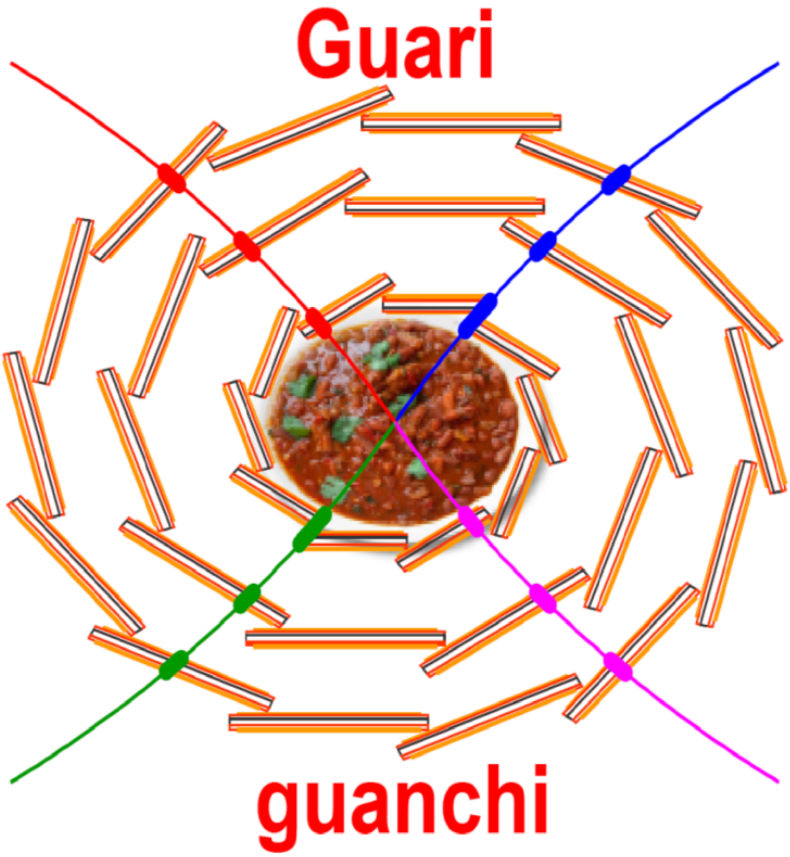

User Manual

A. Pérez Pérez

PICSEL Group, IPHC - CNRS
23 rue du Loess 67037 Strasbourg, France

November 2017

Abstract

GUARIGUANCHI is a tool for a simplified calculation of tracking performances which is fast and can be used as an intermediate step in the process of tracking system optimization. It allows to quickly compare different detector geometries as well as combinations of sensor performances leading to a reduced set of configurations for deeper studies with full-simulation. The tool also allows to optimize beam-test setups in terms of telescope pointing resolution at the device under test location. This document is intended to present the basis to install and start using GUARIGUANCHI. The document also provides all the information about the code structure and presents guide-lines on how to extend it.

Contents

1 Introduction

GUARIGUANCHI is a package created in the context of the ILD collaboration [1] and managed by the PICSEL group at IPHC (Strasbourg). It actually started as a tool for calculating telescope's pointing resolution at device under test (DUT) position for beam-test setups. It then quickly transformed for the more challenging task of tracker optimization by comparing tracking performances of different detector configurations.

The package has a set of tools which allows to calculate for a given momentum (p, θ, ϕ) the track parameters covariance matrix. It also allows to calculate a tracking efficiency with simplified pattern recognition algorithms. The details of the calculations are explained in section 3. Of course, these calculations are performed with simplifying assumptions and ignore other effects that can only be taken into account with full-simulation and realistic pattern recognition algorithms. Detector optimization can be guided by plotting the tracking performances as a function momentum, polar and azimuthal angles for the different detector configurations.

1.1 General philosophy

GUARIGUANCHI is a C++ software meant as a lightweight package in which only `ROOT` [2] pre-installation is required. During installation an executable is produced which is later run with a configuration data-card. This data-card contains all the configuration variables for the analyses to be performed on a list of geometries. The work-flow is as follows,

- Readout of the analysis configuration variables, *e.g.* particle type and origin, momentum ranges in terms of (p, θ, ϕ), analysis variables, and list of geometry data-cards (geo-datacard).
- Geometries build-up. The geo-data-cards are readout to build a list of geometries.
- Each geometry has a so-called tracker associated to it. This object is the one able to perform all the tracking performances calculations related with the geometry, *e.g.* track intersections with geometry (including sensitive and insensitive elements), space points covariance matrix (including multiple scattering and energy loss fluctuations), track parameters covariance matrix and tracking efficiency.
- Depending of the analysis configuration, different plots can be produced: geometry visualization (including some tracks if specified), track parameters resolution and/or tracking efficiency vs (p, θ, ϕ).

The resulting plots are always stored in a multiple pages pdf file, and can if requested be stored in an output root file as well.

2 Getting Started

2.1 Installation

GUARIGUANCHI has been fully tested with `ROOT` version 5.34/32 and is expected to work with higher versions. To install GUARIGUANCHI you will need to go to the following

website: FFFFF. By clicking at the XXX link you will download a `guariguanchi.tar.gz` file. Unpack it with the following command,

```
>$ tar -xzvf guariguanchi.tar.gz
```

Then go to the `Guariguanchi` directory and simply do,

```
>$ ./Compile.sh
```

It should produce a list compiled objects (`.o` files) inside the `lib/` directory, as well as an executable inside the `bin/` directory called `GuariguanchiApp`. If you get some compilation errors you maybe need to verify the options during your `ROOT` build-up.

2.2 Playing with some examples

Once the package is compiled it is quite simple to use. You just need to get a data-card (`datacard.txt`) and run GUARIGUANCHI as follows,

```
>$ ./bin/GuariguanchiApp datacard.txt
```

Now let's play with some examples and discover the package in the process.

2.2.1 Example-1: Beam-test

This example consist on a classic beam-test configuration, in which only straight tracks will be considered (no magnetic field). What is mainly needed in this analysis is the telescope pointing resolution at the DUT's positions. All the data-cards for this example can be found at the directory,

`DataCards/Examples/BeamTest_Example/`.

Lets open the main data-card, `BeamTest_datacard.txt`, and take a look at it. The configuration is specified with a list of parameters blocks which will be described here below (for a complete description of all the possible parameters specified in the main data-card see sec. 5).

- `ParticleType:` in the present case we consider an electron (`e-`).
- `ParticleOrigin:` point from which the particle will be tracked in the process of geometry navigation. Current value is $(0, 0, -1)$ cm.
- `ReferencePoint:` pivot-point for trajectory parameterization. Current value is $(0, 0, 10)$ cm.
- The momentum values are specified as a set of (p, θ, ϕ) values. In the current case
  - p-values: range inside the `BeginMomentumScan` and `EndMomentumScan` bock $\Rightarrow$ 30 bins between 1 and 40 GeV/c.
  - θ-values: set of discrete values $\Rightarrow$ single value of 0 deg.
  - ϕ-values: set of discrete values $\Rightarrow$ single value of 0 deg.

- `TrkResolAnalysisParams` block. In the present case we are going to study the telescope resolution at DUT positions. The configuration for this analysis is specified inside the block between `BeginTrkResolAnalysisParams` and `EndTrkResolAnalysisParams`. Only three parameters are specified in this case.
  - `NhitsMin`: minimum number of hits for a track. In this case it is set to 2 (minimum number of hits to reconstruct a straight line).
  - `SameRange`: flag to use same vertical range for all plots. Set to `false` in this case.
  - `UseAllMomVals`: flag to use all specified momentum values for additional plots. Set to `true` in this case.
- The list of geometries is specified inside the block between `BeginGeometries` and `EndGeometries`. In there the list of geo-data-cards is specified. In the current case four geometries are given.
- `OutputFile:` output file generic name (no extension). The output of the program will generate a set of output files with the specified generic name plus the corresponding extension, *e.g.* `.pdf` and/or `.root`.
- Finally a set of flags will determine the output plots and files. In the current example we have,
  - `PrintGeometry:` print-out of all specified geometries.
  - `PlotGeometry:` visualization of all specified geometries.
  - `PlotWorldVolume:` if `PlotGeometry` is set to `true` will also visualize geometry's world volume.
  - `DoTelescopeAnalysis:` flag to perform a Telescope analysis.
  - `SavePlots:` flag to save plots in a `.root` output file.

Now lets run this example as follows,

```
>$ ./bin/GuariguanchiApp fullpath/BeamTest_datacard.txt
```

where `fullpath` is the full path to the main data-card location. The execution should take about 20 seconds. You should see a lengthy printout with all the geometries (`PrintGeometry` set to `true`). It also produced two output files, one `.pdf` and one `.root`, inside the `Plots/Examples/BeamTest_Example/` output directory. Lets open the `.pdf` file. This is a multi-page file with several plots.

The first four pages is a visualization of the four specified geometries. The geometries are projected in the $Z-Y$, $Z-X$ and $X-Y$ planes. We can see 9 yellow elements, which are silicon planes. Six of them are the telescope planes (big ones located between $(0, 10)$ cm and $(10, 20)$ cm in Z). The smaller planes are DUT-1,2 and 3, located at $Z = 0, 10, 20$ cm, respectively. The red-dotted line represents the geometry's world volume, which in this case is a box. It can be seen that the only change from one geometry to the next are the locations of the telescope planes.

The 5th page shows the track's parameters resolution vs momentum for the only $(\theta, \phi) = (0, 0)$ deg value specified. As there is no magnetic field the trajectories are straight

lines, needing only four parameters defined at the pivot point: $\tan(\alpha_x)$, x_0, $\tan(\alpha_y)$ and y_0. The color code is described in the legend at the bottom-right part of the page, each color corresponding to one of the specified geometries. The momentum-dependence of tracking performances is due to the multiple-scattering (MS), which decreases as momentum increases.

The last three pages show the telescope pointing resolution at the DUT positions vs momentum. The plots on the top show the telescope pointing resolution on the DUT-plane local U and V coordinates, top-left and top-right plots, respectively. The bottom-left plot shows the $1-\sigma$ area S (taking correlations into account) of the telescope's pointing resolution. Finally, the bottom-right plot shows the number of background hits in this S surface calculated using the DUT's readout time $t_{\text{r.o.}}$ and background level R_{bkg} (hits per unit time per unit surface) as follows,

$$\text{N}_{\text{bkg}} = S \times t_{\text{r.o.}} \times R_{\text{bkg}}. \tag{1}$$

This last quantity is useful in studying the probability of wrong hit association in tracking pattern recognition.

Just to finalize discussing this example lets explore the geo-data-cards. All the data-cards are inside the directory

`DataCards/Examples/BeamTest_Example/GeometryCards/`

All are very similar, so it will be enough to take a look of just one of them, *e.g.* `BeamTestGeometry_v1.txt`. It starts with the specification of the geometry name `GeometryName:`, followed by a set of input files (`InputFile:`) in which different aspects of the geometry are specified. The content of the input files could directly be written inside the geo-data-card, but this modularity has the advantage to simplify the reading of the geometry as well as allows to specify just once aspects shared by different geometries.

In this example each geo-data-card has four input files, three of them being common to all. The content of the input files is the following,

- **Geometry's world volume:** It is a volume which contains all the elements of a geometry. In the current case it is just a box located at the origin and with widths $(W_x, W_y, W_z) = (3, 4, 22)$ cm.

- **DUT planes:** this file contains the set of DUT planes. Each one is a geometry-object called `GeoPlane`, corresponding to so-called "planes", which are boxes with one of the dimensions being small compared to the others two. The variables inside the block between `BeginGeoPlane` and `EndGeoPlane` are self evident. More details will be given in sec. 4. Just note that the in this file there are specified three DUT planes, with thicknesses of 50 μm, widths of $(W_u, W_v) = (1, 1)$ cm, parallel to the $X-Y$ plane and located at $(X, Y) = (0, 0)$ and $Z = 0, 10, 20$ cm, respectively. For the beam-telescope analysis one important variable is the `LayerName`, which will be used to identify the planes as DUTs.

- **Telescope planes:** each element in this input file is a `GeoPlane` object. Six planes are specified with thicknesses of 50 μm, widths of $(W_u, W_v) = (2, 3)$ cm, parallel to the $X-Y$ plane and located at $(X, Y) = (0, 0)$ and $Z = 1, 3, 5, 15, 17, 19$ cm, respectively. As in the case of the DUT planes, one important variable is the `LayerName`, which will

be used to identify the telescope planes. The other geometries differ in the position of the telescope planes.

- **Beam-Test configuration:** in this file the telescope and DUT planes are identified. It is done by specifying inside the block between `BeginBeamTestConfiguration` and `EndBeamTestConfiguration` the list of telescope planes (`TelescopeLayersList`) and DUT planes (`DUTLayersList`). This is how the analysis knows which planes are going to be used for tracking (telescope) and which ones considered as DUTs. If there is any other geometry-object not belonging to the telescope or DUT list, it will be considered as a insensitive element only contributing to multiple-scattering.

The output `.root` file contains a set of canvases and `TGraph` objects. By their naming it is self evident what they are. We strongly recommend the user to open this file with `ROOT` and plot its contents.

Something that should have been noticed by now is that GUARIGUANCHI has a unit system. This is why all the specified quantities have to be accompanied of the corresponding units. Furthermore, in the data-cards files all contents after a double-slash "//" are considered as commentary and not readout.

2.2.2 Example-2: All-silicon vertex detector & tracker

This example considers an all-silicon vertex detector and tracker with cylindrical detection layers similar to the design of the ALICE Inner-Tracking-System upgrade [6]. The system is inside a solenoidal magnetic field of 0.5 Tesla along the z-axis. In this case particles will describe Helix trajectories, which can be described with 5 parameters. In this package we use the same parameters as in Belle [3]: d_ρ, d_z, ϕ_0, $tan\lambda$ and signed p_t. All the data-cards for this example can be found at the directory,

`DataCards/Examples/VertexDetector_and_Tracker_Example/`.

As you can notice, there are four main data-cards. Each one of them will serve to illustrate different analyses that can be performed with this package, and will be discussed in the following.

Geometry visualization

Lets start with a simple visualization of the geometries. For this lets open the data-card with the name

`VertexDetector_and_Tracker_datacard_GeometryVisualization.txt`.

Much of the parameters in this file have already been described in the previous example (c.f. sec. 2.2.1). In this case we consider μ^+ as primary particles created at the origin, the pivot point is also fixed at the origin, and the momentum variables are specification in the same way as in the previous example. Two geometries are considered here. In order to visualize them just execute `./bin/GuariguanchiApp` with this data-card. The list of analysis flags set to `true` in the main data-card should give an idea of what is going to be the output of this command,

- `PrintGeometry:` print out of the geometries.
- `PrintGeometryWeight:` print out of the geometries weight. As can be seen, the weight are separated by systems. The total weight is also printed-out.
- `PlotGeometry:` a visual representation similar to the previous example will be created. If this variable is set to `true`, other options can be considered.
  - `PlotWorldVolume:` a visual representation of the geometry's world volume will be created.
  - `DoRZGeoRepresentation:` a $R-Z$ representation of the geometry will be created. This representation is useful for geometries with cylindrical symmetry with respect to the z-axis.
  - `PlotSomeTracks:` for each value of (θ, ϕ) specified a set of particle's average trajectories (no MS) as well as their intersections with the geometry will be displayed.
- `SavePlots:` as in the previous example, a `.root` file will be created.

Given the options above, a `.pdf` and `.root` files are crated. The `.pdf` file is again a mult-page file with several plots. Pages 1 and 2 show a visualization of the first geometry specified (Vertex & Tracker Setup v1). Page 1 shows a visualization on the $Y-Z$, $X-Z$ and $X-Y$ planes of the geometry. Page 2 shows a $R-Z$ representation of the geometry. It can be seen that this geometry is mainly made of cylinders of different radii and lengths. The innermost one in green is a representation of a beam-pipe, made of beryllium. The yellow cylinders are the layers of the vertex detector and tracker, seven 50 μm thickness silicon layers. Finally, the dotted-red line is a representation of the geometry's world volume. Pages 3 and 4 show similar visualizations for the second geometry specified (Vertex & Tracker Setup v2). The difference of this last geometry with the first one is the radius and length of the 3rd tracker layer.

Pages 5-10 show a display of the first geometry with a set of track superimposed. Each set of plots corresponds to a fixed value of the specified (θ, ϕ) (see plot titles). The different colors correspond to the momenta specified in the legend box, which is a set of 10 values between the minimum and maximum specified momenta in the main data-card. The intersections of the tracks with the geometries are also shown (filled-circular-dots). Pages 11-16 show similar plots for the second geometry.

The output `.root` file contains a set of canvases with the same geometry representations, which can be manipulated by the user in order to better understand the geometry as well as tracks intersections with geometries.

This geometry visualization feature is very useful in the process of building a geometry, as well as to understand the tracking performances, as it depends on the track intersections with sensitive layers (number of hits) and insensitive elements (material budget) of the geometry elements.

Before exploring the other analyses data-cards, lets take a look of the geo-data-cards which are inside the `GeometryCards` directory. The configuration files of both geometries are very similar, so it will be enough to take a look of one of them, *e.g.* `Si_Tracker_Geometry_v1.txt`.

The first parameter specified is the geometry name (`GeometryName:`). As in the previous example, there is a set of input files, each one describing one feature of the geometry,

- **Geometry's world volume:** in this case it is a cylinder centered at the origin and with radius and length of 50 cm and 170 cm, respectively.

- **Magnetic field:** a constant magnetic field of magnitude of 0.5 Tesla and along the positive z-axis.

- **Beam-pipe:** a cylinder (`GeoCylinder` object) made of beryllium centered at the origin, with radius, length and thickness of 1.96 cm, 20 cm and 0.8 mm, respectively. Notice that the `LayerName` has been set to "Beam-Pipe". It will be used later to define the different systems of the geometry.

- **The Tracker:** a set of 7 cylinders (`GeoCylinder` objects), all made of silicon with 50 μm thickness at different radii and lengths. All of them also share the same single point resolution (`ResolutionU` and `ResolutionV`), readout time (`ROtime`) and detection efficiency (`Efficiency`) of 4 μm, 10 μs and 99 %, respectively. Notice the value of the `LayerName` variable. It will be used later to setup a telescope-DUT configuration as well as to define the different systems of the geometry.

- **Telescope-DUT config:** this file is similar to the corresponding one of the previous example. In this case the innermost layer is considered as DUT and all the rest as telescope.

- **Systems definition:** this is a new feature in which different "layers" can be put together to define a system. In this example there are three systems defined,

  - Beam-Pipe system: including the beam-pipe;
  - Tracker-Inner Barrel: including the three innermost tracker layers;
  - Tracker-Outer Barrel: including the four outermost tracker layers.

  This system definition can be useful to perform material budget analyses as well as performing cut for good tracks when doing efficiency calculations.

- **Track-Finding lagorithms:** this file contains all the needed information for the tracking efficiency calculation. A list of so-called track-finder algorthms will be specified for different regions in the particle's origin and momenta. A track finder algorithm is specified in the block between `BeginXXXTrackFinderAlgo` and `EndXXXTrackFinderAlgo`, where `XXX` refers to a given track-finder algorithm. In the current case it is used the FPCCD track-finder (`FPCCDTrackFinderAlgo`), one of the pattern recognition algorthms used by the ILD collaboration [4].

  The region where this track-finder will be applied is defined in the block between `BeginTrackFinderRegion` and `EndTrackFinderRegion`. In the current case it corresponds momentum θ range between $(25.0, 110.0)$ deg.

  The "parameters" of this tracking finder are listed below,

  - `NhitsMin`: minimum number of hits.
  - `PtMin`: minimum transverse momentum cut for track-seeding.
  - `CenterPosition:` coordinates of the track-center used for the minimum track-seeding cut.

- `PurityMin`: minimum track purity, defined as the ratio (# good hits)/(# total hits). This allows for a number of so-called fake hits associated to the track.
- `NfakesMaxSeeding`: maximum number of fake hits for track-seeding. This parameter can take either the values 0 or 1.
- `Chi2OndfSeed`: χ^2 cut for seed tracking.
- `Chi2OndAdd`: χ^2 cut for hit-track association in the process of inward tracking.
- `InwardTracking`: bool parameter to define the tracking direction, either inward or outward.
- NmcSeedEffic: number of samplings for a MC calculation of the seeding probability.

The list of geometry elements to be considered in the tracking efficiency calculation is defined by specifying a list of systems inside the `BeginSystems` and `EndSystems` block. This allows to use a sub-set of the tracking system for tracking efficiency calculation. In the current case all the systems are considered.

Finally, a list of so-called "seeding configurations" are specified in the block inside `BeginSeedConfigs` and `EndSeedConfigs`. Each element is a set of 3 `LayerNames` to be used as a seed.

More detail about the configuration of a track-finder algorithm can be found in sec. 4.10 and a full explaination of the formalism for the tracking efficiency calculation is is given in sec. 3.2.

Tracking resolution analysis

The data-card for this analysis is the one with the name

`VertexDetector_and_Tracker_datacard_TrackerResolution.txt`.

It is very similar to the one for geometry visualization. The main difference is the specification of the `TrkResolAnalysisParams` block and a set of different flags set to `true` at the end of the file. In this case the `NhitsMin` is set to 3 (the minimum number of hits to reconstruct a helix), and a new flag `UseLogY` is used and set to `true`. This is only to use a logarithmic scale in the vertical axis of the track performances plots that will be produced. If not specified or set to `false` a linear scale will be used instead.

Only two flags are set to `true` at the end of the file, `SavePlots` and `DoTrkResolAnalysis`. This last one is for turning-on the tracking resolution analysis. This analysis will produce plots of the resolution on the track parameters vs momentum. The tracking calculation will be performed using all the track intersections with the geometry's sensitive elements. Lets execute this analysis as follows,

```
./bin/GuariguanchiApp fullpath/VertexDetector_and_Tracker_datacard_TrackerResolution.txt.
```

A `.pdf` and `.root` files have been produced. The `.pdf` file have several pages showing the resolution of track parameters vs momentum: $\sigma(d_\rho)$ (top-left), $\sigma(\phi_0)$ (top-middle), $\sigma(d_z)$ (top-right), $\sigma(tan\lambda)$ (bottom-left) and $\sigma(p_t)/p_t$ (bottom-middle). The color code is explained

at the bottom-right legend, each one corresponding to the geometries specified. A vertical logarithmic scale is used as the `UseLogY` parameter have been set to `true` inside the `TrkResolAnalysisParams` block in the main data-card file. Each page of this file show the tracking performances vs momentum for each of the (θ, ϕ) values specified in the main data-card.

The `.root` file content is self-explaining. We strongly advice the user to plot its contents.

Telescope analysis

The data-card for this analysis is the one with the name

`VertexDetector_and_Tracker_datacard_TelescopeAnalysis.txt`.

It is very similar to the one of the Tracking resolution analysis, the only difference being the analysis flag set to `true` at the end of the data-card, `DoTelescopeAnalysis`. An analysis similar to the one of the beam-test configuration, with the only difference that in this case the tracks wont be straight but helices. Lets execute it as follows,

```
./bin/GuariguanchiApp fullpath/VertexDetector_and_Tracker_datacard_TelescopeAnalysis.txt.
```

A `.pdf` and `.root` files have been produced. A very similar set of telescope tracking performances and pointing resolution plots as in first example (c.f. sec. 2.2.1) are shown in the different pages of the output `.pdf` file. The main differences here is that the track has five parameters instead of four and there is only one DUT specified.

Efficiency analysis

The data-card for this analysis is the one with the name

`VertexDetector_and_Tracker_datacard_TrackingEfficiencyAnalysis.txt`.

It is very similar to the one from the previous analyses, the main difference being the specification of the `EfficAnalysisParams` block and a set of different flags set to `true` at the end of the file. The `EfficAnalysisParams` is very similar to the `TrkResolAnalysisParams` block. The parameters inside control the plots to be produced by the analysis. A set of similar variables as for the track parameters resolution analysis are available.

There is an additional variable called `MCSeed`. This parameter is a seed number used for the initialization of the random generator used for the MC track-seeding efficiency calculation. If not specified a default value will be used.

Only two flags are set to `true` at the end of the file, `SavePlots` and `DoPseudoEfficVsMon`. This last one is for turning-on the tracking efficiency analysis. This analysis will produce plots of the tracking efficiency and average track parameters resolution vs momentum. Lets execute this analysis as follows,

```
./bin/GuariguanchiApp fullpath/VertexDetector_and_Tracker_datacard_TrackingEfficiency.txt.
```

A `.pdf` and `.root` files have been produced. The `.pdf` file have several pages showing the tracking efficiency vs momentum for the different values of the (θ, ϕ) specified in the main data-card. Each tracking efficiency page shows 4 plots. The one at the top-left is the total tracking efficiency, including configurations with fake hits. The top-right plot shows the tracking efficiency for track without fake hits associated. The bottom-left (bottom-right) plot show the tracking effciency for tracks having one (two or more) fake hit associated.

Other pages of the `.pdf` file show the average of the track parameter resolution, where the average is weighted with each track configuration probability. The plots show a "threshold" behaviour, raising from zero above a given momentum corresponding to non-zero tracking efficiency.

At the end of the file (pages 9-28) show the tracking efficiency and average track parameters resolution as a function of θ for a fix value of the particle's momentum.

The `.root` file content is self-explaining. We strongly advice the user to plot its contents.

Material budget analysis

Finally, this analysis illustrates the kind of plots that can be produced about a geometry's material budget. The main data-card for this analysis is,

`VertexDetector_and_Tracker_datacard_MaterialBudgetAnalysis.txt`.

This data-card is pretty similar to the previous ones in this example. A new way of specifying the polar angle is illustrated in here, where it is the $\cos\theta$ and not θ which is specified. As this data-card intends a material budget analysis the `MatBudgetAnalysisParams` block needs to be specified. In there a minimum and maximum value of the momentum are specified. Finally, at the bottom the flag `DoMaterialBugdetAnalysis` is set to `true`.

Lets execute this analysis as follows,

```
./bin/GuariguanchiApp fullpath/VertexDetector_and_Tracker_datacard_MaterialBudgetAnalysis.txt.
```

A `.pdf` and `.root` files have been produced. The `.pdf` file has just a couple of pages, corresponding to the couple of geometries specified. Each page contains three plots showing the material budget vs the "polar variable" specified in the main data-card, which in present case is $\cos\theta$. The material budget a particle encounters inside a magnetic field will depend on its momentum. This is the reason of the three plots in each page of the `.pdf` file, each one corresponding momentum values: `mom_min`, $0.5\times$(`mom_min` + `mom_max`) and `mom_max`, where `mom_min` and `mom_max` are specified inside the `MatBudgetAnalysisParams` block in the main data-card file. The fill color code is explained in the bottom-right legend, where the names corresponds to the systems defined inside the systems definitions input file for the geometries.

2.2.3 Example-3: Simplified ILD tracking system

This example considers a simplified model of the ILD tracking system [1]. This model includes the vertex detector (VXD), the Silicon-inner-tracker (SIT), the Forward-Tracking-Detector (FTD) and the Time-projection-Chamber (TPC). The VXD, SIT and TPC systems are modeled as in the previous example as a set `GeoCylinder` objects. The FTD system is

modeled as a set of disks (`GeoDisk` objects). The geometry description in this example also includes a set of insensitive elements such as the beam-pipe, supports and data/power cables which are a combination of cylinders, disks and cones (`GeoCylinder`, `GeoDisk` and `GeoCone` objects, respectively), in order to have a more realistic estimation of the MS impact on tracking performances. This example will be a good opportunity to show the implementation of a gas tracking detector (TPC) in a geometry.

All the data-cards for this example can be found in the following directory,

`DataCards/Examples/Simplified_ILD_Tracking/`.

Before running any analysis lets take a look at the geometry, which is the file `ILC_TrackinSystem_Geometry.txt` inside the `GeometryCards` directory. As in the previous examples, there are a number of input files, which describe the different features of the geometry. Many features have been already described in previous examples, we will only focus on the new ones,

- **The VXD system:** this input file is made in turn of several inputs files, each one describing the different features of the VXD system, which include the sensitive layers as well as insensitive materials such as supports, Faraday cage (for electrical isolation) and power/data cables.

  The inputs files describing the insensitive materials are a good example of implementation of cylinders, cones and disk which are not sensitive.

  The input file for the VXD sensitive layers show a new feature for geometry description. Each element is a so-called `LadderCylinder`, inside the `BeginLadderCylinder` and `EndLadderCylinder` block. The `LadderCylinder` object is an imaginary cylinder which contains additional cylinders inside, which are specified inside the `BeginCylinder` and `EndCylinder` block. All the cylinders are centered at the `LadderCylinder` position. Each one has its own attributes (*e.g.* radius, length, ...), but the radii are specified with respect to the `LadderCylinder` radius. For example, A cylinder with local radius r_i, will have a global radius $R_i = r_i + R_{\text{ladder}}$, where R_{ladder} is the `LadderCylinder` radius. In the current example each `LadderCylinder` object contains two cylinders separated by 2 mm in radius, which represents the concept of VXD as a set of three so-called "double-sided" layers.

- **The SIT system:** this system is composed of two double-sided layers as in the case of the VXD detector. As such, its geometry description is the same.

- **The FTD system:** this system is made of a set of seven disks both in the forward and backward direction.

- **The TPC system:** this input file is made in turn of three input files,

  - *System walls:* this input file describes the TPC walls, which are a combination of cylinders and disks made of iron.
  - *Sensing layers:* the ILC TPC provides up to 220 space points, which in the current case are modeled as a set of 220 cylindrical sensitive layers between the inner and outer cylindrical walls. The geometry description is shown between the

BeginGasDetector and EndGasDetector block, where the minimum and maximum radii, length, and number of layers are specified. This structure will build 220 sensitive GeoCylinder objects uniformly distributed between GasDetRin and GasDetRout and with thickness of (GasDetRout - GasDetRin)/GasDetNLayers.

Each TPC sensitive layer is made of a material called TPCGAS which is similar to air and has a parameter called ResoutionModel which is set to TPC_ILD_Resol_Model. A resolution model is the modeling of the layer resolution in terms of its intrinsic resolution (ResolutionU and ResolutionV), the track position and momentum at intersection, as well as other parameters. If no resolution model is specified for a sensitive layer, the layer resolution is just the intrinsic resolution.

The TPC_ILD_Resol_Model is defined at the top of this input file, inside the BeginTPCResolutionModel and EndTPCResolutionModel block, where a set of parameters are set. This is the same model used for the ILD TPC system as described in table 1 of [7]. A full description of these parameters is given in sec. 4.

- **Systems configuration:** this feature has already been discussed in the previous example. Just notice that in this case the systems configuration is quite more complex, but the concept is the same.

- **Track-Finding lagorithms:** this feature has already been discussed in the previous example. Just notice that there are several FPCCDTrackFinderAlgo object defined in different θ region of the particle's momentum. It mainly separates the track-finder algorthms in the FTD syste, the VXD + SIT, and the region in between the last two. Also notice that the TPC system is not included in the systems list of any of the FPCCDTrackFinderAlgo defined, meaning that TPC wont be used for the tracking efficiency calculation.

Now that the geometry is understood, lets proceed to run a couple of analyses.

Geometry Visualization

The data-card for this analysis is the one with the name

`ILC_TrackingSystem_GeometryVisualization.txt`.

It is similar to the corresponding one of the previous example. Lets run it and see the output. Part of the system's response is the print-out of the geometry weight, where several systems are listed. Again, there are two outputs, one `.pdf` and one `.root` file. The contents of the files are self-evident. Lets just open the `.root` file and plot. In there there is list of canvases. By drawing the `c_geo21` canvas, it can be appreciated the complexity of the geometry by zooming-in in different regions and recognize the different geometry elements, *e.g.* sensitive elements, beam-pipe, supports, cables. The other canvases show the intersections of tracks with a variety of momenta with the geometry, where the hits produced at the TPC can be appreciated.

Tracking resolution analysis

The data-card for this analysis is the one with the name

`ILC_TrackingSystem_TrkResolutionAnalysis.txt`.

Running this analysis takes a while (∼10 min) because of the presence of the TPC. This systems adds hundreds of points (up to 220) to the track, which significantly increases the number of calculations to be performed.

Inside this analysis data-card is introduced the feature of voxeling, the data structure between the `BeginVoxeling` and `EndVoxeling` block. Inside this block at set of voxels can be defined, where a set of ranges are specified. These ranges define a subset of the geometry elements which will then be considered for track navigation, reducing the calculation time. This concept is useful when it is known in advance the region in space where the trajectories will be located. In the current example only one voxel is specified with a range of $[0, 10]$ m in the z coordinate. The specified momenta ensure that all the trajectories will be on the positive hemisphere of the z-axis, so the geometry elements located at $z < 0$ can be safely ignored when determining the tracks intersections with the geometry, which is what the specified voxels do.

When a voxel is specified a print-out message shows for each geometry the number of voxeled geometry elements with respect to the total. In the current case there are 262 voxeled elements out of 288.

The output `.pdf` files show the resolution on the track parameters for the specified (θ, ϕ) values. The flags called `PlotDOCAatHighMom` and `PlotPerformancesVsTheta` have been turned on inside the `TrkResolAnalysisParams` data block. The first one triggers the plot of the resolution on the Distance-Of-Closest-Approach (DOCA) in the transverse plane at hight momentum, *i.e.* without MS effects. The other flag triggers the plot of tracking performances as a function of θ for the different values of the specified momenta. All these plots are shown from page 7 on.

Efficiency analysis

The data-card for this analysis is the one with the name

`ILC_TrackingSystem_TrackingEfficiencyAnalysis.txt`.

A similar set of plots to the previous example is produced, *i.e.* the tracking efficiency and average track parameters resolution vs momentum and θ. Furthermore, a corresponding `.root` file is also produced.

2.2.4 Example-4: More realistic ILD tracking system

This example considers a more realistic model of the ILD tracking system [1] by introducing the concept of mosaic structures. In the previous example the tracking subsystems where modeled with simple geometrical shapes (*e.g.* cylinders and disks). In real life a layer of a silicon detector cannot be build as a perfect cylinder as the elementary building blocks are flat rectangular silicon sensors. Instead the sensors are assembled in ladders (rectangular array of several sensors) and these in turn are assembled in either spiral or alternating arrays in order to cover the full 360 degrees ϕ range in a near cylindrical shape. In the same token, disks are an assembly of so-called petals. These set of mosaic structures are going to be presented in this example, which allow to have a more realistic model of the geometry's material budget.

All the data-cards for this example can be found in the following directory,

`DataCards/Examples/Mosaic_ILD_Tracking/`.

In order to simplify things, the TPC system is excluded from the geometry. The geo-data-card for this example is the file with the name `ILC_TrackinSystem_Geometry.txt` inside the `GeometryCards` directory. The structure is very similar to the previous example, what changes is the modeling of the VXD, SIT and FTD systems. The first two are modeled as a `MosaicLadderPlane` objects in an alternating pattern. The FTD systems is modeled as a `MosaicLadderPetal`. More information about the parameters of these mosaic structures can be found in sec. 4.

Lets now run a couple of analysis.

Geometry Visualization

The data-card for this analysis is the file with name

`ILC_TrackingSystem_GeometryVisualization.txt`.

It is pretty similar to the one of the previous example. The execution time is a bit longer due to the complexity of the geometry. The mosaic structures can be better appreciated in the $X - Y$ geometry projection.

Tracking resolution analysis

The data-card for this analysis is the file with name

`ILC_TrackingSystem_TrkResolutionAnalysis.txt`.

In this data card it can be appreciated that a range on the azimuthal (ϕ) angle is specified, 30 values between range $(0, 30)$ deg. In the `TrkResolAnalysisParams` data block the flag `PlotOnlyPhiAveraged` is turned on. This triggers the production of tracking performances averaged on the specified ϕ values. This is useful in geometries like this without a full ϕ-symmetry, but with a certain periodicity in ϕ.

Furthermore, as in the previous example a voxel is specified. The main difference is that in addition to the z-range, a ϕ-range is also specified. This reduces the number of geometry elements to be considered for geometry navigation from 398 to 223, almost a factor of two.

Material budget analysis

It is also interesting to perform a material budget analysis of this complex geometry. The corresponding data-card is the file with name

`ILC_TrackingSystem_MatBudgetAnalysis.txt`.

It is very similar to the one presented in sec. 2.2.2. The main difference is that a range of ϕ values are specified. The material budget vs polar angle variable plots will be averages on the ϕ values.

The output `.pdf` file has only one page (just one geometry specified). In these plots the complexity of the geometry can be appreciated, where even the effect of cables and supports are considered.

2.2.5 Example-5: SITRINEO Setup

This example illustrates a configuration with a piece-wise definition of the magnetic field, which corresponds to a set of constant values of the B-field inside a set of non-overlapping volumes and a constant value outside all of them. In this example a magnetic field is specified inside a certain volume and zero outside. A very simple geometry is specified with 4 detection planes for tracking. All the data-cards for this example can be found at the directory,

`DataCards/Examples/SITRINEO/`.

Lets open the main data-card, `SITRINEO_datacard.txt`. The configuration is very similar to the one of the beam-test example (sec. 2.2.1). This data-card configures the geometry visualization and track resolution parameters analyses. The two specified geometries only differ on in the magnetic fields. Lets open either the `Bfield.txt` or `HighBfield.txt` files inside the `GeometryCards` directory.

In this case the B-field is specified inside the `MultipleStepsBfield` block. In there a set of volumes are indicated as well as a list of "inside B-fields". In the current case only one volume is specified (`BeginBoxVolume` block), which is a box centered at $(0, 0, 1.15)$ cm and with widths $(W_x, W_y, W_z) = (2, 2, 0.5)$ cm, and the corresponding inside B-field (`InsideBfield`) is along the x-axis with a magnitude of 1 T. The number of specified volumes and "inside B-fields" has to be the same, otherwise the program will crash with an error message. As there is not an outside B-field specified (`OutsideBfield` block), it is set to zero. For a more complete description of the configuration of this a kind of B-field see sec. 4.
Now lets run this example as follows,

```
>$ ./bin/GuariguanchiApp fullpath/SITRINEO_datacard.txt
```

The execution should take a couple of mins. Only a `.pdf` inside the `Plots/Examples/SITRINEO/` output directory. Lets open the `.pdf` file. This is a multi-page file with several plots.

The first two pages show a visualization of the 2 specified geometries. The geometries are the same, only differing in the magnetic field inside the volume represented in totted-blue in the figures. Pages 3 to 8 show the "low-field" geometry with some tracks. The pages corresponds to the different values of (θ, ϕ) specified. Pages 9 to 14 show the "high-field" geometry with some tracks. In can be appreciated the higher field.

The last 6 pages show the track parameter resolution vs momentum for the different specified values of (θ, ϕ). The set of track parameters is five in this case, but differing to the ones of the helix track: x_0, y_0, $t^0_x = p^0_x/p^0_z$, $t^0_y = p^0_y/p^0_z$ and the signed momentum p. This is the same track parameterization as used by the LHCb collaboration. As expected the performances of the two geometries is the same of the x_0, y_0, t^0_x and t^0_y parameters, mainly

determined by the first two measurement points. The main difference in performance is on p, with the high-field geometry having better performances.

2.2.6 Other examples

A list of other examples can be found inside the `DataCard/Examples/` directory. Each one tries to model the tracking system of some previous, present and future particle physics experiments. They are listed here below, with the corresponding location of all needed data-cards. The user should be able by now to understand their contents.

- **SiD tracker** [5]: all the needed data-cards for this example can be found in `DataCards/Examples/SiD_Tracker/`. **NOTE: This feature is still waiting for implementation!!!**.
- **ATLAS Itk** [8]: all the needed data-cards for this example can be found in `DataCards/Examples/ATLAS_Itk/`. **NOTE: This feature is still waiting for implementation!!!**.
- **CMS upgraded tracker** [9]: all the needed data-cards for this example can be found in `DataCards/Examples/CMS_Tracker_Upgrade/`. **NOTE: This feature is still waiting for implementation!!!**.
- **Belle-2 tracker** [10]: all the needed data-cards for this example can be found in `DataCards/Examples/Belle2_Tracker/`. **NOTE: This feature is still waiting for implementation!!!**.
- **LHCb tracker upgrade** [11]: all the needed data-cards for this example can be found in `DataCards/Examples/LHCb_Tracker_Upgrade/`. **NOTE: This feature is still waiting for implementation!!!**.
- **FOOT tracker** [12]: all the needed data-cards for this example can be found in `DataCards/Examples/FOOT_Tracking_System/`.

3 Mathematical Formalism

This section will describe the mathematical formalism for the calculation of the track parameters covariance matrix including MS and energy loss fluctuations. Furthermore, the principles for the tracking efficiency calculation will also be described.

3.1 Track parameters covariance matrix calculation

The process begins with the calculation of the track average trajectory. This corresponds to trace the particle's trajectory ignoring MS and E_{loss} fluctuations, as these effects average out to zero. The trajectory will be the solution of the equation of motion within a magnetic field,

$$\frac{d^2\vec{r}}{ds^2} = \frac{q}{p}\frac{d\vec{r}}{ds} \times \vec{B}, \tag{2}$$

where q is the particles charge, p is momentum magnitude which is constant, $\vec{B}$ the B-field and s is the path-length along the trajectory. The solution of this equation for a given set of initial conditions, $\vec{x_0}$ and $\vec{p_0}$, can be written as,

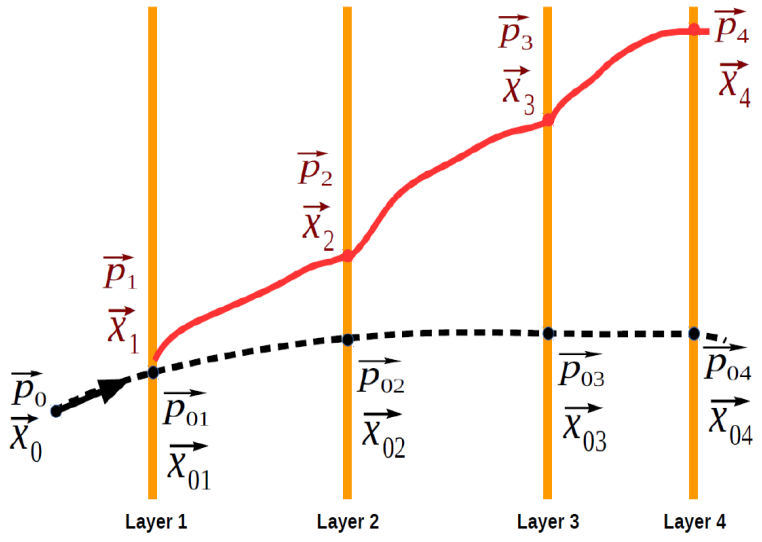

Figure 1: Schematics for calculation of space point covariance matrix due to MS.

$$\vec{r} = \vec{f}(s; \vec{x}_0, \vec{p}_0). \tag{3}$$

The corresponding momentum along the trajectory is given by,

$$\vec{p} = \vec{\kappa}(s; \vec{x}_0, \vec{p}_0) = p\frac{\partial \vec{f}(s; \vec{x}_0, \vec{p}_0)}{\partial s}. \tag{4}$$

Lets suppose that the particle starts at the initial location $\vec{x_0}$ with momentum $\vec{p_0}$, and intersects a set of materials as shown in figure 1. The black-dotted line shows the average trajectory. The red-solid line shows the actual trajectory including the MS and $E_{\rm loss}$ fluctuations at the intersection with every material. The intersection coordinates and momenta, $\vec{x}_{0,i}$ and $\vec{p}_{0,i}$, of the average trajectory are given by

$$\vec{x}_{0,i} = \vec{f}(s_i; \vec{x_0}, \vec{p_0}) = \vec{f}(s_i - s_{i-1}; \vec{x}_{0,i-1}, \vec{p}_{0,i-1}), \tag{5}$$
$$\vec{p}_{0,i} = \vec{\kappa}(s_i; \vec{x_0}, \vec{p_0}) = \vec{\kappa}(s_i - s_{i-1}; \vec{x}_{0,i-1}, \vec{p}_{0,i-1}). \tag{6}$$

We start by considering the effects of MS. Lets denote the coordinates and momenta after intersecting the material of the actual particle trajectory by $\vec{x}_i$ and $\vec{p}_i$. At the intersection with layer-1 we have,

$$x_1^k = x_{0,1}^k, \tag{7}$$
$$p_1^k = p_{0,1}^k + \delta p_1^k; \tag{8}$$

where $k = 1, 2, 3$, and δp_1^k is the effect of the multiple scattering at layer-1. This effect can be written as,

$$\delta p_1^k = |\vec{p}_{01}|(\hat{m}_{11}^k \theta_{11} + \hat{m}_{12}^k \theta_{12}), \tag{9}$$

where $\hat{m}_{11}$ and $\hat{m}_{12}$ are any two mutually orthogonal unitary vector orthogonal to the momentum just previous to the intersection with the material; and θ_{11} and θ_{12} are the multiple scattering angles along the the $\hat{m}_{11}$ and $\hat{m}_{12}$ directions.

The coordinates and momenta of the actual trajectory just after layer-2 are given by,

$$\begin{aligned} x_2^k &= f^k(s_2 - s_1; \vec{x_1}, \vec{p_1}) = f^k(s_2 - s_1; \vec{x}_{0,1}, \vec{p}_{0,1} + \delta\vec{p_1}), & (10)\\ p_2^k &= \kappa^k(s_2 - s_1; \vec{x_1}, \vec{p_1}) + \delta p_2^k = \kappa^k(s_2 - s_1; \vec{x}_{0,1}, \vec{p}_{0,1} + \delta\vec{p_1}) + \delta p_2^k; & (11)\end{aligned}$$

where

$$\delta p_2^k = |\vec{p}_{02}|(\hat{m}_{21}^k\theta_{21} + \hat{m}_{22}^k\theta_{22}). \quad (12)$$

with $\hat{m}_{2i}$ and θ_{2i} a similar meaning as in the case of the intersection with layer-1.
Equation 10 can be rewritten as,

$$\begin{aligned} x_2^k &= f^k(s_2 - s_1; \vec{x}_{0,1}, \vec{p}_{0,1}) + \frac{\partial f^k(s_2 - s_1; \vec{x}_{0,1}, \vec{p}_{0,1})}{\partial p^j}\delta p_1^j = x_{0,2}^k + \Delta x_2^k, & (13)\\ p_2^k &= \kappa^k(s_2 - s_1; \vec{x}_{0,1}, \vec{p}_{0,1}) + \frac{\partial \kappa^k(s_2 - s_1; \vec{x}_{0,1}, \vec{p}_{0,1})}{\partial p^j}\delta p_1^j + \delta p_2^k = p_{0,2}^k + \Delta p_2^k; & (14)\end{aligned}$$

with

$$\begin{aligned} \Delta x_2^k &= \mathcal{F}_{p21}^{kj}\delta p_1^j, & (15)\\ \Delta p_2^k &= \mathcal{K}_{p21}^{kj}\delta p_1^j + \delta p_2^k; & (16)\end{aligned}$$

where the matrices $\mathcal{F}_{p21}$ and $\mathcal{K}_{p21}$ are given by

$$\begin{aligned} \mathcal{F}_{p_{21}} &= \frac{\partial \vec{f}(s_2 - s_1; \vec{x}_{0,1}, \vec{p}_{0,1})}{\partial \vec{p}}, & (17)\\ \mathcal{K}_{p_{21}} &= \frac{\partial \vec{\kappa}(s_2 - s_1; \vec{x}_{0,1}, \vec{p}_{0,1})}{\partial \vec{p}}. & (18)\end{aligned}$$

In a similar fashion, the coordinates and momenta of the actual trajectory just after layer-3 are given by,

$$\begin{aligned} x_3^k &= f^k(s_3 - s_2; \vec{x_2}, \vec{p_2}) = f^k(s_3 - s_2; \vec{x}_{0,2} + \Delta\vec{x}_2, \vec{p}_{0,2} + \Delta\vec{p}_2), & (19)\\ p_3^k &= \kappa^k(s_3 - s_2; \vec{x_2}, \vec{p_2}) + \delta p_3^k = \kappa^k(s_3 - s_2; \vec{x}_{0,1} + \Delta\vec{x}_2, \vec{p}_{0,1} + \Delta\vec{p}_2) + \delta p_3^k; & (20)\end{aligned}$$

with $\Delta\vec{x}_2$ and $\Delta\vec{p}_2$ given by Equation 15, and δp_3^k is the effect of MS after traversing the layer-3, with similar expression as in the previous cases. Again, Equation 19 can be rewritten as

$$\begin{aligned} x_3^k &= x_{0,3}^k + \frac{\partial f^k(s_3 - s_2; \vec{x}_{0,1}, \vec{p}_{0,1})}{\partial x^j}\Delta x_2^j + \frac{\partial f^k(s_3 - s_2; \vec{x}_{0,1}, \vec{p}_{0,1})}{\partial p^j}\Delta p_2^j, & (21)\\ p_3^k &= p_{0,2}^k + \frac{\partial \kappa^k(s_3 - s_2; \vec{x}_{0,1}, \vec{p}_{0,1})}{\partial x^j}\Delta x_2^j + \frac{\partial \kappa^k(s_3 - s_2; \vec{x}_{0,1}, \vec{p}_{0,1})}{\partial p^j}\Delta p_2^j + \delta p_2^k; & (22)\end{aligned}$$

which can be rewritten as

$$\begin{aligned} x_3^k &= x_{0,3}^k + \Delta x_2^k, & (23)\\ p_3^k &= p_{0,3}^k + \Delta p_2^k; & (24)\end{aligned}$$

with

$$\Delta x_3^k = \mathcal{F}_{r32}^{kj}\Delta x_2^j + \mathcal{F}_{p32}^{kj}\Delta p_2^j, \tag{25}$$

$$\Delta p_3^k = \mathcal{K}_{r32}^{kj}\Delta x_2^j + \mathcal{K}_{p32}^{kj}\Delta p_2^j + \delta p_3^k; \tag{26}$$

where the matrices $\mathcal{F}_{r32}$, $\mathcal{F}_{p32}$, $\mathcal{K}_{r32}$ and $\mathcal{K}_{p32}$ are given by

$$\mathcal{F}_{r32} = \frac{\partial \vec{f}(s_3 - s_2; \vec{x}_{0,2}, \vec{p}_{0,2})}{\partial \vec{x}}, \tag{27}$$

$$\mathcal{F}_{p32} = \frac{\partial \vec{f}(s_3 - s_2; \vec{x}_{0,2}, \vec{p}_{0,2})}{\partial \vec{p}}, \tag{28}$$

$$\mathcal{K}_{r32} = \frac{\partial \vec{\kappa}(s_3 - s_2; \vec{x}_{0,2}, \vec{p}_{0,2})}{\partial \vec{x}}, \tag{29}$$

$$\mathcal{K}_{p32} = \frac{\partial \vec{\kappa}(s_3 - s_2; \vec{x}_{0,2}, \vec{p}_{0,2})}{\partial \vec{p}}. \tag{30}$$

In general the Δx_n^k and Δp_n^k can be written as

$$\Delta x_n^k = \sum_{l=1}^{n-1} F_{n,l}^{kj}\delta p_l^j, \tag{31}$$

$$\Delta p_n^k = \sum_{l=1}^{n-1} K_{n,l}^{kj}\delta p_l^j + \delta p_n^j. \tag{32}$$

Where δp_l^j is the MS effect just after layer-l and given by,

$$\delta p_l^j = |\vec{p}_{0l}|(\hat{m}_{1l}^j\theta_{1l} + \hat{m}_{1l}^j\theta_{1l}), \tag{33}$$

and where the $F_{n,l}^{kj}$ and $K_{n,l}^{kj}$ matrices are given by the recurrent formulas

$$F_{n,l}^{kj} = \begin{cases} \mathcal{F}_{r\,n,n-1}^{km}F_{n-1,l}^{mj} + \mathcal{F}_{p\,n,n-1}^{km}K_{n-1,l}^{mj} & \text{if } l < n-1 \\ \mathcal{F}_{p\,n,n-1}^{km} & \text{if } l = n-1 \end{cases} \tag{34}$$

and

$$K_{n,l}^{kj} = \begin{cases} \mathcal{K}_{r\,n,n-1}^{km}F_{n-1,l}^{mj} + \mathcal{K}_{p\,n,n-1}^{km}K_{n-1,l}^{mj} & \text{if } l < n-1 \\ \mathcal{K}_{p\,n,n-1}^{km} & \text{if } l = n-1 \end{cases} \tag{35}$$

This allows to calculate the covariance matrix due to MS among the Δx_n^k as follows

$$\left\langle \Delta x_n^i x_m^j \right\rangle_{\text{MS}} = \sum_{l=1}^{n-1}\sum_{a=1}^{m-1} F_{n,l}^{ik}F_{m,a}^{jb}\left\langle \delta p_l^k \delta p_a^b \right\rangle. \tag{36}$$

The covariance matrix among the δp_l^k is given by

$$\left\langle \delta p_l^k \delta p_a^b \right\rangle = |\vec{p}_{0l}||\vec{p}_{0a}|(\hat{m}_{1l}^k\hat{m}_{1a}^b\left\langle\theta_{1l}\theta_{1a}\right\rangle + \hat{m}_{1l}^k\hat{m}_{2a}^b\left\langle\theta_{1l}\theta_{2a}\right\rangle + \hat{m}_{2l}^k\hat{m}_{1a}^b\left\langle\theta_{2l}\theta_{1a}\right\rangle + \hat{m}_{2l}^k\hat{m}_{2a}^b\left\langle\theta_{2l}\theta_{2a}\right\rangle) \tag{37}$$

The MS angles θ_{ik} are totally uncorrelated, *i.e.* $\langle\theta_{kl}\theta_{ma}\rangle = \delta_{km}\delta_{la}(\theta_l^0)^2$, with θ_l^0 the mean-root-square of the MS angle at layer-l, which is given by the usual formula [13],

$$\theta_l^0 = \frac{13.6 \text{ MeV/c}}{c\beta_{0,l}p_{0,l}} q\sqrt{x/X_0}\left[1 + 0.038ln(x/X_0)\right], \tag{38}$$

where q, $p_{0,l}$, $c\beta_{0,l}$, are the charge number, momentum and velocity of the incident particle, and x/X_0 is the thickness of the scattering medium in radiation lengths.

Finally, $\left\langle \Delta x_n^i x_m^j \right\rangle_{\text{MS}}$ can be written as follows,

$$\left\langle \Delta x_n^i x_m^j \right\rangle_{\text{MS}} = p^2 \sum_{l=1}^{n-1}\sum_{a=1}^{m-1} F_{n,l}^{ik} F_{m,a}^{jb} \left(\hat{m}_{1l}^k \hat{m}_{1a}^b + \hat{m}_{2l}^k \hat{m}_{2a}^b\right) \delta_{la}(\theta_l^0)^2, \tag{39}$$

where p is the particle's momentum.

A similar calculation can be used for the energy loss fluctuations. It is only needed to replace in eq. 36 the corresponding expression for the $\langle \delta p_l^k \delta p_a^b \rangle$. The δp_l^k due to energy loss fluctuations can be written as,

$$\delta p_l^k = \left(\frac{E}{p}\right) \hat{p}_l^k \sigma^l(E_{\text{loss}}), \tag{40}$$

where E/p is the particle's energy-momentum ratio, $\hat{p}_l^k$ is the k-component of the particle's momentum direction at intersection l, and $\sigma^l(E_{\text{loss}})$ is the E_{loss} RMS at intersection l. For this last one we use a simple model proposed by the ALICE collaboration,

$$\sigma^l(E_{\text{loss}}) = \kappa \times \sqrt{E_{\text{loss}}^l}, \tag{41}$$

where E_{loss}^l is calculated from the famous dE/dx Bethe-Bloch formula [13], and κ is a parameter which has been tuned to the value 0.015 when the both $\sigma^l(E_{\text{loss}})$ and E_{loss}^l are expressed in GeV. The κ-tunning has being made by comparing the GUARIGUANCHI output with the ILD full-simulation tracking performances. We can now calculate $\langle \delta p_l^k \delta p_a^b \rangle$ due to the E_{loss} fluctuation,

$$\left\langle \delta p_l^k \delta p_a^b \right\rangle = \left(\frac{E}{p}\right)^2 \left(\hat{p}_l^k \hat{p}_a^b\right) \delta_{la} \sigma^l(E_{\text{loss}}), \tag{42}$$

where the δ_{la} is due to the independence of the E_{loss} fluctuations at the different intersections.

Finally, $\left\langle \Delta x_n^i x_m^j \right\rangle_{E_{\text{loss}}}$ can be written as follows,

$$\left\langle \Delta x_n^i x_m^j \right\rangle_{E_{\text{loss}}} = \left(\frac{E}{p}\right)^2 \sum_{l=1}^{n-1}\sum_{a=1}^{m-1} F_{n,l}^{ik} F_{m,a}^{jb} \left(\hat{p}_l^k \hat{p}_a^b\right) \delta_{la} \sigma^l(E_{\text{loss}}). \tag{43}$$

In GUARIGUANCHI package the measurement coordinates are expressed in terms of the sensor's local reference frame $\vec{x'}$, where the third coordinate is always perpendicular to the sensitive surface. The latest are functions of the global coordinates $\vec{x}$. The covariance matrix among the measurement coordinates in the sensor's local frame can be written as,

$$\left\langle \Delta x_n'^i x_m'^j \right\rangle = \delta_{ij}\delta_{nm}\sigma_n^i\sigma_m^j + \frac{\partial x_n'^i}{\partial x_n^k}\frac{\partial x_m'^j}{\partial x_m^r}\left(\left\langle \Delta x_n^k x_m^r \right\rangle_{\rm MS} + \left\langle \Delta x_n^k x_m^r \right\rangle_{E_{\rm loss}}\right) \quad (44)$$

$$= V_{\rm sp} + V_{MS} + V_{E_{\rm loss}}, \quad (45)$$

where the first term describes the sensor's intrinsic resolution, the second the MS and the third the $E_{\rm loss}$ fluctuations.

Finally, the expression for the track parameters covariance matrix can be calculated as follows. Lets arrange the set of measurement coordinates in the array $\Gamma_i^{\rm meas}$, and lets $\Gamma_i(\vec{\alpha})$ be the track intersection coordinates with the different measurement layers as a function of the track parameters $\vec{\alpha}$. As an example, in the case of a straight line (helix) $\vec{\alpha}$ is a vector of size 4 (5). The track fitting consist at minimizing the χ^2 given by

$$\chi^2 = \sum_{i,j}(\Gamma_i(\vec{\alpha}) - \Gamma_i^{\rm meas})\,(V)_{ij}^{-1}\,(\Gamma_j(\vec{\alpha}) - \Gamma_j^{\rm meas}), \quad (46)$$

where V is the total covariance matrix $V = V_{\rm sp} + V_{MS} + V_{E_{\rm loss}}$. The track parameters covariance matrix is given by

$$(V_{\vec{\alpha}})_{km}^{-1} = \frac{1}{2}\frac{\partial^2\chi^2}{\partial\alpha_k\partial\alpha_m} = \sum_{i,j}\frac{\partial\Gamma_i(\vec{\alpha})}{\partial\alpha_k}\,(V)_{ij}^{-1}\,\frac{\partial\Gamma_j(\vec{\alpha})}{\partial\alpha_m}. \quad (47)$$

where the derivatives are evaluated at the actual known values of the track parameters, which can be easily calculated from the initial conditions $\vec{x}_0$ and $\vec{p}_0$ at the particle's origin.

3.2 Tracking efficiency calculation

The tracking efficiency calculation depends on the track-finding algorithm used. For the time being only one such algorithm has been implemented and described in the next section. Later in this document (c.f. sec. 6.9) some guide lines to implement a new pattern recognition algorithms will be given.

3.2.1 FPCCD track finder

This track finder is inspired in the FPCCD track-finder, one of the pattern recognition algorithms used by the ILD collaboration [4], which was developed to take advantage of fine pixel CCD silicon sensors. The tracking efficiency calculation that will be detailed here below uses the same features as the ALICE ITS Upgrade fast-simulation [14].

The procedure starts by calculating the efficiency ($P_{\rm seed}$) of so-called seeding configurations, which consist of sets of three layers for the reconstruction of a seed track. The set of seed configurations should be adapted to the application and has to be defined by the user. The seed track parameters covariance matrix is then calculated and used to obtain the track pointing resolution at the next sensitive layer. If previous to that an insensitive layer is intersected, the track parameters covariance matrix is updated including the MS and $E_{\rm loss}$ fluctuations of the insensitive layer. This track pointing resolution is then used to calculate the probability of good, fake or null track-hit association (P_l). If a hit is associated (either good or fake) the track parameters covariance matrix is updated adding this new

measurement point. The extrapolation process is continued until no more sensitive layers are found.

The tracking efficiency of a given seed configuration with the corresponding configuration of associated good, fake and null hits is given by,

$$\epsilon^k = P^k_{\text{seed}} \prod_l P_l, \tag{48}$$

where P^k_{seed} is the efficiency for the seeding configuration k and P_l is the probability of either good, fake or null track-hit association at layer l. This equation ignores possible correlations among the seeding efficiency and the track-hit associations of the subsequent layers.

The total tracking efficiency is given by

$$\epsilon = \sum_k \epsilon^k, \tag{49}$$

where the sum runs over all the seeding configurations plus associated hits respecting some constrains such as: minimum number of hits in the track, minimum purity (# good hit association)/(# total hits), ...

Each configuration k will have as well its own track parameters covariance matrix $V^k_{\vec{\alpha}}$. The average covariance matrix will be give by,

$$\langle V_{\vec{\alpha}} \rangle = \frac{\sum_k \epsilon^k V^k_{\vec{\alpha}}}{\sum_k \epsilon^k}. \tag{50}$$

The output of the whole calculation will then be the tracking efficiency as well as the average track parameters covariance matrix.

In the following sub-sections are detailed the calculations for the seeding configuration efficiency and the track-hit association probabilities.

Seeding probability calculation

As mentioned above, the seeding configurations are a list of combinations of three layers used to reconstruct a seed track. The actual combinations are application dependent and should then be specified by the user. In the process of pattern recognition in a real experiment the list of seed configurations will be hierarchized, starting by the most likely one. In a collider experiment with a tracker with cylindrical symmetry with respect to the beam lines the seeding should start with combinations of the outermost layers, the ones with the lower hit rates density (hit per unit of time per unit of surface), as this reduces the amount of combinatorics. If some of the track hits on the outermost layers go undetected the corresponding seeding configuration will not be found. In order to increase tracking efficiency some other combinations with inner layers could also be tested but at the expense of increased combinatorics.

The seeding efficiency is modeled as the product of three components,

$$P_{\text{seed}} = P(\text{intrinsic}) P(p_t^{\min}) P(\chi^2/ndf \ cut), \tag{51}$$

where $P(\text{intrinsic})$ is the intrinsic probability of the seeding configuration, $P(p_t^{\min})$ the probability for the seed configuration to pass a minimum $p_t^{\min}$ cut, and $P(\chi^2/ndf \ cut)$ the probability of the seed track to pass a maximum χ^2/ndf cut.

$P(\chi^2/ndf\ cut)$ is easily calculated as the probability for a χ^2 distribution with ndf degrees of freedom to be smaller than a certain maximum $(\chi^2/ndf)_{\rm max}$ specified by the user. The ndf is calculated as the difference $ndf = 2n_{\rm hits} - n_{\rm trk-params}$, where $n_{\rm hits}$ is the number of hits to reconstruct the seed track (in this case three) and $n_{\rm trk-params}$ is the number of track parameters (in this case five).

P(intrinsic) depends on the number of good, fake and null hits related with the seed configuration. This will be easier to understand by considering an example. Lets consider a track that intersects the four outermost layers of a tracker, denoted by l_1, l_2, l_3, l_4, respectively. The following set of seed configurations could be specified,

- (l_2, l_3, l_4),
- (l_1, l_3, l_4),
- (l_1, l_2, l_4),
- (l_1, l_2, l_3).

The first configuration (l_2, l_3, l_4) corresponds to a seed track with hits on the 3 outermost layers. The intrinsic probability for this configuration is given by $P^2_{\rm hit}P^3_{\rm hit}P^4_{\rm hit}$, where $P^k_{\rm hit}$ is the probability of having either a good or fake hit at layer k. The second configuration (l_1, l_3, l_4) means that no hit was found at layer 2, so the intrinsic probability is given by $P^1_{\rm hit}P^2_{\rm null}P^3_{\rm hit}P^4_{\rm hit}$, where $P^2_{\rm null}$ is the probability of not finding a hit on layer 2. The other seed configurations will have similar expressions.

The probability for finding a good hit in the seed layer k ($P^k_{\rm good}$) is simply the layer intrinsic detection efficiency $P^k_{\rm good} = \epsilon^k_{\rm det}$. The probability for no hit in the seed layer k ($P^k_{\rm null}$) is given by the product of not detecting the real track hit $(1 - \epsilon^k_{\rm det})$ and not finding a fake hit ($P^k_{\rm no-fake}$),

$$P^k_{\rm null} = (1 - \epsilon^k_{\rm det})P^k_{\rm no-fake}. \tag{52}$$

$P^k_{\rm no-fake}$ is calculated by using the layer hit rate $R^k_{\rm bkg}$, read-out time $t^k_{\rm r.o.}$ and a surface S^k. The average number of fake hit will be given by $N^k_{\rm bkg} = R^k_{\rm bkg}t^k_{\rm r.o.}S^k$. Assuming that the number of fake hits are Poisson distributed, $P^k_{\rm no-fake}$ is given by,

$$P^k_{\rm no-fake} = Poisson(0, N^k_{\rm bkg}) = e^{-N^k_{\rm bkg}}. \tag{53}$$

The last element to define is the surface S^k. This is given by the size of the region where the real track hits have a sizable probability. This is given by the surface of the $1 - \sigma$ ellipse ($S^k_{1\sigma}$), which includes the sensor intrinsic resolution, MS and $E_{\rm loss}$ fluctuations, multiplied the a factor related to the $(\chi^2/ndf)_{\rm max}$ on the seed track,

$$S^k = \left[(\chi^2/ndf)_{\rm max}\right]^{1/2} S^k_{1\sigma}. \tag{54}$$

The probability of finding a fake hit at the seed layer k ($P^k_{\rm fake}$) is given by,

$$P^k_{\rm fake} = 1 - P^k_{\rm good} - P^k_{\rm null} = (1 - \epsilon^k_{\rm det})(1 - P^k_{\rm no-fake}). \tag{55}$$

The Last component of the seeding efficiency to discuss is the term related to the $p_t^{\rm min}$ cut,

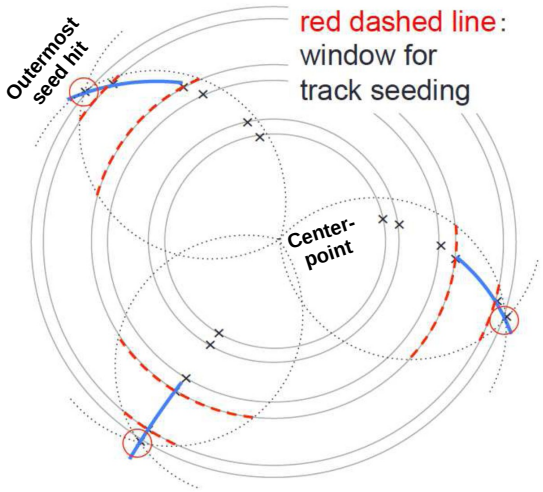

Figure 2: Graphical representation of track seeding process for the FPCCD track-finder algorithm. Crosses denote hits on the tracker. Crosses surrounded by red circles in the outer layer are used to determine a wide enough search region to catch track seeds with $p_t > p_t^{\min}$. Dotted curved lines denote tracks passing through the outermost seed hit, through a center-point and with $p_t = p_t^{\min}$. Red dashed lines denote search windows for generating track seeds, and their length is determined by intersections between layers and the dotted curved lines. Blue curved lines denote the track seeds generated by this process.

$P(p_t^{\min})$. This term is characteristic of the FPCCD pattern recognition algorithm, which track seeding process is illustrated in figure 2. This algorithm looks for a hit in the outermost seed layer of a given seeding configuration. If a hit is found then it proceeds to draw on the plane perpendicular to the uniform magnetic field two hypothetical tracks generated at the center-point, going through the hit on the outer layer, having $p_t = p_t^{\min}$ and having a ± 1 electric charge. This defines a search window for looking for hits in the other inner seed layers. If hits are found within this search window, then the track is fit and a $(\chi^2/ndf)_{\max}$ cut is applied. This is how a $p_t^{\min}$ cut is applied on the seed tracks. The efficiency due to this $p_t^{\min}$ cut should be 0 (1) for tracks with p_t much below (above) $p_t^{\min}$, and should show a quick $0 \to 1$ transition for p_t around $p_t^{\min}$.

In this package this $p_t^{\min}$ cut seed efficiency is calculated with a MC method. For this the three seed hits are sampled N_{sampling} times with a multi-variate Gaussian distribution with the their actual covariance matrix. For each sampling, the outermost seed hit is used to define the search window as described above. It is then count the amount of samplings (N_{passed}) in which the two innermost seed hits were found inside such search window. The efficiency is then simply defined as,

$$P(p_t^{\min}) = \frac{N_{\text{passed}}}{N_{\text{sampling}}}. \tag{56}$$

For this calculation N_{sampling} should be defined by the user by finding a trade-off between precision and calculation time.

Track-hit association probabilities

The track-hit association probability at a layer l depends on the track pointing resolution onto the layer, the layer intrinsic resolution and the hit rate at this layer. The hit rate depends on many factors and tries to include effects such as sensor electric noise, event multiplicity, background and possible pile-up.

One can show that the probability that the correct cluster will be associated to extrapolated seed with smaller χ^2 than any random hits can be expressed as (assuming infinite size search-window),

$$P^l_{\rm corr} = \frac{\epsilon^l_{\rm det}}{1 + N^l_{\rm bkg}}, \tag{57}$$

where $\epsilon^l_{\rm det}$ is the layer intrinsic detection efficiency and $N^l_{\rm bkg}$ is the expected number of fake hits within the region S^l defined by the convolution of the track pointing resolution and the layer intrinsic resolution, *i.e.* $N^l_{\rm bkg} = R^l_{\rm bkg} t^l_{\rm r.o.} S^l$. The above can be extended to use search windows of finite size. The correct track-hit association probability is then transformed to,

$$P^l_{\rm corr} = \frac{\epsilon^l_{\rm det}\left(1 - \gamma^{1+N^l_{\rm bkg}}\right)}{1 + N^l_{\rm bkg}}, \tag{58}$$

where γ is the fraction of correct hits lost to the χ^2 cut in the track-hit association process (*e.g.* $\gamma \simeq 5\%$ with $\chi^2/ndf < 3$ cut). The probability of no track-hit association is then,

$$P^l_{\rm null} = \left(1 - \epsilon^l_{\rm det} + \epsilon^l_{\rm det}\gamma\right)\gamma^{N^l_{\rm bkg}}, \tag{59}$$

and finally, the probability to have a fake associated to the track is,

$$P^l_{\rm fake} = 1 - P^l_{\rm corr} - P^l_{\rm null}. \tag{60}$$

Therefore, three states (correct, fake and no association) are possible per layer. The number of possible outcomes also depends on the number of layers and is therefore $K_p = 3^N$. Only one outcome corresponds to the simplified picture of correct cluster association on each single layer (N correct track-hit associations). By adding the probabilities of *e.g.* having $N-1$ correct associations and no association at the remaining layer would correspond to "correct tracks with at least $N-1$ clusters" and so on.

4 Geometry configuration

In this section are described the different elements for a geometry configuration as well as the syntax for the specification of the world volume, magnetic field, different geometry elements, resolution model, efficiency model, and systems and Telescope-DUT configurations.

In GUARIGUANCHI there is an internal unit system. All physical quantities have to be specified with the corresponding units, otherwise the program will crash with an error message.

The volumes in space are placed by specifying a position and a set of three rotation angles $(\alpha_x, \alpha_y, \alpha_z)$ to determine its orientation. The rotation convention is

$$R = R_z(\alpha_x)R_z(\alpha_y)R_z(\alpha_z) \tag{61}$$

where R is the full rotation matrix, and $R_i(\alpha_i)$ are rotation matrices along the axis-i by an angle α_i. The rotation order if along the z-axis first, then along the y-axis and finally along the x-axis. The specification of the rotation angles is not mandatory, if not specified they will all be set to its default value $(0, 0, 0)$.

4.1 World volume

The world volume should contain all the geometry elements, and only inside it the track navigation will be performed. Currently, only two kinds of world volumes are possible, either a box or a cylinder. The specification is as follows,

4.1.1 Box world volume

The box world volume is specified as follows,

```
BeginBoxWorldVolume
   Position x y z units // (Mandatory)
   widthX value units   // (Mandatory, value > 0)
   widthY value units   // (Mandatory, value > 0)
   widthZ value units   // (Mandatory, value > 0)
EndBoxWorldVolume
```

where `Position` specifies the center of the box, and `widthX,Y,Z` the widths along the global frame axes.

4.1.2 Cylinder world volume

The cylinder world volume is specified as follows,

```
BeginCylinderWorldVolume
   Position x y z units // (Mandatory)
   Radius value units   // (Mandatory, value > 0)
   Length value units   // (Mandatory, value > 0)
EndCylinderWorldVolume
```

where `Position` specifies the center of the cylinder. The other parameters are evident. The cylinder is always oriented along the global z-axis.

4.2 Magnetic field

Each geometry has its own magnetic field. If not specified it is assumed to be zero. Two types of B-field are currently implemented.

4.2.1 Constant Magnetic field

A constant B-field is build by specifying its magnitude and direction as shown below.

```
BeginConstantBfield
   Magnitude value units // (Mandatory, value > 0)
```

```
  Direction x y z       // (Mandatory, vector will be normalized to unity)
EndConstantBfield
```

4.2.2 Piece-wise Magnetic field

This magnetic field consist of a set of non-overlapping volumes with a constant B-field inside each of them. If any pair of the specified volumes overlap, the program will crash with an error message. Outside all the volumes the B-field is zero by default, but it can be set to a non-zero values if specified. A set of volumes of different types can be specified. A set of so-called "inside-fields" have to be specified as well. The number of volumes have to match the number of inside-fields, otherwise the program will crash with an error message. The syntax for specifying this kind of B-field is as follows.

```
BeginMultipleStepsBfield
  BeginSomeTypeVolume
   ...
  EndSomeTypeVolume
  BeginInsideBfield
   Magnitude value units // (Mandatory, value > 0)
   Direction x y z       // (Mandatory, vector will be normalized to unity)
  EndInsideBfield
  ...
  BeginOutsideBfield
   Magnitude value units // (Mandatory, value > 0)
   Direction x y z       // (Mandatory, vector will be normalized to unity)
  EndOutsideBfield
EndMultipleStepsBfield
```

The data structure inside `BeginOutsideBfield` and `EndOutsideBfield` is optional. If not specified then the "outside-field" will be set to zero.

The `SomeType` on `BeginSomeTypeVolume` and `EndSomeTypeVolume` represents the different types of volumes which can be specified, each of them with their own syntax. There are several volume types which are mentioned here below.

- **Box volume**: specified as follows,

  ```
  BeginBoxVolume
    Position x y z units     // (Mandatory)
    RotAngles αx αx αx units // (Optional)
    widthX value units       // (Mandatory, value > 0)
    widthY value units       // (Mandatory, value > 0)
    widthZ value units       // (Mandatory, value > 0)
  EndBoxVolume
  ```

- **Cylinder volume**: specified as follows,

  ```
  BeginCylinderVolume
    Position x y z units     // (Mandatory)
  ```

```
  RotAngles αx αx αx units // (Optional)
  Radius value units       // (Mandatory, value > 0)
  Length value units       // (Mandatory, value > 0)
EndCylinderVolume
```

- **Cone volume**: specified as follows,

```
BeginConeVolume
  Position x y z units     // (Mandatory)
  RotAngles αx αx αx units // (Optional)
  Radius1 value units      // (Mandatory, value > 0)
  Radius2 value units      // (Mandatory, value > 0)
  Length value units       // (Mandatory, value > 0)
EndConeVolume
```

- **Disk section volume**: specified as follows,

```
BeginDiskSectionVolume
  Position x y z units     // (Mandatory)
  RotAngles αx αx αx units // (Optional)
  Rin value units          // (Mandatory, value > 0)
  Rout value units         // (Mandatory, value > 0)
  Length value units       // (Mandatory, value > 0)
  DeltaPhi value units     // (Mandatory, value > 0)
EndDiskSectionVolume
```

  The disk section will expand an angle `DeltaPhi` around the local x-axis.

- **Cone section volume**: specified as follows,

```
BeginConeSectionVolume
  Position x y z units     // (Mandatory)
  RotAngles αx αx αx units // (Optional)
  Radius1 value units      // (Mandatory, value > 0)
  Radius2 value units      // (Mandatory, value > 0)
  Length value units       // (Mandatory, value > 0)
  DeltaPhi value units     // (Mandatory, value > 0)
EndConeSectionVolume
```

  The cone section will expand an angle `DeltaPhi` around the local x-axis.

As an example, here below there is an example of a piece-wise B-field with two cylindric volumes along the z-axis with fields pointing in opposite directions, both along the x-axis.

```
BeginMultipleStepsBfield
  BeginCylinderVolume
   Position 0.0 0.0 0.0 cm
   RotAngles 0.0 0.0 0.0 deg
   Radius 5.0 cm
   Length 2.0 cm
  EndCylinderVolume
  BeginInsideBfield
```

```
    Magnitude 1.0 T
    Direction 1 0 0
  EndInsideBfield
  BeginCylinderVolume
    Position 0.0 0.0 10.0 cm
    RotAngles 0.0 0.0 0.0 deg
    Radius 5.0 cm
    Length 2.0 cm
  EndCylinderVolume
  BeginInsideBfield
    Magnitude 1.0 T
    Direction -1 0 0
  EndInsideBfield
  BeginOutsideBfield
    Magnitude 0.5 T
    Direction 0 1 0
  EndOutsideBfield
EndMultipleStepsBfield
```

4.3 Simple geometries

In this section will be described the syntax for the geometrical volumes up to now implemented in GUARIGUANCHI. A complex geometry will be build using the simple geometrical volumes described in the sections below. In the specification of a geometry element a set of parameters are always mandatory, those which allow to place the volume on space and to assign its material properties. These volumes could be sensitive, *i.e.* detector, depending on the value of the bool variable `IsSensitive`. In case it is set to `true`, then the specification of additional parameters becomes mandatory. Those parameters refers to the detection performances of the sensitive element, such as intrinsic resolution (`ResolutionU/V`), detection efficiency (`Efficiency`), readout time (`ROtime`) and insensitive borders (`InsensFracU/Vneg` and `InsensFracU/Vpos`). The amount of hit rate (`BkgRate`, hits per unit of time and unit of surface) can also be specified. It is mainly used in the calculation of the tracking efficiency.

Lets now describe the syntax of the different geometrical volumes. The meaning of some parameters common to all volumes will be fully described in the section 4.3.1.

4.3.1 Plane

A so-called `GeoPlane`, is a box in which one of its dimensions is much smaller than the other two. Its syntax is as follows,

```
BeginGeoPlane
  Name string              // (Mandatory)
  Position x y z units     // (Mandatory)
  RotAngles α_x α_y α_z units // (Optional)
  Thickness value units    // (Mandatory, value > 0)
  Material string          // (Mandatory)
  X0X0 value               // (Optional)
```

```
  widthU value units       // (Mandatory, value > 0)
  widthV value units       // (Mandatory, value > 0)
  IsSensitive bool         // (Mandatory)
  ResolutionU value units // (Optional, value > 0)
  ResolutionV value units // (Optional, value > 0)
  Resolution value units // (Optional, value > 0)
  ROtime value units      // (Optional, value > 0)
  Efficiency value        // (Optional, 0 <= value <= 1)
  InsensFracUneg value    // (Optional, default is zero)
  InsensFracUpos value    // (Optional, default is zero)
  InsensFracVneg value    // (Optional, default is zero)
  InsensFracVpos value    // (Optional, default is zero)
  BkgRate value units     // (Optional, default is zero)
  SystemName string       // (Optional)
  LayerName string        // (Optional)
  ResolutionModel string // (Optional)
  EfficiencyModel string // (Optional)
EndGeoPlane
```

The material budget in units of radiation lengths seen by a particle traversing some path x the volume is calculated using the `Material` name. GUARIGUANCHI has a table of materials with their corresponding X_0. The material budget in then simply x/X_0. The material budget of a given element at normal incidence can be set by hand with the parameter `XOX0`. If it is set, then the material budget at normal incidence, *i.e.* for a traversed path equal to the thickness, becomes `XOX0`, and the material budget of an arbitrary path x becomes $x/X_0 = (x/\text{Thickness})\times$`XOX0`. This feature is useful when to model the material budget of a complicated composite of materials like a ladder with sensors, where several material are used to build the object: sensors, cables, supports, ... With this feature a single value of `XOX0` has to be specified. This same feature is shared by all the other geometrical volumes.

The variables U and V above refer to the local coordinates on the so-called "main-surface" of the volume. It is a plane in the middle of the volume which is used as reference for the local reference frame. In the case of the `GeoPlane` object, the main surface is a plane in the middle of the thickness, located at `Position` with the `RotAngles` orientation, and with widths `widthU` and `widthV`.

If the `IsSensitive` parameter is set to `true` the intrinsic resolutions along the local U and V directions has to be specified. They can be specified separately with the `ResolutionU` and `ResolutionV` parameters, or if they are the same it is only needed to specify the `Resolution` parameter (`ResolutionU/V = Resolution`).

The `InsensFracUneg` and `InsensFracUpos` only count for the case of `IsSensitive` set to `true`. These can be used to specify an insensitive length at the borders of the `GeoPlane`. These parameters are fractions, *i.e.* the values they can have are inside $[0, 1]$, and their sum cannot exceed 1. The insensitive length at the low-edge (high-edge) along the local U coordinate is calculated as `InsensFracUneg`$\times$`widthU` (`InsensFracUpos`$\times$`widthU`). So, even if a particle traverses the volume it can do it in a non-sensitive region. In such a case this intersection contributes with no measurement but only with material budget. The parameters `InsensFraVcneg` and `InsensFracVpos` have a similar meaning.

The `SystemName` and `LayerName` parameters are optional. See the examples in section 2.2

for the usage of these parameters.

The `ResolutionModel` parameter is a string referring to the name of an `GResolutionModel` object. This object calculates the sensitive element spatial resolution of the measured point as a function of the intrinsic resolutions (`ResolutionU/V`), the intersection coordinates and momentum of the particle with the main surface, and some other parameters. Its default value is an empty string, in such case the measurement spatial resolution is equal to the intrinsic resolution. See the example in section 2.2.3 for an illustration on how it can be used.

Finally, the `EfficiencyModel` parameter is a string referring to the name of an `GEfficiencyModel` object. This object calculates the sensitive element detection efficiency of the measured point point as a function of the intrinsic detection efficiency (`Efficiency`), the intersection coordinates and momentum of the particle with the main surface, and some other parameters. Its default value is an empty string, in such case the measurement detection efficiency is equal to the intrinsic detection efficiency.

4.3.2 Cylinder

A so-called `GeoCylinder`, is a cylinder with a small radial thickness. Its syntax is as follows,

```
BeginGeoCylinder
  Name string              // (Mandatory)
  Position x y z units     // (Mandatory)
  RotAngles αx αy αz units // (Optional)
  Thickness value units    // (Mandatory, value > 0)
  Material string          // (Mandatory)
  X0X0 value               // (Optional)
  Radius value units       // (Mandatory, value > 0)
  Length value units       // (Mandatory, value > 0)
  IsSensitive bool         // (Mandatory)
  ResolutionU value units // (Optional, value > 0)
  ResolutionV value units // (Optional, value > 0)
  Resolution value units // (Optional, value > 0)
  ROtime value units      // (Optional, value > 0)
  Efficiency value        // (Optional, 0 <= value <= 1)
  InsensFracVneg value    // (Optional, default is zero)
  InsensFracVpos value    // (Optional, default is zero)
  BkgRate value units     // (Optional, default is zero)
  SystemName string       // (Optional)
  LayerName string        // (Optional)
  ResolutionModel string // (Optional)
  EfficiencyModel string // (Optional)
EndGeoCylinder
```

In the case of the `GeoCylinder` object, the main surface is a cylinder in the middle of the thickness, located at `Position` with the `RotAngles` orientation, and `Radius` and `Length`. The V coordinate goes along the cylinder axis, while the U coordinate goes around the circle for fixed V.

For a full discussion of all the other parameters see section 4.3.1.

4.3.3 Cylinder Section

A so-called `GeoCylinderSection`, is a section of a cylinder with a small radial thickness. Its syntax is as follows,

```
BeginGeoCylinderSection
   Name string              // (Mandatory)
   Position x y z units     // (Mandatory)
   RotAngles α_x α_y α_z units // (Optional)
   Thickness value units    // (Mandatory, value > 0)
   Material string          // (Mandatory)
   XOX0 value               // (Optional)
   Radius value units       // (Mandatory, value > 0)
   Length value units       // (Mandatory, value > 0)
   DeltaPhi value units       // (Mandatory, value > 0)
   IsSensitive bool         // (Mandatory)
   ResolutionU value units // (Optional, value > 0)
   ResolutionV value units // (Optional, value > 0)
   Resolution value units // (Optional, value > 0)
   ROtime value units      // (Optional, value > 0)
   Efficiency value        // (Optional, 0 <= value <= 1)
   InsensFracVneg value    // (Optional, default is zero)
   InsensFracVpos value    // (Optional, default is zero)
   BkgRate value units     // (Optional, default is zero)
   SystemName string       // (Optional)
   LayerName string        // (Optional)
   ResolutionModel string // (Optional)
   EfficiencyModel string // (Optional)
EndGeoCylinderSection
```

In the case of the `GeoCylinder` object, the main surface is a section of a cylinder in the middle of the thickness, located at `Position` with the `RotAngles` orientation, `Radius` and `Length`, and aperture `DeltaPhi`. The V coordinate goes along the cylinder axis, while the U coordinate goes around the circle section for fixed V.

For a full discussion of all the other parameters see section 4.3.1.

4.3.4 Disk

A so-called `GeoDisk`, is a disk with a small thickness. Its syntax is as follows,

```
BeginGeoDisk
   Name string              // (Mandatory)
   Position x y z units     // (Mandatory)
   RotAngles α_x α_y α_z units // (Optional)
   Thickness value units    // (Mandatory, value > 0)
   Material string          // (Mandatory)
```

```
  XOX0 value                // (Optional)
  Rin value units       // (Mandatory, value > 0)
  Rout value units       // (Mandatory, value > 0)
  IsSensitive bool          // (Mandatory)
  ResolutionU value units // (Optional, value > 0)
  ResolutionV value units // (Optional, value > 0)
  Resolution value units // (Optional, value > 0)
  ROtime value units      // (Optional, value > 0)
  Efficiency value        // (Optional, 0 <= value <= 1)
  InsensFracVneg value    // (Optional, default is zero)
  InsensFracVpos value    // (Optional, default is zero)
  BkgRate value units     // (Optional, default is zero)
  SystemName string       // (Optional)
  LayerName string        // (Optional)
  ResolutionModel string // (Optional)
  EfficiencyModel string // (Optional)
EndGeoDisk
```

In the case of the `GeoDisk` object, the main surface is a disk in the middle of the thickness, located at `Position` with the `RotAngles` orientation, and inner and outer radii `Rin` and `Rout`, respectively. The V coordinate goes radially, while the U coordinate goes around the circle for fixed V.

For a full discussion of all the other parameters see section 4.3.1.

4.3.5 Disk Section

A so-called `GeoDiskSection`, is a section of a disk with a small thickness. Its syntax is as follows,

```
BeginGeoDiskSection
  Name string              // (Mandatory)
  Position x y z units     // (Mandatory)
  RotAngles α_x α_y α_z units // (Optional)
  Thickness value units    // (Mandatory, value > 0)
  Material string          // (Mandatory)
  XOX0 value               // (Optional)
  Rin value units       // (Mandatory, value > 0)
  Rout value units       // (Mandatory, value > 0)
  DeltaPhi value units       // (Mandatory, value > 0)
  IsSensitive bool         // (Mandatory)
  ResolutionU value units // (Optional, value > 0)
  ResolutionV value units // (Optional, value > 0)
  Resolution value units // (Optional, value > 0)
  ROtime value units      // (Optional, value > 0)
  Efficiency value        // (Optional, 0 <= value <= 1)
  InsensFracVneg value    // (Optional, default is zero)
  InsensFracVpos value    // (Optional, default is zero)
```

```
  BkgRate value units     // (Optional, default is zero)
  SystemName string       // (Optional)
  LayerName string        // (Optional)
  ResolutionModel string // (Optional)
  EfficiencyModel string // (Optional)
EndGeoDiskSection
```

In the case of the `GeoDiskSection` object, the main surface is a section of a disk in the middle of the thickness, located at `Position` with the `RotAngles` orientation, inner and outer radii `Radius` and `Length`, respectively, and aperture `DeltaPhi`. The V coordinate goes radially, while the U coordinate goes around the circle section for fixed V.

For a full discussion of all the other parameters see section 4.3.1.

4.3.6 Cone

A so-called `GeoCone`, is a cone with a small radial thicknesses. Its syntax is as follows,

```
BeginGeoCone
  Name string              // (Mandatory)
  Position x y z units     // (Mandatory)
  RotAngles α_x α_y α_z units // (Optional)
  Thickness1 value units    // (Mandatory, value > 0)
  Thickness2 value units    // (Optional)
  Material string          // (Mandatory)
  XOX0 value               // (Optional)
  Radius1 value units       // (Mandatory, value > 0)
  Radius2 value units       // (Mandatory, value > 0)
  Length value units       // (Mandatory, value > 0)
  IsSensitive bool         // (Mandatory)
  ResolutionU value units // (Optional, value > 0)
  ResolutionV value units // (Optional, value > 0)
  Resolution value units // (Optional, value > 0)
  ROtime value units      // (Optional, value > 0)
  Efficiency value        // (Optional, 0 <= value <= 1)
  InsensFracVneg value    // (Optional, default is zero)
  InsensFracVpos value    // (Optional, default is zero)
  BkgRate value units     // (Optional, default is zero)
  SystemName string       // (Optional)
  LayerName string        // (Optional)
  ResolutionModel string // (Optional)
  EfficiencyModel string // (Optional)
EndGeoCone
```

In the case of the `GeoCone` object, the main surface is a cone in the middle of its thicknesses, located at `Position` with the `RotAngles` orientation, and radii at the low-edges and high-edges along its axis of `Radius1` and `Radius2`, respectively, and `Length`. The `Thickness1` (`Thickness2`) parameter refers to the thickness at the low-edge (high-edge) of the cone.

Thickness2 is optional, if not specified then the same thickness will be used for both edges. The V coordinate goes along the cone slope, while the U coordinate goes around the circle for fixed V.

For a full discussion of all the other parameters see section 4.3.1.

4.3.7 Cone Section

A so-called GeoConeSection, is a section of a cone with a small radial thicknesses. Its syntax is as follows,

```
BeginGeoConeSection
   Name string              // (Mandatory)
   Position x y z units     // (Mandatory)
   RotAngles α_x α_y α_z units // (Optional)
   Thickness1 value units    // (Mandatory, value > 0)
   Thickness2 value units    // (Optional)
   Material string          // (Mandatory)
   X0X0 value               // (Optional)
   Radius1 value units       // (Mandatory, value > 0)
   Radius2 value units       // (Mandatory, value > 0)
   Length value units       // (Mandatory, value > 0)
   DeltaPhi value units     // (Mandatory, value > 0)
   IsSensitive bool         // (Mandatory)
   ResolutionU value units // (Optional, value > 0)
   ResolutionV value units // (Optional, value > 0)
   Resolution value units // (Optional, value > 0)
   ROtime value units      // (Optional, value > 0)
   Efficiency value        // (Optional, 0 <= value <= 1)
   InsensFracVneg value    // (Optional, default is zero)
   InsensFracVpos value    // (Optional, default is zero)
   BkgRate value units     // (Optional, default is zero)
   SystemName string       // (Optional)
   LayerName string        // (Optional)
   ResolutionModel string // (Optional)
   EfficiencyModel string // (Optional)
EndGeoConeSection
```

In the case of the GeoConeSection object, the main surface is a cone section in the middle of its thicknesses, located at Position with the RotAngles orientation, and radii at the low-edges and high-edges along its axis of Radius1 and Radius2, respectively, Length, and aperture DeltaPhi. The Thickness1 (Thickness2) parameter refers to the thickness at the low-edge (high-edge) of the cone. Thickness2 is optional, if not specified then the same thickness will be used for both edges. The V coordinate goes along the cone slope, while the U coordinate goes around the circle section for fixed V.

For a full discussion of all the other parameters see section 4.3.1.

4.3.8 Petal

A so-called `GeoPetal`, is a trapezoid with a small thickness. Its syntax is as follows,

```
BeginGeoPetal
   Name string              // (Mandatory)
   Position x y z units     // (Mandatory)
   RotAngles αx αy αz units // (Optional)
   Thickness value units    // (Mandatory, value > 0)
   Material string          // (Mandatory)
   XOX0 value               // (Optional)
   bottomWidth value units // (Mandatory, value > 0)
   topWidth value units     // (Mandatory, value > 0)
   Height value units       // (Mandatory, value > 0)
   IsSensitive bool         // (Mandatory)
   ResolutionU value units // (Optional, value > 0)
   ResolutionV value units // (Optional, value > 0)
   Resolution value units // (Optional, value > 0)
   ROtime value units      // (Optional, value > 0)
   Efficiency value        // (Optional, 0 <= value <= 1)
   InsensFracVneg value    // (Optional, default is zero)
   InsensFracVpos value    // (Optional, default is zero)
   BkgRate value units     // (Optional, default is zero)
   SystemName string       // (Optional)
   LayerName string        // (Optional)
   ResolutionModel string // (Optional)
   EfficiencyModel string // (Optional)
EndGeoPetal
```

In the case of the `GeoPetal` object, the main surface is a trapezoid in the middle of the thickness, located at `Position` with the `RotAngles` orientation, bottom and top widths of `bottomWidth` and `topWidth`, respectively, and height `Height`. The V coordinate goes along the trapezoid height, while the U coordinate goes orthogonal to it for a fixed V.

For a full discussion of all the other parameters see section 4.3.1.

4.4 Ladder geometries

Ladder geometries is a way of specifying geometry elements with respect to a reference volume. Two such structures, plane and cylinder ladders, are implemented in GUARIGUANCHI and described in the following sections.

4.4.1 Plane Ladder

A plane ladder is a set of plane objects located in space with respect to a reference plane object. The syntax is as follows,

```
BeginLadderPlane
   LadderName string              // (Mandatory)
```

```
  LadderPosition x y z units     // (Mandatory)
  LadderRotAngles αx αy αz units // (Optional)
  BeginPlane
   Name string              // (Mandatory)
   Position x y z units     // (Mandatory)
   Thickness value units    // (Mandatory, value > 0)
   Material string          // (Mandatory)
   XOX0 value               // (Optional)
   widthU value units       // (Mandatory, value > 0)
   widthV value units       // (Mandatory, value > 0)
   IsSensitive bool         // (Mandatory)
   ResolutionU value units // (Optional, value > 0)
   ResolutionV value units // (Optional, value > 0)
   Resolution value units // (Optional, value > 0)
   ROtime value units      // (Optional, value > 0)
   Efficiency value        // (Optional, 0 <= value <= 1)
   InsensFracVneg value    // (Optional, default is zero)
   InsensFracVpos value    // (Optional, default is zero)
   BkgRate value units     // (Optional, default is zero)
   SystemName string       // (Optional)
   LayerName string        // (Optional)
   ResolutionModel string // (Optional)
   EfficiencyModel string // (Optional)
  EndPlane
   ...
EndLadderPlane
```

Where as many planes can be specified following the syntax inside `BeginPlane` and `EndPlane`. The `Position` parameter in the `Plane` data structure is the plane position with respect to the ladder, *i.e.* `Global-position` = `LadderPosition` + `Position`. The result of this syntax will be the construction of as many `GeoPlane` objects as specified planes, all with the same orientation given by the `LadderRotAngles`, and differing in position. The user has to make sure that the planes positioned in this way don't overlap.

4.4.2 Cylinder Ladder

A cylinder ladder is a set of cylinder objects located in space with respect to a reference cylinder object. The syntax is as follows,

```
BeginLadderCylinder
  LadderName string              // (Mandatory)
  LadderPosition x y z units     // (Mandatory)
  LadderRotAngles αx αy αz units // (Optional)
  LadderRadius value units       // (Mandatory)
  BeginCylinder
   Name string              // (Mandatory)
   Thickness value units    // (Mandatory, value > 0)
```

```
    Material string          // (Mandatory)
    XOXO value               // (Optional)
    Radius value units       // (Mandatory)
    Length value units       // (Mandatory, value > 0)
    IsSensitive bool         // (Mandatory)
    ResolutionU value units // (Optional, value > 0)
    ResolutionV value units // (Optional, value > 0)
    Resolution value units // (Optional, value > 0)
    ROtime value units      // (Optional, value > 0)
    Efficiency value        // (Optional, 0 <= value <= 1)
    InsensFracVneg value    // (Optional, default is zero)
    InsensFracVpos value    // (Optional, default is zero)
    BkgRate value units     // (Optional, default is zero)
    SystemName string       // (Optional)
    LayerName string        // (Optional)
    ResolutionModel string // (Optional)
    EfficiencyModel string // (Optional)
  EndCylinder
   ...
EndLadderCylinder
```

Where as many cylinders can be specified following the syntax inside `BeginCylinder` and `EndCylinder`. The `Radius` parameter in the `Cylinder` data structure is the cylinder radius with respect to the ladder radius, *i.e.* `Global-radius` = `LadderRadius` + `Radius`. The result of this syntax will be the construction of as many `GeoCylinder` objects as specified cylinder, all with the same orientation given by the `LadderRotAngles`, and differing in radii. The user has to make sure that the planes positioned in this way don't overlap.

4.5 Mosaic geometries

A mosaic geometry is a further steps from the ladder geometries in building complex geometries. A plane ladder can then be placed in space in a pattern to resemble a cylinder. In the same token, a set of petals can be placed in space in such a way to resemble a disk. These two kind of structures are described in the sections below.

4.5.1 Plane ladder mosaic

A plane ladder mosaic is complex structure made from plane ladders to resemble a cylinder. There are two kinds of such plane ladder mosaic than can be made, the so-called "spiral" and "alternating" arrangements, which are illustrated in Figures 3 and 4, respectively. The figures also show the parameters needed to define such structures. These is a relationship among these parameters for each configuration.

- **Spiral configuration**:

$$\frac{d}{L} = 1 - \left(\frac{1-\cos(\Delta\alpha)}{\sin(\Delta\alpha)}\right)\left(\frac{w+2R}{L}\right) \tag{62}$$

$$\frac{s}{L} = \left(1-\frac{d}{L}\right)\left(\frac{R-(R+w)\cos(\Delta\alpha)}{(1-\cos(\Delta\alpha))(w+2R)}\right) \tag{63}$$

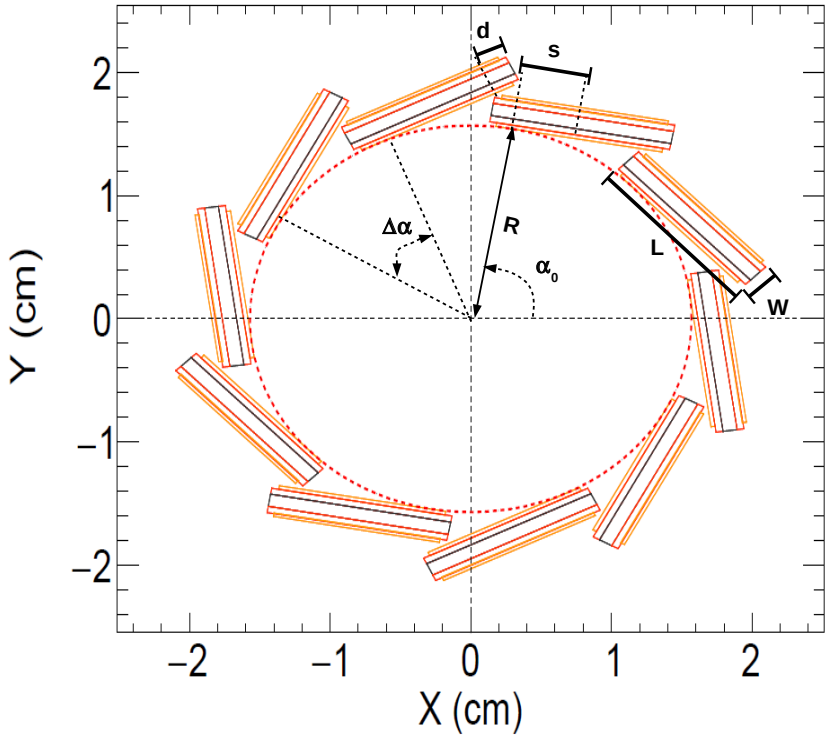

Figure 3: Spiral plane ladder mosaic configuration.

$$\Delta\alpha = \frac{2\pi}{N_{\text{ladders}}} \tag{64}$$

- **Alternating configuration**:

$$\frac{d}{L} = \frac{L(1+\cos(\Delta\alpha)) - 2(w+R)\sin(\Delta\alpha)}{2L} \tag{65}$$

$$\frac{s}{L} = \frac{(1+w/R)[1+(1-2\frac{d}{L})\cos(\Delta\alpha)]}{1-2\frac{d}{L}+\cos(\Delta\alpha)} \tag{66}$$

$$\Delta\alpha = \frac{2\pi}{N_{\text{ladders}}/2} \tag{67}$$

Where N_{ladders} is the number of ladders.

The syntax to specify such structures is as follows.

```
BeginMosaicLadderPlane
   LadderName string                 // (Mandatory)
   LadderMosaic string               // (Mandatory)
   LadderPosition x y z units        // (Mandatory)
   LadderRotAngles αx αy αz units // (Optional)
   LadderVarPar string               // (Mandatory)
   LadderRadius value units          // (Mandatory)
```

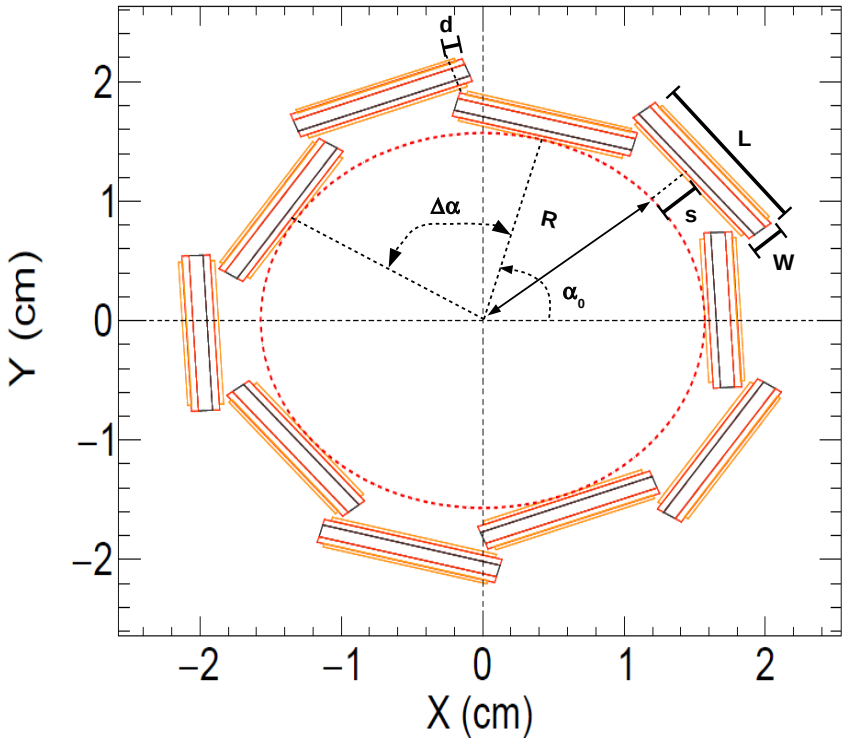

Figure 4: Alternating plane ladder mosaic configuration.

```
LadderOverlap value units       // (Mandatory)
LadderClearanceH value units    // (Optional)
LadderClearanceV value units    // (Optional)
LadderAlpha0 value units        // (Optional)
LadderN value                   // (Mandatory)
LadderShift value               // (Optional)
LadderShiftFix bool             // (Optional)
BeginPlane
 Name string               // (Mandatory)
 Position x y z units      // (Mandatory)
 Thickness value units     // (Mandatory, value > 0)
 Material string           // (Mandatory)
 X0X0 value                // (Optional)
 widthU value units        // (Mandatory, value > 0)
 widthV value units        // (Mandatory, value > 0)
 IsSensitive bool          // (Mandatory)
 ResolutionU value units // (Optional, value > 0)
 ResolutionV value units // (Optional, value > 0)
 Resolution value units // (Optional, value > 0)
 ROtime value units       // (Optional, value > 0)
 Efficiency value         // (Optional, 0 <= value <= 1)
 InsensFracVneg value     // (Optional, default is zero)
 InsensFracVpos value     // (Optional, default is zero)
```

```
    BkgRate value units     // (Optional, default is zero)
    SystemName string       // (Optional)
    LayerName string        // (Optional)
    ResolutionModel string // (Optional)
    EfficiencyModel string // (Optional)
   EndPlane
   ...
EndMosaicLadderPlane
```

As in the case of the plane ladder, an indefinite number of planes can be specified. Out of this list the system will calculate L and w (see figures 3 and 4) as the maximum width and total thickness of the ladder adding a clearance of `LadderClearanceH` and `LadderClearanceV`, respectively. The `LadderMosaic` parameter defines the type of structure to build, it can only have the values "Spiral" or "Alternating". Another important parameter is `LadderVarPar`, which is the parameter to be calculated out of the others using Eq. 62 or 65. This parameter can have either the value "Overlap" or "Radius". In the first (second) case the overlap d/L (radius R) is calculated fixing all the other parameters. The shift s/L is then calculated from Eq 63 or 66. If the boolean parameter `LadderShiftFix` is set to `true`, the shift will be fixed to the specified value `LadderShift`.

4.5.2 Petal ladder mosaic

A petal ladder mosaic is complex structure made from petal ladders to resemble a disk as shown in figure 5. The syntax to specify such structure is as follows.

```
BeginMosaicLadderPetal
   LadderName string              // (Mandatory)
   LadderPosition x y z units     // (Mandatory)
   LadderRotAngles α_x α_y α_z units // (Optional)
   LadderRadius value units       // (Mandatory)
   LadderAlpha0 value units        // (Optional)
   LadderN value                   // (Mandatory)
   LadderShift value units         // (Optional)
   BeginPetal
    Name string              // (Mandatory)
    Position x y z units     // (Mandatory)
    Thickness value units    // (Mandatory, value > 0)
    Material string          // (Mandatory)
    X0X0 value               // (Optional)
    bottomWidth value units // (Mandatory, value > 0)
    topWidth value units     // (Mandatory, value > 0)
    Height value units       // (Mandatory, value > 0)
    IsSensitive bool         // (Mandatory)
    ResolutionU value units // (Optional, value > 0)
    ResolutionV value units // (Optional, value > 0)
    Resolution value units // (Optional, value > 0)
    ROtime value units      // (Optional, value > 0)
```

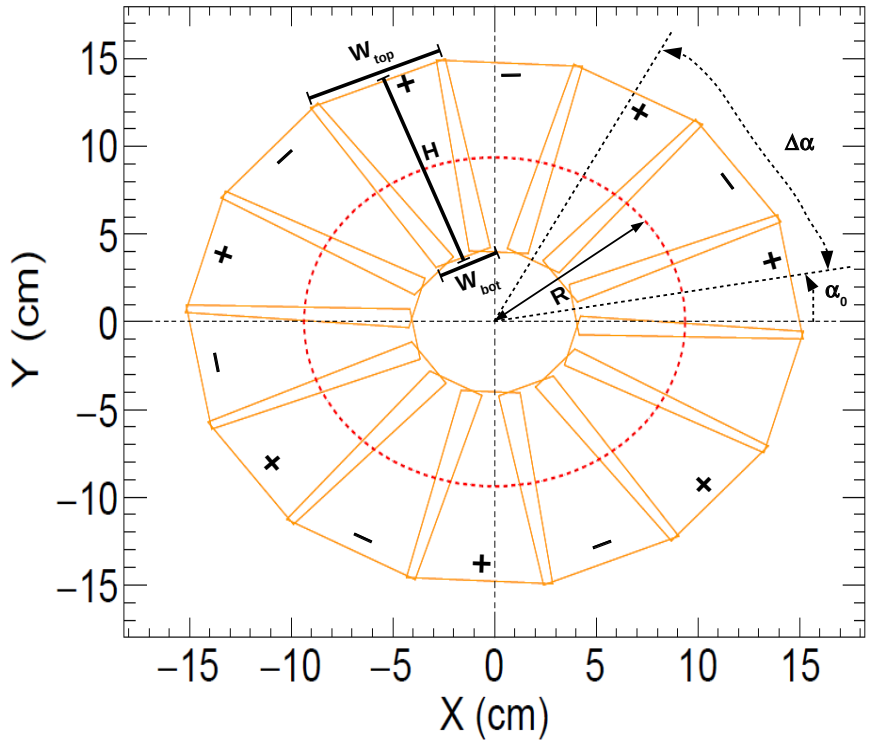

Figure 5: Petal ladder mosaic configuration.

```
    Efficiency value         // (Optional, 0 <= value <= 1)
    InsensFracVneg value     // (Optional, default is zero)
    InsensFracVpos value     // (Optional, default is zero)
    BkgRate value units      // (Optional, default is zero)
    SystemName string        // (Optional)
    LayerName string         // (Optional)
    ResolutionModel string // (Optional)
    EfficiencyModel string // (Optional)
   EndPetal
    ...
EndMosaicLadderPetal
```

As in the case of the plane ladder, an indefinite number of petals can be specified taking care that they not overlap. In figure 5 can be appreciated the parameters needed to build this structure. `LadderRadius` is R, `LadderAlpha0` is α_0, and `LadderN` is half the number of petals. The `LadderShift` parameter in this case is a shift along the disk normal vector. The petals marked by "+" ("−") in figure 5 are shifted by +`LadderShift` (−`LadderShift`) along this direction.

This completes the kind of geometrical structures that can be build in GUARIGUANCHI.

4.6 Intrinsic resolution model

By default in GUARIGUANCHI the spatial resolution of any measurement is equal to the intrinsic resolution (`ResolutionU/V`) of the sensitive element. Moreover, the spatial resolution of a sensor could depend as well on the intersection coordinates and momentum with the sensor's main surface, and on some other parameters. This is a reason behind of the resolution model.

4.6.1 Gas detector resolution model

For the moment there is only one resolution model implemented, mainly to model the spatial resolution of gas detectors as a TPC or a drift-chamber. The syntax is as follows.

```
BeginTPCResolutionModel
   Name string              // (Mandatory)
   A_U value units          // (Mandatory, value > 0)
   B_U value units          // (Mandatory, value > 0)
   C_U value units          // (Mandatory, value > 0)
   A_V value units          // (Mandatory, value > 0)
   B_V value units          // (Mandatory, value > 0)
EndTPCResolutionModel
```

The `Name` parameter defines the resolution model. Any sensor with its parameter `ResolutionModel` set to this name will use this resolution model. The resolution is calculated as follows,

$$\text{ResolutionU}^2 = \text{A}_\text{U}{}^2 + \text{B}_\text{U}{}^2 \sin\phi + \text{C}_\text{U} \sin\theta \times z, \quad (68)$$

$$\text{ResolutionV}^2 = \text{A}_\text{V}{}^2 + \text{B}_\text{V} \times z; \quad (69)$$

where θ, ϕ and z are the polar and azimuthal angles of the momentum and the z-coordinate at the intersection of the track with the volume main surface.

See section 2.2.3 for an example illustrating of this resolution model.

4.7 Detection efficiency model

As in the case of the spatial resolution, by default in GUARIGUANCHI the detection efficiency of any measurement is equal to the intrinsic detection efficiency (`Efficiency`) of the sensitive element. Moreover, the detection efficiency of a sensor could depend as well on the intersection coordinates and momentum with the sensor's main surface, and on some other parameters. This is a reason behind of the efficiency model.

For the time being there is no Efficiency model implemented in GUARIGUANCHI, only a base class.

4.8 Systems configuration

It is useful to label a set of geometry elements as a system. The usefulness has already being shown the examples of sections 2.2.2 and 2.2.4, for the material budget studies. The syntax to define the systems configurations is as follows.

```
BeginSystemsConfiguration
   BeginSystem
    Name string                        // (Mandatory)
    LayersList string1 string2 ....    // (Mandatory)
   EndSystem
    ...
EndSystemsConfiguration
```

where `LayerList` is a list of geometry element `LayerName`. The user can define as many systems as needed, but no two systems can share the same `LayerName`.

4.9 Telescope-DUT configuration

In some cases it is interesting to understand the pointing precision of tracks reconstructed with a set of sensitive elements (Telescope) toward some other elements (DUTs). The use of this feature has been illustrated in examples of sections 2.2.1 and 2.2.2. The syntax for configuring a Telescope-DUT setup is as follows.

```
BeginBeamTestConfiguration
   TelescopeLayersList string1 string2 string3 ...  // (Mandatory)
   DUTLayersList string1 string2 string3 ...        // (Mandatory)
EndBeamTestConfiguration
```

where `TelescopeLayersList` and `DUTLayersList` are lists with the `LayerName`s which define the Telescope and DUTs, respectively.

4.10 Track-Finder algorithms

When calculating the tracking efficiency a co-called `TrackFinderAlgo` object has to be specified. Each such object will be used for a certain regions of the tracker. No two `TrackFinderAlgo` can have overlapping regions, in such a case the program will crash with an error message.

For the time being only one track-finder object has been implemented, the so-called FPCCDTrackFinderAlgo. Its syntax is described in the section below.

4.10.1 FPCCD Track-Finder algorithm

The syntax for this track finder is as follows,

```
BeginFPCCDTrackFinderAlgo    Name string (Mandatory)
   PtMin val units // val > 0 (Mandatory)
   NhitsMin val // val > 0 (Mandatory)
```

```
  PurityMin val // val > 0 (Optional, default value is 1)
  Chi2OndfSeed val // val > 0 (Optional, default value is 3)
  Chi2OndfAdd val // val > 0 (Optional, default value is 3)
  NfakesMaxSeeding val // val >= 0 (Optional, default value is 0)
  NmcSeedEffic val // val >= 0 (Optional, default value is 1000)
  BeginTrackFinderRegion
   positionRangeX min max units // max > min (Optional)
   positionRangeY min max units // max > min (Optional)
   positionRangeZ min max units // max > min (Optional)
   positionRangeR min max units // max > min (Optional)
   positionRangeTheta min max units // max > min (Optional)
   positionRangePhi min max units // max > min (Optional)
   momentumRangeP min max units // max > min and both >= 0 (Optional)
   momentumRangeTheta min max units // max > min (Optional)
   momentumRangePhi min max units // max > min (Optional)
  EndTrackFinderRegion
  BeginSystems
   System string
   ...
  EndSystems
  BeginSeedConfigs
   SeedConfig string1 string2 string3 ...
   ...
  EndSeedConfigs
EndFPCCDTrackFinderAlgo
```

where

- `PtMin` is the $p_t^{\min}$ cut for seeding.
- `NhitsMin` is the minimum number of hits allowed to reconstruct the track.
- `PurityMin` is the minimum allowed value of the track purity, defined as (# good hits)/(# hits). It defined the maximum number of fake hits in the track as follows, $N_{\rm fake}^{\rm max} = N_{hit}(1 - \text{PurityMin})$.
- `Chi2OndfSeed` is the $(\chi^2/ndf)_{\rm max}$ for seed tracks.
- `Chi2OndfAdd` is the χ^2/ndf maximum cut for track-hit association.
- `NfakesMaxSeeding` is the maximum allowed number of fake hits for seeding.
- `NmcSeedEffic` is the number of samplings for the seeding efficiency MC calculation.

The region where this `TrackFinderAlgo` object will be used is specified in the block between `BeginTrackFinderRegion` and `EndTrackFinderRegion`. The region is specified in terms of the track origin and momentum. At least of the possible ranges has to be specified.

The systems considered for the tracking is specified in the block between `BeginSystems` and `EndSystems`. Each entry in this block should be the full system name (`SystemName`).

Finally, the seeding configurations are specified in the block between `BeginSeedConfigs` and `EndSeedConfigs`. Every seed configuration is specified by listing sets of three layer-names (`LayerName`).

See section 3.2.1 for a detailed explanation of the parameters above as well as the process for the tracking efficiency calculation.

4.11 Track cuts

In some cases it is needed to perform cuts on the number of hits within a system for tracks in some ranges of its momentum components (p, θ, ϕ). The syntax is as follows.

```
BeginTrackCuts
   SystemName string          // (Mandatory)
   BeginCut
    pRange min max units      // (Optional)
    phiRange min max units    // (Optional)
    thetaRange min max units  // (Optional)
    NhitsRange min max units  // (Mandatory)
   EndCut
    ...
EndTrackCuts
```

Where SystemName is the name of the system in which the cut on the number of hits will be applied. The data structure between `BeginCut` and `EndCut` implement the cut. The p, θ and ϕ ranges (`pRange`, `phiRange` and `thetaRange`, respectively) identify the range of momentum where the cut `NhitsRange` will be applied. While implementing a cut, at least one range, either p, θ or ϕ range has to be specified.

4.12 Global hit rate scaling

This is a feature is for globally scaling the hit rates on all the sensitive elements of the geometry. This could be useful to study tracking performances in terms of the hit rate labels. The syntax for this feature is very simple.

```
GlobalBkgRateScaling:  scaling
```

where the `scaling` has to be ≥ 0. If not the program will crash with an error message.

This concludes all what is needed to build a geometry. Lets now continue to the next section where the analysis configurations will be described.

5 Analysis configuration

In this section is described the syntax for the main data-card file. The items below describe general parameters for all the analysis which are either mandatory or optional. Later sections will describe the syntax to setup the different analysis. See section 2.2 for some examples illustrating this features.

- **Verbosity**: the `Verbose:` parameter is a boolean optional parameter (default value is false). It is a flag used for debugging. If true is several print-outs will be displayed.
- **Particle parameters**:
  - `ParticleType:` a mandatory parameter. Possible values is the list of stable particles: `e-` or `e+` (electron/positron), `mu+` or `mu-` (muon), `pi+` or `pi-` (pion), `K+` or `K-` (Kaon), `p+` or `p-` (proton), `4He++` (double-ionized Helium-4).
  - `ParticleOrigin:` a mandatory parameter. It is the point (x, y, z) from which the particle is going to be tracker in the process of geometry navigation.
- **Pivot point**: `ReferencePoint` is a mandatory parameter. Trajectories are always parameterized in terms of some quantities defined with respect to a pivot-point.
- **MC seed**: `MCSeed:` is an optional parameter. This seed is used to initialize the random-generator used for the MC calculation of the seeding efficiency (c.f. sec. 3.2.1).
- **Include E_{loss} calculation**: the `IncludeEloss:` parameter is a boolean optional parameter (default value is true). It is a flag used for including the E_{loss} fluctuations in the measurements covariance matrix (c.f. sec. 3.1).
- **Tunning the E_{loss} fluctuations model**: the κ parameter in the E_{loss} fluctuations model (c.f. eq. 41) can be modified if needed with the following syntax,

  ```
  KappaElossFluctuation:  value units
  ```

  where the units should be those of energy.
- **The momentum scan parameters**: a set of values for the momentum or energy variable (see below), polar variable (θ, $\cos\theta$ or η) and azimuthal angle (ϕ). All of them can be specified as a range or as a set of discrete values. Here below are the possibilities
  - **Momentum or energy variable**: the momentum or energy variable can be specified in four ways, either p (momentum), E (energy), E_{kin} (kinetic energy) or E_{kin}/u (kinetic energy per nucleon); where the last one is only useful when studying ionized heavy nuclei. For each of them the either a list of discrete values or a range can be specified. The syntax for all these possibilities is as follows,
    - **List of discrete momentum values**:
      `MomentumValues:  value1 value2 value3 ...  units`
      where as many values as needed can be specified. At least one value needs to be specified.

* **Momentum range**: syntax is as follows,
```
BeginMomentumScan
  Nbins integer    // (Mandatory)
  pMin value units // (Mandatory)
  pMax value units // (Mandatory)
EndMomentumScan
```

* **List of discrete energy values**:
`EnergyValues: value1 value2 value3 ... units`
where as many values as needed can be specified. At least one value needs to be specified.

* **Energy range**: syntax is as follows,
```
BeginEnergyScan
  Nbins integer    // (Mandatory)
  EMin value units // (Mandatory)
  EMax value units // (Mandatory)
EndEnergyScan
```

* **List of discrete kinetic energy values**:
`EkinValues: value1 value2 value3 ... units`
where as many values as needed can be specified. At least one value needs to be specified.

* **Kinetic energy range**: syntax is as follows,
```
BeginEkinScan
  Nbins integer    // (Mandatory)
  EkinMin value units // (Mandatory)
  EkinMax value units // (Mandatory)
EndEkinScan
```

* **List of discrete kinetic energy per nucleon values**:
`EkinPerUValues: value1 value2 value3 ... units`
where as many values as needed can be specified. At least one value needs to be specified.

* **Kinetic energy per nucleon range**: syntax is as follows,
```
BeginEkinPerUScan
  Nbins integer    // (Mandatory)
  EkinPerUMin value units // (Mandatory)
  EkinPerUMax value units // (Mandatory)
EndEkinPerUScan
```

If no p, E, E_{kin} or E_{kin}/u values are specified, then a single $p = 2$ GeV/c value will be will be used. Non of the momentum or energy values can be negative, if so the program will crash with an error message. Furthermore, in the case of

specifying energy values, E, all the specified values have to be larger than the particle mass, otherwise the program will crass with an error message.

- **Polar variable**: the polar variable can be specified in three ways, either θ, $\cos\theta$ or η. For each of them the either a list of discrete values or a range can be specified. The syntax for all these possibilities is as follows,
  - **List of discrete θ values**:
    `PolarAngleValues: value1 value2 value3 ... units`
    where as many values as needed can be specified. At least one value needs to be specified.

  - θ **Range**:
    ```
    BeginPolarAngleScan
      Nbins integer        // (Mandatory)
      thetaMin value units // (Mandatory)
      thetaMax value units // (Mandatory)
    EndPolarAngleScan
    ```

  - **List of discrete $\cos\theta$ values**:
    `CosThetaValues: value1 value2 value3 ... units`
    where as many values as needed can be specified. At least one value needs to be specified.

  - $\cos\theta$ **Range**:
    ```
    BeginCosThetaScan
      Nbins integer           // (Mandatory)
      costhetaMin value units // (Mandatory)
      costhetaMax value units // (Mandatory)
    EndCosThetaScan
    ```

  - **List of discrete η values**:
    `EtaValues: value1 value2 value3 ... units`
    where as many values as needed can be specified. At least one value needs to be specified.

  - η **Range**:
    ```
    BeginEtaScan
      Nbins integer      // (Mandatory)
      etaMin value units // (Mandatory)
      etaMax value units // (Mandatory)
    EndEtaScan
    ```

- **Azimuthal angle**:
  - **List of discrete values**: syntax is as follows,
    `AzimuthalAngleValues: value1 value2 value3 ... units`

where as many values as needed can be specified. At least one value needs to be specified.

  * **Range**: syntax is as follows,

```
BeginAzimuthalAngleScan
  Nbins integer      // (Mandatory)
  phiMin value units // (Mandatory)
  phiMax value units // (Mandatory)
EndAzimuthalAngleScan
```

- **Momentum resolution representation**: In the case where p or p_t is part of the track parameters, its resolution can be represented in different ways: $\sigma(p)/p$, $\sigma(p)$ or $\sigma(1/p)$. The default one is $\sigma(p)/p$. A parameter allows to select the representation as `MonResolRepresentation: string`, where `string` can only have the values: `sigma(Pt)/Pt`, `sigma(Pt)` or `sigma(1/Pt)`, corresponding to the $\sigma(p)/p$, $\sigma(p)$ or $\sigma(1/p)$ representations, respectively.

- **Geometry list**: the list of geometries is specified as follows,

```
BeginGeometries
  fullpath/geo-data-card1.txt
  fullpath/geo-data-card2.txt
  ...
EndGeometries
```

  where `fullpath` is the full path of the location of the geo-data-card. The user can specify as many geometries as needed. At least one geometry needs to be specified, otherwise the program will crash with an error message.

- **Ranges for geometry visualization**: as a default a geometry will be visualized in a range on X, Y and Z that contain all the geometry elements. It is possible to specify a set of sub-ranges for a better appreciation of the geometry. This can be made as follows,

```
BeginGeoRanges
  BeginRange
    XRange min max units
    YRange min max units
    ZRange min max units
  EndRange
  ...
EndGeoRanges
```

  As many ranges as needed can be specified. At least one of the `XRange`, `YRange` or `ZRange` parameters have to be specified. For example, if only the `ZRange` is specified, the other two will be set equal to a high range.

- **Voxeling**: this is a feature that allows to reduce the computation time in cases where it is known beforehand the regions of the geometry where the particles will pass through. It mainly consist in defining a set of regions. For each specified geometry, a sub-list of geometry elements will be built consisting of those contained in the specified regions. This sub-list will then be used for the calculation of the track-geometry intersections.

An illustration of its usage can be found in the example section 2.2.3. The syntax is as follows,

```
BeginVoxeling
  BeginVoxel
    xRange min max units
    yRange min max units
    zRange min max units
    rRange min max units
    thetaRange min max units
    phiRange min max units
  EndVoxel
  ...
EndVoxeling
```

As many voxels as needed can be specified. At least one of the `xRange`, `yRange` or `zRange`, or the polar-coordinates equivalent, `rRange`, `thetaRange` or `phiRange`, parameters have to be specified. For example, if only the `zRange` is specified, the other two will be set equal to a high range.

- **Output file generic name**: this a mandatory parameter which is specified in the following way `OutputFile: fullpath/outfile`, where `fullpath` is the location where the output file will be written. The output files, either `.pdf` or `.root` will be named as `fullpath/outfile.pdf` and `fullpath/outfile.root`, respectively. If the `fullpath` directory doesn't exist it will be created.

- **Save output in a .root file**: the output from all the analysis turned on can be saved in a `.root` file by setting the boolean parameter `SavePlots:` to `true` (default value is `false`).

The set of configuration parameters described above are generic for all the analyses. In the sections below is described how to configure the different analyses.

5.1 Geometry visualization analysis

All the examples in section 2.2 have illustrated the geometry visualization feature of the package. There are several parameters to set this up.

- **Geometry printout**: this can be turned on by setting the boolean parameter `PrintGeometry:` to `true` (default value is `false`). If set to `true` a printout of all the specified geometries will be made.

- **Geometry weights printout**: this can be turned on by setting the boolean parameter `PrintGeometryWeight:` to `true` (default value is `false`). If set to `true` a printout of weights all the specified geometries will be made with a separation by system.

- **Geometry plotting**: this can be turned on by setting the boolean parameter `PlotGeometry:` to `true` (default value is `false`). By default a projection of the geometry on the $Z-Y$, $Z-X$ and $X-Y$ will be plotted. Additional features can be turned on in the following way,

– **World volume plotting**: this can be turned on by setting the boolean parameter `PlotWorldVolume:` to `true` (default value is `false`). The world volume will be represented by a set of dotted-red lines.

– **R-Z geometry projection**: this feature is only useful for geometries with cylindrical symmetry with respect to z-axis. It can be turned on by setting the boolean parameter `DoRZGeoRepresentation:` to `true` (default value is `false`).

– **Plotting some tracks**: this is an useful feature to understand which geometry elements the track will intersect, for a given initial particle origin and momentum. It can be turned on by setting the the boolean parameter `PlotSomeTracks:` to `true` (default value is `false`). This feature turns on the plotting of a set of trajectories along with its intersections with the geometry. For each specified (θ, ϕ) values, a number of trajectories corresponding to some values of momentum will be plotted. By default, 10 momentum values uniformly spaced between the minimum and maximum specified momenta will be used. If the boolean parameter `UseAllMomVals:` is set to `true` (default value is `false`), then all the specified momentum values will be used.

5.2 Material budget analysis

The material budget analysis has been illustrated in the sections 2.2.2 and 2.2.4. To turn on this analysis the boolean parameter `DoMaterialBugdetAnalysis:` has to be set to `true`. Furthermore, the following data block has to be specified,

```
BeginMatBudgetAnalysisParams
  mom_min value units // (Mandatory)
  mom_max value units // (Mandatory)
EndMatBudgetAnalysisParams
```

The ϕ averaged material budget, cumulatively separated for the different systems, will be plotted vs the values of the polar angles specified. For each geometry three plots will be produced, one for the minimum and maximum momenta specified, `mom_min` and `mom_max`, respectively, as well as one for their average.

5.3 Track parameters resolution analysis

This analysis has been illustrated in all the examples described in section 2.2. Its output is a set of plots of the track parameters resolution vs momentum. To turn it on the boolean parameter `DoTrkResolAnalysis:` has to be set to `true`. Furthermore, the following data block has to be specified,

```
BeginTrkResolAnalysisParams
  NhitsMin integer             // (Mandatory, integer > 0)
  SameRange bool               // (Optional, default is true)
  UseAllMomVals bool           // (Optional, default is false)
  PlotMaterialBudget bool      // (Optional, default is false)
  PlotDOCAatHighMom bool       // (Optional, default is false)
  PlotPhiAveraged bool         // (Optional, default is false)
```

```
  PlotOnlyPhiAveraged bool     // (Optional, default is false)
  PlotPerformancesVsTheta bool // (Optional, default is false)
  UseLogYAxes bool             // (Optional, default is false)
EndTrkResolAnalysisParams
```

where this set of parameters have the following meaning,

- `NhitsMin`: Minimum number of hits requested to fit the track.
- `SameRange`: if false each parameter resolution vs momentum plot will have an adapted vertical axis range for each specified value of (θ, ϕ). If true, the same vertical range will be used.
- `UseAllMomVals`: Some plots will be made using a reduced number of p values (10 values). If set to `true`, then all the specified p values will be used.
- `PlotMaterialBudget`: if set to `true` the total material budget encountered by the particles vs p will be plotted.
- `PlotDOCAatHighMom`: if set to `true` the distance of closest approach to the pivot point as high-momentum will be plotted.
- `PlotPhiAveraged`: if set to `true` the ϕ-averaged track parameters resolution vs momentum will be plotted.
- `PlotOnlyPhiAveraged`: if set to `true` only the ϕ-averaged track parameters resolution vs momentum will be plotted.
- `PlotPerformancesVsTheta`: if set to `true` the track parameters resolution vs polar variable will be plotted for a set of values of p.
- `UseLogYAxes`: if set to `true` a logarithmic vertical scaled will be used for all the plots.

5.4 Telescope analysis

This analysis has been illustrated in the examples of sections 2.2.1 and 2.2.2. It is very important that for each specified geometry a Telescope-DUT configuration has been set (see section 4.9). The output will be a two set of plots: the track resolution parameters vs momentum calculated with the telescope planes, and the telescope pointing resolution at the set of DUTs specified. To turn it on the boolean parameter `DoTelescopeAnalysis:` has to be set to `true`. Furthermore, the `TrkResolAnalysisParams` data block has to be specified as in the case of the track parameters resolution analysis (see section 5.3).

5.5 Tracking efficiency analysis

This analysis has been illustrated in some of the examples described in section 2.2. Its output is a set of plots of the tracking efficiency and average track parameters resolution vs momentum. To turn it on the boolean parameter `DoPseudoEfficVsMon:` has to be set to `true`. Furthermore, the following data block has to be specified,

```
BeginEfficAnalysisParams
```

```
  SameRange bool                // (Optional, default is true)
  UseAllMomVals bool            // (Optional, default is false)
  PlotPhiAveraged bool          // (Optional, default is false)
  PlotOnlyPhiAveraged bool      // (Optional, default is false)
  PlotPerformancesVsTheta bool // (Optional, default is false)
  UseLogYAxes bool              // (Optional, default is false)
EndEfficAnalysisParams
```

where this set of parameters have the following meaning,

- `SameRange`: if false each tracking efficiency and parameter resolution vs momentum plot will have an adapted vertical axis range for each specified value of (θ, ϕ). If true, the same vertical range will be used.
- `UseAllMomVals`: Some plots will be made using a reduced number of p values (10 values). If set to `true`, then all the specified p values will be used.
- `PlotPhiAveraged`: if set to `true` the ϕ-averaged tracking efficiency as well as track parameters resolution vs momentum will be plotted.
- `PlotOnlyPhiAveraged`: if set to `true` only the ϕ-averaged tracking efficiency as well as track parameters resolution vs momentum will be plotted.
- `PlotPerformancesVsTheta`: if set to `true` the tracking efficiency and track parameters resolution vs polar variable will be plotted for a set of values of p.
- `UseLogYAxes`: if set to `true` a logarithmic vertical scaled will be used for all the track parameters resolution plots.

It is very important that for each geometry a set track-finder algorithms have been specified (c.f. sec. 4.10). In the contrary the tracking efficiency will be zero for all the specified values of the momentum.

6 Class structure

This section describes the code structure of the GUARIGUANCHI package. It is a series of modules or classes, each one describing objects with specific properties and functions. Roughly, the code is divided as follows,

- **Global tools**: this a class has a set of features used by the whole code.
- **Geometry class**: there is a class (`GGeometry`) containing the geometry description. Each geometry contains a set of geometry elements (`GGeoObject`), a magnetic field (`GBField`), a list of resolution models (`GResolutionModel`) and a list of efficiency models (`GEfficiencyModel`).
- **Trajectory class**: there is a class (`GTrajectory`) which contains all the information for the calculation of the particle's trajectory from the initial position and momentum. It contains a magnetic field (`GBField`) and in it is also defined the track parameterization.

- **Tracker class**: this object contains a geometry (`GGeometry`) as well as a trajectory (`GTrajectory`) as internal variables. It performs all the need calculations related to the track parameters resolution and pseudo-efficiency: track-geometry intersections, space point covariance matrix due to multiple scattering, track intersections derivatives, ... (see section 3).

- **Central class**: the main class of the code is called `Guariguanchi`. It is the class which takes as input the data-card, builds the geometries and performs all the booked analyses.

All these objects mentioned above will be detailed in the following sections. Some guide lines on how to extend the code are also given.

6.1 Global tools class

The class `GGlobalTools` contains a set of global variables used by the whole package. In it are also defined global functions. Every object of the package of has as an internal member a pointer to a global `GGlobalTools` object.

`GGlobalTools` contains the GUARIGUANCHI package unit system. It also contains the list of particles attributes (charge, mass, atomic weight, anti-particle, ...) and possible material attributes (X_0, density, mean ionization energy, average Z/A, ...) needed for the calculation of the MS covariance matrix as well as E_{loss} fluctuations.

The class also contains a set of global variables (*e.g.* minimum distance cuts for numerical derivatives calculations) used in the numerical calculations in the code.

6.1.1 Implementing a new unit

To implement a new unit, or combination of units just add the corresponding element to the `units std::map` inside the function `GGlobalTools::FillTables`.

6.1.2 Implementing a new particle

To implement a new particle just add the corresponding particle name with its attributes to the `ParticleMap std::map` inside the function `GGlobalTools::FillTables`.

6.1.3 Implementing a new material

To implement a new material just add the corresponding material name with its attributes to the `MaterialMap std::map` inside the function `GGlobalTools::FillTables`.

6.2 Surface object classes

A so-called `GGeoObject` (see section 6.3) is a physical volume, which is defined by a set of boundary surfaces. In GUARIGUANCHI, these delimiting surfaces are implemented in a class called `GSurfaceObject`. This class is a base class from which different kind of surfaces are derived. Currently these are:

- `GSurfacePlane`: surface plane class.

- `GSurfacePetal`: surface "petal" or trapezoid class.

- `GSurfaceDisk`: surface disk class.
- `GSurfaceDiskSection`: surface disk section class.
- `GSurfaceCone`: surface cone class.
- `GSurfaceConeSection`: surface cone section class.
- `GSurfaceCylinder`: surface cylinder class.
- `GSurfaceCylinderSection`: surface cylinder section class.

Each surface object is built by specifying a name, position, rotation matrix, pointer to a `GGlobalTools` object, and a set of parameters which depend on the surface type. Each surface has a local coordinate system (U, V, W) with the proper transformation equations to and from the global coordinate system.

6.2.1 Implementing a new Surface class

In order to implement a new surface type just follow the examples of the classes above. The developer should fill-out the "virtual" functions in `GSurfaceObject` with the corresponding algorithms adapted to the new surface type.

6.3 Geometry object classes

Each element of a geometry is an object derived from the base-class `GGeoObject`. Every GGeoObject is a physical volume delimited by a set of surfaces (`GSurfaceObject`, see section 6.2). It also contains a so-called "main-surface", from which local coordinates (U, V, W) are defined. As in the case of a surface object, a `GGeoObject` is built by specifying a set of mandatory parameters such as name, position, rotation matrix, material, pointer to a `GGlobalTools` object, and a boolean (`IsSensitive`) specifying if the physical volume is a sensitive material, and a set of parameters which depend on the physical volume type. If the `IsSensitive` parameter is true then an additional set of parameters need to be specified,

- Intrinsic single point resolution, *i.e.* σ_U and σ_V,
- Intrinsic measurement detection efficiency (for pseudo-efficiency calculation),
- Readout time (for pseudo-efficiency calculation),
- Base rate in hits per unit of surface per unit of time (for pseudo-efficiency calculation).

The `GGeoObject` is a base-class from which different kind of physical volumes are derived. Currently these are:

- `GGeoPlane`: physical volume plane class.
- `GGeoPetal`: physical volume "petal" or trapezoid class.
- `GGeoDisk`: physical volume disk class.
- `GGeoDiskSection`: physical volume disk section class.

- `GGeoCone`: physical volume cone class.
- `GGeoConeSection`: physical volume cone section class.
- `GGeoCylinder`: physical volume cylinder class.
- `GGeoCylinderSection`: physical volume cylinder section class.

6.3.1 Implementing a new geometry object class

In order to implement a new physical volume type just follow the examples of the classes above. The developer should fill-out the "virtual" functions in `GGeoObject` with the corresponding algorithms adapted to the new physical volume type.

6.3.2 Implementing a new Mosaic geometry

As mentioned in section 4.5 the so-called Mosaic geometries define ways to position in a regular pattern in space simple geometry elements (`GGeoObject`). If a new pattern has to be implemented follow examples mentioned in section 4.5, which are implemented in the file `src/Setup.cc`.

6.4 B-field classes

Each geometry also contains a magnetic field. It is implemented with a set of classes derived from the base-class `GBField`. The derived classes currently implemented are the following,

- `GBFieldConstant`: this is a simple implementation of a constant magnetic field on the whole space.
- `GBFieldMultipleSteps`: this is a piece-wise implementation of the magnetic field. It is defined with a list of non-overlapping volumes with a constant magnetic field defined inside each one, and a constant B-field outside each of the volumes.

6.4.1 Implementing a new B-field class

In order to implement a new magnetic field type just follow the examples of the classes above. The developer should fill-out the "virtual" functions in `GBField` with the corresponding algorithms adapted to the new magnetic field type.

6.5 Resolution model class

Each geometry contains also a list of resolution models. It is a function which calculates the measured space point resolution from the intrinsic physical volume resolution, the coordinates and momentum of the particle's intersection with the volumes, as well as a set of parameters. A resolution model base-class is called `GResolutionModel`. The derived classes currently implemented are the following,

- `GResolutionModelTPC`: this class is intended to describe the resolution model of gas tracking detector (TPC or drift chamber). It definition has been already described in section 4.6.1.

6.5.1 Implementing a new resolution model class

In order to implement a new resolution model type just follow the examples of the classes above. The developer should fill-out the "virtual" functions in `GResolutionModel` with the corresponding algorithms adapted to the new resolution model type.

6.6 Efficiency model class

Each geometry contains also a list of efficiency models. It is a function which calculates the measured space point detection efficiency from the intrinsic physical volume detection efficiency, the coordinates and momentum of the particle's intersection with the volumes, as well as a set of parameters. Currently only a base-class called `GEfficiencyModel` has been implemented.

6.6.1 Implementing a new efficiency model class

In order to implement a new efficiency model type just follow the examples of the classes above. The developer should fill-out the "virtual" functions in `GEfficiencyModel` with the corresponding algorithms adapted to the new efficiency model type.

6.7 Geometry class

A geometry is described by an object called `GGeometry`. It contains a world volume (`GGeoObject`), a list of geometry elements (`GGeoObject`), a list of resolution (`GResolutionModel`) and efficiency (`GEfficiencyModel`) models. It contains a set of functions to access it geometry elements and to calculated the measured space point resolution and efficiency.

6.8 Trajectory classes

The particle's trajectory and parameterization will depend on the magnetic field of the region where the particle is moving. The particle trajectory is implemented in GUARIGUANCHI with a set of classes derived from the `GTrajectory` base-class. The derived classes currently implemented are the following,

- `GTrajectoryStraight`: straight trajectory in zero constant magnetic field. The track is parameterized with four parameters: positions x_0 and y_0, and tilts $\tan\alpha_x$ and $\tan\alpha_y$ of the trajectory at the pivot point.
- `GTrajectoryHelix`: helix trajectory in a non-zero constant magnetic field. The track is parameterized as in [3] with five parameters: d_ρ, d_z, ϕ_0, $tan\lambda$ and signed p_t.
- `GTrajectoryMultipleSteps`: corresponding trajectory on a piece-wise magnetic field (`GBFieldMultipleSteps`), which consist of pieces of straight and helix trajectories. The track is parameterized as in LHCb [**?**] with five parameters: positions x_0 and y_0, tilts $\tan\alpha_x$ and $\tan\alpha_y$, and signed momentum p of the trajectory at the pivot point.

The equation of motion is implemented in the functions

`GetTrueTrajectoryCoordinates(double s)`,

`GetTrueTrajectoryUnitMon(double s)`,

where s is the path length along the trajectory from origin of the particle. The track parameterization is implemented in the functions

`GetFitTrackCoordinates(double dummy, double sinit)`,
`GetFitTrackMomentum(double dummy, double sinit)`,

where `dummy` is track parameterization dummy parameter, and `sinit` is the path length from the particle's origin corresponding to the dummy parameter.

6.8.1 Implementing a new Trajectory class

In order to implement a new trajectory type just follow the examples of the classes above. The developer should fill-out the "virtual" functions in `GTrajectory` with the corresponding algorithms adapted to the new trajectory type.

6.9 Track Finder class

For the tracking efficiency calculation each geometry should contain a list of so-called `TrackFinderAlgo` objects. Each object will define a track-finder algorithm in a detector region for the calculation of the tracking efficiency. This is implemented with a set of classes derived from the base-class `GTrackFinderAlgo`. These classes should just contain the set of parameters for the specific track-finder algorithm to be used in the tracking efficiency calculation. The derived classes currently implemented are the following,

- `GTrackFinderAlgoFPCCD`: FPCCD tracking finder parameters (c.f. section 3.2.1).

The actual implementation of the tracking efficiency calculation is performed inside the Tracker class (c.f. section 6.10) inside the function `GTracker::GetFitTrackPseudoEfficiency`.

6.9.1 Implementing a new Track-finder algorithm

In order to implement a new `GTrackFinderAlgo` object just follow the examples of the classes above. The user should define the parameters needed by the new track-finder algorithm. For the corresponding tracking efficiency calculation the user should also follow the implementation of the tracking efficiency using the FPCCD algorithm. A new function similar to `GTracker::GetFitTrackPseudoEfficiency_FPCCD` should be implement.

6.10 Tracker class

The `GTracker` class is the master class for the calculation of tracking performances. It contains a geometry (`GGeometry`) and a trajectory (`GTrajectory`). This class contains all the functions needed to calculate the resolution on the track parameters, the material budget and the pseudo-efficiency. These set of function are implemented numerically, so than their implementation is independent on the nature of the trajectory and geometry.

6.11 Guariguanchi class

Finally, the main class of the GUARIGUANCHI package is called `Guariguanchi`. This class is able to take as input a data-card and out of that build the geometries and perform all the booked analysis and plots.

The readout of the main and geometry data-cards is implemented in the `src/Setup.cc` file. For every new feature implemented in the GUARIGUANCHI class structure, a corresponding reader needs to be implemented in this file, *e.g.* a new geometry, B-field, resolution or efficiency model. The developer is adviced to follow the examples already implemented in this file.

All the geometry related operations are implemented in `src/GeometryHandler.cc` file: geometries fill-up, printout, weight printout and plotting (geometry visualization, see section 5.1).

Finally, all the other analyses, Track parameters resolution (see section 5.3), telescope (see section 5.4) and material budget (see section 5.2), are implemented in `src/AnalysisHandler.cc` file with the functions, `doTrkResolAnalysis()` and `doMaterialBudgetAnalysis()`, respectively.

6.11.1 Implementing a new analysis

The developer can modify, improve and extend the code as its needed for his application. If no new features need to be implemented but just new plots, we advise the user to follow the examples already implemented inside `src/AnalysisHandler.cc`.

7 Summary and outlook

GUARIGUANCHI is a standalone package meant for fast calculations of tracking performances in terms of track parameters resolution, tracking pseudo-efficiency and material budget. In its current state a set of simple geometries and magnetic fields can be implement, but which cover most of the applications in the realm of particle and nuclear physics. A full description of the code structure has been given as well as guide-lines to extend the code to cover further applications. We expect that this tool box will be useful in the process of geometry optimization for applications in domains where charged particle tracking is an important feature of the experimental setup. GUARIGUANCHI package is a free-sourced software and any improvements to it are more than welcome.

8 Acknowledgements

This package is the product of my activities on R&D of CMOS pixel sensors within the PICSEL group at IPHC Strasbourg in the context of the ILD vertex detector optimization. I am grateful for the useful discussions with the members of the PICSEL group M. Winter, J. Baudot and A. Besson. I also would like to acknowledge the useful feedback from the ILD vertex detector community. Finally, I would like to thank I. Belikov from the ALICE group at IPHC Strasbourg for the long discussions about the implementation of subtle calculations performed in this package.

References

[1] ILD collaboration web page:
https://confluence.desy.de/display/ILD/The+ILD+Organisation.

[2] ROOT web page: *https://root.cern.ch/*.

[3] Y. Ohnishi, *Track parameterization*, KEK, National laboratory of high energy physics, Tsukuba, Japan, 1997.

[4] T. Mori *et al.*, *Study of Tracking and Flavour Tagging with FPCCD Vertex Detector*, *Proceedings of the International Workshop on Future Linear Colliders (LCWS13)*, 11-15 November 2013, Tokyo, Japan. arXiv:1403.5659 [physics.ins-det].

[5] SiD collaboration web page: *http://pages.uoregon.edu/silicondetector/*.

[6] B. Abelev *et al.* [ALICE collaboration], *Technical Design Resport for the Upgrade of the ALICE Inner Tracker System*. See also: `http://aliceinfo.cern.ch/ITSUpgrade/`.

[7] F. Gaede *et al.* [ILD collaboration], *Track reconstruction at the ILC: the ILD tracking software*, *Journal of Physics: Conference Series* **513** (2014) 022011. doi:10.1088/1742-6596/513/2/022011.

[8] S. Terzo [ATLAS collaboration], *The Phase-II ATLAS ITk pixel upgrade*, *Journal of Instrumentation*, Volume 12, July 2017.

[9] S. Mersi [CMS collaboration], *Phase-2 Upgrade of the CMS Tracker*, *Nucl. Part. Phys. Proc.* 273-275 (2016) 1034-1041 (2016-06-02). DOI: 10.1016/j.nuclphysbps.2015.09.162.

[10] Belle-II web page: `https://www.belle2.org/`.

[11] A. Alves *et al* [LHCb collaboration], *LHCb Tracker Upgrade Technical Design Report*, CERN-LHCC-2014-001, LHCB-TDR-015.

[12] V. Patera *et al*, *The Foot (Fragmentation Of Target) Experiment*, *Proceedings of the 26th International Nuclear Physics Conference* (INPC2016). 11-16 September, 2016. Adelaide, Australia. FOOT web page: `https://web.infn.it/f00t/index.php/it/`.

[13] K.A. Olive *et al.* (*Particle Data Group*), Chin. Phys. C, **38**, 090001 (2014).

[14] A. Mastroserio *et al.*, *Simulation tools for the ALICE ITS Upgrade*, *http://aliceinfo.cern.ch/ITSUpgrade/sites/aliceinfo.cern.ch.ITSUpgrade/files/documents/Upgrade_IN.pdf*.