Designing a Particle Detector for the Analysis of (p,2p) Reactions using the Geant4 and ROOT frameworks

Conner Grice

Department of Physics, University of York

May 2022

Abstract

A particle detector was designed to make use of the quasi-free (p,2p) knockout reaction in order to probe the observables of protons in the Carbon-12 nucleus and simulated using the Geant4 framework. This report will be beneficial in making design choices at R3B, at which a physical detector is being developed for similar purposes. The detector system is made up of two silicon cylindrical layers surrounded by a spherical silicon calorimeter. To investigate the proficiency of the detector, the missing mass reconstruction was calculated using the outgoing proton's 4-momentum vectors. Dimensions tested are the length of the cylinders, the gap between them and their thickness, as well as the thickness of the calorimeter. An optimal detector for this situation would be at least 80 cm long, have a gap of 3 cm, a layer thickness of 0.01 mm or less, and a calorimeter of 160 cm, giving a detector with an 85% efficiency.

Introduction

Quasi-free scattering is a tool that can be used to probe nuclear structure. This can be done using direct kinematics or inverse kinematics. The direct kinematics approach involves firing a particle beam at a target made up of the atom that is being investigated, then analysing the outgoing particles[1, 2]. While the inverse kinematics method takes an ion beam, consisting of the ion that is being investigated, and fires it at some target, normally liquid hydrogen, solid hydrogen or CH_2.

The inverse kinematic approach on the (p,2p) reaction, shown in figure 1, can be used to investigate short-lived nuclei. (p,pn) is another quasi-free reaction where a neutron is knocked out[3]. (p,pα) can also be called a quasi-free reaction because the data collected after firing a proton beam into a 12C atom, causing the knockout of a α particle resembles that of a free scattering event of a proton and an alpha particle[4].

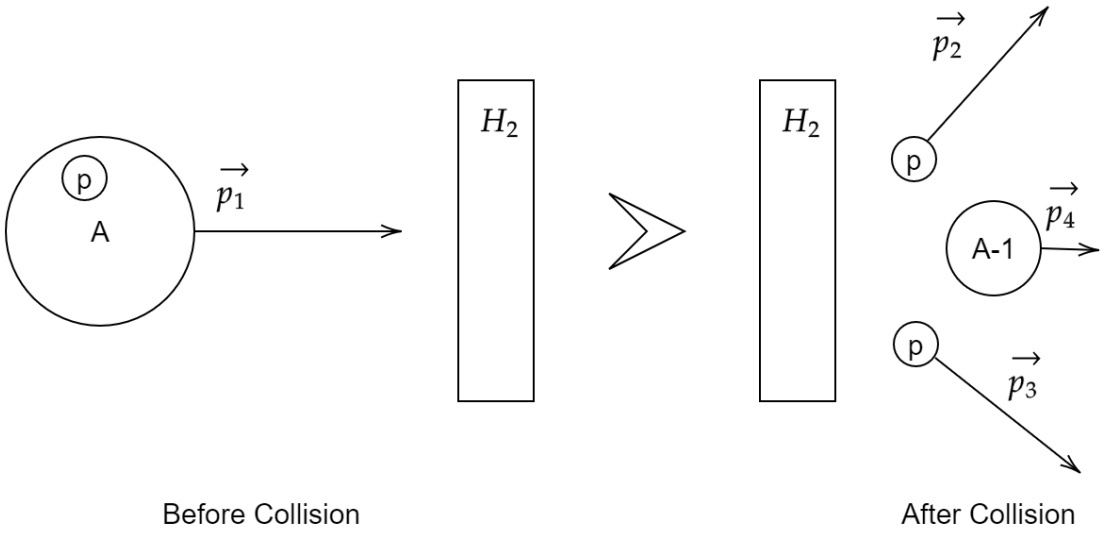


Figure 1: Schematic of an inverse kinematic (p,2p) reaction where an ion beam of atom A is incident on a liquid hydrogen target, causing a proton to be knocked out of atom A and a hydrogen atom.

The reason quasi-free reactions are significant is because they can be used to find the binding energy of nucleons within atoms. The nuclear angular momentum of the shell that the proton was knocked from can also be deduced due to the variations in the momentum distribution width. This is done by calculating the missing mass after analysing the outgoing protons, and their momentum[5]. Finally, another use is for the study of the single-particle strength of a nucleon, this is done by analysing the cross-section of the reaction. One can study the reduction of the spectroscopic reduction factor in different isotopes [6, 7, 8], due to the use of the mean-field, independent particle model.

Calculations to find the missing mass during a single reaction utilises the outgoing particles 4-momentum, p^{μ}. 4-momentum is a vector containing the energy and momentum information of the particle, taking the form:

$$p^{\mu} = \left(\frac{E}{c}, p_i\right), i = 1,2,3 \tag{1}$$

Where, E is the relativistic energy, c is the speed of light and p_i is the projection of the 3-momentum vector. The invariant mass, m, is the mass found from the Lorentz invariant quantity of the momentum, $p_{\mu}p^{\mu} = m^2c^2$. However, when investigating quasi-free scattering, the invariant mass can be seen as the energy that was used to bind the nucleon inside the nucleus. It is known that:

$$\begin{aligned} E &= E_k + mc^2 \\ E^2 &= p^2c^2 + m^2c^4 \end{aligned} \tag{2}$$

Where, E_k is its kinetic energy and p is the momentum magnitude. A mass defect is present due to the binding energy between nucleons. By knowing the initial state of the ion beam, and the 4-moment of the outgoing particles, the difference can be found, allowing for the mass defect due to that singular proton to be calculated. This calculation is carried out in multiple experiments in order to produce the missing mass reconstruction/distribution.

In this project, a particle detector was designed and coded from nothing to investigate a simulated reaction between a beam of Carbon-12 and a liquid hydrogen target to estimate the binding energy of the valence proton within the Carbon-12 nucleus. Such detectors are being developed, and will be used in the R3B experiments[6]. To accomplish this, the direction the particles are moving and the energy they carry must be found. Energy is found using a calorimeter, a thick silicon shell surrounding the detector, where the particle is fully absorbed. The direction is found using two silicon layers, generating two points in the particle path. A unit vector between them is calculated, shown in figure 2. The momentum magnitude can be found using equation 2, and therefore, the missing mass.

There are some aspects of the detector that may effect the performance of the detector. These aspects are the length of the barrel, the thickness of the layers, the gap in-between the layers and the thickness of the calorimeter. The driving force behind the effect of the thickness of the layers and calorimeter will be determined by the Bethe-Bloch formula, which describes the particles energy dissipation:

$$\left\langle \frac{\partial E}{\partial x} \right\rangle = -Kz^2 \frac{Z}{A\beta^2} \left[\frac{1}{2} \ln \left(\frac{2m_e c^2 \beta^2 \gamma^2 T_{max}}{I^2} \right) - \beta^2 - \frac{\delta(\beta\gamma)}{2} \right] \tag{3}$$

Where,$\langle dE/dx \rangle$ is the mean of the derivative of energy in some x-direction. $K = 4NAre^2mec^2$, N_A is Avogadro's constant, r_e is the classical electron radius (2.818 fm), z is the charge of the incident particle, Z,A are the atomic number and the atomic mass of the absorber, respectively, β is the ratio of the particle's velocity to the speed of light, γ is the Lorentz factor, T_{max} is the maximum kinetic energy that is able to be transferred to a free electron in a single collision, I is the mean excitation energy, and $\delta(\beta\gamma)$ is the density effect correction to ionisation energy loss. This correction is added because the electric field will flatten at higher energies, increasing the effects of the distant-collision contribution on energy dissipation[9].

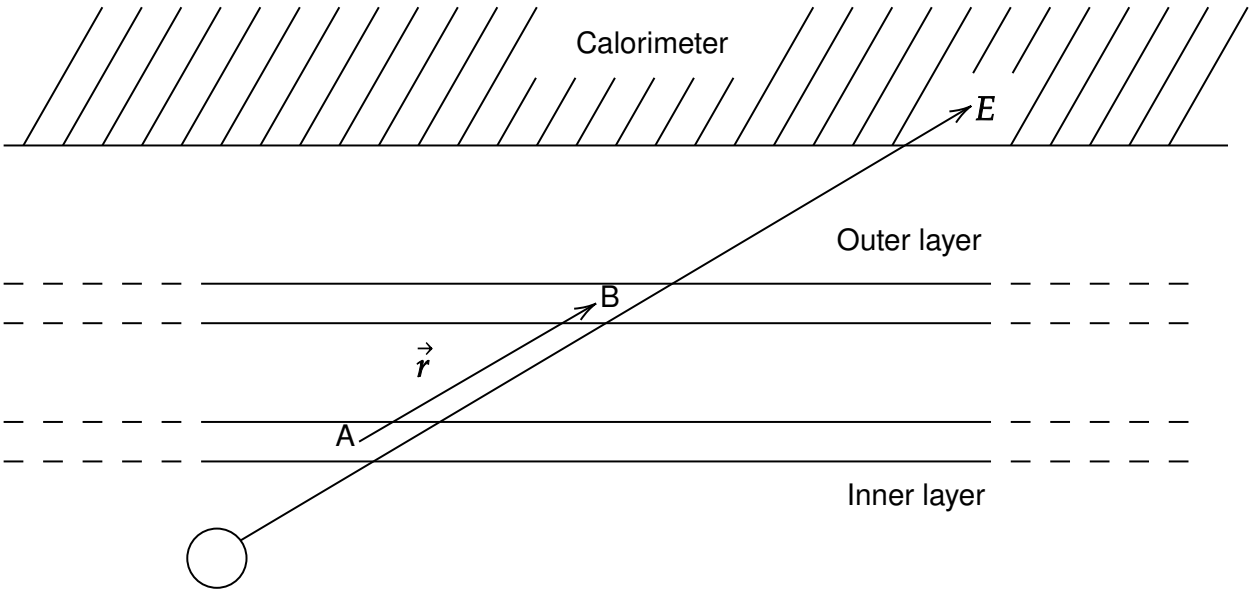


Figure 2: A schematic of the general layout of a particle detector. Where, a particle passes through 2 layers of silicon at points A and B, depositing all of its remaining energy into the calorimeter. Allowing for the vector, r to be found, and the 4-momentum to be calculated.

Methods

Geant4 is a framework created by CERN to simulate particle behaviour via a Monte Carlo method[10] and was used to model and simulate the detector performance. A Monte Carlo simulation is where the main driving principle is random sampling. These samples will come from some appropriate probability density, depending on the interaction that is being dealt with[11]. During this report, full working knowledge of the data structure of Geant4 was gained and used in the creation of the detector and simulations in question.

The basis of Geant4 is the use of base classes that the user can customise for their own needs. Everything is done in C++. The "Physics List" class is responsible for telling the simulation which interactions should be considered. E.g, electromagnetic interactions, hadronic interactions, and/or the decay of unstable particles. Another use of the Physics List is to specify which particles will be present during the simulations, leptons, hadrons, ions and such.

The "Detector Construction" class defines the 3D geometry of the simulation. Geometry is structured using three forms, the solid, logical and physical volumes. The solid volume is the most elementary form, and just contains the coordinates needed to fully define the shape. E.g, a solid rectangle would be 3 values, 1 for each side length. The Logical volume will take the solid volume and specify what material it will be made from, determining its properties. The Physical volume will take the logical volume and define the coordinates and rotation. Another important role of this class is to define the fields that will be present and to also specify which volumes will be "sensitive detectors".

A Sensitive Detector (SD) is a volume that can be chosen to output data. This class specifically, will state the data that is output. E.g, the coordinates of the sensitive detector that had a particle incident on it. This data will then be put into a "Hits" object. The "Hits" class will be used to process the data output by the sensitive detector. During the simulation, a collection of Hits are generated to be post-processed. If multiple SDs are defined, multiple collections are generated.

Geant4 has three levels of tracking particles through the simulation. The top-level is called the "Run" which encapsulates the entirety of a single execution of the program. Below this is the "Event". An Event is an individual simulation within a Run. The lowest level is the "Step", which is the size of a timestep within each event. Each of these levels will have a class associated with them called "Actions". E.g Run action, Event action and Stepping action. The Action classes are used so the user can tell the program what should be done within each of the levels. E.g, in the Event action, one could tell the program to count the number of collisions that occur. These classes are optional and the program must be told which user made actions should be taken into account. This will be done in a small class called "Action Initialization".

The final compulsory class is the "Primary Generator" where the initial state of the simulation will be defined. The types of particles, their origin point, and their initial energy or momentum are set. The generator will be run at the start of each Event.

The main function will initialise all the classes needed to run the simulation, along with the visualiser class that is provided, illustrated in figure 3. The user can choose to run in batch or visualiser mode. Batch mode is when the simulation is run without the visualiser and in one go, outputting results to the terminal. The visualiser mode will launch the GUI and allow for modification of the geometry, and primary generator and it gives the ability to run each Event individually. All the data will be fed into a root file for analysis within the ROOT framework.

These files can contain data in different forms. The two main forms are the Histogram and a Tree. These differ in the fact that a Tree contains Branches, which is essentially a table of data, within a Branch, are columns called a Leaf, also called an "nTuple". For example, there may be 3 different Leaves, each containing either the x,y or z coordinates of each particle in the experiment. A Histogram will contain the number of values within each bin. Therefore, in this example, would store the number of particles within some range of x,y and z coordinates. This allows for a large amount of data to be stored within a Histogram while taking up very little storage space because a Tree of 1,000,000 values could be condensed into a Histogram of 100 bins, therefore, only storing 100 values.

Code Description

Detector Geometry

The whole simulation takes place in a vacuum. The calorimeter can be seen figure 5a. It is made of silicon and the central space has a radius of 80 cm to accommodate the varying length of the

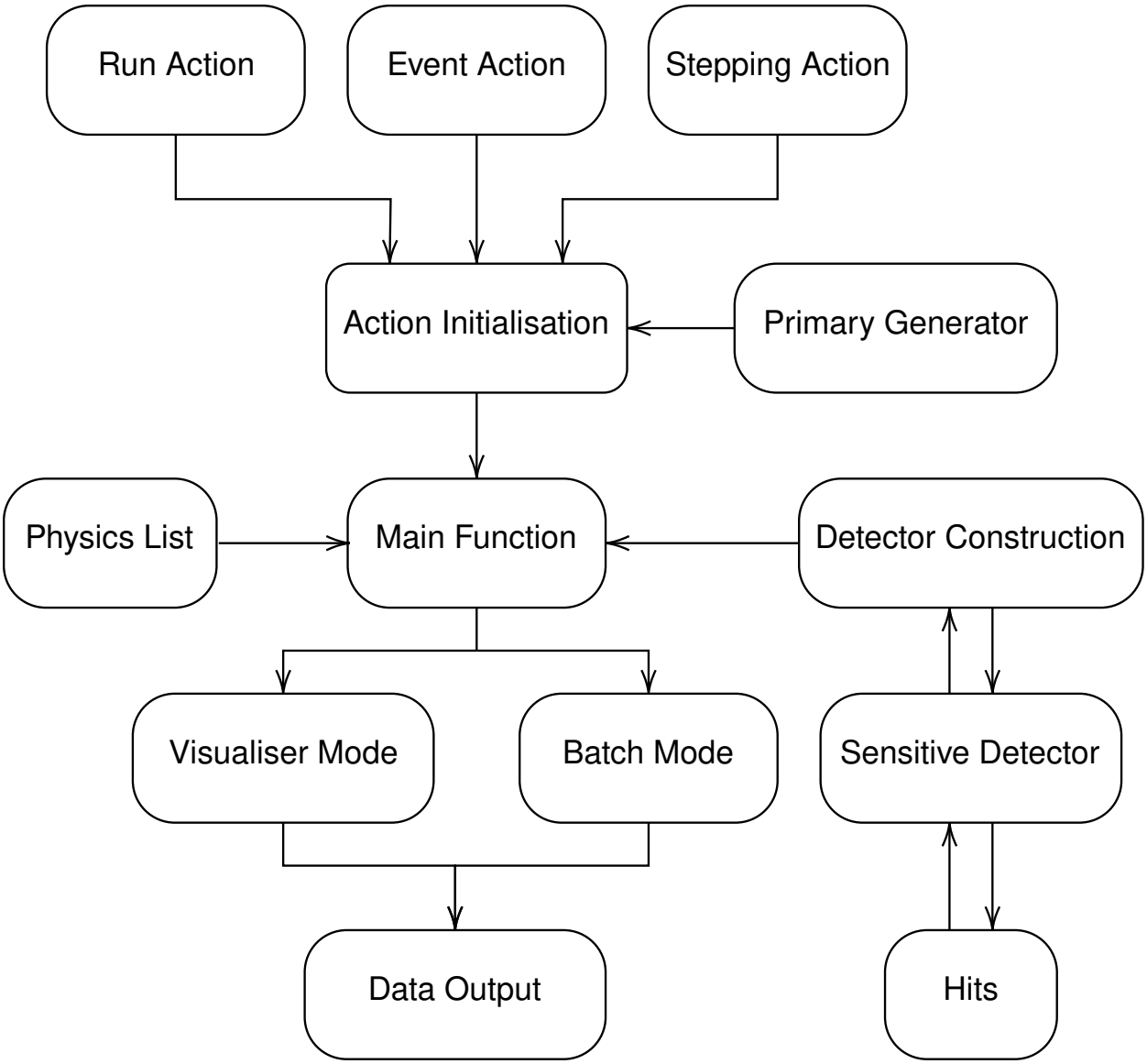


Figure 3: A data flow diagram of a general Geant4 program. Where user-made actions and the Primary Generator are initialised, and then fed into the main function. The geometry is defined in the Detector Construction, where the sensitive detector geometry is defined, the way the sensitive detector reacts is specified in its own class, which feeds the data into a Hit class for processing, which is then fed back into the sensitive detector class for recording. The Physics list and detector construction are also fed into the min function, which can either enter batch mode or visualiser mode, and then output the data into the wanted format.

main detector. The thickness will be varied between 80 cm and 200 cm to demonstrate particle punch-through.

The duel silicon layers are segmented via radial segments that span over the same angle; the length of the cylinder is also segmented. This is done with the nested loop and rotations in figure 4. One radial segment is defined as a solid and logical volume, then within the loop, the segment

is rotated by some angle that depends on the number of radial segments, then a physical volume is placed. This is repeated until the full 360 degrees are filled. The outer loop is then enumerated, which will translate the segment down the z-axis. When the nested loop is complete, inner and outer segmented layers should have been generated. Note that the outer segments and the inner segments are given the "copy numbers" of 1 and 0 respectively. This number is a tag given to differentiate between the layers when collecting and processing data. The final piece of geometry

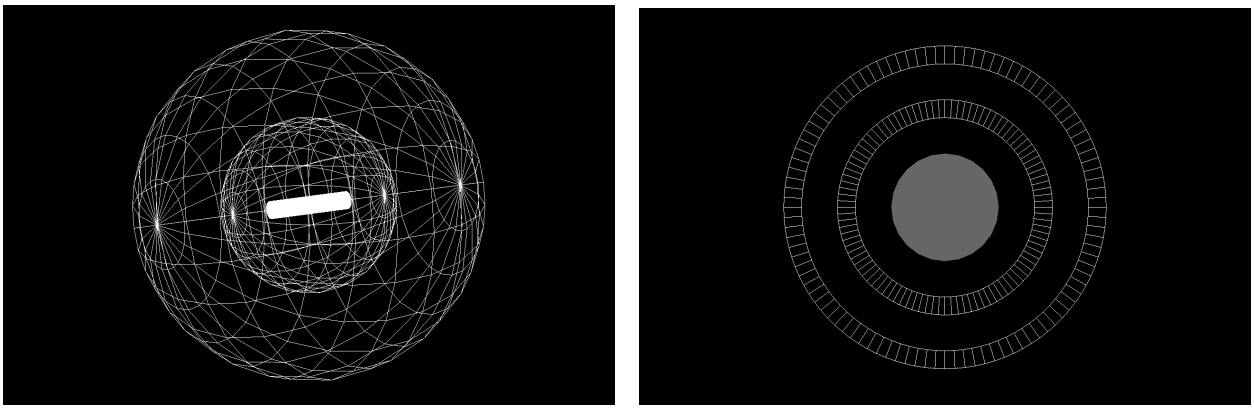
```
88          //Loop to generate tube of cylinders in the z axis
89          for(G4int j=0;j<numRow;j++){
90                  //Loop to generate full cylinder
91                  for(G4int i=0;i<numSeg;i++){
92
93                          rot = new G4RotationMatrix();
94
95                          //Moving the segment
96                          rot->rotateZ(i*innEnd);
97                          trans = G4ThreeVector(0,0,segLen*(1-numRow)/2+2*j*innHHight);
98
99                          physicalInn = new G4PVPlacement(rot,trans,logicInn,"Inner",logicWorld,false,0,checkOverlaps);
100                         physicalOut = new G4PVPlacement(rot,trans,logicOut,"Outer",logicWorld,false,1,checkOverlaps);
101                 }
102         }
```

Figure 4: The nested loop that generates each segment of the silicon layers within "p2pDetectorConstruction.cpp". "rot" is a rotation matrix responsible for the radial rotations of the segment, while "trans" is a 3D vector responsible for the z axis translated of the segment. "numSeg" is the number of radial segments. "numRow" is the number of translated full 360 degrees of segments. "innHHight" is the half length in the z direction of the radial segments. "logicInn" and "logicOut" are the predefined logical volumes of the inner and outer layer. "innEnd" is the angle swept by an individual segment.

is the liquid hydrogen target, in figure 5b. This was implemented as the particles will be originating from this target, and will be distorted by it, creating a theoretical ceiling to how accurate the detector can be. The target is a small cylinder that fits at the very centre of the detector. It has a diameter of 3 cm and a length of 3 cm down the z-axis.

Primary Generator

The data about the initial state of the generated protons due to a (p,2p) reaction is given within a ROOT file containing 100,240 unique reactions, meaning the program has 100,240 events within each run. These events were generated from a Carbon-12 ion beam of energy 500 MeV/u with no excitation energy, meaning the valence proton is knocked out. This class will read this data and configure the "particle gun" accordingly. The particle gun is an object that is used to give the particles their initial 3-momentum and origin point.



(a) (b)

Figure 5: (a) Visualisation of the spherical calorimeter and the duel layered silicon cylinder detector within the void of the calorimeter. (b) A barrel view of the silicon cylinder detectors and the liquid hydrogen target in the centre. Note, the thickness of the layers has been greatly exaggerated in order to visualise the radial segments.

Sensitive Detector

The layers were set as the sensitive detectors. The Sensitive Detector class will digitise the particle position data by finding the coordinates of the segment that the particle passed through. First, the copy number of the volume that was hit is found to distinguish the inner and outer layers. The ID of the particle that hit is also found, which is either 1 or 2. Using the physical volume, the translation down the z-axis (barrel) and the rotation matrix can then be found. This and the copy number are used to find the position vector of the segment. The copy number, the particle ID, the position of the segment and the event number are fed into a "Hit" object, which is added to a Hits collection for that event.

The Hits class can be seen as a simple container for data. Each event should only generate two collections for each SD, containing two hits each, one for each proton. However, there are instances where either more or less are recorded, meaning that data from each event would not correlate when using a ROOT script to calculate the missing mass. More hits could be recorded if the generated protons generate electrons, and these hit the sensitive detector. Too few hits may be recorded in the case that a proton misses the detector or is absorbed before hitting both layers, therefore, data cleaning is done in the "EndOfEvent" method, which is called at the end of each event.

The number of times each particle is incident on the layer is found by looping through the Hits collection. If the count for either particle is equal to 0, meaning the particle missed, a Hit object is added that only contains a particle ID of -1 in order to tell if this Hit represents a miss. The updated collection of Hits is then looped over, and the data is extracted. If the particle hits the layer more than once, only the first Hit is used. This is done by having a counter for each particle and only allowing the data to be recorded if that counter remains at 0, as shown in figure 6. The data cleaning means that exactly 200,480 data points are recorded for each Leaf. A Branch is

generated for each layer.

```
153         //Loops through each hit in collections
154         for (G4int i=0;i<nHits;i++){
155
156                 //Getting hit data individual hit
157                 hit = (*fHitsCollection)[i];
158                 particle = hit->GetTrack();
159                 G4ThreeVector detPos = hit->GetPos();
160                 G4int copyNo = hit->GetCopy();
161
162                 /*
163                  * Only records the data for each particle for their initial
164                  * hit to the detector (In case the particle hits the detector
165                  * multiple times)
166                  */
167
168                 //Checks if particle 1
169                 if (particle == 1){
170                         //Checks if particle 1 hit has already been recorded
171                         if (nump1 == 0){
172                                 G4cout<<"RECORDED P1"<<G4endl;
173                                 recorded += 1;
174
175                                 //copyNo = 0 (Inner), copyNo = 1 (Outer)
176                                 manager->FillNtupleIColumn(copyNo,0,particle);
177                                 manager->FillNtupleDColumn(copyNo,1,detPos[0]);
178                                 manager->FillNtupleDColumn(copyNo,2,detPos[1]);
179                                 manager->FillNtupleDColumn(copyNo,3,detPos[2]);
180                                 manager->FillNtupleIColumn(copyNo,4,event);
181                                 manager->AddNtupleRow(copyNo);
182                         }
183                         nump1 += 1;
184                 }
185                 //Checks if particle 2
```

Figure 6: The loop that only records the first detection of each particle, inside the "EndOfEvent" method in "p2pSensitiveDetector.cpp". "nHits" is the number of Hits in the collection after the missed Hits are added. "Particle" is the ID of the incident proton, either 1, 2 or -1 (missed Hit). "nump1" is the counter for the number of particles with the ID 1 that have been recorded. Once one is recorded, it will ignore all other Hits with an ID of 1. This loop extends below the snippet for protons with the ID 2 or -1, the structure is the same. The methods being called by the "manager" is what records the data into a ROOT Branch.

Calorimeter

The calorimeter is a type of logical volume called a "scoring volume". It is similar to a sensitive detector, but it can accumulate data over an entire event, not just singular particle collisions, allowing for the deposited energy to be measured. The Stepping Action and the Event Action are used in conjunction to achieve this. Within the Stepping Action, the energy deposited from each particle into the volume each time step is recorded, shown in figure 7, this value is then fed into the Event Action, where it is summed for the duration of the event. At the end of the event, the total energy

deposited by each particle is recorded into a ROOT file.

```
void p2pSteppingAction::UserSteppingAction(const G4Step* step){
        //Gets track ID
        G4int trackID = step->GetTrack()->GetTrackID();

        //Gets the logical volume that was hit
        G4LogicalVolume* volume = step->GetPreStepPoint()->GetTouchableHandle()->GetVolume()->GetLogicalVolume();

        //Gets the whole constructor object
        const p2pDetectorConstruction* construction = static_cast<const p2pDetectorConstruction*> (G4RunManager::GetRunManager()->GetUserDetectorConstruction());

        //Gets the scoring volume (calorimeter)
        G4LogicalVolume* fScoringVolume = construction->GetScoringVolume();

        //Only gets the energy deposited into the scoring volume
        if (volume != fScoringVolume) {return;}
        G4double edep = step->GetTotalEnergyDeposit();

        //Checks which particle the energy is associated with
        switch(trackID){
        case 1:
                EventAction->AddEdep1(edep);
                break;
        case 2:
                EventAction->AddEdep2(edep);
                break;
        default:
                break;
        }
```

Figure 7: The method in "p2pSteppingAction.cpp" that will check if the particle is inside the calorimeter scoring volume before getting the energy deposited, and feeding this into "p2pEventAction.cpp" to be summed up and the total energy deposited from each particle is recorded. "TrackID" is either 1 or 2 depending on the particle.

ROOT Script

The output data is processed within a ROOT script in order to reconstruct the missing mass. The initial 4-momentum is calculated from the given dataset of 100,240 events. The 4-momentum after the reaction is found using data from the protons, given by the detector. Some of the data includes the missed Hits that were added during the simulation, making the events where this occurred unusable since the proton trajectory can't be found. So, events that contain a particle ID of -1 are ignored. Once valid events are found, a loop would go through the data from each proton, and calculate the unit vector between each contact point using, $\vec{AB} = (\vec{B} - \vec{A}) / \left|(\vec{B} - \vec{A})\right|$, where $\vec{A}$ and $\vec{B}$ are the position vectors of the segments hit. The magnitude of the momentum is then calculated using equation 2. The energy values given by the calorimeter are 100% accurate, in reality, the calorimeter would have some variation due to finite resolutions. To simulate this, the momentum is randomised slightly using a gaussian random number generator with a 1% full width at half maximum. This modified momentum is used along with the energy to get the 4-momentum. The missing mass is then found by calculating the difference between the initial and final 4-momentum, the magnitude of this minus the mass of the fragment is put into the Histogram. This is repeated for all valid events, shown in figure 8.

```
143         //Loop though all particles in the dataset
144         for(int i=0;i<(int)(data.size()/2);i++){
145                 //Checks if ANY particle missed ANY of the detectors
146                 if (data[2*i][0] != -1 && data[2*i][1] != -1 && data[2*i+1][0] != -1 && data[2*i+1][1] != -1){
147
148                         eventMom.clear();
149                         //Loops though the pair of particles for each event that has no misses
150                         for (int j=0;j<2;j++){
151
152                                 //Displacement 3-vector of particle
153                                 TVector3 mom = {data[2*i+j][6]-data[2*i+j][3],
154                                                 data[2*i+j][7]-data[2*i+j][4],
155                                                 data[2*i+j][8]-data[2*i+j][5]};
156
157                                 //Converts displacement to momentum 4-vector
158                                 //Checks if particle 1
159                                 if (data[2*i+j][0]==1){
160                                         p = momentum(allEnergy[i][0],proM);
161                                         pRand = r3.Gaus(p,p*FWHW);
162                                         mom = pRand*mom.Unit();
163                                         lmom.SetPxPyPzE(mom.x(),mom.y(),mom.z(),allEnergy[i][0]);
164                                 }
165                                 //Checks if particle 2
166                                 if (data[2*i+j][0]==2){
167                                         p = momentum(allEnergy[i][1],proM);
168                                         pRand = r3.Gaus(p,p*FWHW);
169                                         mom = pRand*mom.Unit();
170                                         lmom.SetPxPyPzE(mom.x(),mom.y(),mom.z(),allEnergy[i][1]);
171                                 }
172                                 eventMom.push_back(lmom);
173
174                         }
175                         //Calculates total final 4-momentum
176                         TLorentzVector momOut = eventMom[0] + eventMom[1];
177                         //Calculates missing mass 4-momentum
178                         missingMom = momIn - momOut;
179                         //Calculates missing mass
180                         missingMass = missingMom.M()-fragM;
181                         std::cout<<"Missing Mass: "<<missingMass<<std::endl;
182                         //Puts value into histogram
183                         missHist->Fill(missingMass);
```

Figure 8: The loop that checks for invalid events, and calculates the missing mass for all the valid events in the "missingmass.cpp" ROOT script. "data" is a 2d vector containing the positional and particle data for both particles for each event. Each row of the vector is ordered as: Inner ID, Outer ID, Event number, Inner x, Inner y, Inner z, Outer x, Outer y, Outer z, where Inner indicates data related to the inner layer and Outer is data related to the outer layer. Every pair of 2 rows correspond to both particles of a single event. "Mom" is the momentum 3-vector, while "lmom" is the momentum 4-vector of the proton. "momIn" and "momOut" are the total initial and final 4-momentum vectors. "fragM" is the mass of the fragment after the reaction.

Variables to Optimise

Between experiments, the efficiency of the detector, the mean missing mass and the width of the reconstruction will be measured. This is done by fitting a Gaussian curve to the reconstruction and taking the centre and the standard deviation/sigma values. This fit is used because the mean and standard deviation given my ROOT may be influenced by outliers. The detector was split into 100 radial segments, and 100 rows, giving a total of 10,000 segments for these experiments.

The length of the detector will be varied between 20 cm and 80 cm in 1cm intervals. The thickness of the layers is varied between 0.01 mm to 1 mm in 0.01 mm intervals. The gap in-between is tested with a range of 0.1 cm to 5 cm in 0.1 cm intervals. The calorimeter is tested by changing the thickness from 80 cm to 200 cm in 5 cm intervals. The efficiency is the proportion of value events detected to total events.

Results

For reference, the missing mass was found using the data given about the proton's initial momentum and direction. This would give the theoretical, perfect missing mass for this specific reaction setup, shown in figure 9.

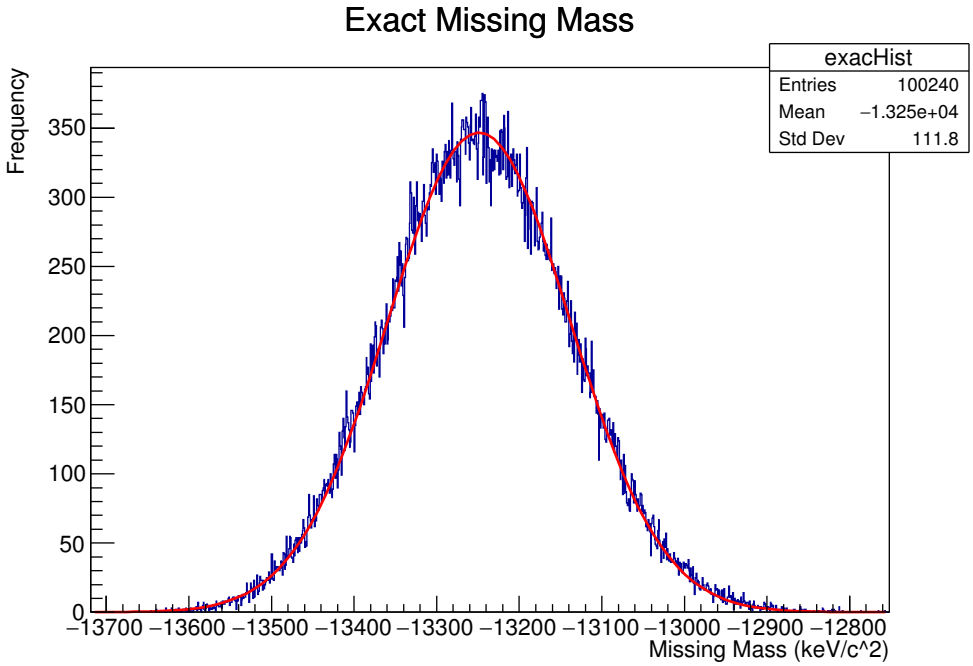


Figure 9: The missing mass reconstruction generated directly from the p,2p reaction data given, with a Gaussian curve fitted. The Gaussian has a maximum value at -13249 keV/c^2, while its standard deviation is 110.68 keV/c^2, which implies the binding energy of the valence proton in a carbon-12 atom is -13249 keV

While varying the length, the thickness of the layers were set to 0.01 mm, the thickness of the calorimeter was set to 80 cm and the gap between the layers was 3 cm. Figure 10 shows that as the length of the detector layers was increased, the efficiency of the detector was seen to increase non-linearly, and seems to asymptote. The number of valid events was either 0 or extremely low for detectors of lengths less than 25 cm.

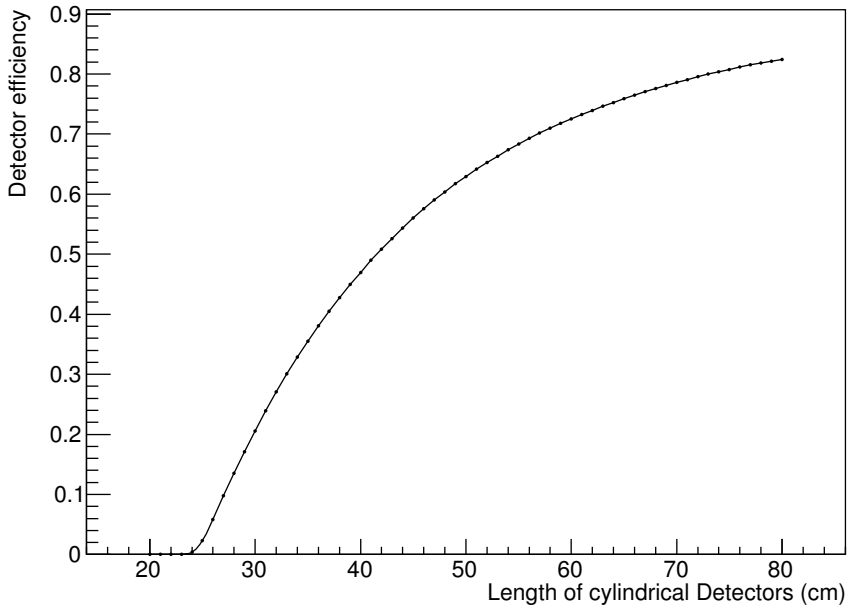
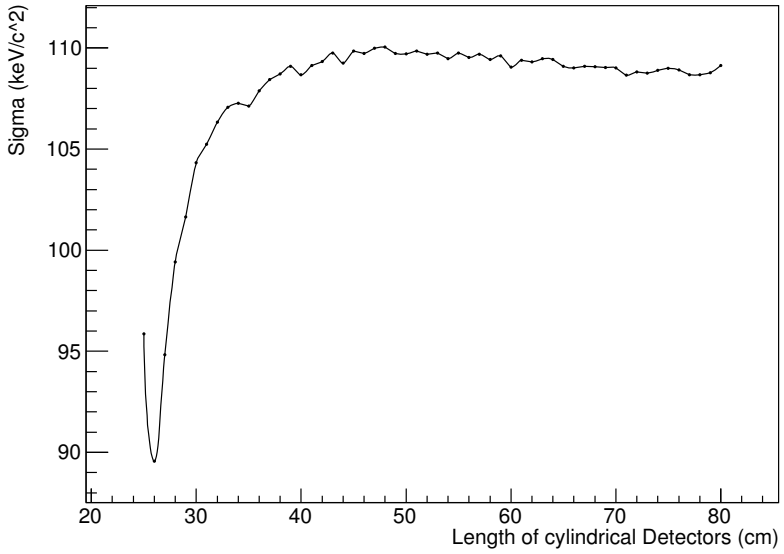


Figure 10: The proportion of fully detected p,2p reactions as the length of the detector layers was increased. The detector had a layer thickness of 0.01 mm, and a gap of 3 cm, while the calorimeter has a thickness of 80 cm. The efficiency of the detector was also 0 for lengths below 24 cm.

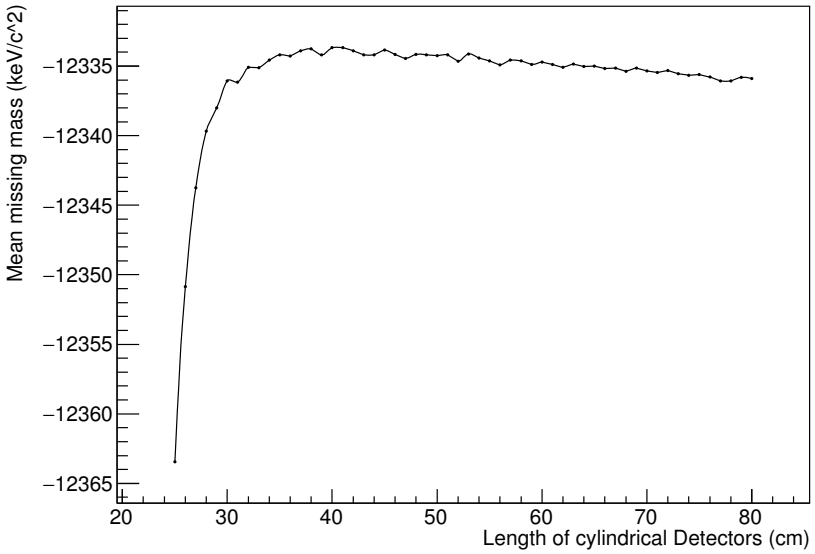
The sigma values of the fitted Gaussian and the maximum missing mass values were found to increase very fast and level out, as shown in Figures 11a and 11b respectively. The sigma values would level out at a value of roughly 109 keV/c^2 if the detector was about 40 cm or longer, while the missing mass would level out at around 35 cm with a value of -12335 keV/c^2. After this length, the trend becomes linear with a slight negative gradient. For these measurements, the experiments for lengths between 20-25cm have been omitted because the number of valid events was so low, an accurate Gaussian could not be fitted and would result in a misrepresentation of the values.

The thickness of the layers is varied with a detector length of 80 cm and a layer gap of 3 cm. The efficiency decreases linearly as the thickness increases, in figure 12. The reduction in the efficiency over the test range was approximately 0.01%.

Figure 13a shows that the missing mass follows a similar trend to the efficiency, a linear plot with a negative gradient. Within the tested range, the missing mass only decreased by about 6 keV, a percentage change of 0.05%. The width of the missing mass increases as the thickness increases, linearly, as evident in figure 13b. The change over the tested range is small, at 2%.



(a)



(b)

Figure 11: (a): The variation in the standard deviation/sigma of the Gaussian fitted to the missing mass reconstruction as the length of the detector layers was increased from 26 cm to 80 cm. The sigma values level off very quickly to roughly 109 keV/c^2, after the detector reached a length of around 40 cm. (b): The variation on the centre of the Gaussian fitted to the missing mass reconstruction, effectively giving the missing mass calculated for each experiment between detector lengths of 26cm to 80cm. The missing mass becomes relatively linear with a slight negative gradient at roughly -12335 keV/c^2.

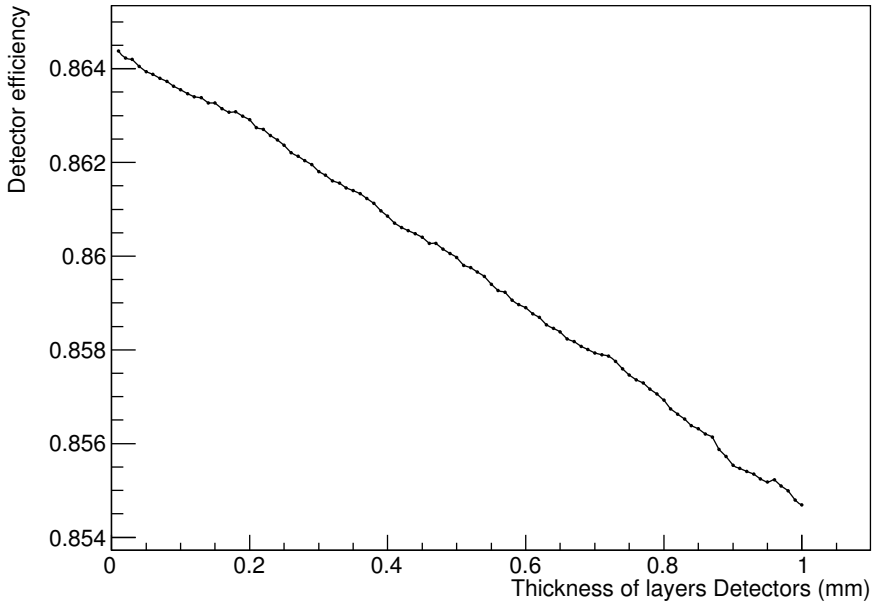
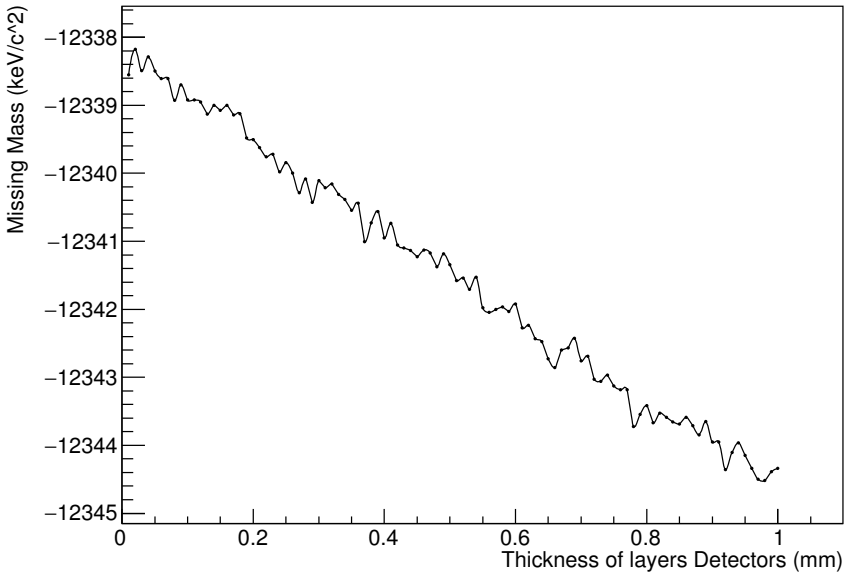


Figure 12: The effects the thickness of the dual-layered detectors have on the overall efficiency of the detector system. Data points were carried out through a range of 0.01 to 1 mm.

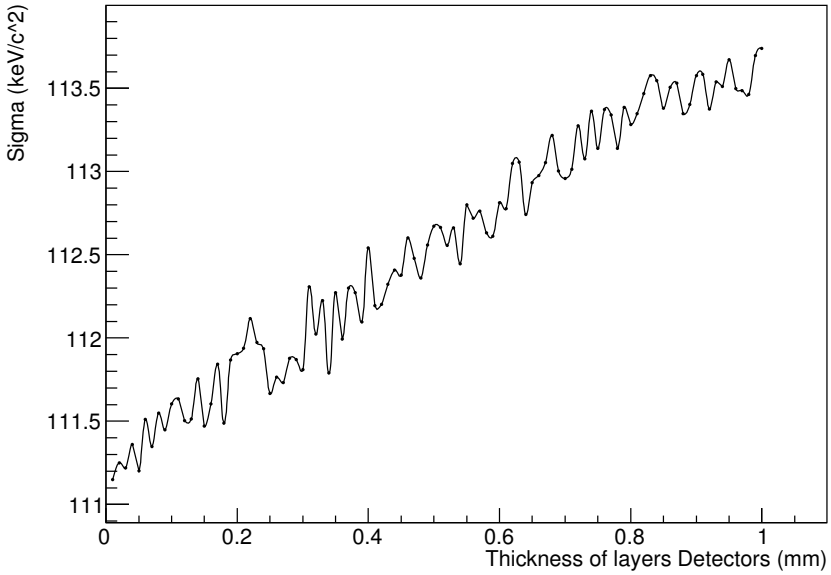
The gap between the detector layers is tested with a detector length of 80 cm and a layer thickness of 0.01 mm. Figure 14a demonstrates that increasing the gap caused the efficiency to decrease in a non-linear way. The total percentage decrease is approximately 0.07%, similar to the effects of increasing the thickness. Even though the experiments took place with a minimum gap of 0.1 cm, the first 6 experiments are omitted due to the shape of the missing mass reconstruction not resembling a Gaussian, therefore, accurate values could not be found. An example of this is shown in figure 14b, which shows the missing mass while the gap is set to 0.5 cm.

Figure 15a shows that the missing mass shifts in the positive direction then becomes constant when the gap is around 4 cm. The percentage increase that can be seen is 0.3%. In figure 15b, The sigma value follows an inverse relation to the gap of the layers increasing and becomes relatively constant by the time the gap is set to 4 cm.

A comparison of the exact energy and the measured energy by the calorimeter was also found. For these experiments, the length of the layers was 80 cm, while the thickness of the layers was 0.01 mm and the gap in between them was 3cm. Figure 16a shows the energy comparison when the calorimeter is 80 cm thick, the effects of punch-through can be clearly seen. There is a linear relationship where the gradient is very close to one. However, particles that have higher initial energy, are not depositing all of that into the calorimeter. After multiple experiments with varying thicknesses, when the calorimeter was 160 cm thick, the punch-through stopped entirely, shown in figure 16b. Note that the gradient is slightly more than 1.

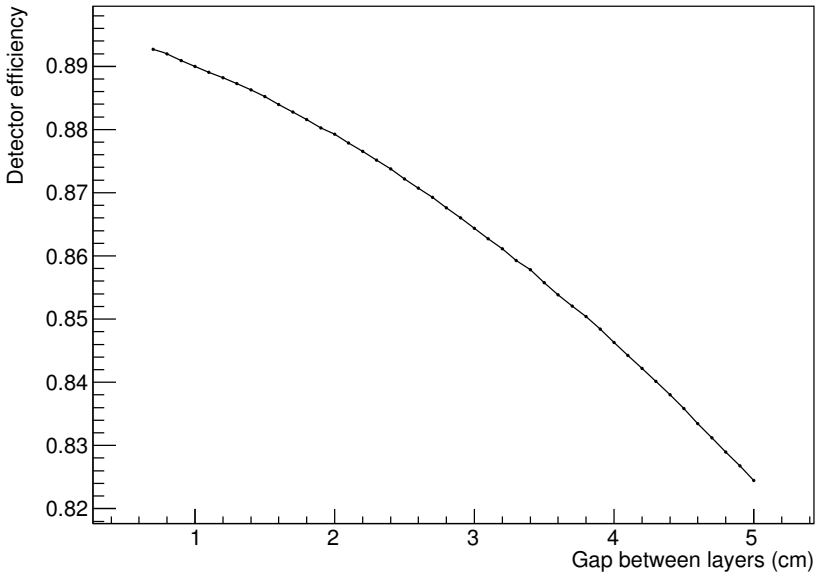


(a)

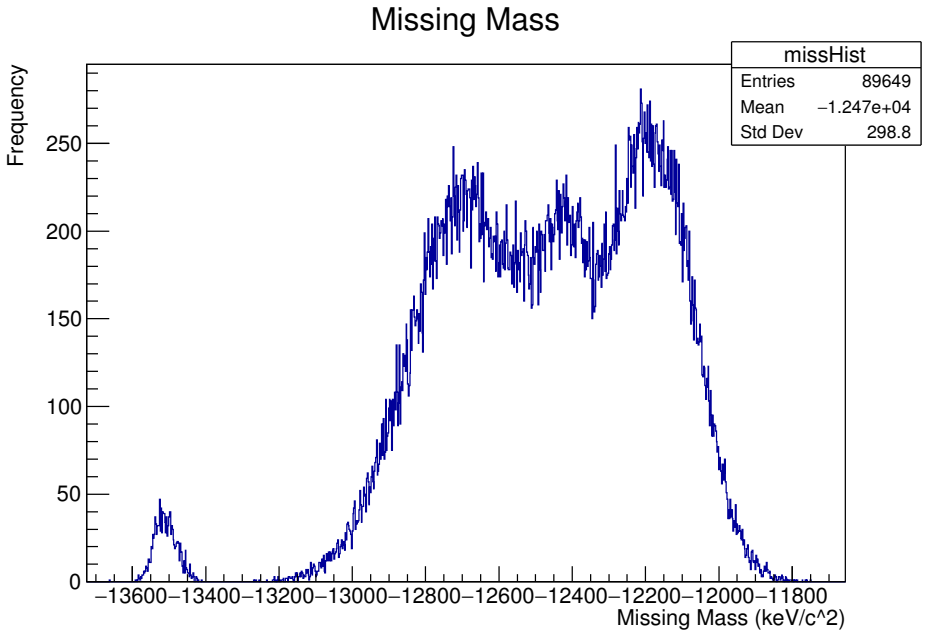


(b)

Figure 13: (a): The trend of how the missing mass value, found from a fitted Gaussian on the missing mass reconstruction, is affected by the change in the thickness of the detector layer. The trend is linear with a very shallow gradient. (b): The standard deviation of the Gaussian fitted over the missing mass reconstruction as the thickness of the detector layers is increased. The trend is linear with a very slight positive gradient.

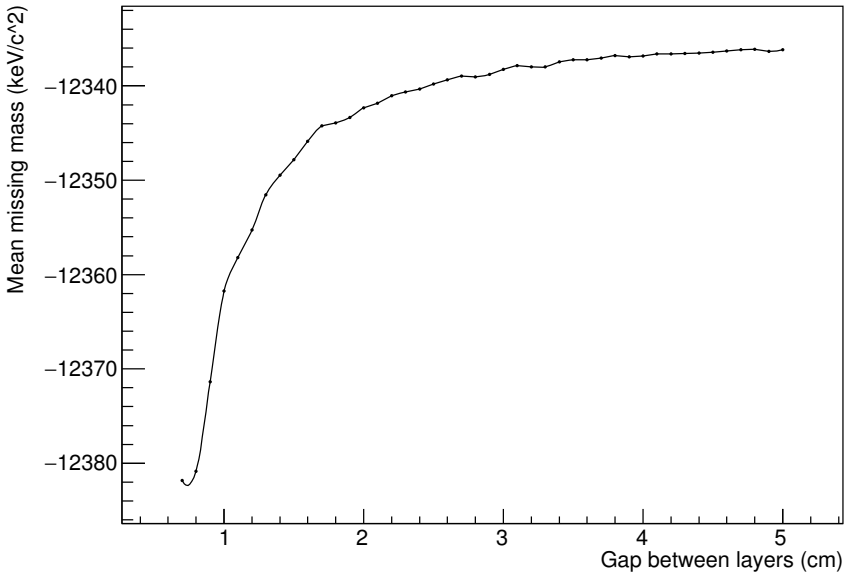


(a)

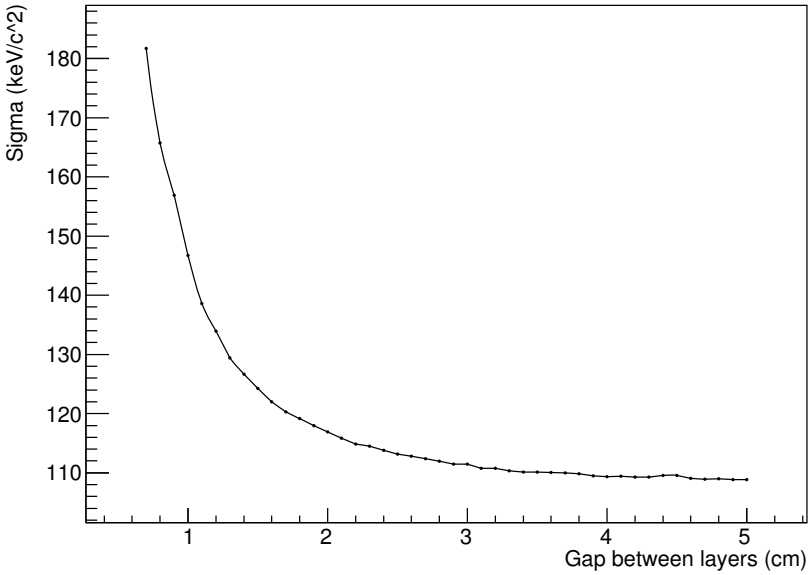


(b)

Figure 14: (a): The effects of the number of valid events recorded as the gap of the detector was increased from 0.7 cm to 5 cm. A non-linear negative trend can be seen, as the slope is getting more negative for larger gaps. Over this range, the detector loses approximately 0.07% efficiency. (b): The missing mass reconstruction of a p,2p reaction where the detector has a length of 80cm, a layer thickness of 0.01 mm and a layer gap of 0.5 cm. A Gaussian cannot be fitted to this reconstruction due to its shape.

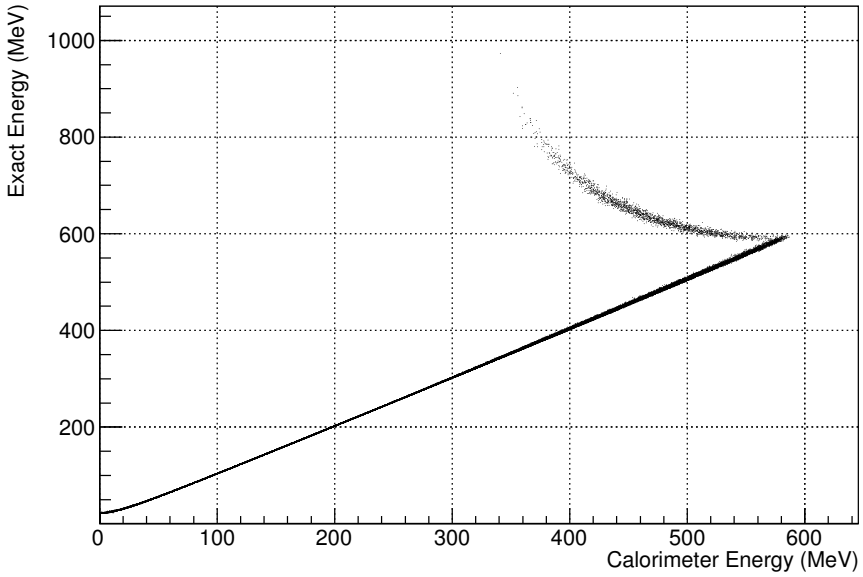


(a)

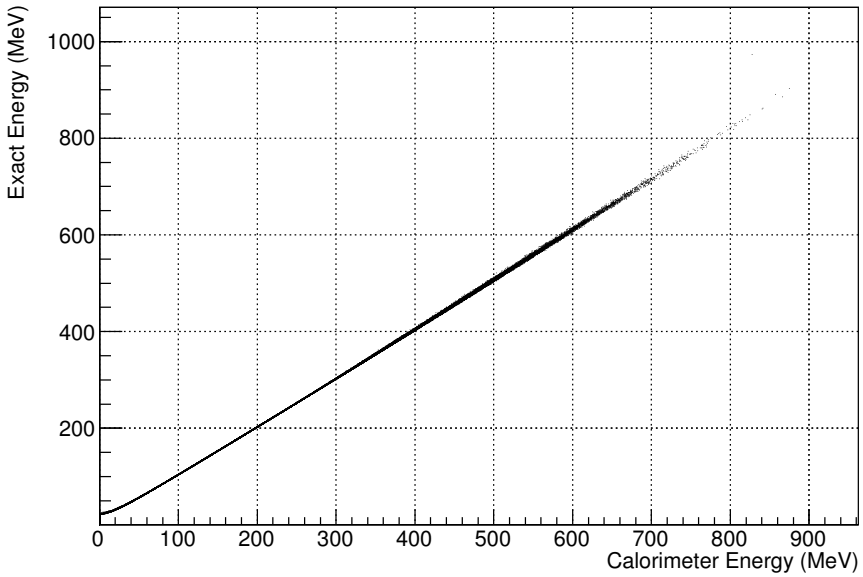


(b)

Figure 15: (a): The centre of the Gaussian fitted to the missing mass reconstruction increases, then levels out as the gap in between the detector layers is increased. The missing mass levels out asymptotically, until it is relatively constant above a gap of 4cm. (b): An inverse relation shown in how the sigma value of the fitted Gaussian curve on the missing mass reconstruction as the gap in between detector layers is increased between 0.7 cm to 5 cm. The sigma value becomes relatively constant when the gap becomes 4 cm and above.



(a)



(b)

Figure 16: (a):A comparison between the exact energy that the protons of a (p,2p) reaction have and the energy that was recorded by a calorimeter with a thickness of 80 cm. As the energy of the particles increased, the amount of that energy deposited into the calorimeter decreased, having an inverse relationship. (b): A comparison of the exact energy of the protons in the (p,2p) reaction dataset given, and the energy measured by the calorimeter.

Discussion

There is a difference in the missing mass shown in figure 9, compared to results shown in figures 11b, 13a and 15a by approximately 1 MeV. This will be due to the unaccounted energy dissipation in the target and layers.

Seen in figure 10, the reason the efficiency increases with a longer detector is because the detector spans over a larger solid angle, therefore, more protons are caught. Due to the direction of the incoming ion beam, the area where the protons can move is a cone, this is why the efficiency asymptotes, because to encompass the entire cone, the detector will have to be infinitely long. The reason why figures 11a and 11b show a very fast increase into a linear relationship is because at lower detector lengths, the number of valid events is so low that the distribution is not represented fully. There could be multiple factors that cause the slight negative gradient in figure 11b. First, higher energy protons will have a shallower trajectory, relative to the direction of the ion beam/along the detector length. Meaning that they will traverse a longer distance through the layers, however, it can be seen in equation 3 that, $\langle dE/dx \rangle \propto 1/v^2$, so less energy will be deposited. On the other hand, since the protons are produced in pairs, if one of the protons has extremely high energy, the other will have low energy and a steeper trajectory. This lower energy proton will deposit a higher proportion of its energy into the layers before hitting the calorimeter. This effect could be removed if the energy deposited into the detector layers was also recorded and corrected for. This could be done via the Bethe-Boch formula, or Geant4 could directly give the energy deposited. Using figures 10,11a and 11b, to get the most accurate results, the longer the detector, the better. Since the efficiency increases and both the sigma values and the missing mass values level off, this leads to the belief that the best length for this scenario is 80cm. Longer lengths could be used if cost-efficient.

The small drop in efficiency seen in figure 12 can be attributed to the lower energy protons being absorbed by either the inner or outer detector, and never making it to the calorimeter. This effect is very small because a huge majority of the protons have enough energy to make it through. This, therefore, influences the missing mass reconstruction, evident in figure 13a and 13b. Since more energy is being lost in the layers, the energy measured by the calorimeter is smaller, causing the missing mass to decrease. This is a small shift because the energy being absorbed is minimal. In contrast to the most effective length, having layers as thin as possible would be best, as the efficiency of the detector would be maximised, while the small shift in sigma and missing mass is also minimised. The optimal thickness for this case would therefore be 0.01mm. Thinner layers would be more effective if they are practical to use for a real detector.

As seen in figure 14a, the efficiency is reduced in a non-linear way as the gap between layers is increased. This may be due to the outer layer moving further from the detector axis, so the solid angle swept over is reduced, meaning that more protons will pass through the inner layer, but miss the outer layer. The effects in figures 14b, 15a and 15b are due to the trajectory of the particles that is calculated compared to the actual direction they move. The main reason for this is the resolution of the detector. If a particle travels through the inner layer at an angle, hitting some segment, but the outer is extremely close, then the particle may then hit the outer segment

that is directly above the inner segment. This means that the calculated vector would be pointing at a right angle, compared to the particle that is actually moving at some angle. This effect is shown in figure 17 and could be reduced with a higher resolution detector. However, before each simulation is run, Geant4 generates each segment individually, and since there is a number for the radial segments and the translated segments, increasing the number of segments would increase the time for each simulation significantly. In this simulation, a type of physical volume called "PVPlacement" was used. This just generated each segment independent from the others. Another physical volume type called "PVReplica" that is specifically used for splitting larger volumes into identical, smaller segments could have been used. Using this method would reduce memory usage and could possibly allow for a larger number of segments to be used.

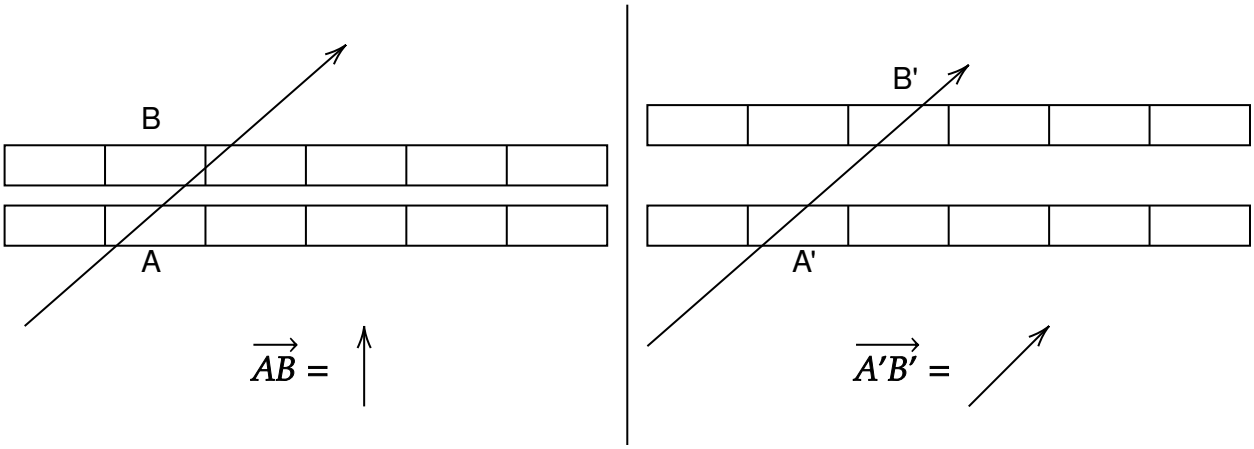


Figure 17: A schematic showing the effect of having a small gap between detector layers has on the perceived direction of travel of a particle. For a small gap, the segment's hit causes the detector to perceive the direction of travel to be vertical, while with a larger gap, the calculated vector is closer to the actual direction of travel.

For a gap that is less than 0.7cm, this effect is so pronounced that the missing mass is distorted to the point where it is no longer a Gaussian, as the example in figure 14b shows. However, for gaps larger than this the missing mass rapidly increases then levels out in a similar way to the sigma values in figures 15a and 15b respectively. This may be due to the effect in figure 17, but less pronounced, so the calculated trajectory gets more accurate as the gap increases, giving a more accurate missing mass reconstruction. Having the smallest possible gap would maximise efficiency, however, the accuracy of the results would suffer as a result. Figures 15a and 15b indicate that the best gap to have while still allowing for more reactions to be valid would be around 3cm. At this point, the sigma values and missing mass values have levelled out and increasing the gap would cause further reduction in the detector efficiency.

The visible punch-through in figure 16a is due to the inverse proportionality of the stopping power to the velocity seen in equation 3. This means that the faster the particles are moving through the material, the less energy they deposit so the calorimeter is not measuring the total energy the particles have because they are not being fully absorbed. However, figure 16b is the result of increasing the thickness of the calorimeter until the particles deposit all of their energy.

The gradient being slightly more than 1, more visible in figure 16b, is expected. This is a result of some of the energy being absorbed by the layers of the detector before hitting the calorimeter. Like the negative gradient in figure 11b, this effect can be minimised by correcting the energy value by measuring the energy deposited into both layers and the calorimeter.

Conclusion

To conclude, Geant4 is a useful tool for designing and optimising detectors, as well as testing them in many different situations. Only a basic understanding of C++ and object-oriented programming is needed to understand the data structure of the framework and to actively implement it. ROOT is similar in that only a basic understanding of C++ would be needed to understand how the functions are carried out. The flexibility of the ROOT file system is large, and very memory efficient and therefore can be used for much larger datasets.

For this specific project, where the Carbon-12 ion beam has an initial energy of 500MeV/u, the most effective detector parameters would be to have a calorimeter that is at least 160cm thick surrounding two cylindrical layers of silicon that each have a thickness of 0.01mm or smaller. A gap between these layers should be around 3cm, while the length of them should be 80cm or longer. However, a detector that has a higher resolution than 100 radial segments and 100 transverse segments may be able to have a smaller gap between layers, giving the system a better efficiency, while still maintaining reasonable accuracy. Another factor that was not touched upon in this report was the effects of the radius of the cylinders, this may have a similar effect to increasing the gap or decreasing the length of the detector as the solid angle would be reduced as the radius increases.

Some factors could not be explored, however. For a real detector, there would be structural supports and electronic components, along with glues and methods of cooling that could all contribute to distortions in the paths and energies of the generated protons. With more time and greater knowledge of Geant4, this would be possible to model and simulate, but as a proof of concept, the p,2p reaction can be utilised in such a way as to extract a significant amount of information about atoms, exotic and otherwise. But overall, this data will prove useful for further developments in the detector used for R3B experiments in the future.

Appendices

Executing code

All the code and ROOT scripts can be downloaded via GitHub using the command line "`git clone https://github.com/ConnerGrice/Geant4-P2P-Investigation`", or accessed directly on the GitHub website using the link: https://github.com/ConnerGrice/Geant4-P2P-Investigation. The versions of Geant4 and ROOT that were used are 10.07.02 and 6.18.04 respectively. Once Geant4 and ROOT have been installed and compiled, to compile the project code, CMake is used, while in a build directory use the command "`cmake -DGeant4_DIR="Geant4 install directory" ../build`", then use the command "`make -jx`, where x is the number of processor cores that is wanted for compilation. To run the full experiments, use the command "`./p2p run.macs`", to launch the visualiser, use "`./p2p`".

The main ROOT file to calculate the missing mass along with all of the user created classes are listed below, leaving out their header files.

Missing mass ROOT script

```
1  #include <math.h>
2  #include <vector>
3  #include <iostream>
4  #include <TTree.h>
5  #include <TFile.h>
6  #include <TH1F.h>
7  #include <TF1.h>
8  #include <TCanvas.h>
9  #include <TTreeReader.h>
10 #include <TTreeReaderValue.h>
11 #include <TVector3.h>
12 #include <TLorentzVector.h>
13 #include <TSystem.h>
14 #include <TRandom3.h>
15 #include "info.hh"
16
17 double momentum(double energy, const double mass){
18   return sqrt((energy*energy)+(2*energy*mass));
19 }
20
21 void missingmass(){
22
23   TH1F* missHist = new TH1F("missHist","Missing␣Mass",1000,-1100,-1500);
24
25   missHist->GetXaxis()->SetTitle("Missing␣Mass␣(keV/c^2)");
26   missHist->GetYaxis()->SetTitle("Frequency");
27
28   //Gets data file
```

```
29    TFile input("data.root");
30    TFile quasi("quasi.root");
31
32    //Gets separate trees within data file
33    TTree* inner=nullptr;
34    input.GetObject("Inner",inner);
35    TTree* outer=nullptr;
36    input.GetObject("Outer",outer);
37    TTree* particles=nullptr;
38    input.GetObject("Protons", particles);
39
40    TTree* other=nullptr;
41    quasi.GetObject("Particles",other);
42
43    //Combines inner and outer trees
44    inner->AddFriend(outer);
45    inner->AddFriend(particles);
46    inner->AddFriend(other);
47
48    //Reader for combined tree
49    TTreeReader reader(inner);
50
51    TTreeReaderValue<int> innID = {reader,"ID"};//Track ID
52    TTreeReaderValue<double> innX = {reader,"X"};//Inner detector X coord
53    TTreeReaderValue<double> innY = {reader,"Y"};//Inner detector Y coord
54    TTreeReaderValue<double> innZ = {reader,"Z"};//Inner detector Z coord
55    TTreeReaderValue<int> innEvent = {reader,"Event"};
56
57    TTreeReaderValue<int> outID = {reader,"Outer.ID"};//Track ID
58    TTreeReaderValue<double> outX = {reader,"Outer.X"};//Outer detector X
          coord
59    TTreeReaderValue<double> outY = {reader,"Outer.Y"};//Outer detector Y
          coord
60    TTreeReaderValue<double> outZ = {reader,"Outer.Z"};//Outer detector Z
          coord
61    TTreeReaderValue<int> outEvent = {reader,"Outer.Event"};
62
63    TTreeReaderValue<double> E1 = {reader,"Protons.E1"};//Particle 1 energy
64    TTreeReaderValue<double> E2 = {reader,"Protons.E2"};//Particle 2 energy
65
66    TTreeReaderValue<double> fragX = {reader,"Particles.PBx"};//Fragment x
          Mom
67    TTreeReaderValue<double> fragY = {reader,"Particles.PBy"};//Fragment y
          Mom
68    TTreeReaderValue<double> fragZ = {reader,"Particles.PBz_lab"};//Fragment
           z Mom
69
70    std::vector<std::vector<double>> data; //Contains all particle data
71    std::vector<std::vector<double>> allEnergy; //Contains all energy values
72
73    int c=0;
74
75    //Loops over all data in collected by detector
76    while (reader.Next()){
```

```
77      //Data for each recorded particles (missed hits included)
78      std::vector<double> part = {(double)*innID,//0
79      (double)*outID,//1
80      (double)*innEvent,//2
81      *innX,//3
82      *innY,//4
83      *innZ,//5
84      *outX,//6
85      *outY,//7
86      *outZ};//8
87
88      data.push_back(part);
89
90      //Tree containing energy values only contains 100240 data points
91      if (c <= 100239){
92        std::vector<double> energy = {*E1,*E2};
93        allEnergy.push_back(energy);
94      }
95      c++;
96    }
97
98    //Random number generator
99    gRandom = new TRandom3();
100   gRandom->SetSeed(0);
101   TRandom3 r3;
102   r3.SetSeed(0);
103
104   //Variables for random momentum
105   const double FWHW = 0.01/2.35;  //1% FWHW
106   double p,pRand; //Particle momentum
107   const double proM = 938.272;  //Mass of a proton
108
109   //Beam info (C-12)
110   const double beamM = MA;  //Mass
111   const double beamE = ENERGY;      //Energy
112   const double beamP = momentum(beamE,beamM);     //Momentum
113   TVector3 beam = TVector3(0,0,beamP);  //Momentum 3-vector
114   TLorentzVector lbeam = TLorentzVector(beam,beamE);  //Momentum 4-vector
115
116   //Target info (Proton)
117   const double targetM = proM;              //Mass
118   TVector3 target = TVector3(0,0,0);            //Momentum 3-Vector
119   TLorentzVector ltarget = TLorentzVector(target,targetM);//Momentum 4-
        vector
120
121   //Total input momentum 4-vector
122   TLorentzVector momIn = lbeam + ltarget;
123
124   //Fragment
125   const double fragM = MB;  //Fragment mass
126
127   //Contains momentum for particles pairs for a single event
128   std::vector<TLorentzVector> eventMom;
129   //Momentum 4-vector for a particle
```

```
130   TLorentzVector lmom;
131
132   //4-vector of the missing mass
133   TLorentzVector missingMom;
134   //Missing mass value
135   double missingMass;
136
137   //Number of events where both particles hit both detectors
138   int hitEvent = 0;
139   //Total number of events
140   int event = 0;
141
142   //Loop though all particles in the dataset
143   for(int i=0;i<(int)(data.size()/2);i++){
144     //Checks if ANY particle missed ANY of the detectors
145     if (data[2*i][0] != -1 && data[2*i][1] != -1 && data[2*i+1][0] != -1
          && data[2*i+1][1] != -1){
146
147       eventMom.clear();
148       //Loops though the pair of particles for each event that has no
            misses
149       for (int j=0;j<2;j++){
150
151         //Displacement 3-vector of particle
152         TVector3 mom = {data[2*i+j][6]-data[2*i+j][3],
153                 data[2*i+j][7]-data[2*i+j][4],
154                 data[2*i+j][8]-data[2*i+j][5]};
155
156         //Converts displacement to momentum 4-vector
157         //Checks if particle 1
158         if (data[2*i+j][0]==1){
159           p = momentum(allEnergy[i][0],proM);
160           pRand = r3.Gaus(p,p*FWHW);
161           mom = pRand*mom.Unit();
162           lmom.SetPxPyPzE(mom.x(),mom.y(),mom.z(),allEnergy[i][0]);
163         }
164         //Checks if particle 2
165         if (data[2*i+j][0]==2){
166           p = momentum(allEnergy[i][1],proM);
167           pRand = r3.Gaus(p,p*FWHW);
168           mom = pRand*mom.Unit();
169           lmom.SetPxPyPzE(mom.x(),mom.y(),mom.z(),allEnergy[i][1]);
170         }
171         eventMom.push_back(lmom);
172
173       }
174       //Calculates total final 4-momentum
175       TLorentzVector momOut = eventMom[0] + eventMom[1];
176       //Calculates missing mass 4-momentum
177       missingMom = momIn - momOut;
178       //Calculates missing mass
179       missingMass = missingMom.M()-fragM;
180       std::cout<<"Missing␣Mass:␣"<<missingMass<<std::endl;
181       //Puts value into histogram
```

```
182         missHist->Fill(missingMass);
183
184         hitEvent++;
185       }
186       event++;
187     }
188
189     std::cout<<"Number of hit events: "<<hitEvent<<std::endl;
190     std::cout<<"Number of events: "<<event<<std::endl;
191
192     //Canvas to be drawn on
193     TCanvas* cv = new TCanvas();
194     missHist->Draw();
195     missHist->SaveAs("figs/missing.root");
196
197     //Close files
198     input.Close();
199     quasi.Close();
200 }
```

Main function (p2p.cpp)

```
 1 #include "G4RunManagerFactory.hh"
 2 #include "G4UImanager.hh"
 3 #include "G4VisExecutive.hh"
 4 #include "G4UIExecutive.hh"
 5 #include "G4MTRunManager.hh"
 6
 7 #include "p2pActionInitialization.h"  //Class that generates the actions
       by the particle gun
 8 #include "p2pDetectorConstruction.h" //Class that generates the objects
 9 #include "p2pPhysicsList.h"     //Class that defines the physical
       processes
10
11 int main(int argc,char** argv)
12 {
13
14   #ifdef G4MULTITHREADED
15     G4MTRunManager* runManager = new G4MTRunManager;
16   #else
17     G4RunManager* runManager = new G4RunManager();
18   #endif
19   // construct the default run manager
20   //auto runManager = G4RunManagerFactory::CreateRunManager();
21
22
23   // set mandatory initialization classes
24   runManager->SetUserInitialization(new p2pDetectorConstruction());
25   runManager->SetUserInitialization(new p2pPhysicsList());
26   runManager->SetUserInitialization(new p2pActionInitialization());
27
28     // initialize G4 kernel
29   //runManager->Initialize();
```

```
30
31    //Allows for the use of the visualizer
32      //Initializing visualizer
33      //the G4VisExecutive can take verbose options e.g "new G4VisExecutive
            ("Quite")"
34    G4VisManager* visManager = new G4VisExecutive;
35    visManager->Initialize();
36
37      //get the pointer to the UI manager
38    G4UImanager* UI = G4UImanager::GetUIpointer();
39
40      //Initializes UI Executive object
41    G4UIExecutive* session = nullptr;
42
43      //fills session object if no arguments are given at command line
44    if (argc == 1){ session = new G4UIExecutive(argc,argv);}
45
46      //If session is empty, run in batch mode
47    if (! session){
48      G4String command = "/control/execute␣";
49      G4String macro = argv[1];
50      UI->ApplyCommand(command+macro);
51    }
52      //else, run init_vis macro and start the visualizer
53    else{
54      UI->ApplyCommand("/control/execute␣init_vis.mac");
55      session->SessionStart();
56      delete session;
57    }
58      // job termination
59    delete runManager;
60    delete visManager;
61    return 0;
62  }
```

Physics List (p2pPhysicsList.cpp)

```
1  #include "p2pPhysicsList.h"
2
3  p2pPhysicsList::p2pPhysicsList() {
4
5  }
6
7  p2pPhysicsList::~p2pPhysicsList() {
8
9  }
10
11 void p2pPhysicsList::ConstructParticle(){
12   //G4Alpha::AlphaDefinition();
13   // Construct all baryons
14   G4BaryonConstructor bConstructor;   //pointer to constructor
15   bConstructor.ConstructParticle(); //method that constructs
16
```

```
17    // Construct all leptons
18    G4LeptonConstructor lConstructor;
19    lConstructor.ConstructParticle();
20
21    // Construct all mesons
22    G4MesonConstructor mConstructor;
23    mConstructor.ConstructParticle();
24
25    // Construct all Boson
26    G4BosonConstructor bsConstructor;
27    bsConstructor.ConstructParticle();
28
29    // Construct all Shortlived
30    G4ShortLivedConstructor slConstructor;
31    slConstructor.ConstructParticle();
32
33    // Construct Ion
34    G4IonConstructor iConstructor;
35    iConstructor.ConstructParticle();
36  }
37
38  void p2pPhysicsList::ConstructProcess(){
39    //Allows for tracking
40    AddTransportation();
41
42    //Allows for Ion scattering
43    G4VPhysicsConstructor* scatter = new G4EmStandardPhysics();
44    scatter->ConstructProcess();
45  }
```

Detector Construction (p2pDetectorConstruction.cpp)

```
1   #include "p2pDetectorConstruction.h"
2
3   p2pDetectorConstruction::p2pDetectorConstruction() :
        G4VUserDetectorConstruction(),
4   fScoringVolume(0){
5
6     DefineMaterial();
7
8   }
9
10  p2pDetectorConstruction::~p2pDetectorConstruction() {
11
12  }
13
14  void p2pDetectorConstruction::DefineMaterial(){
15    //Allows for the use of Nist Material list
16      G4NistManager* nist = G4NistManager::Instance();
17
18      //Defines some materials
19      worldMat = nist->FindOrBuildMaterial("G4_Galactic");
20      detectorMat = nist->FindOrBuildMaterial("G4_Si");
```

```
21      emissionMat = nist->FindOrBuildMaterial("G4_lH2");
22
23  }
24
25  G4VPhysicalVolume* p2pDetectorConstruction::Construct() {
26
27    G4bool checkOverlaps = true;
28
29    //World
30    G4double worldx = 500*cm;
31    G4double worldy = 500*cm;
32    G4double worldz = 500*cm;
33
34    solidWorld = new G4Box("World",worldx,worldy,worldz);
35    logicWorld = new G4LogicalVolume(solidWorld,worldMat,"World");
36    physicalWorld = new G4PVPlacement(0,G4ThreeVector(0,0,0),logicWorld,"
        World",0,false,0,checkOverlaps);
37
38    //Spherical calorimeter
39    G4double calMin = 80*cm;
40    G4double calThick = CALTHICKNESS;
41    G4double calMax = calMin+calThick;
42
43    solidCal= new G4Sphere("Calorimeter",calMin,calMax,0,2*M_PI,0,M_PI);
44    logicCal = new G4LogicalVolume(solidCal,detectorMat,"Calorimeter");
45    physicalCal = new G4PVPlacement(0,G4ThreeVector(0,0,0),logicCal,"
        Calorimeter",logicWorld,false,3,checkOverlaps);
46
47    fScoringVolume = logicCal;
48
49    //Emission Surface
50    G4double emiDiameter = 3*cm;
51    G4double emiLength = 3*cm;
52
53    solidEmi = new G4Tubs("Emission",0,emiDiameter,emiLength,0,2*M_PI);
54    logicEmi = new G4LogicalVolume(solidEmi,emissionMat,"Emission");
55    physicalEmi = new G4PVPlacement(0,G4ThreeVector(0,0,0),logicEmi,"
        Emission",logicWorld,false,4,checkOverlaps);
56
57
58
59    //Detector surfaces (CHANGE IN SENSITIVE DETECTOR ASWELL!)
60    G4int numSeg = SEGMENTS;         //Number of segments
61    G4int numRow = ROWS;         //Number of rows
62    G4double len = LENGTH;
63    G4double segLen = len/numRow; //Length of each segment
64    G4double gap = GAP;
65    G4double thick = THICKNESS;
66
67    G4double innMin = INNERRAD;        //Inner radius of segment (5cm)
68    G4double innMax = innMin + thick; //Outer radius of segment
69    G4double innHHight = segLen/2;     //half the length of cylinder segment
70    G4double innStart = 0;         //segment starting angle
71    G4double innEnd = (2*M_PI)/numSeg;  //segment ending angle
```

```
72
73    //Inner cylinder
74    solidInn = new G4Tubs("Inner",innMin,innMax,innHHight,innStart,innEnd);
75    logicInn = new G4LogicalVolume(solidInn,detectorMat,"Inner");
76
77    //Outer cylinder dimensions
78    G4double outMin = innMin + gap;
79    G4double outMax = outMin + thick;
80
81    //Outer cylinder
82    solidOut = new G4Tubs("Outer",outMin,outMax,innHHight,innStart,innEnd);
83    logicOut = new G4LogicalVolume(solidOut,detectorMat,"Outer");
84
85    G4ThreeVector trans;  //Segment translation
86    G4RotationMatrix* rot;  //Segment rotation
87
88    //Loop to generate tube of cylinders in the z axis
89    for(G4int j=0;j<numRow;j++){
90      //Loop to generate full cylinder
91      for(G4int i=0;i<numSeg;i++){
92
93        rot = new G4RotationMatrix();
94
95        //Moving the segment
96        rot->rotateZ(i*innEnd);
97        trans = G4ThreeVector(0,0,segLen*(1-numRow)/2+2*j*innHHight);
98
99        physicalInn = new G4PVPlacement(rot,trans,logicInn,"Inner",
              logicWorld,false,0,checkOverlaps);
100       physicalOut = new G4PVPlacement(rot,trans,logicOut,"Outer",
              logicWorld,false,1,checkOverlaps);
101     }
102   }
103   return physicalWorld;
104 }
105
106 void p2pDetectorConstruction::ConstructSDandField(){
107   //Defining which parts of the geometry will become the sensitive
          detectors
108
109   //HOW I ORIGINALLY SET THE SENSITIVE DETECTORS
110   //logicInn->SetSensitiveDetector(new p2pSensitiveDetector("InnerSens","
          InnerCollection"));
111   //logicOut->SetSensitiveDetector(new p2pSensitiveDetector("OuterSens","
          OuterCollection"));
112
113   //NEW WAY, USING SENSITIVE DETECTOR MANAGER
114   auto innSD = new p2pSensitiveDetector("InnerSens","InnerCollection");
115   G4SDManager::GetSDMpointer()->AddNewDetector(innSD);
116   SetSensitiveDetector("Inner",innSD);
117
118   auto outSD = new p2pSensitiveDetector("OuterSens","OuterCollection");
119   G4SDManager::GetSDMpointer()->AddNewDetector(outSD);
120   SetSensitiveDetector("Outer",outSD);
```

```
121
122
123 }
```

Sensitive Detector (p2pSensitiveDetector.cpp)

```
 1 #include "p2pSensitiveDetector.h"
 2
 3 p2pSensitiveDetector::p2pSensitiveDetector(G4String name,G4String
       collection) : G4VSensitiveDetector(name),
 4 fHitsCollection(0){
 5
 6   collectionName.insert(collection);
 7 }
 8
 9 p2pSensitiveDetector::~p2pSensitiveDetector() {
10 }
11
12 void p2pSensitiveDetector::Initialize(G4HCofThisEvent* hce){
13
14   //G4cout<<"Detector "<<SensitiveDetectorName<<" initialised"<<G4endl;
15
16   fHitsCollection = new p2pHitsCollection(SensitiveDetectorName,
         collectionName[0]);
17
18   G4int hcID = G4SDManager::GetSDMpointer()->GetCollectionID(
         fHitsCollection);
19   hce->AddHitsCollection(hcID,fHitsCollection);
20
21 }
22
23 G4bool p2pSensitiveDetector::ProcessHits(G4Step* aStep, G4TouchableHistory
       *){
24
25
26   //Gets position of particles hitting detector
27   G4StepPoint* preStepPoint = aStep->GetPreStepPoint();
28   G4ThreeVector prePos = preStepPoint->GetPosition();
29
30   //Gets the copy number of hit detector
31   const G4VTouchable* touchable = preStepPoint->GetTouchable();
32   G4int copyNo = touchable->GetCopyNumber();
33
34   //Track related variables
35   G4Track* track = aStep->GetTrack();
36   G4int trackID = track->GetTrackID();
37
38
39
40   //Gets translation and rotation of detector segment that was hit
41   G4VPhysicalVolume* physVol = touchable->GetVolume();
42   G4ThreeVector posDetector = physVol->GetTranslation();
43   G4RotationMatrix* rotDetector = physVol->GetRotation();
```

```
44
45    //Gets actual angle of segment from horizontal
46    G4double theta = (rotDetector->getPsi())*2;
47
48    G4int numSeg = SEGMENTS;         //Number of segments
49    G4double thick = THICKNESS;    //Thickness of tubes (0.5mm)
50    G4double innMin = INNERRAD;      //Inner inner radius
51    G4double gap = GAP;      //Gap between detectors
52
53    G4double outMin = innMin + gap;    //Outer inner radius
54    G4double innRad = innMin+(thick/2); //Inner central radius
55    G4double outRad = outMin+(thick/2); //Outer central radius
56
57    G4ThreeVector detPos; //Calculated detector position
58
59    G4double delta = theta + (M_PI/(numSeg)); //Angle about z axis
60
61    //Decides which radius to use when calculating coordinates
62    switch(copyNo) {
63    case 0:
64      detPos = G4ThreeVector(innRad*std::cos(delta),innRad*std::sin(delta),
          posDetector.getZ());
65      break;
66    case 1:
67      detPos = G4ThreeVector(outRad*std::cos(delta),outRad*std::sin(delta),
          posDetector.getZ());
68      break;
69    default:
70      break;
71    }
72
73    //Gets event number
74    G4int event = G4RunManager::GetRunManager()->GetCurrentEvent()->
        GetEventID();
75
76    //Records the coordinates of the detector segment hit
77    if (trackID == 1 || trackID == 2){
78      //G4cout<<"Event: "<<event<<", Detector: "<<copyNo<<", Particle: "<<
          trackID<<G4endl;
79      p2pHit* newHit = new p2pHit();
80
81      newHit->SetCopy(copyNo);
82      newHit->SetTrack(trackID);
83      newHit->SetPos(detPos);
84      newHit->SetEvent(event);
85      newHit->Hit(trackID);
86
87      fHitsCollection->insert(newHit);
88    }
89    return true;
90  }
91
92  void p2pSensitiveDetector::EndOfEvent(G4HCofThisEvent*){
93
```

```
94    //Number of hits in this event
95    G4int nHits = fHitsCollection->entries();
96
97    //Event number
98    G4int event = G4RunManager::GetRunManager()->GetCurrentEvent()->
          GetEventID();
99    //Detector copy number
100   G4int hcID = G4SDManager::GetSDMpointer()->GetCollectionID(
          fHitsCollection);
101
102   G4cout<<"Detector:␣"<<SensitiveDetectorName<<G4endl;
103   G4cout<<"Event:␣"<<event<<G4endl;
104
105   //Hit counter for each particle
106   std::vector<G4int> tot = {0,0};
107
108   //Empty hit container
109   p2pHit* hit = new p2pHit();
110
111   //Counting number of times each particle hit detector
112   for (G4int i=0;i<nHits;i++){
113     hit = (*fHitsCollection)[i];
114     tot[0] += hit->Getp1(); //Particle 1
115     tot[1] += hit->Getp2(); //Particle 2
116   }
117
118   G4cout<<"particle␣1:␣"<<tot[0]<<",␣Particle␣2:␣"<<tot[1]<<G4endl;
119
120   //Empty hit containers for when particles miss detector
121   p2pHit* missHit1 = new p2pHit();
122   missHit1->SetEvent(event);
123   missHit1->SetCopy(hcID);
124   p2pHit* missHit2 = new p2pHit();
125   missHit2->SetEvent(event);
126   missHit2->SetCopy(hcID);
127   std::vector<p2pHit*> missHit = {missHit1,missHit2};
128
129   //Checks if either particle missed any of the detectors
130   for (G4int i=0;i<2;i++){
131     if (tot[i] == 0){
132       //Adds missed hit container
133       fHitsCollection->insert(missHit[i]);
134     }
135   }
136
137   //Tracker for recording data
138   G4int nump1 = 0;
139   G4int nump2 = 0;
140
141   //Particle ID
142   G4int particle;
143
144   //Number of recorded hits per event
145   G4int recorded = 0;
```

```
146
147   //UPdates number of hits after adding missed hit containers
148   nHits = fHitsCollection->entries();
149
150   //For recording data
151   G4AnalysisManager* manager = G4AnalysisManager::Instance();
152
153   //Loops through each hit in collections
154   for (G4int i=0;i<nHits;i++){
155
156     //Getting hit data individual hit
157     hit = (*fHitsCollection)[i];
158     particle = hit->GetTrack();
159     G4ThreeVector detPos = hit->GetPos();
160     G4int copyNo = hit->GetCopy();
161
162     /*
163      * Only records the data for each particle for their initial
164      * hit to the detector (In case the particle hits the detector
165      * multiple times)
166      */
167
168     //Checks if particle 1
169     if (particle == 1){
170       //Checks if particle 1 hit has already been recorded
171       if (nump1 == 0){
172         G4cout<<"RECORDED␣P1"<<G4endl;
173         recorded += 1;
174
175         //copyNo = 0 (Inner), copyNo = 1 (Outer)
176         manager->FillNtupleIColumn(copyNo,0,particle);
177         manager->FillNtupleDColumn(copyNo,1,detPos[0]);
178         manager->FillNtupleDColumn(copyNo,2,detPos[1]);
179         manager->FillNtupleDColumn(copyNo,3,detPos[2]);
180         manager->FillNtupleIColumn(copyNo,4,event);
181         manager->AddNtupleRow(copyNo);
182       }
183       nump1 += 1;
184     }
185     //Checks if particle 2
186     if (particle == 2){
187       //Checks if particle 2 has already been recorded
188       if (nump2 == 0){
189         G4cout<<"RECORDED␣P2"<<G4endl;
190         recorded += 1;
191
192         //copyNo = 0 (Inner), copyNo = 1 (Outer)
193         manager->FillNtupleIColumn(copyNo,0,particle);
194         manager->FillNtupleDColumn(copyNo,1,detPos[0]);
195         manager->FillNtupleDColumn(copyNo,2,detPos[1]);
196         manager->FillNtupleDColumn(copyNo,3,detPos[2]);
197         manager->FillNtupleIColumn(copyNo,4,event);
198         manager->AddNtupleRow(copyNo);
199       }
```

```
200         nump2 += 1;
201       }
202       //Checks if the particle is a missed hit
203       if (particle == -1){
204         G4cout<<"RECORDED␣MISS"<<G4endl;
205         recorded += 1;
206
207         //copyNo = 0 (Inner), copyNo = 1 (Outer)
208         manager->FillNtupleIColumn(copyNo,0,particle);
209         manager->FillNtupleDColumn(copyNo,1,detPos[0]);
210         manager->FillNtupleDColumn(copyNo,2,detPos[1]);
211         manager->FillNtupleDColumn(copyNo,3,detPos[2]);
212         manager->FillNtupleIColumn(copyNo,4,event);
213         manager->AddNtupleRow(copyNo);
214       }
215     }
216     G4cout<<"RECORDED:␣"<<recorded<<G4endl;
217
218
219 }
```

Hits (p2pHit.h)

```
1  #ifndef CLASSES_P2PHIT_H_
2  #define CLASSES_P2PHIT_H_
3
4
5  #include <G4VHit.hh>
6  #include <G4THitsCollection.hh>
7  #include <G4ThreeVector.hh>
8
9  class p2pHit: public G4VHit {
10 public:
11   p2pHit();
12   virtual ~p2pHit();
13
14   //Print data
15   virtual void Print();
16
17   //Set methods
18   void Hit(G4int particle);
19   void SetTrack(G4int trackID) {fTrack = trackID;};
20   void SetEvent(G4int event){fEvent = event;};
21   void SetPos(G4ThreeVector pos){fPos = pos;};
22   void SetCopy(G4int copy){fCopy = copy;};
23
24   //Get methods
25   G4int Getp1() {return p1;};
26   G4int Getp2() {return p2;};
27   G4int GetTrack() {return fTrack;};
28   G4int GetEvent() {return fEvent;};
29   G4int GetCopy() {return fCopy;};
30   G4ThreeVector GetPos() {return fPos;};
```

```
31 private:
32   G4int p1,p2,fTrack,fEvent,fCopy;
33   G4ThreeVector fPos;
34 };
35
36
37 typedef G4THitsCollection<p2pHit> p2pHitsCollection;
38
39 #endif /* CLASSES_P2PHIT_H_ */
```

Run Action (p2pRunAction.cpp)

```
1  #include "p2pRunAction.h"
2
3  p2pRunAction::p2pRunAction() : G4UserRunAction() {
4    //Creates the tree when the run action is constructed
5    G4AnalysisManager* manager = G4AnalysisManager::Instance();
6    manager->SetNtupleMerging(true);
7    //manager->SetVerboseLevel(1);
8
9    manager->SetFileName("data.root");
10
11   //Data for particles entering inner detector
12   manager->CreateNtuple("Inner","Position");
13   manager->CreateNtupleIColumn("ID");
14   manager->CreateNtupleDColumn("X");
15   manager->CreateNtupleDColumn("Y");
16   manager->CreateNtupleDColumn("Z");
17   manager->CreateNtupleIColumn("Event");
18   manager->FinishNtuple(0);
19
20   //Data for particles entering outer detector
21   manager->CreateNtuple("Outer","Position");
22   manager->CreateNtupleIColumn("ID");
23   manager->CreateNtupleDColumn("X");
24   manager->CreateNtupleDColumn("Y");
25   manager->CreateNtupleDColumn("Z");
26   manager->CreateNtupleIColumn("Event");
27   manager->FinishNtuple(1);
28
29   //Data for the energy deposition into the calorimeter by each outgoing
       proton
30   manager->CreateNtuple("Protons","Energy");
31   manager->CreateNtupleDColumn("E1");
32   manager->CreateNtupleDColumn("E2");
33   manager->FinishNtuple(2);
34
35
36 }
37
38 p2pRunAction::~p2pRunAction() {
39   delete G4AnalysisManager::Instance();
40 }
```

```
41
42 void p2pRunAction::BeginOfRunAction(const G4Run*){
43   G4AnalysisManager* manager = G4AnalysisManager::Instance();
44
45   //Opens the tree for data collection at the start of each run
46   manager->OpenFile();
47
48 }
49
50 void p2pRunAction::EndOfRunAction(const G4Run*){
51   G4AnalysisManager* manager = G4AnalysisManager::Instance();
52
53   //Writes then closes the tree at the end of the run
54   manager->Write();
55   manager->CloseFile();
56
57 }
```

Event Action (p2pEventAction.cpp)

```
1  #include "p2pEventAction.h"
2
3  p2pEventAction::p2pEventAction(p2pRunAction* ) {
4    Edep1=0;
5    Edep2=0;
6  }
7
8  p2pEventAction::~p2pEventAction() {
9
10 }
11
12 void p2pEventAction::BeginOfEventAction(const G4Event*){
13   G4cout<<"EVENT␣START"<<G4endl;
14   Edep1 = 0;
15   Edep2 = 0;
16 }
17
18 void p2pEventAction::EndOfEventAction(const G4Event*){
19   G4cout<<"EVENT␣END"<<G4endl;
20   //G4cout<<"Energy Deposition 1: "<<Edep1<<G4endl;
21   //G4cout<<"Energy Deposition 2: "<<Edep2<<G4endl;
22
23
24
25   //Records final energy deposition by each particle
26   G4AnalysisManager* manager = G4AnalysisManager::Instance();
27   manager->FillNtupleDColumn(2,0,Edep1);
28   manager->FillNtupleDColumn(2,1,Edep2);
29   manager->AddNtupleRow(2);
30 }
```

Stepping Action (p2pSteppingAction.cpp)

```
#include "p2pSteppingAction.h"

p2pSteppingAction::p2pSteppingAction(p2pEventAction* eventAction) {
  EventAction = eventAction;
}

p2pSteppingAction::~p2pSteppingAction() {

}

void p2pSteppingAction::UserSteppingAction(const G4Step* step){
  //Gets track ID
  G4int trackID = step->GetTrack()->GetTrackID();

  //Gets the logical volume that was hit
  G4LogicalVolume* volume = step->GetPreStepPoint()->GetTouchableHandle()
     ->GetVolume()->GetLogicalVolume();

  //Gets the whole constructor object
  const p2pDetectorConstruction* construction = static_cast<const
     p2pDetectorConstruction*> (G4RunManager::GetRunManager()->
     GetUserDetectorConstruction());

  //Gets the scoring volume (calorimeter)
  G4LogicalVolume* fScoringVolume = construction->GetScoringVolume();

  //Only gets the energy deposited into the scoring volume
  if (volume != fScoringVolume) {return;}
  G4double edep = step->GetTotalEnergyDeposit();

  //Checks which particle the energy is associated with
  switch(trackID){
  case 1:
    EventAction->AddEdep1(edep);
    break;
  case 2:
    EventAction->AddEdep2(edep);
    break;
  default:
    break;
  }
}
```

Primary Generator (p2pPrimaryGeneratorAction.cpp)

```
#include "p2pPrimaryGeneratorAction.h"

p2pPrimaryGeneratorAction::p2pPrimaryGeneratorAction():
   G4VUserPrimaryGeneratorAction(),fParticleGun(0){
  G4int nofParticles = 1; //Number of particles
  fParticleGun = new G4ParticleGun(nofParticles);
```

```
6
7  }
8
9  p2pPrimaryGeneratorAction::~p2pPrimaryGeneratorAction() {
10   delete fParticleGun;
11 }
12
13 void p2pPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent){
14
15   //Setting the values for the particle gun
16   fParticleGun->SetParticleDefinition(G4Proton::Definition());
17
18   //Create analysis reader to read data file
19   G4AnalysisReader* reader = G4AnalysisReader::Instance();
20   reader->SetVerboseLevel(1);
21
22   //Read an individual row of the "Particles" tree in the "quasi.root"
        file
23   G4int ntupleID = reader->GetNtuple("Particles","quasi.root");
24   //G4cout<<"ID: "<<ntupleID<<G4endl;
25
26   G4double x,y,z,px1,py1,pz1,px2,py2,pz2;
27
28   //Pick out the wanted values in that row
29   reader->SetNtupleDColumn(ntupleID,"Xint",x);
30   reader->SetNtupleDColumn(ntupleID,"Yint",y);
31   reader->SetNtupleDColumn(ntupleID,"Zint",z);
32   reader->SetNtupleDColumn(ntupleID,"P1x",px1);
33   reader->SetNtupleDColumn(ntupleID,"P1y",py1);
34   reader->SetNtupleDColumn(ntupleID,"P1z",pz1);
35   reader->SetNtupleDColumn(ntupleID,"P2x",px2);
36   reader->SetNtupleDColumn(ntupleID,"P2y",py2);
37   reader->SetNtupleDColumn(ntupleID,"P2z",pz2);
38
39   //Get values
40   reader->GetNtupleRow();
41
42   /*
43   G4cout<<"Origin: ("<<x<<","<<y<<","<<z<<")"<<G4endl;
44   G4cout<<"P1 Mom: ("<<px1<<","<<py1<<","<<pz1<<")"<<G4endl;
45   G4cout<<"P2 Mom: ("<<px2<<","<<py2<<","<<pz2<<")"<<G4endl;
46   */
47
48   //Set reaction origin point
49   fParticleGun->SetParticlePosition(G4ThreeVector(x,y,z));
50
51   //Define and shoot outgoing particle 1
52   fParticleGun->SetParticleMomentum(G4ThreeVector(px1,py1,pz1));
53   fParticleGun->GeneratePrimaryVertex(anEvent);
54
55   //Define and shoot outgoing particle 2
56   fParticleGun->SetParticleMomentum(G4ThreeVector(px2,py2,pz2));
57   fParticleGun->GeneratePrimaryVertex(anEvent);
58 }
```

References

[1] GERHARD JACOB and TH. A. J. MARIS. Quasi-free scattering and nuclear structure. *Rev. Mod. Phys.*, 38:121–142, Jan 1966.

[2] GERHARD JACOB and TH. A. J. MARIS. Quasi-free scattering and nuclear structure. ii. *Rev. Mod. Phys.*, 45:6–21, Jan 1973.

[3] P. Díaz Fernández, H. Alvarez-Pol, R. Crespo, E. Cravo, L. Atar, A. Deltuva, T. Aumann, and et al. Quasifree (p, pn) scattering of light neutron-rich nuclei near $n = 14$. *Phys. Rev. C*, 97:024311, Feb 2018.

[4] J. Mabiala, A. A. Cowley, S. V. Förtsch, E. Z. Buthelezi, R. Neveling, F. D. Smit, G. F. Steyn, and J. J. Van Zyl. Analyzing power and cross section distributions of the $^{12}\mathrm{C}(p, p\alpha)^{8}\mathrm{Be}$ cluster knockout reaction at an incident energy of 100 mev. *Phys. Rev. C*, 79:054612, May 2009.

[5] T. Aumann, C. A. Bertulani, and J. Ryckebusch. Quasifree $(p,2p)$ and (p,pn) reactions with unstable nuclei. *Phys. Rev. C*, 88:064610, Dec 2013.

[6] L. Atar, S. Paschalis, C. Barbieri, C. A. Bertulani, P. Díaz Fernández, M. Holl, M. A. Najafi, V. Panin, and et al Alvarez-Pol. Quasifree $(p, 2p)$ reactions on oxygen isotopes: Observation of isospin independence of the reduced single-particle strength. *Phys. Rev. Lett.*, 120:052501, Jan 2018.

[7] Shoichiro Kawase, Tomohiro Uesaka, Tsz Leung Tang, Didier Beaumel, Masanori Dozono, Taku Fukunaga, Toshihiko Fujii, Naoki Fukuda, Alfredo Galindo-Uribarri, and et al Hwang. Exclusive quasi-free proton knockout from oxygen isotopes at intermediate energies. *Progress of Theoretical and Experimental Physics*, 2018(2), 02 2018. 021D01.

[8] T. Aumann, C. Barbieri, D. Bazin, C.A. Bertulani, A. Bonaccorso, W.H. Dickhoff, A. Gade, M. Gómez-Ramos, B.P. Kay, A.M. Moro, and et al. Quenching of single-particle strength from direct reactions with stable and rare-isotope beams. *Progress in Particle and Nuclear Physics*, 118:103847, 2021.

[9] W-M Yao et al. Review of particle physics. *Journal of Physics G: Nuclear and Particle Physics*, 33(1):398–410, jul 2006.

[10] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, and et al. Geant4—a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 506(3):250–303, 2003.

[11] L L Carter and E D Cashwell. Particle-transport simulation with the monte carlo method. Technical report, U.S. Energy Reasearch and Development Administration, 1975.