A preliminary analysis of data from ALICE's new ITS and MFT detectors

Miles Kidson

Supervisors: Prof. Zinhle Buthelezi, Dr. SV Fortsch, & Prof. Tom Dietel
Assisted By: Dr. B Naik (Postdoctoral fellow)

UCT Honours 2022

Abstract

We investigate pilot-beam data from proton-proton collisions in Run 3 at ALICE, tracing its journey through two of the new detectors (the Inner Tracking System and Muon Forward Tracker) and the new Online-Offline analysis framework.

Contents

1 Introduction **3**

2 Background & Detector Theory **3**
2.1 Coordinates . . . 3
2.2 ALICE Run 3 . . . 4
2.3 The Inner Tracking System . . . 5
2.4 The Muon Spectrometer . . . 6
2.5 The Muon Forward Tracker . . . 7
2.6 The Online-Offline Analysis Framework . . . 9
2.7 Online Analysis . . . 10
2.8 Offline Analysis . . . 10

3 Analysing Using O2 **11**
3.1 AOD Structure . . . 11
3.2 Analysis Task Structure . . . 12
3.3 Table Features . . . 12
3.4 Histogram Registry . . . 13
3.5 Folder Structure . . . 13
3.6 Compiling O2Physics . . . 14
3.7 Running a Task with O2 . . . 15

4 Our Analysis **15**
4.1 The Data . . . 16
4.2 Initial MFT Analysis . . . 16
4.3 Comparing pass3 to pass4 . . . 17
4.4 Initial ITS Analysis . . . 17
4.5 Using pass4 for hasITS . . . 17

A Downloading data from the GRID **21**

1 Introduction

The ALICE (A Large Ion Collider Experiment) detector is a detector experiment at the Large Hadron Collider (LHC) at CERN. Its primary goal is the investigation of "strongly interacting matter at extreme energy densities, where a formation of a new phase of matter, the quark-gluon plasma, is expected" [6]. It achieves this goal by studying the products of head-on collisions of heavy ions such as lead, called Pb-Pb collisions for short. It also studies proton-lead (p-Pb) and proton-proton (p-p) collisions.

Run 3 is the latest period of data capture at the LHC, with an intended centre of mass energy per collision of $\sqrt{s} = 13.6\,\text{TeV}$ and increased luminosity of collisions—a factor 10 increase in integrated luminosity for Pb-Pb collisions. For Run 3, ALICE is moving from a triggered readout system to a combination of triggered and continuous readout. In order to achieve this, many detectors and their front-end electronics were upgraded, some new detectors were added, and the analysis framework was overhauled entirely.

Of the upgrades to ALICE, the following are the subject of this report. The Inner Tracking System (ITS) was upgraded with an entirely new pixel detector technology, hoping to greatly increase the resolution when determining the primary collision vertex. The Muon Forward Tracker (MFT) is one of the new detectors added. Its primary use is to assist the Muon Spectrometer (MCH) with vertexing and tracking in the forward region of ALICE and uses the same technology as the ITS. To deal with the increased volumes of data, a new analysis framework was introduced called Online-Offline (O2).

This report aims to introduce the detectors and how they output data, trace the path of that data through the new analysis framework until it arrives at our desk, and show the details of the analysis we performed on it. The data used in this report is from two proton-proton collision runs performed in October 2021, at a centre-of-mass energy of 900 GeV. This is not an energy we expect to use for physics data analysis but is good enough for this purpose.

2 Background & Detector Theory

2.1 Coordinates

The coordinate system used at ALICE needs to be discussed in order to fully explain the scope of this report. A modified cylindrical coordinate system, shown in figure 2.1, is used as most detectors in the experiment are cylindrically symmetric about the beamline of the LHC.

We place the z-axis along the beamline with its origin at the interaction point (IP). The IP is the point at which collisions happen, right in the center of the detector. The angle around the z-axis is called the azimuthal angle, denoted by φ. Sometimes in the literature φ ranges from 0 to 2π and sometimes it ranges from $-\pi$ to π. We will try stay consistent and use the latter in this report, but we may need use the other convention at times. The angle from the z-axis to the x-y plane is called the polar angle, denoted by θ, and runs from 0 to π. We are interested in the standard 3-momentum of particles that we track in the detector, which we call $\vec{p} = (p_x, p_y, p_z)$, but we also define the transverse momentum as

$$p_{\text{T}} = \sqrt{p_x^2 + p_y^2}. \tag{2.1}$$

We define the rapidity, often denoted as y, as

$$y = \frac{1}{2} \ln \left(\frac{E + p_z}{E - p_z} \right) \tag{2.2}$$

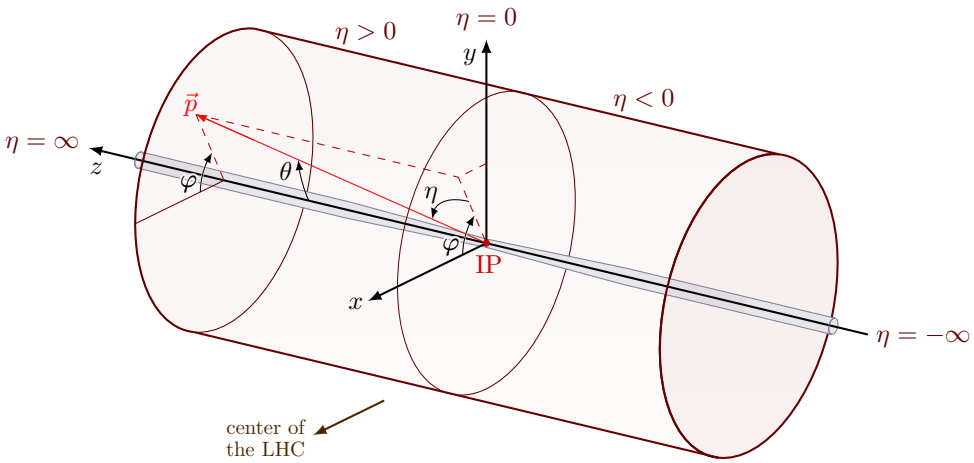


Figure 2.1: Modified cylindrical coordinate system used at the LHC [13].

where E is the total energy of the particle being considered and p_z is the momentum in the z direction [11]. This quantity is useful as differences in rapidity are Lorentz invariant for boosts along the z-axis. One issue, however, is that the energy of a particle is hard to measure, so we instead use pseudorapidity, denoted as η. Rapidity and pseudorapidity are equivalent for massless particles, and near equivalent for particles with total 3-momentum magnitude p much greater than their mass m. Pseudorapidity is much easier to measure as it is defined in [11] as

$$\eta = -\ln \tan \frac{\theta}{2}. \tag{2.3}$$

From figure 2.1 we see that for z positive, η is also positive, and similarly for z negative. Confusingly, we define the "forward region" of the ALICE detector as the region for which z, and thus η, are negative. The forward region is where our interest lies.

The four coordinates that we use most often are z, φ, p_{T}, and η.

2.2 ALICE Run 3

In 2018, the LHC shut down for what was called Long Shutdown 2 (LS2). During this time, the ALICE experiment was being prepared for Run 3, where it will be taking data at higher energies and much higher luminosities than before, from 2022 until 2025 [3]. Figure 2.2 shows the detector configuration for Run 3. The intent of these upgrades was in large part to prepare ALICE for a higher luminosity of collisions in both Pb-Pb and p-p cases.

Part of the upgrades for Run 3, the details of which can be found in [3], were a whole new Inner Tracking System and a brand new detector called the Muon Forward Tracker. These detectors are both silicon-based and their primary purpose is tracking particles and determining the collision vertex, which is the best estimation of where the collision that resulted in these particles happened.

The readout electronics for many detectors were upgraded to allow for continuous readout where necessary. The MCH also had its readout and front-end electronics upgraded but will still work on a triggered readout system.

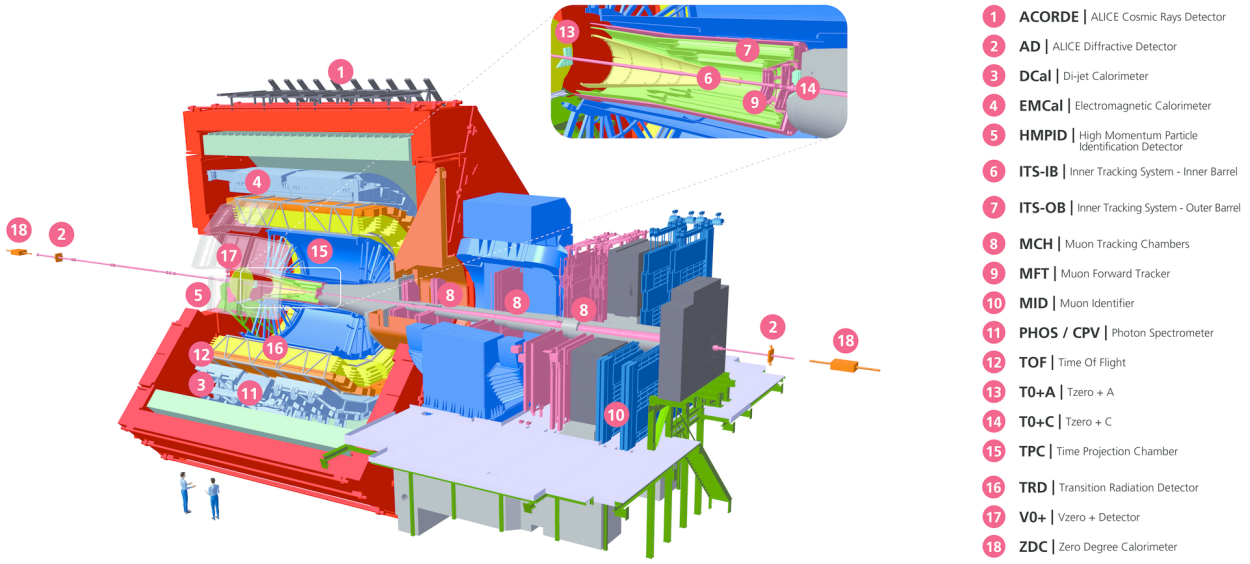


Figure 2.2: Schematic view of the ALICE detector setup for Run 3 of the LHC [1]. Note here that the MCH is shown separate from the MID, which act as the triggering mechanism for the MCH. For the purposes of this report, the MID will be considered part of the MCH. The ITS (6, 7), MCH (8), and MFT (9) are the focus of this report.

2.3 The Inner Tracking System

The Inner Tracking System (ITS) sits in the main barrel of ALICE, as seen in figure 2.2, and covers the range $|\eta| < 1.22$ [2]. For Run 3 it has been upgraded significantly by replacing the old detector with a new layout and new pixel detector technology, leading to an improvement in track position resolution at the primary vertex of a factor of 3 or greater [2]. The ITS's main purpose is to track the particles resulting from the collisions and determine the position of the primary vertex of collisions. It also serves to "reconstruct secondary vertices, track and identify particles with low momentum, and improve the momentum and angle resolution for particles reconstructed by the Time Projection Chamber (TPC)" [9].

The new ITS consists of 7 layers of pixel detectors; 3 in the "Inner Barrel" and 4 in the "Outer Barrel". The innermost layer sits at a radius of only 22.4 mm from the IP thanks to a reduction in beam pipe radius for Run 3 and the outermost layer sits at a radius of 391.8 mm from the IP. Figure 2.3 shows the layout more clearly.

The pixel detectors used are 0.18 µm CMOS chips from TowerJazz. When a charged particle passes through the silicon in the active volume, it liberates the charge carriers in the material, which then collect on electrodes connected to the silicon, telling the detector that a particle has been detected. The fine segmentation of the detectors also allows the detector to determine the point at which the particle hit the detector, up to a resolution of 4 µm in both the $r\varphi$ and z directions [2]. The amount of charge deposited on the detector is dependent on the particle species and momentum (Bethe-Bloch).

Two main methods of readout were considered for the ITS in the new continuous readout scheme. Firstly, a rolling shutter which continually loops through the rows of pixels and reads out the charge deposited on that pixel in the time since the last readout was considered. The time between readings, known as the integration time, for the first method is around 30 µs. The rolling shutter scheme lends itself to needing a small number of transistors within each pixel. The second

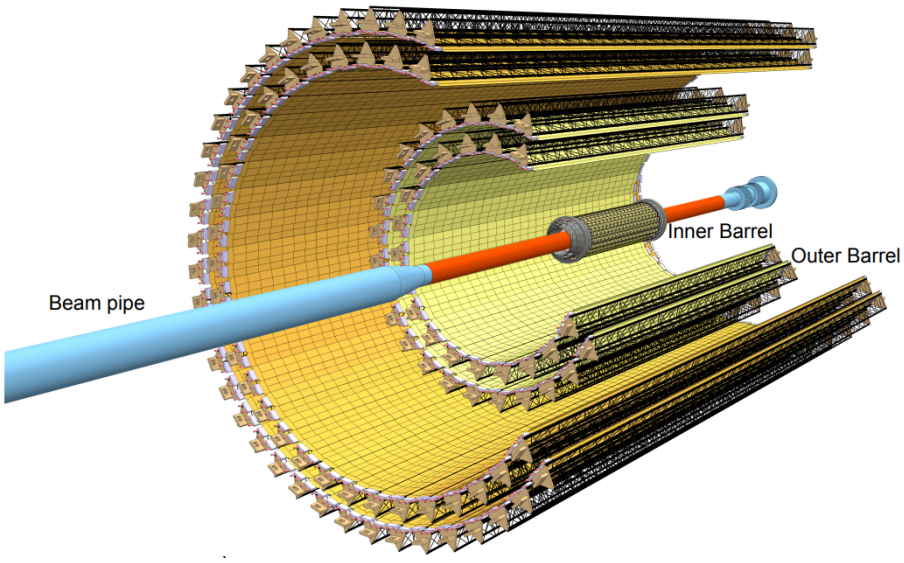


Figure 2.3: Schematic view of the Inner Tracking System [2]. Note the thinner beam pipe and extremely close Inner Barrel.

scheme is known as ALPIDE, where each pixel has a comparator that signals when the pixel has an analogue signal greater than the comparator's threshold. The signalled pixels then get read out asynchronously, according to their priority in the chain. This scheme has an effective integration time of around 4 µs but has a larger material budget.

NOTE THAT I AM TRYING TO FIND OUT WHICH ARCHITECTURE WAS DECIDED ON FOR THE ITS. IT'S NOT IN THE TDR AS FAR AS I CAN SEE SO IF I CAN'T FIND IT I WILL REWORK THIS SECTION A BIT

2.4 The Muon Spectrometer

The MCH sits in the forward region of ALICE, as seen in figure 2.2, and covers $-4 < \eta < -2.5$. It is designed to study heavy quark resonances through their single- and di-muon decay channels. As is shown in figure 2.4, it is composed of a hadronic absorber, 5 tracking chambers, a dipole magnet, another absorber, and finally the 2 trigger chambers. The MFT is often also considered part of the MCH but is not shown in figure 2.4. The section is adapted from the MCH Technical Design Report [5].

Muons don't tend to interact with matter much, especially when compared to hadrons and electrons. This makes studying muons easier, to some degree, since most other particles can be filtered out by making them pass through a large chunk of matter. As is shown in figure 2.4, that is exactly what is done in the MCH. In front of any detector material (ignoring the MFT for now) sits the hadronic absorber, made primarily of carbon and concrete. This is intended to filter out all non-muon particles (mainly hadrons and photons) while not reducing the muon energy much so that they can still be studied.

After the absorber are 5 sets of 2 cathode pad tracking detectors, situated around a large warm dipole magnet. Particles that make it through the absorber get picked up by the first two sets of detectors, then pass through the magnetic field and are deflected according to their charge, mass, and momentum. The third set tracks the particles during deflection and then the last two detect

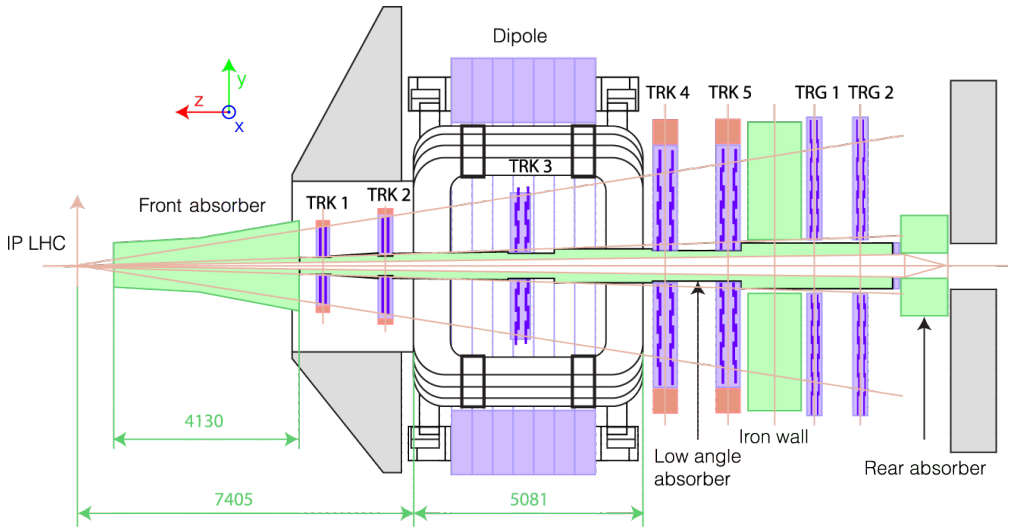


Figure 2.4: Diagram of the layout of the Muon Spectrometer [12]. Muons pass through the absorber, are deflected by the dipole magnet, and hit the trigger chambers at the back. Importantly, all detector material sits behind the hadronic absorber, meaning the amount of data that can be used for tracking and vertexing is much lower than, say, the ITS.

them after deflection. This set-up is particularly useful for studying di-muon events as the muons produced would be a muon-antimuon pair, which would deflect in opposite directions in the dipole magnet. This would leave a characteristic track signature that can be studied.

After the last tracking detector, particles pass through another absorber, which serves to further filter out muons from background, as well as filter out background muons. The muons produced in heavy quark resonances have considerably higher p_{T} than those produced by background processes, so the job of the trigger system is to only trigger on muons with high enough p_{T} to be interesting (this is defined per process) and the second absorber helps reduce the number of background muons incident on the trigger system. The trigger system is made of 2 stations of 2 Resistive Plate Chamber (RPC) detectors. Comparing the measurement of the same particle in the two stations, the p_{T} can be determined. The decision to keep or reject an event takes about 300 ns.

In Run 1 and Run 2, the MCH performed all its own tracking and vertexing on the particles it studied. Particularly for vertexing, where the collision position is estimated, this was not optimal as most of the particles produced in the interactions didn't make it through to sensitive material. With the increased energy and luminosity of Run 3 a better system was needed to perform these tasks, so the MFT was added in front of the first absorber to take over the job.

2.5 The Muon Forward Tracker

The MFT is a brand new detector added to ALICE for Run 3 to assist the MCH with tracking and vertexing. It covers the range $-3.6 < \eta < -2.45$ and was designed in conjunction with the ITS, using precisely the same CMOS pixel detectors. Due to the MFT being placed in front of the absorber, it detects a lot more particles than make it through to the MCH, allowing it to be much better at finding the primary vertex of collisions. Figure 2.5 shows the design of the MFT. The rest of this section is adapted from the MFT Technical Design Report [14].

The MFT is made of two identical half-cones sandwiching the beam pipe from above and below, each with 5 half-disks positioned at different distances from the IP along the z-axis. Each half-disk

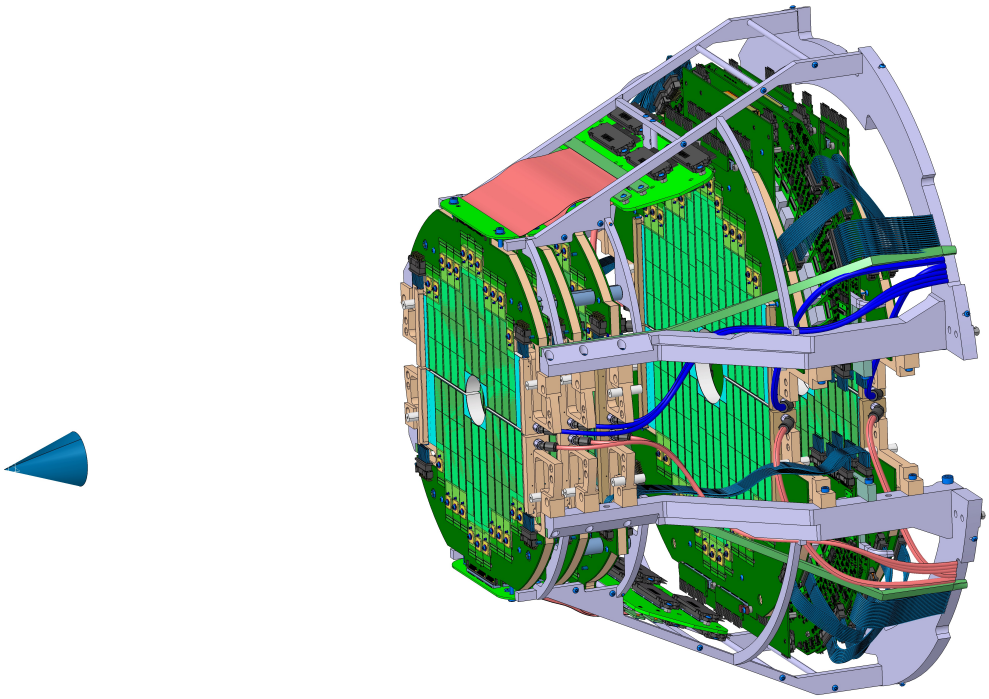


Figure 2.5: Schematic view of the Muon Forward Tracker [4]. The small cone on the left shows the IP. Note that the 5 disks each have a front and back plane of pixel detectors, totalling 10 z-positions for the MFT to "see" particles at.

has a front and back detection plane totalling 10 detection planes. As the disks get further from the IP their radius increases in order to cover the same η range, aside from the second disk, which is identical to the first. The disks sit at z-positions -46.0, -49.3, -53.1, -68.7, and -76.8 cm respectively and each disk is 1.4 cm thick, leading to detector planes at ± 0.7 cm from each of those positions.

The half-disks are made from ladders of 2 to 5 rectangular pixel detector elements. Figure 2.6 shows an example of the layout of the front and back planes of detector elements in both x-y and η-φ.

The MFT performs standalone track reconstruction in two steps. Track finding filters through clusters (hits in a certain layer) to group into track candidates and then track fitting to fit tracks to those clusters and determine kinematics and covariance matrices. A Kalman filter [10] is used for the track fitting but while it could also be used for the track finding, a more advanced algorithm is used to increase efficiency and reduce computation time.

There are two algorithms used in conjunction to group clusters into track candidates. The first, called the Linear Track Finder (LTF) Algorithm, assumes that in the η region that the MFT occupies the solenoid magnet surrounding the central barrel has a reduced bending effect on tracks, leading to effectively straight tracks. A "seeding line" is determined from clusters in the first and last disk, often with the help of the position of the primary vertex, and the algorithm simply minimises the distance of clusters to that line. There is a single parameter for this algorithm which is the radius around a cluster to allow a seeding line to be considered. While the LTF method is very computationally inexpensive, its straight line assumption breaks down at low momenta, so the Cellular Automaton (CA) algorithm is implemented to pick up the pieces.

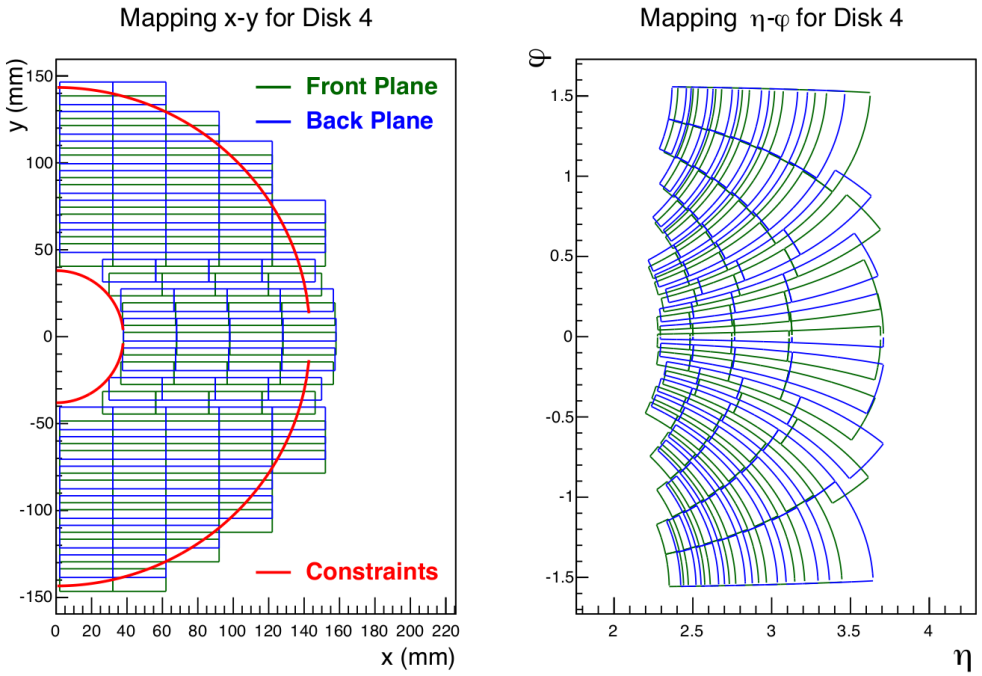


Figure 2.6: The layout of pixel detector elements on the front and back plane of one of the half-disks of disk 4 in the MFT. Note that the front and back plane have layouts offset to each other by half the width of the pixel elements. This figure is taken from [14, fig. 6.1] but we would like to point out that the axes are labelled incorrectly in the left case; x and y should be swapped as the ladders run vertically.

The CA algorithm works by pairing clusters in neighbouring disks into "cells" and then iterating through cells and determining, based on a number of parameters such as maximum deviation angle and maximum θ angle of a cell according to the collision vertex, whether neighbouring cells are "compatible" and grouping those cells together as candidates for a track. This algorithm depends on having a determined position of the collision vertex to both create cells and to match cells together as in both cases they need to point towards the vertex within some pre-defined limit. This vertex position can be supplied by external means, such as the ITS, or it can be estimated using cells between the first two disks. The algorithm also has a default minimum track length in terms of number of disks involved in the process, which is set at 4 out of 5 disks.

Due to the number of cluster combinations that the CA algorithm needs to consider compared to LTF, it ends up being considerably more computationally expensive to run. Since both methods have their advantages, particularly with CA picking up the slack in the low momentum (and specifically low p_{T}) regions, both methods are implemented. LTF goes first, finding as many high p_{T} tracks as it can, and then CA runs on the clusters that the LTF did not find tracks for. This reduces the overall computation time and take advantage of both algorithms as much as possible.

2.6 The Online-Offline Analysis Framework

With the increased interaction rate intended for Run 3, a new system for real-time processing, as well as offline analysis, needed to be constructed [3]. The Online-Offline (O2) framework was developed for this purpose. This section is mostly adapted from the O2 Upgrade TDR [7] as well as the O2 documentation at `https://aliceo2group.github.io/analysis-framework/`.

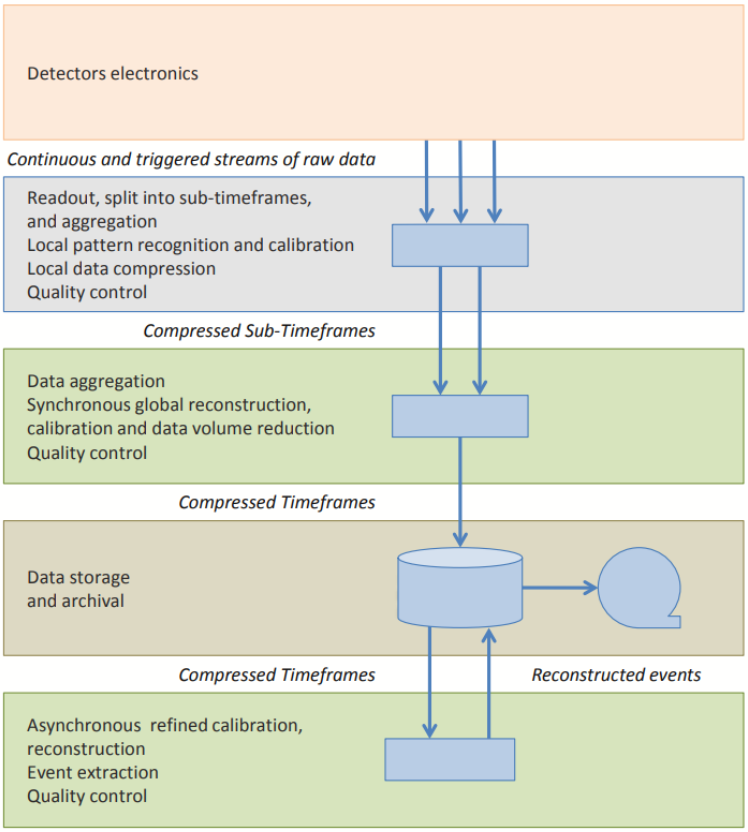


Figure 2.7: Functional flow of the O2 framework [7]. Detectors output their signal continuously in the new configuration and this signal gets split into chunks called timeframes. These timeframes get processed a number of times, reducing the volume of data each time by only extracting the quantities that will be used for analysis. Many choices have to be made at each step to ensure that useful data makes it out the other side and so the details of this process are always in flux.

2.7 Online Analysis

The "Online" portion of the framework is the part that does all the synchronous processing, i.e. as the raw data is continuously read out from the detectors. The data is first stored in "sub-time frames" (STFs), which get processed to generate clusters in order to reduce the volume of data, and then gets built into "time frames" (TFs) of 10 ms chunks of data. From there the data gets its first pass of detector reconstruction, e.g. track finding, and then gets compressed into "compressed time frames" (CTFs). The CTFs are stored on disk and that is where the Online portion ends. Note that at each step, QC and calibration data is extracted and stored for later use.

2.8 Offline Analysis

With the CTFs stored to disk, they can then by processed asynchronously into Event Summary Data (ESD) and then Analysis Object Data (AOD) files. ESDs are not used for analysis and are stored for a while before being deleted. AODs (real and simulated) and CTFs are the only persistent data type.

The process of turning CTFs into AODs is called "reconstruction" and is often performed multiple times on the same data. Iterations of reconstruction are called reconstruction passes and

these passes will usually be done many times if the previous passes missed something in their reconstruction, or perhaps if a certain table was not populated in the reconstruction but someone needs that table for their analysis. As a result of this, two different passes on the same data can end up with very different-looking results.

We distinguish between data taken in Run 3 and data taken in Runs 1 and 2 by calling Run 3 AOD's "AO2D" files, and Run 1 and 2 just "AOD" files. The data is stored on the `alimonitor` system, requiring a certificate to access, which is obtained by joining the ALICE collaboration. Access to all data and most analysis tools used in this report is restricted behind this wall. AODs can then be analysed using an "Analysis Task", which is written in C++ and ROOT.

The focus of the upgraded analysis framework was to reduce disk space usage when processing and analysing, as well as making sure all analysis takes advantage of all processing power available to it at all times.

3 Analysing Using O2

One of the main goals of O2 is to take away as much responsibility from the user as possible when it comes to managing memory, writing efficient code, and disk space usage. To this end, writing an analysis task in O2 is structured quite strictly. This section will outline the information needed when trying to analyse reconstructed data to produce, in our case, histograms of kinematic variables.

It must be noted that learning how to do this, and how to deal with the idiosyncrasies of the O2 software, is what took up the majority of the time spent on this project. O2 is written to do one thing very, very well, but unfortunately that comes with the side-effect of it being extremely picky about the conditions in which the software will actually work. Lastly, a distinction needs to be made between O2 and O2Physics. O2 generally refers to the entire analysis framework, including both the online and offline parts. This encompasses O2Physics, which is used only for offline analysis. Despite this, in terms of the structure of the software, O2 and O2Physics are separate things but O2Physics sometimes depends on O2. For this reason it can become ambiguous when discussing O2 and O2Physics but we will try our best to keep them distinct.

3.1 AOD Structure

The data that we use in our analysis comes in the form of Analysis Object Data. These get produced asynchronously and come in the form of ROOT files containing "Dataframes". These are organised in a tree structure where each tree contains a number of tables, for example a table containing collisions and a table containing tracks. The tables have columns which hold variables corresponding to each entry, or row, in the table. There are 4 types of column:

- Static columns are saved to disk during the reconstruction process and are available at any time. The z-position of a collision vertex, for example, is a static variable.
- Dynamic columns are defined as functions with inputs from static variables (or variables input by the user). They will produce a result on demand in order to save memory and disk space. Momentum in the x-direction is a dynamic variable.
- Index columns point from one table to another, such as from a track to its associated collision. They can also refer to their own table, such as for an MC particle to refer to its mother or daughter particle after decay.

- Expression columns are similar to dynamic columns in that they get calculated on demand, but they get evaluated on all entries in a table upon requesting it and produce a column that can then be accessed as if it were a static column. Importantly, dynamic columns cannot be used as inputs to expression columns.

3.2 Analysis Task Structure

Analysis tasks are written in C++ and need to be structured in a specific way so that O2Physics to use them properly. Each task is written as a `struct` object which is then called at the end of the task. Below is an outline of what is needed for a task.

```
#include "Framework/runDataProcessing.h"
#include "Framework/AnalysisTask.h"

using namespace o2;

struct MyTask {
  // Define things here, such as histogram registries, filters for data, or new tables

  void init(o2::framework::InitContext&) {
    // Here we initialise histograms and other things used in the analysis
  };
  // The arguments of the process function are where we subscribe to specific tables in the AOD
  ↪ that we're analysing.
  void process(aod::Collision const& collision, aod::Tracks const& tracks) {
    // Here we can do any processing that we need, calculating things etc, and then fill the
  ↪ histograms we defined earlier
  };
};

// This is what O2 looks at to run the task
WorkflowSpec defineDataProcessing(ConfigContext const& cfgc)
{
  return WorkflowSpec{
    adaptAnalysisTask<MyTask>(cfgc),
  };
}
```

3.3 Table Features

All variables associated with a track, for example, could be included in a single table. However, since we often only need a few of the variables, the tables are split up into sections that contain variables often used together. If needed, these tables can be joined together when doing analysis using `o2::soa::Join<Table1, Table2>`. Importantly, only tables which correspond row-to-row and have the same number of rows can be joined in this way.

We might also want to only access entries in a table that pass some filter, say having $|\eta| < 0.8$. We can do this by putting `Filter f = nabs(aod::track::eta) < 0.8;` somewhere before the `process` function and then instead of subscribing to `aod::Tracks` as we have done, we can put `soa::Filtered<aod::Tracks> const& filteredTracks` in the `process` function call. Multiple filters can be defined and only compatible tables will be filtered. Once a table has been filtered, the unfiltered version cannot be accessed.

Similar to filters are partitions, which we define like `Partition<aod::Tracks> leftEta = aod::track::eta < 0;`. This will create a separate table that we can subscribe to which will only

contain tracks with the specified η value. This is different to a filter as the original table can also be accessed at any time.

Some tables have pre-defined iterators which can be used to more efficiently loop through all entries in the table. `Collisions`, for example, has an iterator called `Collision` that we subscribed to in the code example above. When a table has an index column relating to `Collisions`, as `Tracks` is, O2 will group the entries in `Tracks` by associated collision.

3.4 Histogram Registry

Histograms are the eventual desired output of an analysis task and O2 has a class designed specifically to make creating and filling histograms easier. Before the `init` function we can define our histogram registry and then in the `init` function we can define the histograms we want and add them to the registry. We can then fill the histograms in the `process` function.

```
HistogramRegistry myRegistry{ // This name and the name in the next line need to be the same
  "myRegistry",
  {},   // Histograms could be defined here but we will do it in init()
  OutputObjHandlingPolicy::AnalysisObject,  // Tells the task which format to output in
  true,   // Sorts the output histograms alphabetically
  false   // Won't create a subdirectory for this registry. Set to true if more than one registry is being used
};

void init(o2::framework::InitContext&) {
  AxisSpec etaAxis = {nBins, binMin, binMax, title} // title is what gets printed on the axis

  myRegistry.add("myHist", "myHist", kTH1F, {etaAxis})  // Initialises the histogram. First argument is the name of the histogram, to use internally, and second gets printed as the title
};

void process(aod::Tracks const& tracks) {
  for (auto& track : tracks) {
    registry.fill(HIST("myHist"), track.eta());   // Fills the histogram with eta from all tracks
  };
}
```

3.5 Folder Structure

With the task written, it then needs to be compiled and added to O2Physics so that it can be run. O2Physics has a number of analysis tasks written by people at ALICE which get compiled automatically. These are sorted into physics working groups such as Heavy Flavour (PWGHF) and Jets (PWGJE). If we want to add our own task to O2Physics, we need to create our own folder with the same structure as the working groups. Below shows the structure of the file system.

```
alice
├── alidist
├── O2
├── O2Physics
│   ├── Functional Things
│   ├── ...
│   ├── myTasks
│   │   ├── CMakeLists.txt
│   │   ├── Tasks
│   │   ├── myTask1.cxx
│   │   │   ├── CMakeLists.txt
│   │   │   ├── myTask2.cxx
│   │   │   └── ...
│   ├── Other Working Groups
│   ├── ...
│   └── CMakeLists.txt
└── sw
```

Here `alidist` is the git repository that handles the versioning of O2 and O2Physics. O2 is what handles the backend of the analysis framework, compiling the tasks written in O2Physics. The `CMakeLists.txt` files are needed at every level of the O2Physics structure to tell O2 what to compile and which commands to use to refer to things.

See below an example of what the `CMakeLists.txt` file in the `myTasks` folder would look like.

```
add_subdirectory(Tasks)                 # Ensures O2 can see the Tasks folder
o2physics_add_dpl_workflow(my-task1     # The command assigned to the task. Note only lowercase
↪  letters, hyphens, and numbers are allowed
                  SOURCES myTask1.cxx   # The source file for the task
                  PUBLIC_LINK_LIBRARIES O2::Framework
                  COMPONENT_NAME Analysis)
```

3.6 Compiling O2Physics

O2 and O2Physics are built on your system using `aliBuild` [8]. They prefers to be built on UNIX systems and requires at least 8 GB of RAM, preferably more. We used both O2 and O2Physics as we wanted to create and run analysis tasks.

Once they are built (those four words are doing some *very* heavy lifting) we can enter the O2Physics environment with `alienv enter O2Physics` and this will place us in a new terminal shell. The magic of the software is that it compiles all the analysis tasks in O2Physics, as well as tools for simulation and the like, such that everything can be done by running commands in that shell. Before running our own tasks, however, we need to tell O2 to build our tasks into O2Physics. To do this we use `ninja`.

If we have our tasks written and files structured as shown in section 3.2, we can enter an O2Physics environment and load ninja alongside using `alienv enter O2Physics ninja/latest`. In the shell we can then navigate to the build of O2Physics, which should be in `alice/sw/BUILD/O2Physics-latest/O2Physics` and run `ninja install myTasks/all` which will rebuild only those parts of O2.

Ninja is the quick and dirty way of doing things so every now and then it's a good idea to rebuild O2 and O2Physics entirely, pulling the latest release. We can do this while also making sure our

own tasks don't get overwritten using the following steps. We first need to make sure that git knows our files are there with `git add path/to/files` and then committing with `git commit -m "commit message"`. With that done, we can systematically update `alidist`, `O2`, and `O2Physics` by navigating to each and running `git pull --rebase`. Finally we can rebuild by navigating to `alice` and running `aliBuild build O2Physics --defaults o2 --debug`. This will take a few hours to complete (if it's even successful) and then will be able to be used again.

3.7 Running a Task with O2

Once we have our tasks created and built in O2Physics, we can then run them. For our purposes, the only commands we need to know are how to run a task and the options that come along with that. All analysis tasks in O2Physics get assigned a unique command that can be used to run that task. They all begin with `o2-analysis...` followed by the name assigned to it in the relevant `CMakeLists.txt` file as shown in section 3.5. In the case of that task, we would run it with `o2-analysis-my-task1`.

Most analysis tasks are run on `AOD.root` or `AO2D.root` files so in order to tell the task which file to use, we use the flag `--aod-file AO2D.root`. We could also supply a list of files in a text file and use `--aod-file @AO2D_list.txt` where `AO2D_list.txt` contains the path to the files we want to run on, one on each line.

Often we want to run multiple tasks in succession on the same data, feeding the output of one into another. To do this, we simply use the pipe symbol `|` between the tasks: `o2-analysis-trackselection | o2-analysis-my-task1 --aod-file AO2D.root`. Here the ordering of the tasks doesn't matter as the input and output format of a task is known before it runs, so O2 does some quick thinking to arrange the workflow such that the tasks get fed the correct format of data.

The `o2-analysis-trackselection` task above is an example of a helper task. These are specifically designed to use the available data to produce tables that are needed in analysis tasks. This is done once again in the interest of saving disk space as calculating the values needed is much less resource-heavy than storing them for all time. A list of helper tasks and the tables they produce can be found at `https://aliceo2group.github.io/analysis-framework/docs/datamodel/helperTaskTables.html`. Note that when running a workflow with multiple tasks, if a configuration file is needed, or some other option, the flag needs to be provided before the pipe symbol for each task that gets called.

For the most part, the output of an analysis task is either a `AnalysisResults.root` file or a `QAResults.root` file, with the former being the most common. This output type is chosen when defining the histogram registry. If all goes well (which is a rare occurrence) the ROOT file produced can be opened with TBrowser some lovely histograms should pop up. These can be saved as is or output using ROOT macros.

4 Our Analysis

The intention of this report is to investigate preliminary data coming from the ITS and MFT in Run 3. The order of the discussion in this section will follow the order that we tackled things. This is done to show both the progression of our knowledge as well as to try to clarify some of the explanations in previous sections as the only way to truly understand some things is through example.

4.1 The Data

The data analysed in this report was taken in October 2021, where protons were collided at a centre of mass energy of 900 GeV. This is not an energy that we expect to use for physics research but it allows us to look at how the detectors are performing with more lightweight data, simply because there will be fewer particles created in the collisions and thus less data to work with. We are using runs 505548 and 505645. In this case, a run specifies a period of data taking for which all global settings remain the same.

4.2 Initial MFT Analysis

In the AOD data model there is a table called `MFTTracks` which contains the tracks detected in the MFT.

When we began this analysis, only two reconstruction passes had been run on the data and while the MFT was switched on for the runs,

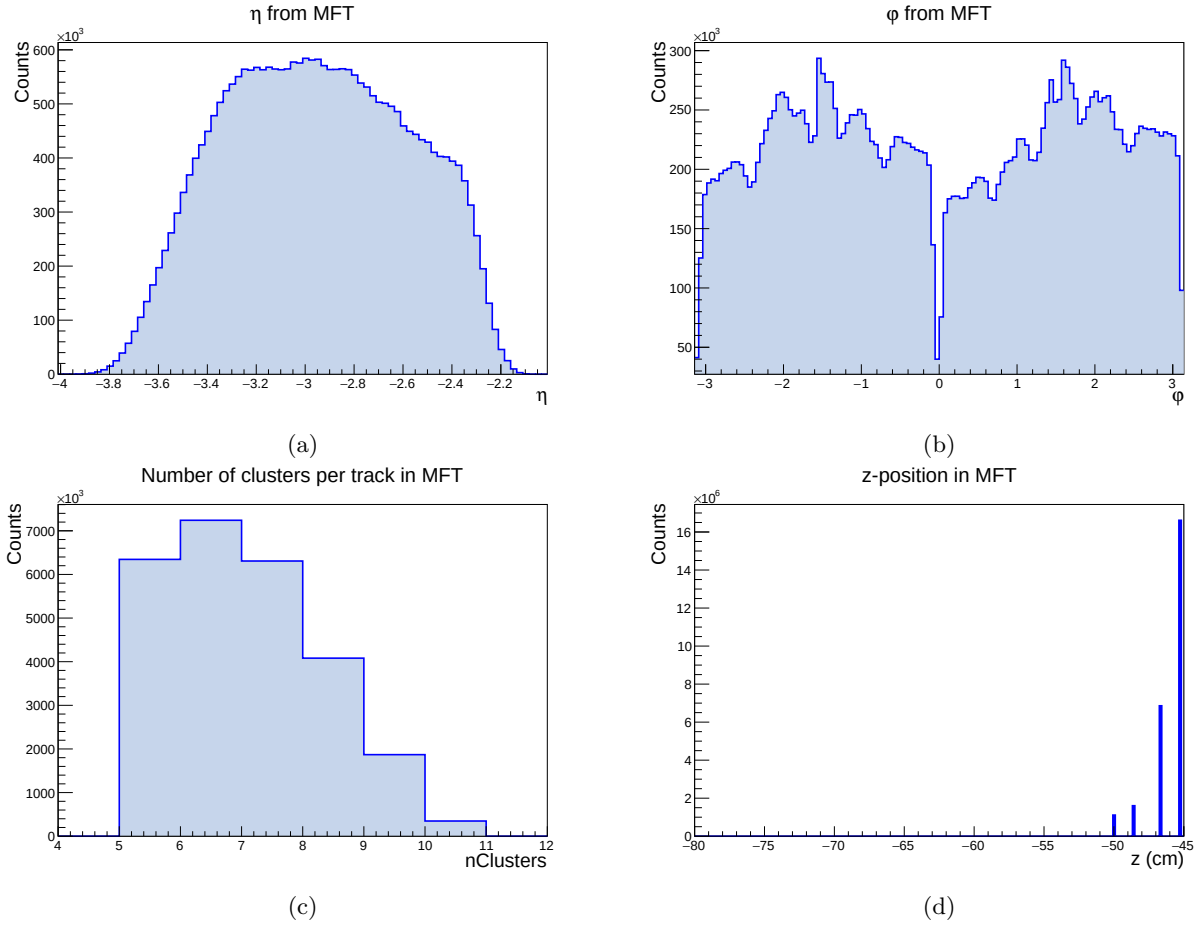


Figure 4.1

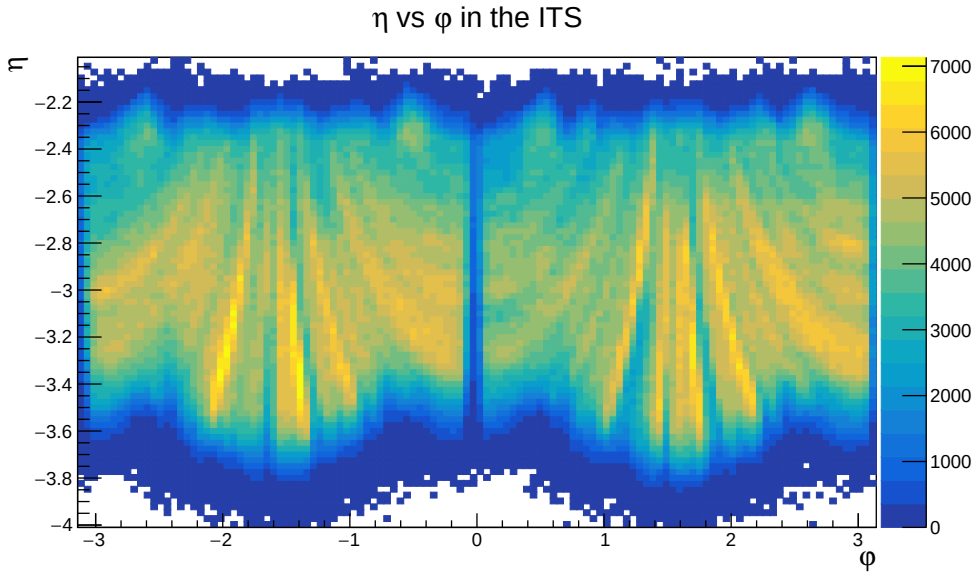
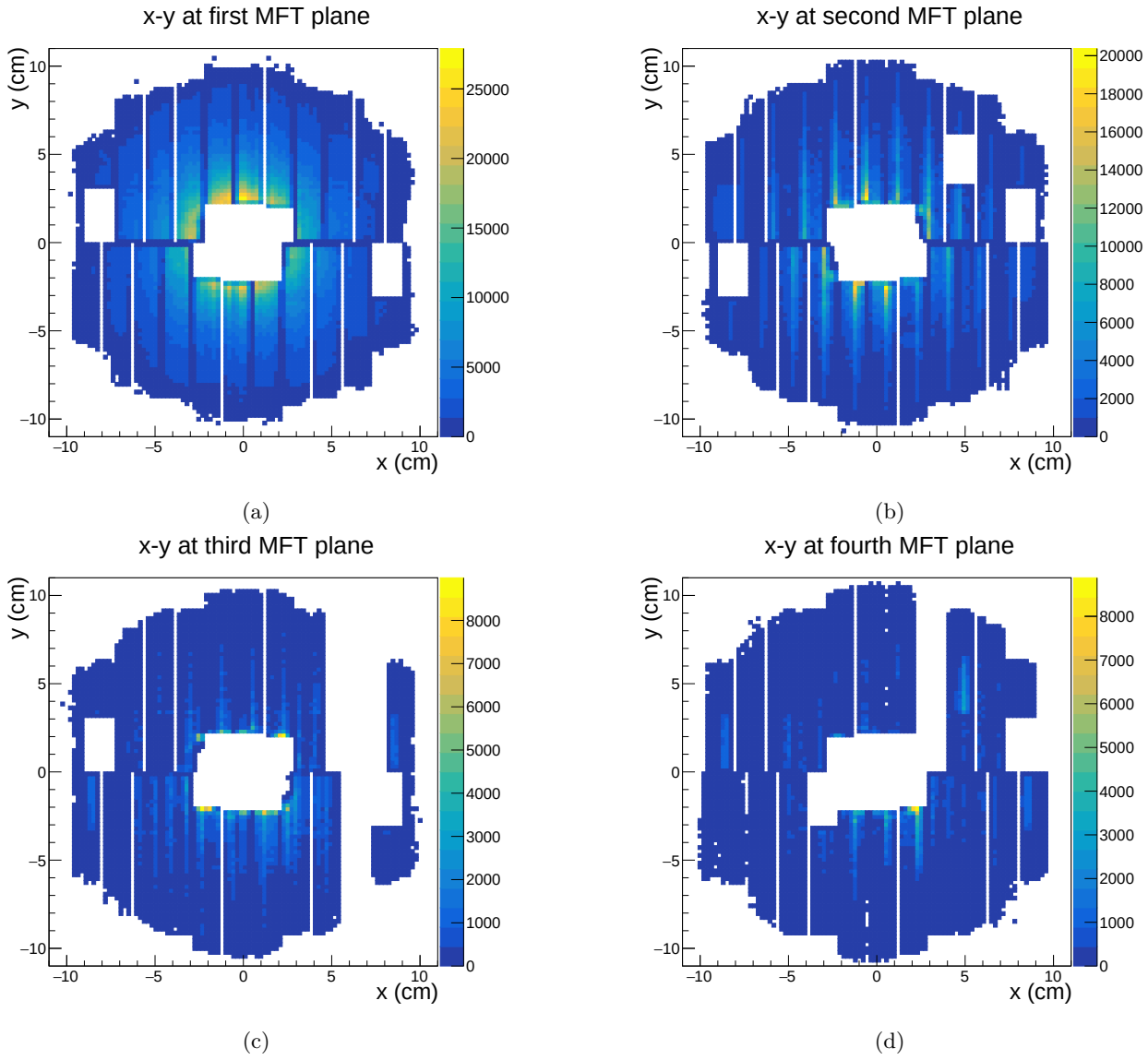


Figure 4.2: caption

4.3 Comparing pass3 to pass4

4.4 Initial ITS Analysis

4.5 Using pass4 for hasITS



(a) (b) (c) (d)

Figure 4.3

References

[1] *3D ALICE Schematic RUN3 - with Description — ALICE Figure*. URL: `https://alice-figure.web.cern.ch/node/11220` (visited on 07/15/2022).

[2] B Abelev et al. *Technical Design Report for the Upgrade of the ALICE Inner Tracking System*. Tech. rep. Nov. 2013. DOI: `10.1088/0954-3899/41/8/087002`. URL: `https://cds.cern.ch/record/1625842`.

[3] B Abelev et al. *Upgrade of the ALICE Experiment: Letter of Intent*. Tech. rep. Geneva: CERN, Aug. 2012. DOI: `10.1088/0954-3899/41/8/087001`. URL: `https://cds.cern.ch/record/1475243`.

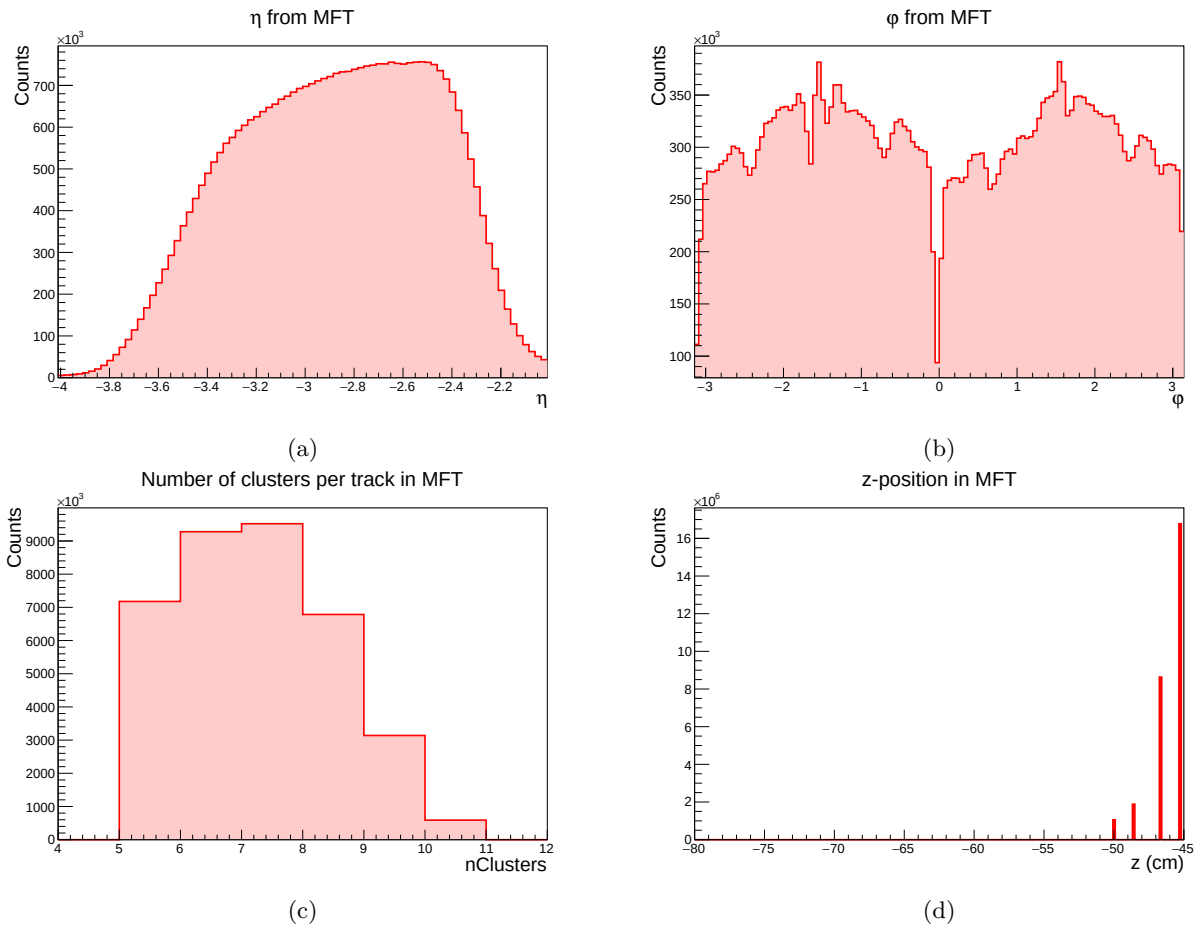


Figure 4.4

[4] MFT Collaboration ALICE. *ALICE Muon Forward Tracker (MFT)*. CERN Bulletin. Number: OPEN-PHO-EXP-2020-004. Dec. 25, 2020. URL: https://cds.cern.ch/record/2748310 (visited on 07/17/2022).

[5] *ALICE dimuon forward spectrometer: Technical Design Report*. Technical design report. ALICE. Geneva: CERN, 1999. URL: https://cds.cern.ch/record/401974.

[6] *Letter of Intent for A Large Ion Collider Experiment [ALICE]*. Tech. rep. Geneva: CERN, 1993. URL: https://cds.cern.ch/record/290825.

[7] P Buncic, M Krzewicki, and P Vande Vyvre. *Technical Design Report for the Upgrade of the Online-Offline Computing System*. Tech. rep. Apr. 2015. URL: https://cds.cern.ch/record/2011297.

[8] *Installation via alibuild · ALICE Analysis Tutorial*. URL: https://alice-doc.github.io/alice-analysis-tutorial/building/custom.html (visited on 09/26/2022).

[9] *ITS Info Page*. URL: https://alice-collaboration.web.cern.ch/menu_proj_items/its (visited on 07/18/2022).

[10] Rudolf E. Kálmán. "A new approach to linear filtering and prediction problems" transaction of the asme journal of basic". In: 1960. DOI: doi:10.1115/1.3662552.

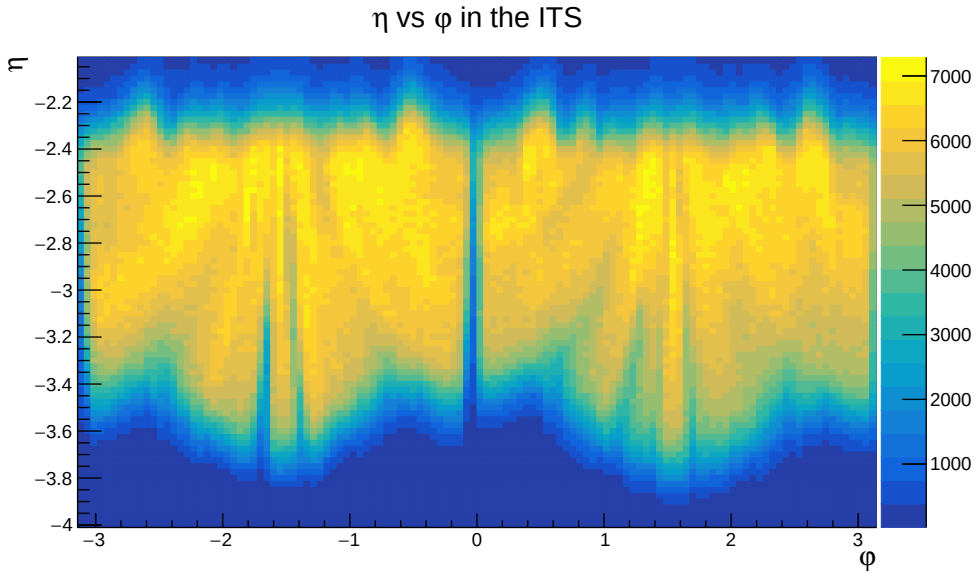


Figure 4.5: caption

[11] Deepak Kar. *Experimental Particle Physics*. 2053-2563. IOP Publishing, 2019. ISBN: 978-0-7503-2112-9. DOI: 10.1088/2053-2563/ab1be6. URL: https://dx.doi.org/10.1088/2053-2563/ab1be6.

[12] *Muon Spectrometer - ALICE Collaboration*. URL: https://alice-collaboration.web.cern.ch/menu_proj_items/Muon-Spect (visited on 07/17/2022).

[13] Izaak Neutelings. *CMS coordinate system*. URL: https://tikz.net/axis3d_cms/ (visited on 07/14/2022).

[14] *Technical Design Report for the Muon Forward Tracker*. Tech. rep. Jan. 2015. URL: https://cds.cern.ch/record/1981898.

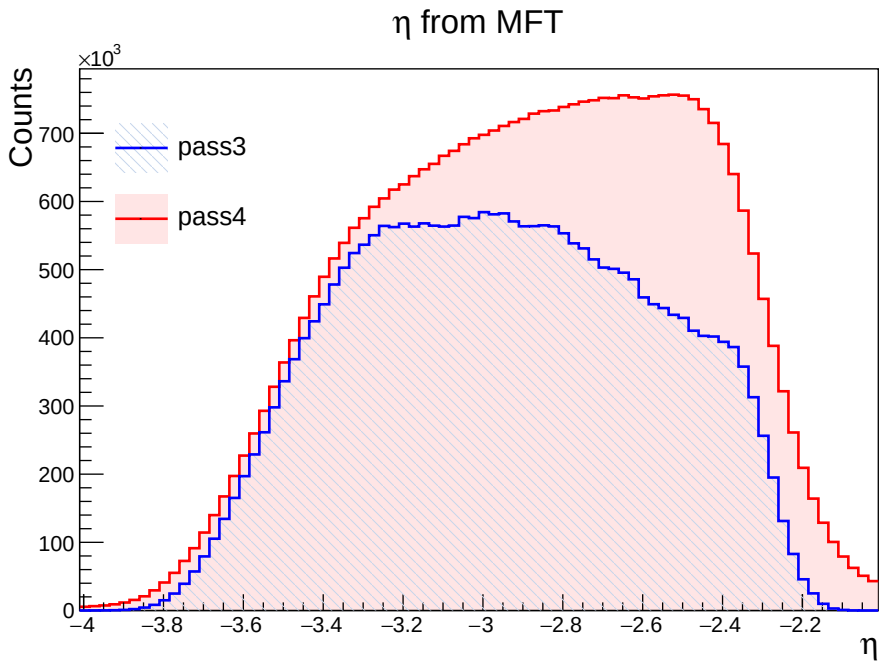


Figure 4.6: caption

Appendix

A Downloading data from the GRID

```
# The directory in alimonitor where you want to get the data from. Should contain a load of numbered directories
# For example, /alice/data/2021/OCT/505548/AOD. Note that it's AOD not AO2D
sourceMotherDir=/alice/data/path/to/AOD
nFiles=$1

# This is the directory on your local machine where you want to store your data
targetMotherDir=/path/to/alice/data/${sourceMotherDir}
mkdir -p ${targetMotherDir}

for ((i=1; i<=nFiles; i++))
do
    if (( i < 10 )); then
        pref=00
    fi
    if (( i > 9 )); then
        pref=0
    fi
    iDir=${pref}${i}
    iSourceDir=${sourceMotherDir}/${iDir}
    iTargetDir=${targetMotherDir}/${iDir}
    mkdir -p ${iTargetDir}

    echo "copying from ${iSourceDir} to ${iTargetDir}"

```

```
25          alien_cp -retry 5 ${iSourceDir}/AO2D.root file://${iTargetDir}/AO2D.root
26      done
```