bi-dimensional localization of tracks.

Apart from the fact that there is a high percentage of chance that MSGCs could get damaged, another reason why there was a need for a better detector was that one did not

have high gains with MSGCs. But for operation in high gain, the chances for breakdown increases. Hence the GEM foil helped overcome this hurdle. Also owing to the high-accuracy localisation properties and fast response, GEM detectors have been of recent interest to the Medical Imaging industry, wherein GEM-based detectors have been used to detect and thus produce X-Ray images of objects. GEM foils have been used in various other technologies like radiography and X-Ray Fluorescence Spectroscopy.

1.7 Thick-Gaseous Electron Multiplier

Structurally, a Thick-Gaseous Electron Multiplier (Thick-GEM) [5] is similar to a GEM electrode, but some structural aspects are different in size when compared to a GEM electrode (See Table 1.1). Thick-GEMs have a cylindrical hole shape, as compared to an hourglass shape in GEMs, which also means the etching process to make these holes are different. Keeping in mind the difficulty in making a precision device such as a GEM foil with its very fine hole diameter along with its small pitch, the Thick-GEM came out to be a device cheaper and easier to manufacture. To make a GEM electrode one needs high-precision tools to make structures with such dimensions. Figure 1.5 and Table 1.1 give us a better idea of the difference between both detectors. The typical potential difference applied across the Thick-GEM electrode depends on the central PCB thickness, but the electric field magnitudes range around 15-25kV/cm. GEM electrodes, however, have field values to be around 30-80kV/cm.

Measurement	GEM	Thick-GEM
Hole Diameter (Inner)	50 µm	0.4mm
Hole Diameter (Outer)	70 µm	0.6mm
Pitch	150 µm	1mm
Thickness	60 µm	0.5mm

Table 1.1: Structural comparison between a typical GEM and Thick-GEM electrode

Owing to the larger dimensions and simpler hole structure, we can see that manufacturing Thick-GEMs are easier compared to GEM electrodes. Thick-GEM foils were developed to provide a more robust charge amplifying stage to operate under critical conditions. The electron collection in Thick-GEMs is more effective than that of GEMs because the hole-diameter is larger than the electron's transverse diffusion range when approaching the hole. Typical GEM electrodes are also sensitive to sparking and can be permanently damaged after a significant discharge, but the etched rim in a Thick-GEM helps reduce edge discharges.

The physics behind what follows once an ionizing particle enters the chamber is almost very similar to what was explained in the GEM section. That is, the ionizing particle produces primary charges by ionization. Then, these charges are transported through the volume by drift and diffusion towards the Thick-GEM electrode, where the high magnitude

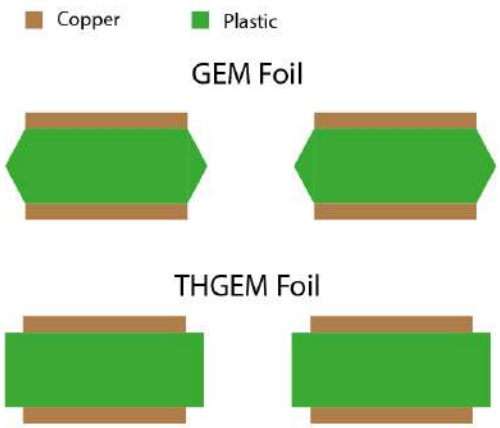


Figure 1.5: Cross section schematic of the GEM and Thick-GEM holes (Not to scale)

of the electric field will cause avalanche effects to obtain huge electron multiplication. These charges are finally collected by the readout pads under the Thick-GEM electrode.

1.7.1 Detector Setup

The detector mainly comprises of 3 main elements: the drift electrode, the Thick-GEM foil and the readout pads; all of them placed inside a gas enclosure, with a window on top that is transparent to incoming particles but yet keeping the detector air-tight. See Figure 1.6 for a better understanding. The drift electrode is usually a metallic mesh with a voltage to provide a drift field, usually of magnitude 5kV/cm. As the name explains, the drift region makes the charges produced, attain a drift velocity so they are directed towards the Thick-GEM electrode. The electric field produced in the Thick-GEM holes due to the voltage supplied on both copper surfaces is usually around 15-25kV/cm. Under the Thick-GEM electrode is the induction gap with an electric field, usually of magnitude 6-7kV/cm. This region helps in directing the electrons to the readout pads where the signal-current is generated, thus giving us a readable output.

The entire setup will be in an airtight gas enclosure made with perspex on the sides, Mylar as the top layer (also serving as a window for incoming particles) and the bottom will be a PCB board with readout pads. See Figure 2.1 for better insight. The contacts for the 4 potentials for the detector will all be connected to a serial 10MΩ protective resistor [6], and a 30Hz low pass filter [7] and thus to CAEN high voltage power supply. The protective resistor is to protect against current surges, and the low pass filter is to filter out high-frequency current. To make the low pass filter, we will be using a 2.2nF capacitor and a 15MΩ resistor. The readout strips will all be connected to a pre-amplifier, linear amplifier and a multi-channel analyser [6].

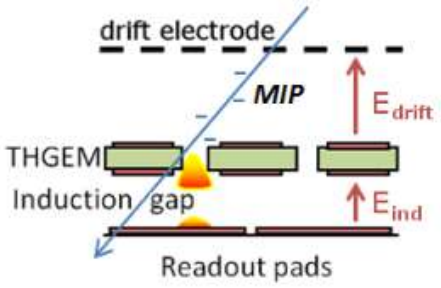


Figure 1.6: A single Thick-GEM detector setup [7]

1.7.2 Gas Selection

For the selection of the gas, one needs to keep in mind that the gaseous constituents should not have a high electron affinity to attract the electron before it even begins the avalanche effect. Keeping in mind the physical conditions required to keep the substance in its gaseous state, noble gases are good choices. After ejecting an electron one might argue that the noble gas atom attains halogen configuration, but by the time it pulls the electron, the electron under the external electric field would have begun subsequent collisions. We add other relatively inert gases like CO_2 and CH_4 called as quenching gases to reduce the gain, as high gain also leads to higher discharge probability. Higher discharge probability means higher chances for the Thick-GEM foil to get destroyed.

Chapter 2

Detector Fabrication

2.1 AutoCAD Models

The preliminary design of the detector was made using AutoCAD, a 3D designing software to serve as a template for the plan, that one could look at for direction/guidance while we were building the detector. These include dimensions of parts, positions of where they were to be placed, design intricacies and material composition.

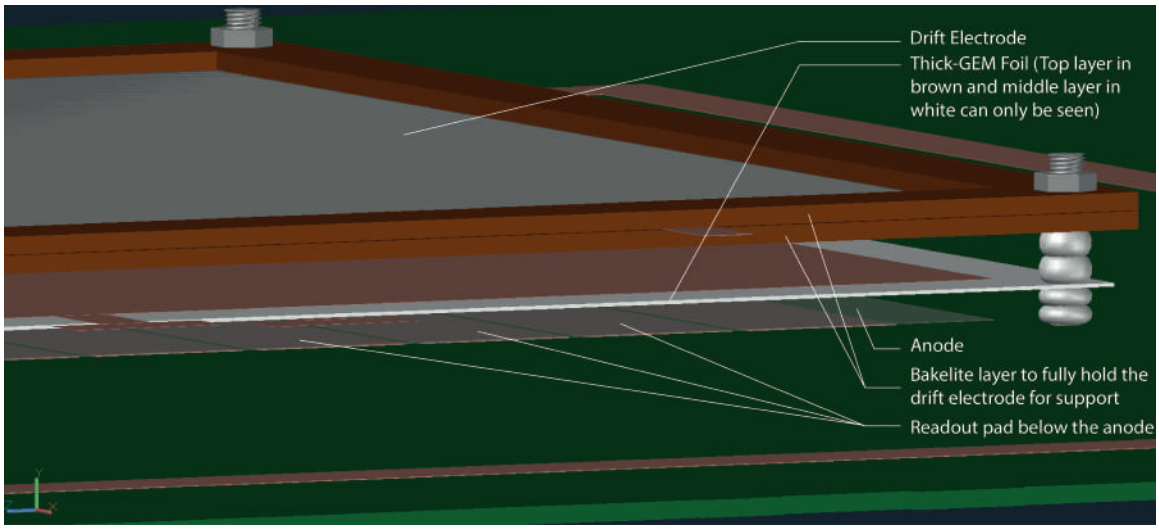


Figure 2.1: Isometric view of the detector showing the two electrodes and the Thick-GEM Foil

The drift electrode is an Aluminium sheet with a contact to provide for a connection to the voltage source. The sheet is held by a Bakelite support to keep it flat so it does not bend due to its weight. The dimensions of the drift electrode open to incident particles is $10\times10\text{cm}^2$. Under the drift electrode, is the Thick-GEM electrode which has three layers, a top copper plate, a middle PCB layer and a bottom copper plate, represented by light

brown, white and light brown layers in Figure 2.1 respectively. The total functioning area of the Thick-GEM foil is 10x10cm^2 and the foil is 0.25mm thick. Under the Thick-GEM foil is the anode with an area of 10x10cm^2 open to incident electrons. The anode is made of mylar and is pasted on top of the readout strips which in turn are attached to the bottom PCB base.

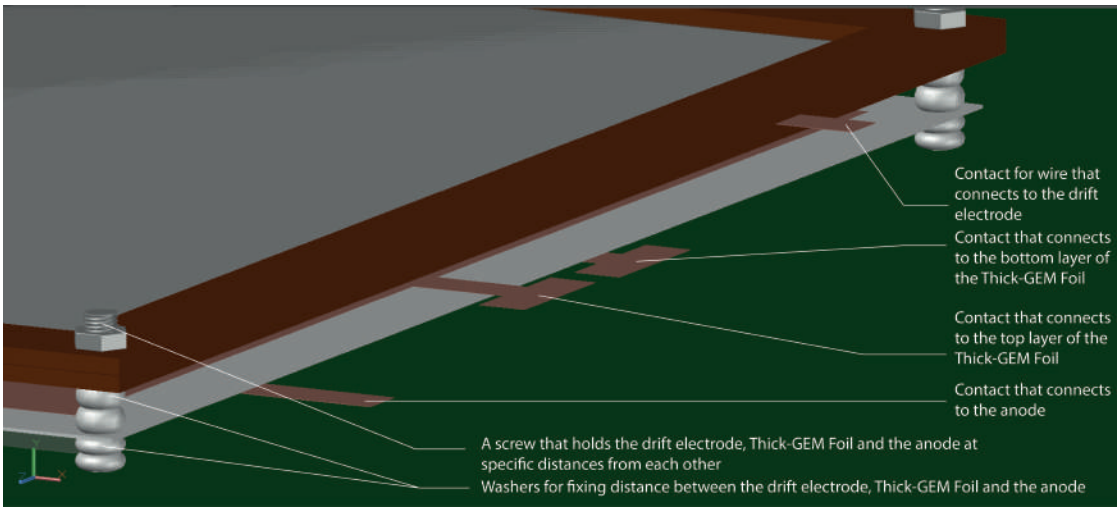


Figure 2.2: Second isometric view of the detector

From Figure 2.2 we can see that the Thick-GEM foil and the drift electrode are kept at a height from the base using 4 screws and washers. There is a gap of 3mm between the anode and the Thick-GEM foil, and a gap of 5mm between the Thick-GEM foil and the drift electrode.

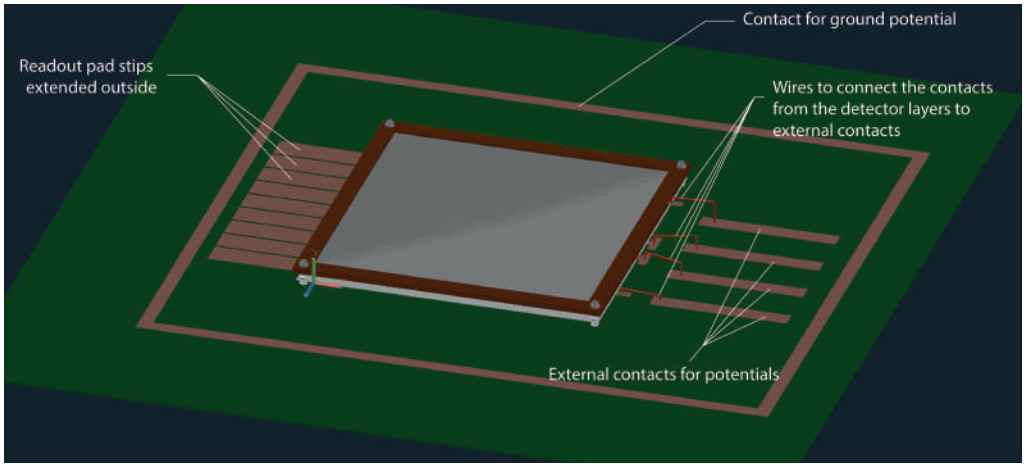


Figure 2.3: Top view of the detector

The readout strips have been extended further to make the connections for data collection

easier. There is an outer rectangular strip for the ground connection. The contacts from different layers of the electrode have been connected to other contacts for extension purposes, which will come out of the gas enclosure and can be understood by looking at Figure 2.4. This makes it easier to apply the required voltages to each layer.

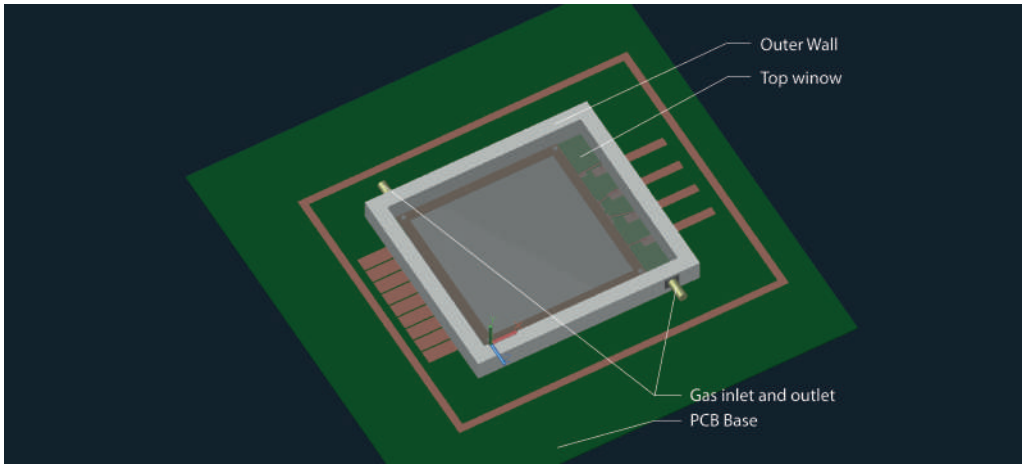


Figure 2.4: Top view of the detector showing the two electrodes and the Thick-GEM Foil

We finally close off everything by an enclosure of height 1.5cm which will make the entire setup airtight, and provide for an inlet and an outlet for gas flow. The top portion of the enclosure is made of Mylar which would serve as a window for incoming particles. Perspex is used for the side walls of the air-enclosure.
The contacts for the 4 potentials for the detector will all be connected to a serial 10MΩ resistor and a 30Hz low pass filter and thus to the power supply. The following potentials will be supplied to the contacts:

Surface	Potential (V)
Drift Electrode	-2400
Top layer of Thick-GEM foil	-1600
Bottom layer of Thick-GEM foil	-1000
Anode	0

Table 2.1: Potentials to supplied to each electrode in the detector

And thus we would have the following field magnitudes for different regions:

Region	Height (mm)	Field (kV/cm)
Drift region	4	2
Thick-GEM foil region	0.25	24
Induction region	2	5

Table 2.2: Electric Field magnitudes for different regions in the detector

The schematic of the setup alongwith the circuitry is as shown below:

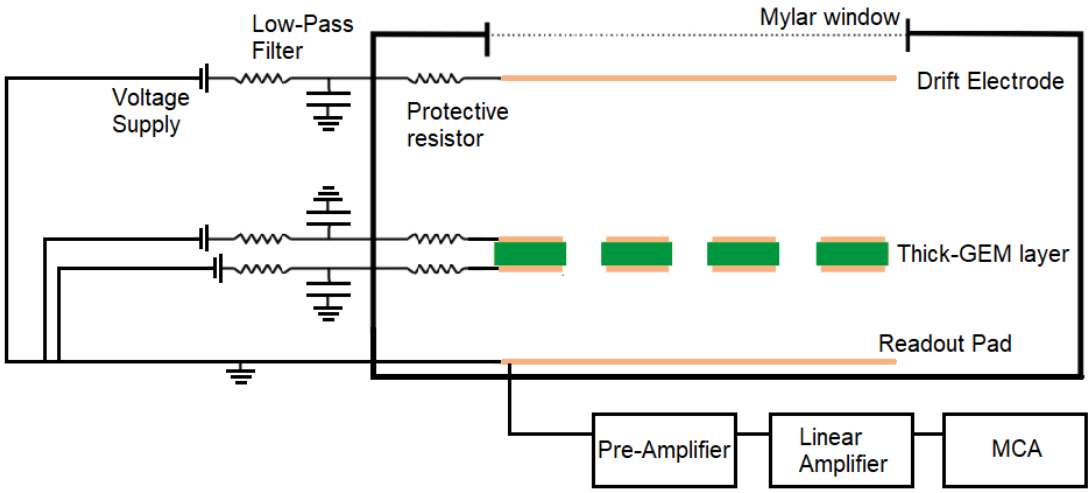


Figure 2.5: Schematic of the circuits involved

2.2 PCB and Readout Strips

Ten readout strips made of conducting material were required to be placed under the Thick-GEM electrode. So a PCB board with copper-clad on both sides was taken, and after etching out certain pattern of the copper, it would serve as the readout strips. The PCB board also serves as the base for the gas enclosure, on top of which the perspex sides were glued.

In order to etch out the unwanted copper, we carried out a displacement reaction using a $FeCl_3$ solution. The reaction involved is:

$$FeCl_3 + Cu \rightarrow FeCl_2 + CuCl \tag{2.1}$$

We prepared a mask, acting as a physical barrier, under whose region, the $FeCl_3$ would not react. So the $FeCl_3$ would remove copper wherever the mask was not present. We printed

the mask layout on A4 sheets using a printer. Then the A4 sheets were placed at appropriate positions on the PCB board and ironed. The toner ink on the sheets got transferred to the PCB board thus completing the masking process. Then an appropriate $FeCl_3$ solution was prepared and the board was placed in the solution and was agitated until all the copper from the desired regions was removed. To then remove the mask, we washed the toner ink with Propanol which removed the ink but not the Copper under it. The final product is as shown in Figure 2.6.

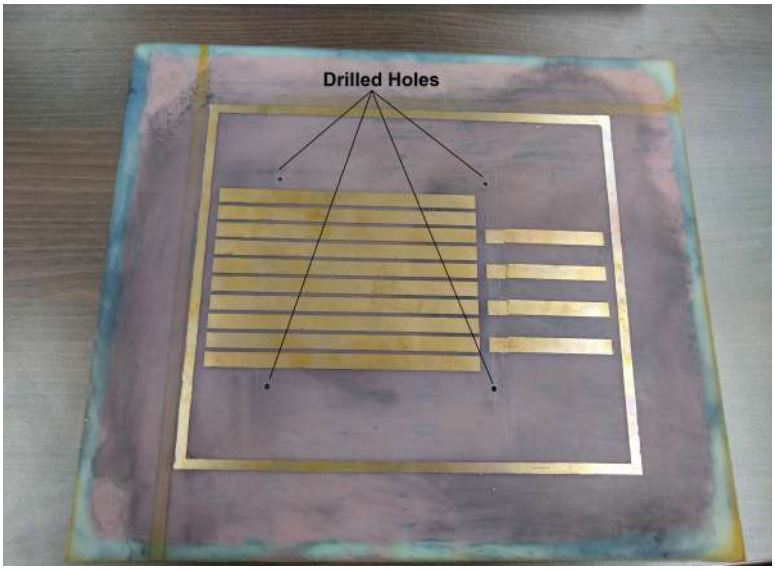


Figure 2.6: The etched PCB with holes

We also drilled holes on the PCB through which screws would pass to hold the Thick-GEM and the drift electrode.

2.3 Air Enclosure

In order to keep everything airtight, a frame made of Perspex along with an inlet and an outlet for the gas flow was built. This would be glued onto the PCB with another Mylar covering on top to seal it off. All components inside this chamber would be in an airtight environment with gas being brought in and flushed out at a constant rate.

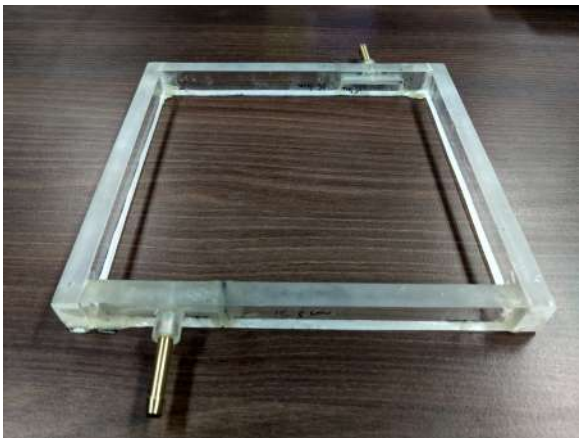
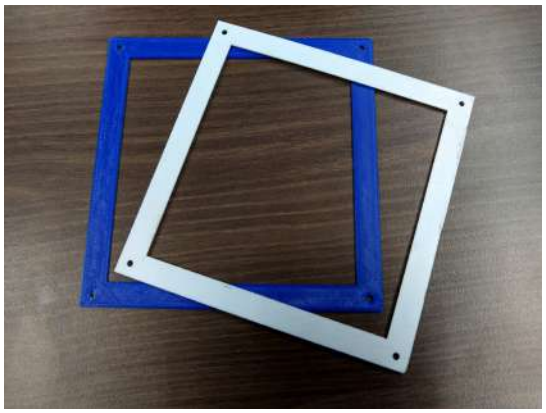


Figure 2.7: The Perspex frame with inlet and outlet valves

2.4 Drift Electrode

The 3D printer of NISER's Robotics Lab was used to print spacers and the drift electrode frames (See Figure 2.8a). To make the drift electrode, we took a Mylar sheet and cut it to the dimensions of the frame and covered the Mylar sheet with Aluminium tape to which the voltage connection would be made. Then this Mylar sheet was pasted on top of the frame (See Figure 2.8b).



(a)

(b)

Figure 2.8: (a) The 3D printed frames for the drift electrode and (b) The final drift electrode

2.5 Current Setup

After constructing all the required parts, the only task left was to assemble all of them. The setup after a temporary assemble looked like as shown in Figure 2.9. The setup shown in Figure 2.9 needed the electrical connections to be made as was shown in Figure 2.5, after which the assembling would have been completed, left for testing.

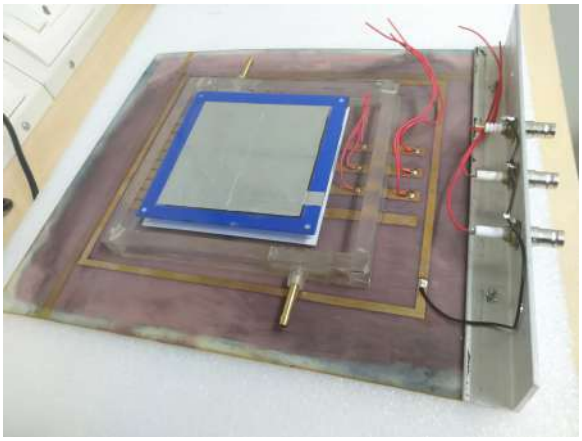


Figure 2.9: The Current Setup of the Detector

Chapter 3

Simulating the Thick-GEM detector

We attempted in simulating the same Thick-GEM detector we were experimenting with, and to do so we used Garfield++ [8], COMSOL Multiphysics® and ROOT [9]. Instead of simulating a full-sized 10×10 cm^2 Thick-GEM electrode at the beginning, we first tried to simulate a single Thick-GEM hole with an appropriately sized readout pad and drift electrode in an Argon-CO_2 (80%-20%) gas mixture. To simulate the Thick-GEM, we are also required to make a gas file that contains electron transport parameter tables (which includes information like drift, diffusion, gain and attachment of electrons in gases with applied electric and magnetic fields), the code for which, is in Appendix B. After generating the field data from COMSOL Multiphysics®, using ROOT we generated a file that contains the field data in a format that can be read by Garfield++. We tried 2 ways of reading this ROOT file in Garfield++ and both were successful. We will compare both these methods and finally show the results we have obtained.

3.1 Approach 1

We generated the electric field data generated on COMSOL Multiphysics® and used it in Garfield++ by writing the COMSOL Multiphysics® data in a ROOT file. Garfield++ needs electric field values at various points in the volume to carry out certain operations, but COMSOL Multiphysics® data has these values of electric field/potential at only finite number of points in the volume. Let us say Garfield++ asks us for the electric field at a point x', so we find a point x nearest to x' where the data is available and substitute it as the data for x'. So the following work flow (Figure 3.1) was included.

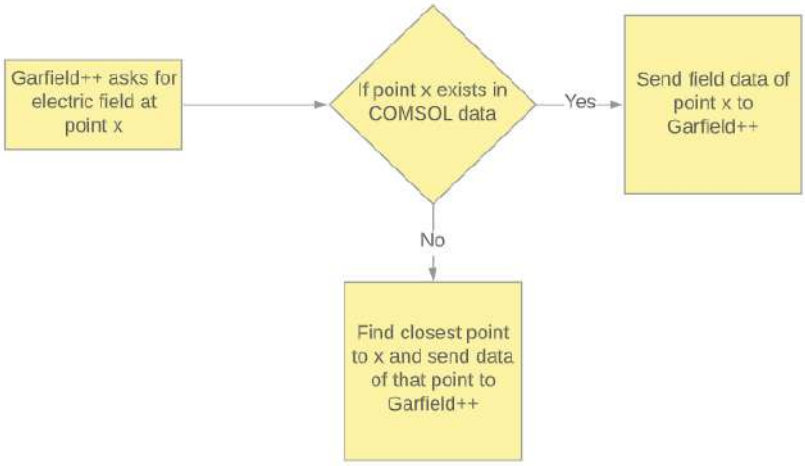


Figure 3.1: Work flow involved in Approach 1

Now given that we found a way to use the COMSOL Multiphysics® data in Garfield++, now we create the geometry of the setup on Garfield++ and then plot drift lines and calculate signal generated. We first tested the process for only one hole of the Thick-GEM electrode and then plan on running it for the entire detector setup.

3.2 Approach 2

Approach 1 involved reading the data from COMSOL Multiphysics® and storing a root file that contained a tree with 6 branches, namely, x coordinate, y coordinate, z coordinate, E_x component, E_y component, E_z component and Potential (E_i is the electric field in the i-direction). While plotting the potential contours, it searched for the nearest point through brute force method, and that posed as a difficulty as 1 potential plot in itself took about 2 hours to produce. The reason for this is that for simulating one Thick-GEM hole itself, COMSOL Multiphysics® generated about 600,000 points, which is the very reason we started with simulating a single hole rather than the entire detector in the beginning. So a need for a faster searching algorithm was required, for which we constructed a KD-Tree, a data structure that reduces the time complexity for searching to O(log(n)), as compared to that of O(n) in brute force method. KD-Tree also presents us to search for the nearest neighbour, which are present in the ROOT class TKDTree.

So one way to create a KD Tree is, let us say we are dealing with 2-dimensional data (represented by the left box in Figure 3.2), so we aim to create a 2D tree. Each point has x and y coordinates. Let us say we are creating the tree and point 1 is entered, since it's the first point, it is the root node. Next, point 2 has a larger x-coordinate that 1, so it is placed to the right of 1. Point 3 has a larger x-coordinate than 1 so it goes to the right of 1. Now point 3 has a larger y-coordinate than 2 so it is placed to the right of 2. For

point 4, x-coordinate of 1 and 4 are compared, since it is larger, it goes to the right, then y-coordinate is compared with 2, so it goes to the left.

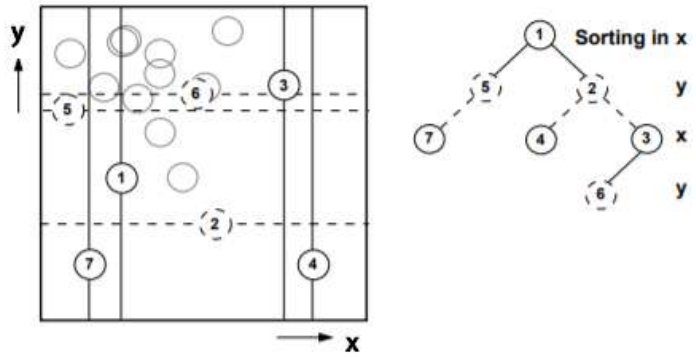


Figure 3.2: Input data points and the generated 2D Tree [10]

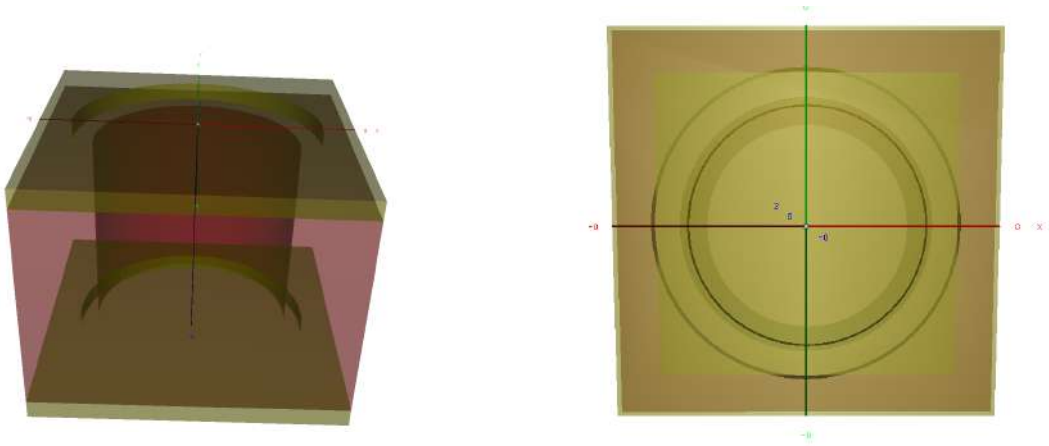
As explained further in [10], the tree is sequentially filled by taking every point and, while descending the tree, comparing its x and y coordinates with the points already in place. Whether x or y are used to compare depends on the level within the tree. On the first level, x is used, on the second level y, on the third again x and so on. The result for the data we have until point 7 is shown in the right part of Figure 3.2.

3.3 Geometry

The geometry of a single Thick-GEM hole was first made on Garfield++, with an appropriately sized readout pad and drift electrode.

- The hole diameter was 200 microns with an etched rim of 20 microns.
- The height of the hole was 286 microns (36 microns for 2 copper layers and 250 microns for the plastic layer).
- The drift electrode and readout pad both were of area 300×300 micron^2 with a thickness of 18 microns.
- The origin is defined to be the at the cross-hairs we see in figures 3.3a and 3.3b.
- Distance from the Thick-GEM to the readout is 2mm and the distance from the drift electrode to the Thick-GEM electrode is 4mm.

Subsequent results will be collected for a single hole. If successful, the code will be run for a Thick-GEM foil of size 5mm×5mm, with a readout strip and a drift electrode.



(a) (b)

Figure 3.3: (a) Geometry of a Thick-GEM hole made using Garfield++ and (b) Top view of the geometry

3.4 Potential Plots

We built the same structure (as shown in Section 3.3) on COMSOL Multiphysics® and exported the required files.

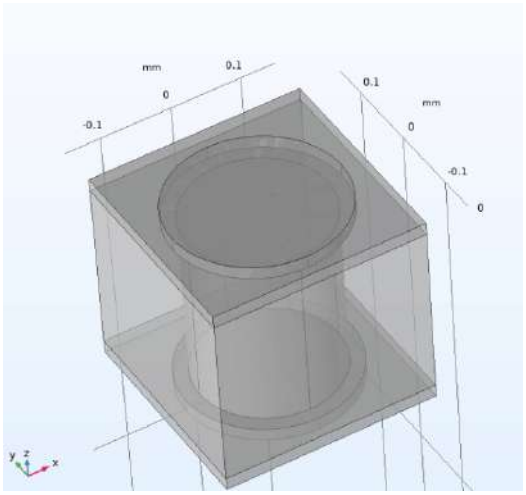
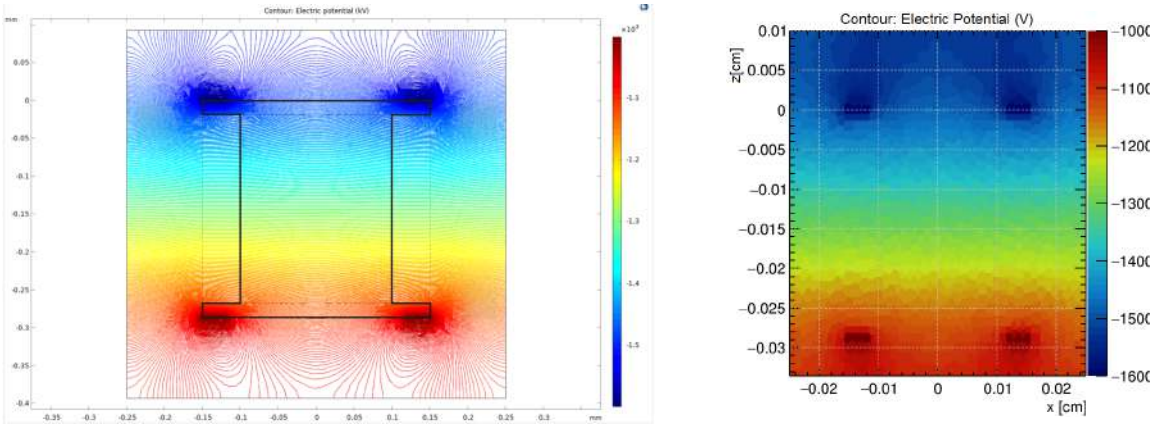


Figure 3.4: Geometry of a Thick-GEM hole made using COMSOL Multiphysics®

We first compared the potential contours generated on COMSOL Multiphysics® and Garfield++. The plots in Figures 3.5a and 3.5b are contour plots across a single Thick-GEM hole. We can see the outline of the Thick-GEM hole in Figure 3.5a.

We see that both plots are similar and we thus move forward to plotting drift lines and



(a) (b)

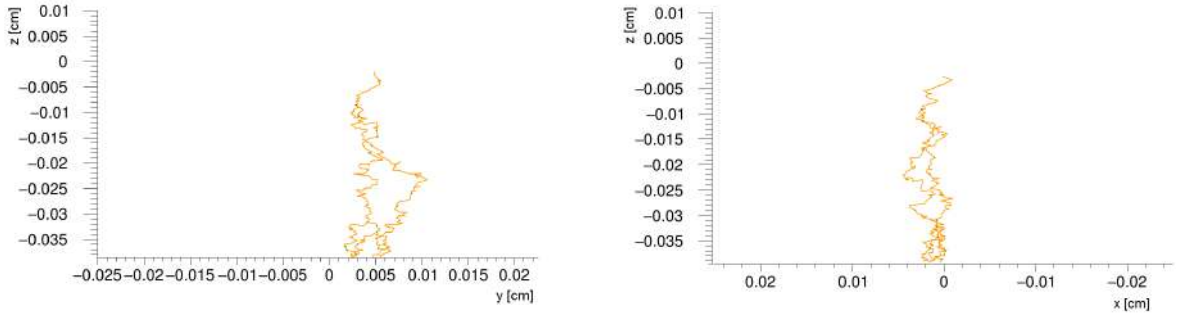
Figure 3.5: Contour plot (across a Thick-GEM hole) of (a) Electric Potential (kV) generated on COMSOL Multiphysics® and (b) Electric Potential (V) generated on Garfield++

calculating the signal.

Same plots were obtained using both Approach 1 and 2, although there was a considerably huge time difference to produce these plots. Approach 1 took about 2 hours to produce the potential plot, whereas, on the other hand, Approach 2 took only about 7 minutes.

3.5 Drift Lines

The drift lines of electrons were also plotted when a muon of momentum 2×10^9eV/c passed above the Thick-GEM hole. The muon started at coordinate (0.0, 0.0, 0.09) cm. The plots obtained are shown in Figures 3.6a and 3.6b.



(a) (b)

Figure 3.6: Drift lines of electrons produced after a Muon passes above a Thick-GEM hole

Chapter 4

Results and Conclusion

During the entirety of the semester, fundamentals of Gaseous radiation detectors were learned. The skills pertaining to constructing and assembling a Thick-GEM detector from scratch was learned. We have successfully constructed and assembled most parts of the detector to run and test it, the results of which have been shown in Chapter 2 pictorially. All the components of the detector were made in India. The GEM foils were manufactured by MIcropack, Bangalore. Although we weren't able to fully execute all the plans we had due to COVID-19.

We have been able to successfully integrate COMSOL Multiphysics® data into Garfield++ for the Thick-GEM detector. We started with simulating a single hole and have been able to correlate the potential contour plots obtained on Garfield++ with that of COMSOL Multiphysics®, which imply the data importing from COMSOL Multiphysics® into Garfield++ was a success. We then tested with another data structure of storing the data to help make the process of simulation faster, namely through the kd-binary tree. We were able to reduce the run time of producing the potential plots from 2 hours (using the brute force searching) to 7 minutes (using the kd-tree searching algorithm). We also simulated the drift lines of electrons when a muon passes through the detector and obtained the respective results as shown Section in 3.4. Although owing to computational needs, only a single hole of the detectors was simulated and not the entire detector. The signal calculation took too long to run, and is something we were not able to present, and looking for even faster algorithms is one way to tackle the problem.

Chapter 5

Outlook

The success of constructing almost everything needed to make the detector work was promising, although the plan was to make the required electrical connections, run gas leak checks and test the detector in the weeks that were to have remained. Due to unfortunate conditions, the project is still on a standstill and we are hopeful to finish it once things settle down.

Looking for faster ways to obtain the results for signal calculation is another problem that needs to be tackled. And in doing so, one could hopefully be able to calculate the signal generated and then simulate the entire detector in much less time and perform further studies as required.

References

[1] Meek J. M. and Cragg J.D. *Electrical Breakdown Of Gases*. The International Series of Monographs on Physics. Oxford (Clarendon Press), 1953.

[2] Sumanya Sekhar Sahoo. Study of gaseous detectors for high energy physics experiments, National Institute of Science Education and Research, 2015.

[3] F. Sauli. *Gaseous Radiation Detectors: Fundamentals and Applications*. Cambridge Monographs on Particle Physics, Nuclear Physics and Cosmology. Cambridge University Press, 2014. ISBN: 9781107043015.

[4] Maxim Titov. Gaseous detectors: recent developments and applications. *arXiv preprint arXiv:1008.3736*, 2010.

[5] R. Chechik, A. Breskin, C. Shalem, and D. Mörmann. Thick gem-like hole multipliers: properties and possible applications. *ucl. Instrum. Meth. A*, 535(1):303–308, 2004. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2004.07.138. URL: http://www.sciencedirect.com/science/article/pii/S0168900204016663. Proceedings of the 10th International Vienna Conference on Instrumentation.

[6] Zheng-Hua An et al. Experimental study on the performance of a single-THGEM gas detector. *Chin. Phys. C*, 34:83–87, 2010. DOI: 10.1088/1674-1137/34/1/015.

[7] S Bressler, L Arazi, L Moleri, M Pitt, A Rubin, and A Breskin. Recent advances with thgem detectors. *Journal of Instrumentation*, 8(12):C12012–C12012, December 2013. ISSN: 1748-0221. DOI: 10.1088/1748-0221/8/12/c12012. URL: http://dx.doi.org/10.1088/1748-0221/8/12/C12012.

[8] H. Schindler. Garfield++. http://garfieldpp.web.cern.ch/garfieldpp/, 2016.

[9] R. Brun and F. Rademakers. ROOT: An object oriented data analysis framework. *Nucl. Instrum. Meth. A*, 389:81–86, 1997. M. Werlen and D. Perret-Gallix, editors. DOI: 10.1016/S0168-9002(97)00048-X.

[10] Thiago Carli and B. Koblitz. A multi-variate discrimination technique based on range-searching. *ucl. Instrum. Meth. A*, 501:576–588, April 2003. DOI: 10.1016/S0168-9002(03)00376-0.

Appendix A

The Code for Generating the ROOT File

Listing A.1: ROOT File generation

```
#include <fstream>
#include <iostream>
#include <iomanip>
#include <cmath>
#include<TKDTree.h>

//This code essentially reads the raw data produced by COMSOL Multiphysics and thus stores the data in ROOT Trees

void read()
{
    int i=0;
    double x,y,z,v,ex,ey,ez,e;
    double min=10000.0, max=0.0;
    TFile file("tfiledata.root", "recreate");
    TTree t1("t1", "Field_Data");
    //x,y,z coordinates, the potential value and the electric field values
    t1.Branch("x", &x, "x/D");
    t1.Branch("y", &y, "y/D");
    t1.Branch("z", &z, "z/D");
    t1.Branch("v", &v, "v/D");
    t1.Branch("ex", &ex, "ex/D");
    t1.Branch("ey", &ey, "ey/D");
    t1.Branch("ez", &ez, "ez/D");
```

```
24      //fielddata is the COMSOL file that contains the required
           ↪ data
25      ifstream
           ↪ infile("/home/danush/garfieldpp-master/Thgem/fielddata.csv");
26      infile.seekg(0, ios::beg);
27      while(!infile.eof())
28      {
29          infile>>x;
30          infile>>y;
31          infile>>z;
32          infile>>v;
33          infile>>ex;
34          infile>>ey;
35          infile>>ez;
36          //We divide the coordinates by a factor of 10 because
               ↪ Garfield's default unit is in cm and COMSOL
               ↪ coordinates are in mm
37          x=x/10;
38          y=y/10;
39          z=z/10;
40          e=sqrt((ex*ex) + (ey*ey) + (ez*ez));
41          t1.Fill();
42          if(max<e)
43          {
44              max=e;
45          }
46          if(e<min)
47          {
48              min=e;
49          }
50          i++;
51      }
52      t1.Write();
53      cout<<"Minimum E = "<<min<<endl<<"Maximum E = "<<max;
54      infile.close();
55  }
```

Appendix B

The Code for Creating the Gas File

Listing B.1: Gas File

```
1
2  #include <iostream>
3  //#include <TCanvas.h>
4  #include "TROOT.h"
5  #include "TApplication.h"
6
7  #include "Garfield/MediumMagboltz.hh"
8  #include "Garfield/FundamentalConstants.hh"
9
10 using namespace Garfield;
11
12 int main(int argc, char * argv[])
13 {
14   // TApplication app("app", &argc, argv);
15   const double pressure = 760.;
16   const double temperature = 293.15;
17
18   // Setup the gas.
19   MediumMagboltz* gas = new MediumMagboltz();
20   gas->SetTemperature(temperature);
21   gas->SetPressure(pressure);
22   gas->SetComposition("ar", 80., "co2", 20.);
23
24   // Set the field range to be covered by the gas table.
25   const int nFields = 5;
```

```
26    const double emin = 0.0000001;  //Minimum value of the
          ↪ electric field (in V/cm) present in your setup
27    const double emax = 300000.;    //Maximum value of the
          ↪ electric field
28    // Flag to request logarithmic spacing.
29    const bool useLog = true;
30    gas->SetFieldGrid(emin, emax, nFields, useLog);
31
32    const int ncoll = 10;
33    // Switch on debugging to print the Magboltz output.
34    gas->EnableDebugging();
35    // Run Magboltz to generate the gas table.
36    gas->GenerateGasTable(ncoll);
37    gas->DisableDebugging();
38    // Save the table.
39    gas->WriteGasFile("ar_80_co2_20.gas");
40
41    // app.Run(kTRUE);
42  }
```

Appendix C

The Garfield++ Simulation Code

Listing C.1: Garfield++ Code

```
1
2  #include <iostream>
3  #include <fstream>
4  #include <random>
5
6  #include <TApplication.h>
7  #include <TCanvas.h>
8  #include <TH1F.h>
9  #include "Garfield/Plotting.hh"
10 #include "Garfield/ViewGeometry.hh"
11 #include "Garfield/ViewField.hh"
12 #include "Garfield/ViewDrift.hh"
13 #include "Garfield/ViewSignal.hh"
14 #include "Garfield/ViewFEMesh.hh"
15
16 #include <TSystem.h>
17 #include <TROOT.h>
18 #include <TGeoManager.h>
19 #include <TGeoBBox.h>
20 #include <TGeoTube.h>
21 #include <TGeoCompositeShape.h>
22 #include "Garfield/ComponentUser.hh"
23 #include "Garfield/GeometryRoot.hh"
24 #include <TStyle.h>
25 #include<TKDTree.h>
26
```

```
27 #include "Garfield/MediumMagboltz.hh"
28 #include "Garfield/ViewMedium.hh"
29 #include "Garfield/ComponentComsol.hh"
30 #include "Garfield/ComponentBase.hh"
31 #include "Garfield/SolidBox.hh"
32 #include "Garfield/Track.hh"
33 #include "Garfield/TrackHeed.hh"
34 #include "Garfield/DriftLineRKF.hh"
35 #include "Garfield/AvalancheMicroscopic.hh"
36 #include "Garfield/AvalancheMC.hh"
37 #include "Garfield/Sensor.hh"
38 #include <TFile.h>
39 #include <TTree.h>
40 #include "Garfield/Random.hh"
41
42 #include <iomanip>
43 #include <sstream>
44
45 //I am grateful to Dr. Abhik Jash and Dr. Varchaswi Kashyap
     ↪ who have contributed immensely in helping me complete
     ↪ this code.
46
47 using namespace std;
48 using namespace Garfield;
49
50 const int k = 3;
51 int length;
52
53 void Field_map(const double, const double, const double,
     ↪ double&, double&, double&);
54 void WtField_map(const double, const double, const double,
     ↪ double&, double&, double&, const string);
55 void Potential_map(const double, const double, const double,
     ↪ double&);
56
57
58 struct Node *root=NULL;
59 //We define a global KDTree so that it is used freely by
     ↪ functions while searching for the nearest point
60 TKDTreeID *kdtree = new TKDTreeID();
61
```

```
62 int main(int arg, char* argv[]){
63   TApplication app("app", &arg, argv);
64   plottingEngine.SetDefaultStyle();
65 
66   //Variables
67   //Default distance unit in Garfield++ is in cm
68   const float x0=0., y0=0., z0=0.;
69   const float x_cen=(0.05*9)/2, y_cen = (0.0866*5)/2;
       ↪ //Centre of the Thgem foils
70   const float x_len = 0.03, y_len = 0.03, cu_height = 0.0018,
       ↪ pcb_height = 0.025;   //Heights of layers
71   const float pcb_hole = 0.01, top_hole = 0.012;   //Hole Radii
72 
73   gSystem->Load("libGeom");
74   TGeoManager *geom = new TGeoManager();
75 
76   //Define materials
77   TGeoMaterial *mat_vacuum = new TGeoMaterial("Vacuum", 0, 0,
       ↪ 0); // Define a new material (name, atomic weight,
       ↪ atomic number, density in gm/cm^3)
78   TGeoMaterial *mat_Cu = new TGeoMaterial("Cu", 63.55, 29,
       ↪ 8.96);
79   TGeoMaterial *mat_bakelite = new TGeoMaterial("Bakelite",
       ↪ 63.55, 29, 1.3);    // Replace molar weight and atomic
       ↪ number by proper values
80   TGeoMaterial *mat_graphite = new TGeoMaterial("Graphite",
       ↪ 12.01, 6.0, 0.86);
81   TGeoMaterial *mat_mylar = new TGeoMaterial("Mylar", 192.16,
       ↪ 29.0, 1.39);
82   TGeoMaterial *matAl = new TGeoMaterial("Al", 26.98,13,2.7);
83 
84   // Define medium
85   TGeoMedium *vacuum = new TGeoMedium("Vacuum", 1, mat_vacuum);
86   TGeoMedium *bakelite = new TGeoMedium("Bakelite", 3,
       ↪ mat_bakelite);
87   TGeoMedium *graphite = new TGeoMedium("Graphite", 4,
       ↪ mat_graphite);
88   TGeoMedium *mylar = new TGeoMedium("Mylar", 5, mat_mylar);
89   TGeoMedium *copper = new TGeoMedium("Copper", 6, mat_Cu);
90   TGeoMedium *Al = new TGeoMedium("Root Material",2, matAl);
91 
```

```
92  // Setup the gas
93  MediumMagboltz* gas = new MediumMagboltz();
94  const double pressure = 760.;
95  const double temperature = 293.15;
96  gas->SetTemperature(temperature);
97  gas->SetPressure(pressure);
98  // Set the gas mixture.
99  gas->SetComposition("ar", 80., "co2", 20.);
100 gas->LoadGasFile("ar_80_co2_20.gas");
101 // Penning transfer probability.
102 const double rPenning = 0.57;
103 // Mean distance from the point of excitation.
104 const double lambdaPenning = 0.;
105 gas->EnablePenningTransfer(rPenning, lambdaPenning, "ar");
106 // Read the ion mobility table from file.
107 const std::string path = getenv("GARFIELD\_HOME");
108 gas->LoadIonMobility(path +
      ↪ "/Data/IonMobility\_Ar+\_Ar.txt");
109
110
111 //Define Translations
112 TGeoTranslation *etch1 = new TGeoTranslation(x0, y0, z0
      ↪ -cu_height/2);
113 TGeoTranslation *pcb1 = new TGeoTranslation(x0, y0, z0
      ↪ -pcb_height/2 - cu_height);
114 TGeoTranslation *etch2 = new TGeoTranslation(x0, y0, z0
      ↪ -pcb_height - cu_height - cu_height/2);
115 TGeoTranslation *drift_h = new TGeoTranslation(x0, y0, z0 +
      ↪ 0.4 + cu_height/2);
116 TGeoTranslation *readout_h = new TGeoTranslation(x0, y0, z0
      ↪ -pcb_height - cu_height - cu_height/2 - 0.2);
117
118 //Define Volumes
119 TGeoVolume *world = geom->MakeBox("World", vacuum,  0.5,
      ↪ 0.5, (pcb_height + 2*cu_height + 0.5 + 0.04)/2);
      ↪ //NOTE GAS
120
121 geom->SetTopVolume(world);
122 TGeoBBox *topbox = new TGeoBBox("t1", x_len/2, y_len/2,
      ↪ cu_height/2);
```

```
123   TGeoBBox *drif = new TGeoBBox("de", x_len/2, y_len/2,
        ↪ cu_height/2);
124   TGeoBBox *read = new TGeoBBox("re", x_len/2, y_len/2,
        ↪ cu_height/2);
125   TGeoBBox *midbox = new TGeoBBox("m1", x_len/2, y_len/2,
        ↪ pcb_height/2);
126   TGeoTube *cyl = new TGeoTube("c1", 0., top_hole,
        ↪ cu_height/2);
127   TGeoTube *cyl2 = new TGeoTube("c2", 0., pcb_hole,
        ↪ pcb_height/2);
128   TGeoCompositeShape *cs3 = new TGeoCompositeShape("cs3", "m1
        ↪ - c2");
129   TGeoCompositeShape *cs = new TGeoCompositeShape("cs", "t1 -
        ↪ c1");
130   TGeoCompositeShape *cs2 = new TGeoCompositeShape("cs2", "t1
        ↪ - c1");
131   TGeoVolume *top = new TGeoVolume("TOP", cs, copper);
132   TGeoVolume *bottom = new TGeoVolume("BOT", cs2, copper);
133   TGeoVolume *pcb = new TGeoVolume("PCB", cs3, bakelite);
134   TGeoVolume *drift = new TGeoVolume("Drift", drif);
135   TGeoVolume *readout = new TGeoVolume("Readout", read);
136   world->AddNode(top, 6, etch1);
137   world->AddNode(bottom, 6, etch2);
138   world->AddNode(pcb, 2, pcb1);
139   world->AddNode(drift, 6, drift_h);
140   world->AddNode(readout, 6, readout_h, "read_0");
141
142   //Visibility
143   world->SetTransparency(0);
144   top->SetLineColor(kYellow);
145   top->SetTransparency(60);
146   pcb->SetLineColor(kRed);
147   pcb->SetTransparency(60);
148   bottom->SetLineColor(kBlue);
149   bottom->SetTransparency(60);
150   //Close Geometry
151
152   geom->CloseGeometry();
153   geom->CheckOverlaps(0.0001);
154   geom->PrintOverlaps();
155   geom->Export("THGEM_1.root");
```

```
156
157   // Create the Garfield geometry.
158   GeometryRoot* geo = new GeometryRoot();
159   // Pass the pointer to the TGeoManager.
160   geo->SetGeometry(geom);
161   // Associate the ROOT medium with the Garfield medium.
162   geo->SetMedium("Vacuum", gas);
163
164   ComponentUser* comp = new ComponentUser();
165   comp->SetElectricField(*Field_map);
166   comp->SetPotential(*Potential_map);
167   comp->SetWeightingField(*WtField_map);
168   comp->SetGeometry(geo);
169
170   //Make a sensor
171   Sensor* sensor = new Sensor();
172   sensor->AddComponent(comp);
173
174   sensor->SetArea(x0-x_len/2-0.01, y0-y_len/2-0.01, z0
        ↪ -pcb_height - cu_height - cu_height-0.01,
        ↪ x0+x_len/2+0.01, y0+y_len/2+0.01, z0+0.01);
175   const double tMin = 0.;   // in ns
176   const double tMax = 40.;  // in ns
177   const double tStep = 0.01;    // in ns
178   const int nTimeBins = 4000;//int((tMax - tMin) / tStep);
179   sensor->SetTimeWindow(tMin, tStep, nTimeBins);
180
181   TStyle *myStyle = new TStyle("myStyle","Manual styles for 3D
        ↪ canvas");
182   myStyle->SetCanvasColor(0);
183   myStyle->SetLabelSize(0.03,"xyz");
184   myStyle->SetTitleOffset(1.2);
185   myStyle->SetTitleX(1);
186   myStyle->SetTitleY(1);
187   myStyle->SetTitleW(1);
188   gStyle->SetAxisColor(kBlack,"X");
189   gStyle->SetAxisColor(kBlue,"Y");
190   gStyle->SetAxisColor(kRed,"Z");
191
192   gStyle->SetLabelColor(kBlack,"X");
193   gStyle->SetLabelColor(kBlue,"Y");
```

```
194   gStyle->SetLabelColor(kRed,"Z");
195
196   gROOT->SetStyle("myStyle");
197
198   sensor->AddElectrode(comp, "read_0");
199
200   //I like to thank Aman Upadhyay and Vijay Iyer who have
        ↪ helped me with ROOT and KDTrees
201
202   //This function generates a a KDTree which was used as
        ↪ explained in Section 3.2
203   //tfiledata is the ROOT file, which will be used to create
        ↪ the KDTree
204   TFile
        ↪ f("/home/danush/garfieldpp-master/Thgem/tfiledata.root");
205   TTree *t;
206   f.GetObject("t1", t);
207   //x,y,z coordinates, the potential value and the electric
        ↪ field values
208   double x2,y2,z2,v2,ex2,ey2,ez2;
209   //x2,y2,z2 are point that run arbitrarily through the data
210   t->SetBranchAddress("x",&x2);
211   t->SetBranchAddress("y",&y2);
212   t->SetBranchAddress("z",&z2);
213   t->SetBranchAddress("v",&v2);
214   t->SetBranchAddress("ex",&ex2);
215   t->SetBranchAddress("ey",&ey2);
216   t->SetBranchAddress("ez",&ez2);
217   length = t->GetEntries();
218   Double_t **data = new Double_t*[3];
219   data[0] = new Double_t[length];
220   data[1] = new Double_t[length];
221   data[2] = new Double_t[length];
222   for(int i=0;i<length;i++)
223   {
224     t->GetEntry(i);
225     data[0][i]=x2;
226     data[1][i]=y2;
227     data[2][i]=z2;
228   }
229   kdtree->SetData(length, k, 1, data);
```

```
230   kdtree->Build();
231
232   //---------------------------------------TASKS-------------------------
233   int draw_geom = 0;
234   int draw_field = 0;
235   int draw_drift = 1;
236   int signal_cal = 0;
237   int draw_signal = 1;
238   if(signal_cal==0)
239     draw_signal = 0;
240
241
242   if(draw_geom==1)
243     {
244       TCanvas *c_geom = new TCanvas("c_geom", "c_geom", 200,
          ↪ 200, 700, 700);
245       geom->SetVisLevel(10);
246       world->Draw("ogl");
247     }
248
249   if(draw_field==1)
250     {
251     ViewField *fieldView = new ViewField();
252     TCanvas *c_field = new TCanvas("c_field", "c_field", 200,
          ↪ 200, 700, 700);
253     fieldView->SetComponent(comp);
254     fieldView->SetSensor(sensor);
255     fieldView->SetPlane(0., -1., 0, 0., 0., 0.);
256     fieldView->SetArea(x0-x_len/2-0.01, z0 -pcb_height -
          ↪ cu_height - cu_height-0.01, x0+x_len/2+0.01,
          ↪ z0+0.01);
257     gStyle->SetPalette(55);  //kRainBow=55
258     fieldView->SetCanvas(c_field);
259     fieldView->PlotContour("v");
260     }
261     else
262     {
263     cout<<"\n Field not drawn, as requested"<<endl;
264     }
265
```

```
266     //For simulating the electron avalanche we use the class
          ↪ AvalancheMicroscopic
267     AvalancheMicroscopic* aval = new AvalancheMicroscopic();
268     const int aval_size = 200;
269     aval->SetSensor(sensor);
270
271     //Switch on signal calculation.
272     aval->EnableSignalCalculation();
273     aval->SetTimeWindow(tMin,tMax );
274     //aval->EnableAvalancheSizeLimit(aval_size);
275     aval->EnableDriftLines();
276
277     //Setup HEED
278     TrackHeed* track = new TrackHeed();
279     track->SetParticle("mu-");
280     track->SetSensor(sensor);
281     track->EnableElectricField();
282     float P_muon = 2.e9;    // 2 GeV muons
283     track->SetMomentum(P_muon); // in eV/c
284
285     //The initial impact position of the incoming ionising
          ↪ track
286     float track_x = x0;
287     float track_y = y0;
288     float track_z = 0.009;  // The starting point of track
          ↪ must be from an ionizing medium
289
290     //Momentum direction of incoming track
291     float track_dx = 0.0;
292     float track_dy = 1.0;
293     float track_dz = -0.08;
294
295     //Cluster info
296     double xcls, ycls, zcls, tcls, ecls, extra;
297     int ncls = 0; // number of electrons in cluster
298
299     //Electron info
300     double xele, yele, zele, tele, eele, dxele, dyele, dzele;
301
302     //Electron start and endpoints, momentum direction and
          ↪ status
```

```
303     float x0ele, y0ele, z0ele, t0ele, e0ele;// start point
304     float x1ele, y1ele, z1ele, t1ele, e1ele;// end point
305     float dx1ele, dy1ele, dz1ele; // momentum direction
306     int status1ele; // status
307     int clust_id=0, elect_primary, elect_total;
308     float charge, muon_energy, muon_theta, muon_phi,
          ↪ E_deposited, clust_density, P_mu, theta_mu, phi_mu,
          ↪ gain;
309
310
311   if(draw_drift==1)
312   {
313   cout<<"Drift of electrons : Calculation in progress"<<endl;
314     TCanvas *c_drift = new TCanvas("c_drift", "c_drift", 150,
          ↪ 1000, 800, 600);
315     c_drift->Range(-1,-1,1,1);
316     c_drift->SetTheta(15);
317     c_drift->SetPhi(-70);
318
319     ViewDrift* driftView = new ViewDrift();
320     driftView->SetCanvas(c_drift);
321     driftView->SetArea(x0-x_len/2-0.01, y0-y_len/2-0.01, z0
          ↪ -pcb_height - 2*cu_height -0.01, x0+x_len/2+0.01,
          ↪ y0+y_len/2+0.01, z0+0.01);
322     driftView->SetClusterMarkerSize(0.2);
323     driftView->SetCollisionMarkerSize(0.5);
324
325     track->EnablePlotting(driftView);
326     aval->EnablePlotting(driftView);
327
328     DriftLineRKF* driftline_e = new DriftLineRKF();
329     driftline_e->SetSensor(sensor);
330     driftline_e->EnablePlotting(driftView);
331
332   // Now plot the drift lines
333     track->NewTrack(track_x, track_y, track_z, tMin, track_dx,
          ↪ track_dy, track_dz);
334     driftView->Plot(0,1);
335
336     bool clust_present=0;
337     do
```

```
338         {
339         xcls=ycls=zcls=tcls=ncls=ecls=extra=0;
340         clust_present=track->GetCluster(xcls, ycls, zcls,
               ↪ tcls, ncls, ecls, extra);
341         for(int j = 1; j <= ncls; j++)
342         {
343           xele=yele=zele=tele=eele=dxele=dyele=dzele=0;
344         track->GetElectron(j-1, xele, yele, zele, tele, eele,
               ↪ dxele, dyele, dzele);
345         aval->AvalancheElectron(xele, yele, zele, tele, eele,
               ↪ dxele, dyele, dzele);
346         }
347         clust_id++;
348   //     cout<<"Event = "<<iEvent+1<<": "<<"Cluster_id =
          ↪ "<<clust_id<<endl;
349         }
350     while(clust_present!=0);
351
352     driftView->Plot(0,1);
353     c_drift->Modified();
354
355   cout<<"SUCCESS--------------------";
356   }
357
358
359   if(signal_cal==1)
360   {
361     cout<<"Signal : Calculation in progress"<<endl;
362     int nEvent=1;
363     fstream outfile, outfile_Q;
364     string outfilename_charge = "charge_Thgem_" +
          ↪ to_string(nEvent) + "Events" + ".dat";
365       outfile_Q.open(outfilename_charge, ios::out);
366       outfile_Q<<"Event ID\tCharge (pC)"<<endl;
367       outfile_Q.close();
368       string rootFileName = "data_Thgem_Ez_" +
            ↪ to_string(nEvent) + "Events" + ".root";
369       TFile *file1 = new
            ↪ TFile(rootFileName.c_str(),"RECREATE","My root
            ↪ tree file");
```

```
370       TH1F *hist_charge = new TH1F("hist_charge",
              ↪ "hist_charge", 1000,0,5000);
371     hist_charge->GetXaxis()->SetTitle("Induced charge");
372     hist_charge->GetYaxis()->SetTitle("Counts");
373       int id_evt, nClust;
374     TTree *tr_event = new TTree("event_info", "Information of
          ↪ various parameters per event");
375     tr_event->Branch("nCluster", &nClust, "nClust/I");
376     tr_event->Branch("clustDensity",&clust_density,
          ↪ "clust_density/F");
377       tr_event->Branch("eDeposite_eV",&E_deposited,
            ↪ "E_deposited/F");
378       tr_event->Branch("nElectron_primary",&elect_primary,
            ↪ "elect_primary/I");
379       tr_event->Branch("nElectron_total",&elect_total,
            ↪ "elect_total/I");
380       tr_event->Branch("detector_gain",&gain, "gain/F");
381       tr_event->Branch("inducedCharge",&charge, "charge/F");
382     double track_x = x0;
383     double track_y = y0;
384     double track_z = 0.199;
385
386     for(Int_t i=0; i<nEvent; i++)
387     {
388       charge=0, elect_primary=0, elect_total=0, gain=0;
389       P_mu = 2e9;
390       theta_mu = 0;
391       phi_mu =  0;
392       double track_dx = 0;
393       double track_dy = 0;
394       double track_dz = -0.1;
395
396       track->SetMomentum(P_mu);
397       track->NewTrack(track_x, track_y, track_z, tMin,
            ↪ track_dx, track_dy, track_dz);
398       bool clust_present=0;
399         clust_id=0, clust_density = 0, E_deposited = 0;
400
401       do
402       {
```

```
403           clust_present = track->GetCluster(xcls, ycls, zcls,
                  ↪ tcls, ncls, ecls, extra);
404           E_deposited = E_deposited + ecls;
405           for(int j=0; j<ncls; j++)
406           {
407             //Retrieve details about the electrons in the
                  ↪ present cluster using GetElectron
408             track->GetElectron(j, xele, yele, zele, tele, eele,
                  ↪ dxele, dyele, dzele);
409             //The calculation of an avalanche initiated by an
                  ↪ electron
410             aval->AvalancheElectron(xele, yele, zele, tele,
                  ↪ eele, dxele, dyele, dzele);
411             elect_primary++;
412           }
413           clust_id++;
414         }while(clust_present!=0);
415         clust_density = track->GetClusterDensity();
416
417           double Qstrip1 = 0.0;
418           float time[nTimeBins], current[nTimeBins];
419
420           for(int i = 0; i < nTimeBins; i++)
421         {
422           time[i] = i*tStep;
423           current[i] = sensor->GetElectronSignal("read_0",
                  ↪ i)*1000;    // Current*1000 to get it in nA
424           Qstrip1 += fabs(sensor->GetElectronSignal("read_0",
                  ↪ i))*i;  // in femtoCouloumb
425           }
426           outfile_Q.open(outfilename_charge, ios::out |
                  ↪ ios::app);
427           outfile_Q<<i<<"\t"<<Qstrip1/1000<<endl;
428           outfile_Q.close();
429
430           string graphName = string("signal") + "_" +
                  ↪ to_string(i);
431           TGraph *gr = new TGraph(nTimeBins,time,current);
432           gr->GetXaxis()->SetTitle("Time (ns)");
433           gr->GetYaxis()->SetTitle("Current (nA)");
434           gr->GetXaxis()->CenterTitle();
```

```
435         gr->GetYaxis()->CenterTitle();
436         gr->SetLineStyle(0);
437         gr->SetMarkerStyle(0);
438         gr->SetMarkerColor(1);
439 
440         nClust = clust_id-1;
441         charge = Qstrip1;
442         gain = (float)elect_total/elect_primary;
443         tr_event->Fill();
444         gr->Write(graphName.c_str());
445         hist_charge->Fill(charge);
446         sensor->ClearSignal();  // Reset signals and free the
              ↪ sensor
447         cout<<"Event = "<<i+1<<" : Done"<<endl;
448     }
449       file1->Write();
450       file1->Close();
451     cout<<"Signal calculations are done."<<endl;
452   }
453   else
454   {
455     cout<<"Signal calculations not done, as requested."<<endl;
456   }
457 
458 
459   // Draw the signals
460     if(draw_signal==1)
461     {
462     cout<<"Drift of electrons : Calculation in progress"<<endl;
463     TCanvas* c_signal = new TCanvas("c_signal", "c_signal",
          ↪ 1000, 20, 1000, 760);
464     ViewSignal* signalViewX0 = new ViewSignal();
465     signalViewX0->SetSensor(sensor);
466     signalViewX0->SetCanvas((TCanvas*)c_signal->cd(1));
467     signalViewX0->PlotSignal("read_0",0,1,0);
468     c_signal->SaveAs("signalShape.png");
469     }
470     else
471     {
472     cout<<"\n Signal not drawn, as requested"<<endl;
473     }
```

```
474
475
476   app.Run(kTRUE);
477 }
478
479 //--------------------------------Function
      ↪ definitions--------------------------------
480
481 //Note that the following definitions have been made according
      ↪ to what was explained in Section 3.2 (Approach 2))
482 void Field_map(const double x1, const double y1, const double
      ↪ z1, double &ex, double &ey, double &ez)
483 {
484   Double_t *trial = new Double_t[3];
485   trial[0] = x1;
486   trial[1] = y1;
487   trial[2] = z1;
488   int *index = new int;
489   Double_t *d = new Double_t[3];
490   kdtree->FindNearestNeighbors(trial, 1, index, d);
491   TFile
        ↪ f("/home/danush/garfieldpp-master/Thgem/tfiledata.root");
492   TTree *t;
493   f.GetObject("t1",t);
494   double x2,y2,z2,v2,ex2,ey2,ez2;
495   //x2,y2,z2 are point that run arbitrarily through the data
496   t->SetBranchAddress("x",&x2);
497   t->SetBranchAddress("y",&y2);
498   t->SetBranchAddress("z",&z2);
499   t->SetBranchAddress("v",&v2);
500   t->SetBranchAddress("ex",&ex2);
501   t->SetBranchAddress("ey",&ey2);
502   t->SetBranchAddress("ez",&ez2);
503   t->GetEntry(*index);
504   ex=ex2;
505   ey=ey2;
506   ez=ez2;
507   f.Close();
508 }
509
```

```
510 void Potential_map(const double x1, const double y1, const
      ↪ double z1, double& V)
511 {
512   Double_t *trial = new Double_t[3];
513   trial[0] = x1;
514   trial[1] = y1;
515   trial[2] = z1;
516   int *index = new int;
517   Double_t *d = new Double_t[3];
518   kdtree->FindNearestNeighbors(trial, 1, index, d);
519   TFile
        ↪ f("/home/danush/garfieldpp-master/Thgem/tfiledata.root");
520   TTree *t;
521   f.GetObject("t1",t);
522   double x2,y2,z2,v2,ex2,ey2,ez2;
523   //x2,y2,z2 are point that run arbitrarily through the data
524   t->SetBranchAddress("x",&x2);
525   t->SetBranchAddress("y",&y2);
526   t->SetBranchAddress("z",&z2);
527   t->SetBranchAddress("v",&v2);
528   t->SetBranchAddress("ex",&ex2);
529   t->SetBranchAddress("ey",&ey2);
530   t->SetBranchAddress("ez",&ez2);
531   t->GetEntry(*index);
532   V=v2;
533   f.Close();
534 }
535
536 void WtField_map(const double x1, const double y1, const
      ↪ double z1, double& wx, double& wy, double& wz, const
      ↪ string strip)
537 {
538   Double_t *trial = new Double_t[3];
539   trial[0] = x1;
540   trial[1] = y1;
541   trial[2] = z1;
542   int *index = new int;
543   Double_t *d = new Double_t[3];
544   kdtree->FindNearestNeighbors(trial, 1, index, d);
545   //Since weighting field values is from a second file from
        ↪ COMSOL Multiphysics, we generate another ROOT file for
```

```
    ↪ weighting field values
  //The process of generating the file is same as how we did
    ↪ it for tfiledata
  TFile
    ↪ f("/home/danush/garfieldpp-master/Thgem/tfiledata2.root");
  TTree *t;
  f.GetObject("t1",t);
  double x2,y2,z2,v2,ex2,ey2,ez2;
  //x2,y2,z2 are point that run arbitrarily through the data
  t->SetBranchAddress("x",&x2);
  t->SetBranchAddress("y",&y2);
  t->SetBranchAddress("z",&z2);
  t->SetBranchAddress("v",&v2);
  t->SetBranchAddress("ex",&ex2);
  t->SetBranchAddress("ey",&ey2);
  t->SetBranchAddress("ez",&ez2);
  t->GetEntry(*index);
  wx=ex2;
  wy=ey2;
  wz=ez2;
  f.Close();
}

//If we are to follow Approach 1 (Section 3.1), then the above
  ↪ functions need to be replaced with the ones below.
/*
void Field_map(const double x1, const double y1, const double
  ↪ z1, double &ex, double &ey, double &ez)
{
    TFile f("/home/danush/garfieldpp/Thgem/tfiledata.root");
    TTree *t;
    f.GetObject("t1",t);
    double x2,y2,z2,v2,ex2,ey2,ez2;
    //x2,y2,z2 are point that run arbitrarily through the data
    t->SetBranchAddress("x",&x2);
    t->SetBranchAddress("y",&y2);
    t->SetBranchAddress("z",&z2);
    t->SetBranchAddress("v",&v2);
    t->SetBranchAddress("ex",&ex2);
    t->SetBranchAddress("ey",&ey2);
    t->SetBranchAddress("ez",&ez2);
```

```
582     double min=100.0,d;
583     int loc=0;
584     for(int i=0; i<t->GetEntries();i++)
585     {
586         t->GetEntry(i);
587         d=sqrt(((x1-x2)*(x1-x2)) + ((y1-y2)*(y1-y2)) +
            ↪ ((z1-z2)*(z1-z2)));
588         if(d<=min)
589         {
590             min = d;
591             loc=i;
592 
593         }
594     }
595     t->GetEntry(loc);
596     ex=ex2;
597     ey=ey2;
598     ez=ez2;
599     f.Close();
600 }
601 
602 void Potential_map(const double x1, const double y1, const
    ↪ double z1, double& V)
603 {
604     TFile f("/home/danush/garfieldpp/Thgem/tfiledata.root");
605     TTree *t;
606     f.GetObject("t1",t);
607     double x2,y2,z2,v2,ex2,ey2,ez2;
608     //x2,y2,z2 are point that run arbitrarily through the data
609     t->SetBranchAddress("x",&x2);
610     t->SetBranchAddress("y",&y2);
611     t->SetBranchAddress("z",&z2);
612     t->SetBranchAddress("v",&v2);
613     t->SetBranchAddress("ex",&ex2);
614     t->SetBranchAddress("ey",&ey2);
615     t->SetBranchAddress("ez",&ez2);
616     double min=100.0,d;
617     int loc=0;
618     for(int i=0; i<t->GetEntries();i++)
619     {
620         t->GetEntry(i);
```

```
621         d=sqrt(((x1-x2)*(x1-x2)) + ((y1-y2)*(y1-y2)) +
               ↪ ((z1-z2)*(z1-z2)));
622         if(d<=min)
623         {
624             min = d;
625             loc=i;
626 
627         }
628     }
629     t->GetEntry(loc);
630     V=v2;
631     f.Close();
632 
633 }
634 
635 void WtField_map(const double x1, const double y1, const
       ↪ double z1, double& wx, double& wy, double& wz, const
       ↪ string strip)
636 {
637     //Since weighting field values is from a second file from
           ↪ COMSOL Multiphysics, we generate another ROOT file
           ↪ for weighting field values
638     //The process of generating the file is same as how we did
           ↪ it for tfiledata
639     TFile f("/home/danush/garfieldpp/Thgem/tfiledata2.root");
640     TTree *t;
641     f.GetObject("t1",t);
642     double x2,y2,z2,v2,ex2,ey2,ez2;
643     //x2,y2,z2 are point that run arbitrarily through the data
644     t->SetBranchAddress("x",&x2);
645     t->SetBranchAddress("y",&y2);
646     t->SetBranchAddress("z",&z2);
647     t->SetBranchAddress("v",&v2);
648     t->SetBranchAddress("ex",&ex2);
649     t->SetBranchAddress("ey",&ey2);
650     t->SetBranchAddress("ez",&ez2);
651     double min=100.0,d;
652     int loc=0;
653     for(int i=0; i<t->GetEntries();i++)
654     {
655         t->GetEntry(i);
```

```
656           d=sqrt(((x1-x2)*(x1-x2)) + ((y1-y2)*(y1-y2)) +
                 ↪ ((z1-z2)*(z1-z2)));
657           if(d<=min)
658           {
659                min = d;
660                loc=i;
661
662           }
663      }
664      t->GetEntry(loc);
665      wx=ex2;
666      wy=ey2;
667      wz=ez2;
668      f.Close();
669 }*/
```