Identificación de partículas en colisiones p-p por medio de AliRoot en distintas configuraciones de energía

Asesor: Mario Rodríguez Cahuantzi (mario.rodriguez@correo.buap.mx).

Benemérita Universidad Autónoma de Puebla.

Estudiantes: Marcel Alfaro[1] (jmarcelalfarosantos@gmail.com), Moisés Lechuga[2] (moises.lechuga@alumnos.udg.mx) & Alexis Ureña[3] (alursa2000@gmail.com).

[1,3] *Universidad de Sonora,* [2] *Universidad de Guadalajara.*

13 de agosto de 2020

Resumen

Se utilizó el software AliRoot para obtener histogramas de datos reconstruidos a fin de identificar y comparar la producción de piones, kaones y protones a partir de colisiones entre protones a 900 GeV y a 13 TeV y contrastando con las gráficas producidas por partículas ya identificadas con ALICE-LHC en las mismas condiciones.

Planteamiento del problema

El experimento ALICE

ALICE es el acrónimo de *A Large Ion Collider Experiment* ("Un Experimento de Colisionador de Iones Pesados"), un detector de iones pesados en el LHC (el Gran Colisionador de Hadrones) [1], [2].

Este colisionador de partículas se construyó entre 1998 y 2008, pero se concibió y planifico muchos años antes. De igual manera, ALICE tuvo sus orígenes antes de que comenzara la construcción del colisionador, en 1992, donde se propuso la idea de un detector de iones pesados [3] para el LHC, que en ese entonces todavía se estaba planificando.

Así pues, es notable la cantidad de tiempo y recursos necesarios para llevar a cabo proyectos de este tamaño, por lo que es importante asegurar que todo funcione como se espera, por lo que se crearon simuladores de colisionadores, donde se pueden reconstruir y simular colisiones de partículas en distintas condiciones, además de calibrar los diferentes componentes de los aparatos.

Uno de estos es AliRoot, que permite reconstruir colisiones de las primeras dos corridas del LHC (2009, 2015) y simular colisiones en la tercera corrida (2021). Este software es el que se usó para nuestra estancia virtual de Física de Altas Energías y Astropartículas.

Identificación de partículas mediante la ecuación dE/dx

La ecuación de Bethe (dE/dx) describe la pérdida energética respecto a la distancia recorrida de partículas ligeras cargadas que atraviesan materia o son sometidas por el poder de frenado del material. Las partículas y la materia que interactúan se transfieren energía mutuamente provocando ionización y cambios en los momentos y las trayectorias de las partículas, debido la diferencia entre cargas [4].

Dependiendo de la densidad y carga, la transferencia es distinta, siendo mayor cuando la densidad de los átomos es más alta y las cargas son muy distintas.

Extrapolando la partícula cargada ligera a protones y la materia a un detector, la detección de partículas es posible, debido a que los detectores no miden la pérdida de energía pero sí la energía depositada en él [5].

Un tipo de detector que utiliza este principio es el TPC (Time Projection Chamber) siendo un cilindro de $88m^3$ lleno de gas el cual se ioniza con la presencia de partículas cargadas y mediante sus dos zonas de detección permite una mejor lectura de la energía depositada por las partículas que posteriormente se proceden a identificar.

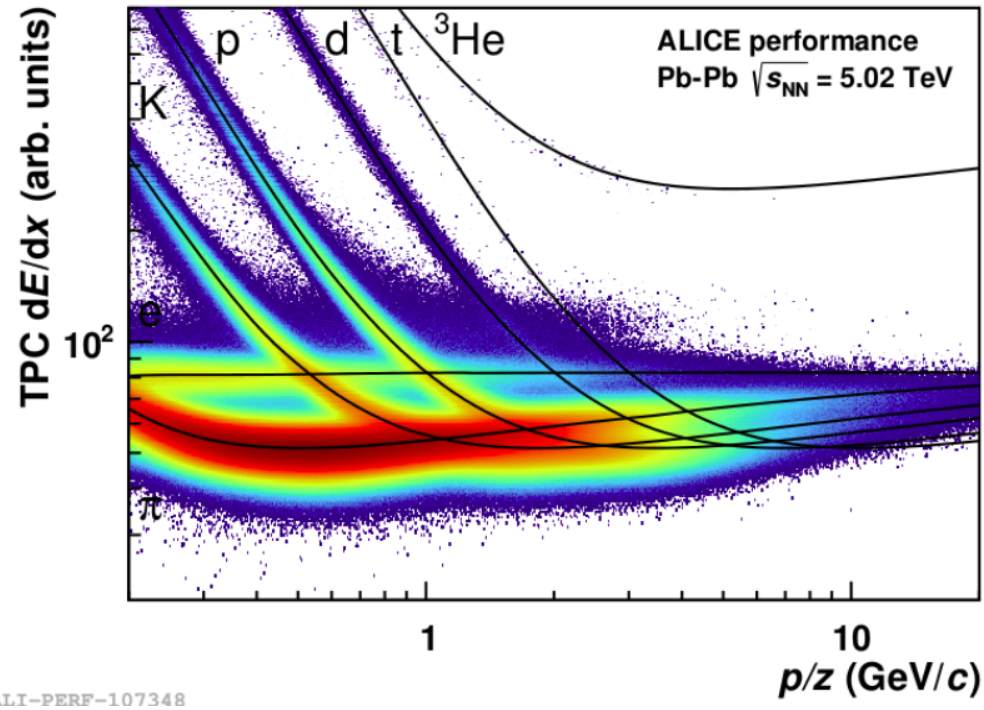

Figura 1: Ejemplo de una gráfica donde se puede apreciar la perdida de energía por distancia viajada en una TPC. [6].

Metodología

Un tutorial más detallado se puede seguir en `https://alice-doc.github.io`. Este método funciona en Ubuntu 20.04 LTS.

Instalación de software

En la realización de este proyecto es necesario contar con:

- Una distribución de Linux o macOS (recomendable).

- Docker
- Alidock
- AliPhysics
- AliRoot

Si no se cuenta con alguno de los sistemas operativos recomendados, todavía es posible realizar la reproducción de este trabajo, sin embargo, será más difícil encontrar soporte técnico.

Si no se tiene una máquina dedicada a alguno de estos sistemas operativos, es posible particionar el disco duro para instalar el sistema operativo deseado. Si no se desea crear una nueva partición, también es posible trabajar por medio de una máquina virtual, aunque esto limitaría la cantidad de recursos disponibles para ejecutar los procesos requeridos.

Contando con el sistema operativo de su preferencia, se procederá a instalar Docker, que es un software que facilita el desarrollo de código haciendo uso de *containers* [7]. Docker nos permitirá comenzar a construir software de ALICE, ya que utilizaremos Alidock.

Alidock es una herramienta aportada por la comunidad que proporciona a los usuarios un entorno coherente y *builds* precompiladas [8]. Es nuestra forma recomendada de instalar y usar el software de ALICE. Después de instalar Docker, es conveniente agregar Docker como usuario privilegiado (en Ubuntu). Hecho esto, procedemos a instalar Alidock como tal, escribiendo

```
bash <(curl -fsSL https://raw.githubusercontent.com/alidock/alidock/master/
alidock-installer.sh)
```

en la terminal. Terminado el proceso, se recomienda abrir una nueva terminal, donde podremos escribir `alidock`, que cargará el entorno.

Una vez en el entorno de Alidock, es recomendable crear un directorio dedicado al uso y construcción de software de ALICE. En este directorio se instalarán las paqueterías necesarias para AliPhysics y AliRoot (y si se desea, O2 – Run 3). Para descargar, escribimos:

```
aliBuild init AliPhysics@master
aliBuild init AliRoot@master
```

Cuando terminen de descargarse los paquetes, podemos ingresar a los directorios que generaron y actualizarlos con

```
git checkout master  # use dev instead of master for O2
git pull --rebase
```

en cada uno. Finalmente, escribimos

```
aliBuild build AliPhysics --defaults user-next-root6
```

Esto nos instalará el software que usaremos para comenzar las simulaciones. Cuando termine la instalación podemos escribir `alienv q` para enlistar todos los paquetes disponibles.

Ejecución del software

El primer paso es clonar el repositorio del tutorial a nuestro directorio de trabajo. Eso lo haremos estando en el entorno de Alidock, escribiendo

```
git clone https://github.com/rbertens/ALICE_analysis_tutorial.git
```

Esto creará una nueva carpeta con los siguientes archivos:

- `AliAnalysisTaskMyTask.cxx`
- `AliAnalysisTaskMyTask.h`
- `AddMyTask.C`
- `runAnalysis.C`

Además de estos archivos, necesitaremos un archivo que contenga información de colisiones reconstruidas. En este caso se solicitó al investigador el archivo `AliAOD.root`, y este archivo debe ir en la misma carpeta que el tutorial.

A continuación podemos ejecutar el programa para verificar que funcione correctamente cargando primero el entorno AliPhysics con

```
alienv enter AliPhysics/latest
```

y una vez dentro escribimos

```
aliroot runAnalysis.C
```

lo que cargará AliRoot y dentro de este correrá el programa `runAnalysis.C`. Posteriormente examinaremos los resultados escribiendo

```
new TBrowser
```

que abrirá un explorador de archivos gráfico. Esto lo veremos con más detalle en los resultados; por ahora sólo hay que verificar que la terminal no arroje ningún error (puede arrojar un par de advertencias que podemos ignorar de momento). Así, pues, se debe haber generado un nuevo archivo en el directorio del tutorial llamado `AnalysisResults.root`.

Ahora podemos comenzar a modificar el código base. Lo primero que se hace es agregar criterios de selección de eventos y de *tracks*. Para ello agregaremos nuevos histogramas donde se guardarán las posiciones de los vértices primarios. Esto lo hacemos agregando los *pointers* pertinentes en el *header* (el archivo `.h`), agregando los inicializadores correspondientes en los constructores de clase del archivo `.cxx` y agregando las instancias de los nuevos histogramas en la función `UserCreateOutput`. Una vez agregados los histogramas, buscamos la función `UserExec`, donde también agregaremos los ciclos *if* (como en el histograma de prueba que ya viene en el archivo), pero añadiendo la línea

```
float vertexZ = fAOD->GetPrimaryVertex()->GetZ();
```

Posteriormente, querremos modificar el código de tal forma que sólo se seleccionen los eventos que ocurren en un radio de 10cm del centro del detector. Esto hará que nos deshagamos de algunos datos; pero además de deshacernos de estos datos, podemos seleccionar otros mientras los tomamos. Esto se hace por medio de *triggers*, que modificamos en el archivo `AddMyTask.C`, en la siguiente línea:

```
task->SelectCollisionCandidates(AliVEvent::kAnyINT);
```

Como se puede observar, los *triggers* están definidos en el archivo `AliVEvent.h`. Si modificamos esta línea con alguno de los otros *triggers*, podremos observar algunos cambios en los histogramas. Por ahora dejaremos la línea como estaba originalmente (con el *trigger* `kAnyINT`, que aceptará cualquier interacción).

Ahora, se grafica las distribuciones de ángulo acimutal y pseudorapidez y nos aseguramos de que los ejes tengan los rangos apropiados. Una forma de extraer esta información dando un vistazo al header `AliAODTrack.h`, que debería encontrarse en los archivos que se instalaron al inicio, si todo se realizó correctamente. Hecho esto, podemos filtrar algunos *tracks* tal como lo hicimos con los eventos que no nos servían, modificando el `filterbit`, que hemos visto en la función `UserExec` del archivo `.cxx` del tutorial. Cambiaremos esta opción de 1 a 128 a 512. Es recomendable guardar un nuevo histograma para cada configuración.

Por último, tratamos de identificar piones cargados usando la energía perdida por unidad de distancia en la TPC (*Time Projection Chamber*). Básicamente:

- Identificar piones.
- Checar que tan 'pura' es nuestra muestra de piones.

Para esto, hay que guardar las señales de la TPC para todas las partículas cargadas en nuestros eventos en un histograma bidimensional. Usaremos

```
fYour2DHistogram->Fill(track->P(), track->GetTPCsignal());
```

en lugar de la línea usual al final de los ciclos en `UserExec`. Como un segundo paso, tenemos que identificar las partículas y guardar solamente la señal de la TPC que corresponde a los piones. A continuación, agregamos al archivo `runAnalysis.C` las siguientes líneas, antes de ejecutar nuestro *task*:

```
// load the macro and add the task
TMacro PIDadd(gSystem->ExpandPathName("$ALICE_ROOT/ANALYSIS/macros/
AddTaskPIDResponse.C"));
AliAnalysisTaskPIDResponse* PIDresponseTask =
reinterpret_cast<AliAnalysisTaskPIDResponse*>(PIDadd.Exec());
```

Además, hay que agregar una nueva clase en el *header*: `class AliPIDResponse`, además del *pointer* en sí, `AliPIDResponse* fPIDResponse; //! pid response object`. Agregamos la clase en el `.cxx`, junto con los inicializadores correspondientes al `fPIDResponse` en los constructores de clase y finalmente incluimos el *header* correspondiente: `AliPIDResponse.h` al inicio del `.cxx`. Esto es por medio del siguiente código:

```
AliAnalysisManager *man = AliAnalysisManager::GetAnalysisManager();
if (man) {
    AliInputEventHandler* inputHandler = (AliInputEventHandler*)
    (man->GetInputEventHandler());
    if (inputHandler)   fPIDResponse = inputHandler->GetPIDResponse();
}
```

que agregamos justo al final del `UserExec`. Con esto implementado, podemos extraer información de los diferentes tipos de partículas con las siguientes funciones, insertándolas al final del ciclo *if* en `UserExec`:

```
double kaonSignal = fPIDResponse->NumberOfSigmasTPC(track, AliPID::kKaon);
double pionSignal = fPIDResponse->NumberOfSigmasTPC(track, AliPID::kPion);
double protonSignal = fPIDResponse->NumberOfSigmasTPC(track, AliPID::kProton);
```

y podemos checar que la señal del `PIDResponse` se encuentre dentro de 3 desviaciones estándar por medio de

```
if (std::abs(fPIDResponse->NumberOfSigmasTPC(track, AliPID::kPion)) < 3 ) {
  // jippy, i'm a pion
};
```

A partir de esto se pueden crear histogramas para el momento transverso, la pseudorapidez y el ángulo de acimutal. Por último, accesamos a la centralidad de la colisión. Esto es por medio del siguiente código, insertándolo, de igual manera, antes de cargar nuestro *task*, en el archivo `runAnalysis.C`:

```
TMacro multSelection(gSystem->ExpandPathName("$ALICE_PHYSICS/OADB/COMMON/
MULTIPLICITY/macros/AddTaskMultSelection.C"));
AliMultSelectionTask* multSelectionTask = reinterpret_cast<AliMultSelectionTask*>
(multSelection.Exec());
```

y en `AliAnalysisTaskMyTask.cxx`, de igual manera, en `UserExec`, al final del ciclo:

```
Float_t centrality(0);
AliMultSelection *multSelection =static_cast<AliMultSelection*>
(fAOD->FindListObject("MultSelection"));
if(multSelection) centrality = multSelection->GetMultiplicityPercentile("V0M");
```

Hasta ahora, hemos usado datos obtenidos del periodo LHC150 con número de corrida 246757. A partir de aquí utilizaremos datos de un generador de Monte Carlo. Estos datos se obtienen de igual manera que el archivo `AliAOD.root`, como al inicio, y de igual manera se solicitó el archivo al investigador. Una vez teniendo el archivo, será recomendable cambiarle el nombre, por ejemplo, a `AliAOD_MC.root`, si se quieren conservar ambos archivos de datos. Si se cambia el nombre, será necesario cambiarlo también en todas las instancias en las que se utiliza el nombre anterior en el archivo `runAnalysis.C`. Una vez hecho esto, nos dirigiremos al *header* y agregaremos la siguiente variable:

```
AliMCEvent*             fMCEvent;       //! corresponding MC event
```

y, correspondientemente, los inicializadores a los constructores de clase en el `.cxx`. En `UserExec`, agregaremos

```
fMCEvent = MCEvent();
```

que nos permitirá utilizar nuestra nueva variable. Ahora usaremos la información creando una nueva función, declarando, primero, en el *header*:

```
virtual void ProcessMCParticles();
```

Ahora, agregaremos una nueva sección en el `.cxx`, como si se tratara de la función `UserExec` o `UserCreateOutput`, introduciendo el siguiente código entre, precisamente, estas dos funciones.

```
void AliAnalysisTaskMyTask::ProcessMCParticles()
{
    // process MC particles
    TClonesArray* AODMCTrackArray = dynamic_cast<TClonesArray*>(fInputEvent->
    FindListObject(AliAODMCParticle::StdBranchName()));
    if (AODMCTrackArray == NULL) return;

    // Loop over all primary MC particle
    for(Long_t i = 0; i < AODMCTrackArray->GetEntriesFast(); i++) {

      AliAODMCParticle* particle = static_cast<AliAODMCParticle*>
      (AODMCTrackArray->At(i));
      if (!particle) continue;
      cout << "PDG CODE = " << particle->GetPdgCode() << endl;
    }
}
```

Finalmente llamamos a la función en `UserExec`, justo debajo de `fMCEvent = MCEvent();`:

```
if(fMCEvent) ProcessMCParticles();
```

Esto nos arrojará una serie de códigos PDG. Seria recomendable, ahora, hacer un histograma con esta información.

Resultados

En la sección anterior se mencionaron varios histogramas generados por el programa. Estos y otros archivos pueden examinarse a través de TBrowser, el explorador de objetos de ROOT. Para acceder, tendremos que entrar al entorno de AliRoot (como al terminar de ejecutar el archivo `runAnalysis.C`. Una vez dentro sólo tenemos que escribir

```
new TBrowser
```

Esto abrirá el explorador, que es como cualquier otro explorador de archivos, con la excepción de poder abrir archivos `.root`. Así, si abrimos la carpeta del repositorio clonado (el tutorial) deberían aparacer, si se siguió este tutorial paso a paso, algunos archivos nuevos. Haciendo doble clic en `AliAnalysisResults.root` podremos observar dos carpetas. Examinaremos primero la de nombre `MyTask;1`. Siguiendo el camino, podremos ver varias gráficas y un archivo que contiene más información del proceso. Por ahora veremos sólamente los histogramas.

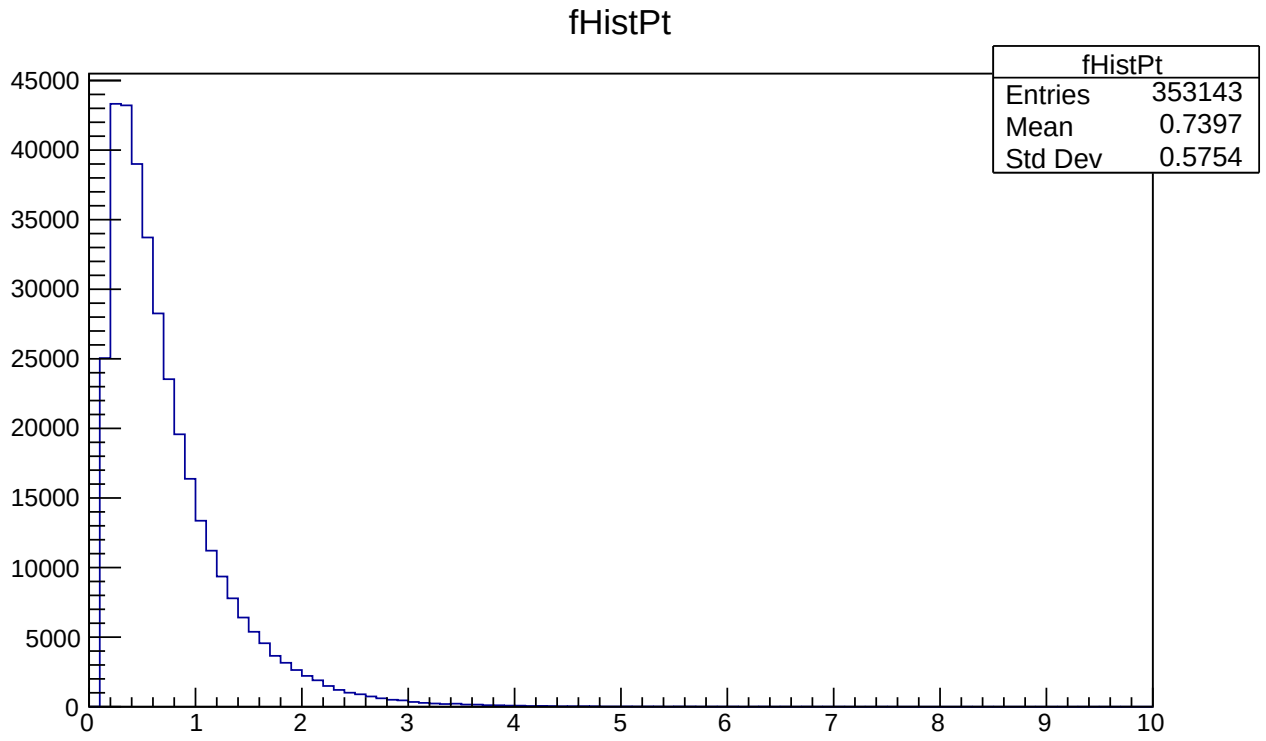

Figura 2: Histograma de prueba.

La figura 2 es la gráfica original que se obtuvo corriendo el código sin modificar, por primera vez.

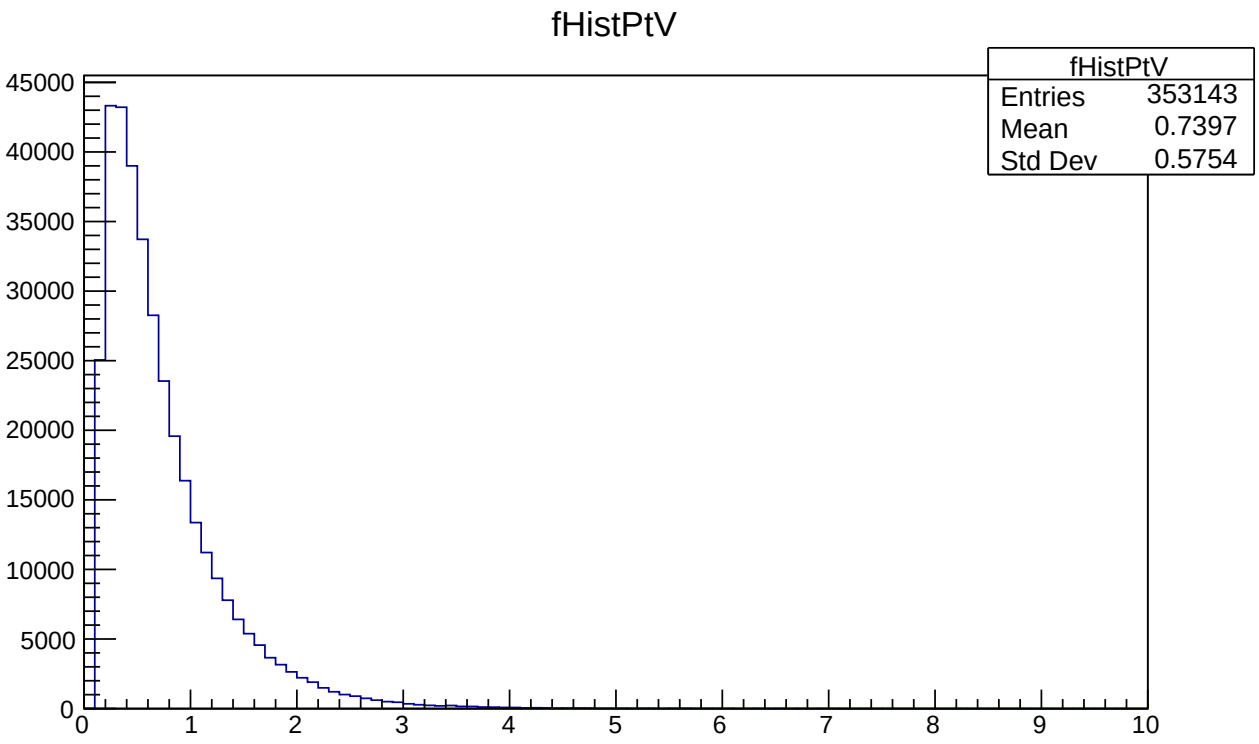

Figura 3: Histograma con vértices.

En la figura 3, se trató de realizar la implementación de los vértices en el histograma. Como se puede observar, fue un intento fallido, pues el histograma es idéntico al original, con excepción del nombre.

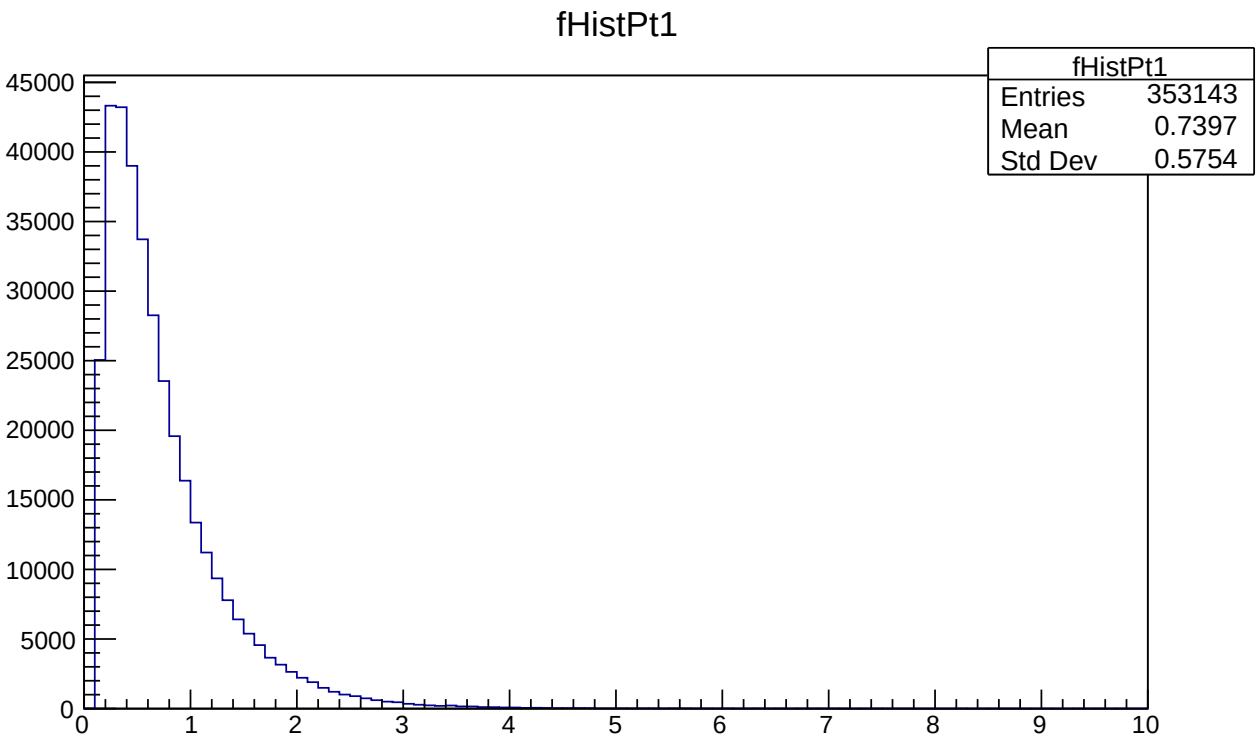

Figura 4: Histograma con `filterbit = 1`.

En la figura 4, el histograma también es idéntico al original, cambiando el nombre, ya que esta gráfica, junto con las siguientes dos, tienen fines ilustrativos en lo que concierne a los cambios del `filterbit`.

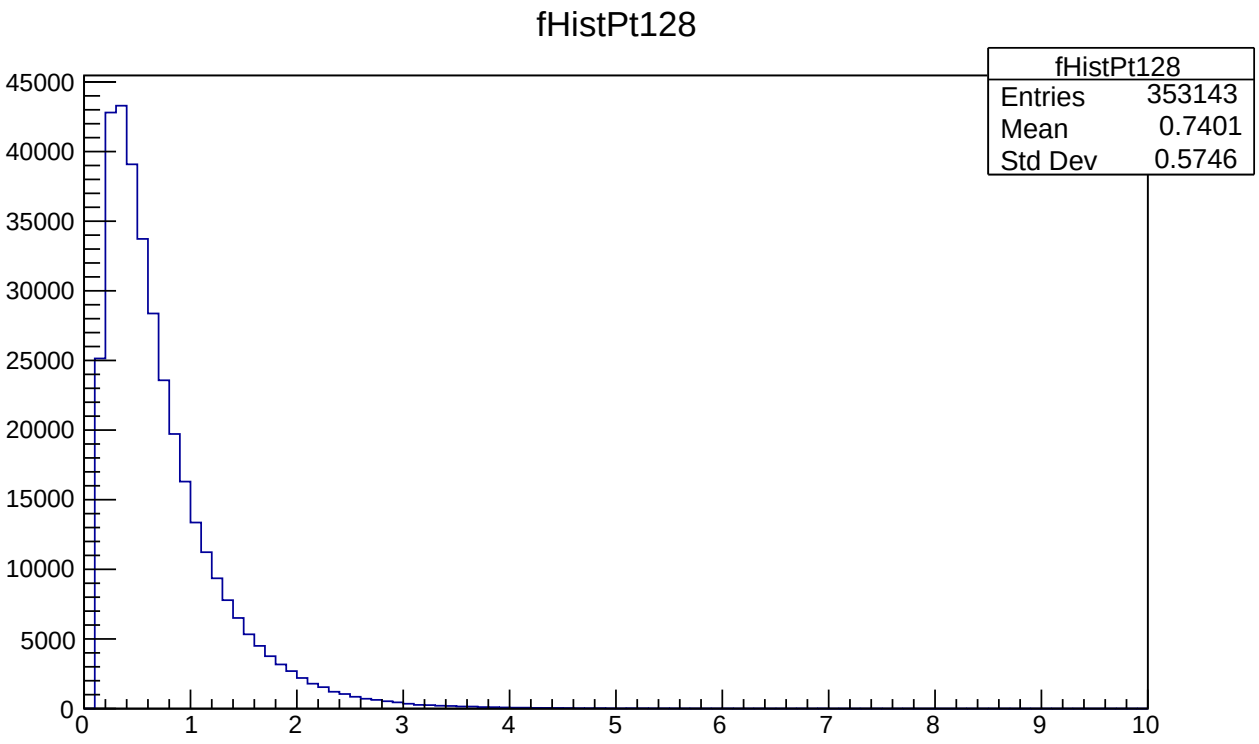

Figura 5: Histograma con `filterbit = 128`

En la figura 5 el histograma es similar al anterior, pero se pueden apreciar algunas diferencias, más notablemente, en el máximo de la curva. Aquí, cambiamos el filtro de 1 a 128.

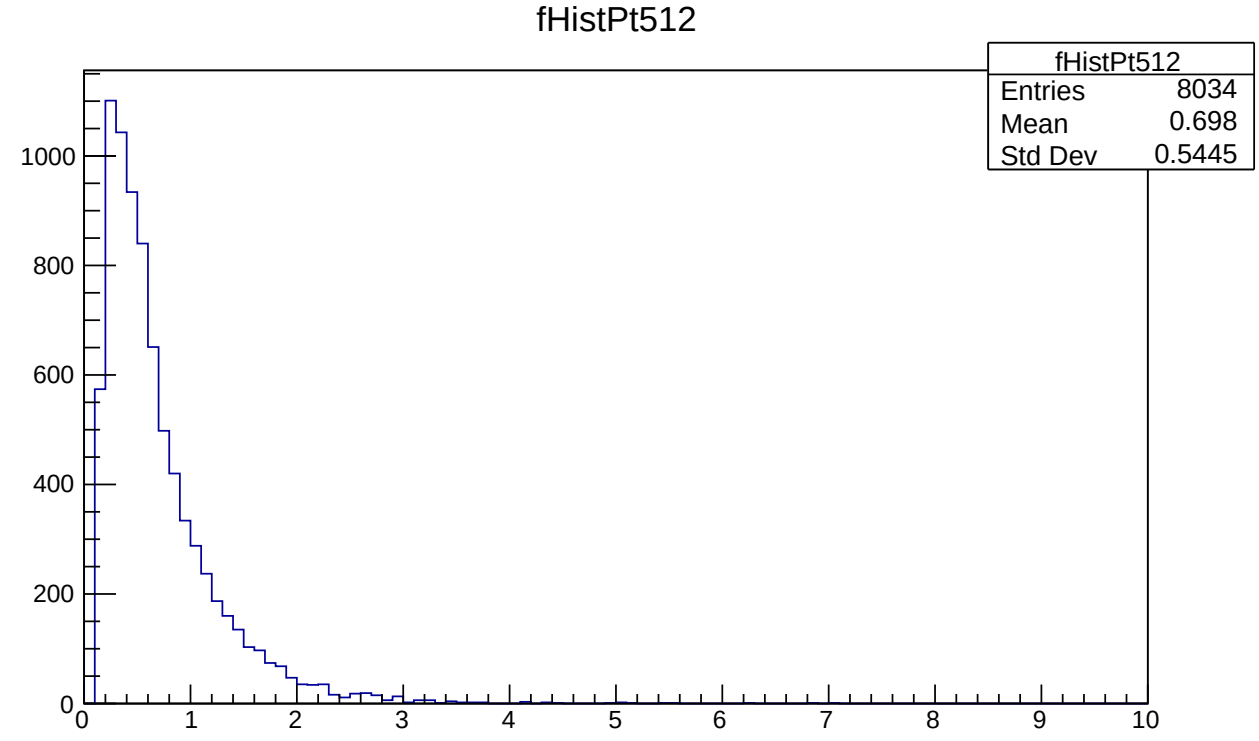

Figura 6: Histograma con `filterbit = 512`.

Aquí, en la figura 6, se puede apreciar un cambio más dramático en la forma de la curva, además del obvio cambio en el rango de los ejes.

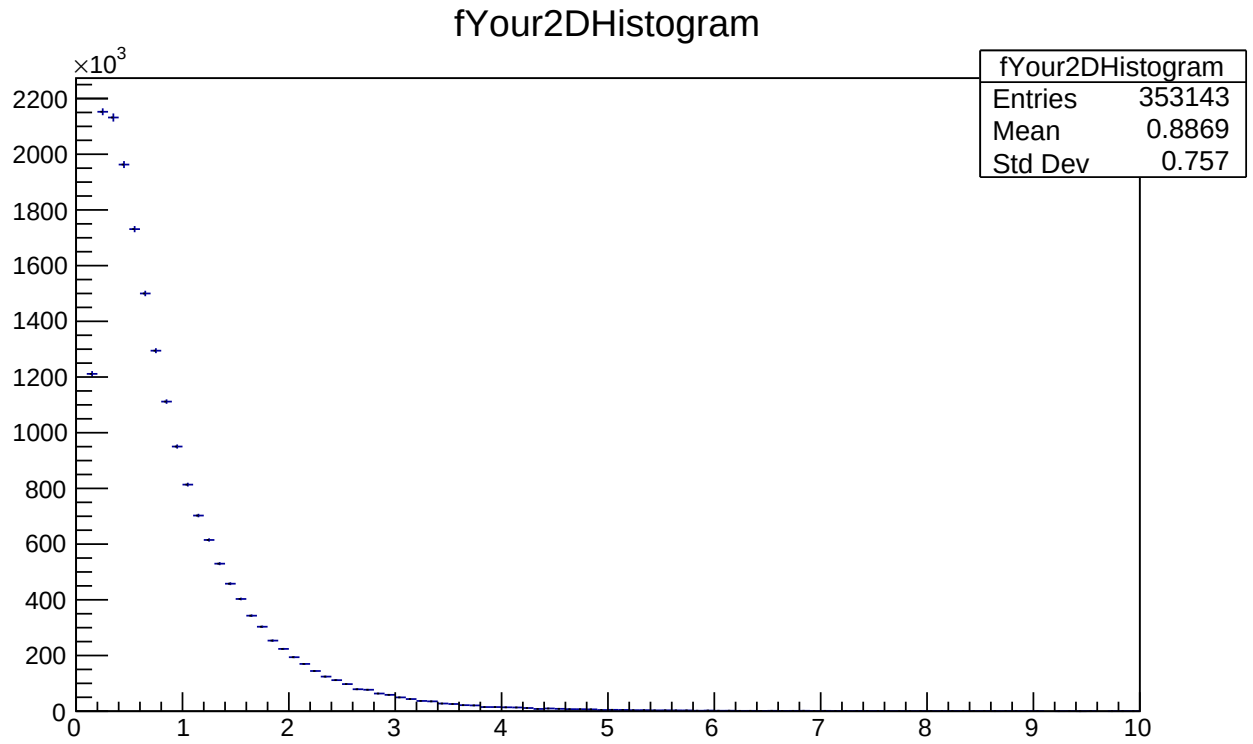

Figura 7: Histograma generado con el primer *snippet* del paso 5 del tutorial original.

La figura 7 es el histograma al inicio del paso 5. Presuntamente se representan las distribuciones de ángulo acimutal y de pseudorapidez.

A partir de aquí, las figuras se encuentran en la subcarpeta `MultSelection;1`, a excepción de la primera (la cual es arrojada por la terminal al terminar el proceso). Estas se generaron después de agregar el código posterior al que genera la figura 7.

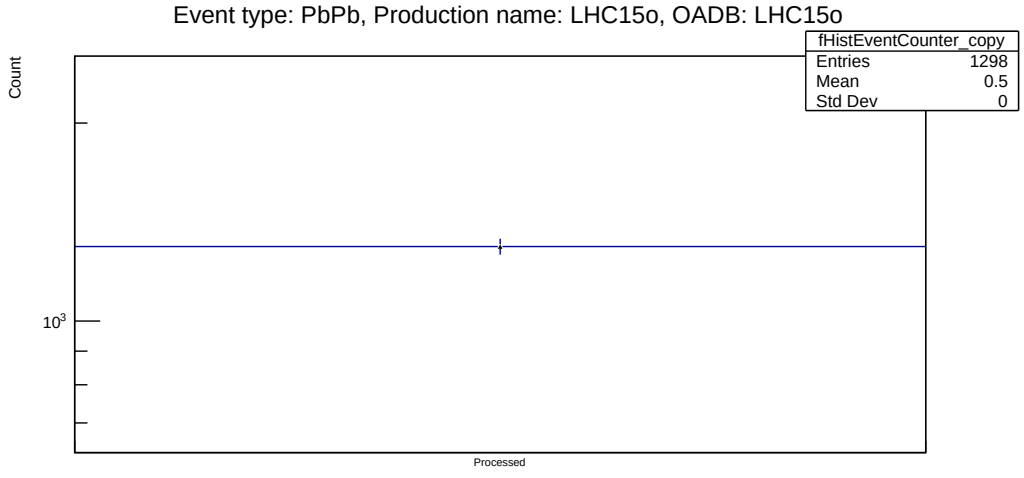

Figura 8: Contador inicial.

Esta figura (8) es la primera que arroja la terminal. No hay necesidad de acceder al `TBrowser`

para examinarla.

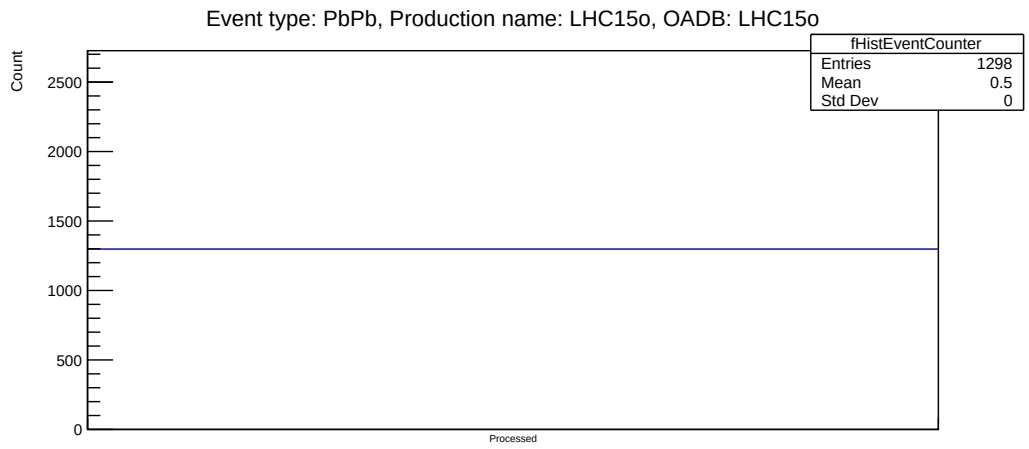

Figura 9: Contador 2

La figura 9 muestra el mismo contador que el que es arrojado por la terminal directamente, pero con algunos cambios en la escala de los ejes y sin la marca en el medio.

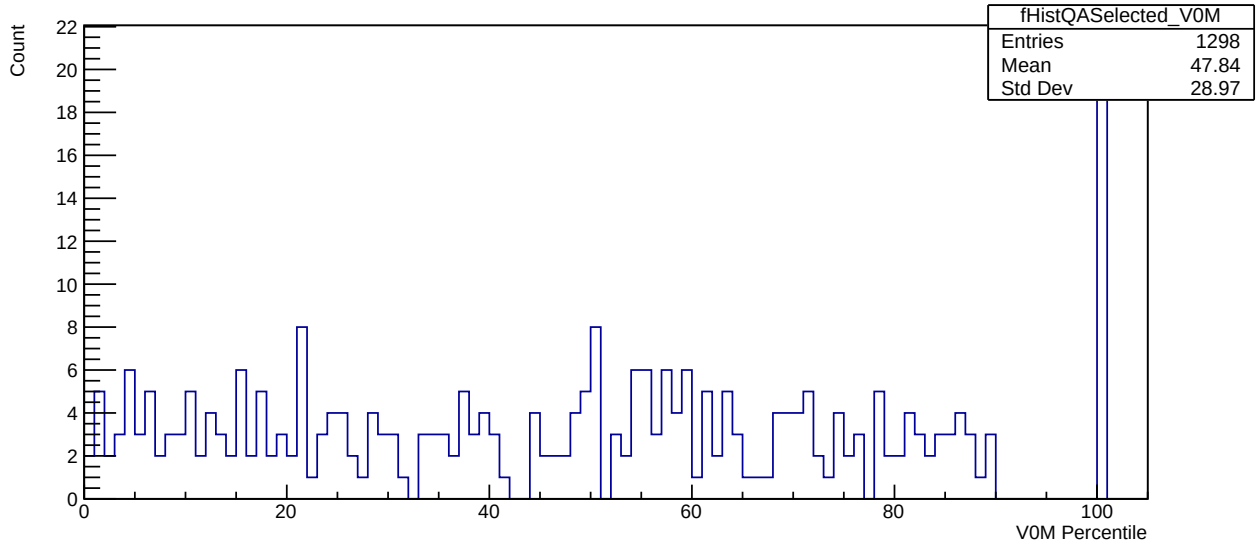

Figura 10: Percentiles.

En la figura 10, se muestra una especie de contador de eventos contra percentiles VOM (al parecer, centralidad).

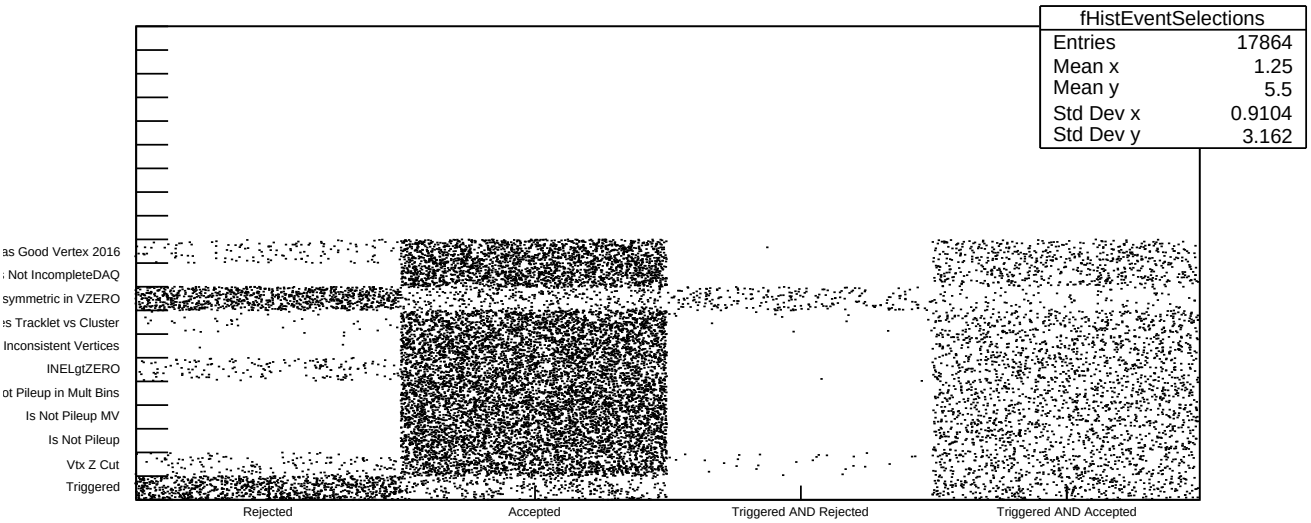

Figura 11: Esta gráfica presenta la selección de eventos.

En la gráfica de la figura 11 podemos observar los eventos seleccionados y los descartados en distintos segmentos del proceso.

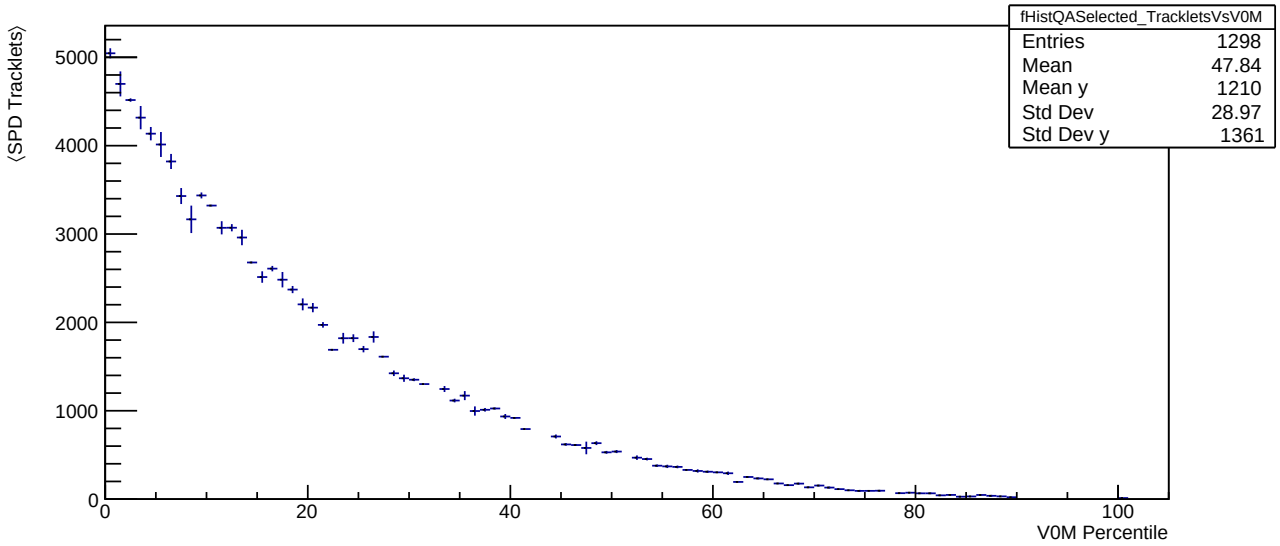

Figura 12: Esta gráfica presenta *tracklets* contra percentiles de centralidad.

Finalmente, en la figura 12, vemos una curva con elementos verticales y horizontales. Parece que se trata de *tracklets* contra percentiles de centralidad.

Las siguientes figuras son del generador Monte Carlo. Cada histograma y gráfica se obtuvo de igual manera que los originales. Nótese la diferencia comparando cada uno con su respectivo análogo.

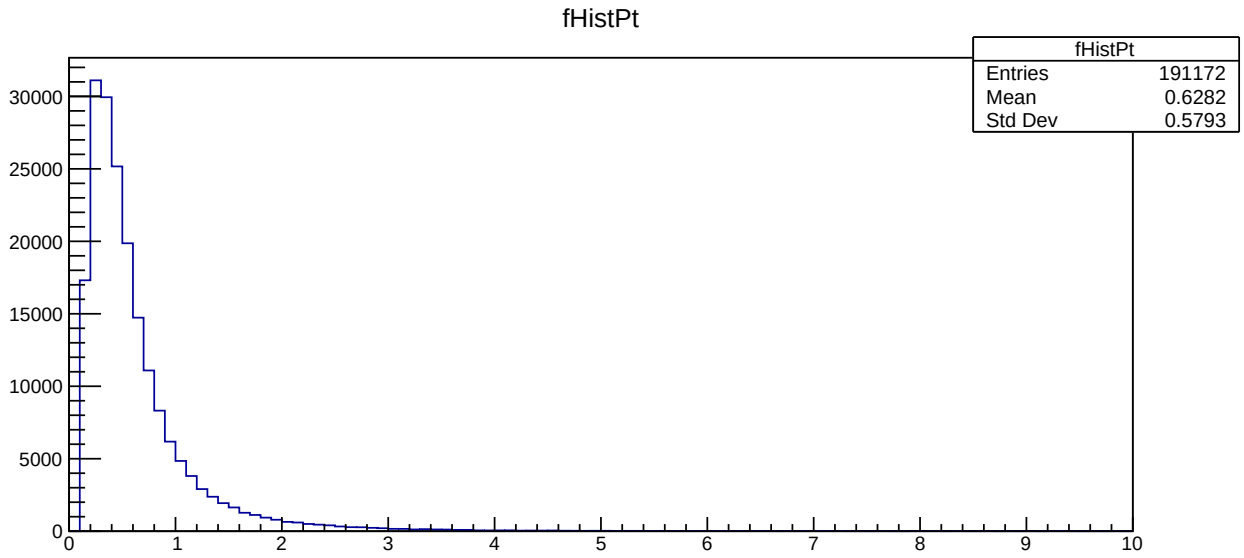

Figura 13: Histograma de prueba de MC.

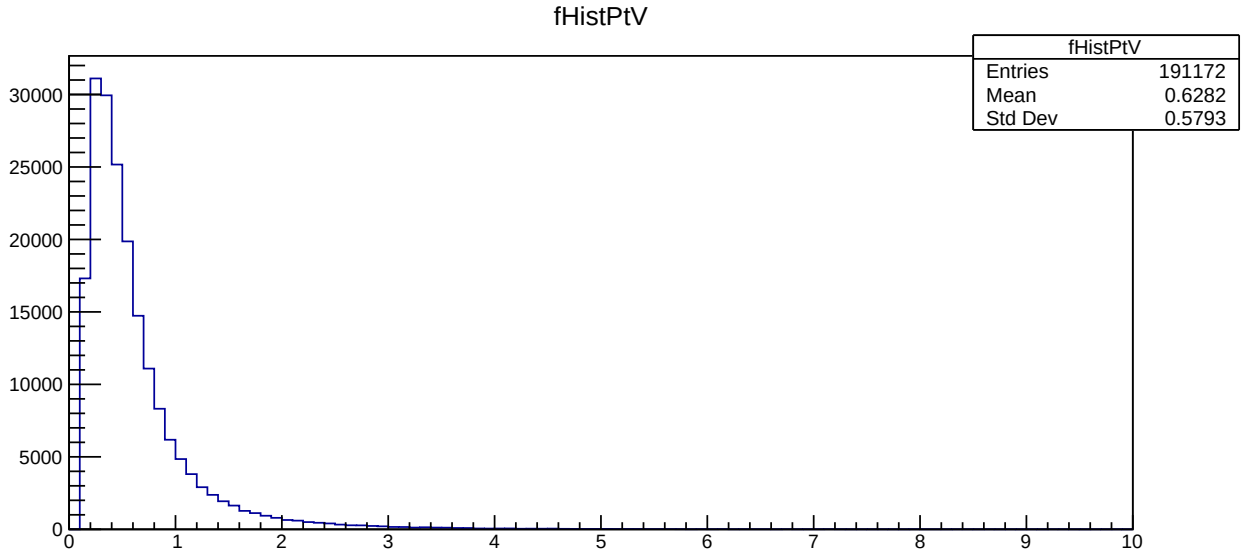

Figura 14: Histograma con vértices de MC.

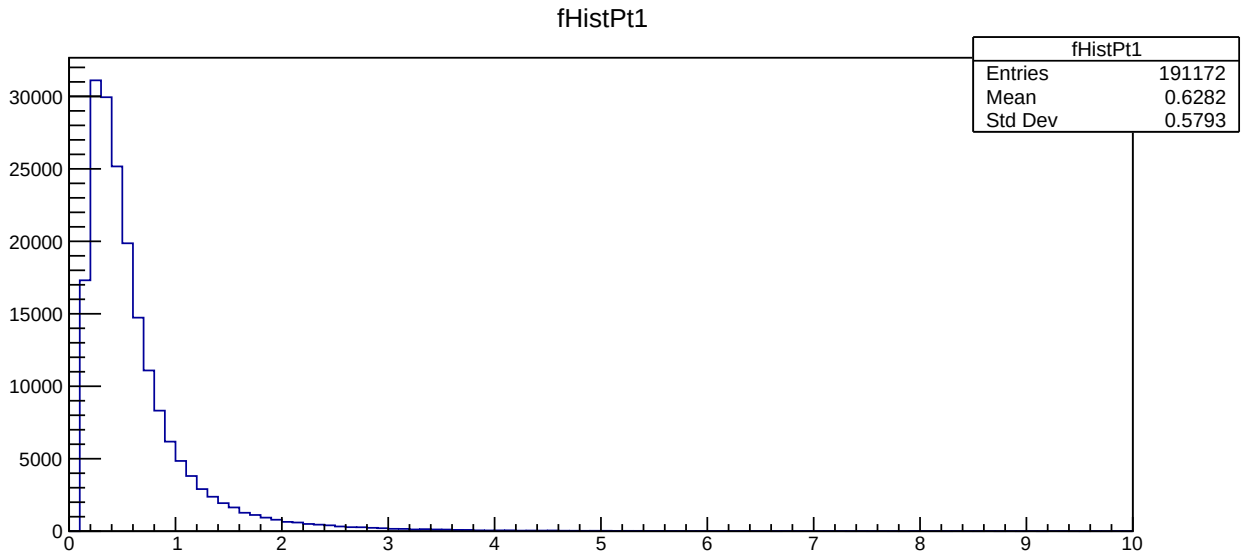

Figura 15: Histograma con `filterbit = 1` de MC.

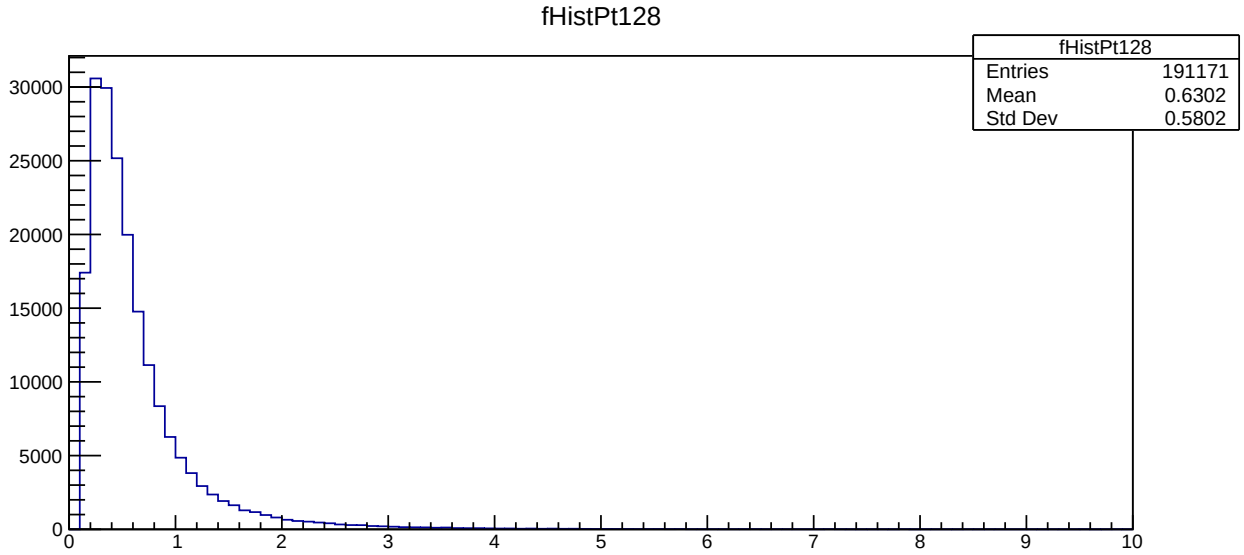

Figura 16: Histograma con `filterbit = 128` de MC.

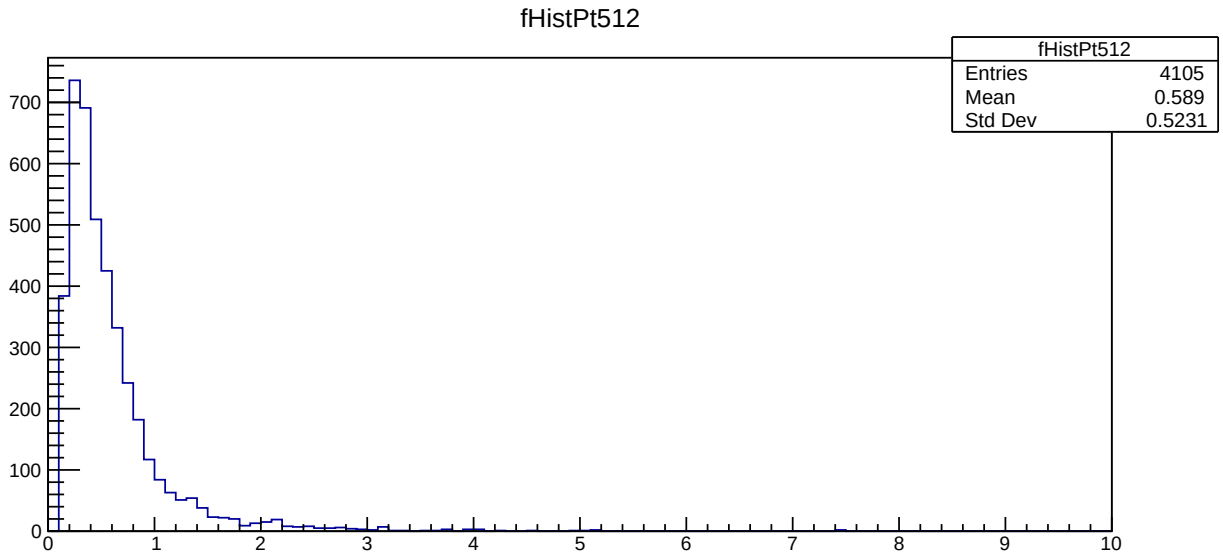

Figura 17: Histograma con `filterbit = 512` de MC.

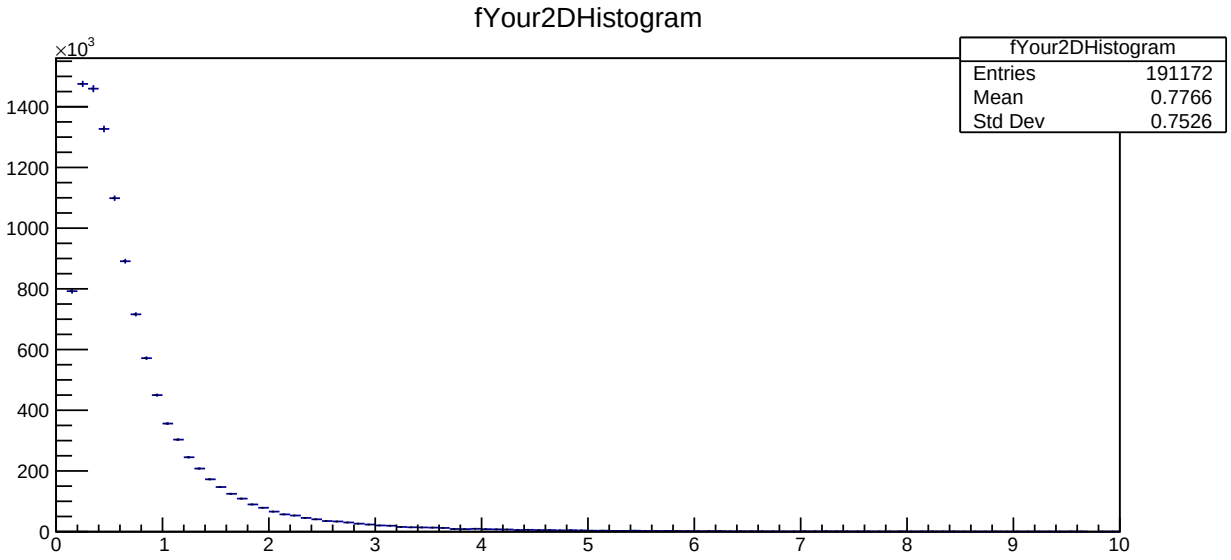

Figura 18: Histograma generado con el primer *snippet* del paso 5 del tutorial original, ahora con datos del MC.

Al igual que en el caso anterior, las siguientes gráficas pertenecen a la carpeta `MultSelection;1`, y la primera de ellas (figura 19) es también arrojada directamente por la terminal, aunque ahora se visualiza solamente después de enlistar todos y cada uno de los códigos PDG.

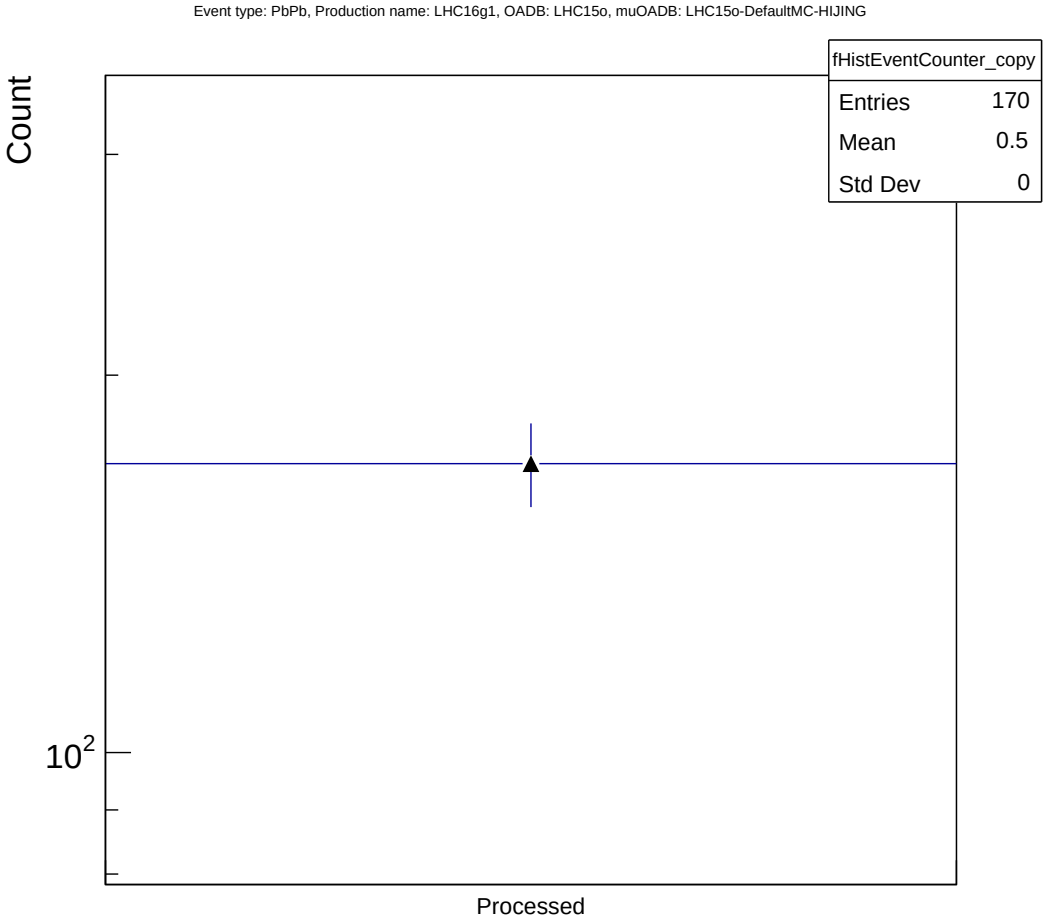

Figura 19: Contador inicial de MC.

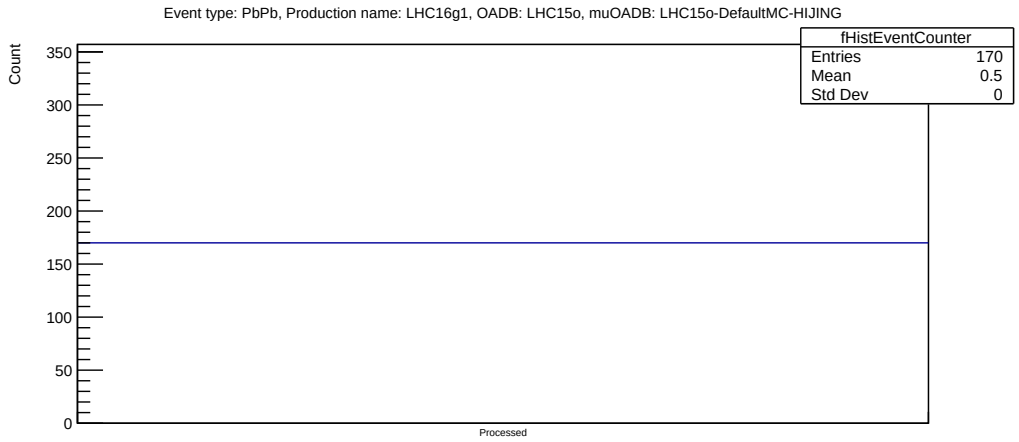

Figura 20: Contador 2 de MC

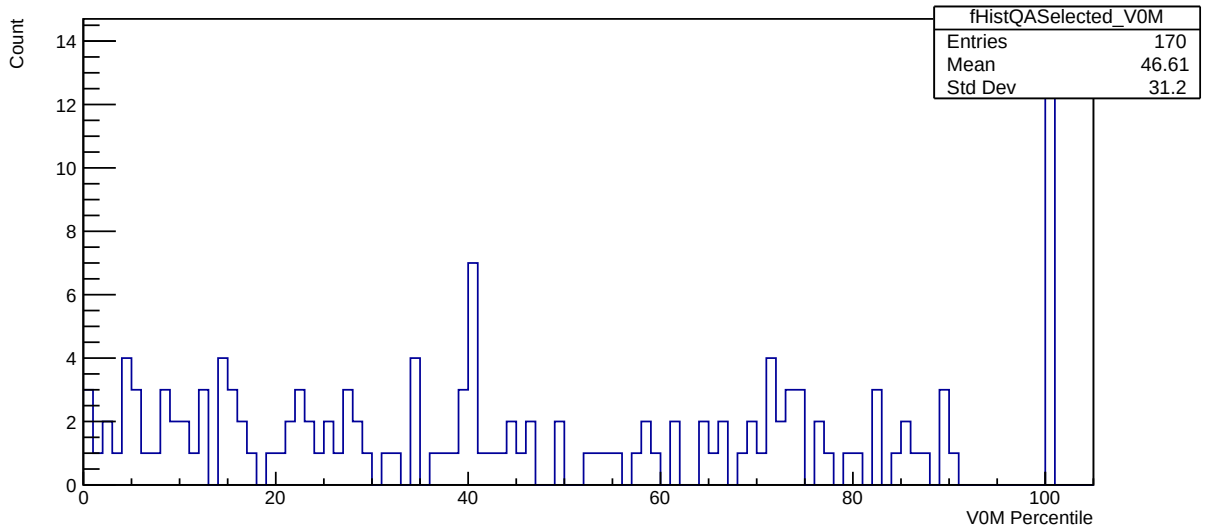

Figura 21: Percentiles de MC.

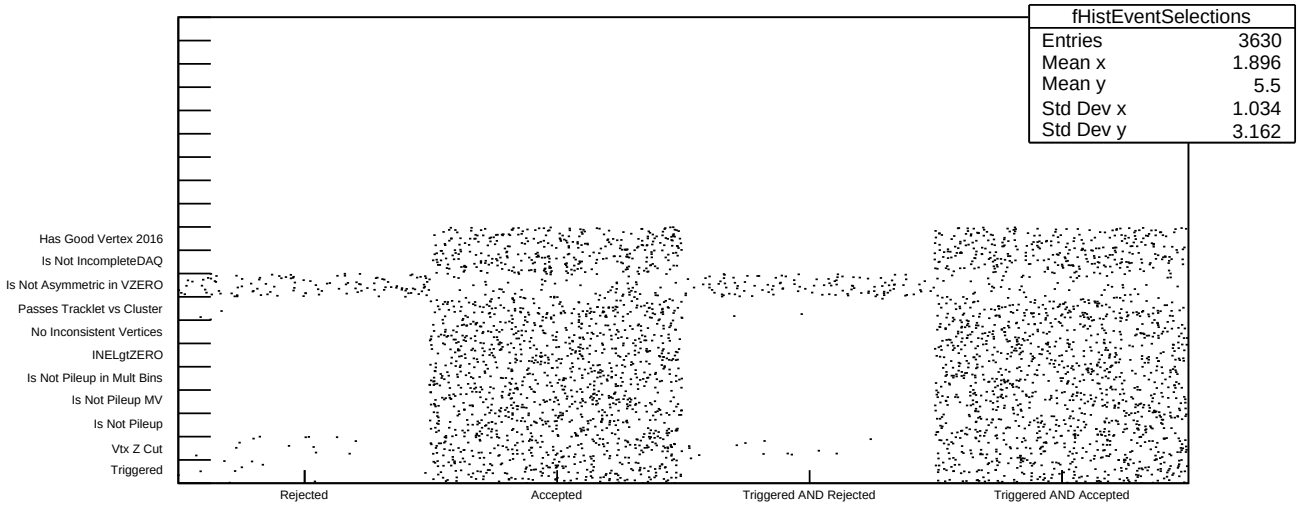

Figura 22: Esta gráfica presenta la selección de eventos de MC.

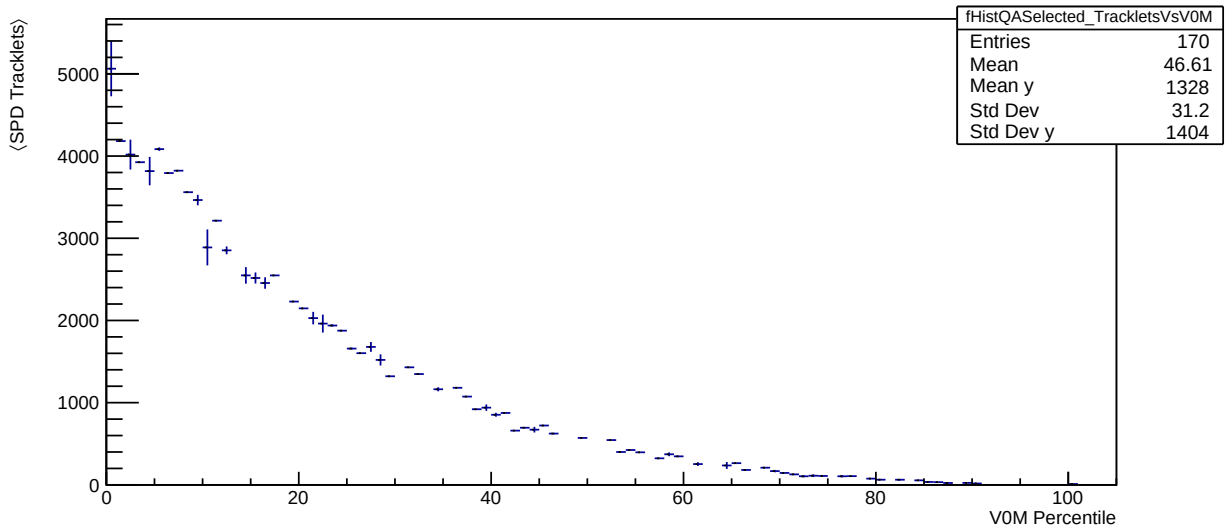

Figura 23: Esta gráfica presenta *tracklets* contra percentiles de centralidad de MC.

Conclusiones

Los datos obtenidos aún no pueden ser transcritos, pero a partir de estos se espera obtener las curvas de pérdida de energía respecto al momento de las partículas, identificando así las partículas producidas por la colisiones entre protones.

Anexos

Código

A continuación se presentan los códigos que se han modificado hasta ahora.

```
/**************************************************************************
 * Copyright(c) 1998-1999, ALICE Experiment at CERN, All rights reserved. *
 *                                                                        *
 * Author: The ALICE Off-line Project.                                    *
 * Contributors are mentioned in the code where appropriate.              *
 *                                                                        *
 * Permission to use, copy, modify and distribute this software and its   *
 * documentation strictly for non-commercial purposes is hereby granted   *
 * without fee, provided that the above copyright notice appears in all   *
 * copies and that both the copyright notice and this permission notice   *
 * appear in the supporting documentation. The authors make no claims     *
 * about the suitability of this software for any purpose. It is          *
 * provided "as is" without express or implied warranty.                  *
 **************************************************************************/

/* AliAnaysisTaskMyTask
 *
 * empty task which can serve as a starting point for building an analysis
 * as an example, one histogram is filled
 */

#include "TChain.h"
```

```
#include "TH1F.h"
#include "TList.h"
#include "AliAnalysisTask.h"
#include "AliAnalysisManager.h"
#include "AliAODEvent.h"
#include "AliAODInputHandler.h"
#include "AliAnalysisTaskMyTask.h"
#include "AliPIDResponse.h" //pid responde
#include "AliMCEvent.h" //para MC
#include "AliAODMCParticle.h" //para MC
#include "AliMultSelection.h" //para centralidad

class AliPIDResponse;   //cargamos la clase de PID
class AliAnalysisTaskMyTask;    // your analysis class

using namespace std;            // std namespace: so you can do things like 'cout'

ClassImp(AliAnalysisTaskMyTask) // classimp: necessary for root

AliAnalysisTaskMyTask::AliAnalysisTaskMyTask() : AliAnalysisTaskSE(),
    fAOD(0), fOutputList(0), fHistPt(0), fHistPtV(0), fHistPt1(0), fHistPt128(0),
    fHistPt512(0), fYour2DHistogram(0), fPIDResponse(0), fMCEvent(0) //agregamos
    inicializadores
{
    // default constructor, don't allocate memory here!
    // this is used by root for IO purposes, it needs to remain empty
}
//_____________________________________________________________________________
AliAnalysisTaskMyTask::AliAnalysisTaskMyTask(const char* name) : AliAnalysisTaskSE(
    name),
    fAOD(0), fOutputList(0), fHistPt(0), fHistPtV(0), fHistPt1(0), fHistPt128(0),
    fHistPt512(0), fYour2DHistogram(0), fPIDResponse(0), fMCEvent(0)  //agregamos
    inicializadores
{
    // constructor
    DefineInput(0, TChain::Class());    // define the input of the analysis: in
    this case we take a 'chain' of events
                                        // this chain is created by the analysis
    manager, so no need to worry about it,
                                        // it does its work automatically
    DefineOutput(1, TList::Class());    // define the ouptut of the analysis: in
    this case it's a list of histograms
                                        // you can add more output objects by
    calling DefineOutput(2, classname::Class())
                                        // if you add more output objects, make
    sure to call PostData for all of them, and to
                                        // make changes to your AddTask macro!
}
//_____________________________________________________________________________
AliAnalysisTaskMyTask::~AliAnalysisTaskMyTask()
{
    // destructor
    if(fOutputList) {
        delete fOutputList;     // at the end of your task, it is deleted from
    memory by calling this function
    }
}
//_____________________________________________________________________________
```

```
70 void AliAnalysisTaskMyTask::UserCreateOutputObjects()
71 {
72     // create output objects
73     //
74     // this function is called ONCE at the start of your analysis (RUNTIME)
75     // here you ceate the histograms that you want to use
76     //
77     // the histograms are in this case added to a tlist, this list is in the end
       saved
78     // to an output file
79     //
80     fOutputList = new TList();          // this is a list which will contain all of
       your histograms
81                                         // at the end of the analysis, the contents
       of this list are written
82                                         // to the output file
83     fOutputList->SetOwner(kTRUE);       // memory stuff: the list is owner of all
       objects it contains and will delete them
84                                         // if requested (dont worry about this now)
85
86     // example of a histogram
87     fHistPt = new TH1F("fHistPt", "fHistPt", 100, 0, 10);       // create your
       histogra
88     fOutputList->Add(fHistPt);          // don't forget to add it to the list! the
       list will be written to file, so if you want
89     PostData(1, fOutputList);           // postdata will notify the analysis
       manager of changes / updates to the
90                                         // fOutputList object. the manager will in
       the end take care of writing your output to file
91                                         // so it needs to know what's in the output
92
93
94     fHistPtV = new TH1F("fHistPtV", "fHistPtV", 100, 0, 10);         //histograma
       vertices
95     fOutputList->Add(fHistPtV);
96     PostData(1, fOutputList);
97
98     fHistPt1 = new TH1F("fHistPt1", "fHistPt1", 100, 0, 10);     //histograma
       filterbit 1
99     fOutputList->Add(fHistPt1);
100    PostData(1, fOutputList);
101
102    fHistPt128 = new TH1F("fHistPt128", "fHistPt128", 100, 0, 10);     //histograma
       filterbit 128
103    fOutputList->Add(fHistPt128);
104    PostData(1, fOutputList);
105
106    fHistPt512 = new TH1F("fHistPt512", "fHistPt512", 100, 0, 10);     //histograma
       filterbit 512
107    fOutputList->Add(fHistPt512);
108    PostData(1, fOutputList);
109
110    fYour2DHistogram = new TH1F("fYour2DHistogram", "fYour2DHistogram", 100, 0, 10)
       ;     //histograma 2d
111    fOutputList->Add(fYour2DHistogram);
112    PostData(1, fOutputList);
113
114    fPIDResponse = new AliPIDResponse();     //response object
```

```
115     fOutputList->Add(fPIDResponse);
116     PostData(1, fOutputList);
117 
118 }
119 
120 //_____________________________________________________________________________
121 void AliAnalysisTaskMyTask::ProcessMCParticles()     //hacemos uso de la nueva
    funcion para MC
122 {
123     // process MC particles
124     TClonesArray* AODMCTrackArray = dynamic_cast<TClonesArray*>(fInputEvent->
    FindListObject(AliAODMCParticle::StdBranchName()));
125     if (AODMCTrackArray == NULL) return;
126 
127     // Loop over all primary MC particle
128     for(Long_t i = 0; i < AODMCTrackArray->GetEntriesFast(); i++) {
129 
130       AliAODMCParticle* particle = static_cast<AliAODMCParticle*>(AODMCTrackArray->
    At(i));
131       if (!particle) continue;
132       cout << "PDG CODE = " << particle->GetPdgCode() << endl;
133     }
134 }
135 
136 //_____________________________________________________________________________
137 void AliAnalysisTaskMyTask::UserExec(Option_t *)
138 {
139 
140 fMCEvent = MCEvent();    //sacamos informacion de los MC
141 if(fMCEvent) ProcessMCParticles();  //y la procesamos
142 
143 
144     // user exec
145     // this function is called once for each event
146     // the manager will take care of reading the events from file, and with the
    static function InputEvent() you
147     // have access to the current event.
148     // once you return from the UserExec function, the manager will retrieve the
    next event from the chain
149     fAOD = dynamic_cast<AliAODEvent*>(InputEvent());     // get an event (called
    fAOD) from the input file
150                     // there's another event format (ESD) which works in a similar
    wya
151                                                          // but is more cpu/memory
    unfriendly. for now, we'll stick with aod's
152     if(!fAOD) return;                                    // if the pointer to the
    event is empty (getting it failed) skip this event
153         // example part: i'll show how to loop over the tracks in an event
154         // and extract some information from them which we'll store in a histogram
155 
156     Int_t iTracks(fAOD->GetNumberOfTracks());            // see how many tracks
    there are in the event
157 
158     for(Int_t i(0); i < iTracks; i++) {                  // loop ove rall these
    tracks
159         AliAODTrack* track = static_cast<AliAODTrack*>(fAOD->GetTrack(i));
    // get a track (type AliAODTrack) from the event
```

```
160         if(!track || !track->TestFilterBit(1)) continue;
    // if we failed, skip this track
161         fHistPt->Fill(track->Pt());                    // plot the pt value of the
    track in a histogram
162     }                                                  // continue until all the
    tracks are processed
163
164     PostData(1, fOutputList);                          // stream the results the
    analysis of this event to
165                                                        // the output manager which
     will take care of writing
166                                                        // it to a file
167
168     fAOD = dynamic_cast<AliAODEvent*>(InputEvent());        //histograma vertices
169     if(!fAOD) return;
170     Int_t jTracks(fAOD->GetNumberOfTracks());
171     for(Int_t j(0); j < jTracks; j++) {
172       AliAODTrack* track = static_cast<AliAODTrack*>(fAOD->GetTrack(j));
173       if(!track || !track->TestFilterBit(1))
174       continue;
175       float vertexZ = fAOD->GetPrimaryVertex()->GetZ();
176       fHistPtV->Fill(track->Pt());
177     }
178     PostData(1, fOutputList);
179
180     fAOD = dynamic_cast<AliAODEvent*>(InputEvent());          //histograma
    filterbit 1
181     if(!fAOD) return;
182     Int_t kTracks(fAOD->GetNumberOfTracks());
183     for(Int_t k(0); k < kTracks; k++) {
184       AliAODTrack* track = static_cast<AliAODTrack*>(fAOD->GetTrack(k));
185       if(!track || !track->TestFilterBit(1)) continue;
186       fHistPt1->Fill(track->Pt());
187     }
188     PostData(1, fOutputList);
189
190     fAOD = dynamic_cast<AliAODEvent*>(InputEvent());          //histograma
    filterbit 128
191     if(!fAOD) return;
192     Int_t mTracks(fAOD->GetNumberOfTracks());
193     for(Int_t m(0); m < mTracks; m++) {
194       AliAODTrack* track = static_cast<AliAODTrack*>(fAOD->GetTrack(m));
195       if(!track || !track->TestFilterBit(128)) continue;
196       fHistPt128->Fill(track->Pt());
197     }
198     PostData(1, fOutputList);
199
200     fAOD = dynamic_cast<AliAODEvent*>(InputEvent());          //histograma
    filterbit 512
201     if(!fAOD) return;
202     Int_t nTracks(fAOD->GetNumberOfTracks());
203     for(Int_t n(0); n < nTracks; n++) {
204       AliAODTrack* track = static_cast<AliAODTrack*>(fAOD->GetTrack(n));
205       if(!track || !track->TestFilterBit(512)) continue;
206       fHistPt512->Fill(track->Pt());
207     }
208     PostData(1, fOutputList);
209
```

```
210     fAOD = dynamic_cast<AliAODEvent*>(InputEvent());          //histograma 2d
211     if(!fAOD) return;
212     Int_t qTracks(fAOD->GetNumberOfTracks());
213     for(Int_t q(0); q < qTracks; q++) {
214       AliAODTrack* track = static_cast<AliAODTrack*>(fAOD->GetTrack(q));
215       if(!track || !track->TestFilterBit(1)) continue;
216       fYour2DHistogram->Fill(track->P(), track->GetTPCsignal());
217     double kaonSignal = fPIDResponse->NumberOfSigmasTPC(track, AliPID::kKaon);
    //identificacion de particulas
218     double pionSignal = fPIDResponse->NumberOfSigmasTPC(track, AliPID::kPion);
219     double protonSignal = fPIDResponse->NumberOfSigmasTPC(track, AliPID::kProton);
220     if (std::abs(fPIDResponse->NumberOfSigmasTPC(track, AliPID::kPion)) < 3 ) {
221    //jippy, i'm a pion
222     };
223     Float_t centrality(0);
224     AliMultSelection *multSelection =static_cast<AliMultSelection*>(fAOD->
    FindListObject("MultSelection")); //centralidad
225     if(multSelection) centrality = multSelection->GetMultiplicityPercentile("V0M");
226     PostData(1, fOutputList);
227     };
228
229 AliAnalysisManager *man = AliAnalysisManager::GetAnalysisManager();
230     if (man) {
231       AliInputEventHandler* inputHandler = (AliInputEventHandler*)(man->
    GetInputEventHandler());     //pid response
232       if (inputHandler)    fPIDResponse = inputHandler->GetPIDResponse();
233     }
234
235 }
236 //-----------------------------------------------------------------------------
237 void AliAnalysisTaskMyTask::Terminate(Option_t *)
238 {
239     // terminate
240     // called at the END of the analysis (when all events are processed)
241 }
242 //-----------------------------------------------------------------------------
```

Listing 1: Archivo `AliAnalysisTaskMyTask.cxx`.

```
1  /* Copyright(c) 1998-1999, ALICE Experiment at CERN, All rights reserved. */
2  /* See cxx source for full Copyright notice */
3  /* $Id$ */
4
5  #ifndef AliAnalysisTaskMyTask_H
6  #define AliAnalysisTaskMyTask_H
7
8  #include "AliAnalysisTaskSE.h"
9
10 class AliPIDResponse;    // agregamos la nueva clase
11
12 class AliAnalysisTaskMyTask : public AliAnalysisTaskSE
13 {
14     public:
15                                 AliAnalysisTaskMyTask();
16                                 AliAnalysisTaskMyTask(const char *name);
17         virtual                 ~AliAnalysisTaskMyTask();
18
19         virtual void            UserCreateOutputObjects();
20         virtual void            UserExec(Option_t* option);
```

```
21          virtual void            Terminate(Option_t* option);
22          virtual void            ProcessMCParticles();   //para los datos MC
23
24      private:
25          AliAODEvent*            fAOD;           //! input event
26          TList*                  fOutputList;    //! output list
27          TH1F*                   fHistPt;        //! dummy histogram
28          TH1F*       fHistPtV;   //! nuevo histograma con vertices
29          TH1F*       fHistPt1;       //! filterbit 1
30          TH1F*       fHistPt128;    //! filterbit 128
31          TH1F*       fHistPt512;    //! filterbit 512
32          TH1F*       fYour2DHistogram; //! histograma 2d
33          AliPIDResponse*  fPIDResponse; //! pid response object
34          AliMCEvent*             fMCEvent;       //! corresponding MC event
35
36          AliAnalysisTaskMyTask(const AliAnalysisTaskMyTask&); // not implemented
37          AliAnalysisTaskMyTask& operator=(const AliAnalysisTaskMyTask&); // not
    implemented
38
39          ClassDef(AliAnalysisTaskMyTask, 1);
40  };
41
42  #endif
```

Listing 2: Archivo `AliAnalysisTaskMyTask.h`.

```
1  // include the header of your analysis task here! for classes already compiled by
       aliBuild,
2  // precompiled header files (with extension pcm) are available, so that you do not
       need to
3  // specify includes for those. for your own task however, you (probably) have not
       generated a
4  // pcm file, so we need to include it explicitly
5  #include "AliAnalysisTaskMyTask.h"
6
7  void runAnalysis()
8  {
9      // set if you want to run the analysis locally (kTRUE), or on grid (kFALSE)
10     Bool_t local = kTRUE;
11     // if you run on grid, specify test mode (kTRUE) or full grid model (kFALSE)
12     Bool_t gridTest = kTRUE;
13
14     // since we will compile a class, tell root where to look for headers
15 #if !defined (__CINT__) || defined (__CLING__)
16     gInterpreter->ProcessLine(".include $ROOTSYS/include");
17     gInterpreter->ProcessLine(".include $ALICE_ROOT/include");
18 #else
19     gROOT->ProcessLine(".include $ROOTSYS/include");
20     gROOT->ProcessLine(".include $ALICE_ROOT/include");
21 #endif
22
23     // create the analysis manager
24     AliAnalysisManager *mgr = new AliAnalysisManager("AnalysisTaskExample");
25     AliAODInputHandler *aodH = new AliAODInputHandler();
26     mgr->SetInputEventHandler(aodH);
27
28
29
30     // compile the class and load the add task macro
```

```
31      // here we have to differentiate between using the just-in-time compiler
32      // from root6, or the interpreter of root5
33      // load the macro and add the task
34
35  TMacro PIDadd(gSystem->ExpandPathName("$ALICE_ROOT/ANALYSIS/macros/
        AddTaskPIDResponse.C"));   //agregamos el pid response
36  AliAnalysisTaskPIDResponse* PIDresponseTask = reinterpret_cast<
        AliAnalysisTaskPIDResponse*>(PIDadd.Exec());
37  TMacro multSelection(gSystem->ExpandPathName("$ALICE_PHYSICS/OADB/COMMON/
        MULTIPLICITY/macros/AddTaskMultSelection.C")); //macro de centralidad
38  AliMultSelectionTask* multSelectionTask = reinterpret_cast<AliMultSelectionTask*>(
        multSelection.Exec());
39  #if !defined (__CINT__) || defined (__CLING__)
40      gInterpreter->LoadMacro("AliAnalysisTaskMyTask.cxx++g");
41      AliAnalysisTaskMyTask *task = reinterpret_cast<AliAnalysisTaskMyTask*>(
        gInterpreter->ExecuteMacro("AddMyTask.C"));
42  #else
43      gROOT->LoadMacro("AliAnalysisTaskMyTask.cxx++g");
44      gROOT->LoadMacro("AddMyTask.C");
45      AliAnalysisTaskMyTask *task = AddMyTask();
46  #endif
47
48
49      if(!mgr->InitAnalysis()) return;
50      mgr->SetDebugLevel(2);
51      mgr->PrintStatus();
52      mgr->SetUseProgressBar(1, 25);
53
54      if(local) {
55          // if you want to run locally, we need to define some input
56          TChain* chain = new TChain("aodTree");
57          // add a few files to the chain (change this so that your local files are
        added)
58          chain->Add("AliAOD_MC.root"); // cambiamos el nombre del archivo
59          // start the analysis locally, reading the events from the tchain
60          mgr->StartAnalysis("local", chain);
61      } else {
62          // if we want to run on grid, we create and configure the plugin
63          AliAnalysisAlien *alienHandler = new AliAnalysisAlien();
64          // also specify the include (header) paths on grid
65          alienHandler->AddIncludePath("-I. -I$ROOTSYS/include -I$ALICE_ROOT -
        I$ALICE_ROOT/include -I$ALICE_PHYSICS/include");
66          // make sure your source files get copied to grid
67          alienHandler->SetAdditionalLibs("AliAnalysisTaskMyTask.cxx
        AliAnalysisTaskMyTask.h");
68          alienHandler->SetAnalysisSource("AliAnalysisTaskMyTask.cxx");
69          // select the aliphysics version. all other packages
70          // are LOADED AUTOMATICALLY!
71          alienHandler->SetAliPhysicsVersion("vAN-20181028_ROOT6-1");
72          // set the Alien API version
73          alienHandler->SetAPIVersion("V1.1x");
74          // select the input data
75          alienHandler->SetGridDataDir("/alice/data/2015/LHC15o");
76          alienHandler->SetDataPattern("*pass1/AOD194/*AliAOD_MC.root");   //cambiamos
         el nombre del archivo
77          // MC has no prefix, data has prefix 000
78          alienHandler->SetRunPrefix("000");
79          // runnumber
```

```
80          alienHandler->AddRunNumber(246994);
81          // number of files per subjob
82          alienHandler->SetSplitMaxInputFileNumber(40);
83          alienHandler->SetExecutable("myTask.sh");
84          // specify how many seconds your job may take
85          alienHandler->SetTTL(10000);
86          alienHandler->SetJDLName("myTask.jdl");
87
88          alienHandler->SetOutputToRunNo(kTRUE);
89          alienHandler->SetKeepLogs(kTRUE);
90          // merging: run with kTRUE to merge on grid
91          // after re-running the jobs in SetRunMode("terminate")
92          // (see below) mode, set SetMergeViaJDL(kFALSE)
93          // to collect final results
94          alienHandler->SetMaxMergeStages(1);
95          alienHandler->SetMergeViaJDL(kTRUE);
96
97          // define the output folders
98          alienHandler->SetGridWorkingDir("myWorkingDir");
99          alienHandler->SetGridOutputDir("myOutputDir");
100
101         // connect the alien plugin to the manager
102         mgr->SetGridHandler(alienHandler);
103         if(gridTest) {
104             // speficy on how many files you want to run
105             alienHandler->SetNtestFiles(1);
106             // and launch the analysis
107             alienHandler->SetRunMode("test");
108             mgr->StartAnalysis("grid");
109         } else {
110             // else launch the full grid analysis
111             alienHandler->SetRunMode("full");
112             mgr->StartAnalysis("grid");
113         }
114     }
115 }
```

Listing 3: Macro `runAnalysis.C`.

Los archivos de las gráficas y los códigos pueden encontrarse en el siguiente repositorio de Git: https://github.com/moiseslz/HEP-IP.

Referencias

[1] CERN.
Welcome to ALICE, a journey to the beginning of the Universe, recuperado el 09 de agosto de 2020.
http://aliceinfo.cern.ch/Public/Welcome.html.

[2] CERN.
ALICE detects quark-gluon plasma, a state of matter thought to have formed just after the big bang, recuperado el 09 de agosto de 2020.
https://home.cern/science/experiments/alice.

[3] CERN.
ALICE: New kid on the block (archive), recuperado el 09 de agosto de 2020.

https://cerncourier.com/a/alice-new-kid-on-the-block-archive/.

[4] Silvia Masciocchi.
Particle detection, recuperado el 09 de agosto de 2020.
https://www.physi.uni-heidelberg.de/~sma/teaching/ParticleDetectors2/sma_InteractionsWithMatter_1.pdf.

[5] Sebastián Rosado Navarro.
Ecuación de Bethe-Bloch, recuperado el 09 de agosto de 2020.
https://indico.nucleares.unam.mx/event/918/material/slides/1?contribId=9.

[6] Erika Garutti.
The Physics of Particle Detectors, recuperado el 07 de agosto de 2020.
https://www.desy.de/~garutti/LECTURES/ParticleDetectorSS12/L2_Interaction_radiationMatter.pdf.

[7] Anusha Ragunathan & Chris Crone Justin Cormack.
Why Docker?, recuperado el 10 de agosto de 2020.
https://www.docker.com/why-docker.

[8] Dario Berzano.
Home, recuperado el 10 de agosto de 2020.
https://github.com/alidock/alidock/wiki.