Vertex reconstruction in ALICE

Anne KIEFFER
Supervisor: Matteo CONCAS

June to August 2019

1 Upgrade of the Inner Tracking System at ALICE

The A Large Ion Collider Experiment (ALICE) at the Large Hadron Collider (LHC) takes data from proton-proton and lead-lead nuclei collisions, and aims to study and characterise the quark-gluon plasma. That is a state of the matter created in a regime of extremely high energy and density. The ALICE apparatus is composed by several sub-detectors developed by mean of many different technologies. The innermost detector, the one involved in this project, is called the Inner Tracking System (ITS). For the Run 3 at the LHC, scheduled to start in 2021, it is planned replacement of the old ITS detector with a brand new one, entirely based on silicon pixel detectors. The new ITS design counts seven concentric cylindrical layers of different lengths, see Figure 1. Each layer presents a finer structure, that is made of partially overlapped *staves* to maximise the active (the part that can actually measure the transition of a charged particle) surface of the cylinder, see Figure 1. The whole ALICE apparatus, ITS included, is embedded into a solenoidal magnetic field of $\sim 0.5T$ with its axis parallel to the particle beam axis.

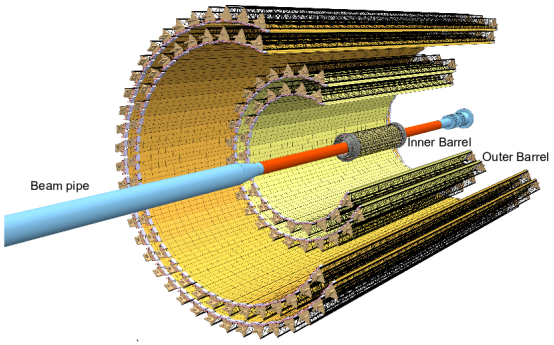

Figure 1: Layout of the new ITS

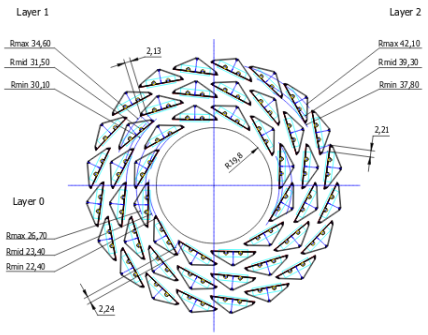

Figure 2: Cross section of the Inner Barrel

Source : *Technical Design Report for the Upgrade of the ALICE Inner Tracking System*

2 Reconstruction in the upgraded apparatus

As per data reconstruction it is intended the process to get, from data acquired and pre-processed by electronic readout and digitisation, the reconstruction of the charged physical objects that generated them, interacting with the detector itself. In particular, ALICE is planning to apply a continuous readout approach, that means to reconstruct and save the data related to collisions with no triggered data acquisition. It becomes necessary to apply some online data reduction to be

able to skim the data to be then stored and later analysed. Otherwise, the overall amount of data produced would be just too large to be kept and processed offline (out of the data taking). One of the mission in data reconstruction for the ITS is to give a good estimation of the point on which the collision system happen, called **interaction vertex**. It is never exactly known *a priori*. To do this, only the three innermost layers of the ITS are used. Particles colliding with the detector activate several pixels grouped in clusters, by releasing some charge. From this raw information, we can extract three dimensional points. The position of these 3D points is obtained by computing the centre of mass of the single cluster of pixel. Therefore, since its position is a placeholder for the whole pixel information, we will still call **cluster** this point. These clusters are then connected and fitted to reconstruct the trajectories of the particles and then, by backward propagation, the interaction vertex. It is important to note that we can only directly track charged particles, due to the technology adopted int the ITS.

2.1 The vertex finding algorithm

To better describe the algorithm for the vertex finding, it is necessary to introduce few more concepts, useful to contextualise also the approximations and the assumptions made. The first is the *transverse plane*: a plane which is orthogonal to the beam axis. This plane is particularly interesting since it is the plane where the projection of the trajectories of the charged particles bend due to the presence of the magnetic field. The reference system usually adopted to describe the geometry has the z axis that is parallel to the beam axis, the x axis that points to the centre of the ring of the accelerator and y axis that completes the right trihedron. In this notation, the transverse plane is the one spanned by the axes x and y. The other useful concept to introduce is the transverse momentum of a particle (p_{T}). It represents the projection of the momentum vector on the transverse plane. This quantity is crucial to do the so-called particle identification, since it carries both information on the mass and the charge of a particle. With a constant magnetic field like the one in ALICE, we associate higher p_{T} with more straight trajectories. The algorithm used to find the vertex works under the assumption that since the layers are very close to each other and to the beam axis (the third layer has a radius of less than 4 cm), the trajectory of a particle can be approximated to a straight line. This is not true for particles which have a very low momentum, which are strongly bend by the magnetic field and therefore show very curved trajectory on the transverse plane. The main idea is to match the right clusters together and then interpolate straight lines passing through them, and use their intersection to localise the vertex. The peculiarity of this approach is that no prior assumption on the vertex position is made, to avoid initial bias.

The algorithm is divided into three phases: the first step is the **tracklet reconstruction**. A *tracklet* is a segment connecting two clusters on two adjacent layers. The whole point of this preliminary selection is to define a criterion to match the clusters. Without any selection this would be a pure combinatorial problem. However, clusters from layer 0 (the innermost) cannot be matched with every cluster from layer 1. To match the cluster, we impose a selection in the difference in the azimuthal Φ coordinate of pairs of clusters. It corresponds to select only the pairs of clusters likely to be created by very high momentum particles.

The subsequent step is **tracklet selection**. We try to find one or several tracklets from layer 1 to 2 that has one cluster in common, and that is well aligned with the first tracklet. If we find one, we can decide to validate the tracklet from 0 to 1. To do this, we compare the tangent of relative angle λ that the projection of the tracklet on the yz plane makes with the detector's z axis. If the difference in $\tan\lambda$ is small, that is if the tracklets are parallel enough, the tracklet from layer 0 to 1 is selected and is now called a **line**.

Ultimately for the **vertex finding**, the lines are grouped together in clusters. We start with two lines and compute their possible intersection or, most frequently, the medium point of the segment representing the distance of closest approach (DCA) of two 3D lines. Only lines whose DCA within a selected threshold are selected. Then, we scan the other lines and test if they are close to the intersection point within the same tolerance selection. If they are, we add them to the cluster and

the new vertex candidate is the centre of mass of all the lines of the cluster. If they are not, we try to create new clusters. Several vertices can be reconstructed, due to the nature of the algorithm, which is conceived to be able to reconstruct more than one vertex on the same fraction of data, whenever we have more than one collision in the same *readout frame (ROFrame)*.

With the continuous readout approach, data acquisition has no trigger, that means that the time is divided into time windows called *timeframes*. For the ITS a further division is done at the level of readout frames (duration $\sim 5\mu s$) which represents the *atomic* data unit. The time windows are often shorter than the overall **event** development, that means that the particle collisions are likely to be shared across up to 3 ROframes.

The two main indicators used to describe how well the algorithm works in the different collision systems (pp, p-Pb, PbPb) are the reconstruction efficiency and the vertex resolution. The efficiency is the ratio between the number of real vertices and reconstructed ones. The resolution depends on the residual between the position of found vertex and the real vertex. Indeed, another indicator is the capability of the algorithm to reconstruct only real vertices and minimise the so-called "fake-pileup" effect, where too loose selections may lead to bad matches, therefore creating enough noise to lead to wrong vertices identification. These values can of course only be estimated by comparing obtained results with the Monte Carlo truth.

3 Goal of my project

The efficiency and precision of the vertex reconstruction algorithm depend on several parameters, many have been mentioned already, that need to be tuned. The Φ angle and the delta tan lambda ($\Delta tan(\lambda)$) selection are examples of very effective cuts. There are also parameters for the last step, when clustering the lines. The main aim of my project was to study the behavior of the algorithm when varying the value of the cuts. This is important to measure and create some calibration that takes account of external conditions such as the multiplicity of generated tracks and the kind of collision systems. The final goal is to find the best cuts that maximise efficiency and minimise residuals (improve resolution).

4 Outline of my work

4.1 Getting familiar with software: the ROOT framework and O^2

The first step when I arrived at CERN was to get familiar with ROOT, the data analysis framework used there, and with the vertex reconstruction algorithm. The latter is included in the so-called Online-Offline (O^2) framework ALICE is developing for the Run3. To do this, I first wrote a simple input/output program (in the form of a C++ macro) that reads some data from a file and creates histograms. This was very useful to understand and familiarise with the way data is stored and histograms are often plotted. Later on, I worked on a macro that plots several histograms to visualize useful insights about clusters on layers, tracklets, lines and reconstructed vertex coordinates. This kind of plots are crucial to profile and check the algorithm behaviour. I improved the readability of the plots by including more complete information and added further plots to complete the inspection. In addition, I wrote a function to plot the histograms to reduce code repetitions and ease further extensions of it.

4.2 Study of the tracklet reconstruction efficiency as a function of the Φ selection

After this introduction, I started to study the performances of the algorithm. I wrote a macro to study how the efficiency of the tracklet reconstruction phase varies with the Φ selection. The

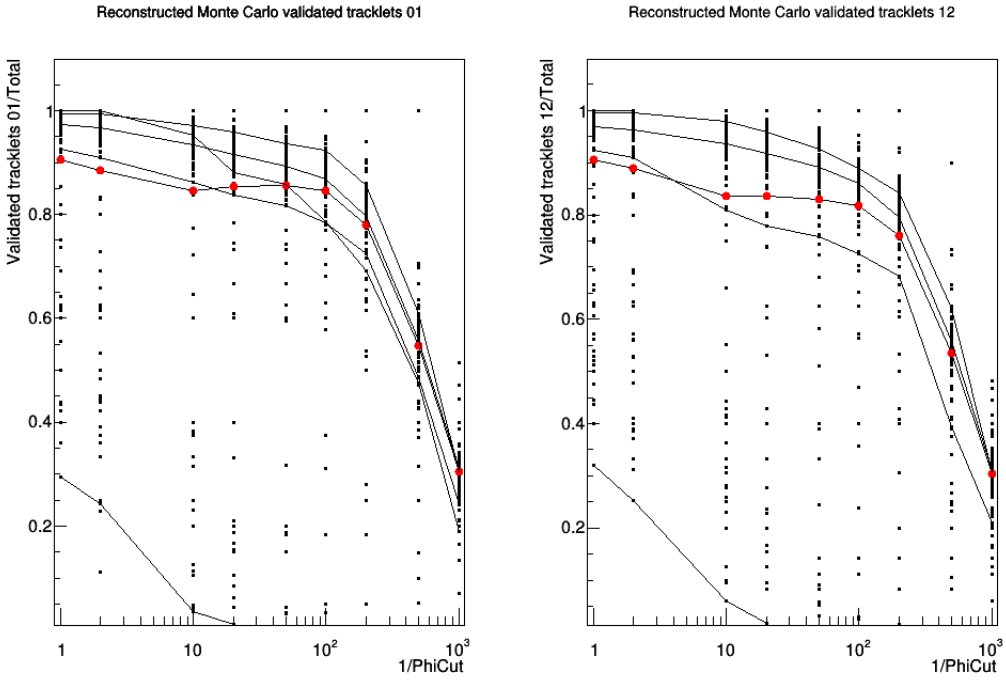

Figure 3: Proportion of reconstructed real tracklets according to the inverse of the cut in Φ

goal was to estimate what fraction of the real tracklets (that is the pair of clusters generated by the same real particle) we lose by varying the cut in Φ. To do this, the total number of real tracklets from one layer to the adjacent one was computed using a trivial method based on the simple check on the Monte Carlo labels available for each cluster. This method is used to connect all the clusters that initially belong to the same track, and leave at least a trace on the first three layers. It just creates all the possible pairs and does not apply any further selection on Φ angle. The number of reconstructed pairs with this method is the maximum number of real tracklets that can be reconstructed. The number of the real reconstructed tracklets is expected to be always smaller because the Φ angle is rather small and some very curved tracks will never be reconstructed. In particular it is also important to mention that the distribution in p_{T} obtained by integrating the spectrum of generated charged particles is not uniform. Indeed, for *pions*, the most abundant species measured by this detector, it has a peak at $0.3 - 0.5GeV$ then it decays exponentially with the growth of the transverse momentum. Hence, the functional dependency of the fraction of correct reconstructed tracklets as a function of Φ_{cut} is far from being linear. Moreover, the simulated data in use by the algorithm contains also electronic (random uniform distribution on *fired* pixel on the layer) and electromagnetic (photon conversions and other particles not really connected to the collision) noise. Therefore, the algorithm also reconstructs *fake* tracklets, that is, those that would have not been validated with a Monte Carlo check. The ratio of reconstructed tracklets and total tracklets is depicted for the tracklets from each couple of layers at different Φ_{cut} values in Figure 3.

4.3 Study of the tracklet selection efficiency as a function of the $\tan\lambda$ selection

The next step of the algorithm is the tracklet selection. We also need an idea of the evolution of the number of real and fake selected tracklets (lines) with the $\Delta tan(\lambda)$ cut. For that purpose, I wrote

a macro that computes again the number of total real lines using the trivial Monte Carlo method, and the number of real and fake lines after the selection. It is important to note that the number of total real lines is not equal to the sum of real and fake selected lines because some tracklets were not even reconstructed. We can notice that there is a plateau on both graphs, because the efficiency is also limited by the selection in Φ.

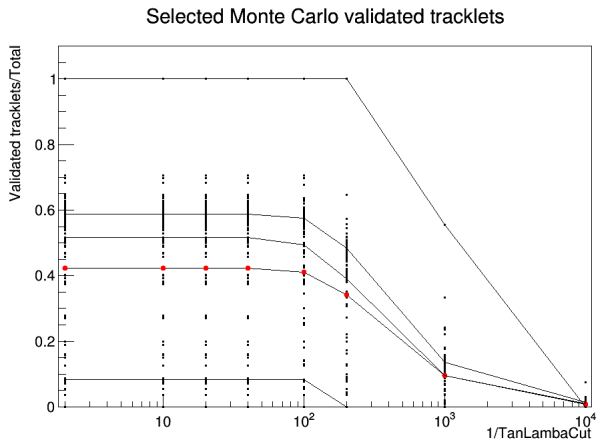

Figure 4: Proportion of selected real tracklets according to the inverse of the tan λ cut

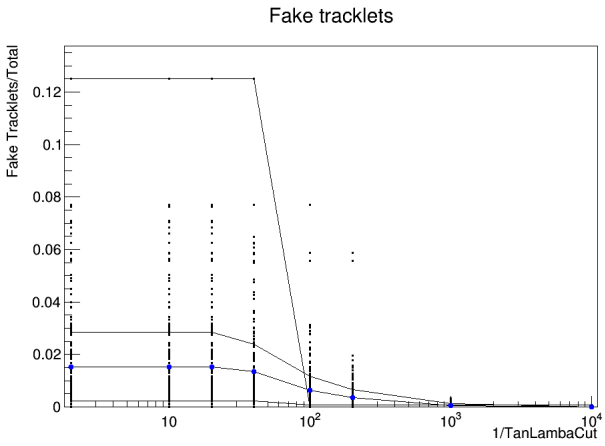

Figure 5: Proportion of fake selected tracklets according to the inverse of the tan λ cut

4.4 Troubleshooting: track and solving of a malfunctioning in the tracklet reconstruction

With the plots I produced with my macros, I noticed that for some Φ cut values, the efficiency of the tracklet reconstruction was higher than 1: we reconstructed more real tracklets than there really were. And, even worse, there were even more lines. This could not be caused by the noise, because we validated those tracklets using the Monte Carlo truth. This could only be connected to a problem in the algorithm. These kind of studies, not yet done in details, are also meant to spot more subtle bugs than the ones one can encounter during the software implementation.

Due to the convoluted nature of the problem, and the not very handful input data to manage, my supervisor advised me to generate a custom dataset as simple as possible. I extended the Input/Output interface of the ITS reconstruction code implementing a function to generate clusters on the seven layers of the ITS, initially only on the transverse plane with z coordinate equal to zero. The clusters on the different layers are all aligned, and the lines passing through them all cut each other in (0,0,0). The cluster are also evenly distributed in Φ. No noise, readout efficiency nor data smearing or multiple scattering due to the interaction of particles with the detector, were included. A *toy generator* for straight lines with a common, well known interaction vertex. With this very simple data, it was easy to see how the clusters were matched with one another. The expected efficiencies in the linear approximations must have been 100%, and the resolution of the vertex just needed to be precise as the resolution of a floating point number on architecture *x86_64*. However, I noticed that some clusters from layer 1 were matched several times with their corresponding cluster in layer 0, creating several copies of the same tracklet, whereas some clusters were never matched. After looking more into details into the code, I found that the problem was related to the index table, an interface written to find and iterate on the clusters container more effectively. Some clusters were iterated over several times. My supervisor and I fixed problem and I helped in the implementation on the upstream version of the code.

4.5 The search for lost vertices

Another problem was the efficiency of the algorithm. Among 100 vertices, it was only able to reconstruct 90, which is not enough. My supervisor had noticed this and asked me to investigate and to try to find all the vertices.

This was not an easy task. The primary idea was that the ROFrames in which we did not reconstruct a vertex contain few clusters, and that we should adapt the parameters of the algorithm to the number of clusters. But it soon got clear that this was not exactly the problem, and that the number of clusters in a ROFrame was not a good criteria to determine whether it contains a vertex or not. At some point, we found out that the noise was not labelled the way we thought it was, and that we were including a lot of noise in our tracklets and vertices. In the meanwhile the interface to test Monte Carlo labels has been improved and this ambiguous result fixed. We then were able to find all the vertices with the trivial Monte Carlo method.

I also switched to a new dataset including more noise, and therefore closer to the real use-case data. The problems that I encountered with this dataset were similar: I did not know where the real vertices were and if the vertices we found were real. The trivial Monte Carlo method is also not entirely reliable because it can match clusters from a particle running in circles and find a fake vertex with two contributing lines.

Hence, we elaborated a strategy to get the Id of the event to which a cluster belongs, at the same time the experts for Monte Carlo data simulation were notified of this problem. This information is very useful because it allows us to know in which ROFrames we expect to find a vertex. We found out that several ROFrames contain clusters from different events, but with different proportions. This explains why we don't find all the vertices but this problem was quite unexpected, since the frequency of the ROFrames is higher than the one of the events. To be able to reconstruct all the vertice,s it became necessary to apply some different Phi_{cut}, in particular the optimal one for PbPb minimum bias has been found to be $0.003[rad]$. The fact this cut was different from the previous one once more demonstrated the usefulness of this kind of studies I was performing.

4.6 Study the tracklet reconstruction and selection efficiency with the transverse momentum

Our search for lost vertices made us think about what could affect the efficiency of our algorithm. For instance, we had to study to what extent our assumption that the particles have a straight trajectory is valid. It is for example important to know how many tracklets we lose regarding of the p_{T}.

For these purposes, I wrote another program. The role of this program is to create the same kind of graphs we had when we studied the efficiency of the reconstruction or selection regarding the Φ or the $\tan\lambda$ selection. But since the transverse momentum is something continuous, and not discrete unlike the different cut values we chose, we have to create bins. In addition, it is not only important to know the fraction of tracklets that we lose, but also the real number of these tracklets. Therefore, plotting some histograms is also useful.

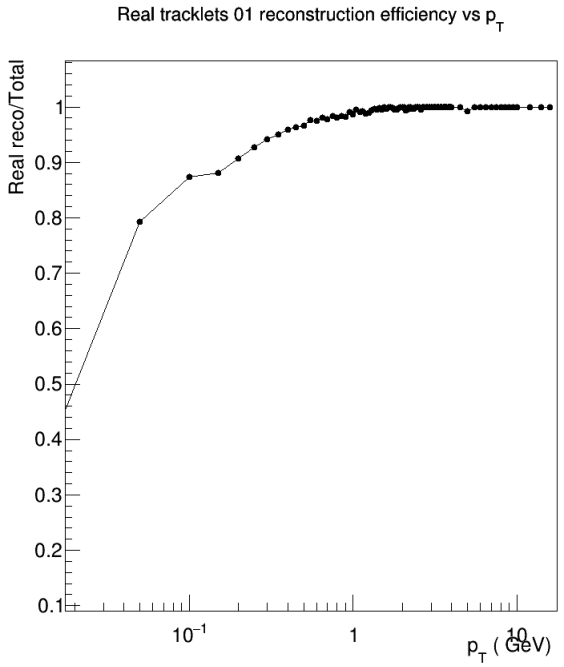

Figure 6: Proportion of real reconstructed tracklets on layers 01

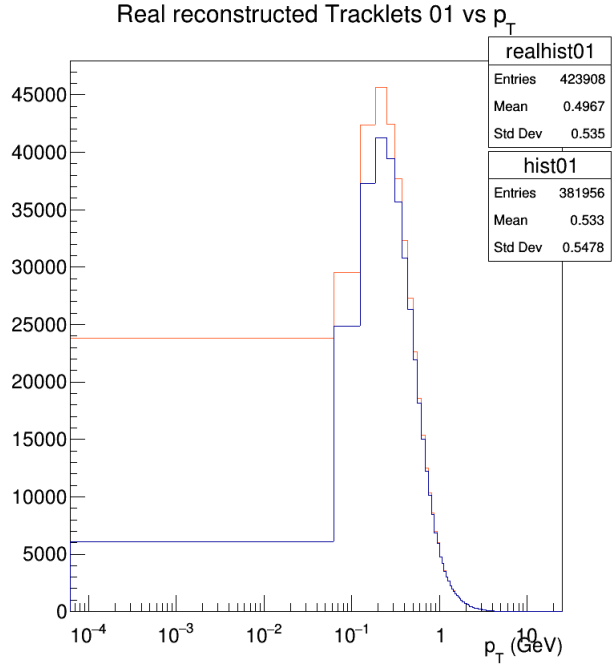

Figure 7: Number of real reconstructed tracklets on layers 01

As expected, the efficiency increases with the transverse momentum.

5 Conclusion and perspectives

As a conclusion, tuning the parameters of the algorithm is way more complicated than we expected. There were still bugs in the code that needed to be fixed. In addition, the code is still under development and truth to be said, sometimes the time spent in adapting to the new changes was more than estimated. The O^2 framework is still under active and fervent development, so it is normal. Also, the format for simulated data changed a lot during my staying. Nevertheless, an analysis to find parameters that fit the different types of events we can encounter still needs to be done. It will also be interesting to study the purity (i.e. the contribution of the different events) of the vertices and how the precision varies regarding to the purity and the number of contributors. It is very important to know what the algorithm is sensitive to and how it behaves with different datasets before trying to use a grid search or more automatised or possibly machine or deeplearning-based algorithm to find the best parameters for each scenario.

We still need to find a solution to find all the vertices when several events can be mixed in one ROFRame. Working with different types of collision may lead us to encounter other problems like this one that will need to be fixed. In the end, when all the features and descriptions of the collisions and the challenges will be clearly identified, it will be possible to try to adapt the algorithm.