RICH 1 SIMULATION USING GEARS

Urh Trinko

September 16, 2022

1 INTRODUCTION AND GOAL

The goal was to make a simulation of the RICH 1 detector in gears. This simulation can help us analyse the distribution of the angle between the beams of light that fall on the PMT plane and the normal of the mentioned plane.

2 ROOT AND GEARS

The main tools I used were root and gears. The former is a program and library, which is used for precise statistical analysis and the latter is an application that allows the user to write Geant4 code in shorter and one-line style. I used Gears to program the geometry and simulate the particles.

It was my first time using these programs so I had some trouble at first, but I slowly got used to them. I should mention a few things that caused me the most problems and could come in handy for future beginners. Regarding root, the most difficult part was the fact that it is made for c++. Because I do not know this programming language, I had to use an extension called PyRoot, so that I could write the code in python. Useful code for beginners would be *myClass.py*. It generally shows which important ROOT commands to import, how to reed a root file and how to iterate through a root.tree and fill a histogram, which you can than display.

A very useful document would be *textgeom.pdf* , which includes data on how to create materials and simple shapes in Gears. A very important thing to remember while using Gears is to always keep track of the number that identifies the geometry. This is the number that you write after the name of the geometry in the line that defines the placement. When the geometry number in the line that specifies a surface is not the same as the geometries numbers that form the surface a problem can occur. I encountered it while trying to make a mirror. Because I did not write the correct numbers, the beam went through the mirror and didn't reflect on it. Useful gears tutorials can also be found *on this site*.

3 CODE ANALYSIS

3.1 Main simulation in gears

As I said the main goal was to make a simulation of the RICH 1 detector. I used a layout of the detector in the literature in order to have roughly the correct scale. In the literature it is written that the container vessel box contains two radiators. It is filled with the C_4F_{10} gas and also contains a small sample of aerogel. In my simulation the the only radiator is aerogel, other than that the box contains vacuum.

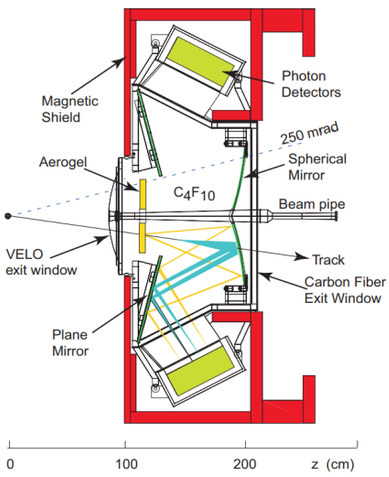

Figure 1: Layout from literature.

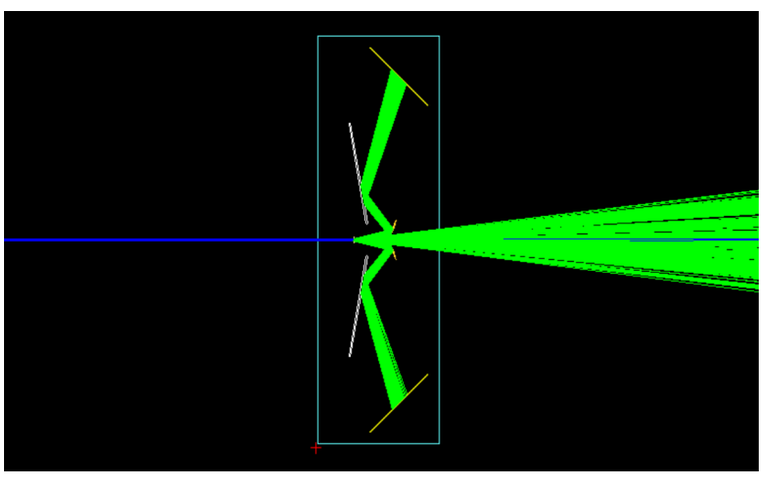

Figure 2: Side view of the completed simulation.

The .mac and .tg files for the simulation can be found *here*. The file *main_detector.mac* can be used to simulate the desired source and save the data in a ROOT tree file. You can also save it to a .heprep file and then view the 3-D simulation using *HepRApp* (this is the application I used to generate the graphic displayed in figure 2). The macro also allows you to edit the source, this means that you can change the type of particle, the way in which the particles are generated (for example

a beam or a point) and their angular/energy distribution. The main .tg file is *main_detector3.tg* and can be found in the folder *geometries*. It includes the geometries of the main components that complete the detector. Mirror 1 is the spherical mirror, the orange component that is closest to the beam path and which first reflects the created light beam (see figure 2). It is made by cutting out a small part of a much larger sphere. The second geometry is under Mirror 3, which is the flat mirror. This is just a reflective flat box. The last main component is the PMT detector, which is also just a box, which I use to analyse the reflected light beam from the mirrors. For each of this components I placed two geometries under "upper" and "lower " corresponding to the two symmetrical parts of the detector. The other geometrical component of the simulation is the aerogel, which is placed on the path of proton beam in the middle of the detector.

The folder *materials* in the folder *geometries* contains many .tg files, which are inputed into *main_detector3.tg*. They describe the properties of the materials and surfaces. They mostly serve the purpose that the light beams abide by the laws of optics.

3.2 Moving the detector components

The literature didn't state the exact orientation and position of the detector components. I placed them according to the shown layout (see figure 1). A problem occurred with the spherical mirror. Because it is actually a part of a sphere, you don't specify the coordinates and tilt of the mirror, but rather of the centre of the sphere. This complicated things because, I didn't know where exactly I was placing the mirror. For this reason I wrote a code, which can be found *here* in the *MirrAndPMT_pos.py* file.

The user can change the position of the main three components by running the command *run_macro(tilt_sph1, tilt_sph2, tilt_flat1, tilt_flat2, tilt_PMT1, tilt_PMT2)*, which takes as an argument the tilt of the spherical mirror, flat mirror and PMT plane, in the mentioned order (upper and lower components are separate, consecutive variables). After this you can also specify the coordinates of the mirror, in the same order as for the tilts. Note that this is not necessary because these arguments have a fixed value, but you can change them if need be. For the spherical mirror you can only specify the position of the edge of the mirror along the z axis, y is always equal 0. But for the other two components you can specify the z as well as the y coordinates of the center of the mirror/plane. However the x coordinate cannot be changed for any of the components and is always equal to 0. The command *run_macro()* changes the necessary angle, position values in the *main_detector3.tg* file and runs the macro code. Note that the code is specific to the radius of the spherical mirror so if you change it in the .tg file you must also change it in the python file.

3.3 Python code for analysis

When I successfully completed the detector and could move its components to desired locations I analysed the results in ROOT. I wanted to analyse two specific parts of the simulation, both on the PMT detector. Firstly I wanted to see the points at which the Cherenkov light beams penetrate the detector plane and secondly I wanted to generate the distribution of the angles between the detector plane normal and the arriving light beams. I accomplished this in the python script *root_analysis.py*. I imported the ROOT tree data generated in gears and looped over the events and step points. For the points on the detector I specified only the events which happened on the PMT plane. I did this by specifying the detector volume copy number, vlm, of the PMT. This is the number which I mentioned as important in the *ROOT AND GEARS* section. By specifying this number I could then fill a histogram with only the points that penetrated the PMT plane. I then displayed the results on a 2d histogram, which showed the x an y local coordinates of the plane (figure 3).

For the angle distribution I wrote a separate file, called *root_anlge.py*, which has the commad *draw_histogram()*. The command is called at the end of *root_analysis.py* file. The *draw_histogram()* function also uses a loop over the ROOT tree to produce a histogram, which is now 1d, and fill it with angles. The process by which it does so is similar to the previous loop, however the code here is a bit more complex. Again I used the information stored in step points of the ROOT file (general information about this can be found here under the section Step point). I used vlm number to get the points at which the light beam hits the upper or lower mirror. I saved this point and then checked if a position exist for the next step point. If this position existed I now had a pair of points which described where the SAME beam reflected of the mirror and then hit the PMT plane. With this point I calculated the directional vector of the beam. I could then use properties of the scalar product to calculate the angle between the mentioned vector and plane normal. All of

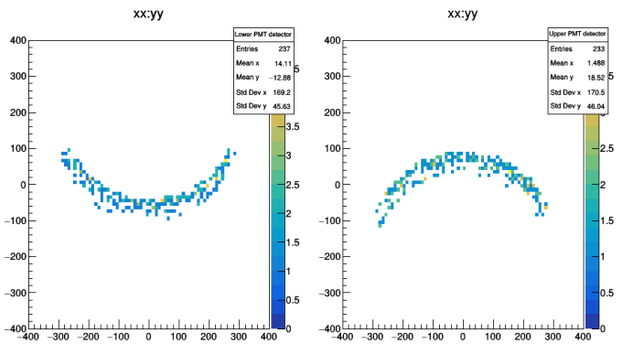

Figure 3: Beams on PMT plane (75°) for the lower (left) and upper detector (right).

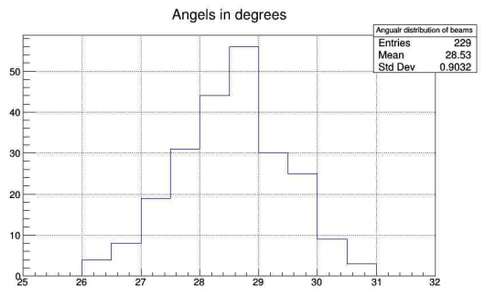

Figure 4: Angular distribution.

these calculations are condensed in the function *angle_BeamPlane()*. It is defined in another file called *subsidiary_code.py*. Here all functions for the calculation of the directional vector, absolute value of vectors, plane normal and scalar product are defined. Using this I created the angular distribution displayed on figure 4.

3.4 Non-beam source

The histograms in figure 3 and 4 represent the situation when the source is a singular beam of protons. This beam falls perpendicular to the Cherenkov radiator (aerogel). In this case the angular distribution is Gaussian around the most common beam angle (figure 4). But we can use the macro file (*main_detector.mac*) to edit the source as mentioned in the subsection *Main simulation in gears*. To make the simulation more realistic I simulated the beams originating from a point and colliding with the aerogel at a certain maximum and minimum angle. This is specified in the lines /gps/ang/mintheta and /gps/ang/maxtheta. You can also set the angular distribution, for example the isometric version would look like /gps/ang/type iso. Furthermore I added two additional sources using the line /gps/source/add intensity (last argument is a number that represents the intensity). The ideas is to have three different sources that radiate three different types of particles.

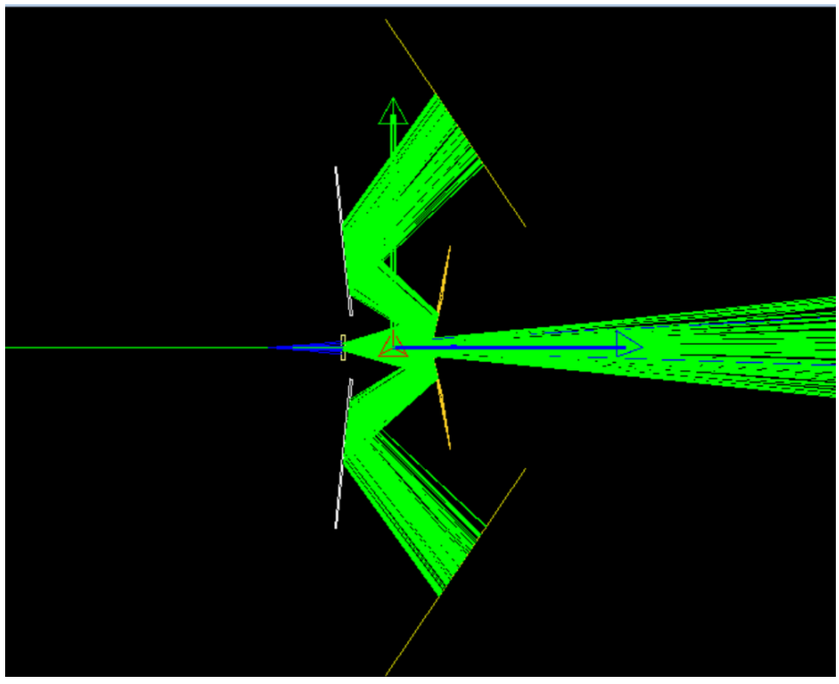

Figure 5: Side view of the detector with three different sources (protons, pi+ and pi-)

As previously mentioned you can also manage the energy distribution here. You can chose between different type of distributions such as /gps/ene/type Mono, Lin, Pow, User... The User command allows you to define your own histogram, to maybe get a more random distibuton. A useful document is also *GPS_manual.pdf*. At the end it contains a table that states all the important /gps macro command lines.

I used the code described in the subsection *Python code for analysis* to display the PMT point and angle distribution, as in the figures 3,4.

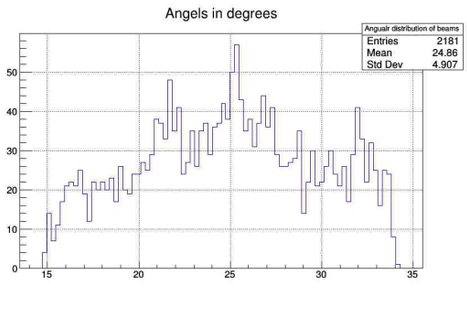

Figure 6: Angular distribution for three sources.

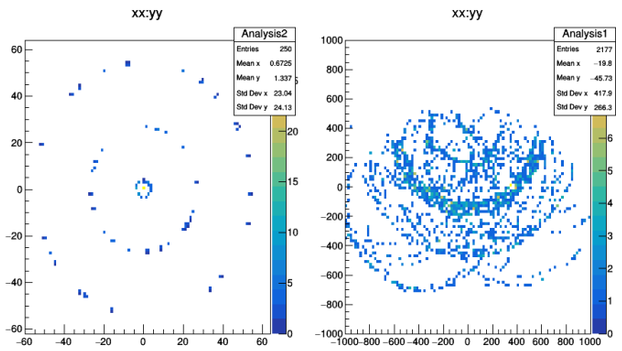

Figure 7: PMT plane for three sources (right) and how the initial particles fall on the aerogel (left).

The differences between figures 3/4 and 6/7 is the consequence of multiple beams at an angle. The angular distribution in figure 6 is not gaussian but seem to have three different peaks. Also you can see multiple potential circles in figure 7. They seem to have different central positions.

In order to make the source settings in the macro file more easily changeable I made a python script called *source_properties.py*. You can change the names in the list properties, which than changes the values of a dictionary that is then used in the function *sources()*. This consequently runs the function *source()* three times, for each source. Using this code you can quickly change the properties of a certain source, such as the type of particle, source and angular distributions. I have not implemented the code to change the energy distribution though. Note that it is very important that the number of lines in the code, macro, always stays the same, meaning that you replace the lines with the same amount of lines. this is necessary otherwise the code *source_properties.py* won't function as it should.

3.5 Other code

I will use this section to describe the code on github that I haven't mentioned so far. The *SphAndFlat_analysis.py* was used to simulate *root_angle.py* multiple times while also using *MirrAndPMT_pos.py* to change the position of the spherical and flat mirror. The purpose of this was to see how the new positon of the mirrors effects the standard deviation of the beam angles. The code *angular_analysis.py* is similar but simpler as you only move the spherical mirror. Lastly *loop_over_files.py* contains code that helps us change certain text files and is imported in alomst very other python script.

4 SYNOPSIS

In conclusion I have made a simple simulation of the RICH 1 detector in gears, which can be changed by any user using the described python scripts. Future options for development can maybe be further avtomatisation of the code, so that you can loop over different positions of the source or it's angular or energy distribution.