TrackG4PS: A Simulation of the PS modules with cosmic rays

Piero Viscone
University of Pisa

GitHub repository

The PS modules are prototypes that will be part of the CMS outer tracker detector during the Run4. These modules are currently being tested in a cryostat acquiring cosmic ray data. This paper presents a simulation of the commissioning with cosmic rays of the PS modules and a toy model for tracking.

1 Introduction

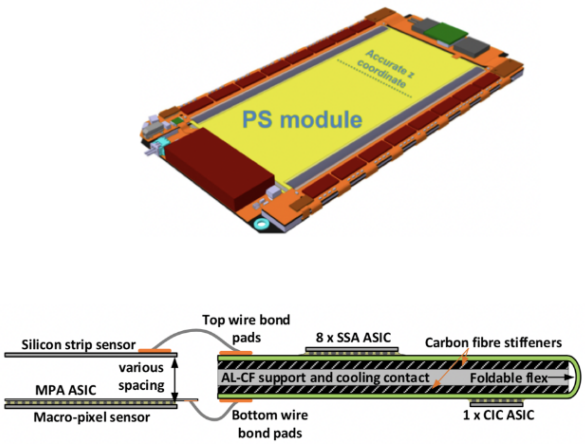


Figure 1: *Assembled PS module (top), front-end sketch (bottom)*

A PS silicon module is made up of one strip sensor with 2 columns of 960 strips with a size of 2.5mm X 0.1 mm and a macro pixel sensor with a matrix of 32 X 960 pixels with a pixel size of 1.5 mm X 0.1 mm. [1]

The strip sensor and the macro pixel sensor are spaced apart by a variable length, from 1 to 4 mm (in the simulation, this value is set to 4mm).

These modules are tested with cosmic rays in a cryostat in a grid of 2 columns and 5 layers.

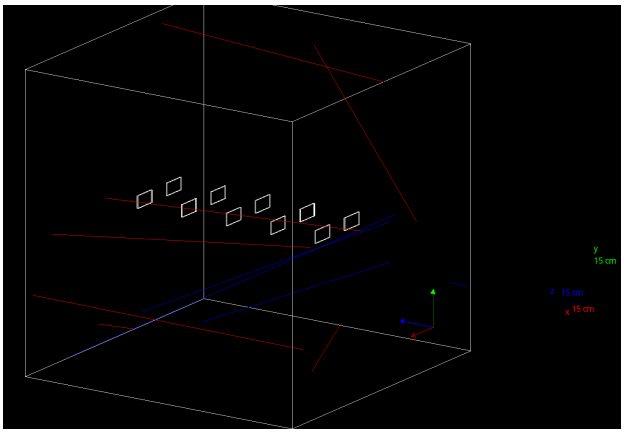


Figure 2: *Geometry of the modules in the cryostat. The particles are impinging along the +z axis (blue in the figure)*

2 Pipeline of the development

This project was developed entirely in C++, and many tools and libraries were used:

- **Geant4**: Framework used to simulate the interaction of particles in the detector
- **ROOT**: Framework used to manage the data and perform the track reconstruction
- **CMake**: Used to automate the generation of the makefile across different systems
- **CCache** (Optional): Used to speed up recompilation by caching previous compilations and detecting when the same compilation is being done again.
- **Catch2**: Library to perform Unit-Testing
- **cppcheck**: Static analysis tool

- **Doxygen**: Used to generate the documentation from the source code
- **Github-Action**: Used to automatically build and deploy the documentation on github-pages
- **cppitertools**: Library to exploit python-like enumerate loops.

Due to the significant size of the dependencies (Geant4 and ROOT), the continuous integration was not implemented because it is impossible to build them at each runner's start. 3 strategies could be implemented to be able to perform the continuous integration:

1. **Self-hosted runners**: Instead of using the machines provided by Github (or any other host), a self-hosted machine (with the dependencies installed) can be used.
   This solution was not implemented due to security issues (A simple pull request can run malicious code on the machine)
2. **Static Linking**: Dependencies can be statically linked to the binaries. This method does not solve the continuous integration problem. The binaries have to be built locally before git-pushing, and the runners can only execute them.
   Furthermore, the binaries will be very large.
3. **Docker Container**: We can create a docker container that contains all the dependencies, so the runner has only to load the container and run the continuous integration in it.
   This is the most convenient solution, but, unluckily, the size of the container will be probably larger than a guaranteed space for a free account (500MB)

Due to these issues, compiling and tests were run only on local machines.

3 Simulation

Geant4 is a toolkit for simulating the passage of particles through matter.
Geant provides many base abstract classes, and the user has to create concrete classes that inherit from these and implement their virtual methods.
All the parameters of the simulation (physical parameters, geometric parameters, trigger parameters, file paths, etc.) were grouped into namespaces in a single header file (UserParameters.hh), so if the user wants to try different configurations, they can do it by editing just one file.

3.1 Physics

In Geant, the user has to manually define the particles and physical processes involved in the simulation.
The abstract class G4VModularPhysicsList provides some standard Physics modules. In this simulation, the module G4EmStandardPhysics was used.
It defines:

- Compton scattering, photoelectric effect, and pair production for photons
- ionization, bremsstrahlung and multiple scattering for e^+, e^- and muons
- positron annihilation
- pair production by muons

To avoid infrared divergences that lead to the production of a lot of low-energy delta rays, a range cut was set (10 cm in air, 0.1mm in silicon).

3.2 Particle Gun

The particles generated by the particle gun are μ^+ and μ^- generated uniformly on a face of the world box and shot along the +z axis according to:

- The muon charge distribution at the sea surface ($\frac{\mu^+}{\mu^-} = 1.3$).
- The angular distribution of cosmic rays at the sea surface $\cos^2(\theta)$
- The energy distribution of cosmic rays at the sea surface between 1 GeV and 1 TeV
  $\frac{dN_\mu}{dE_\mu d\Omega} \approx \frac{0.14 E_\mu^{-2.7}}{cm^2\ s\ sr\ GeV} \left(\frac{1}{1+\frac{1.1 E_\mu \cos\theta}{115 GeV}} + \frac{0.054}{1+\frac{1.1 E_\mu \cos\theta}{850 GeV}} \right)$

These variables are generated using the GetRandom method of the ROOT::TF1 class.
Most of the generated particles do not pass through the detector because the world box is much larger than the detector (the detector is 30 cm x 5 cm x 80 cm, while the world box is 120 cm x 120 cm x 120 cm).
If the world box is smaller, then the tail of the reconstructed angular distribution is suppressed. The most efficient solution would be to convolve the distribution of the parameters of the particles with the geometric acceptance of the detector, but due to the simplicity of the simulation, a brute force approach was preferred, and to obtain more hits, we just generated more particles.

3.3 Read Out

In Geant is possible to define a read-out geometry in parallel to the real geometry of the detector by exploiting abstract classes like G4VReadOutGeometry. Unfortunately, the documentation about these classes is practically inexistent, so a simple ReadOut class was implemented.
Given the position of the modules in the world box and the number and dimension of the strip and pixels, it provides a method that can transform the real position of a hit into the position of the center of the channel that was hit.
To accomplish this task, the algorithm is as follows:

1. find the position of the closest module to the hit on a given axis
2. translates the hit position in the frame of the module

3. calculate the position of the center of the channel that was hit in the module frame
4. translates back to the world frame

This is applied to the 3 axes separately.

3.4 Trigger

The trigger implemented is very simple:

- **Pulse height discrimination:** Rejects all hits with released energy below a given threshold (40 keV). The main scope of the pulse height discrimination is to limit the presence of delta rays' hits in the data.
- **Coincidence**: Accept only events with hits in at least two different PS module layers.

3.5 Event data structure

The data are saved in a root file. The TFile has one TTree with a branch in which a custom Event object is saved. The structure of the Event class is the following:

Event
- eventID
- detectorData
  - TrackID
  - ParticleID (0 μ^-,1 μ^+,2 e^-,3 e^+,4 γ)
  - EnergyDeposited (keV)
  - posX (position of the channel hit in mm)
  - posY
  - posZ
  - Layer (odds are pixels, evens are strips)
- truthBeamData
  - posX (position of the particle gun)
  - posY
  - posZ
  - phi (polar angle)
  - theta (azimuthal angle)
  - energy (GeV)
  - particleID (0 μ^-, 1 μ^+)

.

The Event object contains a DetectorData and a TruthBeamData object.
The Detector data objects contain std vectors filled with data from hits in the detector, while the TruthBeamDataObject contains the data of the beam generated by the particle gun.

3.6 Data management and multithreading

Geant4 provides the class G4VAnalysisManager, a thread-safe wrapper for some simple ROOT classes and methods.
It creates a different file for each worker thread so the threads can work in parallel on different data avoiding any conflict due to data races.
Unfortunately, the G4VAnalysisManager is very inconvenient to use to create and manipulate trees and custom objects, so a DataManager and DataManagerMT classes were created to manage the TFiles, and ROOT was used to fill them with the Event objects.

In Geant, the singleton is the most used design pattern for sharing objects among different classes.
Leaving aside the considerations about this choice, the real pain about singletons comes when they have to deal with thread safety, but for consistency, also DataManager and DataManagerMT are singletons.

DataManager is a singleton that contains a TFile, a TTree, and an Event class object. The strategy to maintain thread safety is the same as Geant: each thread works on a different file. Nevertheless, DataManager is a singleton, so we cannot have multiple TFile. For this reason, we defined DataManagerMT as a friend class of DataManager.
The role of DataManagerMT is to create an std vector containing multiple instances (one each thread) of DataManager (it can do it because it is a friend class of DataManager so it can call the private constructor).

At the beginning of the run, the master thread creates the DataManagerMT instance, and then each worker thread gets the pointer to the DataManagerMT object that allows it to get the pointer to its own DataManager object. In this way, thread safety is accomplished.

3.7 Results

At the end of the simulation, 500.000 particles were generated, and the trigger saved 1819 events.
The big difference between these two numbers is due to different reasons:

- The dimension of the world box (discussed in subsection Particle Gun)
- The large energy threshold set (40keV)

The root file containing the results is output.root. It is in the data folder of the repo In fig 3 and fig 4 there are shown as an example the energy deposited in the detector and the x coordinate of the channel that was hit.

4 Track reconstruction

To reconstruct the tracks, a simple toy model was used to fit two straight lines in the XZ and YZ projection planes.
The initial parameter of the fit is computed simply by sorting the hits along the z-axis and computing the linear function's two parameters, considering only the first and the last point.

Then the fit is performed using the TGraphError class, considering as error bar the pitch of pixels/strips divided by $\sqrt{12}$.

Despite the trigger in the simulation, delta rays' hits are still present.

4.1 Structure of the data

The root file containing the results of the fit is made up a simple NTuple structured as follows:

```
fitResults
├── evID (Id of the reconstructed event)
├── nHit (num of hits)
├── x0
├── x0_err
├── mx
├── mx_err
├── chi2zx (reduced chi2 of the ZX fit)
├── y0
├── y0_err
├── my
├── my_err
└── chi2zy
```

.

The fitted functions are $z = m_x x + x_0$ and $z = m_y y + y_0$ in the frame of the world box.
The length units are millimetres.

The fit_results.root file and the plots of the fits are stored in the data folder of the repo

For example, in fig 6, fig 5 there are shown the best-fit parameters in the XZ projection. In figshown ths plot of a well-fitted event, while in fig7 is shown the plot of an event affected by the presence of delta rays.

5 Conclusion

In conclusion, this is just a basic simulation, but it could be a good starting point to develop more complex simulations accounting for other aspects of the detector (efficiency, radiation hardness, etc.) to characterize the PS modules prototypes of the future CMS outer tracker.

Bibliography

[1] Alessandro La Rosa. "The Upgrade of the CMS Tracker at HL-LHC". In: *Proceedings of the 29th International Workshop on Vertex Detectors (VERTEX2020)*. Journal of the Physical Society of Japan, 2021. DOI: 10.7566/jpscp.34.010006. URL: https://doi.org/10.7566/JPSCP.34.010006.

Images

Simulation

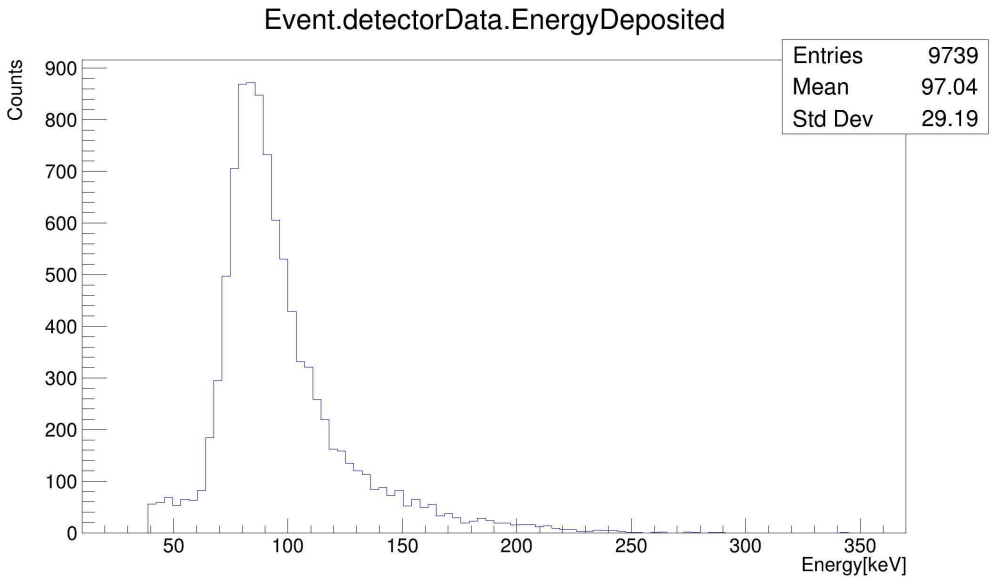


Figure 3: *Energy released in the detector by each hit. The cut at 40 keV is due to the energy threshold set*

.

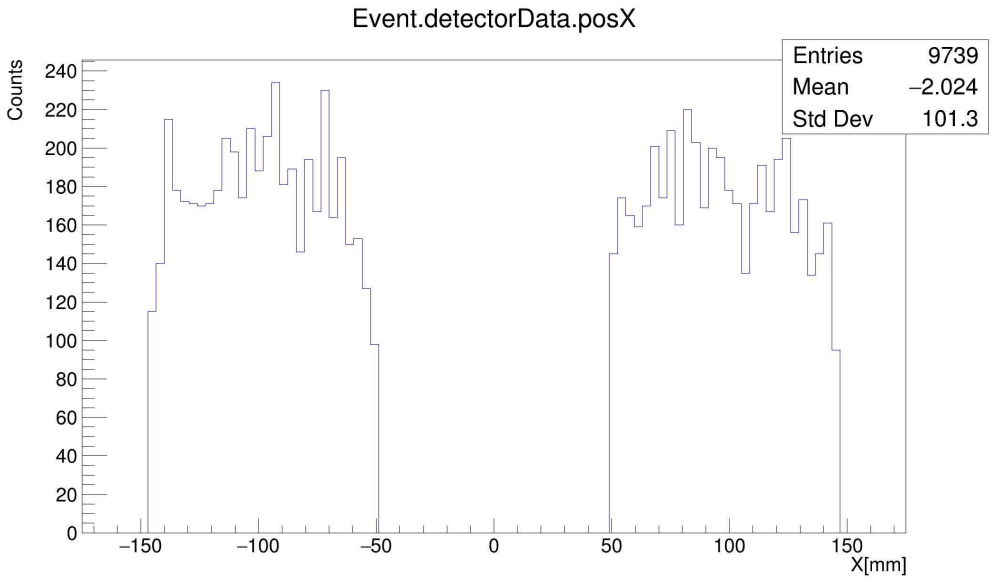


Figure 4: *Position of the hits along the x-axis*

5.1 Fit

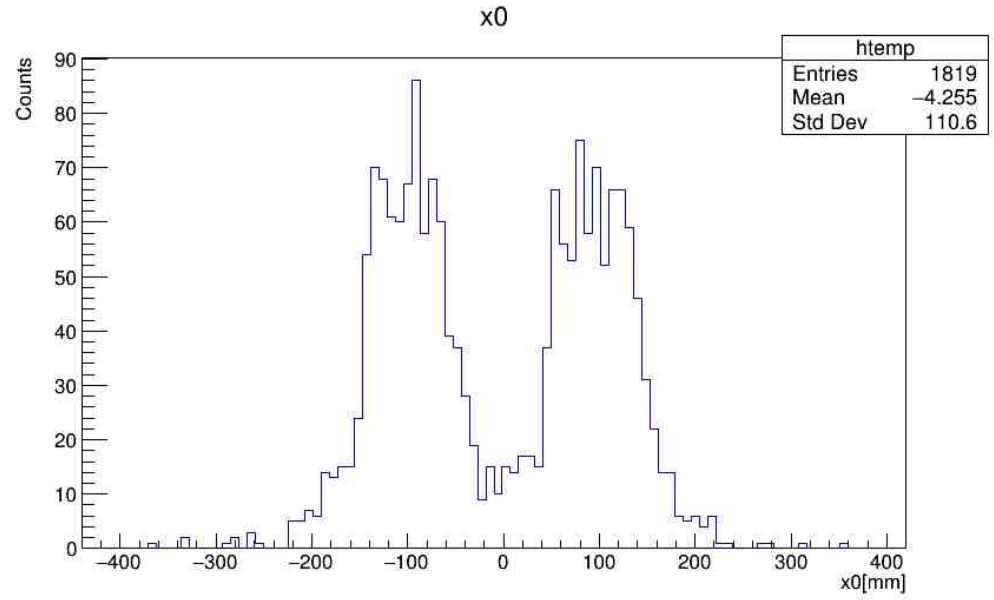


Figure 5: *Fitted x_0 parameter*

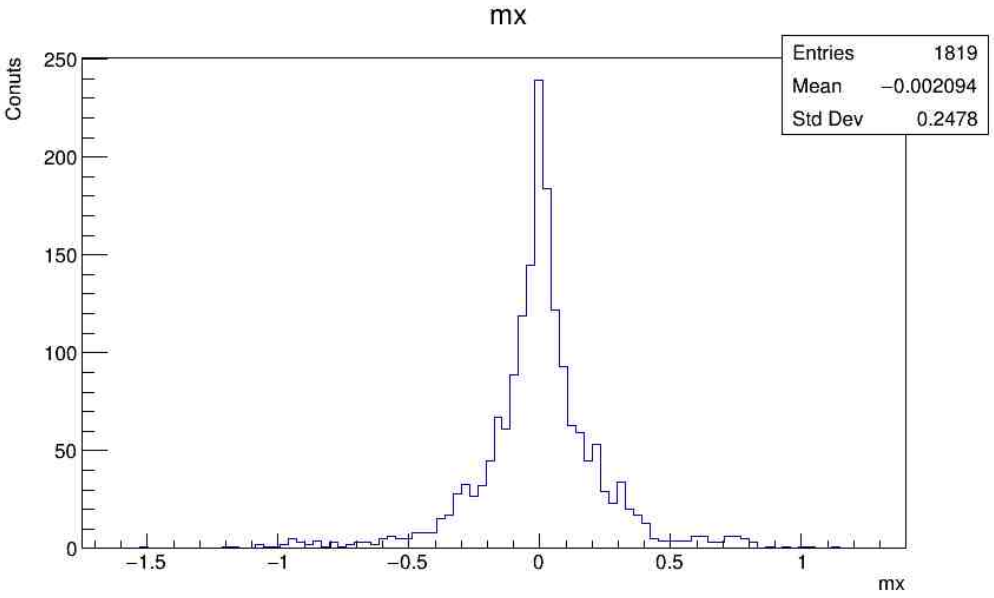
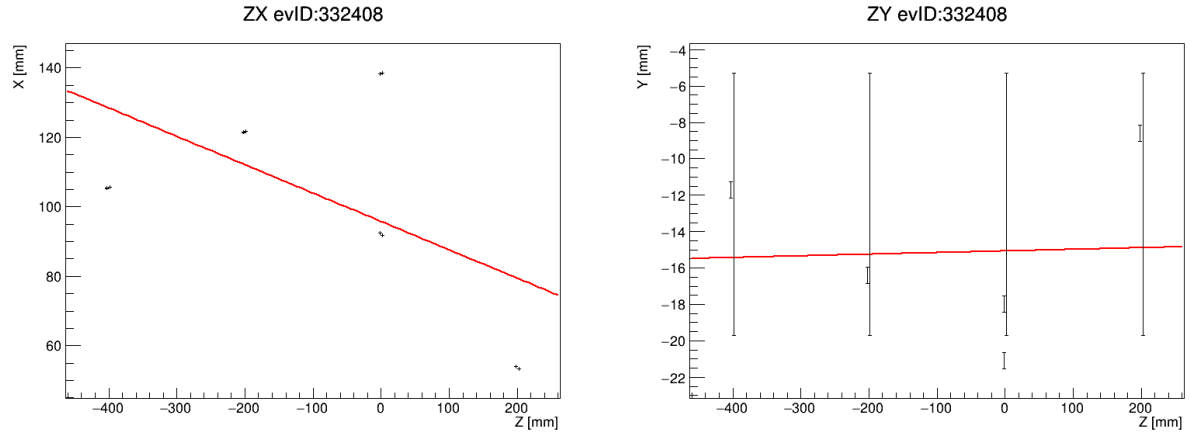
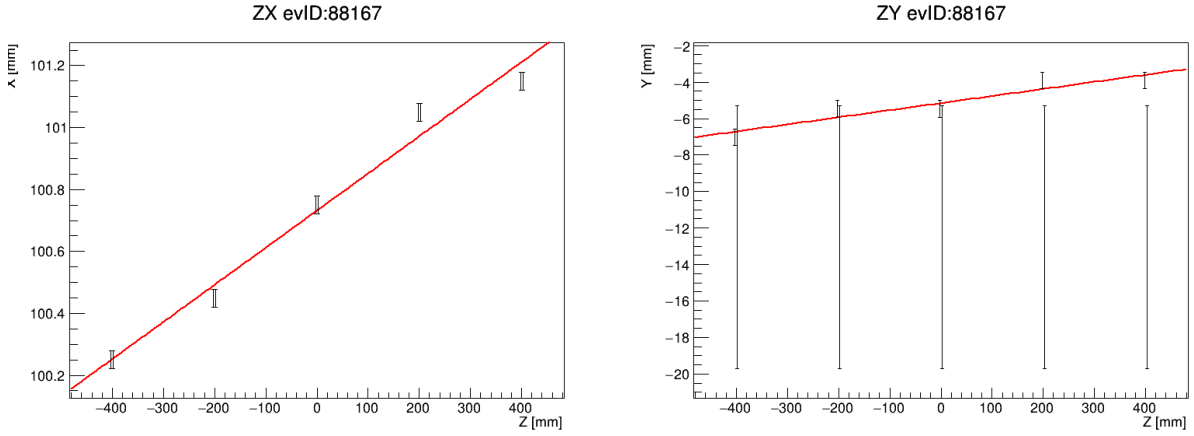


Figure 6: *Fitted m_x parameter*



(a) (b)

Figure 7: *Event affected by the presence of delta rays' hits*



(a) (b)

Figure 8: *Well fitted event. Multiple scattering clearly visible*