UNIVERSIDADE DE SANTIAGO
DE COMPOSTELA

FACULTAD DE FÍSICA

Departamento de Física de Partículas

Momentum Reconstruction and Pion Production Analysis in the HADES Spectrometer at GSI

Memoria presentada para
optar al Grado de Doctor
en Ciencias Físicas por:

Manuel Sánchez García

Abril 2003

Juan A. Garzón Heydt, Profesor Titular
de la **Universidad de Santiago de Compostela**,

CERTIFICA: que la memoria titulada MOMENTUM RECONSTRUCTION AND PION PRODUCTION ANALYSIS IN THE HADES SPECTROMETER AT GSI, ha sido realizada por **Manuel Sánchez García** bajo la dirección de los proferosres **Carmen Fernández Cambronero y Juan A. Garzón Heydt** en el **Departamento de Física de Partículas** de esta Universidad, y constituye la Tesis que presenta para optar al grado de **Doctor en Física**.

Santiago de Compostela, Abril de 2003

Prof. Juan A. Garzón Heydt **Manuel Sánchez García**

In memoriam

Carmen Fernández Cambronero

A mis padres

Acknowledgements
Agradecimientos

During the time span of this work I have found many people to whom I owe in some way or the other. All of them have influenced not only the work I am herein presenting, but me as a person too.

I would like to thank prof. Carmen Fernández Cambronero and Juan A. Garzón Heydt (Hans). They are the persons who gave me the opportunity to enter this world. Carmen's touch is evident in the design of HYDRA and Hans has been a reliable advisor.

Special thanks go to W. Koenig. The discussions in the office around a sheet of paper have been certainly enlightening. At least for the two of us, any other person looking to the paper after two hours of discussion would only see a seemengly random bunch of lines. It would be difficult to find a point in the thesis were his advise is directly or indirectly not present.

Long were also the debugging afternoons and nights I spent with my office mate and friend, Dennis Bertini. We started many projects, not all of them saw light though. I had the fortune to share office with another excellent person, Ilse Koenig, we had time to discuss about many different things.

Romain Holzmann has been the head of the analysis group for years and as such he has done a superb job in keeping it together and maintaining the synergy. Nothing could have been done without those conditions. He has also been the person who suggested me to start working on momentum reconstruction and has provided advice when requested ever since.

I would also like to thank J. Díaz for his thorough revision of the pion analysis chapter in this writing.

During this time I had the opportunity to know many different people, the list is too long. It is them who made this experience one to cherish. I would like to thank Antonio, I could possibly not have survived the first month in Germany without him and Hector, a long term companion.

There have been nice and not so nice periods during these years, in all of them I could count on my family. They have been my greatest support ever. Thanks go to Toño for keeping me in the track all this time with his acid critics. Finally, there has been one person who has recently incorporated to my life and has been pushing me forward ever since. Thanks Yoli.

Contents

Chapter 1

Resumen/Summary

El presente trabajo se enmarca dentro de la colaboracion internacional HADES. Ésta es una colaboración de mas de 100 investigadores en la que participan 18 laboratorios de 8 paises europeos. La sede central está en el instituto GSI (Gesselschaft für Schwerionenforschung) en Darmstadt, Alemania. Es allí donde se dispone del acelerador de partículas y del espectrómetro necesarios para el experimento.

Las tareas necesarias para llevar HADES a buen puerto son dispares. En este documento se trata fundamentalmente del "software" de reconstrucción de momentos. Aún siendo esa la parte central, el documento abarca desde la creación del marco o estructura ("framework") de análisis común al experimento, hasta la aplicación de los algoritmos de reconstrucción de momento al análisis de producción de piones.

Lo aquí presentado representa tan sólo una fracción de todo el trabajo realizado dentro del grupo de análisis en HADES, y éste es a su vez una fracción del realizado por la colaboración. El objectivo es combinar esfuerzos de cara a producir un incremento del conocimiento científico.

1.1 El experimento HADES

El espectrómetro HADES (High Acceptance Di-Electron Spectrometer) está instalado en el GSI. El GSI es un instituto alemán dedicado entre otras cosas a la física de iones pesados. Para ello cuentan con un acelerador capaz de alcanzar energias entre 1 y 2 GeV por nucleón para iones pesados, como el uranio, o ligeros, como el neón, respectivamente.

El principal objectivo de HADES es la contribución al conocimiento de las propiedades no perturbativas de QCD, a través del estudio de modificaciones en las propiedades de la materia hadrónica en un medio denso y caliente. Un medio de esas características es el que se forma en las colisiones entre iones pesados. Se puede hablar de distintas fases en una colisión de este tipo. En una primera fase, los iones al colisionar dan lugar a una región de alta densidad (hasta 3 veces la densidad nuclear normal) y temperatura (en torno a 80MeV). Esta fase dura un período de tiempo muy corto (unos 10 fm/c). Tras ella sobreviene una expansión, debida a la presión acumulada, durante la cual se suceden las interacciónes inelásticas entre nucleones. Estas interacciones dan lugar a la formación de resonancias y estados excitados que decaen emitiendo nuevas partículas (fundamentalmente piones). Al cabo de un cierto tiempo, los nucleones estan separados y dejan de producirse reacciones inelásticas, de modo que la composición hadrónica permanece estable; a esta fase se la llama hadronización o congelación.

Para estudiar la fase de alta densidad necesitamos sondas que sean sensibles a lo que allí ocurre y que no se vean afectadas por la multitud de procesos que tienen lugar en las fases finales de la colisión, de modo que en el estado final conserven memoria de la fase densa. Unas sondas que verifican estos requisitos son los mesones vectoriales: ρ, ω y ϕ. Éstos tienen tres características fundamentales:

- Una vida media corta, del orden o menor que la duración de la fase de alta densidad en la colisión. Esta característica posibilita el hecho de que una de estas resonancias decaiga en la fase densa, transcurriendo toda su vida en dicha fase.

- Un canal de desintegración en e^+e^-. Cuando uno de los mesones vectoriales decae en la zona de alta densidad, puede hacerlo en un par e^+e^-. Estas partículas, al ser leptones, no interaccionan por vía fuerte con el resto de participantes. De esta manera, en el estado final, mantienen memoria de las condiciones en que fueron producidas. A través de un análisis de masa invariante de los leptones en el estado final es posible reconstruir la masa del mesón originario.

- Sensibilidad de las propiedades básicas de los mesones a la densidad del medio en que fueron producidos. Se espera que a altas densidades se produzca una restauración de simetría quiral en QCD, que llevaria a la alteración de las masas de los hadrones y por ende también de los mesones vectoriales.

En definitiva, la técnica consiste en la identificación de pares e^+e^- provenientes de desintegraciones de mesones vectoriales y el cómputo de sus masas invariantes. Las dificultades para la implementación de esta idea son variadas y condicionan el diseño del espectrómetro:

- Se precisa de una resolución en masa invariante del orden del 1% en la región del mesón ω, de modo que se pueda aislar su señal sobre el fondo producido por otras fuentes. Para alcanzar dicha resolución, se necesita un espectrómetro magnético de modo que se pueda medir el momento a través de la deflexión en el campo. Para alcanzar altas resoluciones se necesita de un elevado campo o alta precisión en la medida de la dirección de las partículas antes y después del mismo. En el caso de HADES, dado el efecto de multiple scattering, es recomendable reducir también el tamaño del detector. Asimismo, la necesidad de una alta aceptancia geométrica implica la no utilización de campos excesivos.
  En consecuencia, el espectrómetro magnético de HADES consta de un imán superconductor que crea un campo toroidal compacto antes y después del cual se colocan sendos pares de detectores de deriva (MDC), responsables de la medida de la direccion de la partícula antes y después de la deflexión.

- Los leptones no son las únicas partículas producidas, habiendo también un mar de hadrones y leptones procedientes de otras fuentes como π-Dalitz. Por esa razón, HADES cuenta con detectores especializados en rechazar el fondo hadrónico: un detector tipo Cherenkov de umbral en la parte interior, así como detectores de tiempo de vuelo y Pre-Shower en la parte posterior (detrás del imán).
  El detector Cherenkov se basa en la producción de luz al pasar partículas cargadas por un medio con una velocidad superior a la de la luz en ese medio. Como quiera que los leptones son partículas ligeras y por tanto rápidas en comparación con los hadrones, es posible distinguirlos en base a la producción o no de luz Cherenkov, escogiendo debidamente el radiador.
  Los detectores de tiempo de vuelo permiten identificar leptones mediante una medida directa de su velocidad.
  Por su parte el Pre-Shower analiza el tipo de cascada producida por cada partícula al interaccionar con la materia, distinguiendo los leptones por generar cascadas de tipo electromagnético.

- La baja relación de desintegración (o “branching ratio”) ($\sim 10^{-5}$) del canal de desintegración de leptones trae consigo varias consecuencias sobre el diseño del espectrómetro, si es que se quiere obtener datos estadísticamente significativos en una cantidad razonable de tiempo. Así, es necesario disponer de una gran aceptancia para maximizar la probabilidad de detectar los pares producidos. Es también necesario operar con altas tasas de contaje en los detectores de modo que la intensidad del haz primario se pueda elevar, hasta los $10^8 sucesos/s$, y con ella el numero de interacciones por segundo. Una consecuencia derivada de las altas intensidades de operación es la necesidad de un esquema de "trigger" o disparo, capaz de seleccionar sucesos con candidatos a pares de leptones, de modo que se reduzca la cantidad bruta de datos a tratar por varios órdenes de magnitud. El esquema de HADES se basa en tres niveles:

– En un primer paso se seleccionan colisiones centrales. Ésto se hace poniendo un corte en la multiplicidad mínima de partículas cargadas registradas en los detectores de tiempo de vuelo, los cuales cubren toda la aceptancia. Con esto se consigue un factor de reducción 10.

– En un segundo paso se forman candidatos a dileptones usando conjuntamente la información de los detectores Cherenkov, tiempo de vuelo y Pre-Shower. Se requiere una identificación positiva como leptones por esos detectores, así como que el par formado tenga un ángulo de apertura mínimo y su masa invariante caiga en un cierto rango. A través de este procedimiento se consiguen factores de reduccion del orden de 100.

– En el tercer paso, aún no implementado, se suma la información preliminar de los detectores MDC para mejorar el cómputo de la masa invariante de los candidatos a dileptones formados en el paso anterior. Refinando así el corte en masa invariante. El factor de reducción adicional es del orden de 10.

• Para la obtención de secciones eficaces absolutas se necesita normalizar las medidas a la intensidad del haz. Para ello se cuenta con un detector de diamante colocado antes del blanco en la linea de haz. Este detector cumple la doble función de marcar el tiempo de inicio para los detectores de tiempo de vuelo, así como la de contaje del numero de partículas incidente. Por ello debe ser capaz de tolerar tasas de adquisición muy altas ($\sim 10^8 s^{-1}$)

1.2 Software de reconstrucción

La función de los detectores es la de medir características de las partículas producidas en cada suceso que tenga lugar. Ésto lo hacen recogiendo gran cantidad de señales electrónicas por suceso. Así pues, aparte de dichos detectores, es necesaria la existencia de un programa de reconstrucción capaz de interpretar las señales electrónicas y extraer a partir de ellas información contenido físico de más alto nivel. El nombre HYDRA se utiliza en HADES tanto para significar dicho programa de reconstrucción así como el "framework" en el que se basa; aqui trataremos del "framework".

HYDRA está implementado en C++ y basado en el paradigma de programación orientada a objetos. Dos características fundamentales de este paradigma son la encapsulación y el polimorfismo. La encapsulación hace referencia a que el código se divida en entidades cerradas con sentido semántico pleno (clases) que interaccionan con su ecosistema a través de interfaces bien definidos. La encapsulación es pieza clave para diseñar un programa modular, característica necesaria cuando sobre él van a trabajar distintos grupos de programadores, posiblemente separados geográficamente. En esas condiciones la modularidad, si no es excesiva, reduce el nivel de interdependencias entre elementos del código y por tanto su complejidad. Una menor complejidad lleva consigo un menor coste de mantenimiento y mayor facilidad para la evolución del código.

Otra característica fundamental es el polimorfismo; a éste se llega a traves de la "herencia". Decimos que una clase hija hereda de otra madre cuando implementa todas las funciones de la madre y contiene sus mismos miembros de datos. La herencia permite la representación de la jerarquía "es un". La clase hija puede extender a la clase base con nuevas funciones y propiedades, pero tambien puede redefinir el comportamiento de las funciones de la clase base. Dado que el interfaz definido en la clase base es válido también para la hija, cualquier otro objeto puede hacer uso de ese interfaz sin necesidad de saber si trata con la clase madre o la hija, pero obteniendo comportamientos distintos en ambos casos. A esto se le llama polimorfismo.

El polimorfismo es una herramienta fundamental a la hora de diseñar un "framework". La diferencia entre un "framework" y una librería es que en el segundo caso es el usuario de la librería el que controla el flujo del programa. En el caso del "framework", el usuario extiende un diseño básico proporcionando nuevas funcionalidades, ésto es hecho a través del polimorfismo.

HYDRA es un "framework" para el procesado de sucesos registrados en un espectrómetro. En ese sentido, distingue entre los datos a procesar y los algoritmos necesarios para procesarlos, definiendo interfaces específicas para ambos casos.

Todos los algoritmos derivan de una clase base: HTask. Ésta se refiere a un único algoritmo. Como clases hijas se proporcionan contenedores capaces de albergar secuencias y grafos de algoritmos. El conjunto de estas clases forma uno de los subsistemas de Hydra.

Otro subsistemas son el encargado de gestionar los parámetros de los distintos algoritmos, el encargado de leer y guardar los datos de sucesos de forma persistente, el encargado de proporcionar acceso a datos en tiempo de ejecución etc.

Todos ellos forman un esqueleto puesto al servicio de los desarrolladores de algoritmos para que éstos, al extender el “framework”, lo conviertan en un programa completo de reconstrucción.

1.3 Reconstrucción del vértice

Uno de los algoritmos que corren dentro de Hydra es el de reconstrucción de vértice. El objectivo de este algoritmo es obtener la posición espacial del vértice de interacción en una colisión. Para ello se cuenta con la información de la trayectoria de las trazas, registradas por las Mdc, que salen del punto de interacción.

En primera aproximación podemos suponer que las trayectorias antes del imán son líneas rectas. En este caso el vértice será un punto común a todas ellas. Como quiera que la resolución finita de los detectores hace que las trazas no se corten en un único punto, es necesario redefinir el vértice como el punto de máxima aproximación al conjunto de trazas. Para obtener dicho punto se puede minimizar una funcional definida como la suma de las distancias al cuadrado a cada una de las rectas, dando así lugar a un estimador para el vértice del tipo de Mínimos Cuadrados o “Least Squares Method” (Lsm).

El mínimo de Lsm así definido se puede encontrar de forma analítica. Sin embargo, en la anterior definición se ha obviado el efecto de la resolución en la determinación de las trazas. Cuando esa resolución se traslada a la cantidad de información que cada traza da sobre la posición del vértice ocurre que no todas son iguales. Así por ejemplo, una traza paralela al eje z no contiene ninguna información sobre la coordenada z del vértice, aunque sí sobre x e y. Para tener ésto en cuenta se introducen pesos en la funcional haciendo que deje de ser lineal en las coordenadas del vértice y por tanto que deje de ser resoluble analíticamente.

También es necesario añadir pesos extra para compensar el efecto de los "puntos lejanos". Una característica común a los métodos basados en LSM es que la presencia de puntos fuera de la tendencia común tiene efectos muy negativos sobre la estimación final de parámetros. Esta situación se puede mejorar añadiendo pesos adicionales que limiten la contribución al funcional de aquellos puntos. Los pesos introducidos son de nuevo no lineales en las coordenadas del vértice.

Dadas las no linealidades, se impone la necesidad de un algoritmo de minimización numérica. En este caso el algoritmo es iterativo y consiste en utilizar las coordenadas del vértice cada iteración para calcular los pesos en la iteración posterior. El algoritmo converge típicamente al cabo de 10 iteraciones.

1.4 Reconstrucción de momentos

Un conjunto de algoritmos de gran importancia para HADES son los de determinación de momento. Se trata de un conjunto de ellos, y no de uno sólo porque existen varios, desarrollados respondiendo a diferentes necesidades del espectrómetro; necesidades éstas derivadas de la voluntad de tomar y analizar datos aún cuando no todos los detectores están disponibles. Cada algoritmo tiene ventajas e inconvenientes.

Basicamente hay dos algoritmos, aplicados a tres configuraciones de detectores. Los algoritmos son "Kick Plane" y "Trayectorias de Referencia". Las tres configuraciones consideradas corresponden al numero de cámaras MDC disponibles en cada caso. Cada cámara añadida supone un salto en la resolución alcanzable así como en el nivel de redundancia en los datos. El sistema completo consta de 4 cámaras, 2 delante y 2 detrás del imán. Las dos cámaras antes del imán están siempre disponibles mientras que detrás del imán puede haber dos, una o ninguna.

El algoritmo de kick plane se basa en la idea de que la deflexión de una traza en un campo magnético es proporcional a la integral de camino del campo a lo largo del recorrido de la traza. Siendo esto asi, se puede sustituir la integral por el producto de un campo promedio y la distancia total recorrida. Una reducción en la distancia recorrida se puede compensar con un aumento en el campo promedio. En el límite en que la distancia tiende a cero toda la deflexión ocurre en una superficie. De este modo podemos modelar la traza como dos lineas rectas que se cruzan en un punto sobre la superficie de deflexión. La relación entre deflexión y momento vendra entonces determinada por constantes que solo dependen de ese punto. Así pues es posible tabular dichas constantes en base a un Monte Carlo, para luego ser usadas a posteriori.

El algoritmo de trayectorias de referencia se basa en la utilización de un Monte Carlo para construir una tabla de todas las trazas posibles. Para cada traza en la tabla se registran sus puntos de intersección con cada uno de los detectores y se almacenan. Cuando se necesita calcular el momento para una traza real basta con buscar la traza más parecida en la tabla y realizar una interpolación lineal teniendo en cuenta a los vecinos más proximos. Éste es el método que se utiliza para conseguier la máxima resolución.

El método del Kick Plane puede aplicarse a todos los escenarios, pero es especialmente indicado para aquellos en los que parte de las MDC están ausentes, ya que el conocimiento de la superficie de deflexión sirve como apoyo para el emparejamiento de trazas.

Cuando no hay ninguna cámara MDC detrás del imán la redundancia disponible es baja. En ese caso se utiliza los detectores más externos (tiempo de vuelo y Pre-Shower) para obtener un punto sobre la traza tras el campo. Mientras en la parte interior se conoce una linea recta, que es la de entrada en el campo. Entonces el método consiste en extrapolar esa recta hasta la superficie de deflexión y anotar el punto de intersección. La unión de dicho punto con el proporcionado por los detectores externos proporciona la recta correspondiente a la traza detrás del imán. Conocidas ambas trazas se puede medir la deflexión y, por ende, el momento.

Como quiera que el campo es toroidal y la deflexión ocurre fundamentalmente en el ángulo polar, tan sólo queda una variable redundante: la diferencia entre el azimut de las trazas antes y después del imán. A primer orden esa diferencia debe ser cero, permitiendo el emparejamiento de trazas al proporcionar un criterio que distingue las combinaciones buenas de las que no lo son.

Cuando hay una o dos cámaras externas, se dispone de mas variables redundantes para usar en el emparejamiento de trazas. En total 3: la deflexión azimutal y dos variables más dadas por la medida de la dirección de la traza en las MDC externas.

1.5 Producción de piones

El presente trabajo no estaría completo sin la aplicación de los algoritmos anteriormente presentados al análisis de datos reales.

Se ha estudiado la producción de piones en colisiones $^{12}C + ^{12}C$ a 2AGeV utilizando datos tomados en Noviembre del 2001.

En el primer estudio se demuestran las capacidades de identificación de partículas de HADES, en concreto de hadrones. La idea básica es que disponemos del momento de las partículas y también de su tiempo de vuelo así como de la distancia recorrida. Con esa información es posible calcular la velocidad de cada partícula y representar su momento frente a la velocidad. Dado que ambas variables estan relacionadas por la masa en reposo de la partícula y que ésta es distinta para cada tipo de ellas, en la representación se observan curvas correspondientes a los distintas especies: piones negativos y positivos, protones etc.

Lo siguiente es mirar las razones de producción entre las distintas especies. Así esperamos el mismo número de piones positivos que negativos, con las devidas correcciones por aceptancia. También se espera, en base a medidas anteriores, que el número total de piones esté en torno al 10% del número de protones. Ambos resultados son reproducidos en HADES.

Un resultado de mas alto nivel es el espectro de momento transverso de piones negativos. Se espera que este responda básicamente a la superposición de dos distrubuciones térmicas (de Boltzmann). Se usan los piones negativos porque son más fáciles de identificar. Es de más alto nivel porque requiere una serie de correcciones para poder ser comparado con la información en la literatura

- Se necesita una correción por perdida de energía. Los distintos métodos de reconstrucción de momento estiman el momento que las partículas tienen al entrar en el sistema MDC. Sin embargo, hasta llegar a ese punto una partícula ha podido perder energía por ionización en el blanco o en el detector Cherenkov que se encuentra por el camino. Esta perdida de energía viene bien reproducida por la fórmula de Bethe-Bloch y conlleva una pérdida de momento, con lo que la estimación del Kick Plane o Trayectorias de Referencia presenta un error sistemático por defecto que debe ser corregido.

- Se precisa una corrección por aceptancia del espectrómetro

- Se necesita de una estimación de la eficiencia y nivel de ruido introducidos por los algoritmos de emparejamiento de trazas. Ésto es debido a que, en algoritmos como el Kick Plane, las trazas de ruido tienden a acumularse en la zona del espectro de bajo momento, desvirtuando las medidas.

Al cabo de las distintas correcciones se obtienen resultados comparables con los de anteriores experimentos.

Chapter 2

The HADES experiment

Investigation on the properties of nuclear matter in extreme conditions (high temperature and densities) is essential to the understanding of processes like, for example, those giving birth to the Universe in the Big Bang and its later evolution, since at those moments the medium was one of high temperature and density. To some extent we can find similar conditions nowadays inside dense neutron starts. This line of investigation contributes also to obtain the equation of state of nuclear matter which is not only important to Nuclear Physics, but also to understand physics processes taking place during the latest stages of stars evolution

HADES (6; 5), together with other experiments at GSI[1], CERN[2] and BNL[3], contributes to that line of investigation. The focus of HADES is the study of in medium modifications to the properties of vector mesons. Calculations based on QCD, and some hadronic models, predict detectable changes in the width and mass of hadrons produced in a dense nuclear medium. From the point of view of QCD such modifications could be a signal of the so called chiral symmetry restoration, which is a non perturbative phenomenon. HADES' main goal is to provide experimental insight for the study of QCD on the non perturbative regime and possibly see a signal of the expected chiral symmetry restoration.

[1]Gessellschaft für SchwerIonen forschung
[2]Centre Europee de Recherche Nucleaire
[3]Brookhaven National Laboratory

2.1 The HADES physics

When it comes to observing the characteristics of vector mesons in dense nuclear matter several techniques have been used. The basic idea is to produce them in heavy ion collisions and then analyze the different variables in the collision's final state. The different particle species: pions, kaons, etc. have been studied. HADES is focused on the study of lepton pairs produced in the decays of vector mesons inside the hot, dense medium. However, the spectrometer is also able to detect and study the properties of hadrons.

During the initial stage of a heavy ion collision at 2 AGeV, a compression phase is created where density reaches values of up to 3 times that of normal nuclear matter. This compression phase lasts for about 10 fm/c and is followed by an expansion phase. During the expansion, meson scattering, absorption and reemission equilibrate the various hadronic species. After a few tens of fm/c the expansion makes inelastic reactions between constituents impossible, hence the hadrochemical composition is frozen. This is the so called freeze out point.

What we want to study is the high density phase. For that purpose light vector mesons are a well suited probe. Their lifetimes (see table 2.1) are short enough for them to have a significant chance of decaying in the same hot and dense medium were they are created. When they decay, they may do so in two leptons. Since leptons do not experiment *strong* interaction, when the leave the interaction zone they retain memory about how they were produced. Hence they carry information about the properties of the vector mesons in the dense medium. Whether if their masses or widths have changed due to a partial restoration of chiral symmetry we should be able to tell by looking at the lepton pairs' invariant masses.

The problem is the low branching ratio for the dilepton channel in the vector meson decays. This needs to be compensated with high statistics, which translate into a need for a high acceptance spectrometer and high beam intensities. Another problem is the presence of several background sources, like pion Dalitz decays, which also produce leptons (and lepton pairs) so that the vector meson signal is sitting on top of a continuous background.

Vector mesons are hadrons of spin 1, either isoscalars (isospin 0) or isovectors (isospin 1). They are composed of a quark and an anti-quark. The neutral component of the ρ triplet and ω meson carry the same quantum numbers of the photon, which allows their decay in dileptons through virtual photon. The ϕ meson is made of strange quarks, this manifests in the presence of kaons in its decay channels. If we look to the mean life of these mesons it seems the best candidate to be used as a probe is the ρ meson since

Meson	Mass $(\frac{MeV}{c^2})$	Width $(\frac{MeV}{c^2})$	$c\tau$ (fm)	Dominant channel	e^+e^- branching ratio
ρ	768	152	1.3	$\pi\pi$	4.4×10^{-5}
ω	782	8.43	23.4	$\pi^+\pi^-\pi^0$	7.2×10^{-5}
ϕ	1019	4.43	44.4	K^+K^-	3.1×10^{-4}

Table 2.1: Light vector mesons life times. Data from (1)

it has a large chance of decaying in the dense zone. However, it has a very large width, complicating its recognition on top of the background. On the other hand the ω meson has a smaller chance of decaying in the interaction zone, but it is a well defined resonance. This is also true for the ϕ meson, but its extended lifetime makes it less likely to decay in the high density phase. Nevertheless, all three of them contribute and will be studied.

2.1.1 Previous experiments

Other experiments have made similar studies at higher energies, like HELIOS1-3(8) or more recently CERES(10; 11; 12) at CERN. In particular CERES finds an excess in the lepton pair production for intermediate invariant masses from 0.2 to 1.5 GeV (see Fig. 2.1) which cannot be described by the superposition of the conventional hadronic sources. This excess is only observed in heavy ion systems, like Sulfur on Gold, but not in lighter systems like proton on Beryllium or proton on Gold.

In the energy regime of HADES, the only existent data are from the original DLS (13; 14; 15; 16; 17) experiment at BEVALAC, Berkeley. The DLS data came in two generations. The first generation was well reproduced in transport model calculations by an incoherent sum of dilepton yields from pn bremsstrahlung and free hadron decays. On a second generation (Fig. 2.2) with more statistics and a refined analysis, an excess of up to a factor 7 was found in the intermediate mass region. The problem with DLS was, however, the limited mass resolution and acceptance (0.5-1%) which translated in rather low statistics.

HADES aims to improve on the resolution and statistics from previous experiments to be able to put the different models to test.

2.1.2 Theoretical frame

When it comes to explanations of the data, several models are available (see (3; 4) for a couple of reviews).

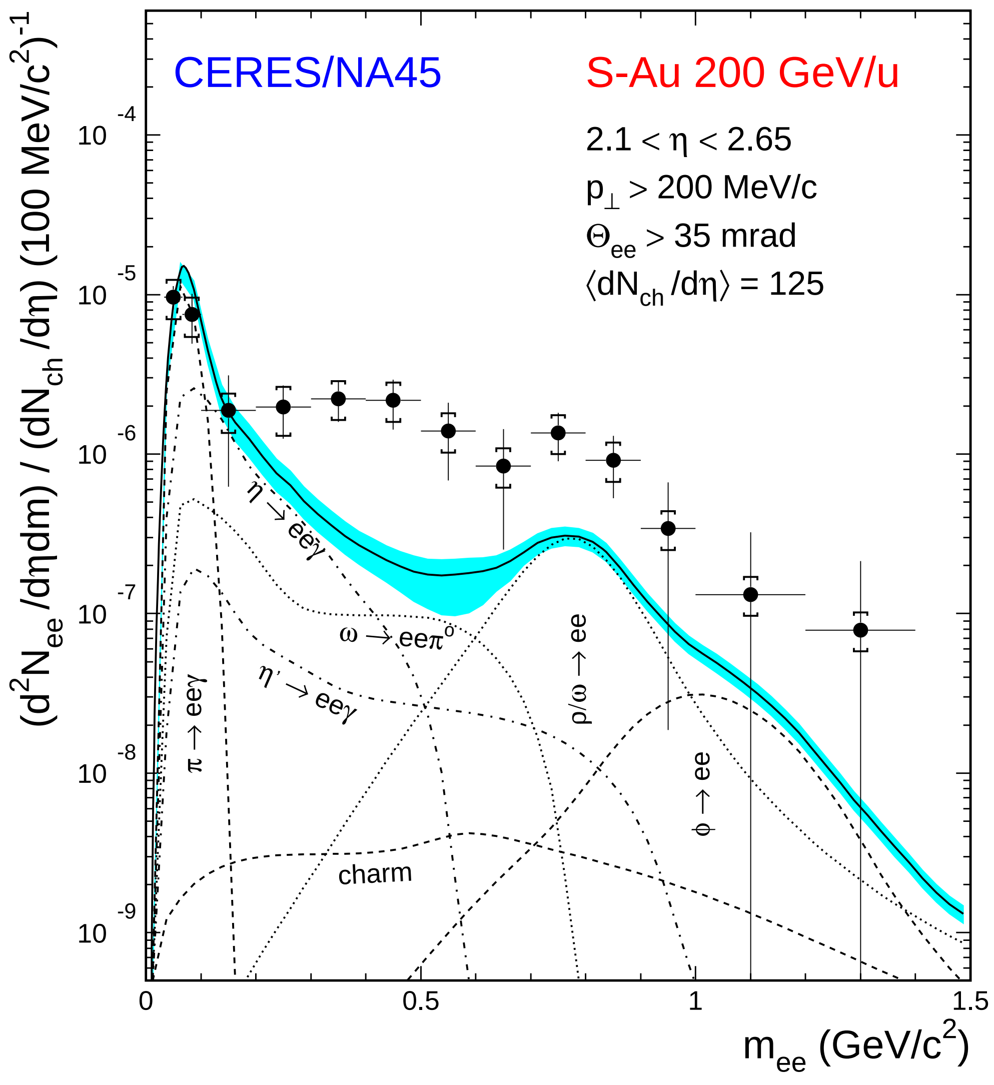


Figure 2.1: CERES results: the data points correspond to the experimental data, including the statistical error bars and the systematical errors (brackets). The individual contributions to the theoretical predictions are quoted in the graph. the dilepton enhancement is apparent for intermediate masses. Taken from (10)

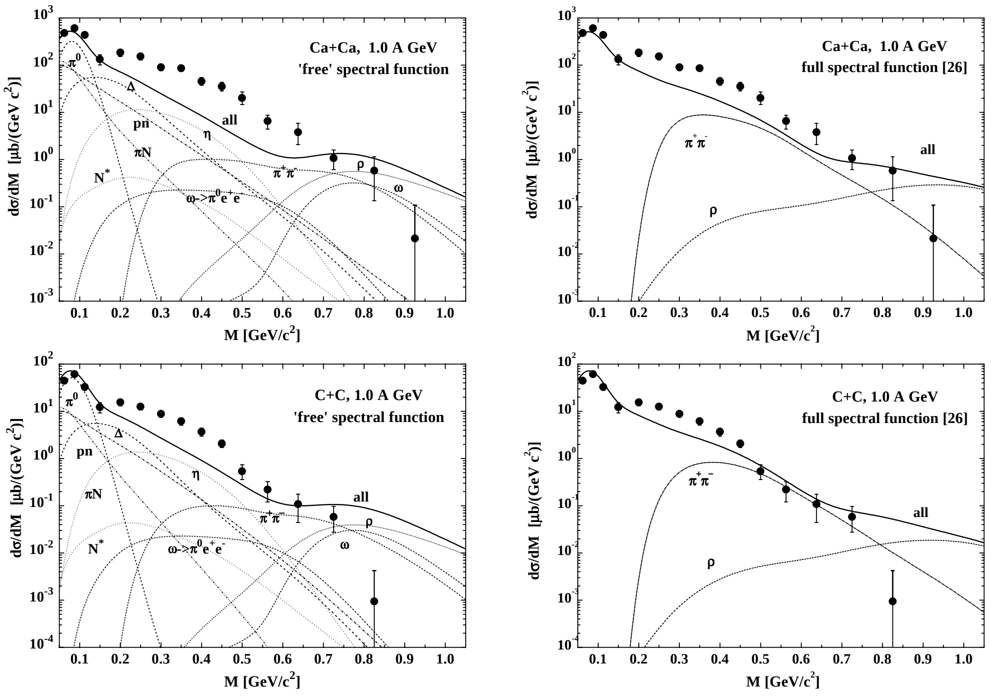


Figure 2.2: Comparison between the differential cross section for dilepton production in DLS and the BUU transport calculation. In the left part, the ρ free spectral function is used. In the right part, the full in medium spectral function of ρ is used in the calculation of the decay yields and $\pi^+\pi^-$ annihilation. The thin lines indicate the individual contributions from the different production channels: starting from low masses, Dalitz decay $\pi^0 \rightarrow \gamma e^+e^-$ (dashed line), $\eta \rightarrow \gamma e^+e^-$ (dotted line), $\Delta \rightarrow Ne^+e^-$ (dashed line), $\omega \rightarrow \pi^0 e^+e^-$ (dot-dashed line), $N^+ \rightarrow Ne^+e^-$ (dotted line), proton-neutron bremsstrahlung (dot-dashed line), πN bremsstrahlung (dot-dot-dashed line). Around 0.8 GeV: $\rho^0 \rightarrow e^+e^-$ (sashed line), $\omega \rightarrow e^+e^-$ (dot-dashed line) and $\pi^+\pi^- \rightarrow \rho^0 \rightarrow e^+e^-$ (dot dashed line). Taken from (9).

From the point of view of QCD the variation in the meson's mass and width may be due to a partial restoration of chiral symmetry at high baryonic densities. The QCD Lagrangian with (u,d,s) quarks can be divided in two parts:

$$\mathcal{L}_{QCD} = \mathcal{L}^0_{QCD} + \Delta\mathcal{L}_{mass}$$

with

$$\begin{aligned} \mathcal{L}^0_{QCD} &= \bar{q} i \gamma_\mu (\partial^\mu - igG^\mu) q + \frac{1}{4} F_{\mu\nu} \\ \Delta\mathcal{L}_{mass} &= -\bar{u} m_u u - \bar{d} m_d d - \bar{s} m_s s \end{aligned}$$

where G_μ are the gluon fields, m_i the quark masses and $q = (u, d, s)$. Let's consider the projector

$$q_{R,L} = \frac{1}{2}(1 \pm \gamma_5) q$$

It separates the mass-less quark field into left and right components which remain uncoupled under the gluon field interaction (as long as they are massless). Then $\mathcal{L}^0_{QCD}$ is invariant under the chiral flavor group $SU(3)_R \times SU(3)_L$. The $\Delta\mathcal{L}_{mass}$ term breaks chiral symmetry explicitly. All the terms in $\Delta\mathcal{L}_{mass}$, the part breaking chiral symmetry, contain a mixing term of the type

$$\bar{q}q = \bar{q}_L q_R + \bar{q}_R q_L$$

Hence a restored chiral symmetry can be characterized by a $\langle \bar{q}q \rangle_0 = 0$. In other words, the spontaneously broken chiral symmetry leads to a finite quark condensate in vacuum:

$$\langle \bar{q}q \rangle_0 \simeq -(245 MeV)^3$$

Lattice QCD calculations, at 0 chemical potential, show a dropping quark condensate in vacuum for increasing temperature as well as in Chiral Perturbation Theory computations and other models. The reduction in the quark condensate may lead of a reduction in the meson's mass in the nuclear medium, as seen in studies based on QCD sum rules.

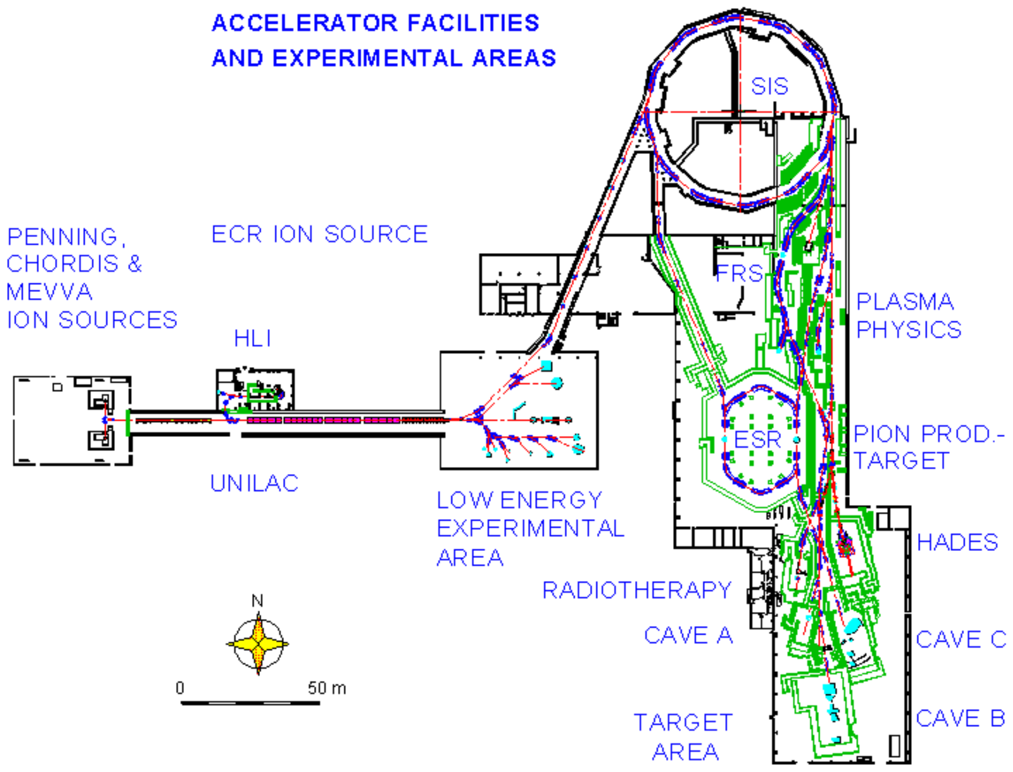


Figure 2.3: GSI's accelerator complex

2.2 The accelerator

The accelerator machine providing the beam for HADES is located at the GSI[4] Institute in Darmstadt, Germany.

The accelerator complex consists of 4 major structures : a linear accelerator (UNILAC) injecting ions into a 60 meter diameter Synchrotron (SIS). From there the beam can be extracted to the FRagment Separator (FRS), to the Electron Storage Rings (ESR) or the experimental areas.

The UNILAC was constructed in 1975 as a Wideroe-Alvarez linear accelerator and was recently upgraded (in 1999) to become a high current injector for the SIS. The new High Current Injector (HSI) provides an increase in the beam intensities filling the synchrotron up to its space charge limit for all ions. Two ion sources feed the HSI. After stripping and charge state separation, the beam from the HSI is matched to the Alvarez accelerator, which accelerates the highly space charge dominated ion beams without any significant particle loss, up to a few AMeV.

The SIS is a synchrotron with a circumference of 216 meters consisting of 24 curvature magnets and 36 magnetic lenses. Before entering the SIS, ions from the UNILAC interact with a carbon foil achieving ionization states up to 72^+ for Uranium. In those cases SIS allows energies up to 1GeV per nucleon, while for light systems (like Neon) it is possible to achieve full ionization and consequently increase the energy up to the 2 GeV per nucleon. The acceleration takes place in two resonance cavities diametrically opposed, where ions see a potential of 15 kV. The operation frequency ranges from 800KHz to 5.6MHz. The vacuum in the beam line is lower than 10^{-11} Torr.

2.3 The HADES spectrometer

HADES is a second generation experiment in high resolution dilepton spectroscopy. It intends to precisely measure the invariant mass of lepton pairs produced by the decay of vector mesons in heavy ion collisions. That goal puts a number of requirements on the design of HADES, shaping it.

This sections starts detailing what those requirements are and how HADES deals with them. Then, a quick overview of the spectrometer will be offered before entering into a more detailed description of the different sub detectors.

[4] Gessellschaft für SchwerIonen forschung

2.3.1 Requirements

The probability of producing a lepton pair as a consequence of a vector meson decay in a heavy ion collision is rather low. This is due to the fact that the branching ratio of the dileptonic channel is around 10^{-5}. As a consequence, in order to accumulate significant statistics in a reasonable time, HADES needs to have some characteristics:

- *Large acceptance* in order to maximize the probability of detecting a pair once it is produced. The acceptance of HADES is $\epsilon_{pair} = 40\%$
- *High count rates* need to be supported. The goal is to be able to operate with beam intensities as high as 10^8 particles per second. With a 1% interaction length target, that yields 10^6 minimum bias events, or 10^5 central ones.
- *A trigger* system able to downscale the amount of raw data by several orders of magnitude. The trigger scheme in HADES is made of three stages, ideally the joint rejection power would be in the order of 10^4. That is not the case yet, though.

Recording the data is not the whole thing. The ability to reconstruct the invariant mass of the recorded dilepton pairs with enough resolution is also critical. The design goal for HADES is to be able to achieve an *invariant mass resolution* in the order of the ω width in the mass region of the ω, so that it can be resolved. That yields $\frac{\Delta M}{M} \simeq 1\%$. As a consequence, a magnetic spectrometer is needed with high position resolution before and after the magnet field. Furthermore, at the operating energy range (1-2 AGeV) multiple scattering may adversely affect position resolution and therefore momentum resolution too. This imposes the usage of *low mass materials* in the different detectors.

The need to measure heavy systems imposes a relatively *high granularity* on the detectors, so that it is possible to deal with particle multiplicities as high as 120 per event.

Lepton pairs are not the only thing produced in a heavy ion collision; both hadrons and leptons from other sources, like pion Dalitz, are produced. As a result, the dilepton signal is embedded in a soup of particles. For that reason the spectrometer must contemplate systems providing rejection of hadronic and electromagnetic background.

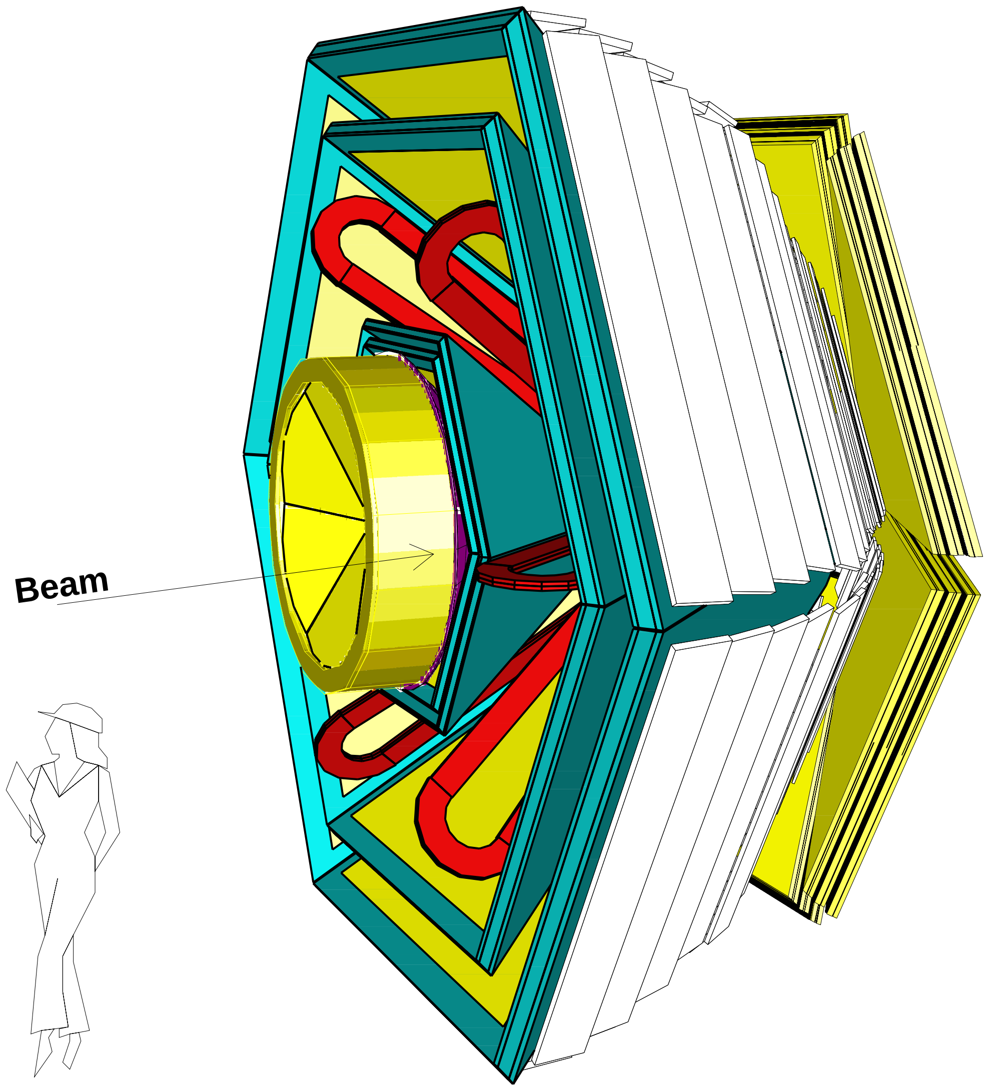


Figure 2.4: 3D view of the HADES detector. The hexagonal symmetry is apparent

2.3.2 Overview of the spectrometer

Fig. 2.4 shows an schematic view of the HADES spectrometer. It shows how the spectrometer is divided azimuthally in six identical sectors, each covering polar angles $18^o < \theta < 85^o$, with practically full azimuthal coverage, besides the shadow regions introduced by the coils and detector frames. This gives an acceptance for lepton pairs of 40%(1). The detector systems in HADES are, from inner to outer:

- A Ring Imaging CHerenkov (RICH) threshold detector, placed around the segmented target. This detector provides lepton identification, being unresponsive to fast hadrons (pions up to 3GeV).
- A magnetic spectrometer composed of two sets of Multi-wire Drift Chambers (MDCs) separated by a superconducting magnet made up of six coils shaping a toroidal field embedded in the space left between the chambers. The drift chambers determine the track's direction and position before and after the magnet and the magnet provides the deflection between them. The magnet field is inhomogeneous with a momentum kick ranging from 40 to 120 MeV. The larger momentum kick is located in the lower polar angles, which corresponds to the larger particle momenta due to the Lorentz boost. This map is intended to provide bending power, which improves momentum determination resolution, without neglecting the need for a large momentum acceptance.
- Lepton identification in the outer part of the spectrometer is performed by
  - A Time Of Flight (TOF) plastic scintillator wall made of thin scintillator strips for large polar angles. For polar angles below 45^o four plastic scintillator (TOFINO) cover the whole area providing the time of flight determination in low multiplicity reactions. TOF is able to discriminate electrons from pions and protons up to 500MeV and 2GeV respectively.
  - For the lower polar angles, the higher particle's momenta reduce the effectiveness of the lepton discrimination based only on time of flight. For that reason, the SHOWER detector is needed in that region. It is made of three gaseous chamber separated by lead converters, even though it is not large enough to be considered a calorimeter. The detector is able to register the first steps in the shower produced by a particle crossing it. From the multiplication

factors on those first steps, the SHOWER tries to separate leptons from hadrons. That is, particles giving raise to an electromagnetic or hadronic shower.

- Start and Veto detectors sitting before and after the magnet respectively. These are fast diamond detectors providing a 'start' signal for the TDCs and vetoing events with no interaction in the target respectively.

2.3.3 The START and VETO detectors

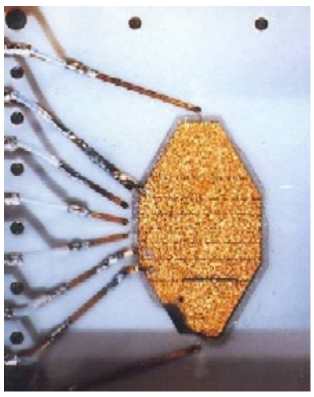


Figure 2.5: START detector

START and VETO are two identical diamond detectors placed before and after the target. The first of them provides the "start" signal, while the second vetoes all particles not interacting with the target's nuclei.

Both diamond detectors are synthesized using a Chemical Vapor Deposition (CVD) technique (18), which allows the diamond to grow in an environment under control.

The detector is capable of a time resolution, including electronics, of up to 29ns and tolerates rates of more than 10^8 particles per second for a single detector channel. The detectors are radiation resistant and can be constructed in very thin layers.

START and VETO are placed 75cm upstream and 75cm downstream of the target. The first of them being the START. Each of them is octagonal in shape and is divided in 8 strips, the width of which are optimized such that a coincidence of one start detector strip with three veto detector strips is sufficient for a veto efficiency of 96.5%. The detectors are kept thin (100 μm) to keep multiple scattering and secondary reactions very low.

2.3.4 The RICH detector

The RICH detector (2) is a threshold Cherenkov detector with the mission of separating leptons from the hadronic background. The working principle is the emission of Cherenkov light whenever a particle crosses a medium with

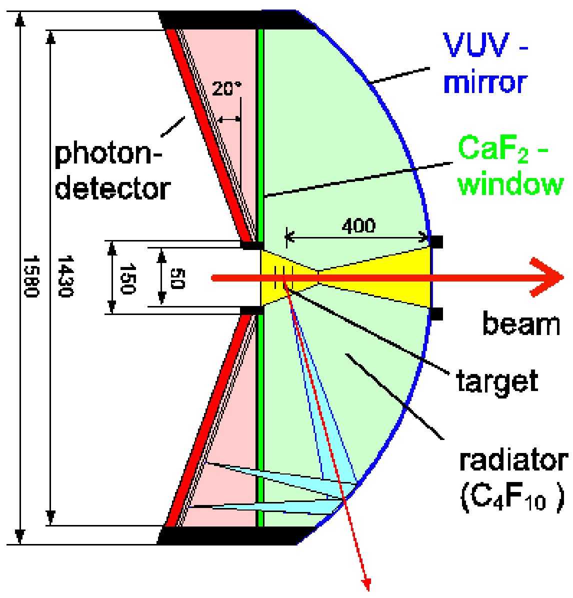


Figure 2.6: Side view of the RICH detector

a velocity larger than that of the light in the medium. In that situation, Cherenkov light is emitted in a well known direction determined by the particle's velocity (β) and the speed of light in the medium (β_t), given by the medium's refraction index (n). It is customary to quote the threshold γ for light production instead of the β. This is related to the refraction index by

$$\gamma_{th} = \frac{n}{\sqrt{n^2 - 1}}$$

The HADES's RICH (Fig. 2.6) consists of a gaseous C_4F_{10} radiator around the interaction point, closed in the back by a VUV spherical mirror and in the front by a gaseous photon detector separated from the radiator gas by a CaF_2 window. The radiator's Cherenkov threshold is $\gamma_{th} = 18.2$. Hence, leptons with momenta from 100 to 1500 MeV produce Cherenkov light, while hadrons in the same momentum range are below the threshold and go undetected. Interesting properties about the radiator gas are that it is transparent to ultraviolet light (down to wavelengths of 145nm) and the absence of significant scintillation light from charged particles.

The Cherenkov light emitted by the leptons is reflected in the VUV spherical mirror. This mirror has a diameter of 145cm and a curvature R=870mm. It is segmented in 18 panels, three per sector. The panels are made of pure

Carbon, machined to a thickness of 2mm. This allows minimizing multiple scattering and photon conversion. The average reflectivity is around 80%. Due to the large acceptance of the mirror and the location of the target (closer to the mirror than the curvature center), the azimuthal and polar focal surfaces have non-negligible curvature, which leads to the formation of ellipses instead of rings for large polar angles.

The reflected light crosses the 5mm thick CaF_2 window separating the radiator gas and the photon detector. CaF_2 is used due to its high transmission in the VUV region (around 70% at 140nm). The photon detector is designed to detect photons only in the VUV region. It consists of a Multi Wire Proportional Chamber (MWPC) with a pad structure behind the photo sensible CsI layer. The chamber operates on pure CF_4 with a gain of 10^5. The size of the 28272 pads ranges from 7x6.6 to 4x6.6 cm^2 in order to compensate for the mirror's spherical aberration. With this pad structure, the eccentricity of the ring images is corrected in first order and leads to rings of an almost constant diameter of 9 pads ($\sim 5.5cm$) for all polar angles.

The overall performance of the detector can be summarized in a figure of merit $N_0 \simeq 109\ cm^{-1}$, corresponding to a number of detected photons per ring between 12 and 21.

The detector analysis comprises several stages. The first stage consists on the cleaning of isolated noise hits, on a second stage hit clusters are identified and labeled. Then two independent algorithms are used to identify the rings:

1. A fast patter matrix search. The basic idea of this algorithm is to overlay a mask on the pads image. The mask is divided in cells corresponding to the pads, each cell has a weight, either positive or negative defining a ring shape. It can be seen as a bitmap of a ring. When the mask is overlaid to any particular region of the pad image the charges registered in each of the pads are multiplied by the corresponding weights and summed up. Due to the way weights have been chosen, the summed value is large for real rings and low for fakes.

2. A Hough transformation. The basic idea in this case is to apply a mathematical transformation to the pad image so that rings become points in the transformed space. This can be achieved by taking any possible combination of three pads and computing their center. Then an histogram is built with the centers of all those combinations, real rings should show up as peaks on that histogram; so the problem of ring recognition has become a much simpler one of peak recognition.

2.3.5 The magnetic spectrometer: MDC and ILSE

In order to achieve an invariant mass resolution of 1% in the ω meson region one needs to be able to reconstruct electron tracks with a resolution in momentum in the order of 1% for electrons with momenta larger than 100 MeV. For that purpose, HADES has a magnetic spectrometer consisting of a superconducting toroidal magnet (ILSE) and 24 multi-wire drift chambers (MDCS) in 6 sectors (see Fig. 2.7).

The magnet consists of 6 superconducting coils separately mounted in 80 mm thick boxes, producing an inhomogeneous magnetic field which reaches a maximum value of 3T near the coil cases, but produces a maximum magnetic field around 1.5T in the acceptance region. The momentum kick ranges from 40 to 120 MeV at full field. Fig. 2.8 shows a transversal cut of the field map at $\phi = 0$.

The 4 drift chambers in each sector are divided into two groups, with two chambers before the magnet and another two after. The toroidal shape of the field allows to confine it in the region between the chambers. The frames of the chambers are situated in the shadow region defined by the magnet's coils so that the acceptance is not further reduced. The size of the drift cell in the chambers ranges from 2.5 to 7 mm, keeping a high enough granularity to deal with occupancies in the order of 0.6 cm^{-1} (in the low polar region). Each chamber has six layers of sense wires with respective orientations +40°,-20°,0°,0°,+20°,-40° providing enough redundancy in the measurements. The wire orientation is chosen to optimize the position resolution in the direction of the momentum kick, thus maximizing the resolution in momentum. Besides, the two 0° layers are displaced by half a cell in the direction perpendicular to the wires.

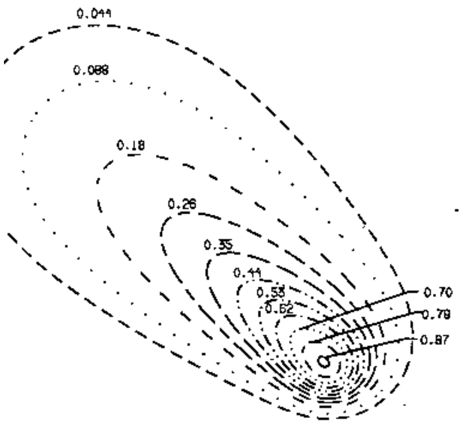


Figure 2.8: Transversal cut of the field map at $\phi = 0$

In order to obtain the required momentum resolution, it is necessary to minimize the effect of multiple scattering. That effect is dominant over the position resolution for momenta below 0.4GeV. To minimize it, low mass materials have been chosen in the manufacturing of the chambers. The Golden Tungsten sense wires have a diameter of $20\mu m$ and the field and cathode wires are $12\mu m$ thick. The gas mixture used is $He - iC_4H_{10}$.

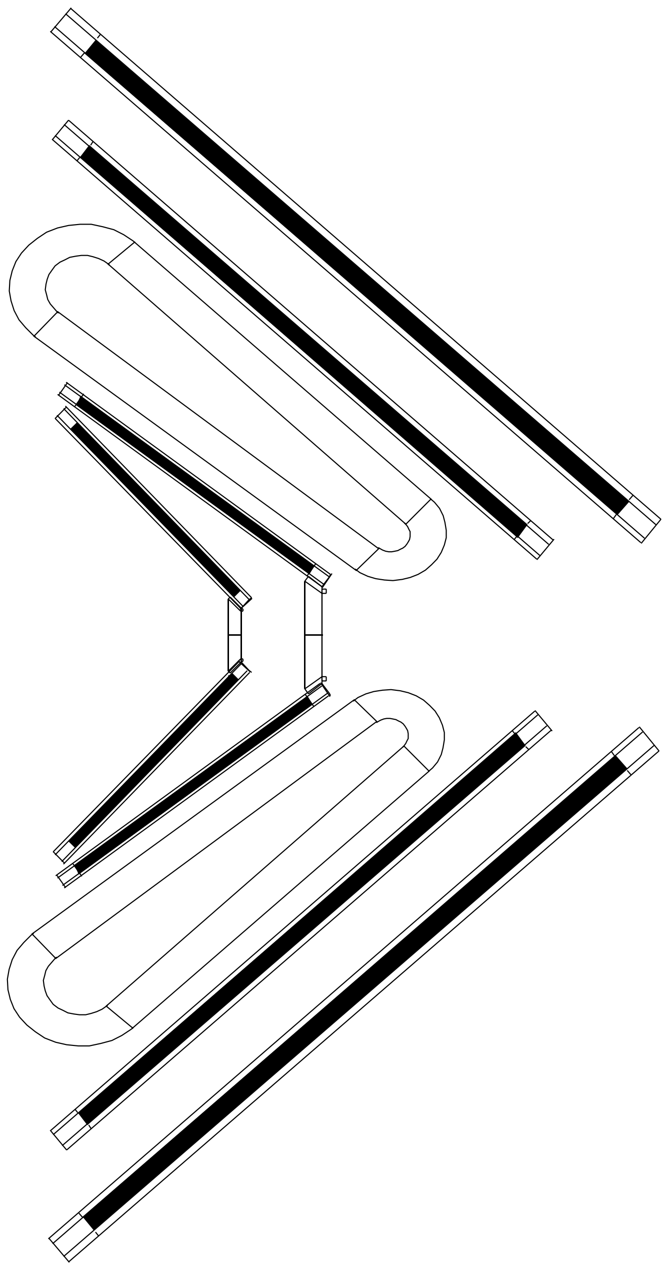


Figure 2.7: Transversal cut showing to opposing sectors of the HADES magnetic spectrometer

Drift velocity for the mixture saturates around $4cm/\mu s$ for the used voltages, managing a nearly linear relationship between drift time and track position in most of the cell (not in the borders). The average resolution for a single hit is around $80\mu m$ over more than 80% of the cell.

In order to deal with the high acquisition rates in HADES a custom TDC has been devised. It can work at 25MHz and is manufactured in 0.6μ technology.

2.3.6 The TOF and TOFINO detectors

The TOF (5) detector measures time of flight of charged particles in the polar region between 45 and 85 degrees. The detector is made of plastic scintillator rods, with length ranging from 2 to 3 meters and profiles of 3x3cm^2 and $5x3cm^2$. The rods are grouped in sets of eight, with 8 such sets per sector.

The light produced in the plastic is read out by 2 photo-multipliers sitting at both ends of the rod, this allows for a time of flight reconstruction with a resolution between 100ps and 150ps. The position where the particle has hit the detector can be inferred from the time difference between the measurements at both edges with a resolution ranging from 1.5 cm to 2.3 cm.

The multiplicity in the TOF detector is already used during the first level trigger on centrality. For the second level trigger time of flight is already available providing a separation between electrons and pions up to 0.5GeV and protons up to 2GeV. On a later stage of the analysis the time of flight information is used in the Particle Identification process.

Besides the time of flight, each rod is equipped with ADCs allowing the measurement of the signal height from the photomultiplier. From that information, energy loss in the plastic can be inferred, which contributes to the particle identification.

Furthermore, when the outer MDC chambers are temporarily not available in one sector, the TOF detector can still provide with some position information used for momentum reconstruction. This functionality is available in the second level trigger also.

On the lower polar regions the TOFINO detector substitutes the TOF. This detector is made of 4 scintillating planes covering the low polar angles up to 45 degrees. Only one photo multiplier is used per scintillator in the TOFINO case. The substitution of the TOFINO detector by a RPC (Resistive Plate Chamber) wall is foreseen in the not so distant future. The new detector must provide the granularity needed to deal with the higher multiplicities in the heavier ion collisions.

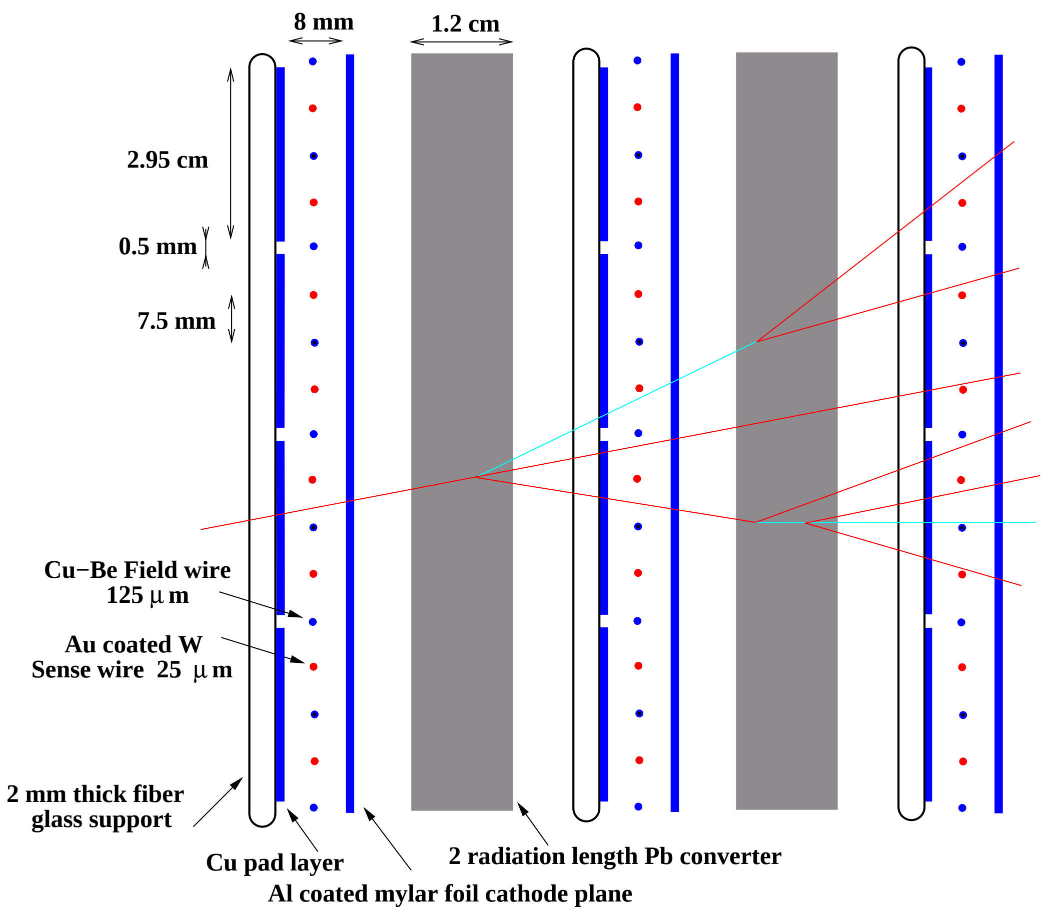


Figure 2.9: Side cut of the SHOWER detector

2.3.7 The SHOWER detector

Due to the larger momenta of particles at low polar angles, the lepton identification based only on time of flight measurements is not enough. Consequently the SHOWER detector (19) is placed in the rear part of the spectrometer, at low polar angles ($\theta < 45$) with the main goal of improving the lepton/hadron discrimination.

The SHOWER detector is made of 6 identical sectors symmetrically distributed around the beam axis. Each sector consists of 3 wire chambers with pad readout, separated by two lead converters 1.2 cm wide (2.5 radiation lengths). The width of the converters is optimized to maximize the probability of producing an electromagnetic shower for leptons in the HADES energy regime, while keeping to tolerable levels the chances of an hadronic one.

The detector has a trapezoidal shape covering polar angles between 18^o

and 45^o on the full azimuthal range of the sector. The three wire chambers are slightly different in size to compensate for the solid angle variation due to their different distances from the target.

Each wire chamber has a wire plane with alternating 25μm golden tungsten sense wires and 125μm copper beryllium field wires separated 7.5mm from each other. The wire plane is situated between two cathode planes symmetrically disposed around it to configure the drift cells. One of the cathode planes is made of stainless steel. The opposite one is made of 8 fiber glass plates covered with a thin copper layer, 1.5 mm thick, where pads are disposed forming a grid. Pads are aligned with respect to the target so that particles coming from it cross a set of fixed pads in the three modules.

There are 32 pad rows and a number of columns ranging from 20 to 32, making up for a total of 942 cells. The pad's heights range from 3cm to 4.5cm. Each pad covers an integer number of drift cells.

The wire chambers are operated in Self Quenching Streamer (SQS) mode. This has the advantage that the induced charge is nearly independent of the particle's energy loss. Hence low momentum protons do not leave larger signals on the post converter and are thus not misidentified as leptons.

The detector's analysis starts by identifying local charge maxima in the wire chambers. The local maxima correspond to the places where a particle has hit. For each hit, the sum of the charge induced in the pad of the local maximum and its 8 neighbor pads is computed. The same sum is computed for the corresponding nine pads in the other two modules. The ratio of those sums is then calculated. The electromagnetic shower differs from the hadronic one on the value of those ratios[5], so a cut can be performed to distinguish between both kind of them. The cut values themselves are determined from simulation and they depend on momentum.

2.3.8 The trigger scheme

The trigger system (5; 20) for HADES is organized in 3 stages or levels. Its task is to select central events with lepton pair candidates out of all the event data generated by the spectrometer.

As already mentioned, the beam intensity HADES is targeted to operate with, is around 10^8 particles per second, with a 1% interaction length target that translates into 10^6 events per second.

[5]The electromagnetic shower is shorter, so the multiplication factors are larger in the early stages

Out of all the events we are only interested in the central ones. To select them is the task of the first level trigger (LVL1), which achieves a reduction factor of 10 on the data by requiring a minimum multiplicity in the TOF and TOFINO detectors. This reduces the amount of data to 10^5 events/s, or 4 GB/s.

The second level trigger is based on the search for lepton candidates in the event. For that purpose only the RICH, TOF and TOFINO information is used, keeping the rest of the data in intermediate memories until a decision is taken. Two stages can be differentiated in the second level trigger logic. In the firs stage, detector specific Image Processing Units (IPU) search for lepton candidates in their respective detectors. The resulting candidates are sent for its processing to the so called Matching Unit (MU).

The MU is a programmable device allowing different working modes. In the simple most one a positive trigger decision is issued any time a lepton candidate is found by some of the IPUs. Several higher level steps can be optionally performed:

1. The RICH and TOF/SHOWER lepton candidates are matched within a window in polar and most importantly azimuthal angles.

2. Two lepton candidates of opposite polarity are required with an opening angle larger than a certain cut value. The cut on opening angle is intended to reduce the background from conversion electrons.

3. The invariant mass of the open lepton candidate is computed accepting only masses within a certain window.

The time available for the second level trigger to make its decision is $10\mu s$ on average. Operating at full potential, the second level trigger can achieve reduction factors around 100. Thus achieving 10^3 events/s or 40 MB/s.

The, still non existent, third level trigger would additionally use the MDC information. It would perform a fast tracking using only the information about what MDC wires are fired. This would allow to cross check the lepton candidates identified by the second level trigger as well as a better invariant mass resolution derived from an improved momentum measurement. The expected reduction factor for the third level trigger is 10.

With the three trigger levels active about 100 events per second (4 MB/s) would finally be written to tape.

2.3.9 Data Acquisition system (DAQ)

The acquisition system for HADES is based on an ATM network connecting the VME crates, used for the detector read out and second level trigger, with a central event builder CPU. The data taping speed in the event builder is 5 MB/s, corresponding to up to 2000 events per second on a C+C collision or around 100 events/s for the heavier systems.

The DAQ uses two pipes. A first level trigger pile where data are pushed after a positive LVL1 trigger decision. The data is then transmitted to a second level trigger pipe after a positive LVL2 decision. If the LVL2 decision was negative the data is removed from the first level trigger pipe.

The second level pipe is implemented in RAM memory of the VME bus system, being directly mapped in their address space by a fast transport into the event builder.

2.4 The HADES reconstruction software

HADES has a reconstruction software called HYDRA. It is written on C++ using the Object Oriented paradigm and it is based on the ROOT framework. The software package is extensively presented in chapter 3.

Bibliography

[1] Reiner Schicker et al. Acceptance and resolution simulation studies for the dielectron spectrometer HADES at GSI. *Nuclear instruments and methods in Physics Research*, A(380):586-596, 1996

[2] K. Zeitelhack et al. The HADES RICH detector. *Nuclear Instruments and Methods in Physics Research*, A(433):201-206, 1999

[3] W. Cassing, V. Metag, U. Mosel and K. Niita. *Production of energetic particles in heavy ion collisions.* Physics Reports, 188(6):363-449, 1990

[4] C.M. Ko and G.Q. Li. *Medium effects in high energy heavy ion collisions.* Nuclear Particle Physics, (22):1673-1725, 1996

[5] The HADES proposal

[6] Heike Neumann. HADES - *a high acceptance dielectron spectrometer proposed for relativistic heavy ion and nucleus-nucleus collisions.* Acta physica slovaca, 44(3):195-205, 1994.

[1] Groom et al, Particle Data Group. *Review of Particle Physics.* Eur. Physics J. C15 1-878, 2000

[8] M. Masera et al. *Dimuon production below mass 3.1 GeV/c^2 in p-w and s-w interactions at 200 AGeV/c^2*. Nuclear Physics , A(590):93c, 1995.

[9] E.L. Bratkovskaya, W. Cassing, R. app, J. Wambach. Nuclear Physics A634, 1998

[10] G. Agakichiev et al. *Enhanced production of low mass electron pairs in 200 AGeV S-Au collisions at the CERN super proton synchrotron.* Physics Review Letter 75(7): 1272-1275. 1995

[11] CERES collaboration. *First results from CERES/NA45 on low mass electron pair production in Pb-Au collisions.* Nuclear Physics, A(610):317c-330c, 1996

[12] G. Agakichiev et al. *CERES results on low mass electron pair production in Pb-Au collisions.* Nuclear Physics A(638):159b, 1997.

[13] Jim Carroll. *Measurement of* e^+e^- *pair production at BEVALAC.* Nuclear Physics, A(495):09c-422c, 1989

[14] C. Naudet et al. *Threshold behavior of electron pair production in p-Be collisions.* Physical Review Letters, 62(23):2652-2655, 1989

[15] G.Roche et al. *First observation of dielectron production in proton-nucleus collisions below 10 GeV.* Physical Review Letters, 61(9):1069-1072, 1988

[16] R.J. Porter et al. *Dielectron cross section measurements in nucleus-nucleus reactions at 1.0 AGeV.* Physical Review Letters, 79(7):1229-1232, 1997

[17] S. Beedoe et al. *Measurements of dielectron production in niobium-niobium collisions at 1.05 AGeV.* Physics Review C(47):2840, 1993

[18] E. Berdermann et al. Nuclear Physics (Proc Suppl.) B(78):533-539, 1999

[19] P.Salabura et al. Nuclear Physics B(44):701, 1995

[20] M. Traxler et al. *The second level trigger system of the HADES detector.* Internal HADES report, 1999

Chapter 3

The Hydra event reconstruction software

The main goal of the HADES experiment is the study of in medium modifications of vector meson properties, through their decay in lepton pairs. In order to achieve that goal, the HADES spectrometer has been built, but we also need software enabling us to make use of the machine and transform the low level information provided by the detectors into some high level data, with higher level physical content. We can divide the software used by HADES in four large working areas:

- Data acquisition and monitoring
- Simulation
- Event reconstruction
- Physics analysis

In this chapter we will concern ourselves mainly with the event reconstruction software. Actually we will speak about the design and inner workings of the framework for the event reconstruction program, called Hydra[1]. Finding a definition for *framework* is a difficult task. We could say a framework is a set of rules, interfaces[2] and services put at the disposal of programmers, who can extend it to perform a set of tasks.

[1]Hades sYstem for Data Reduction and Analysis (see (5))

[2]Interfaces in C++ are typically realized through abstract classes. That is classes defining methods to be overridden by derived ones.

A framework is not a class library. A class library presents the programmer with a set of interfaces and services that the programmer can use at his/her leisure. The program's logic is not specified by the class library but by the programmer's code. In a framework, the programmer is presented with a whole comprehensive unit which he/she can extend, in predefined ways, to perform specific tasks. A class library can be completely generic, while a framework needs to go one step down and put in some knowledge about the problem's domain.

In the case of HYDRA the main goal is the processing of events recorded in a spectrometer. In other words; we read input data and those data are processed by a set of algorithms which depend on parameters and need access to the data in some structured way. As a result, new, elaborated data, are produced. Typically we want to store those data for future analysis. A more comprehensive list of requirements is presented in section 3.1; but this paragraph already introduces the main concepts in HYDRA: algorithm management, data input, data structures, parameter management and data output for analysis. Besides the subsystems, HYDRA has to specify how do they interact.

A framework specifies an overall design. Furthermore it should be tailored such that the design is enforced, taking good care of not being too restrictive so that the programmers, the users of the framework, are deprived of the necessary freedom. For this reason it is the task of the framework to provide modularity by defining interfaces so that the main areas; input, output ... are loosely coupled. Allowing modifications in any of the subsystems without altering the others.

To match those goals an Object Oriented Programming is an ideal approach. Encapsulation is critical in enforcing modularity as opposed to suggesting it. Polymorphism is what allows a framework to be extended by the programmers in a well defined way.

A framework does nothing interesting by itself. In the case of HYDRA all the interesting stuff occurs inside the algorithms which fill the software with life. To a great extent HYDRA only exists to make the life easier for the programmers writing algorithms. Other chapters in this same thesis work will present some of the algorithms running in HYDRA. Certainly, a thorough description of all the different algorithms in HYDRA escapes the scope of this work. By the time of this writing, several thesis are being written to document them.

Section 3.1 presents an analysis of what the requirements are on HYDRA, shaping its design. The following sections present in more detail the actual framework in its current incarnation. The last section in this chapter spends some time on the simulation package for HADES: HGEANT. Even though the design and implementation of HGEANT are not part of my work in HADES, it is presented here since it will be often referenced in the following chapters.

3.1 Software requirements

The main goal of HYDRA is the reconstruction of events recorded by the HADES spectrometer. Reconstruction means application of various algorithms to transform the *Raw* data delivered by HADES into elaborated data, ready for the analysis step.

The reconstruction proceeds in steps. Each algorithm reads some input data, maybe the output of another algorithm, and takes it to a new level of elaboration. In that sense we can speak of *data levels*, which correspond to the different levels of elaboration.

For example, the first data level is the so called *raw* data, corresponding directly to the electronic signals recorded by the spectrometer; typically a number of channel from an ADC. The next level is usually the *calibrated* data; it is a direct mapping of the *raw* data in physical units. While raw data represent channel numbers, calibrated data may represent times, charges etc. depending on the detector being *calibrated*. The algorithm going from raw data to calibrated data is detector specific and generically known as a *calibrator*.

The handling of algorithms and data levels can be seen from two perspectives. From the perspective of the developer using the framework to implement new algorithms, and from the perspective of the final user wanting to do analysis with the full HYDRA and accompanying algorithms. In fact, most often one person is at the same time developer and user. But we will separate here these two ideal personalities since they have different requirements.

3.1.1 The final user's point of view

The final user is concerned about producing a result and not very much about the software's inner workings. The analysis software needs to be flexible enough to adapt to the different uses demanded from it.

A final user wants to be able to analyze not only *raw* data, but also partially elaborated ones. In particular, steps like tracking, which are very expensive CPU wise, should not be repeated every time a new cut wants to be tested on the data. Another "use case" is that the parameters for any of the analysis steps change but we still do not want to repeat the whole reconstruction in order to get updated data. Only the affected steps should be repeated.

This means the concept of *data levels* must show up in the framework. Furthermore, we need for the framework to be able to persistently store any given data level and to start the processing from any other one. As far as possible this should be transparent to the user. She only needs to specify what the input file is and what algorithms to use on it.

The data levels should be stored in a format which allows easy treatment by the physicist in the analysis phase. In other words, the final user needs to take the output data and easily plot the different variables, separately or one versus the other. She needs to be able to quickly apply cuts, or make simple computations on the variables.

The inner structure of a data level may affect the way it is written out and therefore the way it is accessed later. For this reason the user should be allowed to change the inner structure of a data level at runtime, without implying a change in the algorithms filling or reading from that data level. However this should not be a common task.

There are other things the user wants to be able to specify as input to the software. Here is a list:

1. The event data to be reconstructed. Those data may be stored in different formats, and therefore come from different sources:
    - The Data Acquisition system, which records the data taken by the spectrometer in tape. Those tapes need to be read and analyzed
    - Partially reconstructed events, generated by the reconstruction program itself. This enables for example, to try different reconstruction methods without restarting the analysis from scratch in each try.
    - Others[3]
2. Number and kind of the data levels, the so called *event structure*. However this is specified manually in very rare cases. The software must be intelligent enough that given a set of algorithms to apply on the input data, the right data levels are figured out and created without user intervention.

[3]What is meant with *others* is that the program must be flexible enough that new unforeseen data sources can be implemented with a minimum of fuss. For example a data source implementing the merging any other two, enabling the embedding of simulated events into real ones.

3. The reconstruction parameters to be used. Each algorithm depends on a set of parameters to perform its task. The user needs to be able to specify which parameter she wants to be used. However it must also be allowed that the user only asks for the *right* parameters, that is the recommended parameters from the algorithm's developer. Even if the user specifies a source where to get parameters from, there will typically be many versions of those parameters available. The framework must be able to load the right version for the data being analyzed automatically. However, the user needs to be able to override the automatic procedure if she wants to do so. Examples of foreseen, and supported, parameter sources are:

    - A central relational database management system. In our case based in ORACLE
    - A standalone file. Either binary or ASCII.
    - Introduced by hand

4. The algorithms to be applied on the input. Being able to specify this as an input makes the code be usable for different purposes without the need for a recompile relink cycle.
   There should be the possibility to incorporate new algorithms to the program while keeping the compiling and linking time to the minimum possible. That means the organization of the code must be modular; distributed in independent units which can be loaded and unloaded at runtime.
   The mechanism used to select the list of algorithms to apply needs to combine flexibility for the power users and simplicity for the novel ones. A power user may want to exhaustively specify the list of algorithms, maybe introducing between them program flows not necessarily linear. On the other side, other kind of user is only interested in selecting from predefined sets of high level algorithms and apply them one after the other.

Once the reconstruction program has performed its task, the user wants to be able, by looking at the output file, to tell which parameters and algorithms were used to obtain that output.

Lastly, the user may decide to use the software either in interactive mode or batch mode for mass production.

3.1.2 The developers view

The main requirement from a developer is to be able to worry only about her algorithm. She wants to have interfaces enabling her to access the information needed for the algorithm; but hiding the complexity of managing that information.

An algorithm is a closed unit; it defines some hooks to attach itself to the rest of the framework. It is presented with interfaces to access the data without worrying about the internal structure of those data or where did they come from. It is presented with a way to access proper parameters, without it worrying about which is the right parameter version for the data being processed.

An algorithm relies on an input data level, without any assumption about what other algorithm was used to produce those data. That is core to the modularity required by the final user.

Developers are geographically dispersed, so they want their algorithms to be in a separated code unit which can be loaded or unloaded at any moment. So that the code under development can be easily tested.

Also, an algorithm should not need to worry about how the data it produced is going to be stored.

3.1.3 Technical requirements

There are some technical decisions, or conventions, about the paradigms and tools to use for the event reconstruction software. Some of these actually go into the field of system's specifications rather than requirements; but are still shown here. The lists is as follows:

1. Use of Object Oriented Programming. This programming paradigm allows to attack medium and high complexity problems in a more solid way. In particular when the software development is performed by a dispersed group of people.

2. C++ programming language. Among the different programming languages in the market, C++ is chosen for its performance and growing popularity in the field of Particle and Nuclear Physics. That popularity is what makes available specific libraries and tools.

3. The chosen compiler is g++. This compiler is chosen because it is free and available in a large number of platforms. Having one standard compiler makes easier the sharing of code between the different groups

of developers. However it is not allowed to use extensions of C++ allowed by g++ but outside the C++ standard (see (6)).

4. Graphical notation: UML (see A). Software development in an international team makes necessary the usage of well defined communication tools. One of them is a standard notation for diagrams specifying a software's design. To capture the program's design in diagrams may be a tedious task, but useful for new users trying to understand the system.

5. Control versioning system: CVS. The need for a central system of version control is a consequence of the distributed development environment. CVS is chosen because it is free, available in a large number of platforms and abundant tools exist to work with it.

6. The ROOT system (see (2)) is used as a class library. It provides us with

   (a) A user interface

   (b) An structured Input/Output system designed taking into account the necessities of physical data analysis

   (c) Graphics and histograming capabilities

   (d) Documentation generation system

7. Supported platforms

   (a) Linux

8. Regarding the code organization it must be divided in independent shared libraries. A common library should implement the basic code (the framework itself) and all other libraries, fundamentally providing algorithms but not only, may be loaded on demand.

3.2 System overview

It was already mentioned that Hydra touches 5 main areas

- Data Input, either *raw* or partially reconstructed
- Data Output in a form suitable for further analysis

- Management of algorithm's parameters; like detector geometry, calibration constants...
- Data structure
- Management of algorithms

Besides that, we also need an additional subsystem to coordinate them all. This sections presents an overview of the different subsystems in HYDRA with the aim to give the reader the big picture; to be kept in mind while reading the following sections, where the different subsystems are explained in detail. Each subsystem is defined by a set of classes:

- FUNDAMENTAL CLASS: HADES
  Hades is the class which encapsulates the whole reconstruction program, coordinating all HYDRA subsystems. It provides methods to control the different tasks which can be performed. Including methods to set the different inputs, like parameters, algorithms to be ran etc. It also provides methods to launch the reconstruction process itself.
  The HADES class contains objects of different classes which it uses to implement the different subsystems. Those object's classes can be overridden in order to extend the framework at the user's will. The structure of the HADES class can be seen in Fig. 3.1. See also section 3.3
- CLASSES TO CONTAIN DATA
  Since the reconstruction process is done event per event, the basic data structure is the **event**. From the point of view of Physics, an event holds all the information collected by the different detectors in the spectrometer regarding one interaction between one beam particle and the target. Moreover, it also stores the totally or partially reconstructed data which are derived from the original signals. But we can have other types of events which do not contain information about an interaction in the target; for example, signals produced in the detectors by a particle with the purpose of calibration. There can be, therefore, different kind of events: real events, simulated events, calibration events, etc.
  An event is represented by an object implementing the HEVENT interface. This allows (and enforces) to inherit from HEVENT other classes which correspond to the different kind of events we can find. Within an HEVENT data are organized in so called *categories*.
  A *category* is a data container; in Object Oriented Terminology that means it is a container of objects. The main particularity of categories

is that all objects in one category have to instantiate exactly the same class. The reason for this requirement will be explained in section 3.5.1. For the moment it is enough to say that categories present the algorithms with a well defined way to access the data, hiding the data's internal structure as much as possible. That makes easier to change the inner representation of the data, for example, to accommodate a change in the Input/Output system, with minimal effect on algorithm's code.

- CLASSES TO MANAGE INPUT OF DATA
  Once we have a place where to store an event's information, we need a tool to read that information in. Since the data may come from different sources, as explained in the requirements section, a generic interface is needed for the HADES class to use. Each data source is then realized as a different implementation of such interface. The interface is represented by the abstract class HDATASOURCE. See also section 3.6

- CLASSES TO MANAGE DATA OUTPUT
  A method is needed to put the data from the data containers into persistent output in files. For this purpose ROOT's TTREE structure is used. A TTREE is an extension of the well known PAW (see (1)) ntuples to store complex structures. More on this in section 3.4

- CLASSES TO MANAGE PARAMETERS
  In order to do the event reconstruction, several numerical parameters are needed, like for example, calibration parameters or geometry positions of detectors. One common characteristic to most of these parameters is that they will go through different versions, corresponding, for example, to changes in the spectrometer or any other relevant condition. This makes necessary to have a parameter repository with some versioning scheme. For that purpose the concept of a runtime database, implemented in the HRUNTIMEDB class, is introduced. See also section 3.7

- ALGORITHM MANAGEMENT
  For each event we need to accomplish a certain task, which is represented by an HTASK object. Again, HTask is an abstract class defining a generic API[4] allowing to execute one task and to navigate through a task set.

[4]API stands for Application Programming Interface. It is the definition of classes and methods provided by a system for the developer's use.

HReconstructor and HTaskSet are two particular subclasses inheriting from HTask. The first of them is meant for reconstruction algorithms, while the second one allows us to group several tasks in one. For more information see section 3.8.

3.3 The HADES class

At the center of HYDRA we find a class which is responsible for activating the different subsystems in the framework, provide access to them and coordinate them in processes like, for example, initialization or event processing. The class structure is shown in the diagram 3.1. The main components are

- A data source where to read event data from
- Several HTASKSET objects storing the tasks to perform for each event type
- An HEvent where to store the information being processed
- A HSPECTROMETER created during initialization and storing information about the spectrometer's structure
- A database where to read reconstruction parameters from
- A ROOT output tree
- An output file

The HADES class is a singleton (see (4)). There must be one and only one instance of this class active at the same time. There are two ways to access that object, either the *gHades* global pointer or the Hades::instance() method. The second one is recommended over the first, though. The simple global variable has the problem that the instantiation of the HADES class is left to the user of the framework, which can make the mistake of doing it twice. Another problem with the global variable is that it is a potential source of trouble should the framework be extended with multiple threading[5]. Indeed, in a multi threaded environment there is no way to ensure that *gHades* is

[5]Multiple threading refers to a technique where several execution paths are running in parallel inside one single program. This is possible thanks to the multi tasking abilities of modern Operating Systems

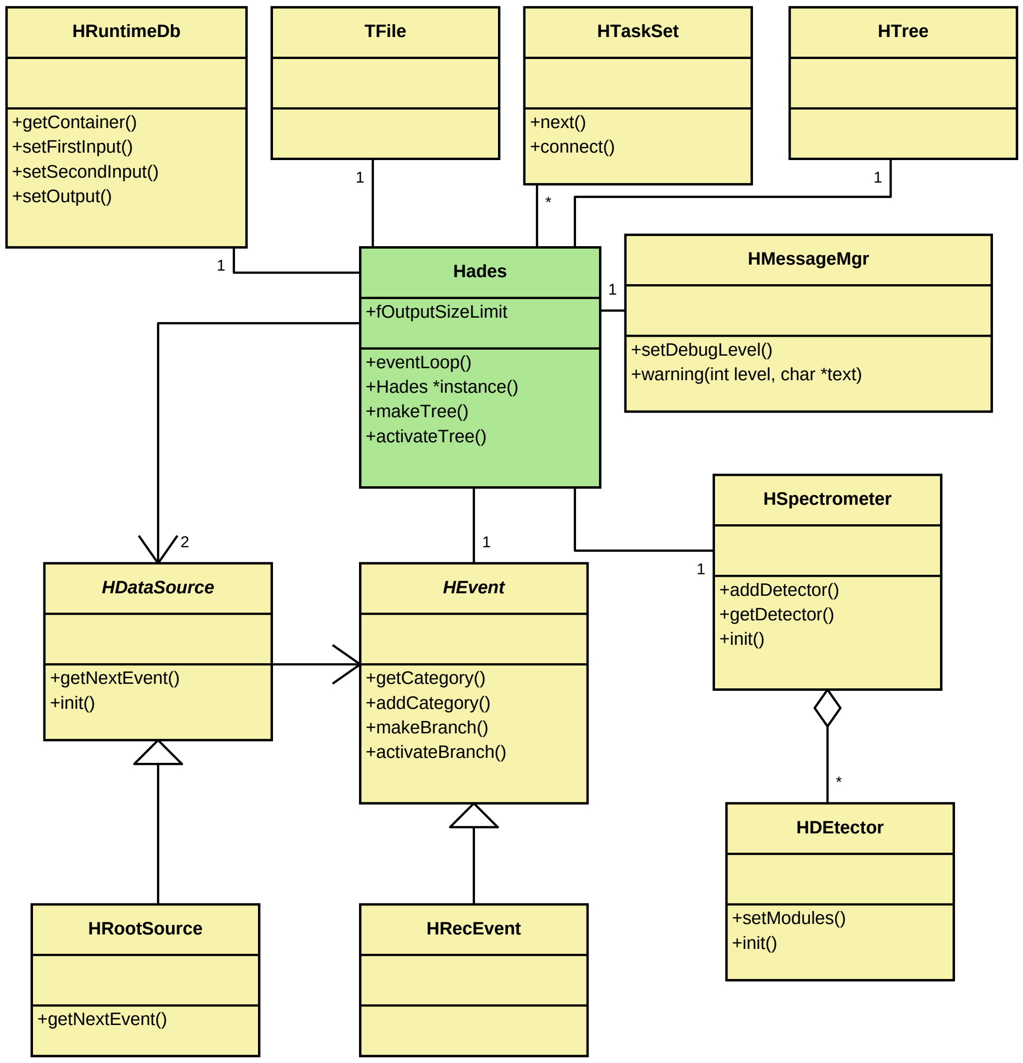


Figure 3.1: Hades class structure

not accessed at the same time from two different threads, potentially giving raise to race conditions[6] and therefore making the program unstable.

On the other hand Hades::instance() gives back a pointer to the current HADES object if it was already instantiated, and instantiates one in case none is in memory. The first advantage is that the instantiation is always done once, without user intervention. The other is that it is very easy to put a semaphore[7] inside that method to provide for multiple threading.

3.4 Data output

The output system is based on the ROOT Input/Output facilities, with functions to store objects in so called ROOT files. This subsystem is responsible not only to store event's data into an output file, but also information about how those data where created.

For the second part, we use ROOT's ability to store any conforming C++ object to file. Then, the event's structure, algorithm's setup, control histograms and so on can be stored by just writing the corresponding manager objects. Since the HADES singleton is the manager of the managers, what is done at the end is simply writing out the HADES object.

To store the event's data a ROOT's TTREE facility is used. This structure is specialized in storing multiple objects of the same class in a friendly way for physics analysis. One can think of a TTREE, "Tree" from here on, as an extension of a PAW ntuple to Object Oriented Programming. That means it has to be able to cope with complex structures made by objects storing arrays of other objects.

A Tree is created by providing the class constructor with the class name and a pointer to the kind of object which will be stored. Then the Tree itself creates an internal representation for the object's structure. The internal representation is defined in terms of branches; for each data member of the object's class a branch is created. In case the data member is itself one object, subbranches are created for each of its data members.

[6]Race condition refers to the situation where a system resource is accessed by one process before it has finished attending the request of some other. Therefore the second request finds the system in an undefined internal state.

[7]A semaphore is a tool used in multi threading programming to protect the access to resources. Whenever a resource is in use by some thread (or process) the associated semaphore is set to *red*, otherwise it would be *green*. A second thread accessing the resource finds the semaphore in red and must wait until it is green in order for its request to be attended.

There is a special case in the previous procedure. When the object to be processed is an instance of the TClonesArray class the behavior is slightly different. In order to explain the special behavior for this case we need first to know what a TClonesArray object is. A TClonesArray is an array of objects instantiating exactly the same class; so all of the objects in the container have the same data members. This fact is used by Root to create branches for each data member of the class instantiated by the objects in the TClonesArray. Typically this class is used to store the ultimate objects containing the physical information. What happens then is that we get independent branches for each of their data members, which are the physical variables defining the event.

Doing it that way is convenient because Root provides plotting functions which operate on individual branches; both to define the variables to be plotted and to place arbitrary cuts, providing a convenient method for data inspection. Following the idea of ntuples it is also possible to read only one branch at a time if it is so desired, thus significantly speeding I/O when we are interested in operating over a subset of the data.

On the other hand for a data structure to be suitable of being stored using TTree it cannot be arbitrarily complex. For example complex template structures are not allowed or pointers between the data objects stored may bring a performance penalty. Still the possibilities offered are powerful enough.

Not the whole data structure needs to be dumped into output. Through the HCategory::setPersistency() method it is possible to define which data levels should and should not be stored.

3.5 Data structures

This section is devoted to the data structures used in Hydra to store event's data. A physical event is an interaction between a beam particle and the target, it can be either real, simulated or a calibration event. A calibration event contains the response of one or several detectors to one or several particles or to a calibration signal (a laser signal, for example). The event is the unit for data processing. It can contain both the original data coming from the spectrometer (*raw* data) and the more elaborated data which is result from the reconstruction process. Since events are independent of each other we only need to store one event in memory at any given time.

We will call *event structure* to the data structure used to keep in memory all relevant information about one event during the reconstruction process.

The event structure is a nexus where several subsystems meet; each of them imposing certain requirements on the event structure. The algorithms need a fast way to access the event's data; both sequentially, i.e. during calibration, or randomly, i.e. in tracking.

Every event is reconstructed in steps, each step in the reconstruction process producing one new level of reconstructed data. The number and kind of levels which are stored in an event is not fixed beforehand. It can change as a function of the kind of event (simulated, real ...) as well as the specific task we are accomplishing at a given moment. If, for example, we are studying the calibration parameters for the MDC, it makes no sense to carry neither every data level for the other detectors nor those MDC levels which are not used.

The event structure is made accessible through the HEVENT interface, which basically works as an array of data levels. Each data level is accessible through the HCATEGORY interface; which works as an object container; providing, among others, functions to access the data in the level. These two classes only define the interfaces, the actual implementation of the data structure is relegated to their subclasses.

In the following sections, the HEVENT and HCATEGORY interfaces are presented in more detail.

3.5.1 The event

All data relative to one event is represented as one object implementing the HEVENT interface. HYDRA's central class, HADES, keeps one HEVENT object in memory at all times, that is the current event. It is globally accessible because it belongs to HADES.

As we have already said, conceptually an HEVENT is nothing else but a container for categories, providing access to any category in the event. That is accomplished through the function HEvent::getCategory(Cat_t aCat) which takes as only argument a unique identifier for a category and returns the corresponding object. This identifier works as the category name. For example, one such identifier is catMdcRaw which corresponds to the category storing raw data relative to the MDC detector. In order to access that category we would write something like

```
HEvent *event = Hades::instance()->getCurrentEvent()
HCategory *c = event->getCategory(catMdcRaw)
```

The result is that now the pointer *c* references the catMdcRaw category and we could use the HCategory interface to access the individual raw signals stored in it.

Other methods allow to manage the categories in the event, adding them. In that sense the event structure is dynamical. The type and number of categories is not specified beforehand. It depends on the tasks to be performed.

Besides the functions dealing with categories, there is a HEvent::makeBranch() method which creates a Root TTree's branch for the event. For simple implementations of HEvent this would be as simple as a call to a standard Root service; for more complex implementations makeBranch() assists Root's mechanism with knowledge about the internals.

Only two interdependent implementations of HEvent have survived in Hydra. They are presented in the following sections.

The event under reconstruction

The HRecEvent class implements the HEvent interface for physical events which can be totally or partially reconstructed. Its class structure can be seen in figure 3.2. We will discuss in this section what is the particular structure for this implementation.

In reality HRecEvent does not physically contain any categories. What it contains is partial events. Partial events will be discussed later, but essentially they serve as a way of grouping related categories.

But HRecEvent is a HEvent and therefore it is supposed to contain categories and provide access to them. When the users asks HRecEvent for a category, this, in turn, selects the corresponding partial event and gets the category's pointer from there.

One needs to store more information in a physical event than the data objects themselves; therefore each HRecEvent has also a header (object of class HEventHeader). This header stores global information about the event, like event number, run number, date, trigger ... To access an HRecEvent's header one can use the HRecEvent::getHeader() method.

Each partial event is identified within the HRecEvent to which it belongs by a numerical constant whose value is that of the first category one can store within that partial event.

The partial event: HPartialEvent

As its name suggests, a partial event is part of an event under reconstruction. It serves to group related data levels or categories. This way all categories

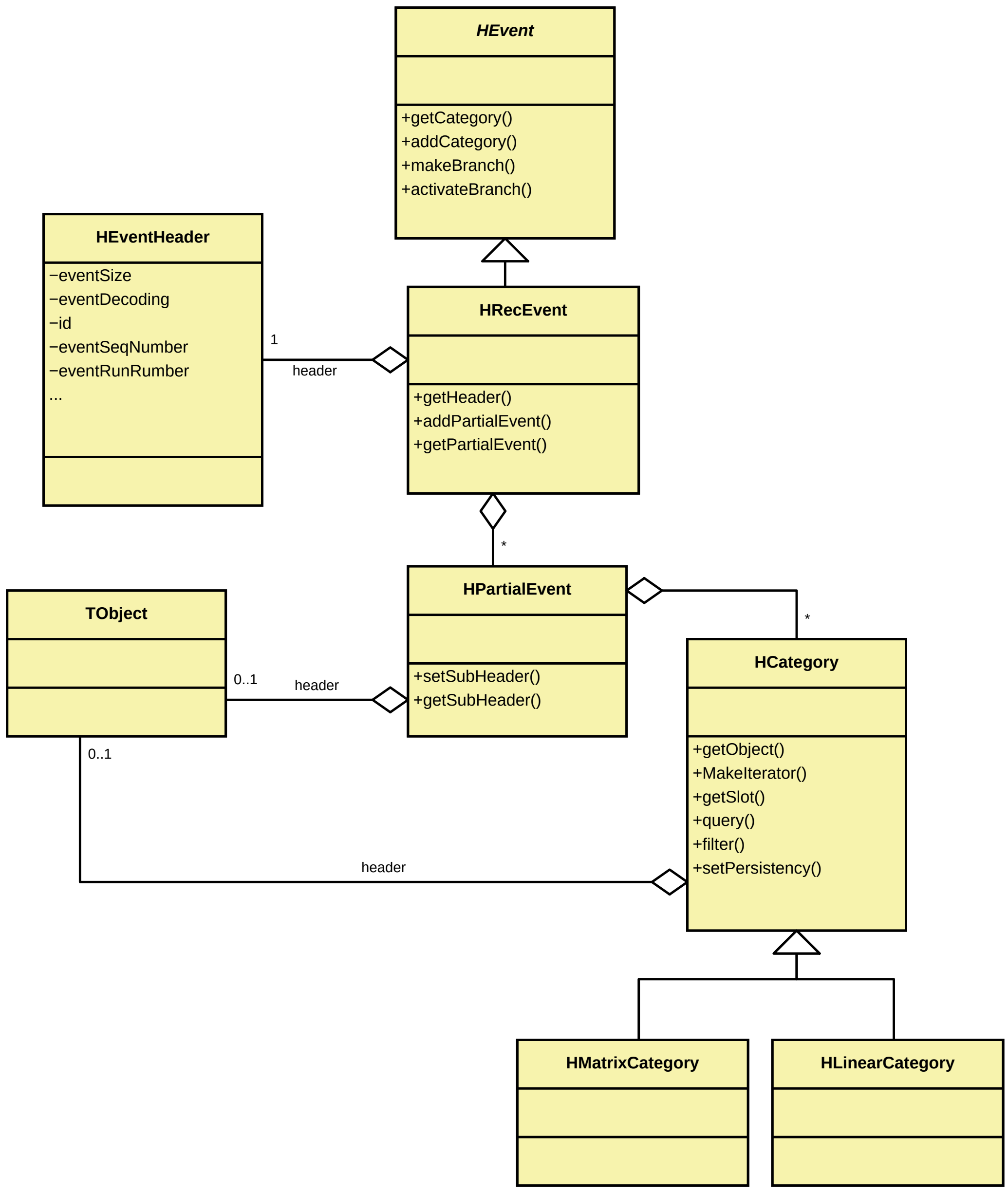


Figure 3.2: HRECEVENT class diagram

related to the Rich detector go to one partial event, those related to the Mdc go to a different one and so on.

Each partial event is itself an HEvent and therefore implements all the methods defined in that interface. The HPartialEvent physically contains a linear array of the managed categories plus a header.

The simulated event

Simulated events are events produced by the Hades simulation package: HGeant. Simulated events can be used as input for the reconstruction program instead of real ones in order to test the software. Therefore, simulated events must conform to the same structure as real ones.

On the other hand, a simulated event holds more information than a real one; in particular in a simulated event we know the collision kinematics, or the dilepton tracks, for example. Therefore, the simulated event class needs to hold more information than what is available in a real one.

No new event class is needed for simulated events. What is done instead is add to a HRecEvent one more partial event, called *Simul*, which stores extra information coming from the simulation. This is however not enough, it only allows to store extra information on a per event basis. We need also to do it on a per hit basis. That is, for every data object, i.e. a HMdcCal object representing one measurement in one of the Mdc, we need to store extra information coming from HGeant. This is accomplished by defining daughter classes, HMdcRawSim, storing the extra information. That way the information is there; but standard algorithms, using the parent classes, can ignore it. Similarly Sim versions of the algorithms can be created if something should be done about the extra information.

3.5.2 The categories

A category is an object container. What makes categories special is that all objects within a category have to instantiate exactly the same class. For example, the raw data objects for Mdc can be stored in the same category, but raw data objects in the Rich have to go to a different category since they instance a different class.

With that definition, there are several possible implementations for a category depending on how the data objects are internally stored. However, all of them must show the same interface to the programmer for accessing the data; this is enforced by making all categories derive from the HCategory class, which is abstract. In other words, HCategory defines the interface

Algorithm 1 Using HCategory::getObject() to retrieve a data object from the catMdcRaw category.

```
//get pointer to a category
HEvent *ev = Hades::instance()->getCurrentEvent()
HCategory *cat = ev->getCategory(catMdcRaw);
HLocation loc;      //Declare a HLocation object
loc.setNIndex(4);   //It's a location with 4 indexes
loc[0]=2;           //Set indexes to (2,2,1,4)
loc[1]=2;
loc[2]=1;
loc[3]=4;
//Get data object
HMdcRaw *raw=(HMdcRaw *)cat->getObject(loc);
```

that must be implemented by the different categories implementations. One implementation can still be a generic object container storing objects of any class, but only one class at the same time. In figure 3.2 you can see the HCategory's definition as well as some inherited classes.

The HCATEGORY interface provides functions to access the stored data objects. The access can be either random or sequential.

Random acces means we want to access one particular object in the category for which we know some kind of identifier. Each object in a category is identified by an array of integers. For convenience reasons, the integer array is presented as an object instantiating the HLOCATION class. There is a well known structure where each element is identified by an array of indexes: a matrix. Conceptually HCATEGORY allows access to the contained objects as if they were in a matrix. The corresponding function in the interface is HCategory::getObject(HLocation &aLoc). Listing 1 shows an example using that method. Typically each of the indexes is assigned a meaning, by convention. For example, for the catMdcRaw category, which contains raw data from the MDC, the number of indexes to identify one object is 4. The four indexes identify one wire out of the whole MDC system; the firs is sector, the second is module, the third stands for layer and the fourth corresponds to wire number

It is also possible to access one object giving only one index with the function HCategory::getObject(int idx); but in this case it is not possible to assign a meaning to the index.

Sequential access means we want a way to loop over a set of objects in the category; it can be all of the stored objects or a filtered set. Sequential access is performed using iterators, in the spirit of STL[8]. Iterators are objects created from the object containers (categories) themselves invoking the HCategory::makeIterator() method. This way, one iterator allows access to the contents of the category which created it. The methods for that access are specified in the HITERATOR interface. There is a method HIterator::Next() which keeps returning one object after the other in the container, until no objects are left and from there on it returns 0. Invoking the Next() function on an iterator until it starts to return 0 is called *iterating* over the category.

A typical situation is when we want to iterate over part of the category. For example, when doing tracking on one chamber we may want to iterate only on the raw data corresponding to that chamber. Let's think of a category, like catMdcRaw, where each object is identified by 4 indexes. In our example those indexes have the conventional meaning of sector, module, layer and cell number. We now want to iterate on all objects in the second sector; that is, we want to retrieve from the category all objects with location (2,*,*,*), where * represents any number. Such an scenario is supported with the HIterator::gotoLocation() method. See listing 2 for an example.

It may also happen that we want to access all data objects in a category passing a particular, arbitrary, filter. To access objects like that there is the function HCategory::query(TCollection *col, HFilter &filter). This function iterates over the category checking the filter for each object. Those that pass the filter are added to the data container pointed by the method's first argument. The second argument is the filter itself. In order to be completely generic the filter is an object defined by the user; the only requirement is that the object has to implement the HFILTER interface. HFILTER defines just one method HFilter::check(TObject *obj), which takes one object as argument and returns true if it passes the filter, false otherwise. It is also possible to use the HCategory::filter() function to delete all objects in the category which do not pass the filter.

Besides having objects stored in a category we must be able to add new objects to it. This is done in two steps, in the first step the users calls the HCategory::getSlot() method to obtain a slot where to instantiate a data object. In a second step, the object is instantiated in that slot; by instantiating an object in a slot given by the category, the object itself is added to the category. In fact, a slot is simply a memory region and instantiating in a slot means using the "new with placement" operator giving the slot address as argument. This operator is an overloaded version of the standard "new"

[8]STL stands for Standard Templates Library and is standardized along with C++. It defines object containers as well as algorithms to operate on them and iterators (see (7) for a recent review).

Algorithm 2 Various forms of sequential access on a category

```
//Retrieve a pointer to the category
HEvent *event = Hades::instance()->getCurrentEvent();
HCategory *cat = event->getCategory(catMdcRaw);
//Iterate over the whole category
HIterator *iter = cat->MakeIterator();
iter->Reset();
HMdcRaw *tmp;
while ( (raw=(HMdcRaw *)iter->Next()) != 0) {
  tmp->Dump(); //Print the object
}
//Iterate over sector 2
HLocation loc;
loc.set(1,2); //Set a 1 index location with loc[0]=2
iter->Reset();
iter->gotoLocation(loc);
while ( (raw=(HMdcRaw *)iter->Next()) != 0) {
  tmp->Dump(); //Print the object
}
```

operator from C++, which takes as an argument the memory address where to instantiate one object. The procedure is clearer by looking at one example, like the one in listing 3. getSlot() requires that the full location where the object will be stored is known beforehand; in case this is not possible the method HCategory::getNewSlot() should be used instead. Both this two methods will return a pointer to the allocated slot or 0 in case no slot is available.

By doing things this way the category is responsible for memory management. The main reason to let the category do the memory management instead of simply using the C++ "new" operator comes from the high number of data objects instantiated per event, and the high number of events to process. The "new" operator calls an expensive routine in the operating system to get the requested memory. However, a category can have a preallocated block of memory for the data objects which are going to be instantiated; this can speed memory management up because the category knows beforehand the size of the data objects which are going to be instantiated, as well as the kind of memory requests it will be asked for[9]

[9]This scheme is inspired by ROOT's TCLONESARRAY class.

Algorithm 3 Adding an object to a category.

```
//Get pointer to the category
HEvent *event = Hades::instance()->getCurrentEvent();
HCategory *cat = event->getCategory(catMdcRaw);
//We want to add one object in location
//2,2,1,3
HLocation loc;
loc.set(4,2,2,1,3);
//Get the slot
TObject *slot=cat->getSlot(loc);
//Instantiate the object. Now raw points to the new object
HMdcRaw *raw=new(slot) HMdcRaw;
```

.

The following sections describe two implementations of the HCategory interface, focusing on the internal structure of the data.

The HMatrixCategory

This kind of category stores data objects in a matrix-like structure. In this way, when we ask for an object in the category, the location indexes which identify the objects are the same as the indexes of the underlying matrix. To initialize a matrix category one needs to provide the following data in the constructor:

- Number of indexes in the matrix
- Maximum value for each of the indexes (that is, matrix dimensions)
- fillRate: this is a number between 0 and 1 which corresponds to the maximum fraction of occupied locations we expect.

Looking in more detail into this category's implementation we don't see the mentioned matrix anywhere. That's because in practice, the data objects are stored in an array (a TClonesArray from ROOT).

The internal structure of the category is as follows: on one side we have a TClonesArray A with every data object, and we have a HIndexTable object T which behaves like a matrix of integers. When we are looking for an object associated to a location, we get from T the matrix element corresponding to the indexes of that location; this matrix element is an integer giving the position in A of the desired data object.

In this way, it is not needed to get for A all the memory which would be used if every location was full; and we can keep the TClonesArray without holes. Not having holes is critical when we want to store the array to an output file.

We have already said that HIndexTable behaves as an integer matrix. However, again, we can see that internally we have an array of integers. This is done like that in order to be able to work with an arbitrary number of indexes in our matrices.

Helper classes have also been implemented to compress the index table on the fly, saving I/O time.

The HLinearCategory

This is the simplest kind of category we'll speak about. In fact, it is nothing more than a lightweight wrapper around a TClonesArray so it can be used within the framework.

Therefore, this category is only able to work with locations consisting of just one index which corresponds to the position of the data object in the underlying TClonesArray.

This category can be useful in a variety of processes like calibration. Let's say, for example, that we want to go from raw data in Mdc to calibrated data; each raw data object is identified by four indexes (sector, module, layer, cell). The first step is to read data from the acquisition system and place them in the "catMdcRaw" category. After that the data are calibrated.

In this example, one possibility is to place the data in the category without any order, using a HLinearCategory, and store the four indexes as a data member of the data object. Later, during calibration, we iterate over all data objects, and for each of them we do the calibration with the parameters specified by the indexes stored in the data object itself. That way we may afford the overhead introduced in the HMatrixCategory by the index table for this particular application.

3.6 Data input

In order to deal with the different data sources which are possible a generic interface is defined, this is HDataSource. That interface is used by the

HADES class to request any particular implementation for event data. Different implementations of HDATASOURCE have been realized trough daughter classes. Each implementation, or back-end, is able to read information from a different place, in a different format.

The most important method in HDATASOURCE is HDataSource::getNextEvent(). When this function is called, one new event is read from the data source into the event structure. The returned value of the operation can be one of the following:

kDsOk: the event was successfully read

kDsEndFile: we have reached the end of one run (set of data with the same reconstruction parameters), but more data are available

kDsEndData: we have reached the end of the data source

kDsError: error

The HADES singleton can combine up to two data sources[10].

Another important function of the HDATASOURCE interface is the method HDataSource::init(), used during initialization. Within this method, each particular data source must check whether an event object exists or not and, if it doesn't exist, then it is the data source's responsibility to instantiate an event object. Usually, the data source will also have to add to the instantiated event object those categories where data will be read. Note that if an event object or the needed categories exist, then the data source is not allowed to destroy them.

In the following sections we will review the most important data sources at use in HYDRA.

3.6.1 Input from the Data Acquisition System: HLDSOURCE

HLDSOURCE is the base class for those data sources reading data in the format produced by the HADES acquisition system (DAQ). The class structure can be seen in figure 3.3.

The HLDSOURCE reads raw data in the order and format provided by the DAQ and places them within the event structure; this usually implies some

[10]The second data source is used to support the embedding of simulated tracks into real data. The alternative would be to define a new data source performing the merge.

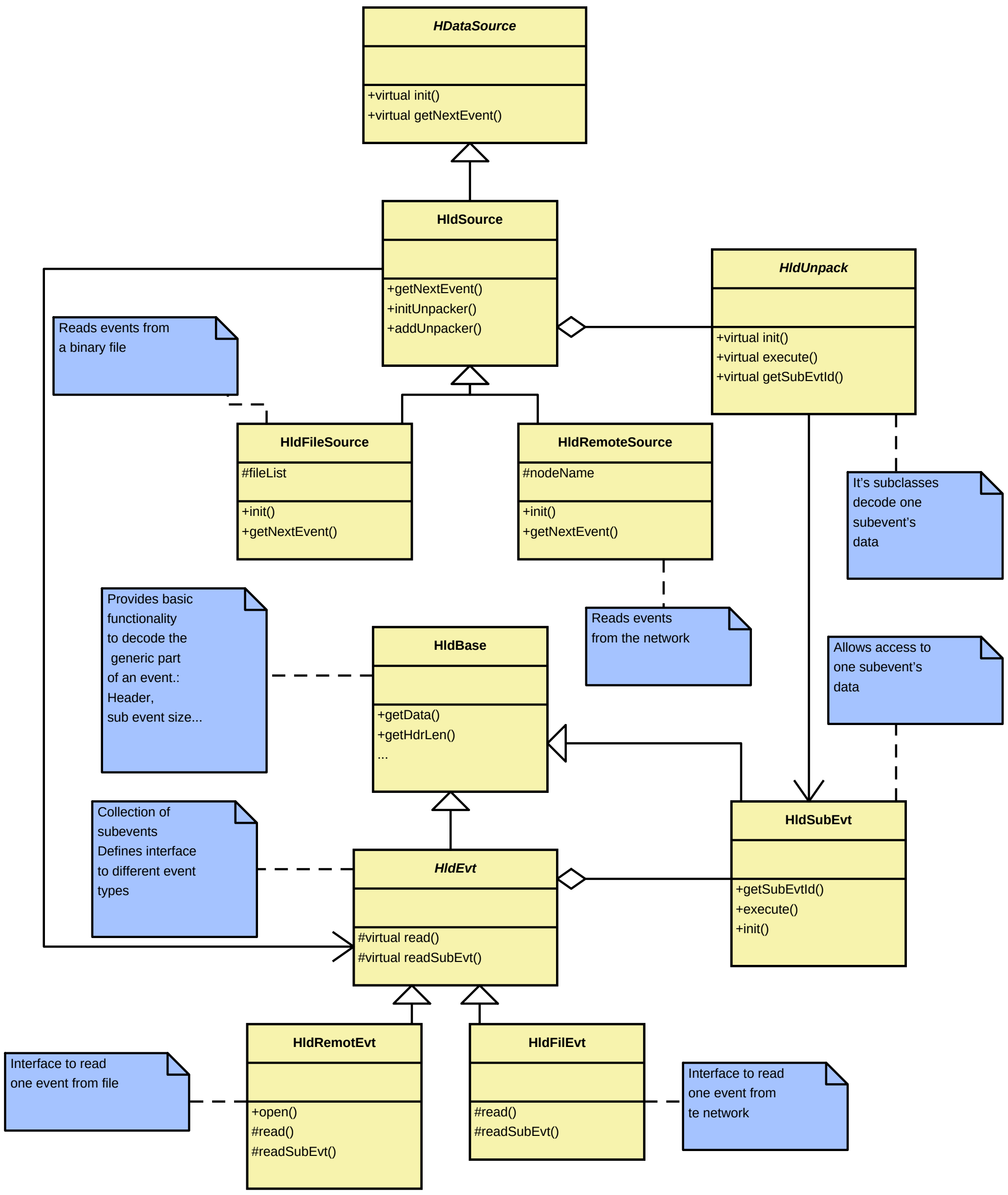


Figure 3.3: HldSource

reordering of the data. That process is known as unpacking and is realized by unpackers within a HldSource. Each unpacker is itself an object of a class inheriting from HldUnpack.

HldUnpack is an abstract class from which several different unpacker classes are inherited, such as HRichUnpacker or HTofUnpacker. In fact, we have one different unpacker class for each detection system in Hades; each of them only understand the binary format for a particular kind of data. What happens is that each even from the Daq is divided in sub events; each sub-event having an identifier which is used to decide which unpacker should be used. The most important method of this class is the HLdUnpack::execute() function, where the unpacking process is realized. Another important function is HldUnpack::init() which is used during the initialization procedure, as we will see in section 3.9.

The HldSource maintains a list of active unpackers at a given time[11]; this list is configured at runtime during initialization. Only the sub-events corresponding to the active unpackers are actually unpacked; thus supporting cases where, for example, not all detectors are available; or we are just not interested in them.

Thus far the general information about a HldSource. However, in practice we will always use one of its subclasses like HldFileSource or HldRemoteSource. Both of them work in a very similar way, being the main difference the fact that the first one reads data from a file and the second one reads data from a network connection to the Daq machines. Therefore, the first one is more suitable for offline analysis, while the second one is more suitable for on-line analysis.

Note that the unpackers used both in HldFileSource and HldRemoteSource are exactly the same.

Reading process

Here is what happens when the HldSource::getNextEvent() function is called:

1. A buffer is filled with the information to be unpacked. This buffer is a HldEvt object inheriting from HldBase; it stores generic information about the read event (event number, bytes used ...). Each HldEvt is made of sub-events, HldSubEvt objects, which are read at the same time as the HldEvt.

[11]This list is built by the user himself in the initialization of the HldSource using HldSource::addUnpacker()

2. The HldUnpack::execute() function is called for each of the active unpackers. Each unpacker has an associated HLDSUBEVT where it reads data from, transforming those data into objects and placing them into the event structure.

3.6.2 Partially reconstructed data: HROOTSOURCE

In this case, the data source is a ROOT file holding an event tree. Usually this tree has been generated by the reconstruction program itself and holds completely or partially reconstructed events.

Besides the generic functionality required from HDATASOURCE. The HROOTSOURCE is able to dynamically select if the whole event from the input file should be read, or just part of it. It is also possible to configure the data source such that not all events in the input file are read, but just a subset of them.

3.7 Reconstruction parameters management

The reconstruction parameters include all those numbers needed for the reconstruction process, like for example, positions or dimensions of the detectors, calibration parameters, pattern recognition parameters, etc. These numbers are organized in functionally related sets. Each of this sets is represented by a subclass of HPARSET known as a parameter container. The generic interface provides methods used by the framework to initialize the parameter containers and manage parameter versions.

The interface class responsible for managing parameter sets in HYDRA is the HRUNTIMEDB class. As the name suggests, it implements a runtime database. Among its responsibilities we find the management of parameter versions.

Each set of parameters can have different versions since they can change with time. Actually there are two time axis in which they change. One is the real time, the parameters for the data taken one day do not need to be valid for the data taken the day after; detector positions could have changed, for example. We define *run* as a set of events taken in a time range where parameters have not changed. Each run has a unique identifier which is used by the parameter management subsystem to determine what parameters should be used when analyzing a particular set of data.

The other time axis has to do with people improving the quality of parameters with time, providing new versions which make previous ones obsolete.

Proper version management is needed in order to know what parameters to use on each moment. It may be needed to reproduce the analysis results obtained at a past point in time and for that reason the history of parameters is kept. There is a function to force the framework to use the parameters for a historic date instead of the current one. The default behavior is to take the latest version which has been declared valid and belongs to a "parameter release". The reason for doing so instead of just taking the latest version, is that parameters in a release are cross checked for consistency among the different groups producing them.

Several versions of the same parameter container can be valid at the same time, for example they could contain cuts for the reconstruction algorithm useful for different analysis, like minimum bias. This possibility is foreseen through the so called contexts. Each parameter container can be assigned a context so two versions of the same container can be valid at the same time if they correspond to different contexts.

In addition to versions the runtime database has to manage parameter input and output. The parameters may come from different sources, each source being implemented as a subclass of HParIo and the runtime database keeps three of them. The HParIo class defines the generic interface every subclass has to implement, allowing the runtime database to read and write specific versions of parameters. There are currently three back-ends available:

HParOraIo: Interfaces to an Oracle database residing at the central institute, Gsi. When this back-end is used, the user is sure that she gets the canonical set of parameters at any given time.

HParAsciiFileIo: Allows storage and retrieval of parameters from Ascii files. This kind of files is convenient in cases where the user wants to change the parameters by hand for experimentation. The main limitation is that Ascii files may only store one version of the parameters per file.

HParRootFileIo: Implements storage and retrieval of parameter from binary Root files. Root files have the advantage over Ascii that they may contain different versions of the parameters. The disadvantage is that it is not easy to modify parameters in a Root file. Compared to the Oracle interface, the advantage is that there is no need for an open network connection to the central server. The disadvantage is that the history of parameters is not stored.

In the following sections we will spend some more time examining in more detail the HParIo interface and the HRuntimeDb class.

3.7.1 Input and output: HParIo

A HParIo is essentially an array of HDetParIo objects. The HDetParIo class defines the generic interface used to actually read and write the parameter containers for a code module. A code module is a set of algorithms under the responsibility of a group. Examples would include the code module implementing tracking on the Mdc or pattern recognition in the Rich. This division is foreseen so that changes by one group of developers do not affect the work of the other groups. There are two basic methods in the HDetParIo interface:

init(HParSet *par, Int_t *set): Fills the parameter container pointed by "par" for the detector modules indicated by "set".

write(HParSet *par): Writes out the parameter container pointed by "par".

The way down to the concrete implementation of these functions has two levels. On the first level, two new classes are defined by subclassing HParIo and HDetParIo for a particular data source. For example, for Root files we would have classes HParRootFileIo and HDetParRootFileIo. The first of them handles generic questions about the particular source, while the second one handles those details which can differ from one code module to the other.

The second level of concretion consists in defining a HDetParRootFileIo subclass for each particular code module. For example HMdcParRootFileIo or HRichParRootFileIo. Each of these subclasses has a init() and write() functions for each supported parameter container.

When the runtime data base needs to read or write a container it calls the init() or write() functions of all registered HXXXParYYYIo classes until one recognizes the parameter container. Where XXX stands for code module name and YYY stands for parameter source. Values for XXX would be: Rich, Mdc, Tof, Shower and so on. Values for YYY are Root, Ascii and Ora.

3.7.2 The runtime database

The structure of the runtime database is shown in figure 3.4. Essentially it consists of 3 HParIo objects, a list of parameter containers (HParSet objects) and container factories.

Each container in the runtime database is identified by a name. The method HRuntimeDb::getContainer() can be used to retrieve a pointer to

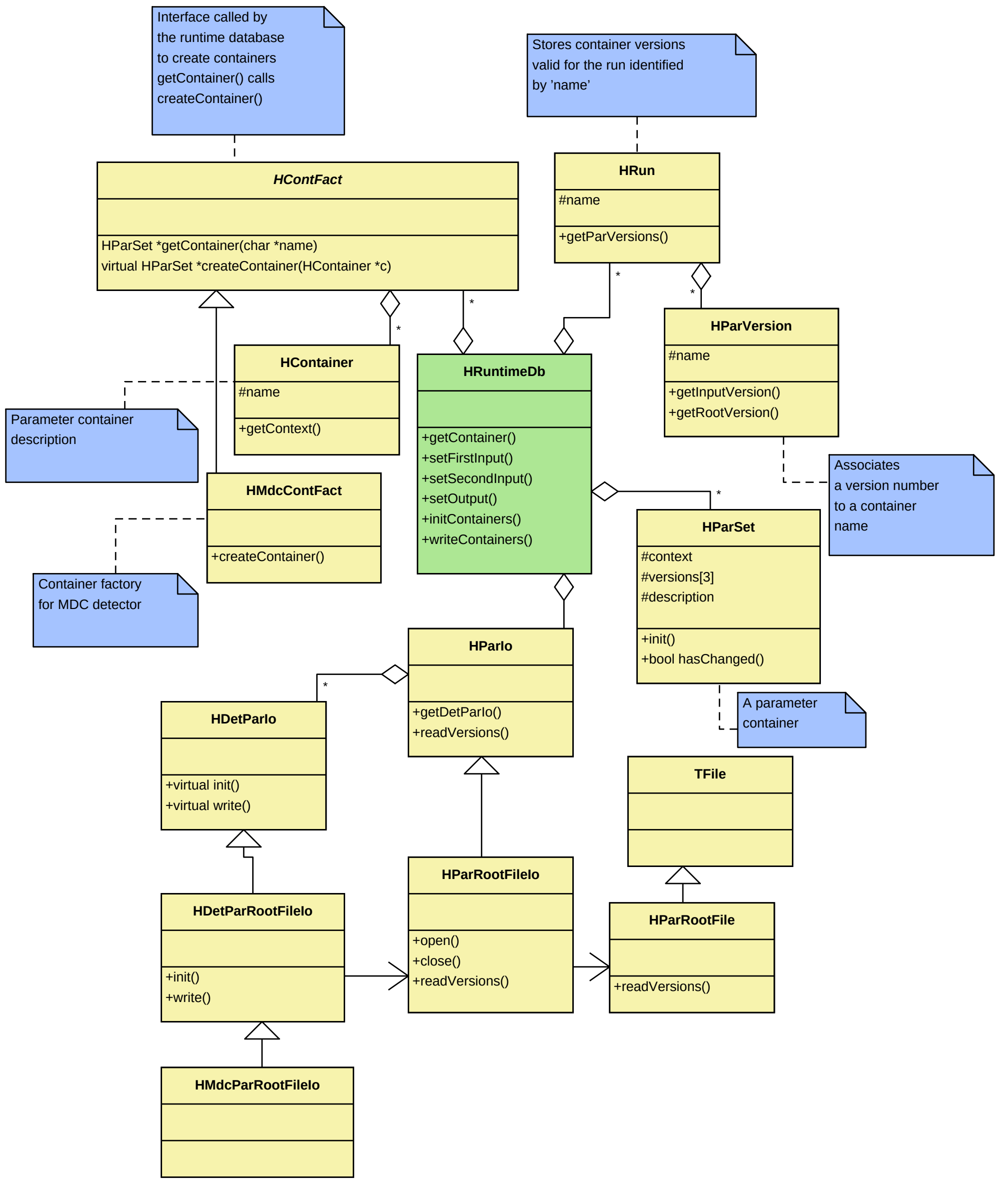


Figure 3.4: Runtime Database structure

a container given its name. In case the requested container is not in the database, it will try to instantiate the container automatically. This is done by iterating on the registered container factories requiring the instantiation of a container of the given name, until one of the factories recognizes the name and succeeds.

The container factories are added to the runtime database during startup. Each code module instantiates a container factory as a static variable. This uses the HRuntimeDb::instance() and HRuntimeDb::addFactory() in its constructor, to register itself in the runtime database.

Out of the 3 HParIo objects, two correspond to inputs, one primary and one secondary input, and the third object corresponds to the parameter output, if any. Having two inputs has the advantage that if some data are not available on the first one, the database will look for data on the second input before returning an error. This is specially useful to combine part of the data from a local file with parameters from Oracle. For example, a developer wants to test her new Rich parameters before changing the official version, but she still wants to get the official parameters for Tof or Mdc.

The runtime database also supports marking containers as static with the HParSet::setStatic() method. When a container is made static it will never be updated, overriding the version management.

3.8 Task management

One of the requirements we've seen in the previous chapter was to have a flexible system allowing us to select which algorithms to use for event reconstruction, as well as in which way those algorithms are going to be combined. In general we will call *task* to a process executed event per event.

Tasks are implemented as subclasses of the abstract class HTask. The HTask interface defines what our understanding of *task* is. It has a function to execute the task; this function is called HTask::next(int &errCode) because it not only executes the task but also returns a pointer to the next task to be executed. Each task has several possible output slots identified by numbers; the different output slots represent different return codes. Any other task can be connected to any of the output slots, defining the work flow. This way of connecting tasks makes it possible to define non linear execution paths. The method to connect tasks is HTask::connectTask(HTask *task,int n).

For example to decide between two execution paths depending on a condition we could have a class with two output slots which checks that condition.

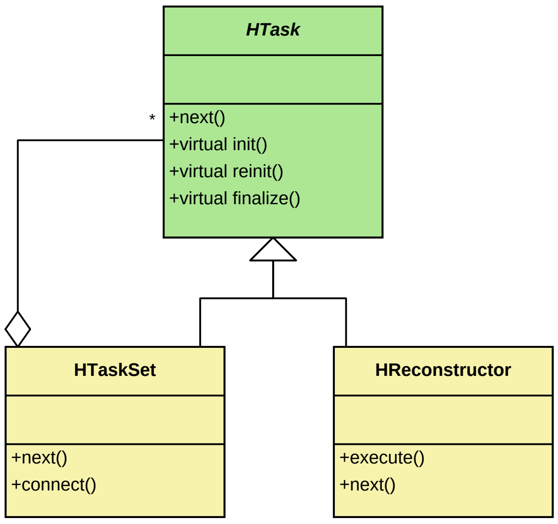


Figure 3.5: Tasks classes

Depending on the condition being true or false, one output slot or the other is chosen.

Usually tasks need access to the event structure and parameter container in order to perform their job. The init() function is where a task asks the framework for those kind of things, as we will see when explain HYDRA's initialization in section 3.9. The symmetric of init() is finalize() which should be used to perform cleanup after all events are processed, or to perform some final computations.

There are two kind of predefined tasks in HYDRA, the HRECONSTRUCTOR and HTASKSET. The first is the one subclassed by developers to implement new algorithms in HYDRA. This class implements the next() method and defines a new function, execute(), to be overridden by subclasses. It is this function the one which gets called by the framework for every event. The execute() method returns an integer. If it is less than 0, it is considered to be an error code and treated accordingly. If the integer is larger than 0 the actual number is used to select which output slot the task should go to next.

The HTASKSET is a way of grouping several tasks together, as the name suggests. There are several ways to add tasks to a task set:

- HTaskSet::add(HTask *t): is used to add one task after the other. All tasks in the task set are executed in the order they are added.
- HTaskSet::connect(HTask *t, HTask *where, int n): is used to connect the task "t" to the n^{th} output slot of task "where". The task "where" has to be already in the task set.
- HTaskSet::connect(HTask *t): is used to connect task "t" as the first one in the task set.

Each task has a name and for convenience there are equivalent methods to the ones above, but taking task names (strings) instead of pointers. A task set is the owner of the tasks in it, that means the task set and not the user is responsible for deletion of the tasks. The Hades class has several predefined task sets: for simulations, for real data, for calibration etc. Therefore, since all tasks are inside a task set, the user is never responsible for deletion of those objects.

It should also be noted that HTaskSet is a Htask itself, making possible to build recursive structures.

3.9 Hydra's Initialization

In this section we will describe how the different systems interact during the program's initialization. This will serve to indicate what does the user need to specify as input and what support are the developers supposed to give in their code. In particular, we will spend some time describing how an algorithm, HReconstructor, interacts with the initialization scheme to obtain the information it needs.

The system is initialized from settings given by the user in a macro. This macro is written in C++ and interpreted in runtime by Cint[12]. Thus the settings are specified instantiating a few classes and directly calling their functions. This has the advantage that the user is allowed to interact with every part of the framework during initialization, allowing a high level of customization. In any case, the basic settings given by the user are:

- What detectors are going to be used. Both what detector systems, like Mdc, Rich, etc. and which detectors are active on each detector system.

[12] Cint is a C++ interpreter integrated within Root

- What inputs, up to a maximum of two, are going to be used for reading reconstruction parameters with the runtime database. This includes both the kind of input and the physical location of the data if needed.
- What output to use for parameters. Both kind and physical location.
- The source where data will be read from. Both the kind of data source and physical location.
- The list of tasks to be performed per event
- What output file to use, if any. It is also possible to create an output file without event data.
- What part of the event structure should go to output. It can be all, a subset or nothing.
- Optionally the user can also specify
  - A specific version of the reconstruction parameters to be used. Overriding the automatic versioning mechanism.
  - A context for parameter containers. The context serves to differentiate between parameter sets which are valid for the same time range at the same date. This may happen when different parameters respond to different analysis objectives.
  - The type of event and the types of categories used to store the data. This is not needed in most of the cases.

Whenever the user sets something, that setting overrides the default behavior. As discussed, the user has a large freedom when setting up the analysis; that means she can do things wrong. It is considered that the default settings are safe, setting other variables besides the ones described above is done at the user's risk.

The order in which the different inputs are set is relevant. Setting up the spectrometer configuration should be the very first thing being done; then the output file should be set. Next would be parameter inputs and output, task list and finally what data to write. An example can be seen in listing 4

3.9.1 Spectrometer configuration

The spectrometer configuration is stored in a HSPECTROMETER class within HADES singleton. During initialization the user is supposed to access that object and add to the active detectors. Each detector is represented by an object of the HDetector class; each of theses objects contains the geometrical information for one detector subsystem. To access the spectrometer object the following line of code can be used

```
Hades::instance()->getSpectrometer()
```

To add a particular detector, the HSpectrometer::addDetector() method is used, giving a pointer to the detector object.

3.9.2 Data base initialization

During the database initialization what the user sets is:

inputs: At least one input must be set, but the user can set up to a maximum of two. To set one input, one only needs to create the corresponding Io object (see 3.7.1) and use either setFirstInput() or setSecondInput() methods from the HRuntimeDb class.

output: The procedure is the same as before: create a HParIo object and call HRuntimeDb::setOutput(), giving a pointer to the object as a parameter.

3.9.3 Tasks selection

As already mentioned, the Hades singleton contains several master task sets for different purposes, like calibration, simulation or real data processing. In order to specify what tasks to perform, the user needs to instantiate the concrete objects implementing those tasks and add them to the corresponding task set.

Since the analysis of a detector subsystem may involve several tasks and the user does not necessarily know about all of them or their order, convenience classes have been implemented to assist the user. Each code module defines one such class, like HMdcTaskSet or HKickTaskSet. These classes define a static function make() which can be called giving as an argument a string of options. The options accepted by each of the classes differ; but what is common is that the result is a full HTaskSet containing the tasks needed to perform a higher level process.

3.9.4 Selecting the data source

Selecting a data source is as trivial as instantiating an object for the chosen data source kind and activate it as the current data source using the setDataSource() method from the Hades class.

Obviously, each data source can get some different initialization parameters. As, for example, the server's IP direction if reading data from Internet, a file name, or nothing. Since our configuration file is a C++ macro, it is enough to call the functions specified in each data source's documentation to set those parameters.

3.9.5 Initialization internals

We will explain what are the steps taken by the framework during initialization. That will serve to introduce the places where algorithm programmers are supposed to supply information and which information should be supplied. The procedure starts when Hades::init() is called.

1. The runtime data base is initialized
   In this step, the Hades class calls functions in the runtime data base to initialize the version management with the first run from the input. This is needed because other steps of the initialization procedure may require that the runtime database is active.

2. The spectrometer is initialized
   In this step the init() function is called for HSpectrometer and all of the active HDetector objects. Those functions are responsible for setting up basic stuff like their respective classes for parameter Input/Output.

3. Initialize data sources
   The function init() is called for each data source. Within this function the data source is responsible of creating an event structure if none is active. It also must add the categories it needs to the event structure and the containers it needs to the runtime database.
   In the case of the HRootSource all information is taken from the input file. However, the HldDatasource and derived classes do it by calling the init() function from the active unpackers (see 3.6.1). The actions that need to be taken during the processing of HldUnpack::init() are the same as the during HReconstructor::init() and will be explained later.

Algorithm 4 Example of initialization macro

```
//
//Setup spectrometer
HMdcDetector *mdc = new HMdcDetector;
Int_t modules[4] = {1,1,1,1};
Hades::instance()->getSetup()->addDetector(mdc);
//
//Set runtime database to read from Oracle
HRuntimeDb *rtdb;
rtdb->setFirstInput(new HParOraIo);
//
//Setup task list. Full MDC reconstruction
HTask *mdcTask = HMdcTaskSet::make(‘“’,’’simulation’’);
//
//Add task to master task set
HTaskSet *master = Hades::instance()->getTaskSet(‘‘simulation’’);
master->add(mdcTask);
//
//Set output file
Hades::instance()->setOutputFile(‘‘test.root’’);
//
//Standard stuff. Intstruct the framework to initialize
Hades::instance()->init();
//
//Post initialization. Store in output all data levels
//but callibration. First we switch off output for
//the category corresponding to callibrated data
HEvent *ev = Hades::instance()->getCurrentEvent();
ev->getCategory(catMdcCal1)->setPersistency(kFALSE);
//
//Next the output is set up.
Hades::instance()->makeTree()
```

Algorithm 5 Adding the catShowerCal category from a HReconstructor::init()

```
HEvent *ev = Hades::instance()->getCurrentEvent();
HCategory *cat = ev->getCategory(catShowerCal);
if (!cat) {
  HSpectrometer *spec = Hades::instance()->getSetup();
  HShowerDetector *det = spec->getDetector(“Shower”);
  cat = det->buildCategory(catShowerCal);
  if (!cat) return kFALSE;
  else ev->addCategory(catShowerCal,”Shower”);
}
```

4. Initialize tasks
   The init() function is called for all HTask objects active in the program including all HReconstructor objects which are active. Within this function the HReconstructor should

   - Get the pointers to the categories it needs from the event structure. If some category is not there, then the task is responsible for adding it to the event structure. See listing 5 for an example.
   - Get pointers to the parameter containers it needs from the runtime database. The runtime database automatically tries to add containers that are not registered, so the user doesn't need to do anything else here. There is one important thing to notice at this point; the task only gets pointers to the containers. The containers are not yet initialized! and the task is not allowed to initialize them. For that purpose the function reinit() is foreseen. This function's context is presented in section 3.10.
   - Perform specific initialization tasks. Things that can be initialized here are those which do not change from one run to the other.

3.10 Event processing

In this section we will see in more detail how the loop on events is realized. This is, in fact, an explanation in pseudo-code of the Hades::eventLoop() implementation. Reading the source code for that function is recommended for anyone wanting to gain a deeper understanding of the details. This function does the following:

1. Ensure there is a current event, that is, an event structure and a data source.

2. Clear the event structure.

3. While the number of processed events is less than "nEvents" and the data source doesn't return an error code or and end of data code:

   (a) Re initialize the task list by calling reinit() on all tasks.
   At this point, the task may assume that all parameter containers are properly initialized for the run currently being processed. So, this is the point where to make all those calculations and initializations which require the parameter containers to have meaningful data.
   Maybe the task needs to go through some heavy initialization procedure which depends on parameters supposed not to change from run to run. For example, for the tracking routines many calculations are done during initialization which only depend on the cell geometry of the MDC chambers, which is hopefully not changing every few minutes. In such cases, to avoid foolishly repeating the initialization procedure, one can check the return value of the container's hasChanged() function and if it is kFALSE nothing needs to be redone. In fact, the hasChanged() function is cheap, so it is fine to use it pervasively.

   (b) While the number of processed events is less than "nEvents" and there are data in the current run.

      i. Read a new event from the data source
      ii. Execute the task set for the current event
      iii. Fill the output ROOT tree if one exists
      iv. Clear the event structure

4. Check if the data source has returned an error code and notify it.

3.11 HGeant simulation packages

HGEANT is a simulation package for HADES written in FORTRAN and built upon the GEANT (see (3)) program from CERN. The purpose of HGEANT is to simulate the detector response of the HADES spectrometer to the passage of charged particles.

This means HGeant is not responsible of simulating any heavy ion collision. For that purpose, external programs, called event generators, are used. The event generator output is a file with the description of particles coming out of a collision; HGeant reads that information and tracks the particles through the spectrometer.

Geant itself is a framework to describe the passage of particles through matter, integrating functions to calculate the effect of processes like energy loss, multiple scattering and others. What HGeant does is extending Geant, implementing the functions left open by Geant in order to make a complete program.

- HGeant provides a way to read configuration options governing the way in which the different physical processes are treated within Geant.
- It also provides a way to read event data, that is, the list of particles which need to be tracked in one event. There are several possibilities to do this in HGeant. Data can be read from a file, obtained from an event generator, for example. Data can also be generated on the fly through the embedded event generators. The last option is to generate the tracks from the C++ interpreter embedded in HGeant, either in an interactive session or with a macro. In particular, the macro option is very useful to quickly prototype dedicated event generators for specific purposes.
- It provides a way to read the geometry describing the spectrometer; this includes both the geometrical shapes of the different detectors as well as their material composition. This geometry information is passed to Geant, which in turn uses it to simulate the different effects affecting particles passing through that matter.
- It provides digitization routines. These routines are called by Geant when specific volumes, called active volumes, are entered or left by a particle. In these functions, HGeant performs a basic simulation of the detector's response, mostly recording relevant properties of the traversing particle to be used in a more detailed digitization.

HGeant uses the Hydra facilities to write its output. This has the advantage that the files generated by the simulation are directly readable from the analysis.

As already noted, HGeant mostly records information about the particles traversing the different detectors. This information is then read by the analysis and processed by the so called digitizers. A digitizer is just another HRECONSTRUCTOR object from the point of view of HYDRA and thus it can be managed as any other task.

The digitizers are responsible for the detailed simulation of the detector response to the passage of particles. Each of them takes care of the specific details of the detector it simulates. Using the output from HGEANT as input, a digitizer is able to reproduce the signals a detector would have recorded had the simulated event occurred in reality.

The digitizer's output looks exactly as calibrated signals from a detector, which can be analyzed with the standard algorithms in HYDRA. Furthermore, since we have information about the tracks originating those signals and we have a way of propagating that information through the whole analysis chain it is possible to compare the output of the full event reconstruction with the original input and study the differences to evaluate the reconstruction's quality.

Bibliography

[1] R.Brun et al. *PAW-Physics Analysis Workstation.* CERN Program Library entry Q121, 1995

[2] Rene Brun and Fons Rademakers. *ROOT - An Object Oriented Data Analysis Framework.* Nuclear Instruments and Methods A(389), 1997. See also http://root.cern.ch

[3] *GEANT – Detector Description and Simulation Tool.* CERN Program Library Long Writeup W5013, 1993

[4] Eric Gamma et al. *Design Patterns.* Addison-Wesley.

[5] M. Sánchez. *Diseño y Programación orientados a objetos de la reconstrucción de sucesos en el experimento Hades de colisiones núcleo-núcleo.* Diploma thesis, University of Santiago de Compostela, 1999

[6] ISO standard ISO/IEC 14882:1998

[7] Mathew Austern. *Generic Programming and the STL.* Addison-Wesley, 1998. ISBN 0-201-30956-4.

Chapter 4

Vertex reconstruction

The HADES experiment deals with collisions between two nuclei; the so called target and projectile. One of the relevant informations in such a collision is where in space did it take place?. That's the question we try to answer through the process of vertex reconstruction. Therefore, with the word "vertex" we refer to the point in space where the interaction between the two colliding nuclei took place.

The raw material used to find the vertex is a set of particle tracks; that is, the recorded trajectories of particles through the HADES spectrometer. But before the algorithm itself and its performance are presented, we will spend some time on its motivation.

4.1 Motivation

We can distinguish between two kind of vertex reconstruction: event-wise and across events. In the first case, the set of tracks is limited to the statistics available on one event; while in the second we can accumulate statistics across many interactions. Those two kind of vertex reconstruction are useful in different scenarios and, even though the main focus of the present work is event-wise reconstruction, the presented tools are useful for both of them.

This is easier to understand by having a look at the different situations to be addressed.

4.1.1 Segmented target setup

One of the problems in HADES is that thin targets have to be used, so that the leptons produced in a collision are not reabsorbed. On the other hand

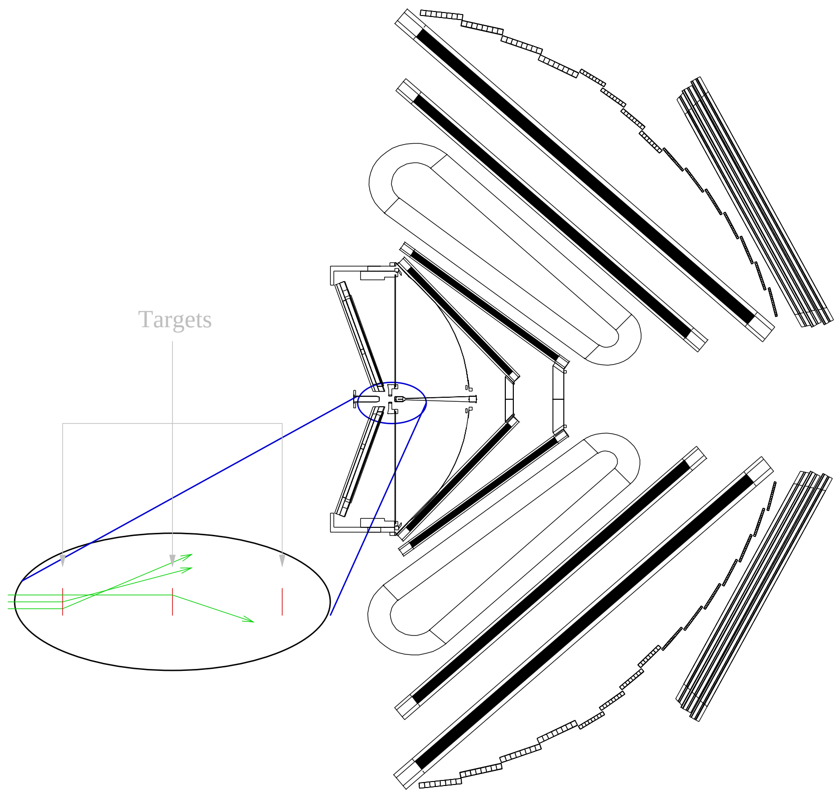


Figure 4.1: Segmented target setup

thick targets have the advantage of providing higher reaction rates.

The compromise solution is to build a segmented target. That is, lay several thin targets in a row (see 4.1); each of the targets being thin allows for the leptons to escape while keeping high reaction rates.

In such scenario an event-wise vertex determination informs about which segment was hit. This, in turn, would allow for a more precise vertex reconstruction, or could be used as a constraint in the track's fitting. It has even been suggested that one possibility is to use different materials for the different segments, thus effectively carrying out several experiments in parallel, minimizing the possible systematic deviations among them since all experiments are using exactly the same setup. In this case knowing the collision vertex is crucial to select the reaction under study.

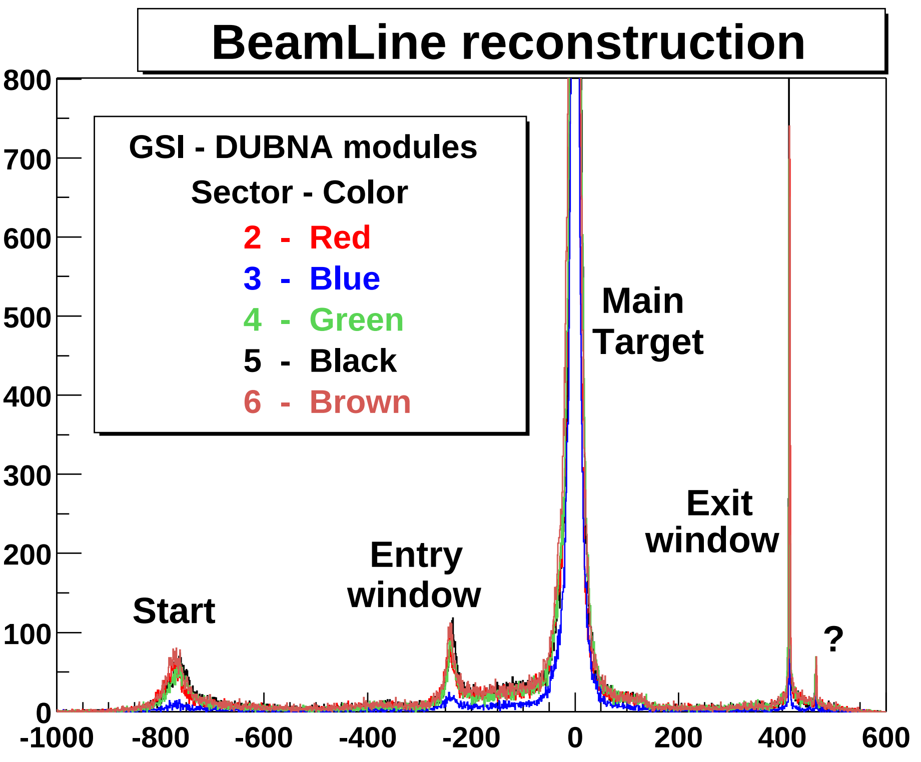


Figure 4.2: z coordinate of maximum approach of tracks to the beam axis. The different elements in the beam line appear as peaks as tracks are produced by collision of beam particles with those elements. The different colors correspond to the different sectors. Taken from (3).

4.1.2 Secondary targets suppression

It has been shown (4.2) the data contain a small contamination from secondary targets. Namely, interactions of the beam particles with detectors like START or VETO, or even the protection foils before and after the target. Obviously, those events are not interesting, since they typically correspond to reactions different from the one under study. In order to reject such events, an event-wise vertex reconstruction becomes handy.

4.1.3 Track fitting

The vertex position can be used as one additional point in the track fit; thus potentially improving the fitting resolution.

4.1.4 Alignment

Vertex reconstruction allows for the determination of the target position relative to the tracking detectors (MDC). This is very useful for alignment purposes (see (3)). In this case the reconstruction doesn't need to be event-wise.

4.2 Reconstruction algorithm

Vertex reconstruction is well known topic with several algorithms in the market. Most of them involve a certain form of a fit, either Least Squares or Maximum Likelihood, with the main differences in the functional to be fitted and how to perform the fit. However, other approaches like neural networks(5; 8) are also being explored.

Even within the methods based on traditional fits there are several approaches depending on the functional to be fitted. While most of them minimize some form of distance between the vertex points and the tracks, others like the dual approach (4) perform first a change in the parameter space. In particular the dual approach consists on using the fact that for a set of straight lines in two dimensions $y = a_i x + b$, with $i : 1 \ldots N$, if the lines are to cross on a given point (the vertex) given by $(x_{v,}y_v)$ then it can be shown that the a_i and b_i are linearly dependent and the vertex point can be obtained from the linear fit of the a_i with respect to the b_i.

In our case we have chosen to minimize the distance of tracks to a given point, which is directly 3D. That distance may be defined in several ways: it can be the 3D distance between a point and the track or we can fix the vertex plane and compute the distance on that plane. The vertex plane is the plane $z = z_v$, with z_v the zeta coordinate of the vertex. Both methods have been implemented with similar results. The one presented here minimizes the 3D distance.

The idea is that in order to estimate distance in any given plane it is necessary to extrapolate tracks to that plane, after extrapolation a symmetric error in the track's slope translates into a non symmetric error (and therefore not Gaussian) on the position in the plane. In fact, the error would still be symmetric if the plane was perpendicular to the track. That is the reason to use the 3D distance; in each case the extrapolation plane is perpendicular to the track.

Regarding the fitting routine itself, Kalman filtering (6; 7) has been widely used to treat the problem of outliers and multiple scattering within the tracking detectors. In our case multiple scattering happens mostly before the tracking (MDC) detectors and outliers have been treated by introducing weights into the Least Squares functional.

4.2.1 Least Squares Method algorithm

Assuming the interaction zone of the two colliding nuclei is point like, we can define the interaction vertex as the point of closest approach to all primary tracks in the event.

Such a definition already suggest to use the output of tracking as the starting point in the vertex fit. In other words, the input will be straight lines reconstructed using the MDC chambers. In general, that will only include the two inner ones, but in case the magnetic field is off all four chambers could be used.

Formally, what we are looking for is to find the point of closest approach to a set of N straight lines (tracks). Let track i^{th} be represented by a position vector $\vec{r_i}$ and a direction vector $\hat{\alpha}_i$, then each point in the track can be written in the form

$$\vec{x_i} = \vec{r_i} + \hat{\alpha}_i t, \ \forall i : 1 \dots N, \ t \in \Re \tag{4.1}$$

Let $\vec{r_v}$ be the vertex position in space. Then, the distance of this point to a given straight line can be computed as:

$$d_i = |(\vec{r_i} - \vec{r_v}) \times \hat{\alpha}_i| \tag{4.2}$$

Summing up to all lines and normalizing by the error we get

$$Q^2 = \sum_{i=0}^{N} \frac{d_i^2}{\sigma_i} = \sum_{i=0}^{N} \frac{|(\vec{r_i} - \vec{r_v}) \times \hat{\alpha}_i|^2}{\sigma_i} \tag{4.3}$$

The vector $\vec{r_v}$ which minimizes Q^2 is the point of closest approach to all tracks. This way the problem has been reduced to a simple Q^2 minimization; which, if we assume the σ_i to be constant, can be solved analytically. In fact the σ_i are not constant, they depend on the vertex position, which is what we are trying to minimize, and do so in a different way for each track. The corrections needed to account for that fact will be presented later.

In order to do such a minimization we need to compute the partial derivatives of Q^2 with respect to the parameters and equal them to zero. That is, we need to solve $\frac{\partial \chi^2}{\partial x_v} = \frac{\partial \chi^2}{\partial y_v} = \frac{\partial \chi^2}{\partial z_v} = 0$.

These derivatives can be obtained from the expression of d_i

$$\begin{aligned} d_i^2 = &[(y_i - y_v)\alpha_{z_i} - (z_i - z_v)\alpha_{y_i}]^2 + [(z_i - z_v)\alpha_{x_i} - (x_i - x_v)\alpha_{z_i}]^2 + \\ &[(x_i - x_v)\alpha_{y_i} - (y_i - y_v)\alpha_{x_i}]^2 \end{aligned} \tag{4.4}$$

So, we end up with

$$\begin{aligned} \frac{\partial \chi^2}{\partial x_v} &= \sum w_i^2 \left[-(x_i - x_v)(\alpha_{z_i}^2 + \alpha_{y_i}^2) + (y_i - y_v)\alpha_{x_i}\alpha_{y_i} + (z_i - z_v)\alpha_{x_i}\alpha_{z_i}\right] = 0 \\ \frac{\partial \chi^2}{\partial y_v} &= \sum w_i^2 \left[(x_i - x_v)\alpha_{x_i}\alpha_{y_i} - (y_i - y_v)(\alpha_{z_i}^2 + \alpha_{x_i}^2) + (z_i - z_v)\alpha_{y_i}\alpha_{z_i}\right] = 0 \\ \frac{\partial \chi^2}{\partial z_v} &= \sum w_i^2 \left[(x_i - x_v)\alpha_{x_i}\alpha_{z_i} + (y_i - y_v)\alpha_{y_i}\alpha_{z_i} - (z_i - z_v)(\alpha_{x_i}^2 + \alpha_{y_i}^2)\right] = 0 \end{aligned} \tag{4.5}$$

where $w_i = 2/\sigma_i^2$ are treated as constants. The equation system (4.5) is linear in the parameters x_v, y_v, z_v and therefore it can be immediately solved. In matrix notation, if we define

$$A_i = w_i^2 \cdot \begin{pmatrix} \alpha_{y_i}^2 + \alpha_{z_i}^2 & -\alpha_{x_i}\alpha_{y_i} & -\alpha_{x_i}\alpha_{z_i} \\ -\alpha_{y_i}\alpha_{x_i} & \alpha_{x_i}^2 + \alpha_{z_i}^2 & -\alpha_{y_i}\alpha_{z_i} \\ -\alpha_{x_i}\alpha_{z_i} & -\alpha_{y_i}\alpha_{z_i} & \alpha_{y_i}^2 + \alpha_{x_i}^2 \end{pmatrix} \tag{4.6}$$

and

$$A = \sum A_i \tag{4.7}$$

and

$$B = \sum A_i \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$$

Then, we can write the equation system 4.5 as

$$A \begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix} = B \Rightarrow \begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix} = A^{-1}B \tag{4.8}$$

Thus obtaining the vertex coordinates.

4.2.2 Error propagation

In the previous section we have assumed that the uncertainties, σ_i, in the distance determination were constant. This assumption allowed us to derivate an analytical solution for the Least Squares Method problem (LSM in short, see (2)) . However, in reality, the fact that σ_i is not constant needs to be taken into account. These uncertainties inform us about how relevant each of the track is when it comes to determining the vertex, they are necessary to give correct weights to the different tracks. One can visualize this intuitively by realizing that a track going parallel to the z axis contains no information as to what the z coordinate of the vertex is.

We will deduce now the expression for σ for a generic track given by parameters (x, y, x', y'), where (x, y) are the track's x and y coordinates for z=0 and (x', y') are the track's slopes in the XZ and YZ planes respectively; x' and y' come defined in terms of the direction vector components as:

$$\begin{aligned} x' &= \frac{\alpha_x}{\alpha_x} \\ y' &= \frac{\alpha_y}{\alpha_z} \end{aligned}$$

The variance can be computed from the well known equation

$$\sigma^2(d) = (\frac{\partial d}{\partial x}, \frac{\partial d}{\partial y}, \frac{\partial d}{\partial x'}, \frac{\partial d}{\partial y'}) \begin{pmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xx'} & \sigma_{xy'} \\ \sigma_{xy} & \sigma_{yy} & \sigma_{yx'} & \sigma_{yy'} \\ \sigma_{xx'} & \sigma_{yx'} & \sigma_{x'x'} & \sigma_{x'y'} \\ \sigma_{xy'} & \sigma_{yy'} & \sigma_{x'y'} & \sigma_{y'y'} \end{pmatrix} \begin{pmatrix} \frac{\partial d}{\partial x} \\ \frac{\partial d}{\partial y} \\ \frac{\partial d}{\partial x'} \\ \frac{\partial d}{\partial y'} \end{pmatrix} \tag{4.9}$$

where σ_{ij} denote the covariance of the i and j variables for $i \neq j$ and the corresponding variance for $i = j$. For convenience, let's also introduce the variables

$$\begin{aligned} \Delta_x &= x - x_v \\ \Delta_y &= y - y_v \\ \Delta_z &= 0 - z_v \end{aligned}$$

where (x_v, y_v, z_v) are still the coordinates of the vertex point. With these definitions and using $\alpha_z^2 = 1 - \alpha_x^2 - \alpha_y^2$, the expression of distance between a track and the vertex, d, becomes

$$\begin{aligned} d^2 &= \left(\Delta_z^2 - \Delta_x^2\right)\alpha_x^2 + \left(\Delta_z^2 - \Delta_y^2\right)\alpha_y^2 + \Delta_x^2 + \Delta_y^2 - \\ &\quad 2\left(\Delta_x\Delta_z\alpha_y\alpha_z + \Delta_x\Delta_z\alpha_x\alpha_z + \Delta_x\Delta_y\alpha_x\alpha_y\right) \end{aligned}$$

For simplicity we will not compute the derivatives with respect to x' and y'; instead the expressions will be shown for the derivatives with respect to α_x and α_y. The former ones can be obtained from the later trivially as:

$$\begin{aligned} \frac{\partial d}{\partial x'} &= \frac{\partial d}{\partial \alpha_x}\frac{\partial \alpha_x}{\partial x'} + \frac{\partial d}{\partial \alpha_y}\frac{\partial \alpha_y}{\partial x'} = \frac{\partial d}{\partial \alpha_x}\frac{1-\alpha_x^2}{L} + \frac{\partial d}{\partial \alpha_y}\frac{x'y'}{L} \\ \frac{\partial d}{\partial y'} &= \frac{\partial d}{\partial \alpha_x}\frac{\partial \alpha_x}{\partial y'} + \frac{\partial d}{\partial \alpha_y}\frac{\partial \alpha_y}{\partial y'} = \frac{\partial d}{\partial \alpha_x}\frac{x'y'}{L} + \frac{\partial d}{\partial \alpha_y}\frac{1-\alpha_y^2}{L} \end{aligned}$$

where $L = \sqrt{x'^2 + y'^2 + 1}$.

The only ingredients missing are the partial derivatives of d, given by

$$\begin{aligned} \frac{\partial d}{\partial x} &= \frac{(1-\alpha_x^2)\Delta_x - \alpha_x(\Delta_z\alpha_z + \Delta_y\alpha_y)}{d} \\ \frac{\partial d}{\partial y} &= \frac{(1-\alpha_y^2)\Delta_y - \alpha_y(\Delta_z\alpha_z + \Delta_x\alpha_x)}{d} \\ \frac{\partial d}{\partial \alpha_x} &= \frac{(\Delta_z^2 - \Delta_x^2)\,\alpha_x - \Delta_x\Delta_y\alpha_y + 2\alpha_x\alpha_y\Delta_y\Delta_z - \Delta_x\Delta_z(1 - 3\alpha_x^2 - \alpha_y^2)}{d} \\ \frac{\partial d}{\partial \alpha_y} &= \frac{\left(\Delta_z^2 - \Delta_y^2\right)\alpha_y - \Delta_x\Delta_y\alpha_x + 2\alpha_x\alpha_y\Delta_x\Delta_z - \Delta_y\Delta_z(1 - 3\alpha_y^2 - \alpha_x^2)}{d} \end{aligned}$$

The final expression can now be obtained just from the matrix multiplication from Eq. 4.9. The fact that the vertex coordinates appear in this expression has the consequence that the LSM functional is not anymore linear in the parameters. In other words, by introducing proper error propagation we are not able to obtain an analytical solution anymore. Numeric methods are then needed to minimize Q^2; the methods used are presented in section 4.2.4. Before that, more non linearities will be introduced in the treatment of outliers presented in the following section.

4.2.3 Robust vertex fit

One problem we see with the method presented in 4.2.1 is how the fitting behaves in the presence of outliers. That is, when we have several tracks

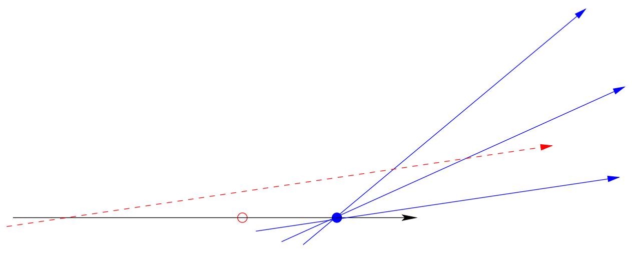


Figure 4.3: This figure shows the effect of an outlier (dashed line) on the vertex estimate. The solid circle is the vertex estimate when only the "good" tracks are taken into account. When the outlier is introduced the new vertex spot (hollow circle) is pushed back by it.

pointing to one common vertex, the real one, and another track going far away from that point (see figure 4.3). In such case too much weight is given to the track passing far away (outlier). This makes the fitted result to be farther away from the real vertex than it would have been if the outlier was not used. In other words, adding wrong information makes the result worse.

This is a well known problem affecting any method which is based on LSM. The reason for it is that LSM assumes the errors in the measurements to be Gaussian. The presence of outliers is a symptom that the Gaussian assumption is wrong; instead the actual distribution may look still like a Gaussian but with longer tails. Then it is more probable to have a measurement far from the model estimation than what we would expect from the Gaussian approximation. Since LSM does not know about it, it will try to modify the model parameters to accept the outlier. In other words, the compact nature of the Gaussian distribution makes it cheaper for the fitting routine to accept a small deviation in many data points than a large deviation in any single one.

From a physics point of view, these outliers may be originated by a variety of reasons. Ranging from badly reconstructed tracks in the chambers to secondary particles not originating from the target. In any case, we cannot get rid of them prior to the vertex reconstruction, and therefore the algorithm devised for that purpose must behave well in the presence of outliers. In other words, we need to make our algorithm *robust*.

A pragmatic way to accomplish this objective would be to somehow choose initial values for x_v, y_v, z_v and then simply remove from the sample those tracks passing far from that initial vertex. The advantage of such a method is its simplicity. However, with such a method, the choice of initial vertex strongly influences the fit result.

A more elaborated approach (1) can be developed by just accepting the fact that the residuals [1] are non Gaussian distributed and using a more general method, which can deal with this fact.

Weighted LSM: Tukey weights

A well known estimator matching the requirements is the maximum likelihood (ML) method (2). Let $f(d)$ be the residuals distribution; then we call likelihood $L = \prod_i^N f(d_i))$. The ML consists on finding the parameters x_i, y_i, z_i which make L maximum. Actually, what is typically maximized is not L, but $\ln L$ as the logarithm's derivative is definite positive. Then the minimum of a function and its logarithm are the same.

An important input to ML is the actual distribution of residuals. Actually, if a Gaussian is used, the LSM method is recovered. In our case, we have some tracks pointing to a common vertex and a few astray ones. Let's call the first subset *signal* and *background* to the second.

Then, the approximation can be made that signal's residuals follow a normal distribution while background is uniformly distributed. The compound residuals distribution can be written as:

$$f(d) = (1-\epsilon)\varphi(d) + \epsilon h_0,\ \epsilon \in [0,1] \tag{4.10}$$

Where ϵ is the background level. The ML function can then be written as

$$\ln \prod f(d_i)_i^N = \sum \ln \left(\frac{1-\epsilon}{\sigma\sqrt{2\pi}} e^{-\frac{d_i^2}{2\sigma^2}} + \epsilon h_0 \right) \tag{4.11}$$

Differentiating with respect to x_v, y_v, z_v and dividing by

$$(1-\epsilon)\frac{e^{-d_i^2/2\sigma^2}}{\sigma\sqrt{2\pi}} \tag{4.12}$$

We recover the LSM system with equations

$$\begin{aligned} \sum \frac{w_i}{\sigma^2} d_i \frac{d_i}{x_v} &= 0 \\ \sum \frac{w_i}{\sigma^2} d_i \frac{d_i}{y_v} &= 0 \\ \sum \frac{w_i}{\sigma^2} d_i \frac{d_i}{z_v} &= 0 \end{aligned} \tag{4.13}$$

[1] The residual for a particular measurement is the distance between that measurement and the vertex point

with the weight function

$$w = \frac{1+c}{1+cexp(t^2/2)},\ t = \frac{e}{\sigma},\ c = \frac{\epsilon\sigma h_0\sqrt{2\pi}}{1-\epsilon} \tag{4.14}$$

which can be approximated by the famous Tukey's bi-squared weights

$$w(t) = \begin{cases} (1-(t/C_T)^2)^2 & if|t| < C_T \\ 0 & otherwise \end{cases} \tag{4.15}$$

Recapitulating, it has been shown that the problem of taking into account outliers can be addressed by introducing weights into a standard LSM functional. Therefore, the name of "weighted LSM" for this algorithm. Fig. 4.4 shows the residuals distribution in the reconstructed vertex's z coordinate, with and without using weights.

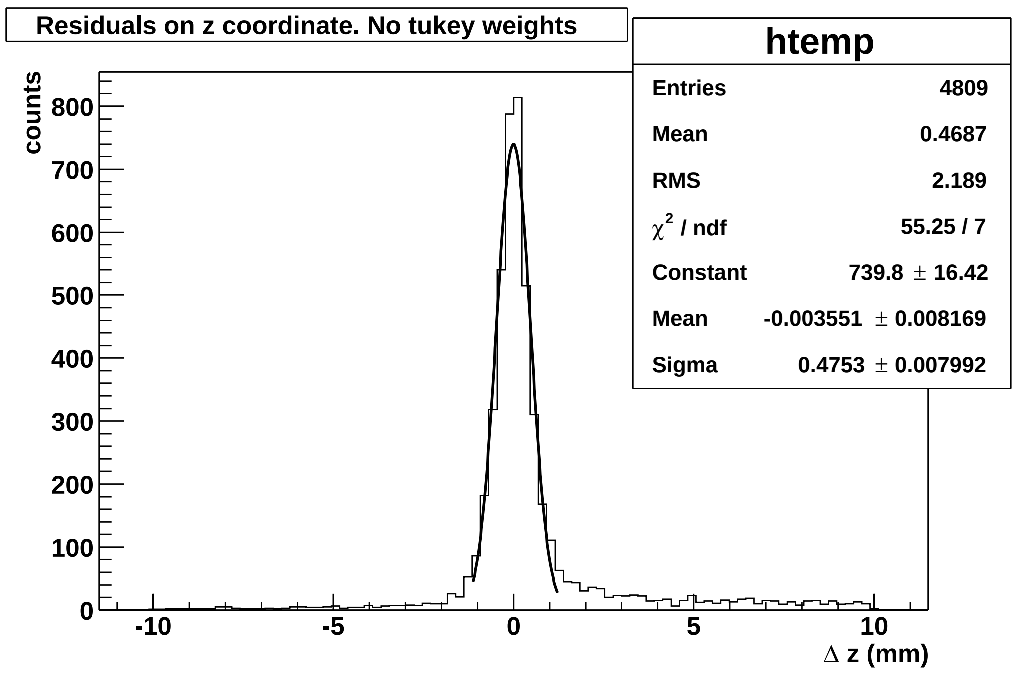
4.2.4 Functional minimization

Both corrections on the original algorithm have the effect of introducing non linearities which make an analytical solution difficult if not impossible. In practical terms, this translates into the need for an iterative minimization procedure.

As shown, all non linearities can be moved into the weight factors accompanying the terms in the sum. Those weights depend on the vertex's position itself. However, we can assume that the weights change slowly with the vertex. Then, a valid procedure is that in each iteration the vertex from the previous one is used to compute the weights. This is repeated until a certain convergence criterion is achieved. In our case the criterion is that the vertex from the new iteration has a distance to the one from the previous iteration which is smaller than a certain value ϵ. Convergence is achieved after 10 iterations typically, for a value of $\xi = 0.01$.

4.3 Achieved resolution

The most obvious ingredient to the vertex resolution is the uncertainty induced by the tracking resolution in the MDC detectors. In particular the resolution in the determination of slope. The slope is specially important because the target is far from the tracking detectors (around 0.5 meters). Thus a small error in the slope translates into a proportionally large error in the track position at the vertex plane.



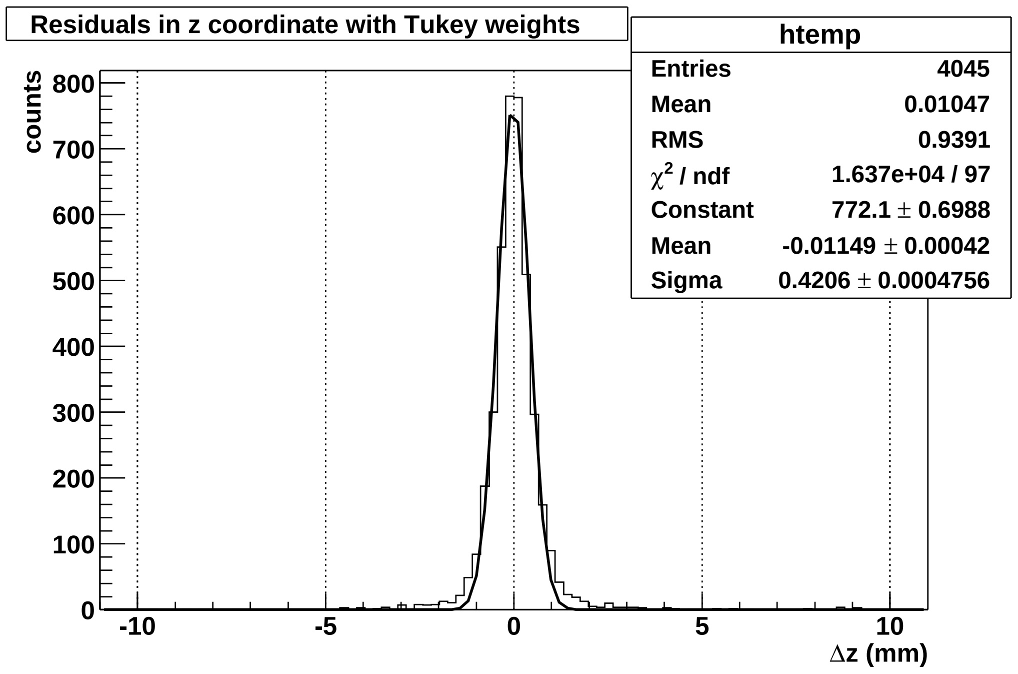


Figure 4.4: Effect of activating the Tukey weights in simulated data from a C+C collision. Both figures show the reconstructed z coordinate of the vertex minus the actual one. In the first case (left) the reconstruction was done without using Tukey weights and we see the effect of outliers in the tails. On the second case outliers (right) have mostly been removed. In both figures multiple scattering was switched off.

Another important problem is the effect of multiple scattering. When a particle traverses a medium it is subject to multiple scattering. Multiple scattering is the effect produced by the multiple elastic collisions a particle experiments with the atoms and molecules of the medium it is traversing. In each of those collisions, the particle is deviated from its original direction. Since it depends on many variables multiple scattering is considered a random process. After several such collisions, information of the incident direction is gradually degraded.

In HADES, when a particle reaches the MDC detectors is has traveled through the target and the RICH detector, that adds up to a distance around 0.5 m through different materials. Consequently, the direction measured in the MDC is not the same as the direction the particle left the target with, but for vertex reconstruction we need the direction at the interaction spot. Since multiple scattering may alter direction with equal probability in any sense, it can be assumed that the direction determined from MDC is a measurement of the one we are interested in, just with an extra error source. This has an effect on our ability to determine the target, introducing an additional error as shown in Fig. 4.5.

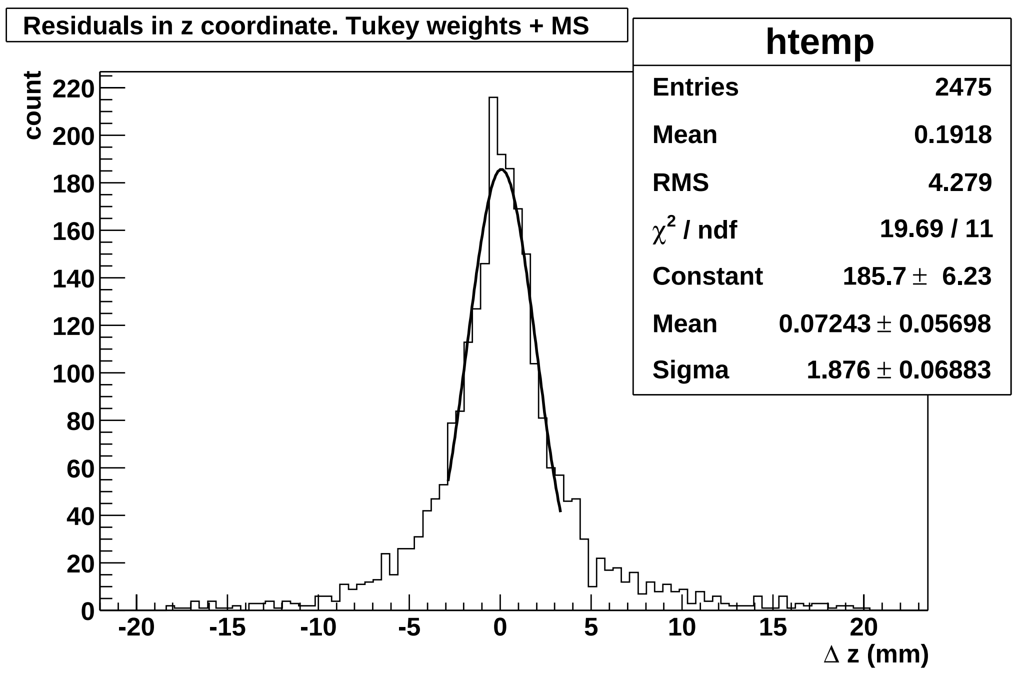
In that figure, the residuals in the reconstructed z coordinate of the vertex are plotted for both the case with and without multiple scattering. Besides the fact that the distribution is wider with multiple scattering on, it is also shown how multiple scattering introduces significant tails. This is due to the fact that multiple scattering cannot be considered a Gaussian process. It certainly is the compound effect of many deviations which may affect the measurement in any direction, however the deviations are not always small.

The next effect is the low available multiplicity for some systems. Even though HADES intends to measure systems as heavy as Au+Au which translate into high track multiplicities, for the moment all data have been taken with lighter systems, like C+C. Lighter systems mean less particles per event and that affects the achieved resolution because, as any other fitting routine, the vertex algorithm requires redundancy to make accurate estimations. Fig. **??** illustrates this effect.

Table 4.1 shows what the average resolution is for different combinations of the variables above. All results are with Tukey weights activated; it should be noted that for low multiplicities Tukey weights does not always converge.

4.3.1 Systematic error sources

Thus far the statistical errors. There are also sources for systematic ones. An example is alignment: if the real position of the MDC chambers is different



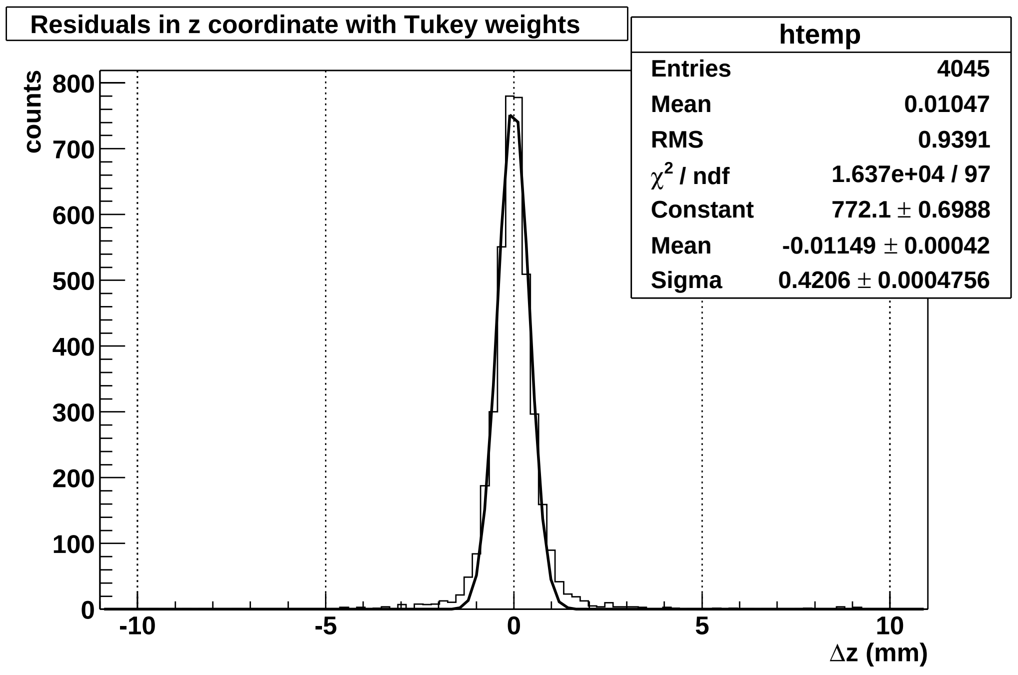


Figure 4.5: Multiple scattering effect on simulated C+C data. Both figures show the reconstructed z coordinate of the vertex minus the actual one. In the first case multiple scattering was turned on in HGEANT while it was off for the second figure. Besides the increase in width, multiple scattering introduces larger tails (see text).

	Multiple Scattering		No Multiple Scattering	
	C+C	Au+Au	C+C	Au+Au
Ideal tracking	1.1, 1.1, 1.9	0.3, 0.3, 0.5	0.4, 0.4, 0.4	0.08, 0.07, 0.09

Table 4.1: Vertex reconstruction resolution in different scenarios using Tukey weights. Each entry in the table is a triplet with the resolutions in x, y and z in mm.

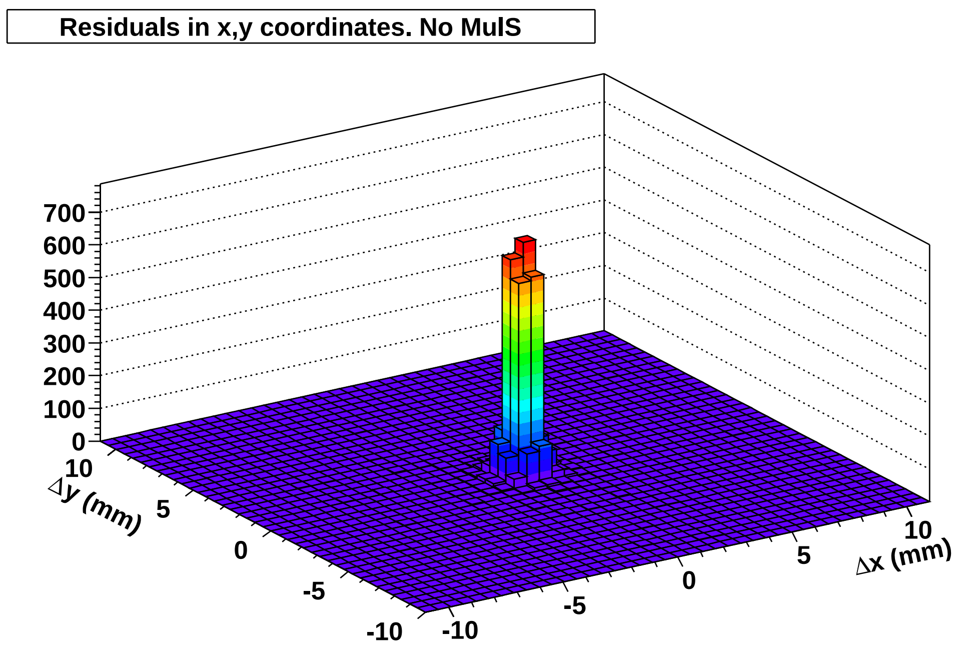


Figure 4.6: Residuals in the x and y coordinates for the scenario C+C without multiple scattering and Tukey weights activated.

from the one used in the analysis one should get systematic errors. The kind of alignment which affects vertex reconstruction is alignment between the MDC chambers themselves. There are several techniques to attack that alignment problem, some of them making use of the vertex information. A review of the techniques and its application to HADES can be found on (3).

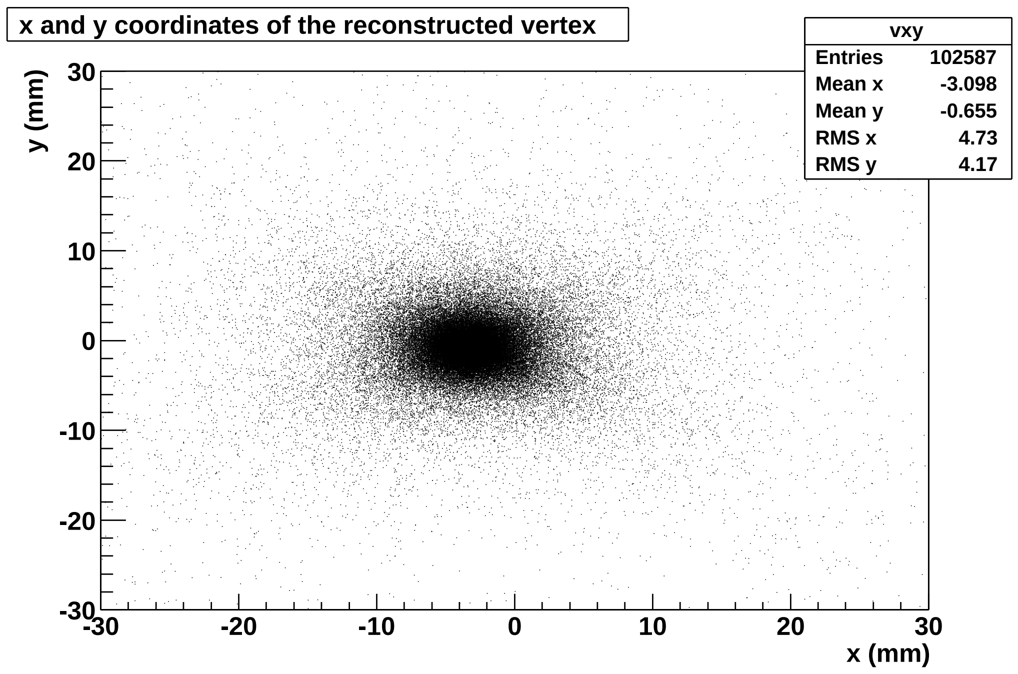
Another effect is the one introduced by secondaries, that is particles not originating from the target. This is in part taken care of by the Tukey weights.

The third effect is the presence of a residual magnetic field between the two inner chambers. Because of that field the track does not follow a straight path, that bending depends on the particle's momentum. If the tracking algorithm assumes the track to be straight, it will introduce a small systematic in slope which translates into a systematic in vertex determination (around $100\mu m$ for full field). There are several solutions:

- Use full tracks for the vertex fit, that is tracks not only formed with the inner two MDC detectors, but also the outer ones. At that level momentum information has been introduced and, depending on the momentum reconstruction algorithm, the curvature in the inner chambers has been taken care of.
- Use tracks formed only with the first MDC. It can be assumed that there is no field around MDC1. The disadvantage is that by using only one MDC the slope resolution is reduced.

4.4 Application to real data

Fig. 4.7 shows the reconstructed vertex for a NOV01 data set. In that case the used target was a cylindrical carbon target with a length of 5mm and a diameter of 8mm. That reflects in the widths of the distributions, while their means tell about the target position.



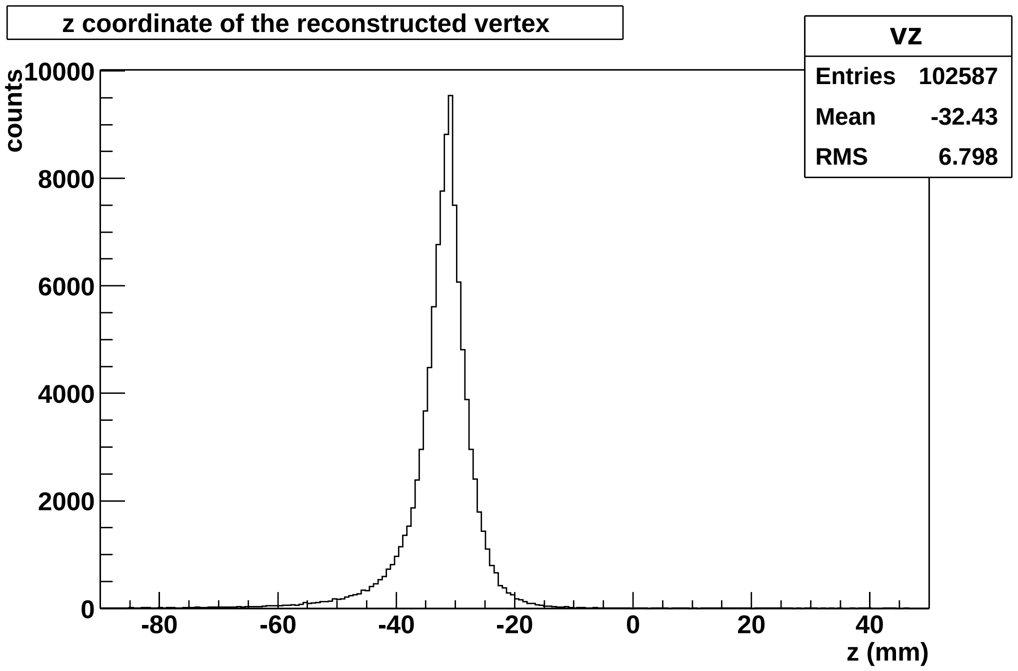


Figure 4.7: Reconstructed vertex coordinates for events in the Nov01 data set. The distributions centers correspond to the target position in their respective coordinates

Bibliography

[1] G. Agakichiev et al. *A new robust fitting algorithm for vertex reconstruction in the CERES experiment.* Nuclear Instruments and Methods in Physics Research A(394):225-231, 1997

[2] W.T. Eadie et al. *Statistical methods in Experimental Physics.* North Holland Publishing Company, Amsterdam, 1971

[3] H. Álvarez Pol. *On the multiwire drift chamber alignment of the Hades spectrometer.* PhD. Thesis, Universidad de Santiago de Compostela, 2002.

[4] E. Calligarich et al. *A fast algorithm for vertex estimation.* Nuclear Instruments and Methods in Physics research A(311):151-155, 1992

[5] G. Stimpft Abele. *Finding the Decay Vertex of a Charged Track with Neural Networks.* Université Blaise Pascal. Unpublished

[6] M. Regler and R. Frühwirth. *Reconstruction of charged tracks.* From Techniques and Concepts of High Energy Physics. Pienum Publishing Corp., 1990

[7] R. Frühwirth. *Application of Kalman filtering to track and vertex fiting.* Nuclear Instruments and Methods in Physics Research A(262):444, 1987

[8] I. Kisel et al. *Elastic neural net for track and vertex search.* Nuclear Instruments and Methods in Physics Research A(389):167-168, 1997

Chapter 5

Momentum reconstruction

In this chapter we will deal with the problem of obtaining the momentum of particles traversing the HADES spectrometer. This task is essential to understand the physics of events recorded by the machine.

A particle's momentum is obtained from its deflection in a magnetic field, which requires measuring the particle's direction before and after the magnet. This information is provided by the inner MDC chambers before the magnet, and either the outer MDC or META detectors in the outer part of the spectrometer. The magnetic field is provided by a toroidal superconducting magnet in between the two inner and the two outer MDC chambers.

The main difficulties we will have to face are related to the effect of multiple scattering and the fact that the magnetic field in use is inhomogeneous. The first of these problems affects mostly the detector design, as materials have to be chosen to minimize this effect. As we will see, even with a careful design, the effect of multiple scattering is not negligible. On the other hand, the additional difficulty of dealing with an inhomogeneous magnetic field is something we will have to deal with in the offline analysis.

In the following sections the requirements on momentum reconstruction will be presented, as well as two particular algorithms: the so called "kick plane" and "reference trajectories". These two algorithms respond to different design goals, their connections and differences will be shown. First both algorithms will be presented from a theoretical standpoint, while the later sections in the chapter deal with their application and performance in different scenarios.

Mainly these different scenarios respond to the temporal non availability of some of the MDC chambers, due to its installation not being completed for some of the data taking periods. Section 5.4.1 describes a setup where the two outer MDC chambers are absent, thus seriously limiting the resolution;

on section 5.4.2 the innermost of the two outer chambers is added (MDC3) enabling a much improved resolution; last but not least the effect of adding the outermost chamber is discussed in section 5.4.3, giving us the final edge on momentum resolution.

5.1 Motivation and requirements

As explained in 2 HADES' mainly strives to obtain the invariant mass of dilepton pairs coming out of heavy ion collisions. For this purpose a reliable momentum reconstruction is needed, since a dilepton's invariant mass directly depends upon the momentum of the participating leptons, as shown in its well known expression.

$$M = \sqrt{(E_1 + E_2)^2 - |\mathbf{p}_1 + \mathbf{p}_2|^2} \tag{5.1}$$

where $(E_i, \mathbf{p}_i)$ is the quadrimomentum vectors for lepton i.

The information on particle momentum has many other applications as well, ranging from assisting the second level trigger with particle identification. Naturally, this broad range of applications translates into different requirements.

The second level trigger's Matching Unit (MU) uses momentum information on-line to select events with an increased probability of containing a dilepton. This is done by identifying lepton candidates and then placing cuts on their invariant mass. Here, the main constraint is in the time used up by the momentum reconstruction algorithm. This time has to be short so that the MU can cope with the high rates of data taking. On the other hand the algorithm's resolution is not of critical importance for the MU.

All other tasks, which take place during the offline analysis, are more concerned about resolution than speed. Some of those tasks include lepton identification in the Shower detector, particle identification etc. But the most demanding is being able to resolve the ω meson; which requires the invariant mass resolution to be at least in the order of the meson's width, and that, in turn, demands a momentum reconstruction resolution of about 1%.

For comparison, the precursor experiment, DLS (see (1)), had a resolution in the order of 10%.

Another requirement is for the algorithm to adapt to experimental setups where not all detectors are present, as already explained.

5.1.1 Applications of momentum reconstruction

This is a list of all known uses of momentum reconstruction in HADES

- In the Matching Unit, to help in taking the trigger decision. The momentum is used to obtain the invariant mass of open pairs, placing a cut on "reasonable" masses.
- The SHOWER detector uses momentum to help in the lepton identification. SHOWER takes a decisions whether if a particle is a lepton or not by looking at the firsts steps in the shower produced by the particle in the detector. The multiplication factors for electromagnetic showers differ with momentum and so does the cut to be applied.
- Particle Identification (PID) a topic where momentum plays a significant role since PID is usually performed with two dimensional cuts on histograms like momentum versus speed.
- Lepton analysis. Momentum is needed to obtain invariant mass
- Hadron analysis. One typically wants to look at cross sections as a function of momentum, transverse momentum, energy etc. Most of those variables involve momentum in one way or the other.

5.2 Reference trajectories algorithm

The standard way to estimate a track's parameters in many experiments is a Least Squares Method (LSM) which requires a track model. The difficulty in our case is precisely in obtaining such a track model, due to the inhomogeneous nature of the magnetic field in HADES. We will see that in fact not the full track model is needed, but only certain information about it. The main idea of the "Reference Trajectories" algorithm is to obtain that information by numerical means.

5.2.1 Fitting procedure

A track is completely defined by 5 parameters, thus it can be represented by a five dimensional vector with components $\mathbf{p} = (1/p, \rho, z, \theta, \phi)$; let's call the parameter space where this vector lives P. The first parameter is the inverse of the the particle's momentum. The reason to use $1/p$ as parameter instead of just p is that deflection in the magnetic field is inversely proportional

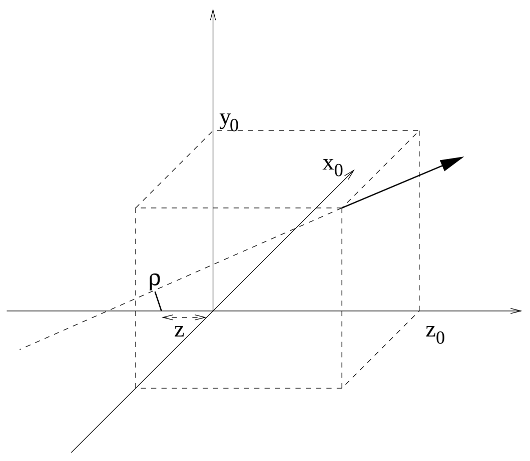


Figure 5.1: Definition of ρ and z

to momentum; what is actually measured is deflection and thus we expect random errors to affect it infinitesimally; thus $1/p$ can be expected to behave in Gaussian way, while p does not. The other 4 components of $\mathbf{p}$ represent a straight line as it comes out of the interaction point. Let's assume the track goes through the point $\mathbf{v} = (x_0, y_0, z_0)$ with direction given by the vector $\alpha = (\alpha_x, \alpha_{y,}\alpha_z)$. Then

$$\begin{aligned}
\theta =& \arccos\left(\frac{\alpha_z}{\sqrt{\alpha_x^2 + \alpha_y^2 + \alpha_z^2}}\right) \\
\phi =& \arctan(\alpha_y/\alpha_x) \\
\rho =& y_0 \cos\phi - x_0 \sin\phi \\
z =& -\frac{(x_0 \cos\phi + y_0 \sin\phi)\cos\theta}{\sin\theta}
\end{aligned}$$

where θ and ϕ are the well known polar and azimuthal angles with respect to the z axis, while ρ and z give information on the track's origin. Intuitively z is the z coordinate of the point in the track which is closest to the z axis (e.g. 0 if they intersect) and ρ is the distance of closest approach between the z axis and the track itself.

In order to estimate the parameter vector $\mathbf{p}$, all the information we have are the coordinates of several points along the track. Let's put those coordinates in a measurement vector $\mathbf{x}_m$. In our case the measurements are provided by the 4 MDC chambers, each of them giving the (x, y) coordinates of the track on the chamber's center plane. Therefore $\mathbf{x}_m$ is a 8 dimensional vector[1] living in a vector space X. Since $\mathbf{p}$ is a complete definition of the track, there must be a function $\mathbf{F} : P \to X$ relating the two vector spaces. In general $\mathbf{x} = \mathbf{F}(\mathbf{p})$

There are two components contributing to $\mathbf{x}_m$, that is $\mathbf{x}_m = \mathbf{x} + \mathbf{x}_r$ where $\mathbf{x}_r$ corresponds to the random measurement errors. Therefore our measurement vector will come with a covariance matrix; let its inverse be W.

The LSM[2] tells us that as far as $\mathbf{x}_r$ are Gaussian an optimal estimation of $\mathbf{p}$ can be obtained from $\mathbf{x}_m$ just minimizing the functional

$$Q^2 = (\mathbf{F}(\mathbf{p}) - \mathbf{x}_m)^T W (\mathbf{F}(\mathbf{p}) - \mathbf{x}_m) \tag{5.2}$$

Up to this point everything is like in a standard LSM, now we would proceed to find $\mathbf{p}_e$ satisfying the following system of 5 equations:

$$\left.\frac{\partial \chi^2}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{p}_e} = \mathbf{0} \tag{5.3}$$

The problem is $\mathbf{F}$ being unknown. Even if we knew $\mathbf{F}$, the problem of solving 5.3 with a non linear $\mathbf{F}$ looks daunting. We have tackled the second of the problems by linearizing the functional. Let's assume $\mathbf{F}$ to be smooth enough around $\mathbf{p}_e$. Then $\mathbf{F}$ admits a Taylor expansion around any point $\mathbf{p}_0$ close to $\mathbf{p}_e$.

$$\mathbf{F}(\mathbf{p}) = \sum_{k=0}^{n} \left.\frac{\partial^k \mathbf{F}}{\partial \mathbf{p}^k}\right|_{\mathbf{p}=\mathbf{p}_0} (\mathbf{p} - \mathbf{p}_0)^k$$

Then, a first order approximation can be done which is good for all $\mathbf{p}$ close to $\mathbf{p}_0$ (and $\mathbf{p}_e$). Consequently, the following simplified model can be used for the functional minimization.

$$\mathbf{F}(\mathbf{p}) \simeq \mathbf{F}(\mathbf{p}_0) + A \cdot (\mathbf{p} - \mathbf{p}_0) + O\left((\mathbf{p} - \mathbf{p}_0)^2\right) \tag{5.4}$$

Where A is the matrix with elements $A(i,j) = \left.\frac{\partial F_i(\mathbf{p})}{\partial p_j}\right|_{p=p_0}$. With this approximation equation (5.2) can be written as

$$Q^2 = \left[\mathbf{F}(\mathbf{p}_0) + A \cdot (\mathbf{p} + \mathbf{p}_0) - \mathbf{x}_m\right]^T W \left[\mathbf{F}(\mathbf{p}_0) + A \cdot (\mathbf{p} + \mathbf{p}_0) - \mathbf{x}_m\right]$$

[1] 4 chambers times 2 measurements per chamber

[2] Least Squares Method

Imposing (5.3) and after some math work we arrive to

$$\mathbf{p}_e = \mathbf{p}_0 + \left(A^T W A\right)^{-1} A^T W \cdot (\mathbf{x}_m - F(\mathbf{p}_0)) \qquad (5.5)$$

There are still two problems: we need methods to obtain a suitable $\mathbf{p}_0$ and to compute both the A matrix and $\mathbf{F}(\mathbf{p}_0)$. In order to tackle the second of the issues, we can restrict ourselves and impose for $\mathbf{p}_0$ to belong to a set of points specified beforehand and for which we have numerically computed the corresponding $\mathbf{F}(\mathbf{p}_0)$. In other words, we tabulate $\mathbf{F}$. The details on how to obtain such a table and how to calculate the A matrix from it are explained later; for the moment let's assume that they are given.

With these ingredients it is easy to devise an iterative algorithm to estimate the track parameters. We start from an initial estimation, this could be taken from an alternative momentum reconstruction algorithm, like the kick plane algorithm from section 5.3; let's note that estimation by $\mathbf{p}_e^0$. From any estimation, a better one can be iteratively obtained by

$$\mathbf{p}_e^{k+1} = \mathbf{p}_e^k + \left(A^T W A\right)^{-1} A^T W \cdot \left(\mathbf{x}_m - F(\mathbf{p}_e^k)\right)$$

Since we don't know how to calculate $F(\mathbf{p}_e^k)$ for a generic $\mathbf{p}_e^k$. In each iteration that vector is substituted by the closes entry in the table of F. This minimization algorithm is sketched in Fig. 5.2. The algorithm typically converges in 1 or 2 iterations

5.2.2 Tabulating F

The goal is to evaluate $\mathbf{F}$ for a set of points forming a grid in the 5 dimensional parameter space, as in Fig. 5.3. The limits of the grid are basically defined by the geometrical acceptance of HADES and the kind of physics to do. The granularity has to be chosen hitting a compromise between resolution and memory consumption. The key of the method explained above is the linear approximation of $\mathbf{F}$ between grid points. The more fine grained the binning the better the linear approximation is, and therefore the better the resolution we get out of the fit routine. On the other hand the more fine grained the binning the larger the memory consumption, so we have to hit a compromise which satisfies the requirement of 1% momentum resolution.

The limits on the parameters are defined by the HADES geometry as follows

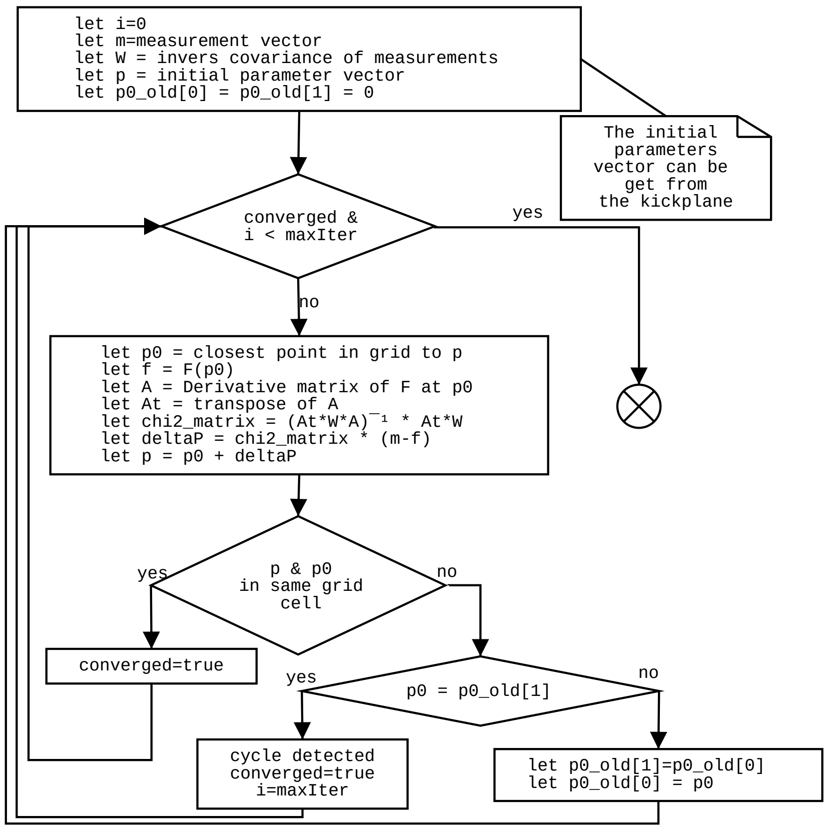

let i=0
let m=measurement vector
let W = invers covariance of measurements
let p = initial parameter vector
let p0_old[0] = p0_old[1] = 0
The initial parameters vector can be get from the kickplane
converged & i < maxIter
yes
no
let p0 = closest point in grid to p
let f = F(p0)
let A = Derivative matrix of F at p0
let At = transpose of A
let chi2_matrix = (At*W*A)⁻¹ * At*W
let deltaP = chi2_matrix * (m-f)
let p = p0 + deltaP
p & p0 in same grid cell
yes
no
converged=true
p0 = p0_old[1]
yes
no
cycle detected converged=true i=maxIter
let p0_old[1]=p0_old[0]
let p0_old[0] = p0

Figure 5.2: Flowchart of the minimization algorithm explained in the text.

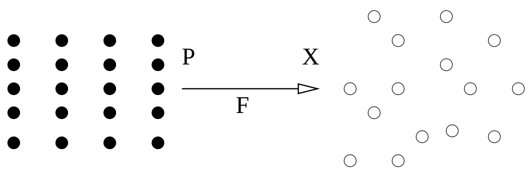


Figure 5.3: Since the track model is difficult to obtain analytically, F is computed on a set of data points forming a grid. The computation is performed with a Monte Carlo

- ρ, z are strongly related with the originating point of the track, which, for physically interesting particles, has to be in the target region. Therefore the limits in ρ, z are chosen to define a cylinder along the z axis encapsulating the target. ρ is the radius of such a cylinder and z the height range. The actual numbers depend on the particular target configuration for a given experiment.

- θ, ϕ are the familiar polar an azimuthal angles. So their ranges are defined by the angular coverage of the spectrometer. In θ this ranges from 18^o to 85^o. However values outside that window need to be accepted to account for detector resolution effects, therefore the window has been extended to span from 14^o to 90^o. In principle ϕ should cover from -180^o to 180^o with 0^o corresponding to the center of the uppermost sector. However we can make use of the spectrometers' sector symmetry to reduce this range to span from -30^o to +30^o; furthermore, the left and right halves of each sector are also symmetrical; so we actually need to tabulate F for ϕ in the range 0^o to 30^o.

- $1/p$ is the most difficult to specify because it means putting a window in the momentum acceptance. Currently the range is set to $\frac{1}{p} \in \left[\frac{1}{1500}, \frac{1}{100}\right]$ in units of MeV^{-1}. 100 MeV is the minimum allowed momentum because it is essentially the momentum kick of the magnetic field, 1500 MeV is set as the maximum to provide good resolution on the region of physical interest (around the ω meson mass) while keeping memory usage at a reasonable level. To determine momentum below 100 MeV either the kick plane can be used or a reference trajectories fit with a specific table for low momentum.

As for the binning, we have not devised any mathematically rigorous method to optimize it; but a hand made approach has been used instead. The first step being to chose an initial binning based on educated guesses. This guesses are based in the intuition gained while implementing the "Kick Plane Algorithm" (see Section 5.3). In that section it is shown that $1/p$ essentially goes linearly with deflection, so the corresponding binning can be rather rough for the high resolution we need to achieve. It is also shown there that the main variation in momentum kick happens when moving along θ, so the binning has to be relatively fine grained. The binning in ϕ could be rougher was it not for the strong variation in the magnetic field at large ϕ, that is, near the magnet's coils.

There are two possible solutions here, one is to allow for variable sized cells in the grid and the other is to reduce the overall bin size in ϕ. Since

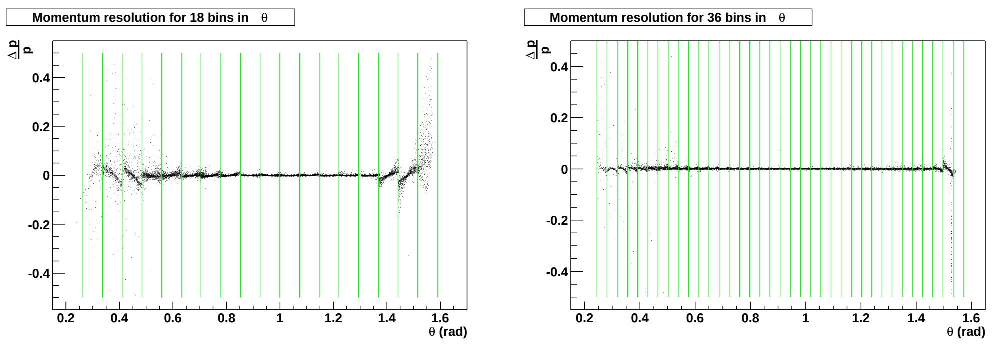


Figure 5.4: The leftmost figure shows $\frac{\Delta p}{p}$ as a function of θ for two different bin sizes. The leftmost figure corresponds to 18 bins while the one at the right is for 36 bins. The vertical bars show the bin boundaries

the first complicates derivatives computation code both complexity and time wise, it has been left for future exploration. In our case we started from the previous work on reference trajectories by R.Schicker (see (7)).

In order to optimize the binning, the final momentum resolution ($\frac{\Delta p}{p}$) can be plotted versus the different parameters; this makes it easy to see if a more fine grained binning is needed in each case. One example is shown in Fig. 5.4 where $\frac{\Delta p}{p}$ is plotted versus θ for two different binning in θ itself. The figure corresponding to the rough binning shows a structure within each bin, indicating that a smaller bin size would be helpful. It should also be noted that this structure is non linear; pointing in the direction of a problem with bin sizes, rather than a problem with the fit routine. Moreover, such a plot allows to directly estimate what will the effect of reducing the bin size be, in terms of momentum resolution.

Another fact that needs to be taken into account is that **F** depends on the charge of the particle being tracked. The reason is that positrons are bent toward the beam pipe and electrons away from it. Then for a given entry point in the field, if the particle is an electron it will see a weakening field along its path while if it is a positron it will see a strengthening one[3], as shown in Figure 5.5 on page 106. Thus, for the same momentum the electron is less deflected; that is, F has different values. The immediate consequence of this is the need for two grids. One for positively charged particles and another one for negatively charged ones.

[3]This is not truth over the whole sector. The argument reverses for the lowermost polar angles. But the point is still valid that electrons and positrons behave differently.

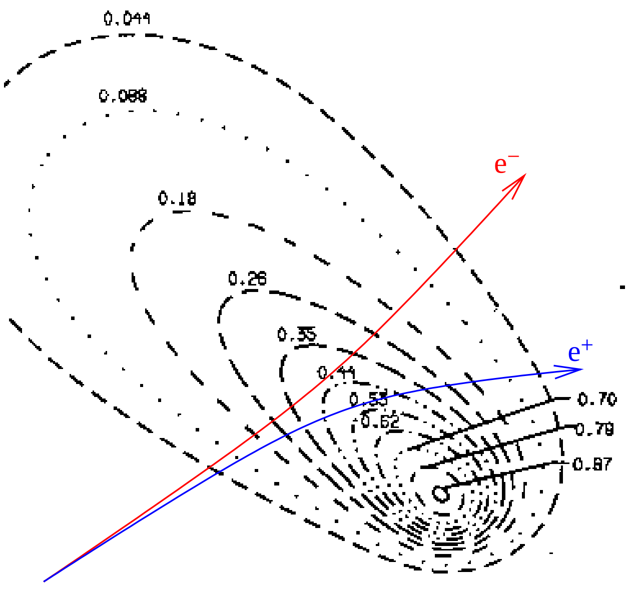


Figure 5.5: Dependency of momentum kick on charge

Numerical computation of F

For each grid point, **F** is computed using the HGeant Montecarlo program described in section 3.11. This package describes the passage of particles through matter, simulating the processes that affect those particles trajectories, including the effect of the magnetic field.

The basic procedure starts by writing a simple event generator for HGeant. Using the generator, particles are fed to the simulation package. HGeant propagates them through a model of HADES and their position at the detection planes is recorded. For each entry in the table, or grid vertex, a set of particles is shot. One of the important effects in HGEANT is of course the multiple scattering on the MDC foils and the air between MDCs. Therefore one wants to average over a few hundred trajectories. Since the fit routine can be said to work by comparison with these tracks, they are called “Reference Trajectories”, naming the algorithm.

It should be noted that it is not a good idea to use uniform distributions in the parameters and then perform a binning (see Fig. 5.6), because in such a case each point in the grid would get the mean value of **F** over the bin and if **F** is not linear, the mean is biased and it does necessarily correspond to the value at the grid point; this mostly affects the computation of derivatives during the fitting phase. The need to match the binning in the event gen-

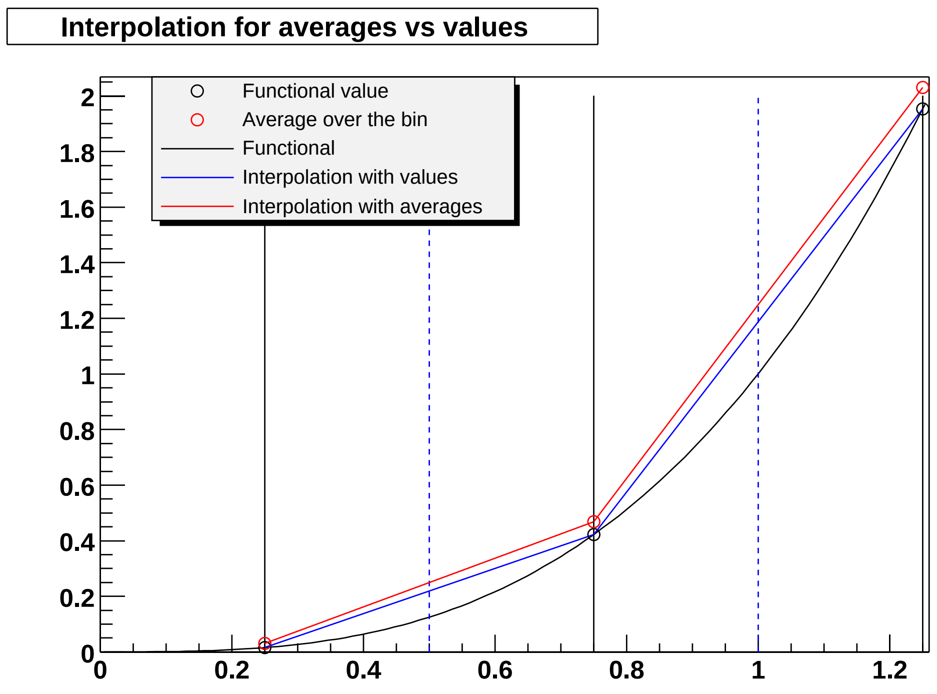


Figure 5.6: This figure shows the difference between interpolating taking the value of the function at the grid points (solid vertical lines), and the average over the bin. Bin boundaries are marked by the dashed vertical lines; the function model used as example is $f = x^3$

erator with the binning in the grid forces the need to recompute the whole sample whenever the binning changes.

We are using simulation to evaluate **F**, this means the method will be as good as the simulation is. That is the reason why a large effort has been made by the collaboration to have an accurate description of HADES in the simulation package; this means not only the geometry has to be reproduced in HGeant with great detail, but also the magnetic field itself. The field map was obtained by two methods, using a simulation package called Tosca, which describes iron less magnets very well; but also a measurement was made to crosscheck the result with discrepancies below 1% (see (8)).

Fig. 5.7 shows a component of **F** for a given grid point evaluated for 150 with the same parameter vector; this component corresponds to the y coordinate in MDC3. It can be clearly seen that the distribution is not even symmetric, let alone single valued. The asymmetry comes from the effect of "Energy Loss" which affects the track before reaching the region

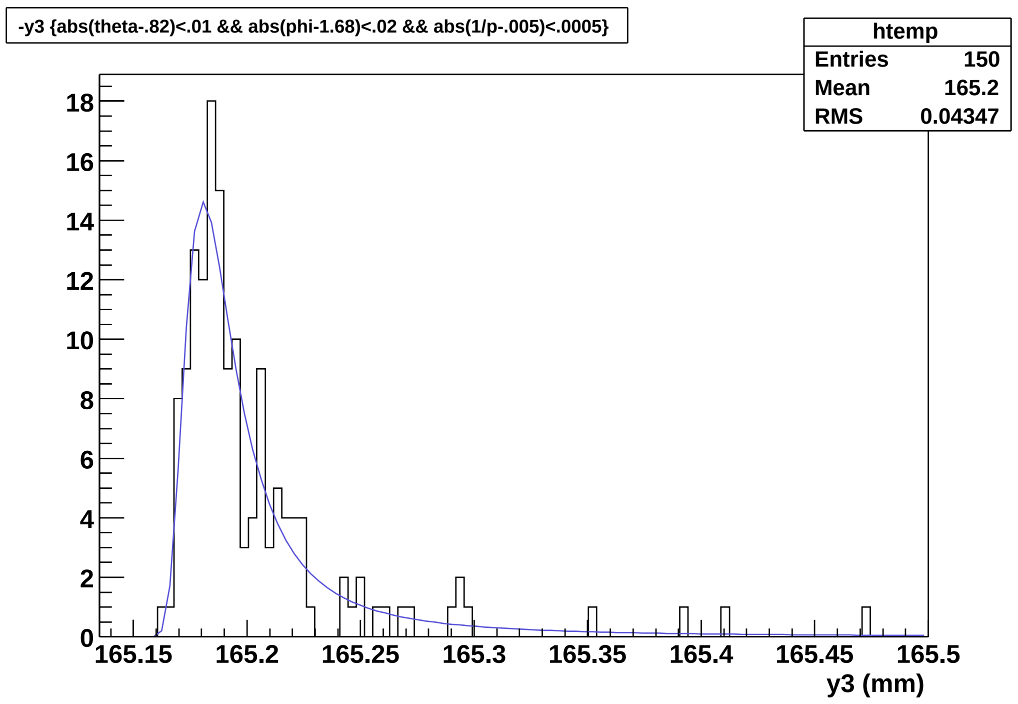


Figure 5.7: A component of F at grid point

of the magnetic field. Energy loss is, in this particular case, due to the interactions of the track with the air and it is a random process described by a Landau distribution. Each energy loss diminishes the particle momentum and that basically increases the deflection of the particle in the field. The larger the energy loss, the larger the momentum change and therefore the larger the alteration in deflection. Given that the incident direction is fixed it seems natural that the variable is also shaped like a Landau distribution and actually it is well fitted by one. The situation is simpler for the components representing the position of the track before the magnet, those are, within numerical resolution, single valued.

Therefore we have two kind of components in the measurement vector $\mathbf{x}_m$: the ones representing position before the magnet, whose parameterization is trivial, and the ones after the magnet. In this second case a solution would be to fit each distribution to a Landau for every single bin in the grid and taking the peak position (MVP). However, a test showed each fit taking close to 1.5 seconds[4]. Assuming a reasonable binning ($16 \times 6 \times 15 \times 18 \times 12$) this would translate into 126 hours of computing time, which is unpractical even

[4]Time taken on a Athlon XP1700 CPU. Running at 1.4GHz

if a factor could be get by optimizing the fitting code.

Instead, the chosen approach is to iteratively filter away the outliers. That is, the entries falling *far away* from the MVP. This way the mean of the filtered distribution and the MVP tend to be the same. To determine what *far away* means, the original distribution is taken for each component of **F** and for each bin. Then the mean value and RMS of the distribution are computed. Those two numbers are used to create a window centered at the mean and with a width of 2 times the RMS. The filtered distribution is obtained from the original one by filtering away the entries falling outside the window.

The resulting distribution can itself be filtered in the same way, repeating the procedure as many times as needed. In each iteration the function value is estimated by the mean. Typically 2 iterations are enough since the difference between the MVP of the Landau distribution and the calculated mean is well below the $80\mu m$ position resolution of the drift chambers.

This simple approach allows to parameterize a full set of reference trajectories in a matter of minutes.

5.2.3 Derivative matrix computation

As explained in 5.2.1 each iteration in the fit routine requires the computation of the A matrix with elements $A_{ij} = \frac{\partial F_i}{\partial p_j}$. This makes up for 40 derivatives which need to be computed in each iteration step. In principle it is possible to compute A for each grid bin in advance and store the results in a table. However such a technique is impractical due to the large amount of memory it requires, about 380Mb for a binning $(16 \times 6 \times 15 \times 18 \times 12)$. Actually dealing with such large arrays has also a possible impact in performance. The reason is that in todays computers the processor is much faster than what standard memory can support, therefore the need for large caches. A matrix of this size will not fit in the cache so each access will produce a cache miss in the processor, thus slowing down the operation. How big this effect is has not been tested, though.

Fortunately there are methods in the literature which allow to efficiently compute the derivatives of a tabulated function; as it is in our case. The one we have implemented is based on the so called Savitzky-Golay filters (see (2)).

Before going into the description of the method let's fix some notation. Let's note the indexes moving in the 5 dimensional grid by greek letters α=$(\alpha_1, \ldots, \alpha_5)$. Each point of the grid in the P space may be noted as

$\mathbf{p}_{(\alpha_1,\ldots,\alpha_5)}$. $F_i\left(\mathbf{p}_{\alpha_1,\ldots,\alpha_5}\right)$ is the i^{th} component of $\mathbf{F}(\mathbf{p})$ calculated at that point, where i runs from 1 to 8.

The idea is to compute the derivatives $\frac{\partial F_i}{\partial p_j}$ at any given point of the grid starting from the known values of $\mathbf{F}$ at the grid points themselves. For this purpose we make a least squares fit of a polynomial of order M to the values of F_i calculated at several points of the grid along the coordinate with respect to which we are differentiating, i.e., having different values of the α_j index, while keeping the remaining coordinates unchanged.

Let's suppose that the grid point where we are calculating the derivative has $\alpha_j = \lambda$ and all the points in the grid are equally spaced by some constant distance Δ. Then, the known values of the $\mathbf{F}$ function may be written as

$$f_k \quad = \quad F_i(\mathbf{p}_{\alpha_1,\ldots,\alpha_j=\lambda+k,\ldots,\alpha_5})$$

where $k = \ldots, -2, -1, 0, 1, 2, \ldots$

Usually it is not necessary, neither advisable, to perform the fit over the n_j points of the grid along the j coordinate; but only over just n points between f_{-n_L} and f_{-n_R}, being $n = n_L + n_R + 1$. Typical values are $M = n_L = n_R = 2$ which makes $n = 5$.

The idea is rather simple and potentially also quite expensive in terms of time complexity. Savitzky-Golay filters allow to reduce the task of fitting a polynomial and computing its derivative to a mere scalar product of two vectors of dimension n and one division. The basic idea is that since the least squares fit is a linear operation on the data, the coefficients of the fitted polynomial are themselves linear on the function values. Then, there is a set of coefficients c_i for which the process of polynomial least squares fitting and evaluation is reduced to:

$$g_\lambda = \sum_{k=-n_L}^{n_R} c_k f_{\lambda+k} \tag{5.6}$$

To derive such coefficients lets write down the LSM fit to a polynomial of order M, assuming the error in the $f_{\lambda+k}$ are the same for all k. Then, the functional to minimize is:

$$Q^2 = \sum_{k=-n_L}^{n_R} \left(f_{\lambda+k} - \sum_{m=0}^{M} s_m \lambda^m \right)^2 \tag{5.7}$$

The set of $\{s_m\}$ coefficients minimizing Q^2 is given by the solution of the system of equations:

$$\frac{1}{2}\frac{\partial Q^2}{\partial s_m} = \sum_{k=-n_L}^{n_R}\left(f_{\lambda+k}k^m - \sum_{j=0}^{M} s_j k^m k^j\right) = 0 \quad \forall m : 0, \ldots, M \tag{5.8}$$

Defining matrix S with elements

$$S_{km} = k^m \quad k = -n_L, \ldots, 0, \ldots, n_R \;\; m = 0, \ldots, M$$

and vectors $\mathbf{s} = (s_0, \ldots, s_M)$ and $\mathbf{f} = (f_{-n_L}, \ldots, f_{n_R})$. We can write the system of equations in matrix notation as

$$(S^t S)\mathbf{s} = S^t\mathbf{f} \Rightarrow \mathbf{s} = (S^t S)^{-1}(S^t\mathbf{f})$$

Now, the value of the polynomial at $k = 0$ is $s_0 = g_\lambda$ and the value of the first derivative is $\frac{s_1}{\Delta}$ where Δ is the grid step size[5]. Therefore the so called Savitzky-Golay coefficients $\{c_k\}$ are given by

$$c_k = \sum_{m=0}^{M}\left[\left(S^t S\right)^{-1}\right]_{0m} k^m$$

If what we want to compute is the derivative we use coefficients:

$$c_k = \Delta^{-1}\sum_{m=0}^{M}\left[\left(S^t S\right)^{-1}\right]_{1m} k^m$$

Since the Savitzky-Golay coefficients themselves are independent of the function values f_λ we can calculate all of them in advance and reuse them each time a partial derivative needs to be computed. It should be stressed that the key factor allowing the derivation above is that the grid points are equally spaced in each dimension.

Implementation considerations

The one problem of the method outlined above is evident when we need to perform the differentiation close to the edges of the grid. In that case we may not get enough points either to the left or the right. Then, the differentiation routine adapts itself by determining how many points can be used for the fit and choosing appropriate Savitzky-Golay parameters from a table. For example, table 5.1 shows the full set of coefficients for $n_L = 2$,

[5] $\Delta = b_j(\alpha_1, \ldots, \alpha_j, \ldots, \alpha_8) - b_j(\alpha_1, \ldots, \alpha_j - 1, \ldots, \alpha_8)$

M	n_L	n_R	Savitzky-Golay Coefficients				
2	2	2	-0.20	-0.10	0.00	0.10	0.20
2	2	1	-0.05	-0.35	-0.15	0.55	
2	2	0	0.50	-2.00	1.50		
2	1	2		-0.55	0.15	0.35	0.50
2	0	2			-1.50	2.00	-0.50
2	1	1		-1.50	2.00	-0.50	

Table 5.1: Sample Savitzky-Golay coefficients for different values of M, n_L and n_R

Another complication appears when differentiating with respect to ϕ near the $\phi = 0$ region. In this region we have to make use of the sector symmetry when looking at the neighbors.

5.2.4 Implementation considerations

The main concern implementing the fit routine explained in 5.2.1 has been its performance. That was the reason to implement Savitzky-Golay filters; but it was also the reason to build a customized linear algebra package, this package is inspired in the Blitz++ library (see (3)). The goal is to efficiently compute the matrix which appears in (5.5) multiplying $(\mathbf{x}_m - \mathbf{F}(\mathbf{p}_0)$, let's call it M for the rest of this section.

Direct inspection shows that computing M involves 3 matrix multiplications, one transposition and one inversion. The single-most standing point is the matrix inversion; but in fact, using the so called LU decomposition of a matrix[6] there is no need for a full matrix inversion, as we can use back-substitution which affords a factor 3 in the linear algebra, compared to calculating the inverse by Gauss-Jordan. The only place where we need the full inverse matrix is when computing the covariance matrix of the parameters; but that is something done after the last iteration in the fit.

As for the code itself it was crafted to make it as efficient as custom Fortran code, but with the added re-usability inherent to Object Oriented Programing (OOP). The all too common mistake in C++ when dealing with matrices is to define a matrix class and overload the arithmetic operators to return objects. Then, code as innocent as this

[6]The LU decomposition consists in expressing a matrix as the product of other two: L and U, both of them triangular. The L matrix has all elements above the diagonal equal to zero, correspondingly U has the elements below the diagonal equal to zero. See ref (9).

```
Matrix a(3,3),b(3,3)
Matrix c(3,3)
c = a * b
```

is actually performing significant work behind the scenes. The line c=a*b creates a temporary matrix, multiplies a and b into it and then copies the result to c element by element. The Fortran equivalent would afford the intermediate step. Therefore in our implementation there is no * operator for matrices and only *= is allowed.

The same happens when transposing a matrix, the obvious solution would be to create a method `transpose()` returning the transposed matrix. Again, this involves copying the matrix elements one by one. A better way to do it is to realize that a matrix and its transposed are two views on the same data. Then the best thing is to separate the *view* and the *data* (also referred as *model*) in two different classes: HRtMatrix for the view and HRtMemoryBlock for the data[7].

Then a matrix and its transposed can be represented by two instances of HRtMatrix on the same HRtMemoryBlock object. Since the model can be shared by several views it has to be reference counted to avoid memory leaks and multiple destruction. The performance gain comes because HRtMatrix objects are now lightweight. Besides it is possible to create the views in advance and keep them around, when new matrix data is computed both views are automatically updated at the same time.

5.3 "Kick plane" algorithm

The kick plane algorithm differs from the reference trajectories in that it models the track's trajectory inside the field region. We will see that even if the full track model is unknown, its momentum can be approximated by the track's deflection alone. It will be shown that the deflection of a particle in a magnetic field is proportional both to the field strength and the distance traveled by the particle in the field. Therefore one can replace the original field by another compressed in space, but increased in strength, such that the effect stays the same. Taking this process to the limit, the field is compressed into a 2 dimensional sheet, the kick plane, where all of the deflection takes place.

[7]For the literate in Object Oriented Design, this is just an application of the design pattern known as "Model View Controller". (see (4)).

In order to cope with the field's inhomogeneities the field region is divided in cells where we can assume the field to be constant. This requires the building of a grid, like for reference trajectories. The difference is basically in the size of the grid. In some sense the kick plane algorithm is a derivation of reference trajectories where we have put some more physics knowledge in.

Given that all six HADES sectors are symmetrical we will concentrate in one of them. Everything said in this section refers to the first, uppermost, sector, unless noted. All other sectors are equivalent.

5.3.1 Charged particle motion in a magnetic field

Let's start by assuming there is not electric field in the region where the magnet sits. That is the magnetic field is static $\frac{\partial \mathbf{B}}{\partial t} = 0$. Then the motion of a single charged particle in the field is governed by the Lorentz force. Its equation of motion can be written as (see (5))

$$m\frac{d^2\mathbf{r}}{d^2t} = q\frac{d\mathbf{r}}{dt} \times \mathbf{B} \tag{5.9}$$

Defining the trajectory length coordinate

$$d\lambda^2 = |d\mathbf{r}|^2 = dx^2 + dy^2 + dz^2$$

(5.9) can be written as

$$\frac{d^2\mathbf{r}}{d^2\lambda} = \frac{q}{p}\frac{d\mathbf{r}}{d\lambda} \times \mathbf{B}$$

This equation has a nice geometrical interpretation, since $\frac{d\mathbf{r}}{dt}$ is a unit vector tangent to the particle's trajectory at any point. $\frac{d^2\mathbf{r}}{d^2t}$ is a unit vector perpendicular to the track and with length $\frac{1}{\rho}$, where ρ is the track's curvature radius.

Defining α as the angle between $\frac{d\mathbf{r}}{dt}$ and $\mathbf{B}$. We can write

$$\begin{aligned}\frac{1}{\rho} =& qB\frac{\sin(\alpha)}{p} \\ psin(\alpha) =& q\rho B\sin^2(\alpha)\end{aligned}$$

Where $p\sin\alpha$ is the component of the momentum perpendicular to $\mathbf{B}$ and $\rho\sin\alpha$ is the curvature of the track trajectories projection into a plane perpendicular to $\mathbf{B}$

Given the geometrical interpretation of $\frac{d^2r}{d^2\lambda}$ we can write the infinitesimal deflection of the track as

$$d\xi = \left|\frac{d^2\mathbf{r}}{d^2t}\right| d\lambda = \frac{q}{p} B\sin(\alpha) d\lambda \tag{5.10}$$

If deflection is additive, that is, if the infinitesimal rotations happen around a fixed axis, the total deflection of a charged particle in a field can be obtained from the integral

$$\Delta\xi = \frac{q}{p}\int_{\lambda_1}^{\lambda_2} B\sin(\alpha) d\lambda$$

5.3.2 Obtaining momentum from deflection

Let's assume the field is confined to some spatial area, like in the case of HADES and most other spectrometers. Then, for any given track, the field can be expressed as

$$\mathbf{B} = \begin{cases} \mathbf{B}(\lambda) & \lambda_1 < \lambda < \lambda_2 \\ 0 & otherwise \end{cases}$$

The particle's deflection can also be calculated making use of the fact that the Lorentz force is perpendicular to the momentum, then $|\mathbf{p}|$ remains constant and only the momentum direction changes. Let $\mathbf{p}_1$ be the momentum at $\lambda = \lambda_1$ and $\mathbf{p}_2$ the momentum at $\lambda = \lambda_2$. Then, as shown in Fig. 5.8

$$|p_1| = |p_2| \Rightarrow p_k = \Delta p = 2p\sin\left(\frac{\xi}{2}\right)$$

In differential form and making use of the fact that $\sin(\xi) \simeq \xi$ for small angles, we come to

$$dp_k = pd\xi$$

Using (5.10)

$$dp_k = qBsin\alpha d\lambda$$

In the HADES case $\sin\alpha = 0$. Also, the field being toroidal implies deflection taking place in one plane. Then it is legal to integrate, that is, for each particular track

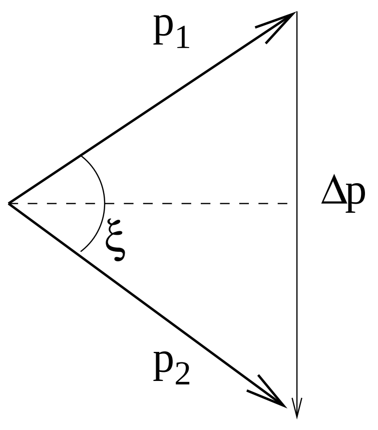


Figure 5.8: Track deflection

$$p_k = q\int_{\lambda_1}^{\lambda_2} B d\lambda = K(\lambda_2 - \lambda_1)$$

where K is a constant, different for each track and $\lambda_2 - \lambda_1$ is the track's path length inside the field. Intuitively we can expect the path length to depend essentially on the tracks deflection, the larger the deflection the larger the path length. So it is justified to make an ansatz $\lambda_2 - \lambda_1 = F(\xi) = f(sin(\xi/2))$

Performing a Taylor expansion in powers of $\sin(\xi/2)$ around $\sin(\xi/2) = 0$ we get

$$f = a + b\sin(\xi/2) + c\sin^2(\xi/2) + O(sin^3(\xi/2))$$

Therefore, the momentum and deflection are related by

$$p = \frac{p_k}{2\sin(\xi/2)} = \frac{A}{\sin(\xi/2)} + B + C\sin(\xi/2) \tag{5.11}$$

The crucial point now is to realize that A, B and C depend essentially on the magnet properties alone. Or, in other words, on the spatial region being traversed by the particle. This implies we can extend the previous equation to arbitrary tracks where A, B, C are functions of the polar and azimuthal angles.

Once we have parameterized A, B, C. Eq. (5.11) allows to calculate momentum from particle's deflection. From the derivation of this formula, it is also clear that the model is better for the smaller deflections, that is, larger momenta.

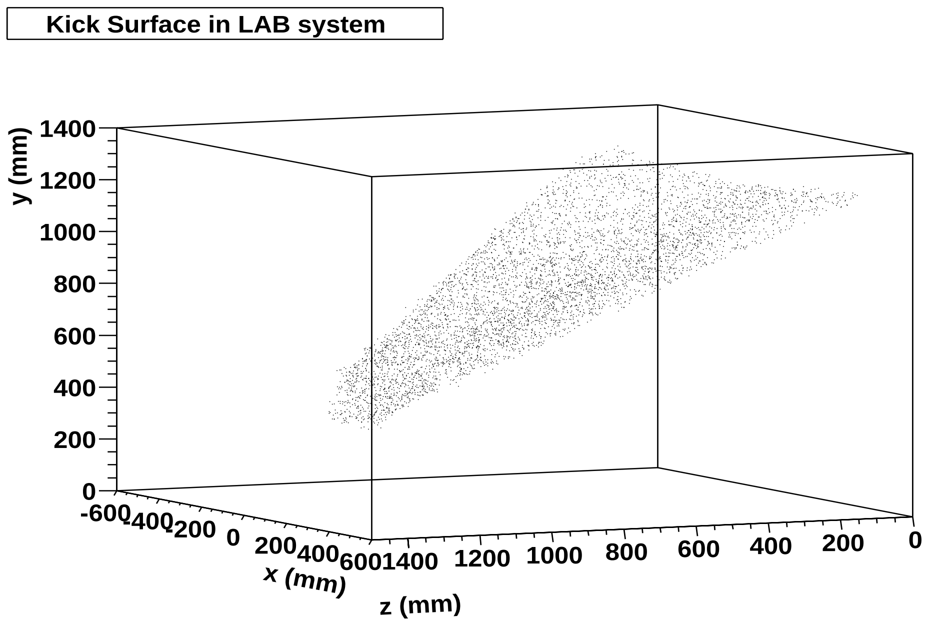


Figure 5.9: 3D view of the kick surface. z axis is the beam-line, y is the gravity axis.

5.3.3 Kick surface

Outside the magnetic field region, the trajectory of any given particle can be approximated by a straight line. We will use the word *segment* for such a track piece which is straight. In HADES, each track has two such segments, one before the magnet and another one after.

Because of the toroidal magnetic field, for each track in HADES the deflection happens in one plane. Then we expect for two segments belonging to the same track to meet at one point. In other words the corresponding straight lines should cross. In practice, numerical and resolution uncertainties prevent this from happening exactly. It, then, makes sense to define the cross point between two segments as the point of closest approach between them. Let's note these cross points for a set of N tracks by $\mathbf{r}_i = (x_i, y_i, z_i)\ \forall i : 1 \ldots N$.

A method on how to calculate the cross point has already been given in Chapter 4. Indeed, the cross point between segments is nothing else than the vertex defined by those two segments. An interesting observation is that if we make a 3D plot with the coordinates of such points, see Fig. 5.9, we discover that they lay in a surface; this is the so called *kick surface.* Knowing about it, is interesting for several reasons.

As we discussed before, the parameters A, B and C are functions of the field region traversed by the particle. Given a track which enters the magnetic field with a certain direction, it will always hit the same spot in the kick surface, regardless of its momentum. Then it is natural to think that A, B, C can be parameterized as functions of position on the kick surface.

By knowing the kick surface we don't need any more two segments in order to obtain a particle's deflection. It is enough with just one segment and one point from the second. The intersection of the first segment with the kick surface gives us a point which also belongs to the second segment by construction of the kick surface. Then, we have two points on the second segment, which completely define it, allowing us to compute the deflection.

During the construction phase of HADES, as more detectors are made available, it happens that while the inner segment of a track is usually well defined, one cannot say the same about the outer. Missing outer detectors have the effect that the direction information after the magnet is either missing or of low quality. The consequences of such partial setups are discussed in detail in other sections of this same chapter. For the moment it is enough to know that the kick surface's equation is very helpful in those cases.

Kick surface parameterization

Even though we have been speaking of defining a track's segment only where there is no magnetic field, the spectrometer's implementation schedule requires for the kick plane algorithm to be functional even when the only outer detector is the MDC3 chamber. In this case the detection plane is still inside the field region. We can redefine the outer segment as the given by the tangent to the track in the detection plane. With this definition the kick surface still exists and it is still possible to use it. However, it comes as no surprise that the parameterization function is different than in the case where the track is measured outside the field region. For this reason, in this section we will only concern ourselves with the generics on how the process of parameterizing the kick surface proceeds given a surface model. The models for the different setups are detailed in the relevant sections: Section 5.4.1 and Section 5.4.2.

A kick surface model is nothing else but the equation of a surface in the space. That is, we start from a model

$$y = f(x, z)$$

Not any model will do. First of all it has to respect the symmetry of a sector; that is

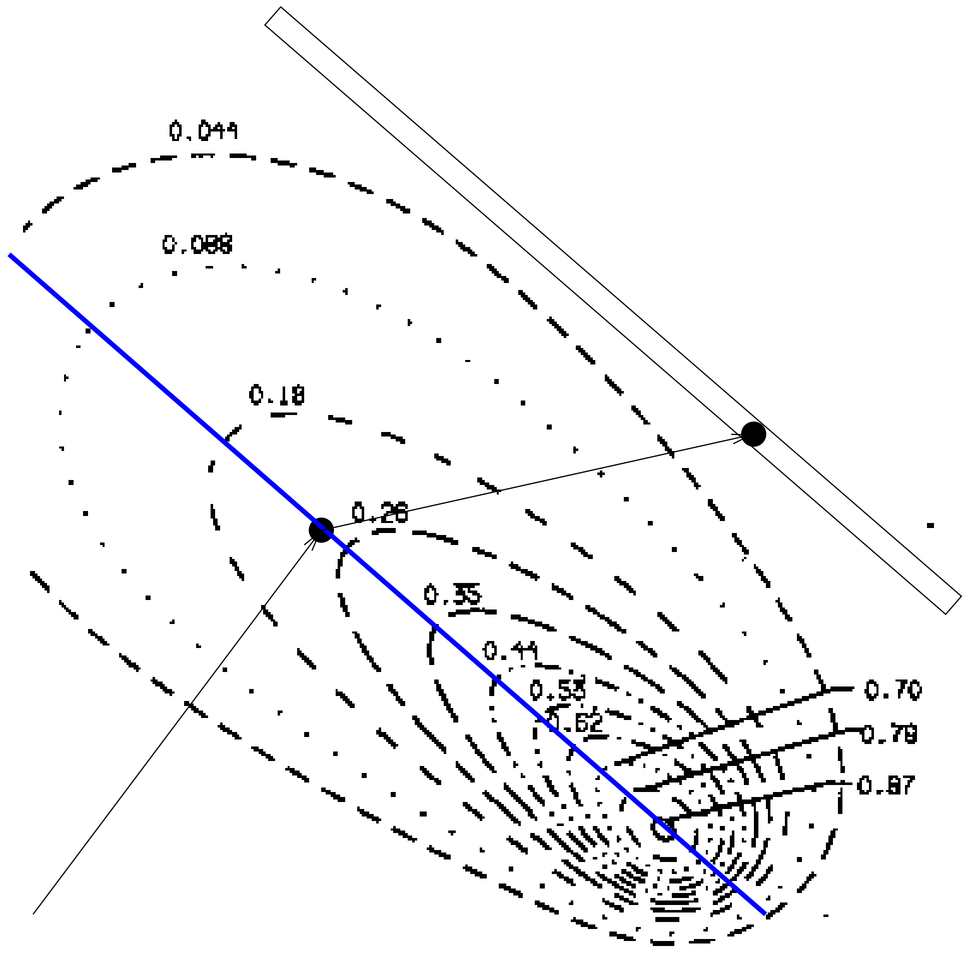


Figure 5.10: Using the kick plane to determine deflection using just one point in the outer region

$$f(x,z) = f(-x,z)\ \forall x$$

Another condition is for f not to be too complex. The reason being the need to compute the intersection of an arbitrary straight line with the surface. As we will see, this computation is performed for each track candidate during software operation. Therefore it not only has to be analytical, but also involve a not too large number of operations. The surface model is obtained empirically, plotting figures like 5.9 as well as its projections. For any given surface model, the surface parameters can be determining with a Least Squares Fit, minimizing the functional

$$Q^2 = \sum [y_i - f(x_i, z_i)]^2$$

In order to obtain the data points (x_i, y_i, z_i) in the fit we could use either simulations or real data. However, with real data we have to confront a number of problems

- Real detectors have finite resolution, which translates into a smearing of the data points. The consequence is the need for larger statistics than if the data points where only affected by numerical resolution.
- It is in general not trivial to match the inner and outer segments of a track. The easiest solution to overcome this issue is selecting events with a multiplicity of one. This represents a significant reduction of available statistics.
- During the first phases of the construction of HADES the information about the outer segment is simply missing since the MDC detectors are missing in the rear part of the spectrometer. In this case it is not possible to use real data to determine the kick surface.

Using a HGeant simulation to obtain the cross points $\mathbf{r}_i$ is the preferred option. The problem with simulations is how well do they reproduce reality; but the motion of a charged particle through a static magnetic field is a well known process. Besides, the magnet is iron less, so it is well described by simulation packages; not only that, but the field map given by simulation has also been cross checked with reality.

Either way, once we have a set of data we need to clean it up. Low momentum tracks ($\sim$50MeV) curl in the magnet losing all information about the incident direction. This kind of curled track cannot be well described by the kick plane algorithm; in particular it does not give raise to a kick surface.

So we need to take them out of the data sample used for the fit. Note that we are not interested in this kind of track. It is very difficult to match such a track with a segment in the outer region of HADES, let alone calculate its momentum.

This is not as big a problem as it may seem. In the production phase of HADES, when examining the in medium modifications to the vector meson properties, we are anyhow not interested in such low momentum particles.

However, that momentum region is interesting when studying π^0 Dalitz decays. This is useful in order to understand the behavior of the detectors and analysis in realistic conditions, because π^0 Dalitz is a well known reaction. For that kind of situation it is enough to downscale the field intensity and then we would see curling for tracks at 5 MeV instead of 50 MeV.

Since such tracks correspond to low momenta, it is enough to place a momentum cut to eliminate them. The value in the momentum cut is not really critical. These low momentum tracks manifest themselves as outliers during the fitting process. Even though LSM gives them too much weight they typically represent only a few percent of the total number of data points used. It is actually possible to perform a fit without any cut and for each $\mathbf{r}_i$ compute d_i as the distance between the point $\mathbf{r}_i$ and the kick surface. Then, we can obtain a reasonable momentum cut by simple inspection of a plot of d_i vs momentum. Such a plot is shown in 5.11.

The fitting itself is performed using the MINUIT package from CERN, which comes integrated in ROOT. This allows us to keep the surface model undefined until the very last moment, enabling us to easily check different possibilities. Note that the slowness of Minuit is not a problem since the process of finding out the kick surface is done very seldom.

5.3.4 Kick plane parameterization

The task at hand is to obtain the form of the functions A, B and C in (5.11). However, we will first spend some time looking to the 0^{th} order approximation of that formula. If we stay at 0 order (5.11) becomes

$$p = \frac{A}{2\sin(\xi/2)} \tag{5.12}$$

In this case A has a clear physical meaning. Comparing equations 5.12 and 5.11 we can interpret A as the momentum kick of the magnet, see Fig. 5.8. Getting familiar with it provides knowledge about the magnet properties, and helps understanding the results in the following sections. Furthermore,

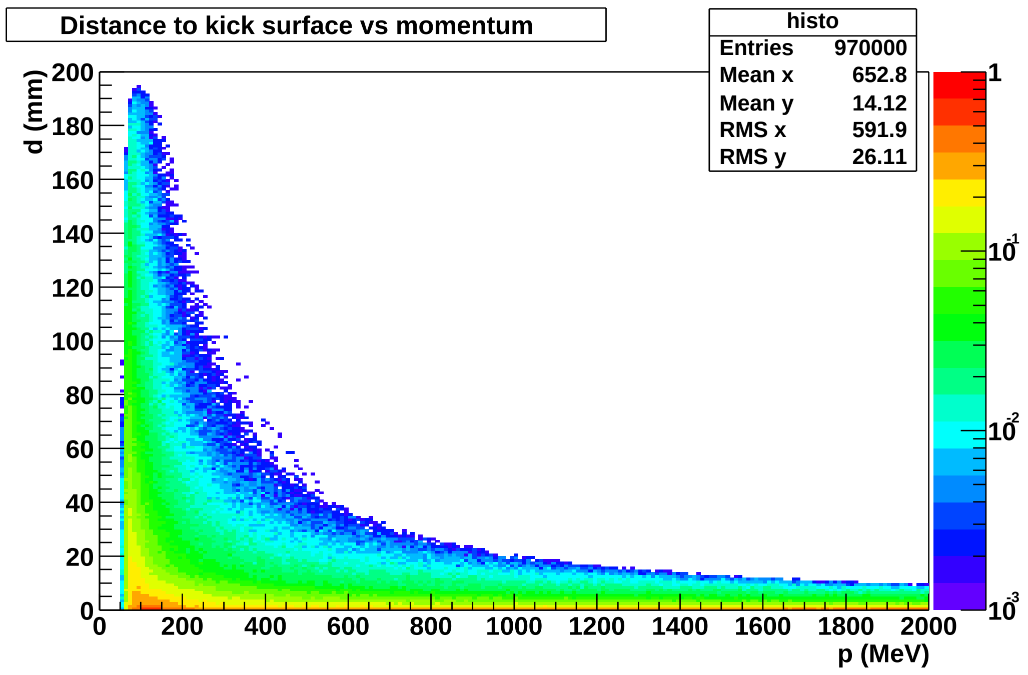


Figure 5.11: Distance to kick surface (d_i) vs momentum. The color scale indicates number of entries relative to the maximum. Note the logarithmic scale.

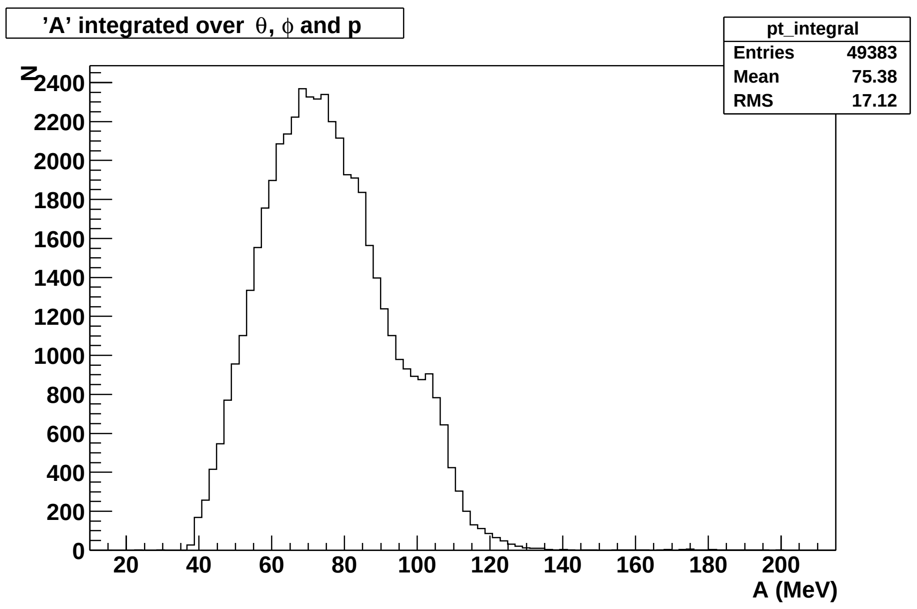


Figure 5.12: Integral spectrum of the magnet's momentum kick (A)

the basics of the parameterization process are the same as compared to the full fledged model.

Once more, analytical derivation of A is an arduous task given the inhomogeneous nature of the magnetic field. Instead, we can use HGeant to obtain it numerically and then either parameterize A with an empirical model, or tabulate its values.

For any given track traversing the field we can compute A if we know the track's momentum and the deflection it suffered. HGeant, being a simulation, is able to provide us with that information; the details on how this is done differ from one setup to another and they are explained in the corresponding sections.

In order to get a glimpse on how A looks like we only need to shoot particles through HGeant. If this is done with uniform distributions in momentum and solid angle what we get back is an integral spectrum of A like the one shown in 5.12. From that figure we learn basically two things: the mean magnet's momentum kick is around 75 MeV and it ranges from 40 to 140 MeV. If we were to approximate the momentum kick by a constant our resolution would be dominated by the width in this distribution. So we could estimate a resolution around 20%, very far from what we want to get. In order to improve that number we need to disclose the ingredients summing up in the integral spectrum.

Since we are staying at 0^{th} order in powers of deflection angle, we should expect one of the ingredients to be the deflection itself, equivalently momentum. This can be taken care of by going to higher orders as shown later; but in fact this is not the main factor. As already noted in Fig. 5.12, A plays the role of the momentum kick. As shown in 5.3.2, this is given by the path integral of the field along the track.

$$p_k = q \int_{\lambda_1}^{\lambda_2} B d\lambda$$

Since B is not constant, and in fact strongly depends on position, it is natural to think that A must depend on the geometrical region of the magnetic field being traversed.

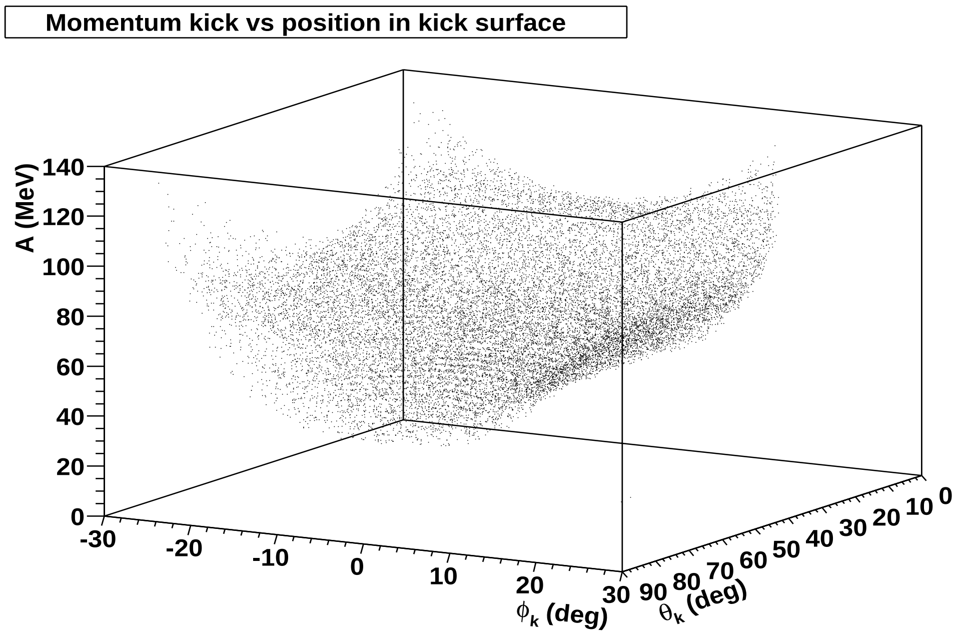
As noted in section 5.3.3 the existence of a kick surface suggests that A can be parameterized as a function of position over the kick surface itself. Thus, for each track, we need not only to record p and ξ, but also a point on the kick surface. Following the scheme on Fig. 5.10 we store the intersection point, $\mathbf{k}$, of the inner track segment with the kick surface. In cartesian coordinates $\mathbf{k} = (x_k, y_k, z_k)$, since the kick surface is a surface we only need two coordinates to specify a point on it. Let them be the well known polar and azimuthal angles from polar coordinates:

$$\begin{aligned}\phi &= \arctan\left(\frac{x_k}{y_k}\right) \\ \theta &= \arccos\left(\frac{z_k}{\sqrt{x_k^2 + y_k^2 + z_k^2}}\right)\end{aligned}$$

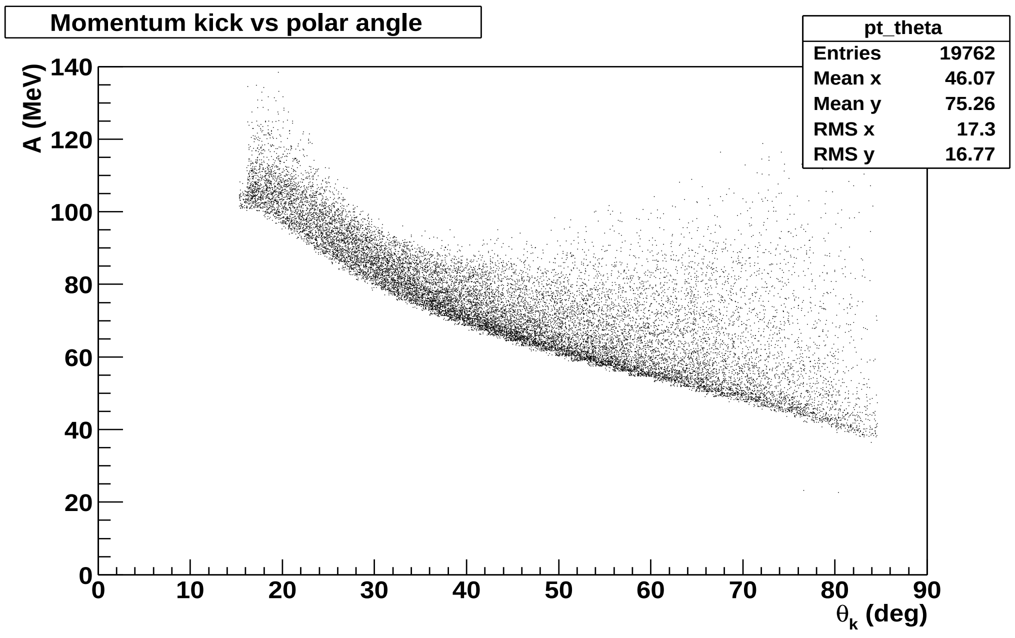
In principle, in order to obtain A it would be enough to shoot a sample of particles with uniform distribution in (θ, ϕ); then examine the resulting distribution, come up with a phenomenological model which depends on parameters α, and use LSM to fit those parameters. Fig. 5.13 shows a set of data obtained with the previous procedure.

From the figure on the facing page we learn several things. First and foremost, A does depend on θ and ϕ; thus our hypothesis $A \equiv A(\theta, \phi)$ makes sense. Looking a bit in more detail to the figure we notice a strong dependency with θ. This dependency essentially responds to the law

$$A = \frac{\alpha_1}{\rho}$$



(a) As function of θ and ϕ



(b) Projection of (a) into the A,θ plane. The observed spread comes from the ϕ dependency

Figure 5.13: Momentum kick on the kick surface.

where $\rho = \sqrt{x_k^2 + y_k^2}$. This dependency is to be expected from the magnet's geometry and just reflects the fact that the field is stronger for the lower polar angles. This simple formula already explains the most outstanding features of A. In order to further refine the parameterization we can plot $A\rho$ versus θ and ϕ as shown in Fig. 5.14. From the leftmost sub-figure we learn that most of the dependency with θ has been taken care of, but some still remains. From the rightmost one we learn that A is flat for most of the sector and increases toward the sector edges; that is the region where the magnet coils sit and therefore the field increases. Already from the figure we can intuitively think of a bi-quadratic polynomial to account for the ϕ dependence. The quartic term is needed in order to reproduce the nearly flat region around $\phi = 0$, besides odd powers of ϕ are not acceptable since they would introduce asymmetries. Then, our next model would be

$$A = \frac{\alpha_0 + \alpha_1 \phi^2 + \alpha_2 \phi^4}{\rho} \tag{5.13}$$

In fact, from 5.14 we also learn that the curvature of the bi-squared polynomial should change with θ. In other words $\alpha_i \equiv \alpha_i(\theta)$. In particular $\alpha_0(\theta)$ is a correction from the $1/\rho$ dependency.

The simplest way to get a feeling on how $\alpha_i(\theta)$ looks like is to plot A versus ϕ for different slices in θ and then, for each slice, perform a fit to a bi-squared polynomial as shown in Fig. 5.15. The evolution of the fit parameters with θ is shown in Fig. 5.16. What we see is a more or less linear region between the two coil edges at low and high polar angles.

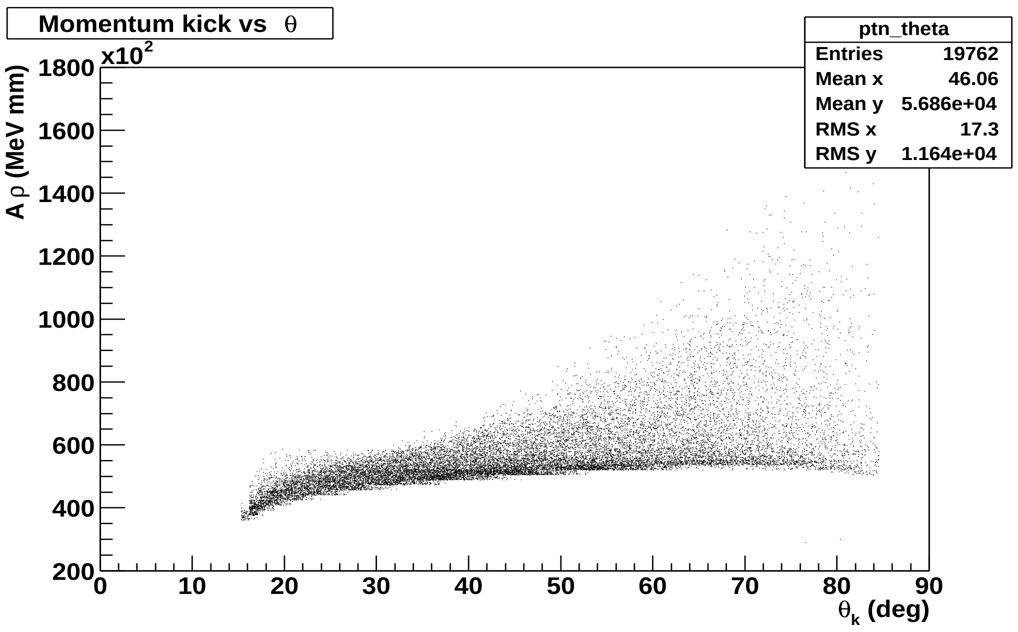
The one at lower angles corresponds to the higher field, but the short distance between left and right coils makes the field very compact. That's why there is not much difference between the middle of the sector ($\phi \simeq 0^o$) and the edges ($\phi \to 30^o$)[8]. This translates into relatively low values for α_1 and α_2.

The situation is very different at high polar angles. There, the left and right coils are much more separated and the field changes more from the middle of the sector to the sector edge[9]. That's why we see an increasing α_1 and a exploding α_2.

We also see how the larger momentum kick happens not for the lower polar region, where the field is stronger, but for the high polar angles, with a weaker field. This behavior cannot be due to field intensity and the only possibility left is the path length; it must be larger for tracks at high polar

[8]Up to a factor 3

[9]Up to a factor 6



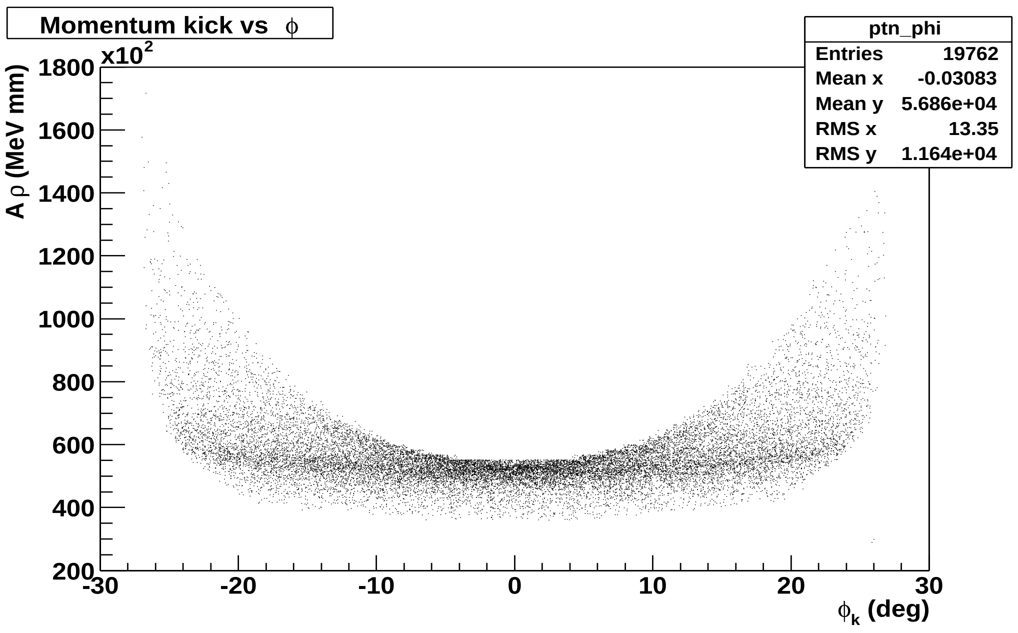


Figure 5.14: Refining the momentum kick (A) parameterization

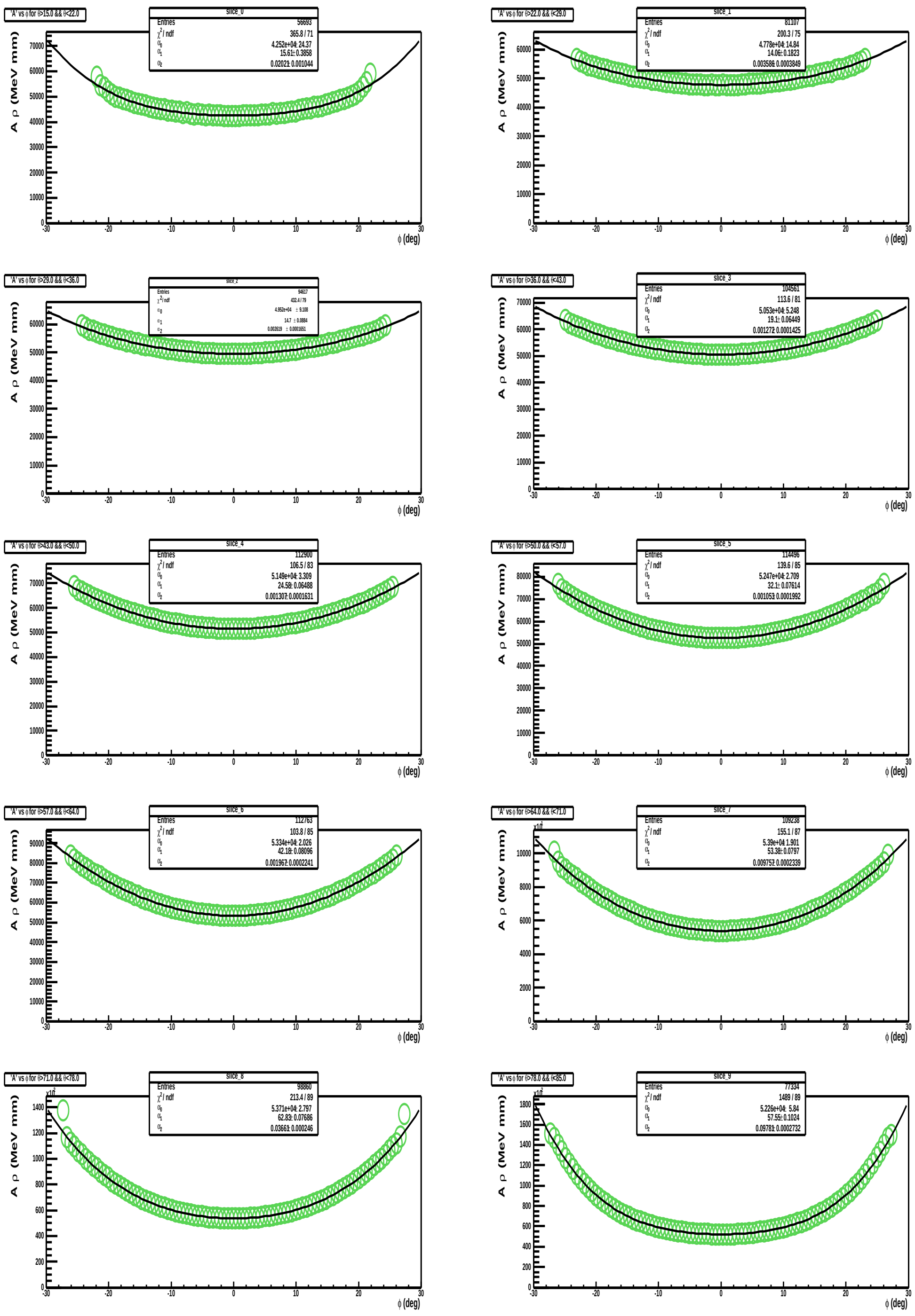


Figure 5.15: Curvature parameters (α_i) for different slices in θ

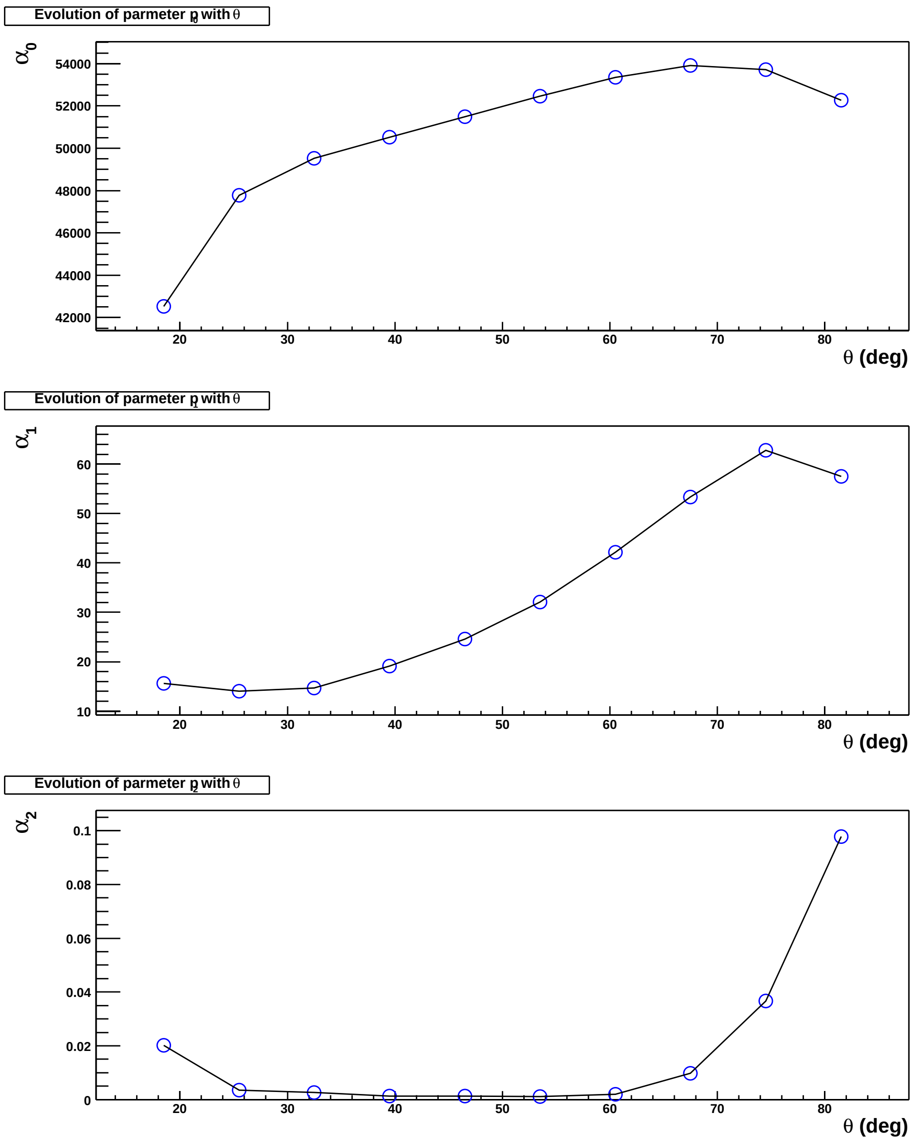


Figure 5.16: Evolution of α_i with the polar angle θ

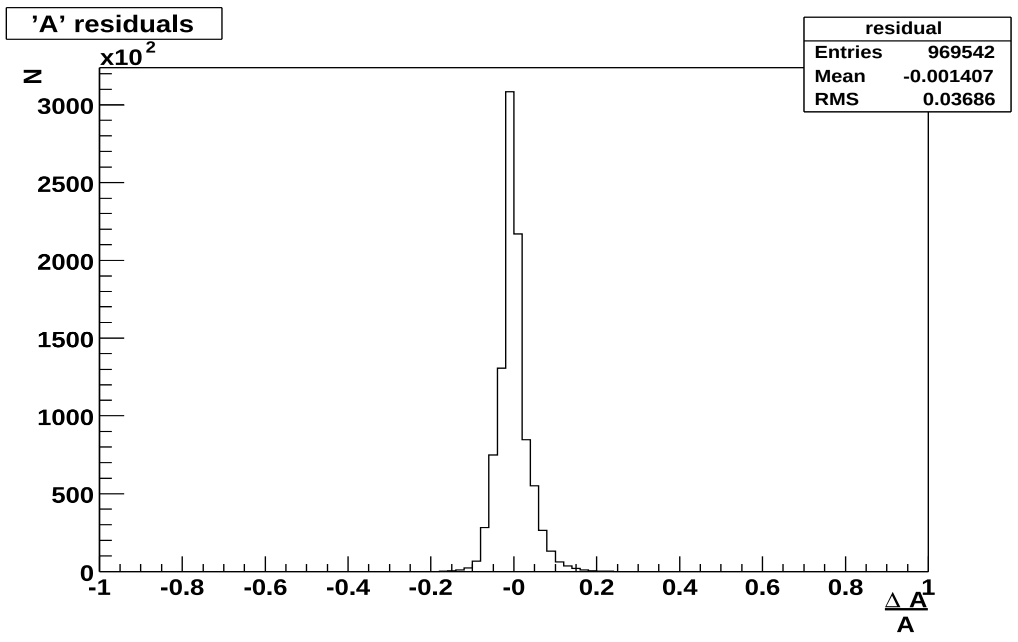


Figure 5.17: Residuals on momentum kick parameterization

angles. That is indeed the case. The coils being more separated make the field to spread over a larger spatial region. Close to the sector's center the increase in path length is not able to compensate the lower magnetic field, but close to the sector edges the field increases and we see how the two factors conspire to boost the momentum kick. Special care was put during the design of the magnet to minimize this feature. Indeed, the tracks traversing HADES at high polar angles are those of lower momenta; if they experiment a too high momentum kick we won't be able to measure them well.

To get an idea on how well we are reproducing the features of the momentum kick, we can do a simple test. Taking the plots in Fig. 5.16 we can use linear interpolation between the data points to estimate the values of α for any θ and then use equation 5.13 to compute the expected value for A. This way we can obtain a residuals distribution like the one shown in Fig. 5.17, which corresponds to a width around 3%. This number can be improved, in particular dropping the 0^{th} order approximation in favor of a higher order one. But that is the task for the following section.

$O(\sin(\xi)^2)$ parameterization

The parameterization method presented in the previous section has the advantage of providing some insight on what the magnetic field features are.

With enough love, its resolution can also be improved, but there is a much simpler and pragmatic way to tackle the problem. That is, simply build a table in θ and ϕ, storing the parameter's value for each bin. This solution is very easy to implement, and more flexible. The downside is that building such a table eats up computer memory. This is, however, not as much of a concern as it was with reference trajectories. The reason for that is that our table is now 2D instead of 5D. Even choosing a bin size of half a degree both in θ and ϕ the memory needed for one such table is around 33Kb; with todays memory sizes, that is negligible.

As we will see it will not be enough with just one table. If at some point the required amount of memory starts to be significant we can apply the tricks explained in the previous section to reduce it.

In fact it is already clear that we will need 6 such tables ($\sim 180Kb$) for a $O(sin(\xi)^2)$ parameterization. The 6 comes from the fact that we have three parameters sets A, B, C which are different for positive charged particles and negative charged particles. The reason why we need to differentiate between positive and negative particles is the same as for the reference trajectories algorithm; because they bend in different directions and since the magnetic field is inhomogeneous, the $\int Bd\lambda$ is different. This is more evident from Fig. 5.5 on page 106.

The basics of the parameterization process are the same as before. The goal is to obtain A, B, C as a function of θ and ϕ on the kick surface. For that purpose, we start from a sample of tracks simulated with the HGeant package. The first difference is that we cannot compute all the parameters for each single track, indeed, we need to accumulate at least three tracks with the same θ, ϕ. At this point using a table in (θ, ϕ) becomes natural. For each cell in the table we store the (p, ξ) of all tracks falling into that cell; then we can just perform a LSM fit to obtain A, B, C in that cell.

Let's start by fixing some notation. Mathematically, making a table is partitioning the θ and ϕ variables. Let's note by greek letters the indexes running on the table, so

$$
\begin{aligned}
\theta_\alpha &= \theta_{min} + \alpha\theta_{step} \qquad \forall\alpha : 1..N_\theta \quad \theta_{step} = \frac{\theta_{max} - \theta_{min}}{N_\theta} \\
\phi_\beta &= \phi_{min} + \beta\phi_{step} \qquad \forall\beta : 1..N_\phi \quad \phi_{step} = \frac{\phi_{max} - \phi_{min}}{N_\phi} \\
F_{\alpha\beta} &= F(\theta_\alpha, \phi_\beta) \qquad \forall F \in (A, B, C)
\end{aligned}
$$

Then, for each table cell, identified by a (α, β) pair, we need to find A, B, C minimizing the functional

$$Q^2(A_{\alpha\beta}, B_{\alpha\beta}, C_{\alpha\beta}) = \sum_{i=1}^{N_{\alpha\beta}} w_i \left(p_i - \frac{A_{\alpha\beta}}{\sin \xi_i} - B_{\alpha\beta} - C_{\alpha\beta} \sin(\xi_i/2) \right)^2$$

where $N_{\alpha\beta}$ is the number of tracks in the cell and w_i is a constant weight for track i. There are two obvious considerations to be made about the previous functional

- First is the lack of a σ_i term to take care of errors. This is due to the fact that both p_i and ξ_i are not real measurements but they are determined from simulations instead. Then we can assume the only errors affecting them are the numerical ones and numerical errors should be the same for all tracks. In other words $\sigma_i = \sigma \;\; \forall i = 1..N_{\alpha\beta}$. If this is the case, choosing $\sigma = 1$ will not change the position where Q^2 is minimum.
- From the functional form we can see how the different parameters are more sensible to different regions in $\sin \xi$. In effect, A is responsible for the behavior at the smaller deflection angles, while C dominates for the larger ones. Thus, in order to consistently estimate the three parameters we need the data sample in each cell to follow a uniform distribution on $\sin \xi$, or equivalently, a uniform distribution in $1/p$.

There are a number of problems appearing when trying to implement this scheme. The first problem comes from the sheer amount of data. We typically have around 100 tracks per bin for the fit. In order not to store all that information at once in memory what is done is an incremental fit. Every time a new track is read the fit parameters are updated for the cell where the track falls. The advantage of this approach, besides speed, is a constant memory size: it can cope with any number of tracks per cell. Fig. 5.18 shows a typical data sample.

The next problem comes from low momenta tracks. A track with momentum close to or below the magnet's momentum kick cannot be well reproduced by the kick plane algorithm, as its trajectory may even make loops. This kind of tracks manifest themselves when trying to perform the fit as outliers (see Fig. 5.19) for an example. When parameterizing the kick surface we took the simple approach of applying a momentum cut in order to get rid of these outliers: this we could do because the kick surface itself was not very sensible to such a momentum cut. However such a simple method cannot be used here. Let's say we apply a rather strong momentum cut, like 200 MeV; in those regions where the momentum kick is around 40 MeV we

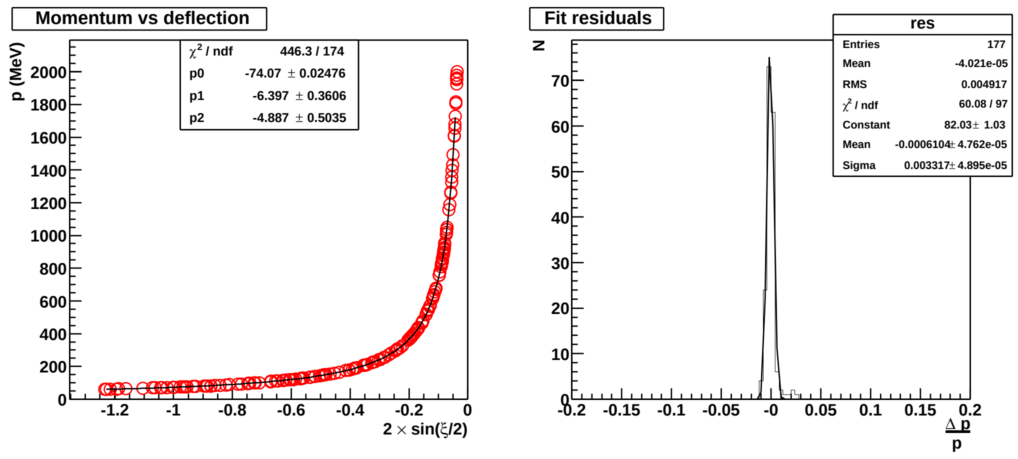


Figure 5.18: Sample fit at $\theta = 40^o$, $\phi = 10^o$. The left side shows momentum versus deflection in the bin, the red circles are the data points and the solid line is the fit. The rightmost figure shows the fit residuals fitted to a Gaussian.

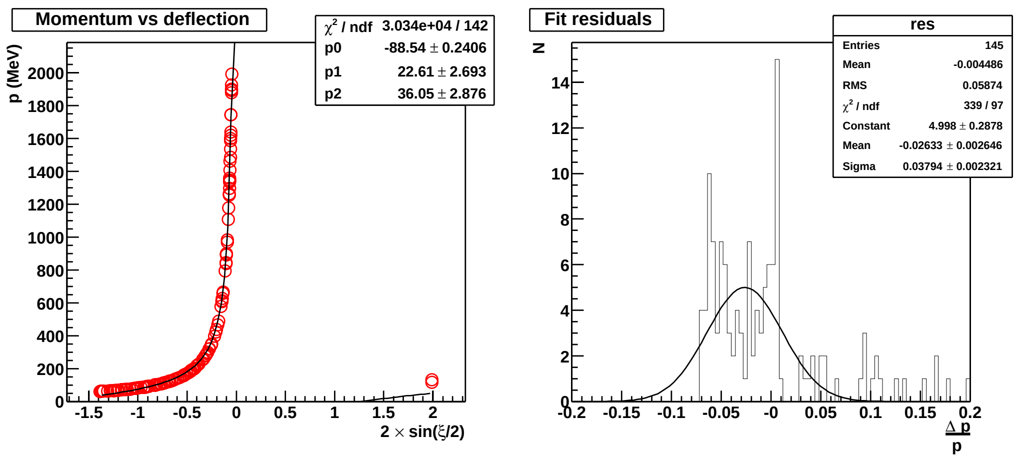


Figure 5.19: Sample fit at $\theta = 30^o$, $\phi = 10^o$. The left side shows momentum versus deflection in the bin. The red circles are the data points and the solid line is the fit. The rightmost figure shows the fit residuals fitted to a Gaussian. We can see how the outliers spoil the fit

would be banishing a large portion of valid low momentum data and this, in turn, would make the estimation of C unreliable.

Since, we know that the momentum where this effect starts to show up is related with the magnet's momentum kick, we could use it as a scale parameter for the momentum cut. The problem is that we need to know the magnet's momentum kick a priori. The momentum kick can be associated to the parameter A which is most sensible to the higher momenta. Therefore we can start with a strong cut in momentum in order to obtain a good estimation of A and then use that estimation as the momentum cut for each bin in a second step.

Actually it turns out that even a third step is needed where the remaining outliers are eliminated by using Tukey Weights to robustify the fitting functional. The procedure is not that different from the one used to robustify the vertex finder at section 4.2.3 on page 86. As explained there, the robustification is achieved by introducing weights in the LSM functional. These weights take the form

$$w(t) = \begin{cases} \left(1 - \frac{t}{C_T^2}\right)^2 & if|t| < C_T \\ 0 & otherwise \end{cases}$$

where $t_i = \frac{p_i - f(\xi_i)}{\sigma_i}$. Introducing such weights makes the functional non linear on the parameters and therefore we are not able to obtain an analytical solution anymore. What is done, instead, is an iterative minimization. In each step the weights w_i are calculated from the parameter estimation in the previous step and treated as constant. For this to work we need an initial value. The initial value needs to be quite close to the real one, otherwise the fit routine will tend to put all $w_i = 0$ which is an absolute minimum. But we already know how to obtain the initial value from the two step method presented before; the second step is in fact needed in order to provide a good enough initial value.

After applying these new stages in the parameterization procedure, we get a much better parameterization as shown in Fig. 5.20.

5.4 Final resolutions

In this section we will describe the application of the above methods to the different setups in HADES and we will show estimates of the achieved resolution in each setup based on simulations.

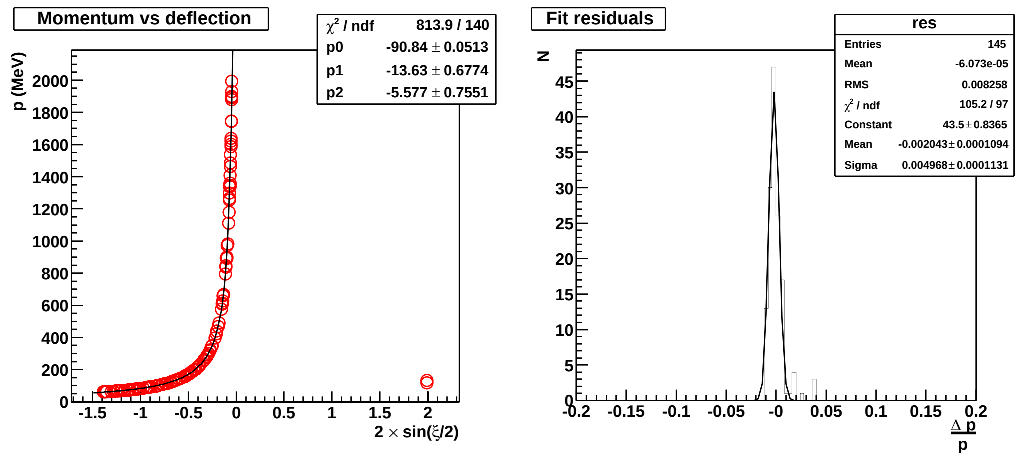


Figure 5.20: Same data as for figure 5.19; but using a robustified fitting technique.

5.4.1 Low resolution: inner chambers and META

In this section we will apply the kick plane algorithm described above to the particular setup where none of the two outer chambers is available. Also we will present a method to build track candidates using the available detectors, that is, the two inner MDC chambers before the magnet and the META detector after the field. This is more clearly visible in Fig. 5.21

Each of the MDC chambers is able to measure the track's position with high accuracy, as well as its direction. So, the track before the magnet is relatively well known. The situation in the outer segment, after the magnet, is completely different because of the poor position resolution of the META detectors and the inexistent direction measurement.

The TOF detector is intended to measure a particle's time of flight since the interaction point. For that reason it is made as an array of scintillators each having a set of two photo-multipliers, one on the left side and another one on the right side. Each photo-multipliers is connected to a TDC giving us two time measurements per hit. So the time of flight can be calculated as the average between the time TDC measurements; see 2.3.6 for a more detailed description on the workings of the TOF detector.

In order to obtain momentum we need to use TOF to get a point on the track. Fortunately, in order to cope with the higher multiplicities to be measured by HADES, the TOF detector is built with a relatively high granularity. That is, we have many relatively small scintillators. Each of these scintillators has a small depth and a height ranging from 2cm to 3cm.

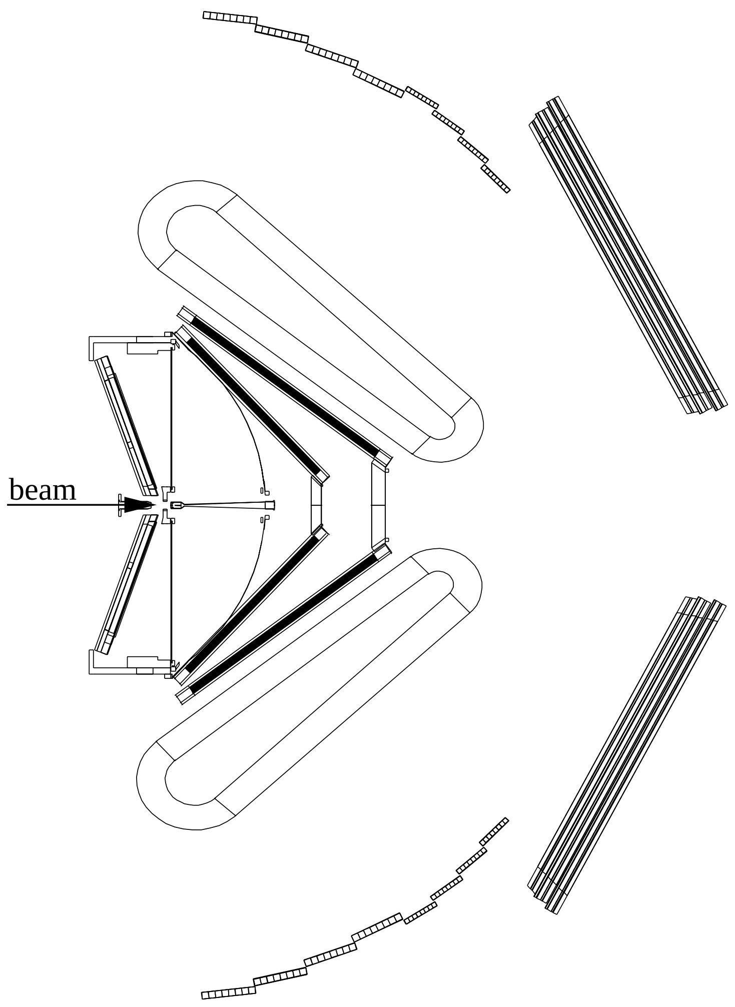


Figure 5.21: The low resolution setup for Hades is shown. Only the two Mdc chambers are available before the magnet. In the outer part of the spectrometer the Tof detector is available at the larger polar angles while the Shower and Tofino fill the lower ones.

Thus, the physical volume of the scintillators allow for an estimation of two track coordinates. The third one can be obtained from the time difference between the two TDC measurements. Knowing the group velocity (v_g) of light inside the scintillator $x = v_g \frac{t_{left} - t_{right}}{2}$. This method gives uncertainties between 1.5 and 2.3 cm for x.

The SHOWER detector is intended to discriminate between leptons and hadrons by examining the track's shower through the interaction with two lead converters. For that purpose three gaseous chambers register the shower before and after each converter. The chambers are the active volumes. Geometrically they are subdivided in pads. So for each registered track we know the pad which was hit. The pad's central position will be our position estimation for the track, and its depth, width and height give the corresponding uncertainties. Both width and height range from 3cm to 4.5cm. For more information on SHOWER consult section 2.3.7.

The main limiting factor in momentum resolution is the relatively poor position resolution of the META sub-detectors: TOF and SHOWER. It should be kept in mind that these detectors where not originally devised to provide an excellent resolution in position. We are only using them for this task temporarily, while the outer MDC chambers are made available.

In spite of this we will see that the achieved resolution is similar to that of predecessor experiment DLS.

The kick plane algorithm is chosen for this setup because it has a smaller memory footprint. It is faster and the worse resolution is completely masked out by the resolution in the measurements themselves. In fact, the kick plane algorithm was originally developed to deal with this particular setup. We will proceed by detailing how the different parts of the kick plane algorithm were implemented in this setup to finally show what are the results in terms of resolution.

Not in this section but still relevant is the treatment of clusters in the TOF detector. We have said before that a track's position can be measured in the TOF detector by knowing which scintillator the track was hitting. There is a non negligible probability that a track crosses more than one scintillator (around 5%). We therefore need an algorithm which is able to recognize those cases and creates a cluster out of them, presenting to the kick plane one position measurement instead of two. Such an algorithm exists and was implemented by Dusan Zovinec. The basic idea is to form cluster candidates from sets of two or more contiguous scintillators. To know if the cluster candidate is really a cluster the estimated direction from the kick plane is taken into account as well as the proximity of the x coordinate measurements of the involved scintillator and an analysis on the track's energy loss.

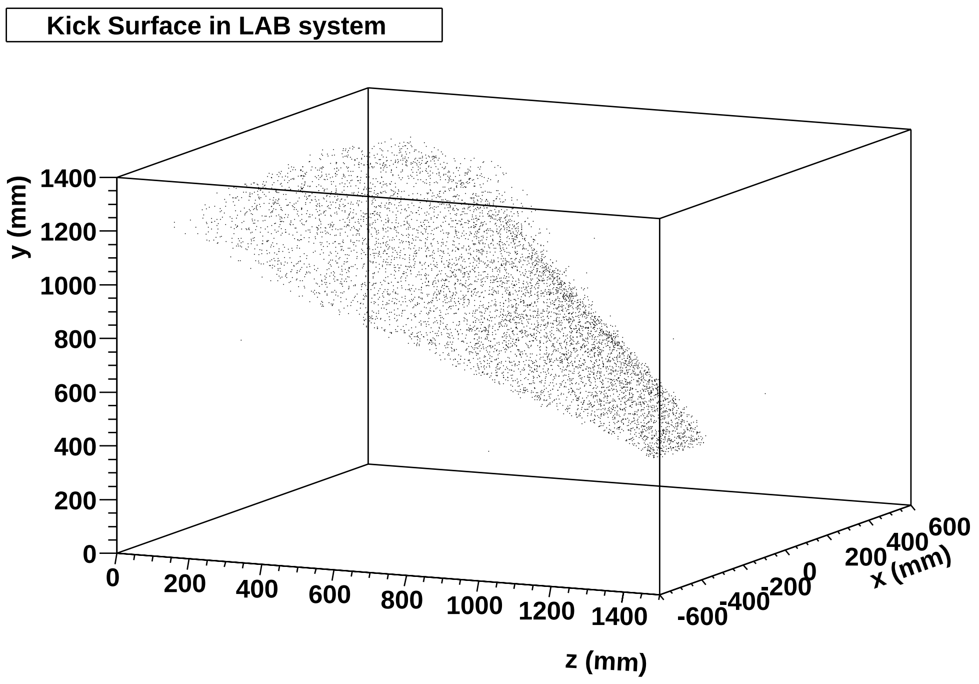


Figure 5.22: Kick surface

Kick surface

In the low resolution setup the kick surface plays a very significant role. Since the META detectors are not able to measure the track's direction; we need to estimate it using the kick surface. The outer segment is defined by the measured point in META and an additional point obtained as the intersection of the inner segment with the kick surface.

Fig. 5.22 shows the kick surface in the LAB system. As shown there, the kick surface still looks mostly like a plane, but with some curvature which can be approximated by a quadratic term in x, giving a parameterization function $y = a + bx^2 + cz$. This simple parameterization function reproduces the main features of the kick surface and it is good enough for momentum reconstruction, as shown later. A local discrepancy between the real kick surface and the model translates into a local systematic which can be reabsorbed in the momentum fit parameters.

The fit procedure introduced in the general explanation of the kick plane yields parameters, in mm

$$a = 1460 \pm 2,\ b = -(3.71 \pm 0.04) \cdot 10^{-4},\ c = -0.866 \pm 0.05$$

We also need to be able to compute the intersection of a straight line with this surface. That is done solving the system composed of the kick surface equation and the equations of a straight line

$$\begin{aligned} x &= x_0 + \beta_x(z - z_0) \\ y &= y_0 + \beta_y(z - z_0) \end{aligned}$$

The system can be reduced to the quadratic equation

$$b\beta_x(z - z_0)^2 + (2bx_0\beta_x - \beta_y + c)(z - z_0) + a + cz_0 - y_0 + bx_0^2 = 0$$

This, in general, yields two solutions, which is fine since in the general case a single track can cross the described surface at two points.

Parameterization quality

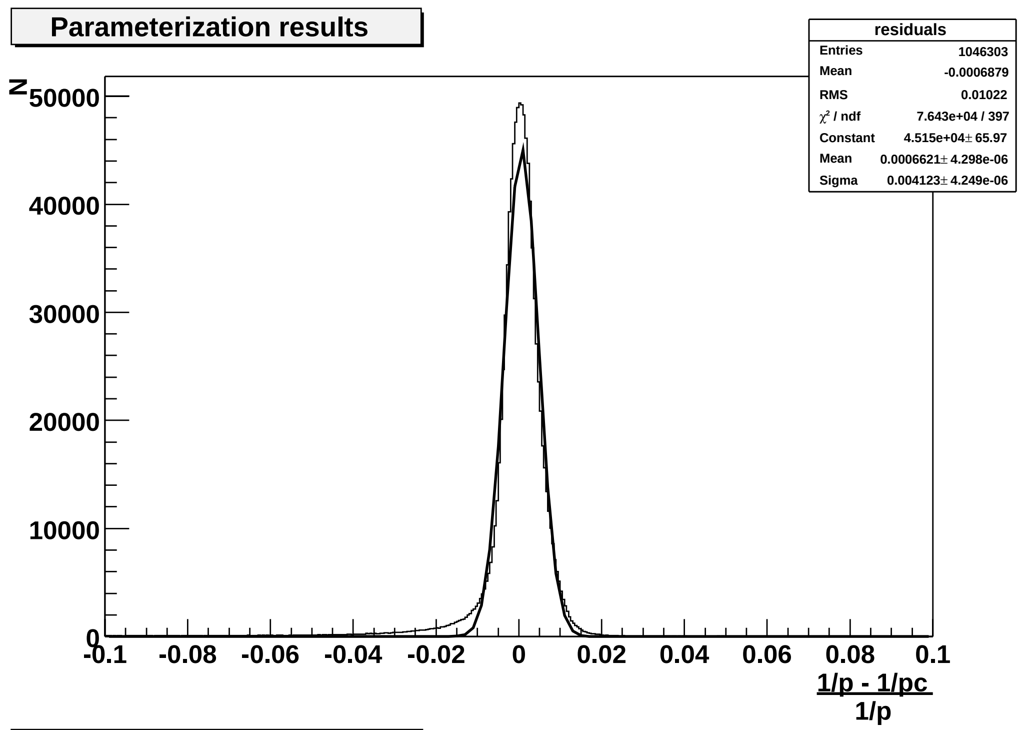
The parameterization of the relation between momentum and deflection angle is done as described in section 5.3.4. In order to get a feeling of the quality we have plotted the residuals distribution integrated over the whole geometrical acceptance and momentum range. The result is shown in Fig. 5.23 and gives and idea of what the intrinsical resolution of the method is.

The existence of a tail for low momenta shows the limits of the method. We will insist on this in the following section where the achieved momentum resolution is presented. It should be noted that such low momentum tail is enhanced in this plot because the input data had a uniform distribution on $1/p$ which does not happen in a real collision.

Achieved resolution

The momentum resolution achieved in this setup with the SHOWER detector is shown in Fig. 5.24 as a function of momentum; Fig. 5.25 is the same for the TOF. Each of the figures is subdivided in two which correspond to simulations where multiple scattering was either activated or not. This gives an idea of what part of the uncertainty comes from multiple scattering. The different lines correspond to different tracking setups:

- Santiago tracking is a state of the art tracking algorithm still under development.



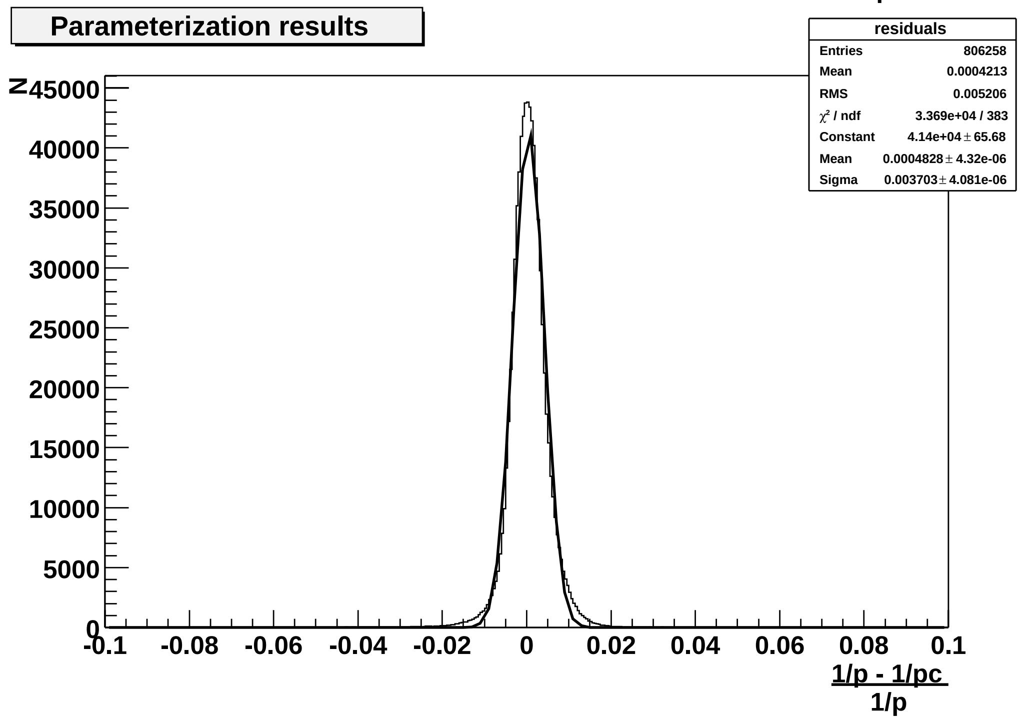


Figure 5.23: Parameterization residuals. The leftmost plot is the same as the rightmost with a cut on momentum larger than 100 MeV, showing that the tail is a low momentum effect.

- Ideal tracking uses information from the simulation to try to reproduce the ideal behavior of the MDC chambers. The measurement is produced by gaussianly smearing the track's position in each chamber, taken from the simulation software. The uncertainty introduced in the y coordinate was 0.08mm and 0.16mm in x. These are the design values for resolution in the MDC chambers

- The line for perfect detectors corresponds to extracting directly the measurements from simulation. That is there are no errors in the measurements besides the numerical ones. This is useful to get an idea on what the intrinsic resolution of the method is.

5.4.2 Medium resolution: inner chambers and MDC3

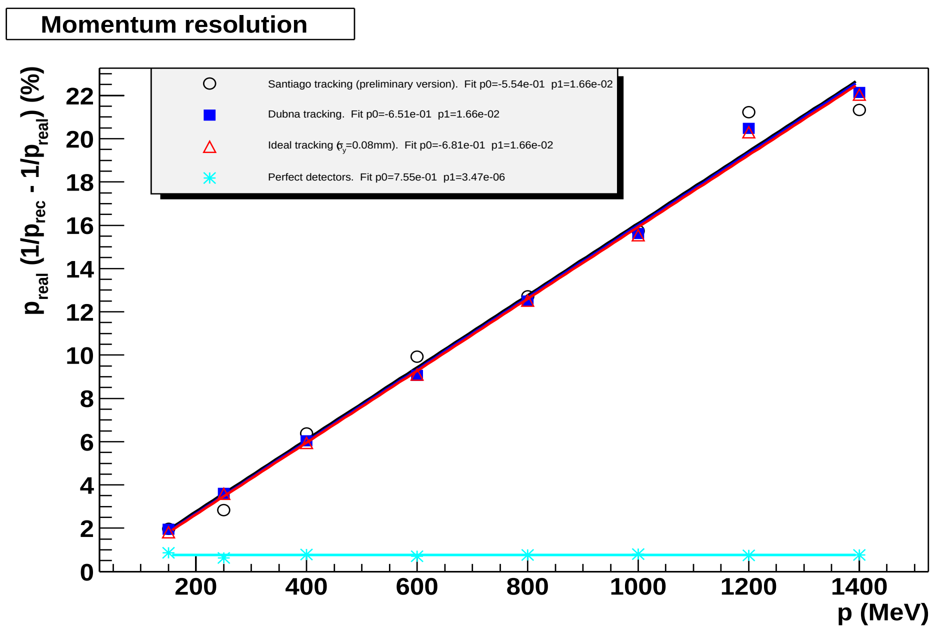
The setup we will be aiming at in this section is the one shown in Fig. 5.26. The main difference with the previous one is the presence of one outer MDC chamber in the sector. We will refer to this detector as MDC3. The fact that it is closer than META to the magnetic field, thus making the deflection angle more difficult to measure, is compensated by the much better position resolution which is in the order of 100 μm.

As explained in section 2.3.5 the MDC chambers not only give information about the track's position but also about its direction. Unfortunately the direction information given by one chamber alone is not very reliable due to its relatively small thickness. The effect of thickness can be easily estimated but there are other effects like not perfect callibration of the TDCs[10], uncertainties about the inner alignment[11] of the detector and the effect in the tracking of the small number of planes. All those effects have a stronger influence in the direction measurement than in the position measurement.

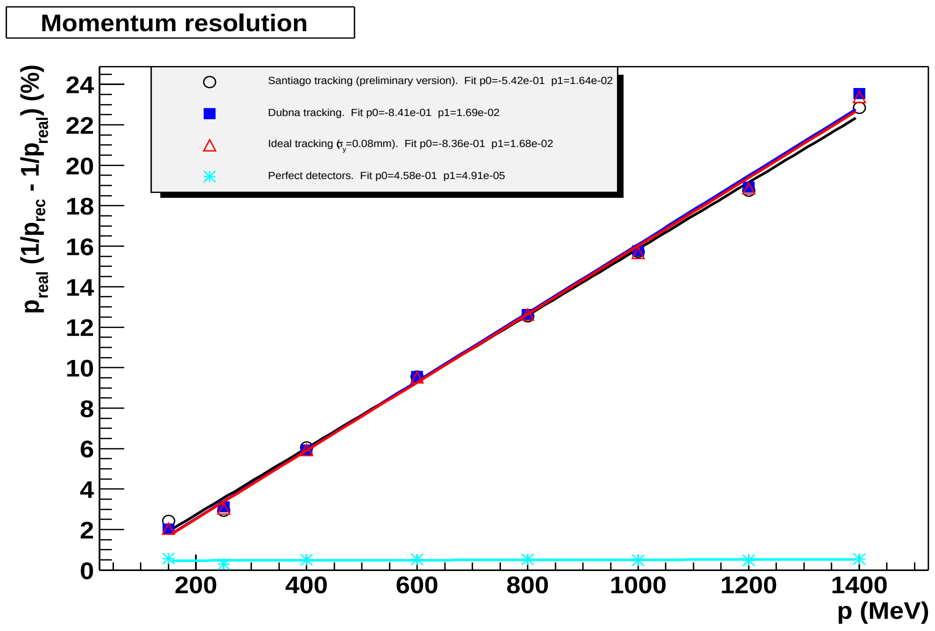
Again, the chosen method is the kick plane for its simplicity. The kick plane algorithm relies on the measurement of the track's deflection in the magnetic field. With this setup, that deflection can be measured in different ways. We have up to three primary ways to determine the track's direction in the spectrometer's outer region.

[10]A high quality callibration requires the on-line recording of the so called callibration events. These callibration events allow to measure the gains of the chamber's TDCs during the data taking. Those data, in turn, are very useful in obtaining callibration parameters. However, taking this kind of events together with the normal ones requires the activation of the so called Mixed Trigger in the data acquisition chain. This is a complicated process only recently performed.

[11]For example, layers are glued to the main frame. But the thickness of the glue is not very well known yet.

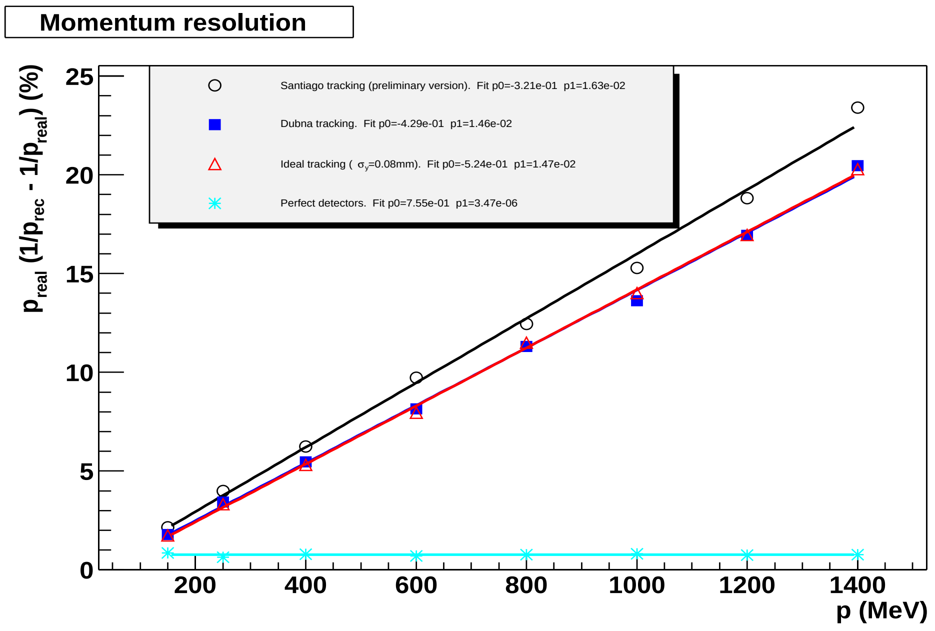


(a) Resolution with multiple scattering activated

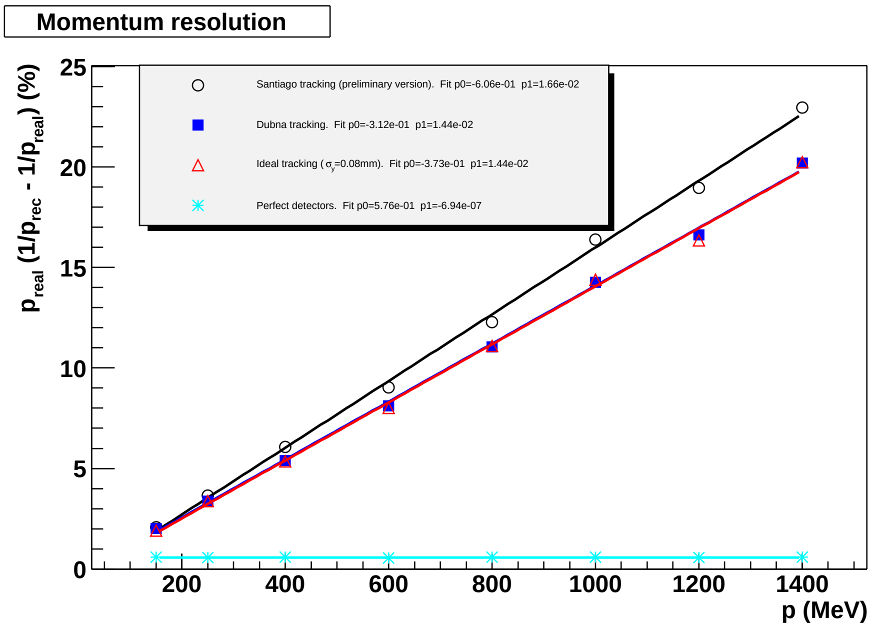


(b) Resolution with multiple scattering switched off

Figure 5.24: Momentum resolution in the low resolution setup using the SHOWER detector



(a) Resolution with multiple scattering switched on



(b) Resolution with multiple scattering switched off

Figure 5.25: Momentum resolution in the low resolution setup using the TOF detector

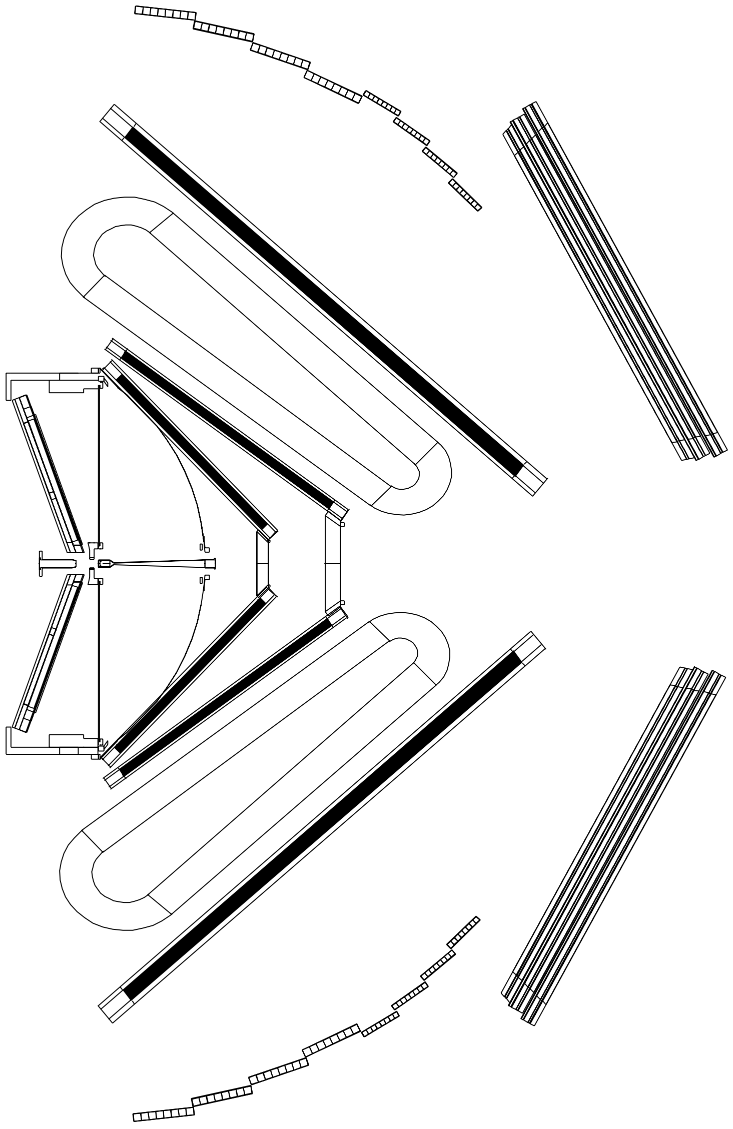


Figure 5.26: Medium resolution setup. Only one of the outer MDC chambers is available. The META detectors are also available.

1. Using the direction information from MDC3
2. Making a straight line going from the intersection of the inner track segment with the kick plane, to the measured point in MDC3
3. Making a straight line going from the MDC3 measurement to the corresponding measurement in META

The three methods have been tested and they are offered as options in the analysis, just as different use cases. However, by the time of this writing, the most reliable one is the second; this situation may change in the future when a better understanding of the MDC3 detector is achieved.

In the first use case there are problems derived from the uncertainties in the direction measurement given by one chamber. It can be shown that even on simulations, where callibration and inner alignment are under control, the residuals in the direction measurement feature long tails, which translate in large errors in the momentum determination.

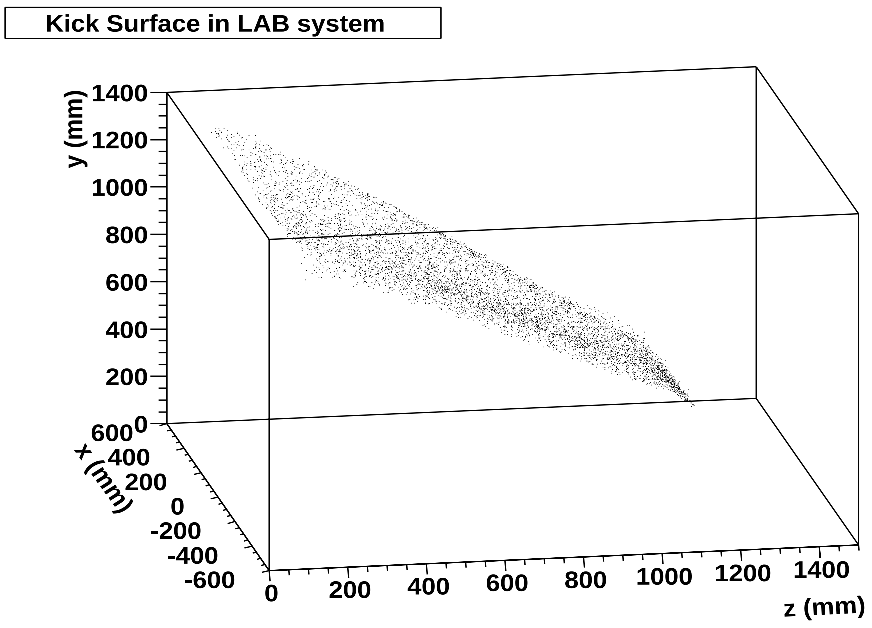
The third method doesn't show those tails but the poor position resolution of META makes up for a resolution in momentum not as good as the one achieved in the second scenario.

On the other hand the second scenario has the problem that the kick surface being very close to the measurement plane makes the goodness of the kick surface parameterization more significant than before. In other words, when the direction is determined from two spatial points, the sensibility to how well those data points are defined increases when decreasing distance between them.

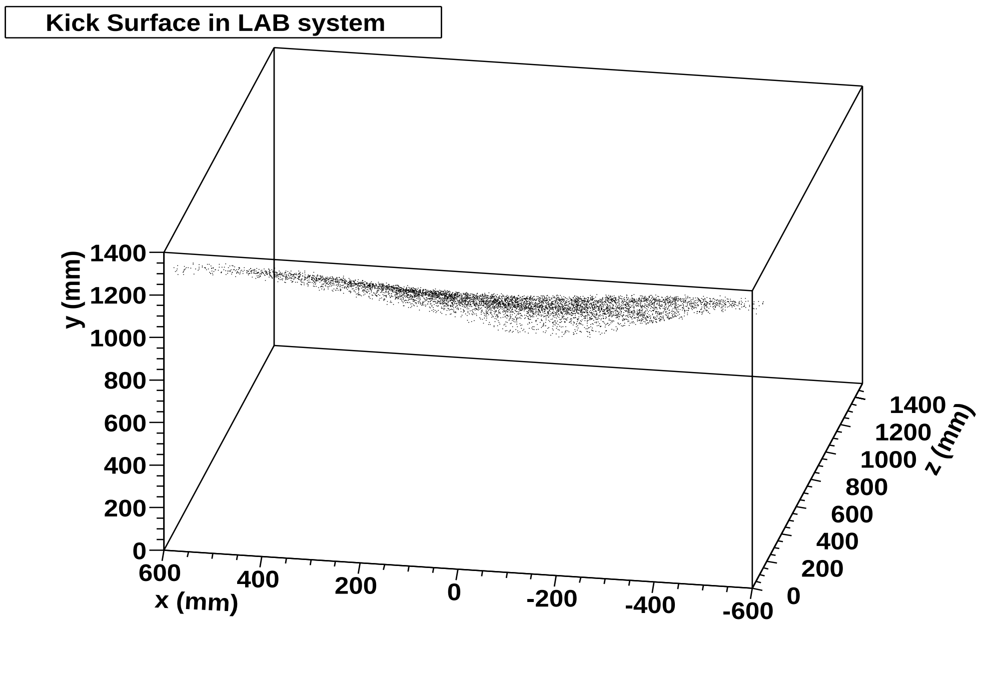
Besides the resolution we may get from the MDC system alone, we may always use the low resolution algorithm to determine momentum from the META detectors. This provides us with an additional, independent, momentum measurement which can be combined with the one obtained from the MDCS. The combination of both measurements is a very simple process, i.e. a weighted mean. For the sake of clarity we will concentrate in this section on the measurement of momentum without the META detector.

Kick surface

The shape of the kick surface in this case is shown in Fig. 5.27. It mainly looks like a plane of the form $y = az+b$. There are two modifications, though, there is a dependency in x to account for the bird wing like structure and also the value of a changes as we move on z. In order to account for the first



(a) The kick surface looks like a plane; but at large z the slope changes



(b) Detail on the azimuthal behavior.

Figure 5.27: 3D views of the kick surface in the medium resolution setup.

of the effects we added a term going like $|x|$. A complex parameterization of a as a function of z would not satisfy the requirement of simple calculation of the intersection point of a straight line with the kick surface; for this reason we just subdivided the kick surface in three. The final model used to parameterize this surface was

$$y = \begin{cases} a_1 z + a_2 + a_3|x| & z \leq z_1 \\ a_4 z + (a_1 - a_4)z_1 + a_2 + a_3|x| & z > z_1 \textit{ and } z \leq z_2 \\ a_5 z + (a_4 - a_5)z_2 + (a_1 - a_4)z_1 + a_2 & z > z_2 \end{cases}$$

where $a_1, \ldots, a_5$ and z_1, z_2 are the surface's parameters. While the a_i have been fitted using the MINUIT package following the prescriptions in section 5.3.3, z_i have been estimated by direct inspection, since MINUIT has problems treating them. The resulting parameters are

a_1	a_2 (mm)	a_3	a_4	a_5	z_1 (mm)	z_2 (mm)
-0.652	1287	1.05	-0.781	-1.47	400	1100

As for the achieved resolution, it can be estimated from the residuals plot shown in Fig. 5.28. The small tail comes from the region with larger z values, where the parameterization becomes more difficult. Looking to the distribution's RMS this is more than three times better than what we got for the low resolution setup. In relative terms the resolution is around 1.5%.

Regarding the intersection with a straight line, the algorithm is straightforward. It is basically the intersection of a plane with a straight line. The only thing one needs to be careful about is to take into account that a plane is infinite; but in our case we have sub-planes. In other words, for each of the three planes defining the kick surface the intersection point is calculated; then we have to check if that point really lies withing the plane boundaries.

Parameterization quality

Like in section 5.4.1 the residuals on the momentum parameterization are shown here. The comments are the same as in that section. The only conclusion is that the fact that MDC3 is inside the magnetic field region does not affect the validity of the model. The width of the peak is still around 0.4%

Achieved resolution

Like for the previous setup. The momentum resolution achieved in this setup is shown in Fig. 5.30 as a function of momentum. The two figures correspond

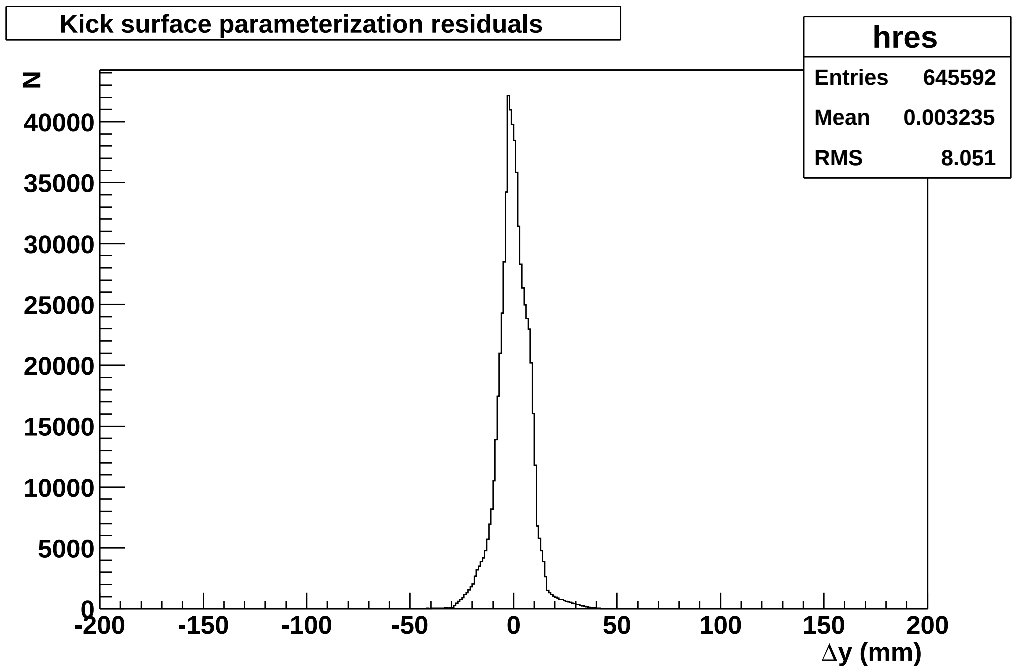


Figure 5.28: Surface parameterization residuals

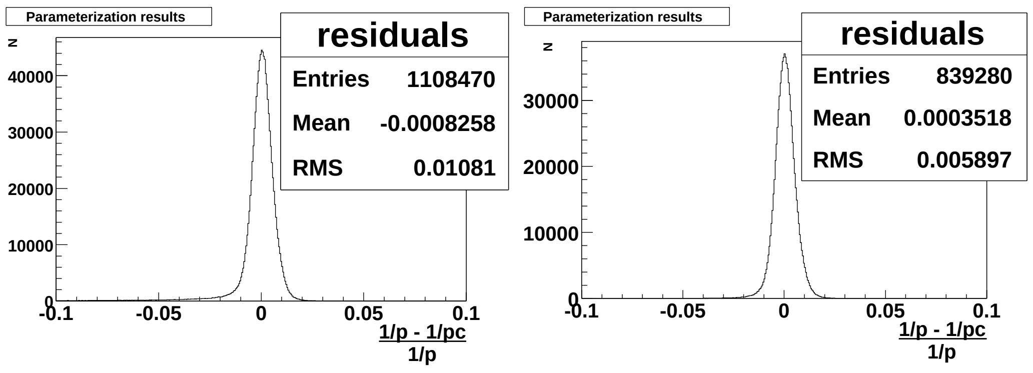


Figure 5.29: Momentum parameterization residuals. The rightmost figure is the same as the leftmost, just with a cut $p > 100 MeV$

to simulations where multiple scattering was either activated or not. This gives an idea of what part of the uncertainty comes from multiple scattering. The different lines correspond to different tracking setups, as explained in section 5.4.2.

5.4.3 Full resolution: full setup

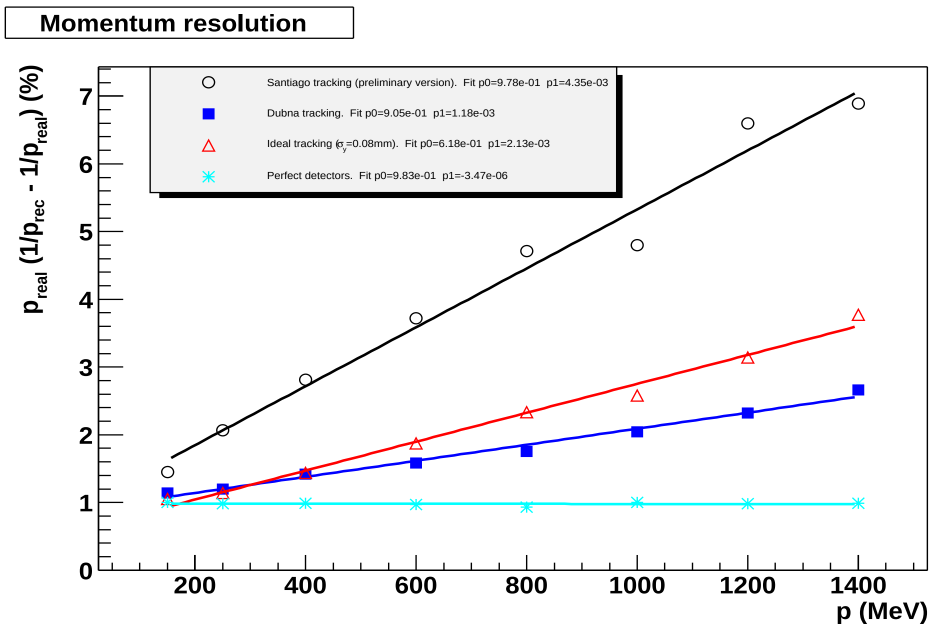
The setup we will be considering in this section is the full setup shown in Fig. 5.31. All four MDC chambers are available in this setup, so the track's position can be measured in four planes. Two in the inner part of the spectrometer and two in the outer part. These four measurements allow to accurately define the track's segments before and after the magnet. This means it is not any more necessary to use the kick surface in order to determine the track's direction after the magnet, the direct measurement can be used instead.

Up to now we had 6 measurements in order to estimate the 5 track's parameters. One of the measurements showing little sensibility to momentum; so we were very much in a situation where the number of measurements and degrees of freedom was the same. However, now we have 8 good measurements of the track and one of the two extra ones is very sensible to momentum; so it makes sense not to do a direct calculation of the momentum but to perform a fit.

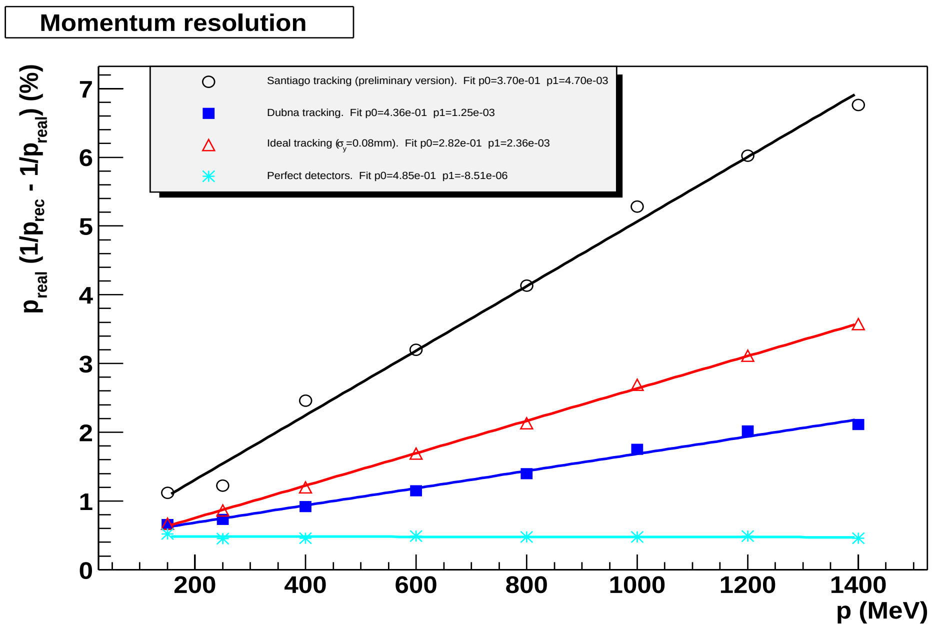
For this reason in this setup the chosen approach is to use the kick plane algorithm to obtain an initial value of the momentum. Then this initial value is used by the Reference Trajectories fit to obtain the final momentum. The initial value being close to the final one has the advantage of making the convergence faster.

Since the kick plane is only used here to give an initial value it was not extended to incorporate a fitting routine. However it is noted that potentially a fitting routine based upon the kick plane model could result in a resolution comparable to that of the Reference Trajectories algorithm, while still retaining the lower requirements of the kick plane in terms of memory compsumtion and parameterization effort. This topic is left for future investigation.

Since the kick surface is not critical we will omit here the section about its parameterization. Also the resolution in the momentum parameterization can be estimated from the plots in the results section. There we will show the



(a) Resolutions with multiple scattering



(b) Resolution with multiple scattering switched off

Figure 5.30: Momentum resolution for the medium resolution setup.

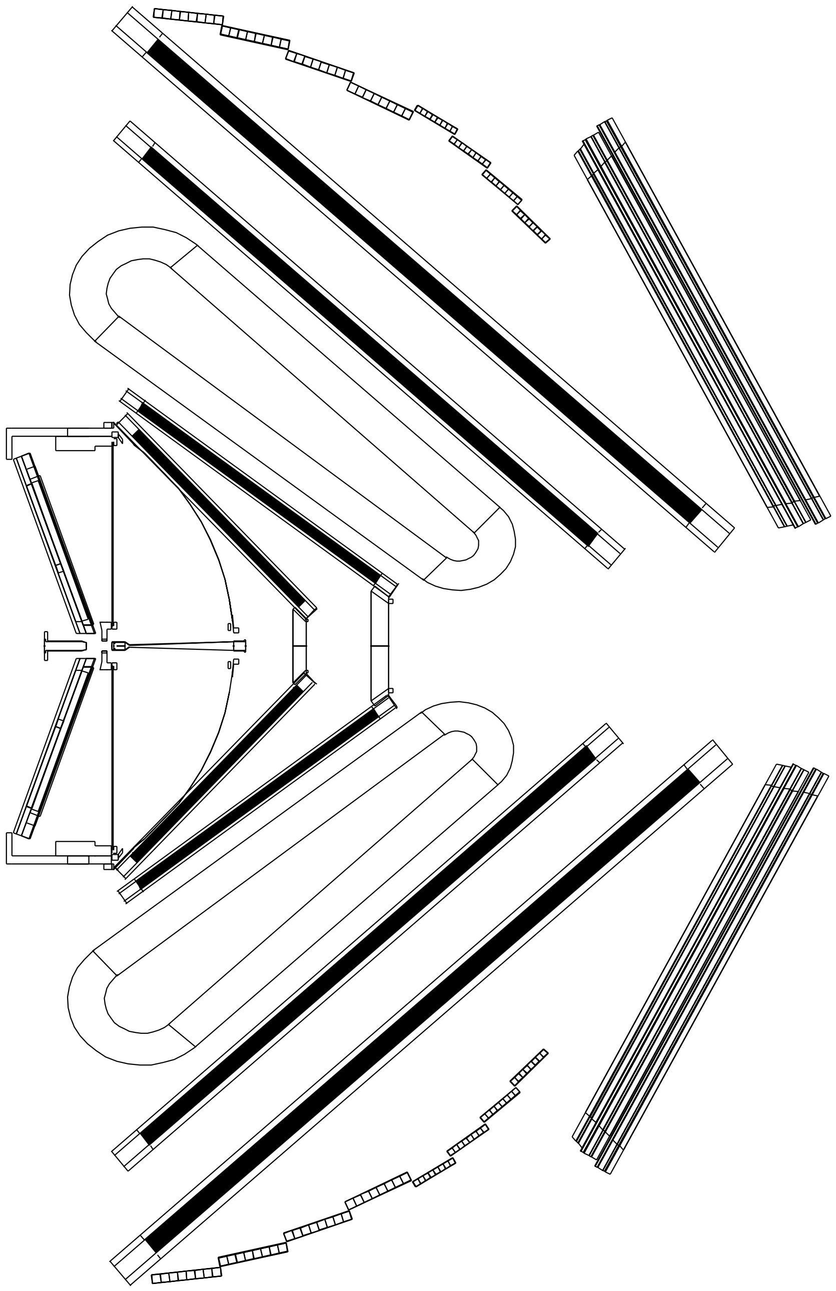


Figure 5.31: Full resolution setup

momentum reconstruction resolution using the kick plane algorithm assuming perfect detectors; this reflects the intrinsic resolution of the kick plane algorithm which is essentially the parameterization resolution. The models used for kick surface and momentum parameterization are the same as the ones in the section describing the medium resolution setup.

Results

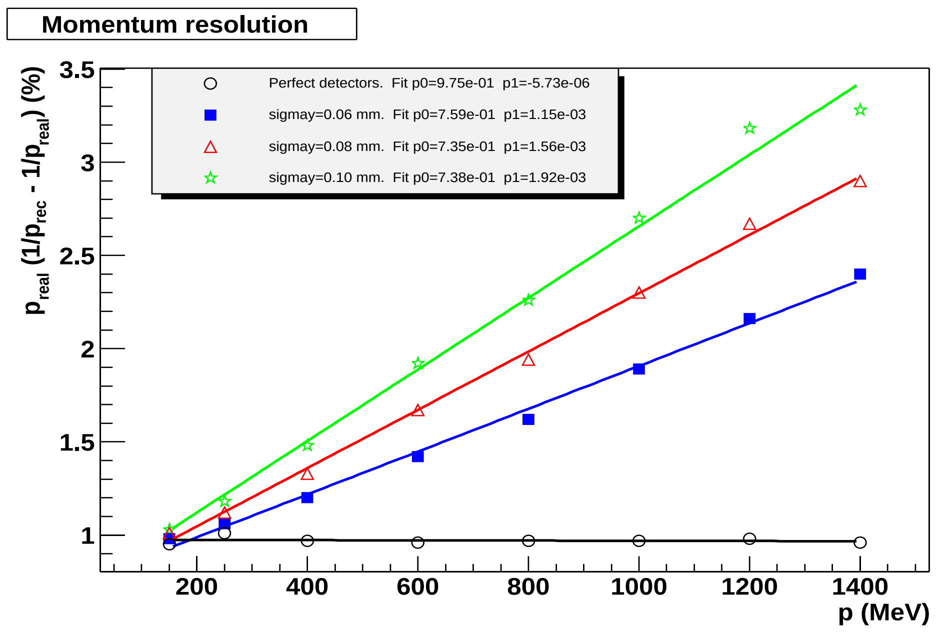
As in the previous sections the results are here summarized in several plots of resolution versus momentum. In this case, however the plot for the Santiago tracking is not presented as this tracking was not completely tested in the full resolution setup by the time of this writing. Instead the results of the ideal tracking are presented assuming different resolutions in the chambers. When the resolution is set to zero we get the data points for perfect detectors showing the intrinsic resolution of the implemented algorithms.

Fig. 5.32 shows the achieved resolutions with the kick plane algorithm both with multiple scattering and without. The final result given by the Reference Trajectories algorithm is shown in Fig. 5.33 again for multiple scattering switched on and off.

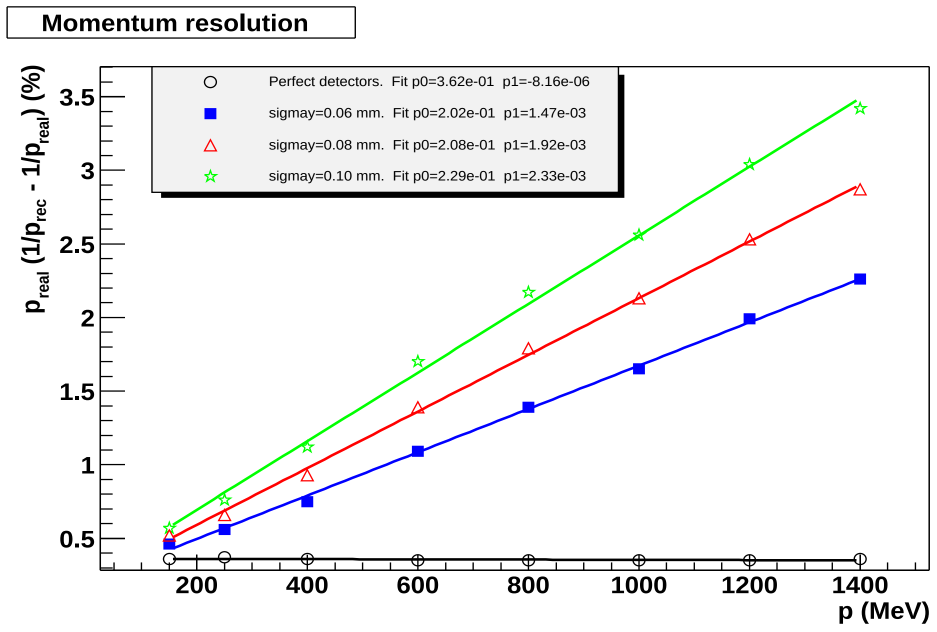
Comparing these two figures we see how the kick plane is affected by not including a fitting algorithm. Since no actual fit is done the kick plane has to accommodate all errors as uncertainty in the momentum. In other words it is like if 4 out of the 5 track parameters were frozen in a fit to values which are not necessarily the correct ones. On the other hand, the reference trajectories algorithm is able to also fit the remaining 4 parameters, thus giving a better resolution in momentum. This works out because with the full resolution setup there is quite some redundancy in our measurements, since, as we have shown, both MDC3 and MDC4 could give independent momentum measurements. The reference trajectories algorithm is able to take full profit of that redundancy, while the kick plane is not. Note how the situation is different in the medium and low resolution setups.

5.5 Track matching

In the previous sections we have concerned ourselves with methods to determine a track's momentum. Our starting point was a set of measurements from one track and the goal was to associate a momentum with those measurements. In real life the different measurements are given by independent detectors and therefore we don't know a priori which combination of the different measurements comes from the same track.

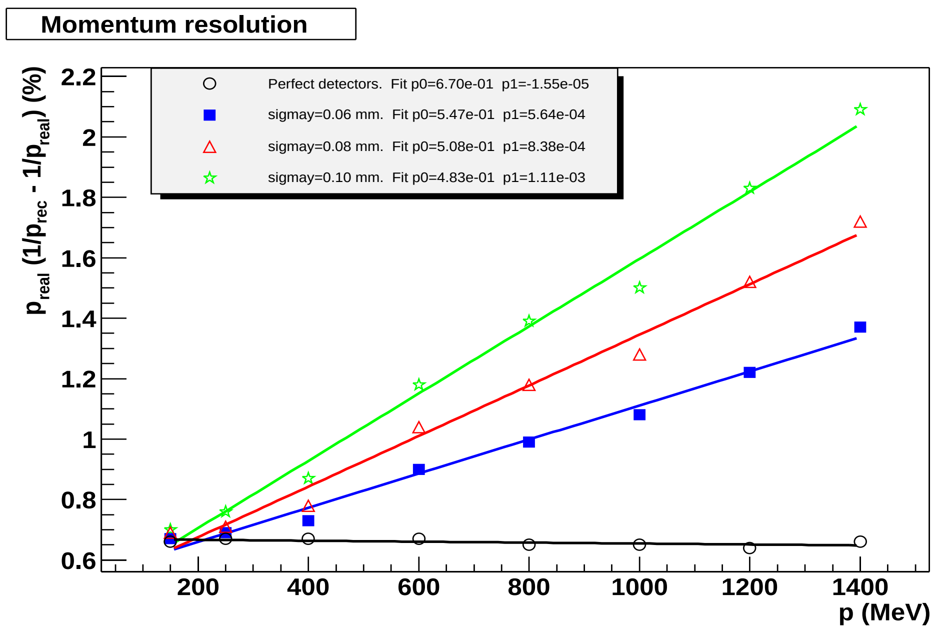


(a) Resolution with multiple scattering switched on

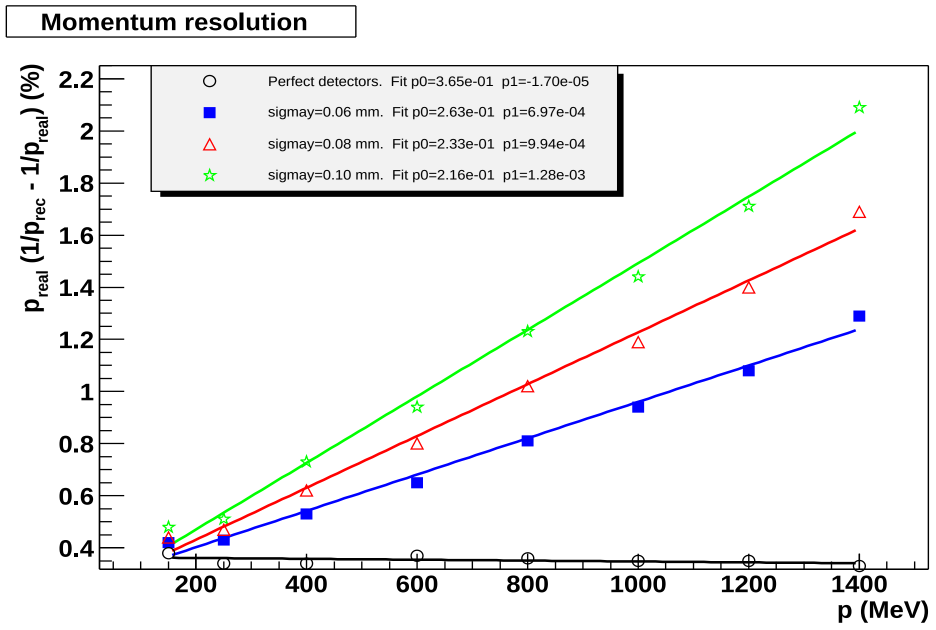


(b) Resolution with multiple scattering switched off

Figure 5.32: Momentum resolution with the full setup using the kick plane algorithm



(a) Resolution with multiple scattering switched on



(b) Resolution with multiple scattering switched off

Figure 5.33: Momentum resolution with the full setup using the Reference Trajectories algorithm

Let's suppose two tracks cross one of the sectors, each track has an inner and outer segment defined by the tangents to the track's trajectory before and after the magnet. This gives us four segments, which can be combined in four different ways. Out of those four combinations only two have physical interest as they are produced by a real particle. We say that only two are *good* combinations, the other two are *bad* ones.

The task we will be developing through this section concerns how to identify the good combinations out of the set of all possible ones. That is achieved imposing some criteria on the set of combinations. The process of imposing those criteria is called detector matching.

In general we will refer to individual measurements by one detector as hits, hits are grouped in combinations. When a combination satisfies the matching criteria it is stored as a candidate. There are different kind of candidates depending on which detectors are being matched. It is also possible that one candidate formed in the matching of two detectors enters as a piece in the matching with a third detector.

The final candidate where all detectors are matched is called track candidate. However we will also miss-use that name for a candidate formed by matching the inner and outer MDC detectors.

Besides the hits belonging to one track there are also hits which do not belong to any physical track. They are called noise hits and are typically produced by electronic noise signals. Those noise hits are not correlated on the different detectors, therefore a matching algorithm is useful to detect and reject them.

There are two main factors defining how good a matching algorithm is:

Efficiency Is the probability that a set of measurements belonging to the same track are identified as a track candidate. The higher the better

Noise Is related to the probability that a set of measurements *not* belonging to the same track are identified as a track candidate. The lower the better.

In the following sections we will describe matching algorithms between different detector sets, intended to give support for the different setups in which we have to do momentum reconstruction. Section 5.5.1 describes the matching of the two inner MDCs with the META detector; the main use of this algorithm is when performing momentum reconstruction with the low resolution setup. Section 5.5.2 describes the matching of the inner MDCs with the outer ones, either MDC3 alone or both of them; this is useful both in the medium and high resolution setups. Section 5.5.3 describes how to match one of the previous candidates with the META detector; this algorithm depends on the one described in section 5.5.1 and it is useful in the medium resolution setup, because it allows to use META to perform a second determination of momentum which can be combined with the first one.

In sectors with the low resolution setup, a candidate coming out of the matching of MDC and META is a full track candidate, while in sectors with outer drift chambers a full track candidate comprises also the matching with those chambers.

5.5.1 Matching between MDC and META

In this case a track candidate is formed by a segment in the inner part of the spectrometer and a hit in any of the META detectors. Our task is, given those two pieces, to determine if they correspond to the same track or not. In case they do, we build a track candidate and compute its momentum for further analysis. We will refer with the name *hit combination* or simply *combination* to each possible combination we can make with MDC and META hits. The task at hand is to define an algorithm endowing us with the ability to discern the bad, not belonging to the same track; and good, belonging to the same track, combinations. So that track candidates are only formed with good combinations.

Unless otherwise noted, in this section we will be working with simulation data, where HGEANT is able to confirm if a combination is good or not.

In order for the matching to be possible we need some degree of redundancy in the measurements, so that we can check a combination for consistency. If there is no redundancy, any combination looks as good as any other one, just delivering different parameter estimates.

In our case a track is defined by 5 parameters, as explained in section 5.2.1. That amounts for 5 degrees of freedom. On the other hand the number of measurements is 6: 4 numbers describing the segment before the magnet as a straight line and 2 additional ones describing the track intersection with the META detector.

There must be 1 (6-5) constraint between the measurements. In other words, one of the measurements can be computed as a function of the other five. The basic idea of the matching algorithm here is to obtain that function and then use it for each combination, taking 5 of the parameters to calculate a sixth and comparing the calculated value with the measured one, normalized by the errors.

Out of the 6 measurements, 4 describe the track segment before the magnet and are directly mapped to 4 of the track parameters; namely ρ, z, θ and ϕ. Out of the other two measurements the most relevant one is the y coordinate on the META, since it has the greater influence in momentum. Actually, if the field was perfectly toroidal we would expect deflection to occur only in the polar angle and not the azimuthal one.

With these considerations in mind it was decided to choose the measurement of the track's x coordinate on META as the redundant variable used in matching. For each combination we use the other five measurements to compute the track's parameters $(p, \rho, z, \theta, \phi)$. From the track's parameters we estimate the track's x coordinate of intersection with the META detector $x_c = x_c(p, \rho, z, \theta, \phi)$. If we note the measured one by x_m we can define the pull variable

$$xPull = \frac{x_c - x_m}{\sigma_{x_c - x_m}}$$

For a good combination x_m and x_c should be the same besides measurement uncertainties. Assuming the uncertainty in the measurement to be gaussian with a width σ_{x_m} and the uncertainty in the calculated x_c to have width σ_{x_c}, $xPull$ would follow a normal distribution[12] with $\sigma_{x_c - x_m} = \sqrt{\sigma^2(x_c) + \sigma^2(x_m)}$. In fact this does not happen because the uncertainty x_m is not always gaussian, but we will just keep the conceptual model for now in order to make the explanation simpler. On the other hand, we expect $xPull$ to be uniformly distributed for bad combinations. The difference is clearly illustrated in Fig. 5.34.

In order to discern between good and bad combinations we can just place a cut on $xPull$; we call *accepted combinations* to those verifying

$$|xPull| \leq c \text{ with } c \in \Re$$

It should be noted how in Fig. 5.34 the red and blue points do overlap; that is, there is no clear separation between good and bad combinations. The consequence is that, for any cut we may put on $xPull$, some good combinations will be left out and some bad combinations will pass in. Therefore the characterization of a cut's quality with two variables: efficiency and noise level.

Let's call N to the total number of combinations, N_g the number of good ones and N_b the number of bad ones, so $N = N_g + N_b$. For a given cut c, let's call N_g^c the number of good combinations passing the cut and N_b^c

[12]Gaussian distribution with a mean of 0 and width equal to 1.

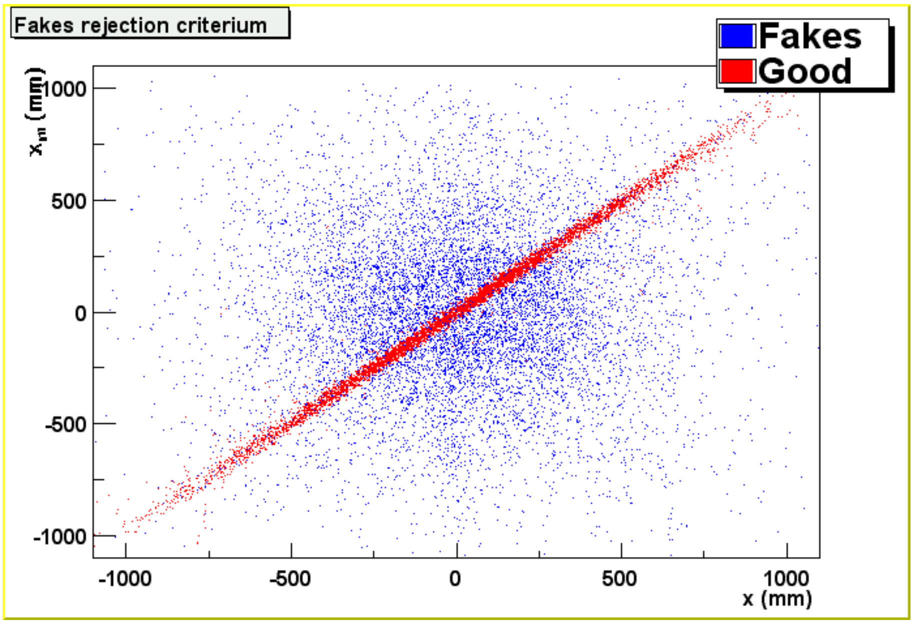


Figure 5.34: x_c is plotted versus x_m for bad combinations in blue and for good ones in red. Note how the two regions overlap.

the number of bad combinations passing the cut, $N^c = N_g^c + N_b^c$ is the total number of accepted, as in passing the cut, combinations. We call efficiency to the probability for a good combination to be accepted. With the definitions above this is expressed as:

$$\epsilon = \frac{N_g^c}{N_g}$$

The other relevant variable is the noise level, defined as the proportion of bad combinations relative to the total amount of accepted ones. That can be expressed as:

$$noise = \frac{N_b^c}{N_g^c + N_g^c}$$

Needless to say that the goal of any matching algorithm is to maximize efficiency and minimize noise. We will spend the rest of this section doing so. We will start by introducing the x_c parameterization, then we will have a closer look to the actual distributions of $xPull$ for good and bad combinations; the last subsection will present plots of efficiency and noise versus the cut value in a C+C simulation.

Computation of x_c: azimuthal deflection

Following the idea of a gaussian $xPull$, in order to attain a 99% efficiency we would set $c = 3$ in the defined cut. In order to estimate the noise level, let's assume for a moment that noise is uniformly distributed with respect to $xPull$. Then the number of noise tracks being accepted would amount to

$$N_b^c \propto \sigma_{x_c - x_m} \times c$$

That is, $\sigma_{x_c-x_m}$ acts as a scale parameter and thus we need to reduce it as much as possible. Now, $\sigma_{x_c-x_m} = \sqrt{\sigma_{x_c}^2 + \sigma_{x_m}^2}$. σ_{x_m} is determined by the detector construction and we have no control over it in the software, so we need to make $\sigma(x_c)$ as small as possible. The goal is actually to make $\sigma(x_c) \ll \sigma(x_m)$ by making a precise enough model for x_c.

If the magnetic field geometry was perfectly toroidal, that is $\mathbf{B} = B\hat{\varphi}$[13], the track would only be deflected in the polar direction. This would make the calculation of x_c pretty simple: we would just account for the change in θ according to momentum and force ϕ to be the same before and after the magnet. The rest is extrapolating a straight line until the META detector. In other words, we would ask for the direction vectors of the segments before and after the magnet to have the same ϕ. Actually this is the algorithm used by the MATCHING UNIT to trigger on events with good track candidates. In that case, more sophisticated algorithms are not allowed because execution speed is of maximum importance.

In the offline analysis time constraints are not so strict and we can spend some more time to realize that the field map is not perfectly toroidal, which implies that deflection also occurs on ϕ.

In order to be able to calculate x_c we need to parameterize this azimuthal deflection. Once more we will resort to HGEANT in order to do it. It turns out that the deflection in ϕ, $\Delta\phi$, is not a well behaved variable for parameterization. What is used instead is the deflection of the track's projection onto the XZ plane (the cave's floor) given by the change in the η angle, as illustrated in Fig. 5.35.

Let's call the η angle before the magnet η_1. This is obtained from the track's parameters ρ, z, θ, ϕ. Similarly let's call η_2 the angle after the magnet. Then the deflection in η is $\Delta\eta = \eta_2 - \eta_1$. Following a line of reasoning similar to that in section 5.3.2 we can assume that the deflection in η is related to momentum by the law

[13] $\hat{\varphi}$ is a unitary vector in the direction of increasing azimuthal angle.

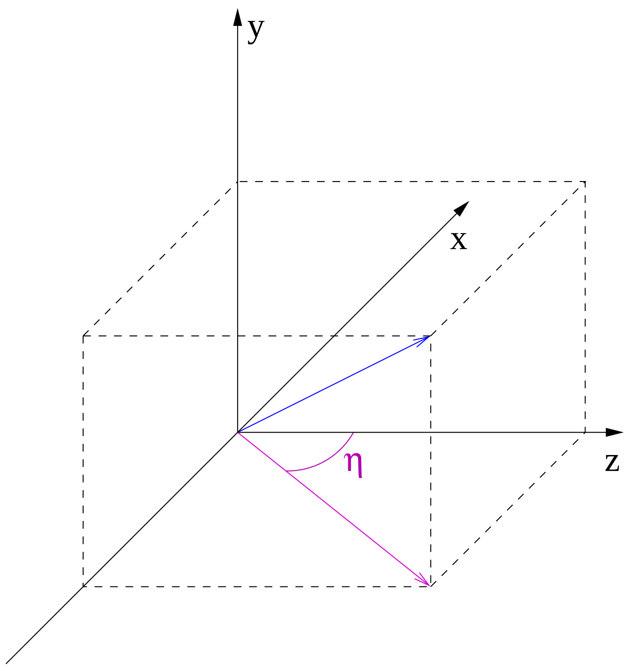


Figure 5.35: η coordinate definition

$$p = \frac{A}{\sin(\Delta\eta/2)} + B\sin(\Delta\eta/2) + C \tag{5.14}$$

where the parameters A, B, C depend on the position on the kick surface. This equation has two possible solutions for $\sin(\Delta\eta/2)$:

$$S^{\pm} = \frac{p - C \pm \sqrt{(C-p)^2 - 4AB}}{2B}$$

For $p \to \infty$ both solutions show different behaviors: $S^+ \to \infty$ while $S^- \to 0$. From physical arguments we expect the deflection to be smaller with larger momenta. So the physical solution is $S^- \equiv S$.

It turns out that obtaining A, B and C is not as simple as applying the general method introduced in section 5.3.4. The reason is the left/right symmetry of the sector. It has already been noted how the right and left sides of a sector are identical, but this has the consequence that in the middle of the sector there cannot be azimuthal deflection, $\Delta\eta = 0\ \forall p$. The parameters are completely undefined when $\Delta\eta = 0$ and ill defined as we get closer to the central region.

This problem can be avoided by making a slight change in the method. Instead of obtaining A, B, C trying to fit p as a function of $\sin(\Delta\eta/2)$ we make the estimation by fitting $p\sin(\Delta\eta/2)$. That is, our model becomes

$$p\sin(\Delta\eta/2) = A + B\sin^2(\Delta\eta/2) + C\sin(\Delta\eta/2)$$

When $\Delta\eta \to 0$ this equation becomes $A = 0$ and the parameters B, C are still not well defined. However if we substitute $A = 0$ in the expression of S^- we get

$$S^- = \frac{p - C - \sqrt{(p-C)^2}}{2B} = 0$$

which is the correct value. The minimization method is still similar to the one described in section 5.3.4, just changing the functional to

$$Q^2 = \sum_{i=0}^{N} \left[p_i \sin(\Delta\eta_i/2) - (A + B\sin^2(\Delta\eta_i/2) + C\sin(\Delta\eta_i/2))\right]^2$$

The same tricks are applied here to reduce memory compsumtion by performing the fits incrementally and to deal with outliers using an initial cut in momentum and Tukey weights in a second stage.

With all this information at hand we are finally able to estimate x_c for any given track candidate. The procedure is as follows, see also Fig. 5.36.

1. Use the ρ, z, θ, ϕ defining the inner segment of the track candidate to calculate its intersection point with the kick surface, let's call it $\mathbf{r}_{kick} = (x_{kick}, y_{kick}, z_{kick})$.

2. Use θ, ϕ to compute η before the magnet. $\eta_1 = \arctan\left[\tan(\theta)\cos(\phi)\right]$

3. Use the track candidate's momentum, p, and the values of A, B, C associated to the kick surface point $\mathbf{r}_k$ to calculate $\Delta\eta = 2\arcsin(S^-)$ and $\eta_2 = \eta_1 + \Delta\eta$

4. If z_m is the measured z coordinate in the META detector we compute x_c by extrapolation as

$$x_c = x_{kick} + \tan(\eta_2)(z_m - z_{kick})$$

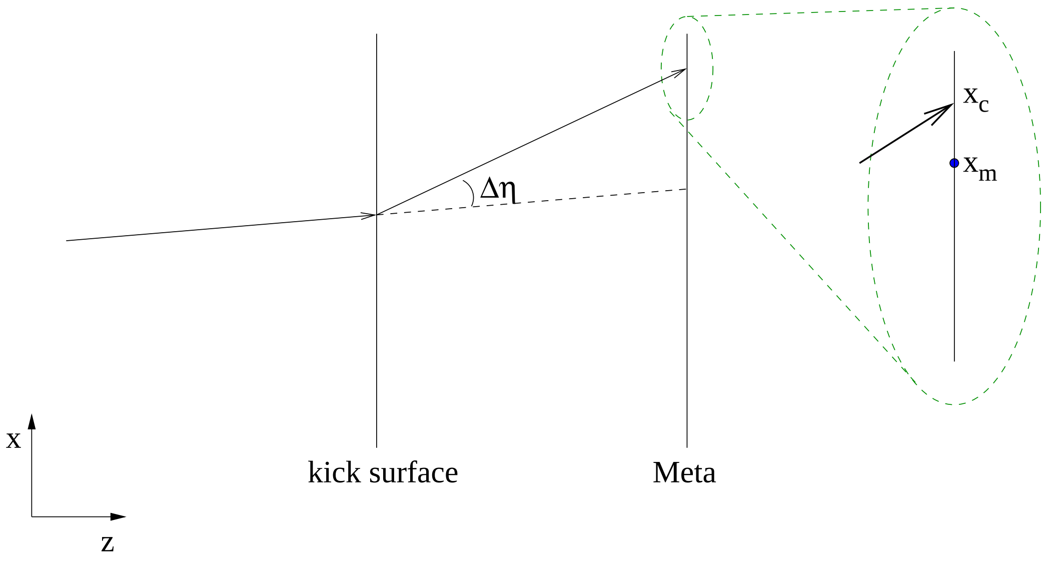


Figure 5.36: Calculation of x_c, shown is a track candidate projection onto the XZ plane; that is, seen from above.

The achieved resolution is directly related to the resolution in the computation of $\Delta\eta$.

$$\sigma(x_c) \simeq \sigma(\tan(\Delta\eta))(z_m - z_{kick})$$

Now, $\sigma(\tan(\Delta\eta))$ has many sources, the most important being the uncertainty in the estimation of the momentum and the quality of the model. In order to estimate this last effect we can compare the values of $\Delta\eta$ given by the model previously introduced and the exact values taken from HGeant. This is shown in Fig. 5.37. From that figure we learn that the uncertainty introduced by the model itself is around 0.002. To keep a number in mind we can say that $z_m - z_{kick}$ is roughly around 1 meter; thus the uncertainty in x_c introduced by the model is about 2mm which have to be compared with the $\sim 15mm$ of the Meta detectors.

$xPull$ distributions for bad and good combinations

We have discussed that $xPull$ should peak around 0 for good combinations and be approximately flat for bad ones. By actually plotting $xPull$, see Fig. 5.38, we learn that good combinations indeed peak at 0, but also bad ones do, to some extent. This is something we cannot get rid of, as it comes from Physics.

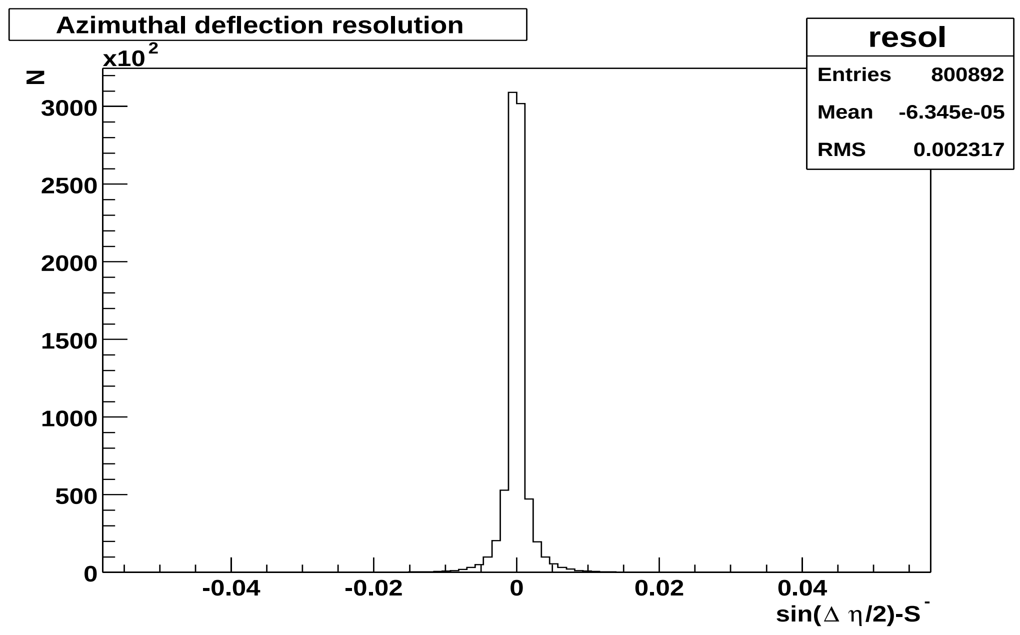


Figure 5.37: Resolution in $\Delta\eta$ parameterization for positrons

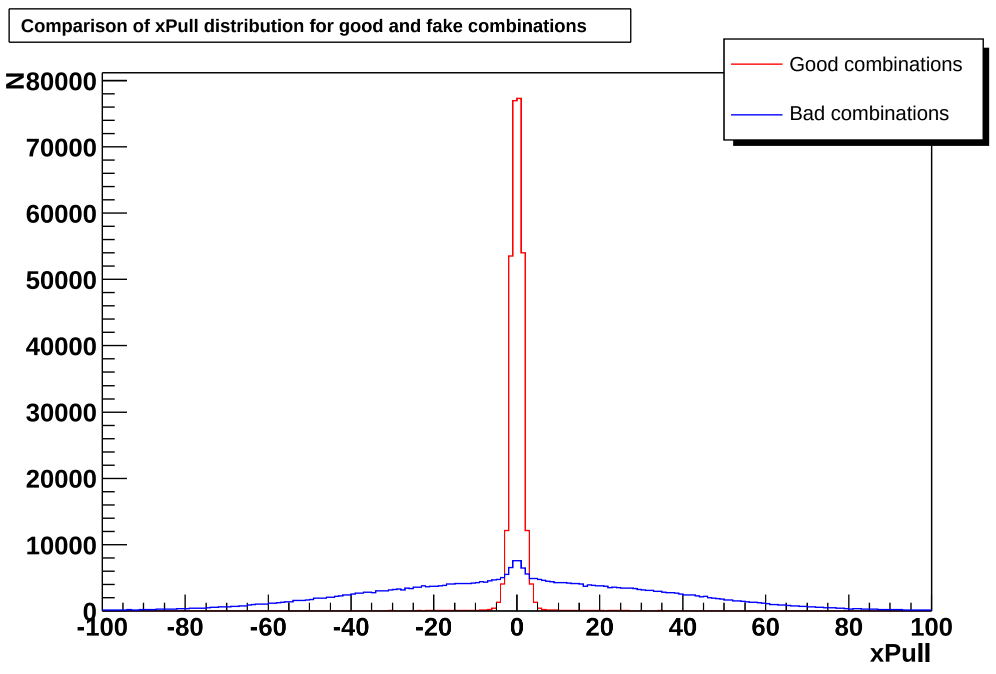


Figure 5.38: $xPull$ for good, left side, and bad, right side, combinations.

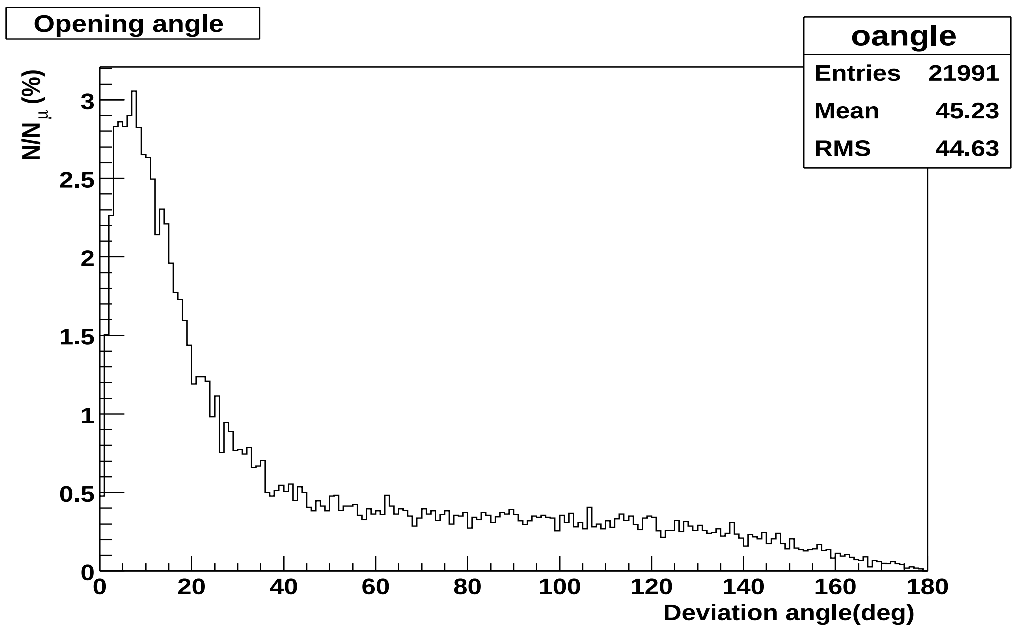


Figure 5.39: Opening angle between the muon an pion in $\pi^{\pm} \rightarrow \mu^{\pm} + \nu_{\mu}$. Produced using an HGEANT simulation

In a typical reaction, after protons, the most common particle species are pions. Pions have a mean life $\tau \simeq 26ns$ and decay 99.987% of the time in the channel

$$\pi^{\pm} \rightarrow \mu^{\pm} + \nu_{\mu}$$

There is a certain probability that this decay happens after the inner MDC chambers, but before the META detector. Since the particle producing the hit in META and MDC are different, we classify the combination of the two as a bad one. On the other hand, as shown in Fig. 5.39, the muon tends to inherit the pion's direction, therefore these combinations have the correct correlation between polar and azimuthal deflection and thus peak at zero in the *xPull* plot.

In order to make a rough estimate of this effect let's calculate the probability of a particle decaying before a time t_0; this probability depends on the particle's proper life time τ and speed β through the expression

$$P(t_0) = 1 - e^{-\frac{t_0}{\tau\sqrt{1-\beta^2}}}$$

A typical pion travels at $\beta \simeq 0.85$. It leaves the MDC after 4ns and reaches the META at around 10 ns. Taking into account that $\tau = 26$ ns, the

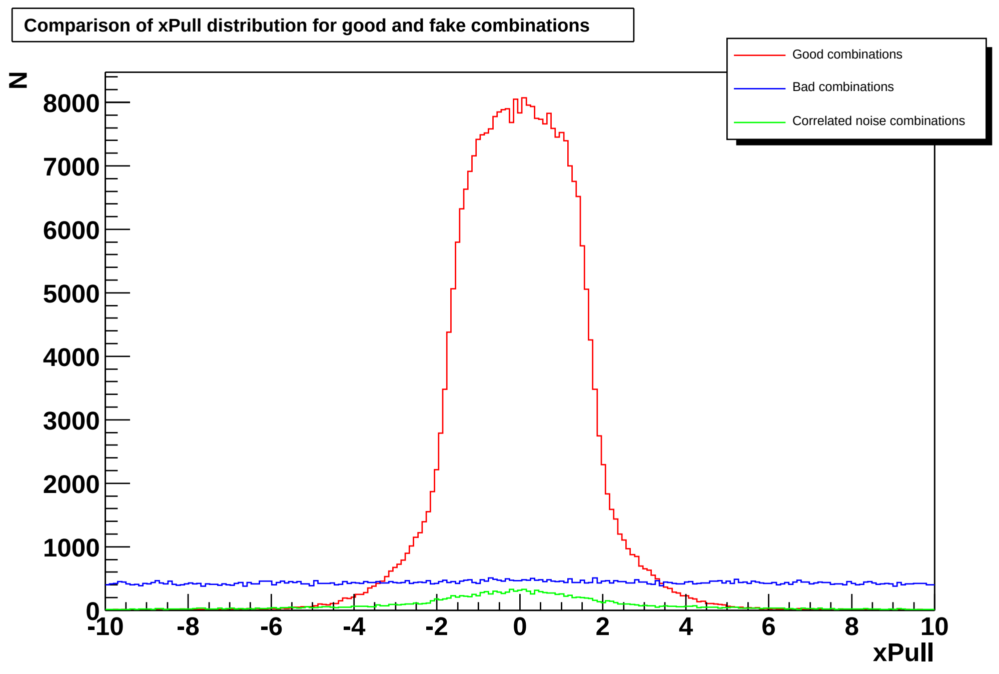


Figure 5.40: Detail of the $xPull$ distributions for good, bad and correlated noise combinations. Note how the shape of $xPull$ is not gaussian for good combinations. Note also how the distribution for bad combinations becomes flat once we take the correlated noise out.

probability for a particle to live between 4 and 10 ns is about 30%. Assuming that everything produced in the reaction is either a pion or a proton and pions represent 10% of the protons, that renders a 3% of the produced particles appearing as correlated noise. Which, even if it is a rough estimate it is still an order of magnitude to keep in mind.

That much for the $xPull$ distribution on bad combinations. On good ones the most outstanding feature is for the shape not to be gaussian like, specially when using the SHOWER detector, as shown in Fig. 5.40. As explained before, when using the SHOWER detector, x_m is estimated as the center of the SHOWER pad hit by the track. So in fact the probability distribution associated with the measurement is not a Gaussian but a flat one, thus the flat shape of $xPull$. Models for the $xPull$ distributions in the different cases will be presented in section 6.5.4.

Fixing the cut value

The one thing left is to find the optimal value for the cut. This can be done by plotting the efficiency, noise and correlated noise as a function of the cut value. Then we can choose a cut depending on the target efficiency and noise level we are willing to tolerate.

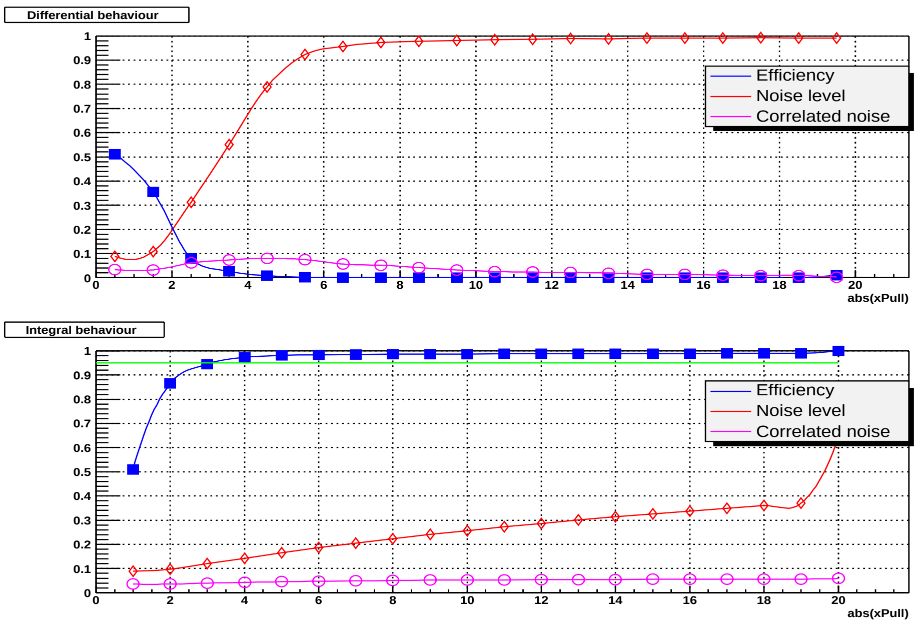
To make this plot, we need a way to confirm if a combination is a good or a bad one we use the HGEANT Monte Carlo. The only relevant information about the usage of a simulation here is that it should reproduce the noise environment in the actual experiment; otherwise the noise estimation is optimistic. For that reason we simulated in HGEANT the same system as in the experiment: C+C, and at the same energy. That takes good care of the physics noise. As for the electronics noise a completely different approach is in the works at GSI; namely embedding tracks simulated with HGEANT into real events. This infrastructure was not fully implemented by the time of this writing.

In any case, the resulting curves from simulation are shown in Fig. 5.41. The noise curve represents the total level of noise, including correlated one. In this case we were targeting an efficiency of 95%, which is shown as an horizontal green line.

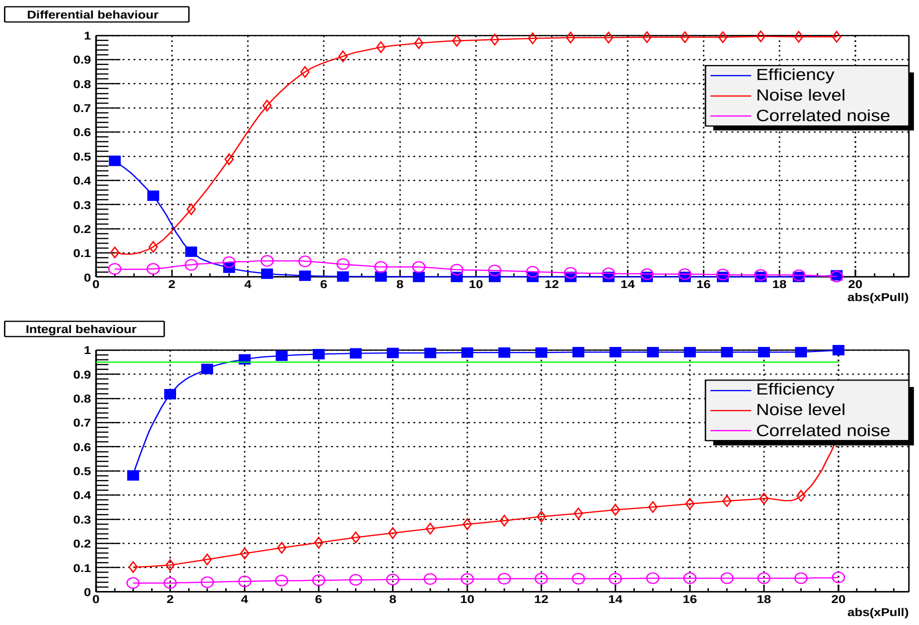
In fact, even though the matching algorithm is based on $xPull$ it is more complex than just performing a cut. As we have shown $xPull$ has tails. That means that in order to get high efficiencies it is necessary to open the cut quite a bit, otherwise we would loose all the good combinations sitting on the tails. However, most of the good combinations are still sitting close to 0. What we do is perform a first, stronger, cut on $xPull$. Then, we not only take out of the sample those combinations passing the cut, but also any other one sharing an element (segment or hit in Meta) with a combination which has passed the cut. On a second stage a wider cut is used, but this time the level of noise is lower because of all the bad combinations taken out in the previous stage.

What we have done for 2 cuts can be generalized for any number of them: $c_1, c_2, \ldots, c_n$. For each cut c_i we identify the combinations with a lower $xPull$ and remove them from the sample so that their hits are not used anymore. Note how this is very different from taking always the combination with the best $xPull$. The reason why doing it stepwise is better is clear when considering close lepton pairs.

It is rather common that two leptons leave the RICH radiator gas with opening angles around 0.5 degrees. These typically come from pair production in the RICH radiator gas. As illustrated in Fig. 5.42 both leptons remain



(a) Ideal tracking with $\sigma_y = 80\mu$



(b) Santiago tracking

Figure 5.41: Efficiency, noise and correlated noise curves

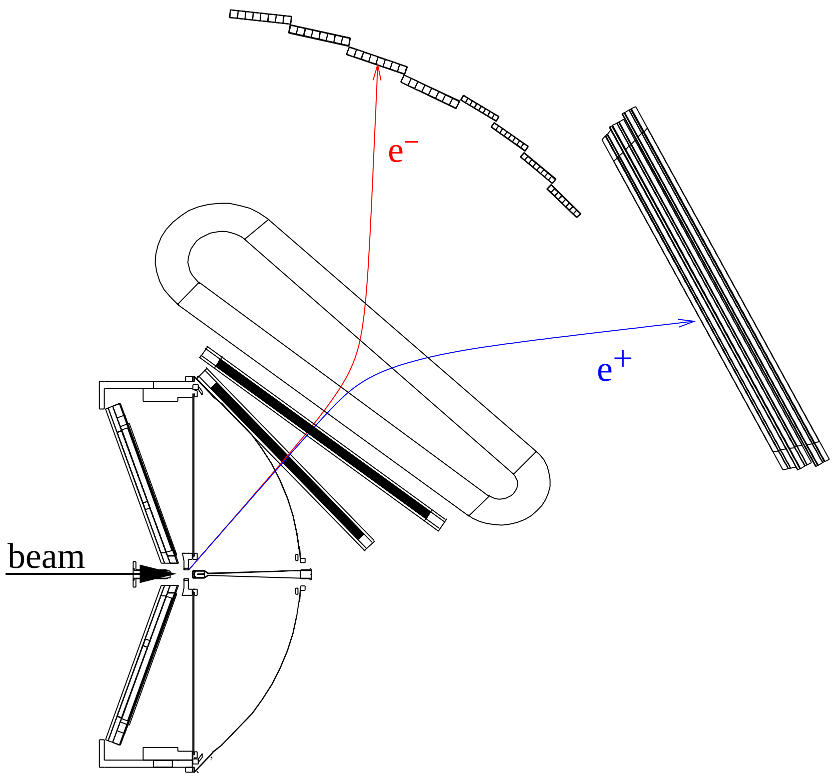


Figure 5.42: Close pair. The same segment in the inner chambers can be matched to two different hits in META

very close in the inner part of the spectrometer, before the magnetic field. So they are typically reconstructed as one single segment[14]. In the magnetic field, the two leptons, having opposite charges, are deflected in opposite directions, so that when they reach the META, they are recorded as two separate hits.

This case shows how it is possible that one inner segment is legally associated to more than one hit in META. For that reason we cannot just take the combination with the best $xPull$. The algorithm shown here allows to reduce a bit further the noise level with minimal impact on efficiency. This way the final average efficiency does not change and the noise level values for C+C on simulation with cuts on $xPull$ 3 and 6 is reduced to around 15%

[14]There is a number of variables one can look at in a segment to know if it is suspicious or not of being a close pair. For the sake of simplicity we will ignore them here. For more information consult J.Bielcik

5.5.2 Matching inner and outer MDCs

The goal is to match a segment from the inner MDCs with a hit in MDC3. Even if only one chamber is available in the outer region of the spectrometer, a segment can be built using the direction information it provides. So, in fact, what we need to match is the inner and outer segments. Each of the segments defines a straight line with parametric equation:

$$\mathbf{r}_i = \mathbf{r}_{0i} + \alpha_i t$$

where i is 1 for the inner segment and 2 for the outer one, t is a real number, $\mathbf{r}_0$ is a point in the straight line and α is the direction unitary vector.

The two segments are defined by eight (4+4) measurements. However, we know that a track is completely characterized by 5 parameters $(1/p, \rho, z, \theta, \phi)$ as shown in section 5.2.1; therefore there are 5 degrees of freedom. There must be 3 constraints relating the eight measurements. There are several standard methods to find those constraints, one of the most notable ones is Principal Component Analysis (PCA, see (6)).

The basic assumption behind PCA is that the constraints among the parameters are linear. In that case, if the measurements corresponding to real tracks verify the constraints then they sit onto a 5 dimensional hyper-surface in the 8 dimensional measurement space. The constraints being linear means that the surface is *flat*, i.e., it is possible to find a coordinate transformation such that in the new coordinates the hyper-surface is defined by making three of the components equal zero. PCA provides a way to obtain such coordinate transformation and an indication of which coordinates become zero. Then the task of matching is simply applying cuts on those three variables.

The problem in our case is that the constraints are not linear. This can be solved in two ways; one is dividing the measurements space in small regions, applying PCA in each of them. This is like approximating the non linear constraint by a set of linear ones. The other approach is to derive a set of non linear constraints based upon a track model. This last method is the one we present here. It is simpler, uses less memory and gives a good enough efficiency on C+C.

The matching algorithm presented here is based in the kick plane model. In this model two segments belonging to the same track cross at one given spatial point and that cross point sits on a surface, the so called kick surface. Using that as inspiration, one can look at two variables for each combination of two segments:

1. The distance between the two segments, (see Fig. 5.43). For good

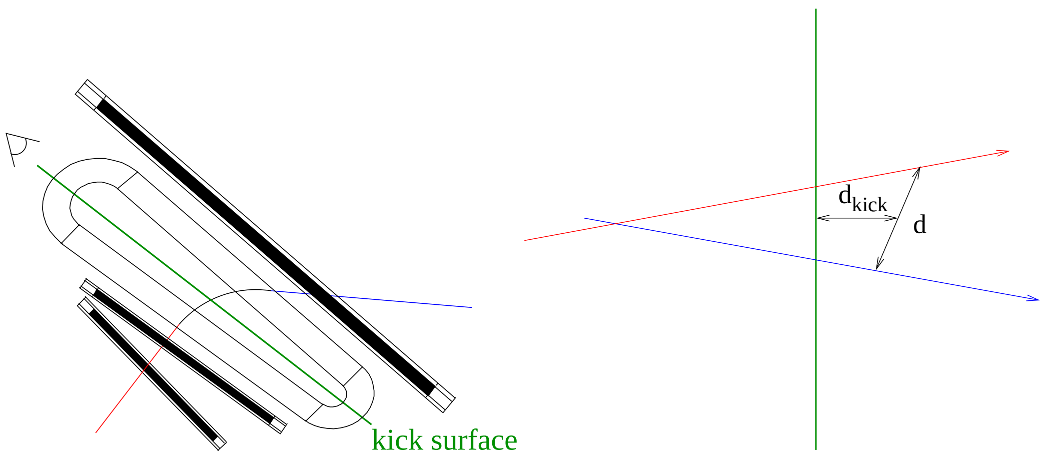


Figure 5.43: Definition of d and d_{kick}. The figure on the left side represents a track traversing one sector, it shows the view point for the figure in the right side.

combinations, this distance should be close to zero, according to the kick plane model. On the other hand, bad combinations have larger distances. The distance between two segments is the distance of closest approach between them, which can be calculated as

$$d = (\mathbf{r}_1 - \mathbf{r}_2) \cdot (\overrightarrow{\alpha}_1 \times \overrightarrow{\alpha}_2)$$

2. Let's call $\mathbf{r}_c$ to the point of closest approach between the two segments (see Fig. 5.43). The second variable is the distance between $\mathbf{r}_c$ and the kick surface: d_{kick}. Ideally, this should be zero since the segments cross on the kick surface. The point $\mathbf{r}_c$ is nothing else but the vertex defined by the two segments. So it can be calculated with the vertex reconstruction algorithm introduced in chapter 4.

These two variables complement each other. Since all tracks come from the interaction vertex, d can be small for a combination of two segments belonging to different tracks, but then the point of closest approach will be close to the target region and therefore d_{kick} would be larger.

Up to know we have identified two out of the 3 constraints: d and d_{kick}. A third constraint is the correlation between polar and azimuthal deflection, which is the same variable as used in the matching between MDC and META. Both the total and the azimuthal deflection depend on momentum. From the total deflection we can estimate momentum and then use that estimate to make a prediction on the expected azimuthal deflection. We define Δ_ϕ as the deviation of the measured azimuthal deflection from the predicted one. For good combinations it should be close to zero.

The matching algorithm presented here works by evaluating the matching variables d, d_{kick} and Δ_ϕ for each possible combination of one inner segment and one outer segment. Then good combinations are discerned from bad ones by placing a three dimensional cut on those variable values. The shape of the cut is what determines the algorithm's efficiency and noise level, therefore we need a way to optimize it. If the evaluated variables were gaussian the cut's shape would just be an ellipsoid in the three dimensional space defined by an equiprobability surface. Unfortunately, that is not the case. In particular, as already commented, the tails in the reconstruction of direction in MDC3 have a strong influence.

For this reason an automatic method to determine the optimal cut shape given a target efficiency is needed. Such an algorithm will be presented in section 5.5.2. But before that we will spend some time looking to the distribution of the different variables both for good and bad combinations.

d, d_{kick}, Δ_ϕ distributions

The shape of the matching variables distributions depends fundamentally on which kind of tracking is used to construct the inner and outer segments in each candidate. There are three tracking algorithms active in HYDRA:

- Ideal tracking: uses information from HGEANT to simulate the response of an ideal tracking algorithm, that is efficiency is 100% and resolution is perfectly Gaussian and according to design values.
- Santiago and Dubna trackers: complex algorithms constructing segments from the low signals recorded by the MDC chambers. Both of them are under development.

Fig. 5.44 shows the distributions on d, d_{kick}, Δ_ϕ both for good and fake combinations in the case of ideal tracking; while Fig. 5.455.45 corresponds to Santiago tracking. We see how in the ideal case the variables have nearly no tails; however in the more realistic scenario tails are very significant. The second bump in d_{kick} corresponds to poorly reconstructed slopes in MDC3. The conclusion is that we need a way to perform matching in the presence of large tails; this method is presented in the following section.

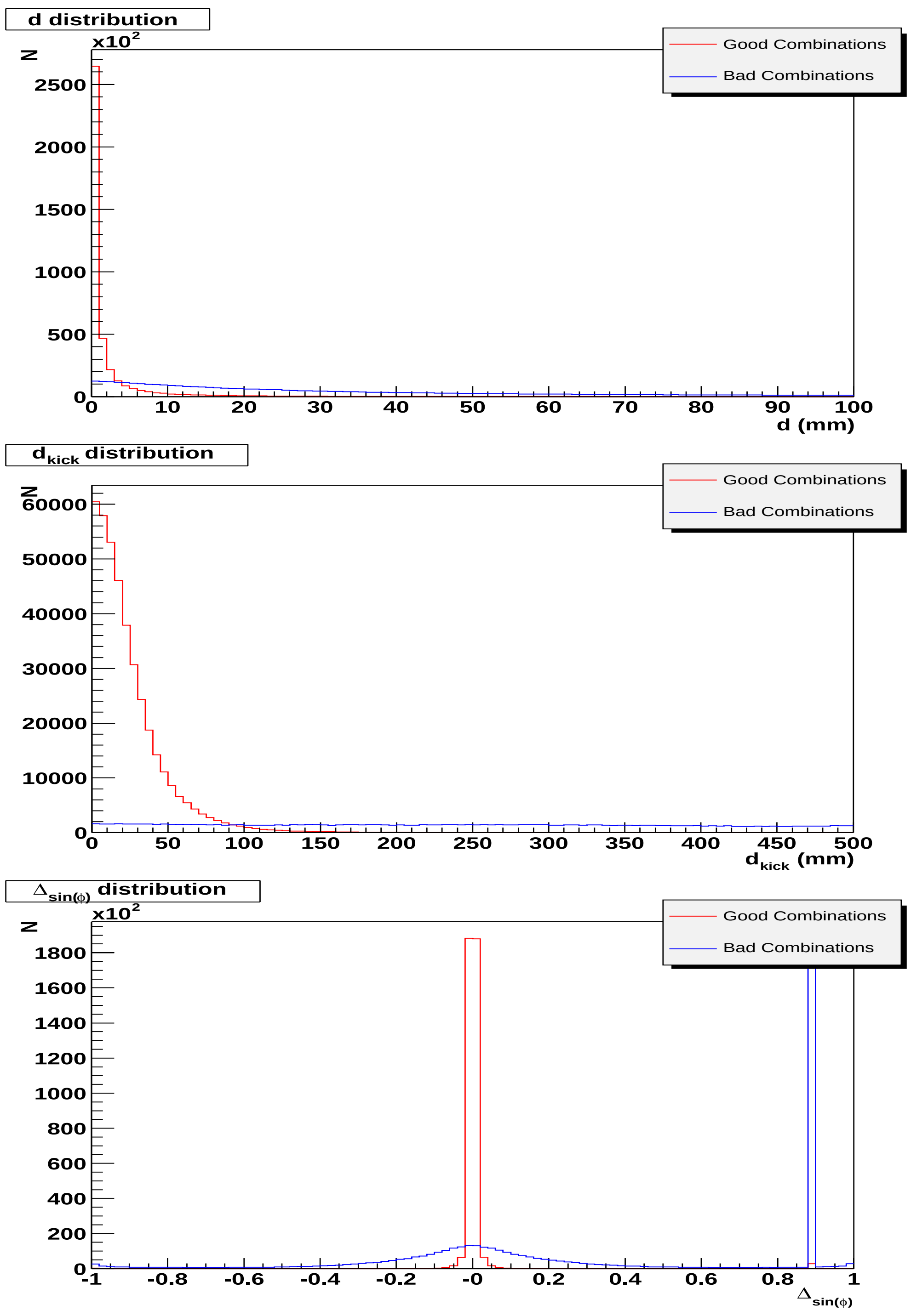


Figure 5.44: Matching variables distributions for ideal tracking. The peak at 0.9 for $\Delta_{sin(\phi)}$ is artificial and corresponds to those tracks where the algorithm is not able to compute the expected azimuthal deflection.

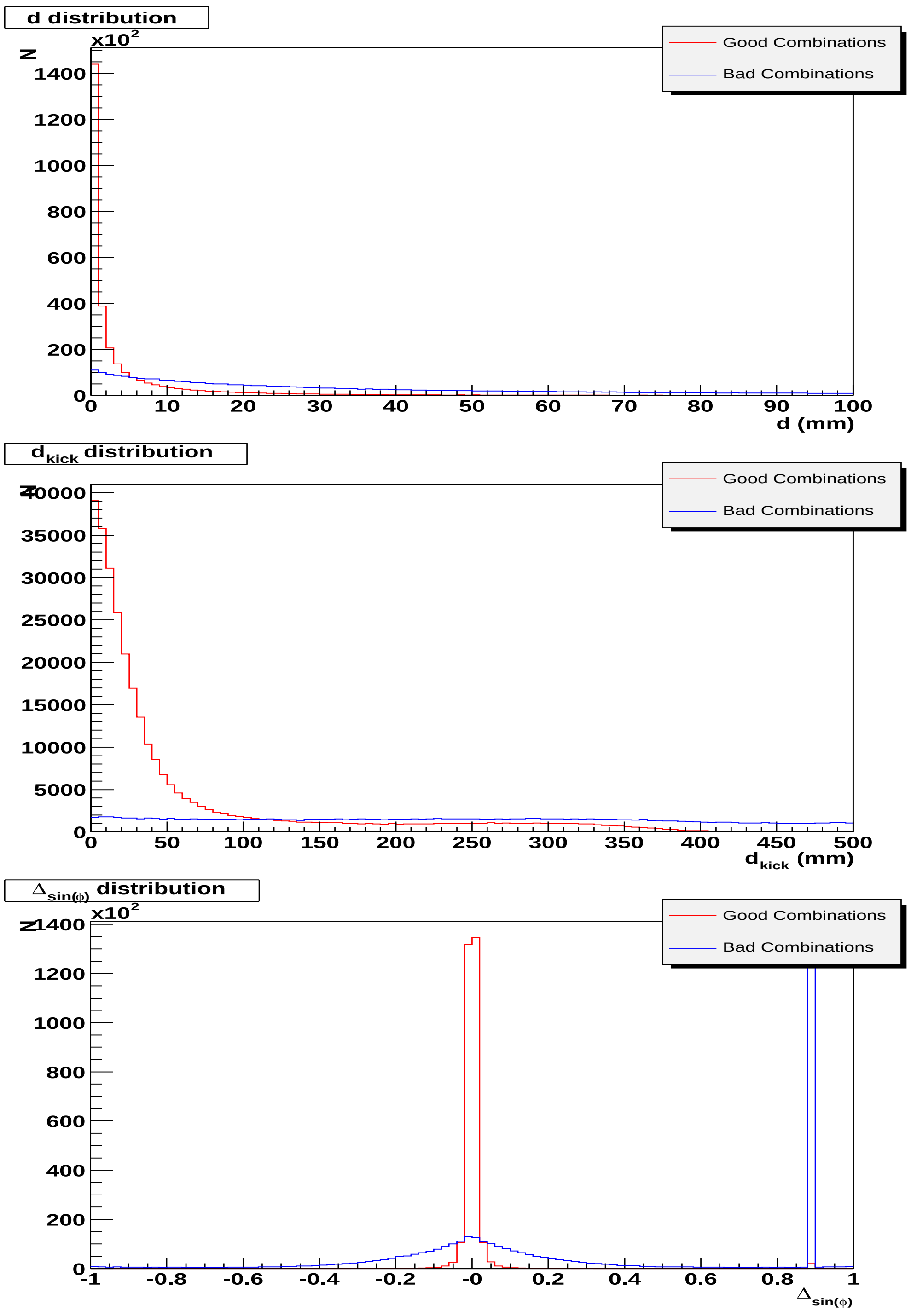


Figure 5.45: Matching variables distributions for Santiago tracking. The peak at 0.9 for $\Delta_{sin(\phi)}$ is artificial and corresponds to those tracks where the algorithm is not able to compute the expected azimuthal deflection.

Determining the cut's shape

We have seen how any of the distributions can have tails, but counts sitting in the tail of one variable do not necessarily sit in the tail of another. There is efficiency to be gained by allowing the cut to have an arbitrary shape.

In order to devise an algorithm capable of automatically deriving that shape we have taken a very pragmatic approach. The three selection variables are used to define a three dimensional space, the selection space; in that space each combination of segments is represented by a point. Good combinations are sitting on a volume, which is itself smeared due to the detector's resolution.

From a simulation of the system of interest, C+C for example, we can populate the selection space with all possible combinations. Internally keeping track of which ones where good and which ones were bad. Then the selection space is divided in cells; for each of them a signal to background ratio is computed. That is simply the number of good combinations divided by the number of bad ones.

The next step is to order the cells in a list going from the higher signal to background ratio toward the lowest. The cells in the list are marked, after marking each one the number of good and bad combinations in the cell is added to the overall ones; thus computing an overall efficiency. When a previously specified overall target efficiency is achieved we stop marking cells. The resulting set of marked cells provides the target efficiency with the lowest possible noise level, thus defining the cut's shape.

When a combination needs to be evaluated we first compute which cell does it fall into, if it is one of the marked ones the combination is accepted. Otherwise is rejected.

It should be noted that total statistics must be chosen such that statistics in each of the cells are significant.

Results

The basic results from a matching algorithm are its efficiency and noise. Table 5.2 summarizes those values for different tracking algorithms in use in the HADES experiment when the matching is done between the inner MDCs and MDC3. Those figures have been obtained from a HGEANT simulation of a C+C system; efficiency and noise have the meanings introduced in section 5.5. The figures in the table represent the efficiency and noise levels integrated over solid angle and momentum with a weight distribution which is given by the physical distribution of particles in a C+C simulation.

	Ideal tracking	Santiago tracking
Efficiency	98.8%	98%
Noise level	1.5%	8.6%

Table 5.2: Efficiencies and noise levels for matching between inner MDCs and MDC3

The dependency of the two variables with momentum, polar and azimuthal angle is shown in Fig. 5.46 for ideal tracking; the results using the Santiago tracking are presented in Fig. 5.47. In order to obtain the numbers in the table from the graphics it should be noted that the physical distribution of particles coming out of a C+C collision is roughly exponential in momentum and peaks at the lower polar angles due to the Lorentz boost.

5.5.3 Matching MDC track candidates with META

When in medium resolution or high resolution setups, we can measure momentum from the MDC alone. But we still need to match the resulting track candidate to a hit in the META detectors. Doing so also allows to compute momentum using the low resolution algorithm. Then the two momentum measurements can be combined to improve resolution.

We could naively expect that matching MDC track candidates with META is simply a matter of extrapolating the outer MDC segment to the META detector as a straight line an then compare the extrapolated and measured points.

The problem with such simple approach is that it ignores the residual magnetic field in the outer part of the spectrometer. Fortunately we already have all the ingredients to take that effect into account.

First let's do the usual counting of degrees of freedom to know how many constraints we can expect. A MDC candidate is defined by 5 parameters, in addition we have two measurements from the META detector. The full track candidate is defined by 5 parameters, thus we can expect two additional constraints.

One constraint we already know, it is the relation between polar and azimuthal angle already parameterized for the META detector in the form of the *xPull* variable. The other constraint is that the polar deflection measured using the META and MDC detectors must respond to the same track's momentum, ignoring the energy loss in the air.

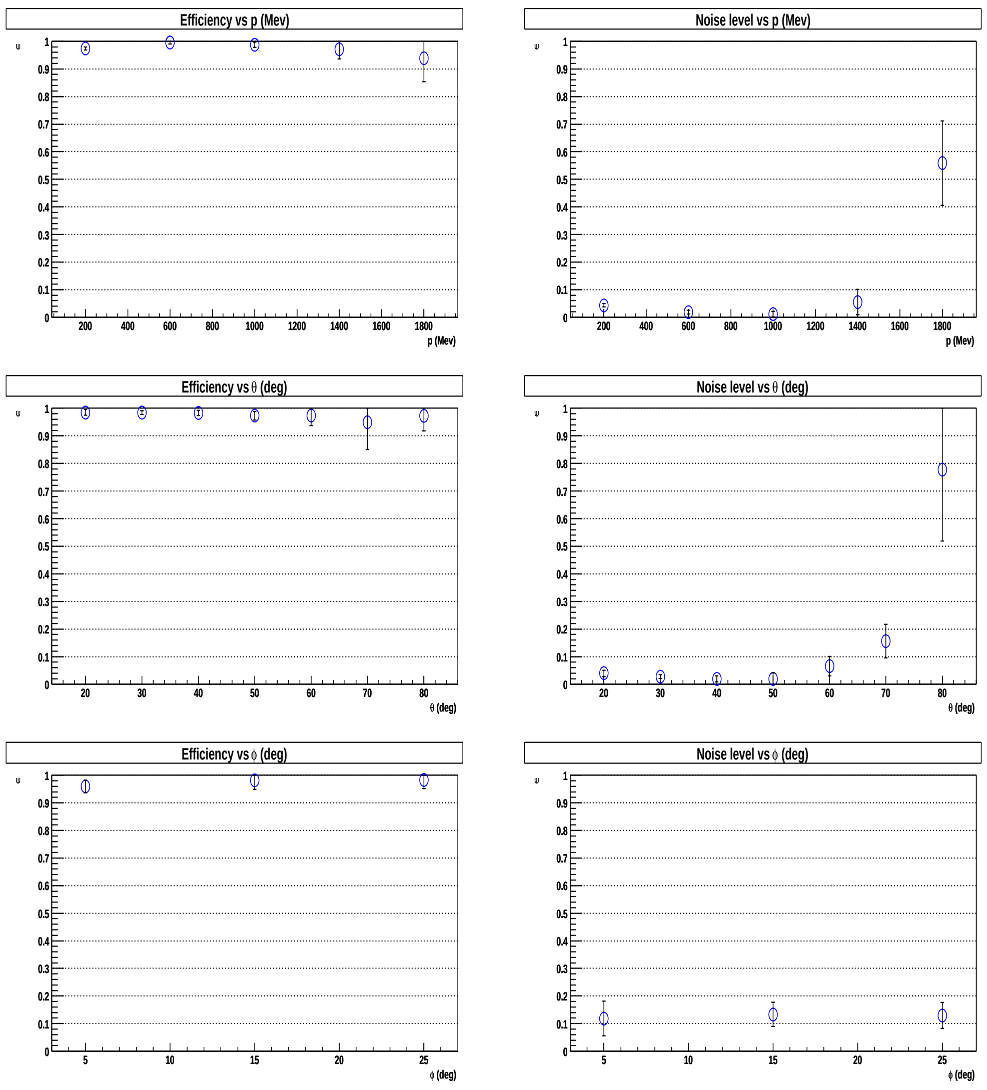


Figure 5.46: Efficiency and noise level of matching between inner MDCs and MDC3 as a function of momentum, polar and azimuthal angle. Ideal tracking.

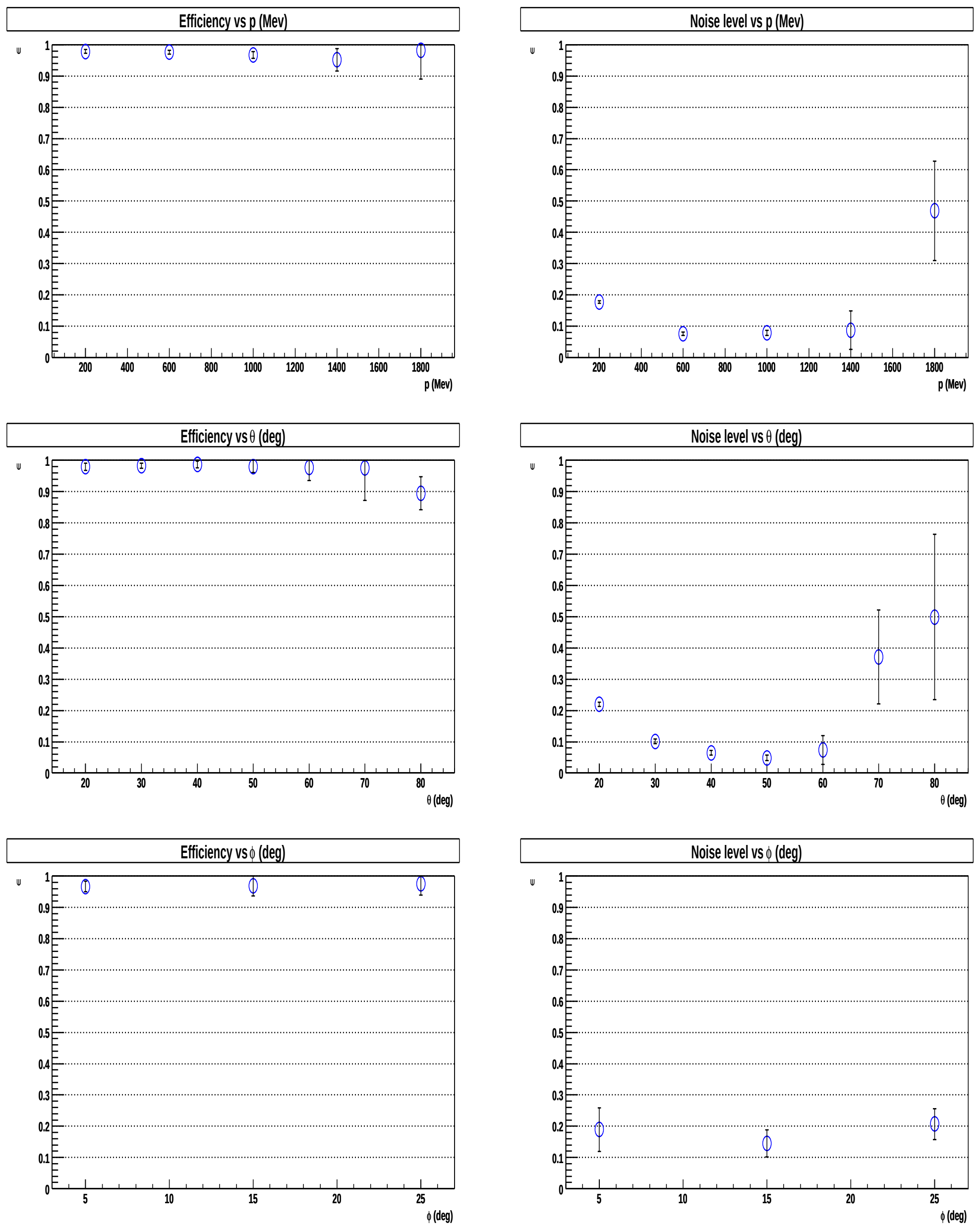


Figure 5.47: Efficiency and noise level of matching between inner MDCs and MDC3 as a function of momentum, polar and azimuthal angle. Santiago tracking.

	Ideal tracking	Santiago tracking
Efficiency	90%	90%
Noise Level	3.5%	4.7%

Table 5.3: Efficiencies and noise levels for matching between MDC track candidates and META. The MDC track candidates are formed using MDC3

Since we know, and anyhow need to, calculate the momentum using META, another selection variable can be constructed as the normalized difference between the momenta determined from META and MDC3

$$dPnorm = \frac{1/p_{Meta} - 1/p_{Mdc}}{\sqrt{\sigma^2(1/p_{Meta}) + \sigma^2(1/p_{Mdc})}}$$

As in the previous case, these two variable's distributions are affected by the characteristics of the tracking algorithms used internally. For that reason the optimal cuts on the selection variables are obtained using the same algorithm as in the previous section. In fact, combinations falling in the tails for the selection variables used to characterize the MDC candidates do not necessarily fall in the tail for the two extra variables. It is then convenient to do the matching in two steps.

In the first step, when creating the MDC candidates, we leave very open cuts; targeting an efficiency of 98% whatever the noise level is.

In the second step, we match all those combinations with the META detectors, and now place more stringent cuts on all five variables; targeting for efficiencies in the order of 95%. In this way the META is used as a filter for the bad combinations accepted in the first step. In other words, suspicious combinations are kept until all information about the track candidate is known.

Results

The resulting efficiency and noise ratios of the preceding algorithm are shown here for the scenario where the MDC track candidate was formed from the matching of the inner MDCs with MDC3. We can get different results depending on the tracking algorithm; those results are summarized in table 5.3 and more detailed behavior is presented in Fig. 5.48 for Ideal tracking and 5.49 for Santiago tracking.

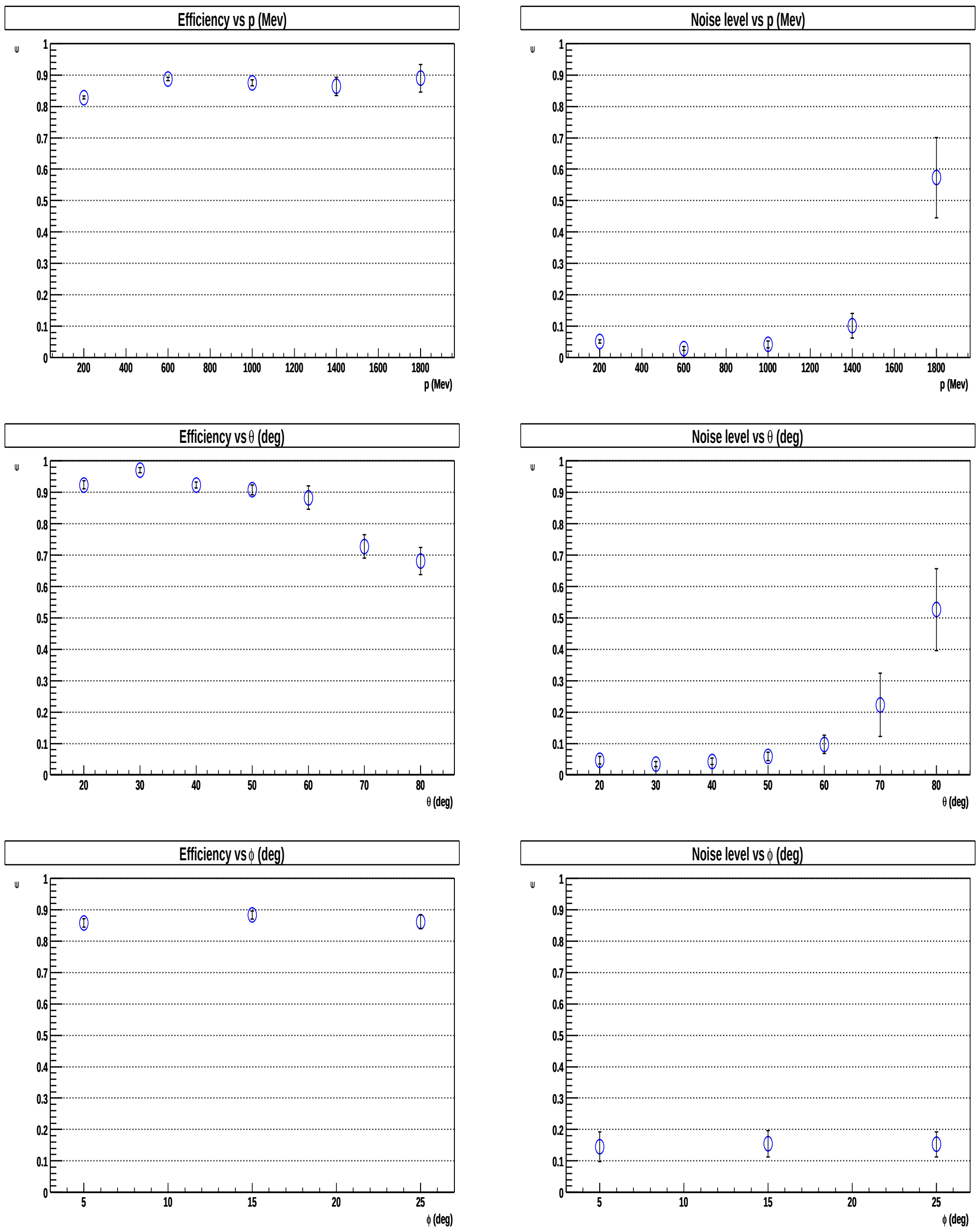


Figure 5.48: Efficiency and noise level of matching between inner MDCs and META as a function of momentum, polar and azimuthal angle. Ideal tracking.

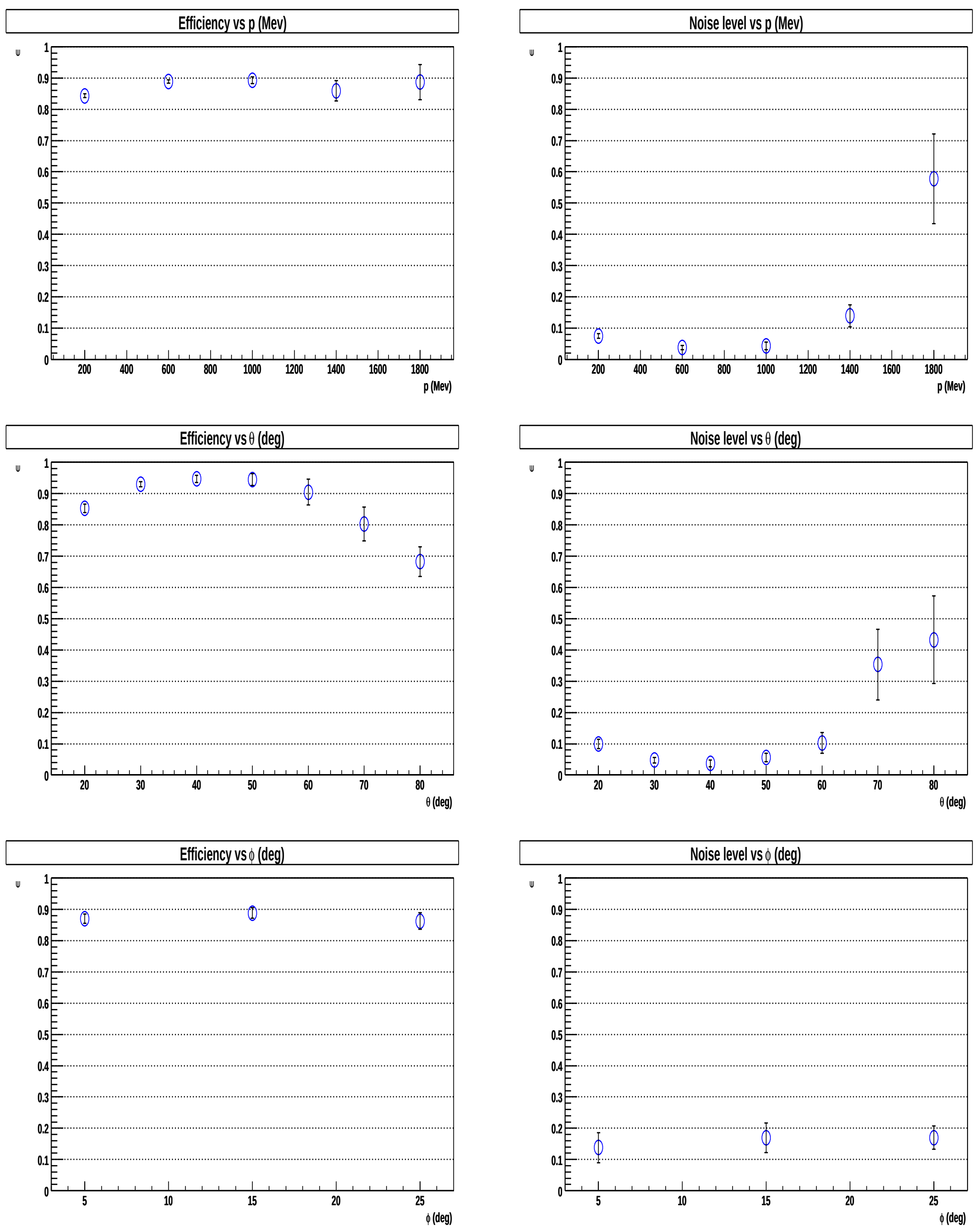


Figure 5.49: Efficiency and noise level of matching between inner MDCs and META as a function of momentum, polar and azimuthal angle. Santiago tracking.

Bibliography

[1] Jim Carroll. *Measurement of* e^+e^- *pair production at BEVALAC.* Nuclear Physics, A(495):09c-422c, 1989

[2] William H. Press, et al. *Numerical Recipes in C: The art of scientific computing. Pag 650*

[3] http://www.oonumerics.org/blitz

[4] Eric Gamma et al. *Desing patterns*

[5] R.Bock. *The particle detector briefbook*

[6] I.T. Jolliffe. P*rincipal Component Analysis.* Springer-Verlag, 1986.

[7] HADES collaboration. *Proposal for a high acceptance dielectron spectrometer.* GSI, 1994

[8] Th. Bretz. *Magnetfeldeigenschaften des Spektrometers HADES.* Diploma Thesis. Technical Universität München, 1999

[9] William H. Press, et al. *Numerical Recipes in C: The art of scientific computing.*

Chapter 6

Pion production

The goal of this chapter is to study the production of charged pions in C+C collisions. The results herein obtained are compared with those from other experiments in order to perform a cross check. That proves to be a very valuable tool for getting information about the quality of our analysis routines and the eventual presence of systematic errors.

The kind of plots to be used should be rather simple, reducing as much as possible the amount of external factors affecting them, but still meaningful. Preferably we should have high statistics so that we have freedom to put cuts on the data.

The data we will be using was recording in a beam time during November 2001 and it is $^{12}C+^{12}C$ at 1.9 AGeV. This is referred as the Nov01 dataset. Even though there are more recent data, we have chosen those because they have already gone through 4 generations of refinement in the alignment and calibration parameters. The first option is to look at leptons, however the dilepton statistics are relatively low for our purposes, due to the fact that the second level trigger was not fully operational at the time, thus reducing the lepton richness of the data. Consequently we have concentrated on a more abundant particle species, namely, pions.

We cannot obtain absolute production cross sections for several reasons. One of them is the lack of scaler data. Scalers are typically used to count the number of beam particles impinging on the target. Without that knowledge we cannot normalize to the beam intensity. We could still normalize to the number of reactions, but then we have the additional problem that the detector efficiencies themselves are not very well known. This point will be treated in more detail in section 6.5.4.

The choice was therefore to look at quantities which are not very sensitive to an absolute normalization but to the shape of the spectra. This means that corrections for acceptance and efficiency are still needed as long as they may depend on phase space, but their absolute normalization is less relevant. The chosen quantities are the pion mass, production ratios of positive to negatively charged pions, production ratio of pions to protons and pion transverse momenta spectrum.

The pion mass is a well known quantity, while pion freeze out temperature has been measured previously by other experiments at SIS energies for the same system as ours. This is done by looking at the transverse momentum spectrum and then fitting it to a model based upon the assumption of a thermal energy distribution given by the Boltzmann equation.

The reconstruction algorithm chosen for this analysis is the kick plane. The reason being that during this data taking the outer MDC chambers were missing in all but one sector. In the sector with outer chambers, only MDC3 was fully operational. That corresponds to the low and medium resolution setups in the previous chapter; so the kick plane is the natural choice.

Some corrections on the data are needed in order to account for the spectrometer acceptance as well as the reconstruction efficiency and noise level of the kick plane. Actually, a first order approach would have been to trust the efficiency and noise level estimations obtained from simulations. However we have gone one step further and tried to estimate them from the real data; this is particularly relevant for the noise level estimation since the noise environment in real life is higher than in simulations.

The first section of this chapter presents the experimental setup available in NOV01. The next sections present the different corrections on the data, while the last sections go into physical results. The experimental results are compared to simulations.

6.1 Experimental setup

The NOV01 data set corresponds to $^{12}C +^{12} C$ collisions at 1.9 AGeV. The experimental setup available at the moment of the data taking consisted of the RICH detector, six sectors equipped with MDC2, one sector with the first MDC and another one with the outer MDC3. All six sectors were equipped with TOF and SHOWER detectors.

This means five sectors with the low resolution setup and one sector with the medium resolution. In both cases the momentum reconstruction algorithm used is the kick plane in its two variants.

The target was a cylindrical carbon target of 5mm length, 8 mm diameter and a density of 2.15 g/cm^3. Interaction length between 4% and 5%.

The vertex was displaced with respect to the design position around 30 mm upstream.

The magnet field was not operating at full current and only 72% of its full power was used. This was dealt with in the kick plane parameterizations by simply scaling the field map in HGEANT. The same effect is achieved by scaling the kick plane parameters themselves. A re-parameterization was anyhow needed since the target was displaced with respect to the ideal position by 30 mm upstream[1]. That systematically changes the incident angle on the kick plane for particles coming from the target and thus, the parameters relating deflection and momentum also change.

A reduced field means the window of momentum acceptance is displaced toward the lower momentum range, since the same momentum corresponds now to a larger deflection. In other words, it makes possible to measure lower momenta than if the full current was used, but on the other end of the spectrum the resolution for high momenta becomes worse. This affects mostly heavy particles like protons or deuterons with high momenta, but is not a problem for pions which have momenta up to 1.5 or 2 GeV.

The total statistics amount to 50 million $C + C$ events.

6.2 Particle Identification

Particle identification is a complex topic in itself. Information can be obtained from a wide variety of variables measured in HADES. In order to maximize the efficiency of the overall particle identification it is necessary to merge the information given by different algorithms, each of them looking to one of the variables. One way to do so is that each algorithm provides a probability vector for each particle candidate, each component in the vector representing the probability for the particle to be of a certain kind (electron, positron etc.). Some algorithms like the one based on the RICH is decisive when it comes to identify a lepton but provides no information about whether if the particle is a pion or a proton. Others, like those based on the examination of time of flight versus momentum or energy loss in the TOF scintillator rods are more sensitive to the different hadron kind.

The overall particle identification algorithm is responsible of forming particle candidates out of track candidates and feed them to the different sub algorithms. Those in turn give back probability vectors which then are merged to output a final vector with the joint probabilities for the particle to be of

[1]In the opposite direction to the beam

the specified type. What was described is the Particle Identification system being developed by HADES were several people from different groups are involved. By the time of this writing the system was not fully completed so we have used a simpler approach.

Our approach is equivalent to using just one of the sub algorithms in the full PID setup, the one built upon the kick plane algorithm. For each track candidate we know its charge because that is given by the sign of the deflection angle. We also know the momentum and we know its associated time of flight (or velocity)[2].

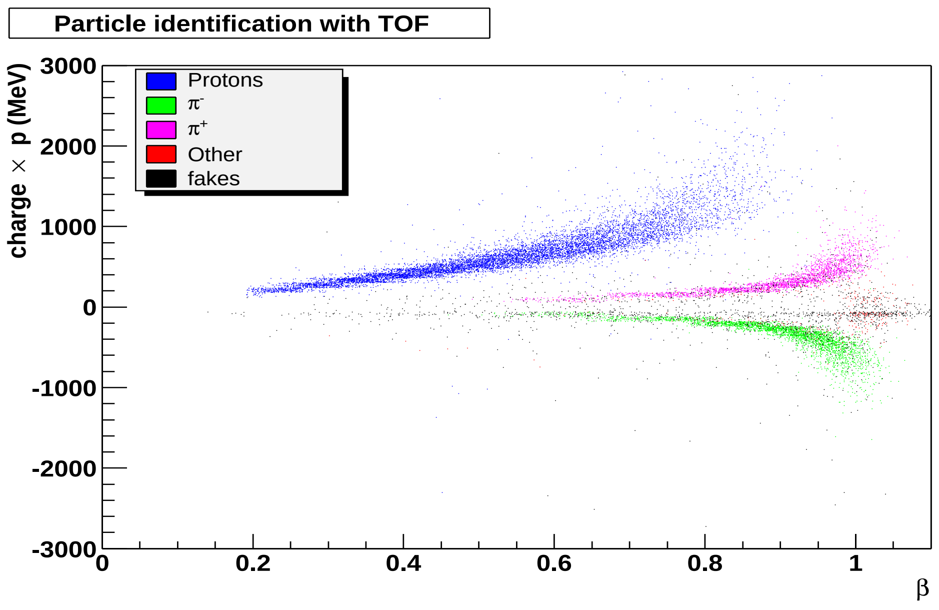
This information is already enough to plot βp versus β and the different particle species should appear in separated curves, each of them characterized by the particle rest mass. Fig. 6.1 shows such plots built on simulations with the *low resolution* setup. The different colors correspond to the different particle species that are seen. In the case of tracks reconstructed using the SHOWER detector the time of flight is measured by a detector placed in front of the SHOWER and which is known as TOFINO. This detector has a worse time resolution that TOF, thus the wider curves, and low granularity (4 pads per sector). The low granularity makes it possible that even on $^{12}C + {}^{12}C$ collisions, two particles hit the same pad in the detector. In such a case, the registered time of flight is that of the fastest particle, but it is not known which of the two was the fastest. So, no time of flight is assigned. In those cases a beta of 0.1 is artificially assigned. This problem does not exist for the TOF because the granularity is much higher (64 rods per sector).

Fig. 6.2 shows the same kind of plot on real data (still low resolution). The noise ratio is increased with respect to the situation on simulation and we see a new particle appearing besides the protons. Those are the deuterons which are not included in the simulation event generator.

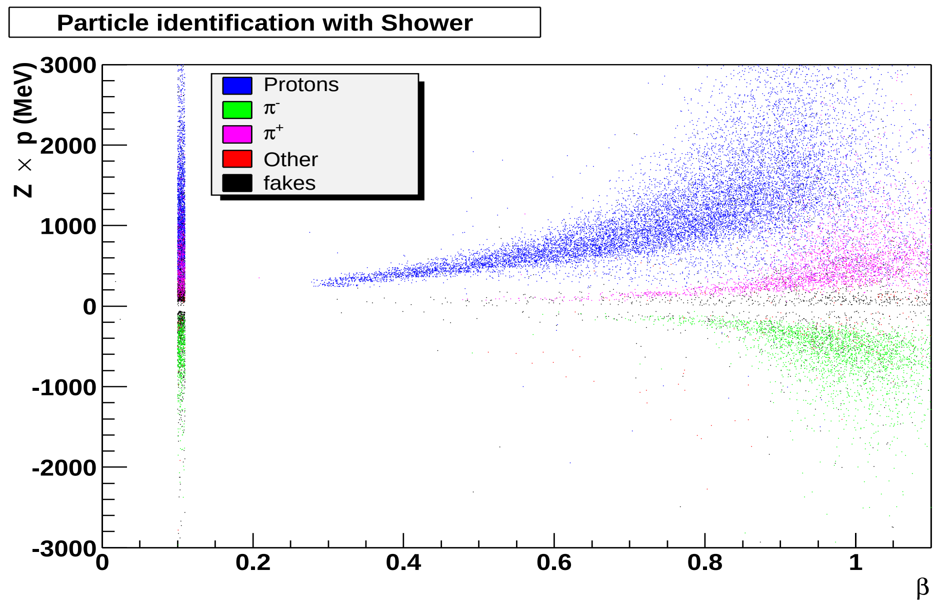
A feature which is present both in simulation and real data, but more clearly visible on real data, is the accumulation of fake tracks with positive charge for the SHOWER detector and negative charge for the TOF. These are low momentum tracks; thus a large deflection is associated with them. For a fake track hitting the TOF detector to have a large deflection its sign is more probably negative due just to phase space, since negative tracks are bent up-wards in the field.

Depending on the kind of analysis to be done on the low resolution data we can have different approaches to perform cuts on the previously presented spectra. If we want to compare π^+ and π^- production, the most sensible way

[2]In the low resolution kick plane we need to match with the META detectors anyhow and those provide the time of flight information. In the medium and high resolution setups algorithms have been presented performing the matching with META.

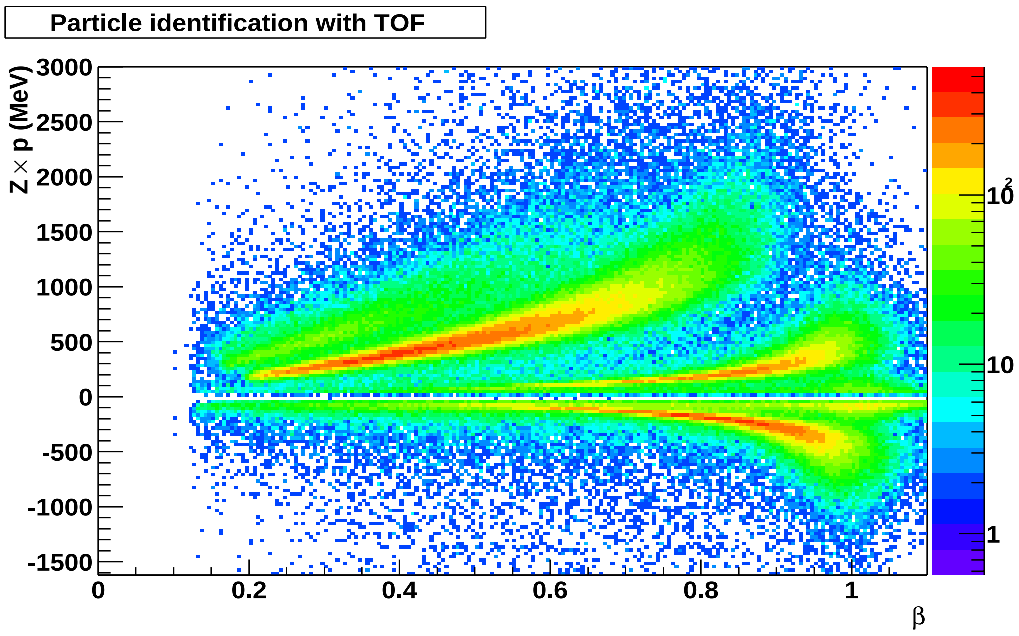


(a) Tracks reconstructed with the TOF detector

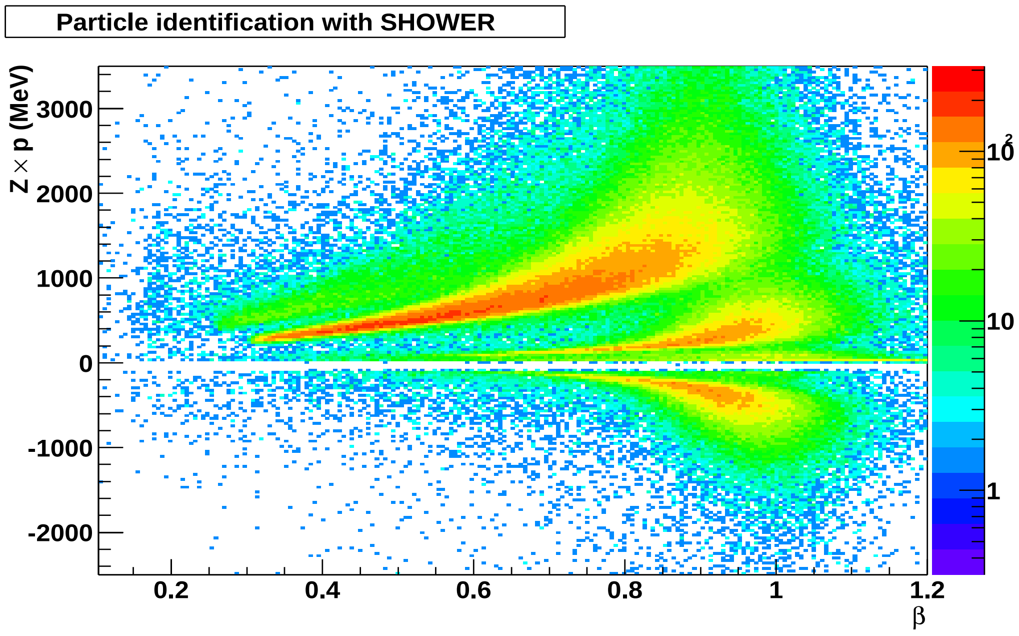


(b) Particles reconstructed with the SHOWER/TOFINO detector. The line at beta=0.1 corresponds to tracks were time of flight was unknown (see text)

Figure 6.1: Particle identification on simulations using the low resolution setup.



(a) PID using the TOF detector



(b) PID with the SHOWER detector. The lower betas have been cut in order not to break the color scale

Figure 6.2: Particle identification using the low resolution setup on real data.

is to impose rather strict two dimensional cuts around the pions using exactly the same, reflected, shape for both polarities. The reason is that by using the same shape we can expect the cut efficiency to be approximately the same[3] and a strong cut contributes to reduce the proton contamination in the π^+. Other kind of analysis will have different requirements.

6.3 Production ratios

In this section we will look at the ratios between different particle species. In particular the π^+ π^- ratio and the ratio of pions to protons. This does not need a knowledge of the absolute efficiencies.

The results presented here are preliminary and will probably be improved once the full PID framework is completed.

6.3.1 $\pi^+\pi^-$ ratio

Due to isospin conservation we would expect this ratio to be 1 in $^{12}C + {}^{12}C$ collisions. That is, the same amount of both kind of pions is produced. Regarding the measured ratio we have to take into account that acceptances for positively and negatively charged particles are different. Most of the tracks produced in the collision have low polar angles. For those kind of tracks a positively charged particles is bent up-wards, that is toward the TOF detector and thus it is registered. Instead a negatively charged particle is bent down-wards, toward the beam pipe where no detector is available and thus the particle has a larger chance to go undetected.

One way to take this into account is running a HGEANT simulation and see what is the ratio using particles inside the acceptance. It can also be done for tracks reconstructed with the TOF and SHOWER detectors separately. The numbers are presented in table 6.1. The table also shows the production ratio between pions and protons. The first thing that catches the eye on that table is the fact that the ratios as measured by the TOF and SHOWER detectors are different, a consequence of their different acceptances as will be shown later. The overall ratio ($R_{\pi\pi}$) can be obtained directly, summing up both contributions, or from the SHOWER one (R_s) and the TOF one (R_T) by

$$R_{\pi\pi} = w_1 R_s + w_2 R_2$$

[3]The absolute value is not relevant as long as we look to ratios.

	TOF	SHOWER	Average	TOF+SHOWER
π^+/π^-	0.73	1.16	0.94	0.94
π/p	0.67	0.40	0.53	0.49

Table 6.1: Production ratios from simulation. Ratios differ from one detector to the other due to acceptance. The simulation used FLUKA for hadronic interactions (pion absorption). The activated physics processes correspond to the defaults in GEANT.

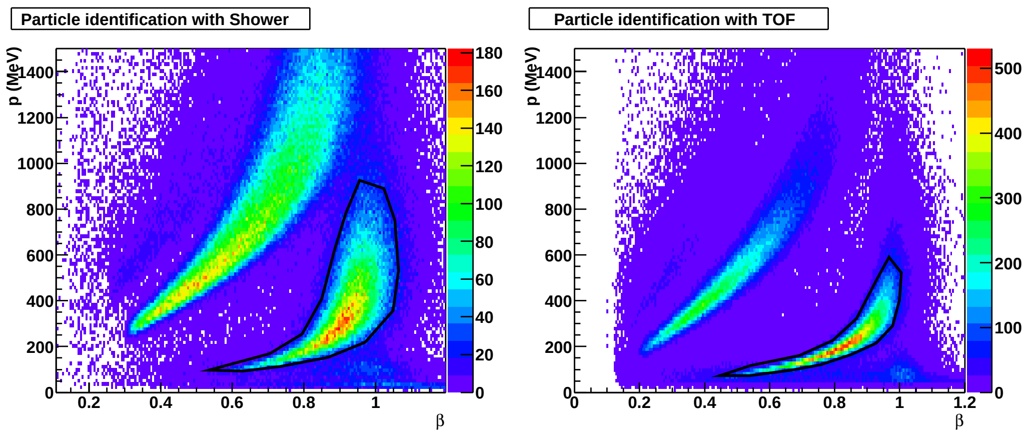


Figure 6.3: Pion identification cut on NOV01 data

Since $w_1 + w_2 = 1$ and from the value for $R_{\pi\pi}$ known for simulation, we can estimate $w_1 = w_2 = 0.5$. This is important because in real data the relative efficiencies of the PID cuts on TOF and SHOWER are not known, so it is not possible to add both contribution in order to obtain the overall ratio. However, the above method is still applicable.

In order to measure this ratio on real data we need first an identification of pions. As explained previously, this is done placing a two dimensional cut on the p versus β spectrum (see Fig. 6.3). There are a number of considerations in the way that cut was chosen. First, the same cut is used for both pion polarities, so that the cut efficiency is the same for both. Second, the cut is intentionally chosen very tight in order to avoid proton contamination in the π^+region.

Two different cuts are needed for the TOF and SHOWER detectors due to their different resolution and due to the problem with low TOFINO granularity. In particular the second effect makes the cut efficiency for both of them to be different. This prevents us from directly adding the number of

	TOF	SHOWER	Average
Ratio ($R_{\pi\pi}$)	0.70 ± 0.02	1.17 ± 0.02	0.93 ± 0.01

Table 6.2: Experimental pion ratios. The individual ratios in each detector differ from simulation due to the higher noise in real data.

pions reconstructed with both systems. Instead, the individual ratios are separately computed and the mean is presented. The TOF tends to assign negative charge to noise thus lowering the ratio. For SHOWER the situation is exactly the opposite: noise tends to be recognized as positively charged particles thus overestimating the ratio. In order to correct for this effect, the algorithm explained in section 6.5.4 was used to estimate the individual noise levels in both detectors, the final experimental results are given in table 6.2. They are compatible with the values from simulation.

6.3.2 π to proton ratio

Another interesting quantity is the ratio of pions to protons. A UrQMD[4] simulation predicts a ratio for $^{12}C +^{12} C$ at 2.0 AGeV of 0.12. From previous experiments a value around 0.1 is expected.

As before, this ratio has to be corrected for acceptance before it can be compared to the experimental results. Once this is done we obtain the numbers in table 6.1. In this case we see that the average of TOF and SHOWER does not correspond to the overall ratio (see table 6.1). This is due to the fact that positive particles tend to end up in the SHOWER detector. When computing the pion ratio, this effect was smaller due to the fact that there was nearly the same number of positive and negative ones.

From the plots already presented, it is evident that it is difficult to clearly separate π^+ from protons, in particular in the SHOWER case, due to the TOFINO resolution. However it is much easier to separate π^- from the sum of both π^+ and protons. That being the case, instead of measuring pion to proton ratio what is determined is the the sum of positive pions and protons divided by the number of negative pions (R_2). The pion to proton ratio (R_p) can be obtained from R_2 and the already known $R_{\pi\pi}$ according to:

$$R_p = \frac{1 + R_{\pi\pi}}{R_2 - R_{\pi\pi}} \tag{6.1}$$

The cut to select π^+ together with protons should be rather loose this

[4]UrQMD is an event generator

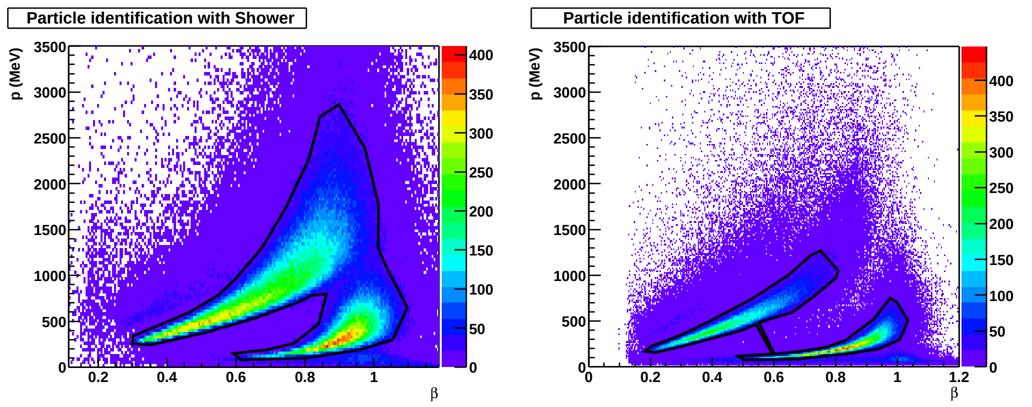


Figure 6.4: Cuts to select $\pi^+ + p$

	TOF	SHOWER	Average
Ratio (R_p)	0.75 ± 0.04	0.41 ± 0.04	0.58 ± 0.03

Table 6.3: Experimental pion ratios. The individual ratios in each detector differ from simulation due to the higher noise in real data.

time. The reason is that, contrary to what happened with pions, we have no direct way to ensure that the cut efficiency for pions and protons is the same, or similar. For that reason the aim is to get a high efficiency so that the fluctuations are necessarily small. On the other side the cut should avoid the low momentum and high beta counts as well as the deuterons, the goal is that the noise is uniform inside the cut to minimize its effect[5]. Fig. 6.4 shows how the cut employed.

Performing the same noise correction as in the previous section the experiment yields the results in table 6.3.

6.4 Mass

Knowing the β and the momentum of a particle, its rest mass is given by the well known equation

$$m = \frac{p}{\beta\gamma} = \frac{p\sqrt{1-\beta^2}}{\beta} = p\sqrt{\frac{1}{\beta^2} - 1}$$

This allows to check for systematics whether in the computation of β or

[5]If it is uniform it could also be estimated by introducing an additional correction

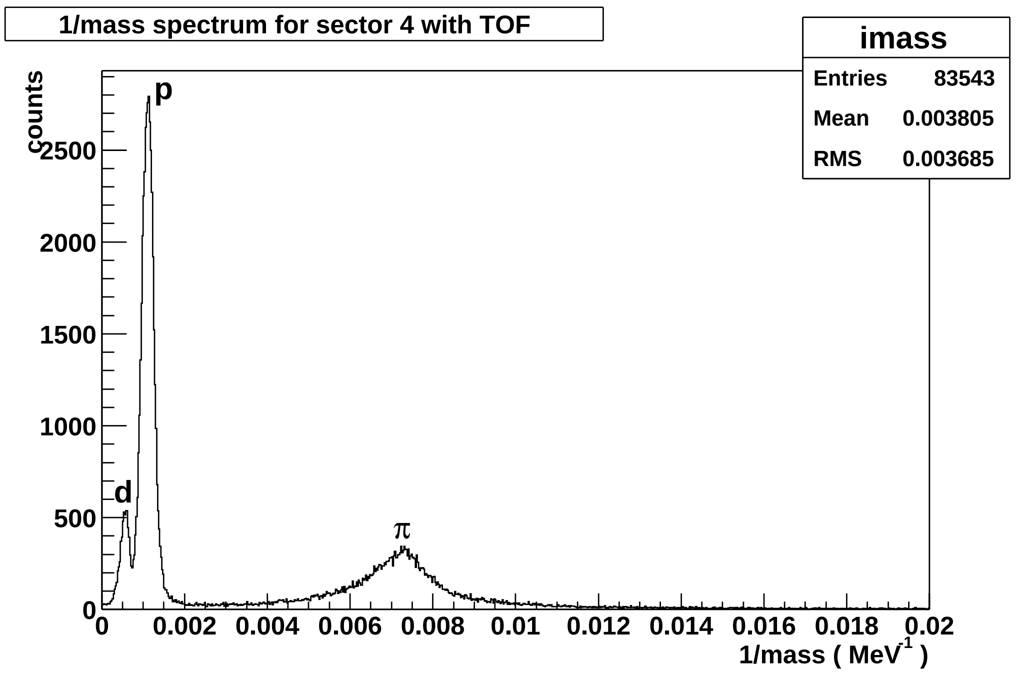


Figure 6.5: Inverse mass spectrum for the Nov01 data. The spectrum corresponds to sector 4 and only the TOF detector was used due to the better time of flight resolution.

the momentum. For fast particles, like pions, systematic errors in β play a decreasingly significant role, while for slow particles the $\frac{1}{\beta}$ behavior makes the calculated mass more sensible to possible systematic errors in velocity rather than in momentum.

Actually, it is not a good idea to measure mass from the formula above directly. The reason is than p is not a gaussian but the inverse of a gaussian distribution[6]. For gaussians with mean close to zero the inverse differs significantly from a gaussian and their peak position is displaced from the mean. Instead of plotting m and fitting as if it were a gaussian, a more accurate approximation is to do it on m^{-1} and then invert the result (see Fig. 6.5). Doing that and correcting momentum for energy loss (see 6.5.2) we get

$$m_\pi = 140 \pm 1 MeV$$

In accordance with the known pion mass. The error comes from effects like the approximation the kick plane algorithm does when computing β. It

[6]p is inversely proportional to deflection and deflection can be approximated by a gaussian.

basically approximates the track by two straight lines and computes β based on that, overestimating the length and underestimating the mass.

6.5 Transverse momentum spectra

The energy distribution of pions is usually expressed in terms of a Maxwell-Boltzmann thermal distribution:

$$E\frac{d^3\sigma}{dp^3} \propto Eexp(-\frac{E}{T})$$

where E is the center of mass energy and T could be associated to the pion temperature at the pion freeze-out in heavy ions collisions. However, there are aspects of pion production that are not well interpreted with a single Boltzmann distribution (see (3)), losing the temperature interpretation. Nevertheless, we will use here the word "temperature" to refer to the inverse slope parameter T. It has also been observed that pion spectra slightly deviate from a single Boltzmann distribution and are thus fitted to a superposition of two Boltzmann distributions:

$$\frac{d^3\sigma}{dp^3} = C_1 e^{-\frac{E}{T_1}} + C_2 e^{-\frac{E}{T_2}}$$

The low temperature component corresponds to a production enhancement at low momenta. Brockmann (4) explains it taking into account the contribution to pion production from the decay of Δ resonances. Table 6.4 presents inverse slope parameters obtained from different experiments for the C+C system at different energies. The NOV01 data set corresponds to 1.9 AGeV. The values of the temperature parameters are correlated with each other and depend on the measured momentum range as well as the angle of emission of the pions.

It has been shown that around mid rapidity the thermal distribution as a function of transverse momentum can be approximated by:

$$\frac{d\sigma}{dp_t} \propto p_t m_t exp\left(\frac{-m_t}{T}\right)$$

with

$$\sqrt{m^2 + p_t^2}$$

The two temperature approximation becomes

$E_{beam}(AGeV)$	Species	$T_1(MeV)$	$T_2(MeV)$	Ref.
0.8	π^0	50 ± 4	-	(7)
1.0	π^0	54 ± 3	-	(7)
1.0	π^+, π^-	45 ± 3	62 ± 3	(8)
1.0	π^-	57 ± 5	-	(6)
1.0	π^+, π^-	76 ± 5	-	(6)
2.0	π^0	83 ± 2	-	(7)
2.0	π^+, π^-	40 ± 3	86 ± 3	(8)

Table 6.4: Inverse slope parameters for C+C at SIS energies. Taken from (5)

$$\frac{d\sigma}{dp_t} = p_t m_t \left[C_1 exp\left(\frac{-m_t}{T_1}\right) + C_2 exp\left(\frac{-m_t}{T_2}\right)\right] \tag{6.2}$$

In this section the p_t distribution for π^- will be obtained and fitted to Eq. 6.2, comparing the results with those in table 6.4. Since this is an analysis on a continuum spectrum, several corrections are needed: energy loss, acceptance, etc. In the first sections, the different corrections are explained, developing an analysis method. Next, the method is applied to simulations in order to check it, and finally the results of analysis of real data are presented.

6.5.1 Particle Identification

This is not properly a correction, but something must be said about the way π^- are selected when building the p_t spectrum. In previous sections this was done through a two dimensional cut on a p versus β histogram. In this case it is convenient to have as high an efficiency as possible. For that reason a wide cut is chosen.

Actually, the reason for looking at π^- rather than π^+ is that π^- can be identified by the charge. That is, we only expect significant contributions to negative charges from electrons and π^-. Electrons correspond to low momenta and high speed and thus can be rather easily rejected, everything left are considered to be negative pions.

6.5.2 Energy loss correction

Charged particles lose energy when traversing matter primarily by ionization and excitation. The mean rate of energy loss is given by the well known Bethe-Bloch equation:

Symbol	Definition
α	Fine structure constant
E	Energy of the incident particle
T	Kinetic energy
m_ec^2	Electron mass in MeV
r_e	Classical electron radius
N_A	Avogadro's number
ze	Charge of the incident particle
Z	Atomic number of medium
A	Atomic mass of medium
K/A	$4\pi N_A r_e^2 m_e c^2/A$
I	Mean excitation energy
δ	Density effect correction to energy loss
T_{max}	Maximum T transferred to an electron in a collision

Table 6.5: Symbols in the Bethe-Bloch formula

$$-\frac{dE}{dx} = Kz^2\frac{Z}{A}\frac{1}{\beta^2}\left[\frac{1}{2}\ln\frac{2m_ec^2\beta^2\gamma^2T_{max}}{I^2} - \beta^2 - \frac{\delta}{2}\right]$$

Which basically is a function of the charge z and β of the incident particle. The different symbols in the equation are defined as in table 6.5.

When a particle, a pion for example, leaves the interaction zone it has a given momentum p_{target}. This particle then suffers energy loss mainly in the target itself and the RICH detector. So when it reaches the MDC chambers and magnetic field region, its momentum is p_{mdc} and $p_{mdc} < p_{target}$. The track deflection in the field is obviously proportional to p_{mdc} as that is the momentum the particle has when entering the magnet. Therefore, what we measure is p_{mdc} and we would need to correct for the energy loss to re obtain p_{target}.

In order to make a first estimation of the expected energy loss suffered by pions we will only consider what is happening in the target itself. The reason is that the target used for the NOV01 data set was a 5mm carbon target, together with the RICH's carbon mirror, that is the most dense material in the trajectory of the pions.

Typical energy loss curves for pions on different materials are shown in figure 6.6. From that figure we learn that pions in our energy regime, from 100 to 1000 MeV suffer energy losses in the range of a few MeV depending on the amount of material they have to traverse. For example a 100 MeV pion would suffer an energy loss around 2 MeV[7] which would yield a variation in momentum around 5 MeV, that is 5% of the original value. This is not negligible. On the higher momenta range, we have the 1500 MeV pion suffering about the same energy loss which results in a net effect in the order of 0.5%.

Actually, what we are really interested in, is the variation in momentum induced by the energy loss. The "momentum loss" can be trivially obtained from the energy loss simply taking into account that

$$\frac{dp}{dx} = \frac{dp}{dE}\frac{dE}{dx} = \sqrt{1 + \left(\frac{m}{p}\right)^2}\frac{dE}{dx}$$

Thus, for a given particle species, the variation in momentum due to energy loss depends only on the particle's momentum. Using the HGEANT simulation packages, an estimation of this effect which is more accurate than the crude approach done before, can be achieved. That package, being based upon GEANT, includes simulation of the energy loss processes suffered by particles when traversing matter. Besides, HGEANT contains the HADES' geometry and materials composition. Hence it can easily estimate the energy lost by any particle traversing the spectrometer.

We have generated pions in GEANT with well defined momenta (p_{target}) and then recorded their momenta when reaching the first of the MDC chambers, p_{mdc}. Since energy loss is a random process, it is necessary to average over several hundred particles to obtain the p_{mdc} corresponding to each p_{target}. Once again, we need that because p_{mdc} is the quantity we measure in real life, but p_{target} is what we are interested in. The difference between those two quantities is plotted in Fig. 6.7 as a function of p_{mdc}. By fitting this dependency to a phenomenological model we can correct for energy loss our momentum measurements. The model is derived by direct inspection of the data:

$$p_{target} - p_{mdc} = p_0 + e^{p_1 + p_2 * p_{mdc}}$$

where p_0, p_1 and p_2 are the model parameters. Their values are on the figure.

In the previous procedure we have made the basic assumption that the energy loss correction does not depend on the direction of the particle leaving the interaction zone. This approximation is correct in first order since the RICH detector is, in first order, spherical.

[7]The target density (2.15 g/cm^3) times half the target width (0.5 cm) times the value of energy loss from figure 6.6.

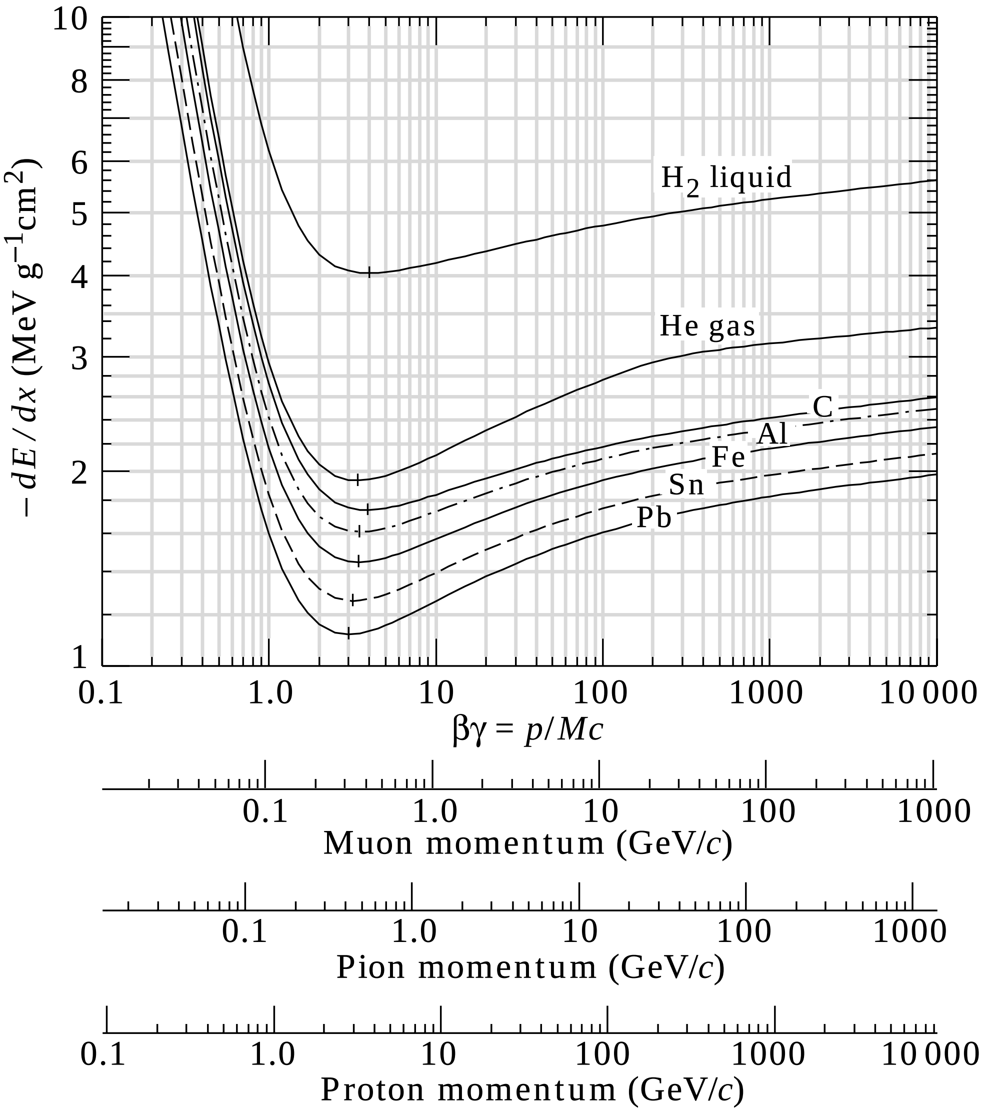


Figure 6.6: Energy loss in various materials. Taken from (1)

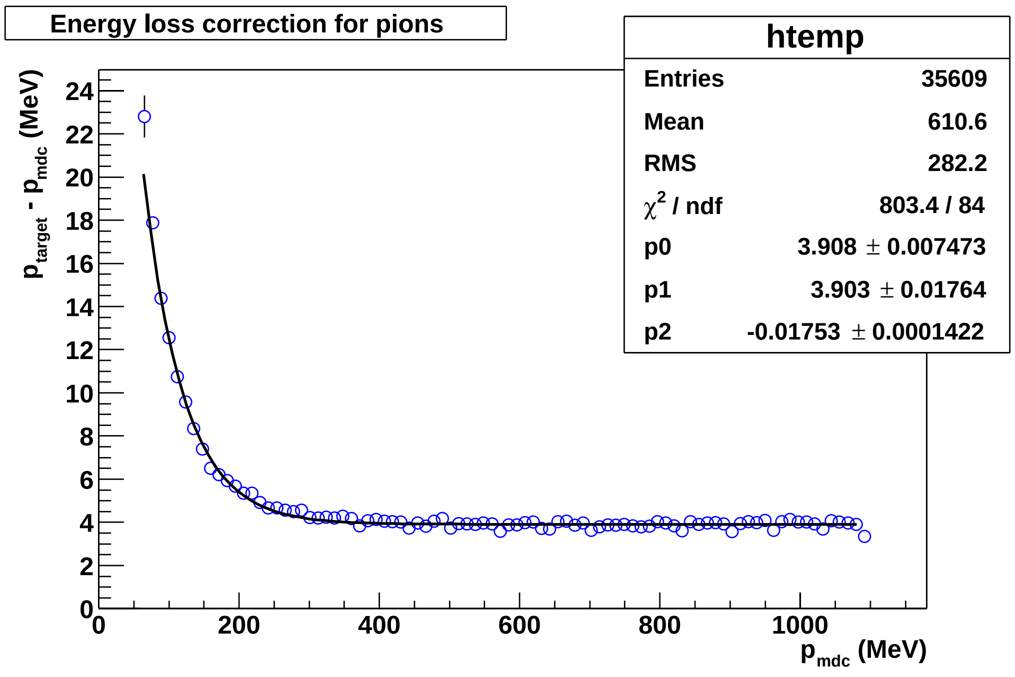


Figure 6.7: Variation in the track's momentum due to energy loss measured between the target point (p_{target} in the text) and the first MDC chamber (p_{mdc} in the text). Each point is the average of several hundred particles with the same p_{target}, error bars are purely statistical.

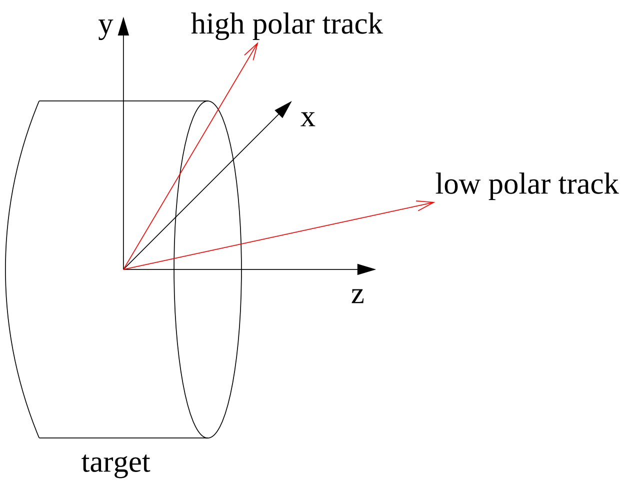


Figure 6.8: Variation of track length inside the target with polar angle. The higher the length, the higher the energy loss experimented by the particle.

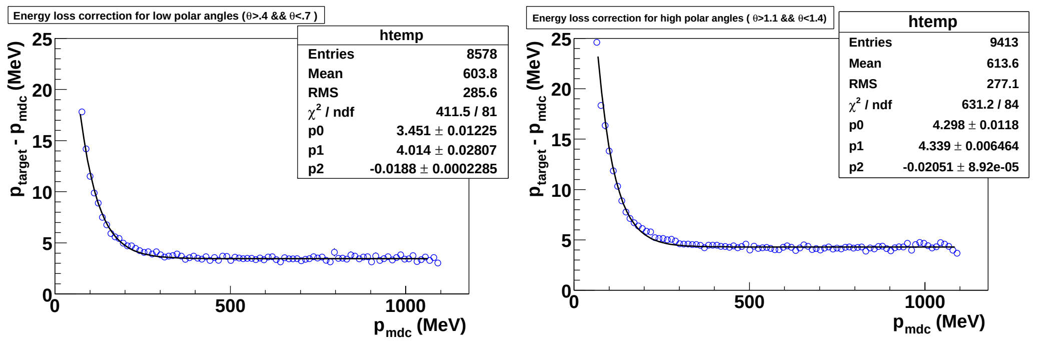


Figure 6.9: Energy loss correction for the lower (left) and higher (right) polar angles.

The biggest deviation is expected at large polar angles where not only the RICH deviates from the spherical shape, but also the path length of the particle inside the target is incremented. This is so because the target is not spherical, but cylindrical, with the symmetry axis in the beam direction. Even though the target length in the beam direction is 5mm its diameter is 8mm, thus increasing the length inside carbon for particles emitted at high polar angles, see Fig. 6.8. This additional correction can be achieved by simply dividing the phase space in bins and repeating the previous procedure in each of them; however such level of accuracy was not considered crucial for the current analysis. In order to get an idea of the effect, Fig. 6.9 shows the same kind of plot as before for two different regions in polar angle close to the limits in the acceptance. The relative difference is significant, but that is a correction on the energy loss, which is already a rather small correction.

6.5.3 Acceptance correction

One of the design goals of HADES is to achieve a large acceptance in order to maximize the detection efficiency for open lepton pairs. That is, lepton pairs coming, for example, from the decay of vector mesons. The reason is the low branching ratio of the two lepton channel in the vector meson decays which forces us to try maximizing the detection efficiency.

Actually, efficiency and acceptance are used in this work with different meanings. Efficiency itself can be divided in detection and reconstruction efficiencies. The first is detector specific and it is the probability for a particle crossing a detector active volume to be identified and it is a characteristic of the hardware. The second is the probability for a detected particle to be reconstructed and it is a characteristic on the tracking software[8].

Acceptance in the sense given here, could be explained as a geometrical efficiency. That is the probability for a particle emitted in the collision to cross the active volumes of the spectrometer's detectors. This includes the effect of the particle's deflection in the magnetic field which depends on momentum.

In order to make a simple estimation of acceptance as function of transverse momentum (p_t) and rapidity (y) we used a Monte Carlo method based on HGeant. Events with uniform distributions in p_t and y where generated and tracked with HGeant assuming a 100% detector efficiency. Then the reconstructed distributions are compared with the original ones.

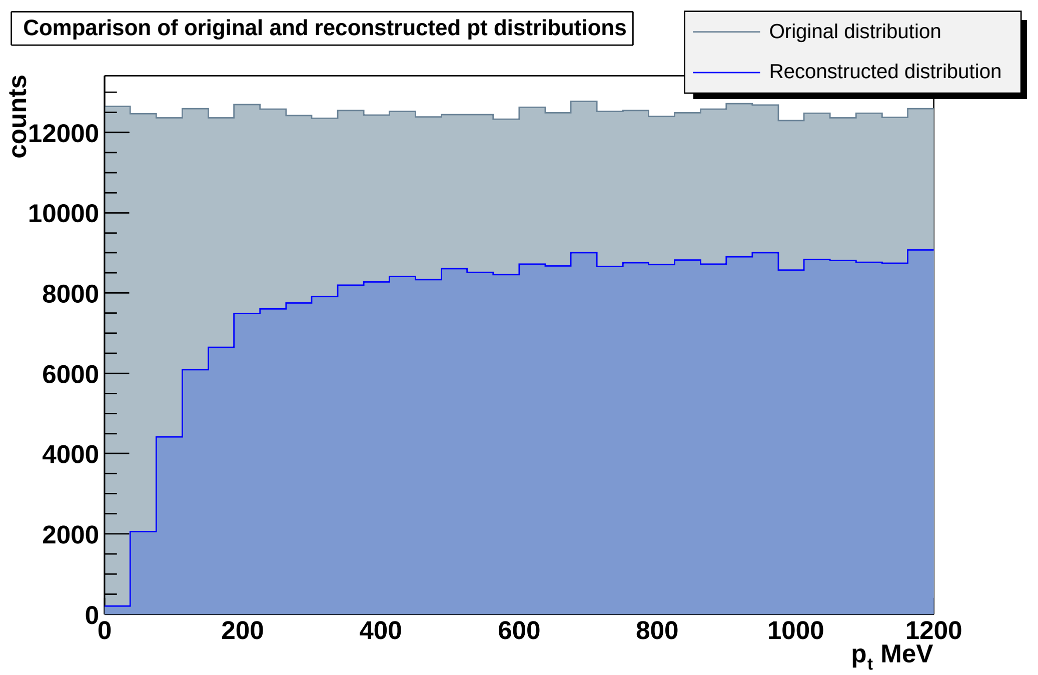
Fig. 6.10 shows the simulated and accepted p_t and rapidity distributions. The main losses happen at the sector edges where the magnet coils and detector frames are sitting. In all these plots a uniform distribution in the azimuthal angle is used. The rapidity distribution shows a feature for values around 1. The excess observed there is product of the overlap of the Tof and Shower detectors. Due to that overlap it is possible that one same track is reconstructed twice, once with the Tof and once with the Shower detector. To really avoid this effect instead work around it, we would need to check for compatibility of the reconstructed segment in the overlap region.

From the data above we can build two dimensional histograms for the reconstructed and original $p_t - y$ distribution. Dividing both 2D histograms we get an acceptance matrix. That is, a matrix telling us how probable it is for a particle with given transverse momentum and rapidity, to be accepted by Hades. The final matrix used throughout this chapter is depicted in Fig6.11.

By dividing the histograms obtained before and after the reconstruction process we get a matrix indicating what is the acceptance for each bin in transverse momentum and rapidity. Linear interpolation or other means can be used in order to smooth the data, even though that was not necessary for the current analysis. Actually, if linear interpolation had been used it would have not been a good idea to use uniform distributions in p_t and y. Instead, we should have to generate particles with very particular p_t and y values, those corresponding to the centers of the bins in case of symmetric interpolation.

More systematic approaches are under study within the Hades collaboration, but the results were not available by the time this analysis was

[8]In particular it is a characteristic of the cuts performed by the matching algorithms inherent to the tracking software



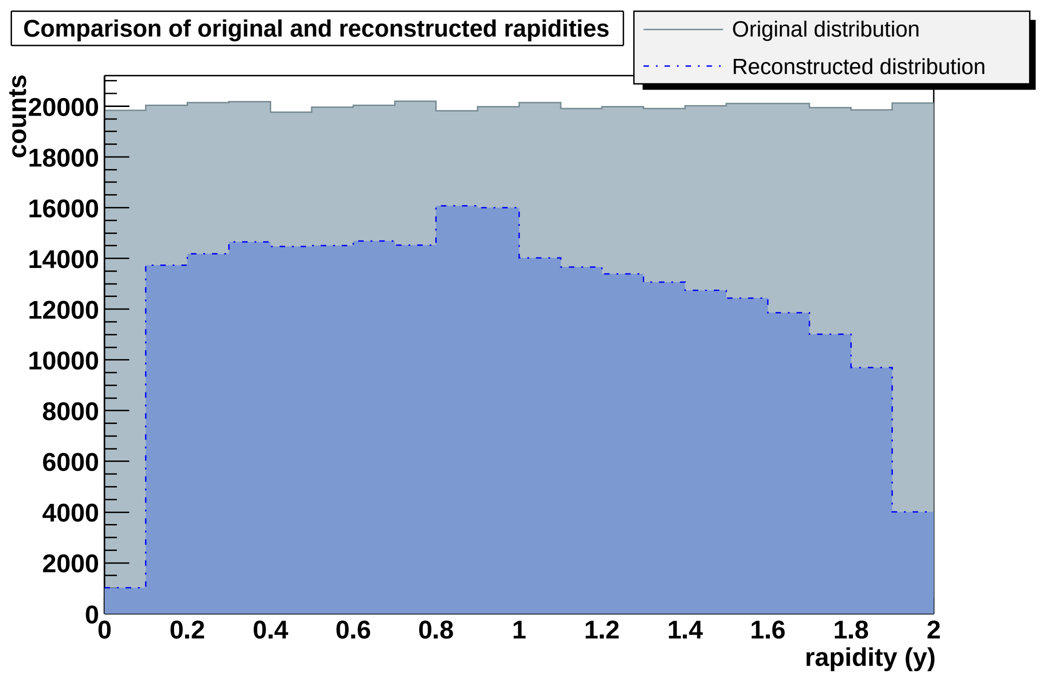


Figure 6.10: Comparison of uniform and accepted p_t (uppermost) and rapidity (lowermost) spectra. Main losses happen in the sector edges.

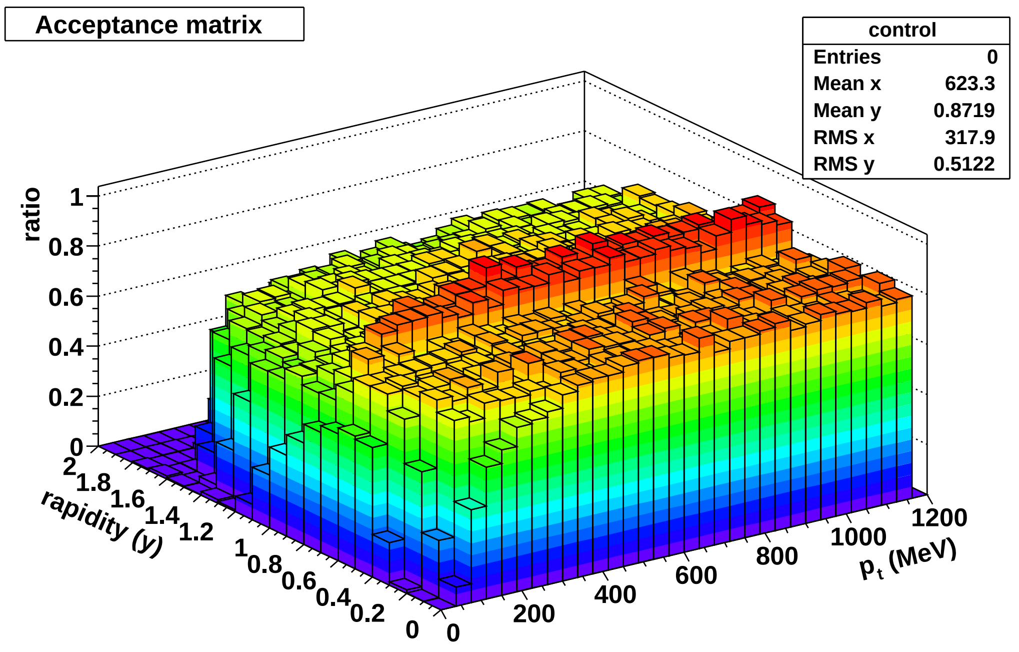


Figure 6.11: Acceptance matrix for negative pions as a function of transverse momentum and rapidity.

performed.

6.5.4 Efficiency determination and noise correction

As already mentioned efficiency has two components: detection and reconstruction efficiencies. Detection efficiency determination as defined in the previous section is too large a topic to be presented here. Suffice it to say that it is an ongoing effort inside the HADES collaboration. For the analysis presented in this chapter we are assuming that the detector efficiencies do not significantly depend on transverse momentum and rapidity.

It should be noted that the detectors used for this analysis are the ones required for the low and medium resolution setups. That is the MDC chambers, TOF detector and first chamber (pre-converter) of the SHOWER detector. The TOF detector, made of plastic scintillators, is assumed to have an efficiency close to 99%. MDC chambers and SHOWER pre-converter have also high efficiencies, so there is little room for variation, and big variations on efficiency on the active volume are not expected.

It could be argued that efficiency and noise level on those detectors depend on multiplicity and multiplicity depends on polar angle, being larger for the smaller polar angles. However, in C+C collisions the average multiplicity per event is low, around 4 charged particles, less than one particle per sector. Therefore there are good reasons to think that we are on the safe side assuming that efficiency variations along the polar angle are not dominant. After all, the granularity in these detectors is chosen to be able to cope with the heavier systems, like Au+Au, with around 20 particles per sector per event.

What remains to be accounted for is the reconstruction efficiency, more concretely the reconstruction efficiency of the low and medium resolution kick planes. This is something we cannot neglect since the efficiency and noise levels of the kick plane reconstruction process depend in particular on momentum. So, if we are to analyze the shape of a momentum spectrum it is necessary to correct for them. We will concentrate for the moment on the low resolution case, since it is simpler and the methods presented here can also be applied to the medium resolutions.

The method

It is known that in the low resolution kick plane, noise tends to accumulate at the lower momenta. This is natural and a consequence of the rater low multiplicity. A random combination of a segment from the inner chambers and a hit in META is most probably corresponding to a large deflection. In other words, the probability to find a random hit in META in the direction pointed by a segment is lower than the probability to find it somewhere else. A large deflection translates into a low momentum, thus the noise accumulation in the lower end of the momentum spectrum.

The matching algorithm for the low resolution setup tries to minimize this effect by placing cuts on the data, as explained in section 5.5.1. In that section the correlation between the polar and azimuthal angles was used in order to build a pull variable, $xPull$, to be used for matching purposes. That is, $xPull$ was computed for all track candidates and the decision of whether a candidate is a real particle or not was taken by comparing its $xPull$ with a certain cut value. Still the effect persists and therefore we need an estimation of the matching algorithm's efficiency and noise levels as a function of momentum in order to correct the transverse momentum spectrum presented later in this chapter.

The physical interpretation of $xPull$ for a track candidate is essentially the difference between the x coordinate measured in the META detector and the prediction for that same quantity given by the kick plane model. This is then divided by the corresponding error in order to obtain a normalized distribution.

In section 5.5.1 it was shown how the efficiency and noise level change for different cuts on the *xPull* variable, but those results were based on simulations. We would like to get the same kind of information directly from the data, where the situation can be significantly different because the noise environment is larger; and we would like to get it for different momentum bins. This can be done by analyzing the shape of the *xPull* distribution.

What is measured in real life is the sum of the *xPull* distributions for real tracks and fake candidates and there is no way to separate them a priori. However it is possible to come up with a model for the compound distribution which contains a noise part and a signal part. Then, a fit can be performed to the data, giving back parameters which completely define the noise and signal components of the distribution and thus also the efficiency and noise level. In mathematical terms, lets call $f_g \equiv f_g(xPull)$ the *xPull* distribution for real tracks and $f_b \equiv f_b(xPull)$ to the corresponding distribution for fakes. The observed distribution is then $f = f_g + f_b$, defined for $xPull \in (-\infty, \infty)$.

If we place a cut on *xPull* such as $|xPull| < c$ the cut's efficiency is given by,

$$\epsilon = \frac{\int_{-c}^{+c} f_g}{\int_{-\infty}^{+\infty} f_g} \tag{6.3}$$

while the noise level is given by:

$$nl = \frac{\int_{-c}^{+c} f_b}{\int_{-c}^{+c} f_g + f_b} \tag{6.4}$$

These are the only two magnitudes we need in order to know the real number of good tracks, N_g, from the number of reconstructed tracks N_c[9]. The correction factor can be computed as:

$$W = \frac{N_g}{N_c} = \frac{1 - nl}{\epsilon} \tag{6.5}$$

The method described above requires integrating f_g between $-\infty$ and ∞. Since this is an analysis which is realized over the output of the reconstruction program, maintaining that integration interval would require storing in the output all possible track candidates whatever their *xPull* may be; this is

[9]That is, the number of tracks passing the cut

anyhow done. It would also require to come up with a model for f_b which is valid in the whole range; that is more difficult and not really needed. In fact f_g tends to zero when $|xPull| \to \infty$, thus it is enough to chose an integration interval large enough.

In the present analysis the full integration interval was chosen to be $xPull \in (-20, 20)$. That provides room enough for the f_g distribution to be contained in[10] and it also provides a region (large $xPull$) where f is completely dominated by f_b; this is important for the fitting algorithm to correctly estimate f_b.

In order to apply this method we need models for the f_b and f_g distributions. In section 5.35 the distribution for real tracks and fake candidates was shown from simulations. In particular it was shown how f_g is not purely gaussian, most notably in the case of candidates formed with the Shower detector. The fakes distribution, f_b, was shown to be rather flat and actually it can be estimated in real life by a phenomenological model derived from the data itself as we will see. The f_g models for the Tof and Shower detectors are presented in the following sections.

xPull distribution in the Tof detector

For a track candidate hitting the Tof detector at a point $\mathbf{r_m} = (x_m, y_m, z_m)$ as measured by the detector itself, $xPull$ is defined as

$$xPull = \frac{x_c - x_m}{\sigma_{x_c - x_m}} \tag{6.6}$$

where x_c is the expected x coordinate of the intersection point of the track with the Tof detector, as obtained from the kick plane model. For more details about this definition section 5.5.1 can be consulted. Here the interesting thing is that we can assume x_c to be gaussian as it is affected by many small error sources. The behavior of x_m responds to the way it was measured and it can also be considered gaussian (see section 5.4.1 for details). There it is shown that x_m is given by

$$x_m = \frac{t_2 - t_1}{v_g}$$

where v_g is a the light speed in a plastic rod and both t_1 and t_2 are time

[10]In section 5.5.1 we saw how the efficiency was close to 100% for $xPull$ already in the order of 10. The situation in real data may be different, but not dramatically different, since $xPull$ is normalized.

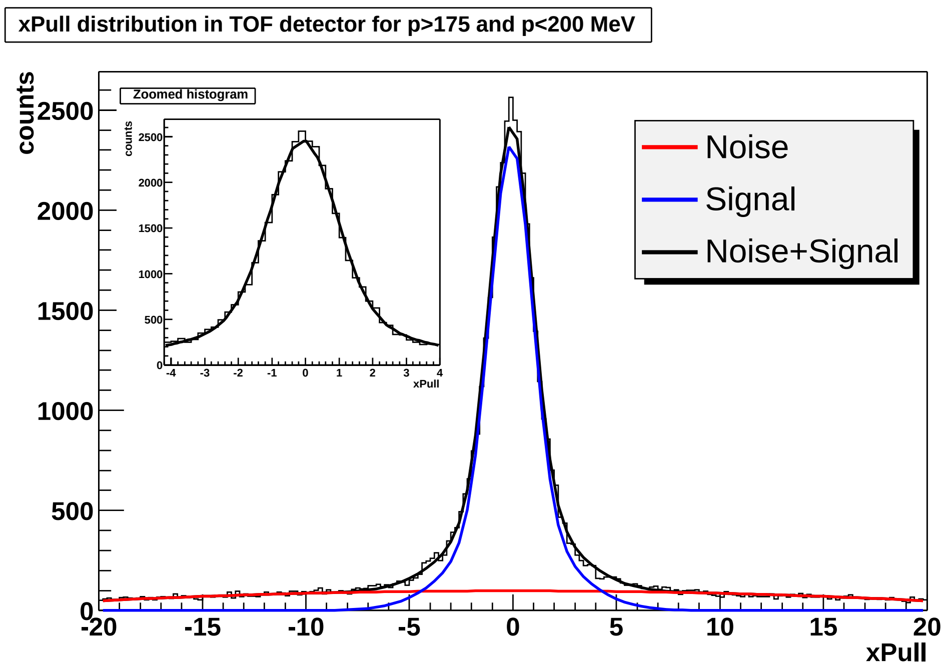


Figure 6.12: $xPull$ distribution in the Tof detector for a transverse momentum ranging from 175 to 200 MeV. The black line represents a fit to the full model, the blue and red lines correspond to the signal and noise components respectively. The plot is integrated over all scintillator rods.

measurements with gaussian errors of the same magnitude; thus x_m can be considered to be gaussian.

If both x_c and x_m are normally distributed then $xPull$, being the difference between them, should also be normally distributed. In practice the model for f_g is the sum of two gaussians which account for the approximations in the previous argumentation and in the computation of $\sigma(x_c - x_m)$. Fig. 6.12 shows this model applied to a Nov01 data set.

$xPull$ distribution in the Shower detector

The case of the Shower detector is slightly more complicated. It is still possible to maintain the gaussian hypothesis for x_c, but not for x_m. This comes from the way in which x_m is measured. As already explained, the Shower detector is subdivided in approximately square pads. There is no position measurement inside each pad but, whenever a particle crosses the Shower detector, we know what pad was hit by the track. x_m is then estimated as the x coordinate of the pad center. However the point in the pad which was hit is not known: it could be any of them with the same probability. In other words a uniform probability distribution is associated to the measurement.

The typical pad size in the relevant direction is around 3cm, that is too large to approximate the uniform distribution by a gaussian one and it is small enough that we do not need to consider deviations from the uniform distribution.

If x_c follows a gaussian distribution and x_m follows a uniform distribution f_{uni}, with

$$f_{uni} = \begin{cases} k & |xPull| < L \\ 0 & otherwise \end{cases}$$

then, the distribution of $xPull$ is the convolution of them (see (2)), that is

$$f_g(xPull) = \int_{-\infty}^{+\infty} f_{gauss}(u) f_{uni}(xPull - u) du$$

Fortunately that is a known problem with known solution. For a gaussian distribution centered around 0 with variance σ, the integral yields (see (2)):

$$f_g = \frac{\Psi\left(\frac{u+L}{\sigma}\right) - \Psi\left(\frac{u-L}{\sigma}\right)}{2L}$$

where

$$\Psi = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{xPull} e^{-\frac{t^2}{2}} dt = 1 - erf(xPull)$$

and where erf is the so called "error function" which is tabulated and readily available in different numerical libraries.

Fig. 6.13 shows this procedure at work on a set of data from NOV01. In this case the model is slightly more sophisticated since the signal is approximated by the convolution of two gaussian with the uniform distribution. This is needed in order to better reproduce the non gaussian tails.

Checks on simulations

A way to check the goodness of the previously sketched method is to apply it to simulated data, where the results for efficiency and noise level can be cross checked with the results from the simulation itself.

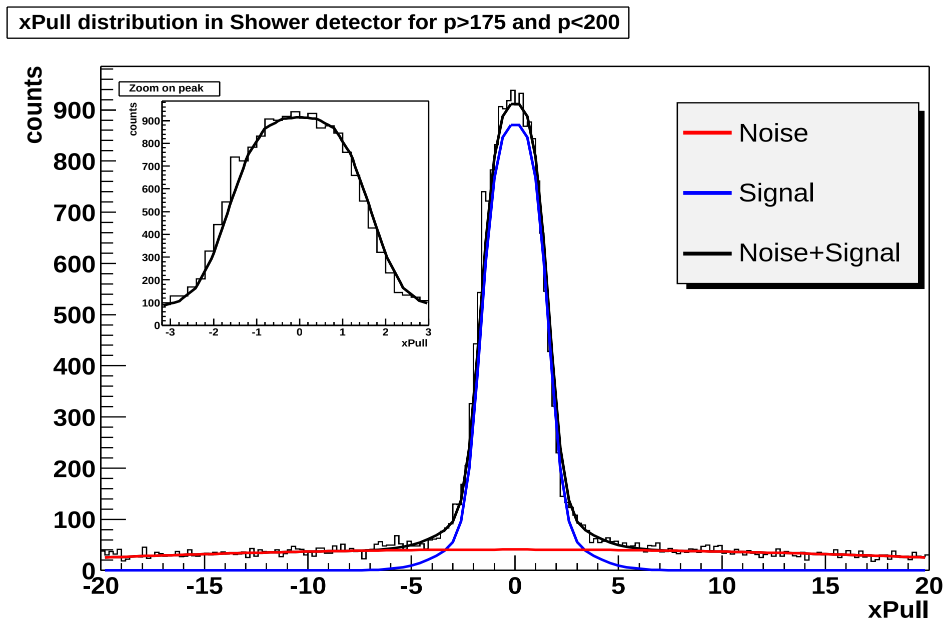


Figure 6.13: *xPull* distribution in the Shower detector for a transverse momentum ranging from 175 to 200 MeV. The black line represents a fit to the full model and the blue and red lines correspond to the signal and noise components respectively.

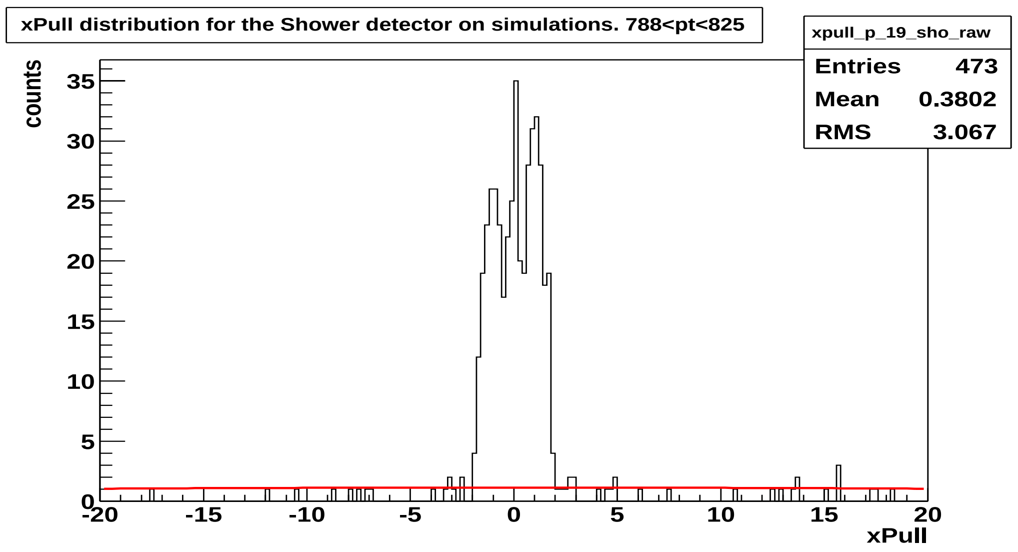


Figure 6.14: *xPull* distribution in the SHOWER detector for transverse momentum from 788 to 825 MeV. The red line is the fitted noise distribution. The figure shows how noise is overestimated due to the few statistics for large *xPull*. The larger momenta are shown here since that is the region were the effect is more significant.

It turns out that a direct comparison is not feasible since the fits involved in the present calculation do not work properly on simulated data: noise levels are systematically overestimated. The reason is that, in order for the fit to be able to properly estimate the noise, we need significant statistics in the spectrum region which is noise dominated (large *xPull*). That is not verified in the case of simulations for two reasons:

- The overall statistics, number of analyzed events, is smaller than in real data.
- The of noise to signal ratio is lower than for real data. So, actually, we would need more simulated events than real ones.

Those two circumstances are avoidable in real data, since statistics is higher and noise levels are also higher. Furthermore we can check for this problem by directly looking to the fits. Still, a number of things are to be learnt by comparing to simulations.

We have said that for the method to work it is enough to do the fit for the *xPull* interval from -20 to +20. It was said that such interval would be large enough, but nothing was said about how good such an approximation would be. This is something we can check on simulations. The restriction to a finite interval does not affect the computed noise level (see Eq. 6.4), but the efficiency can be affected if f_g is different from zero outside the limits (see Eq. 6.3). In order to see how good the approximation is we need to estimate the number of real tracks with an $xPull$ larger than 20. In other words, we need the efficiency for a cut $|xPull| < 20$.

That efficiency can be estimated on simulations and plotted as a function of transverse momentum. That is what is shown in Fig. 6.15, where efficiency is plotted both integrated over the full rapidity range and for mid rapidity. In our case, around mid rapidity, for most of the momentum range the effect is small enough to ignore it, besides the curve is flat. However, there is a clear trend toward the lower transverse momenta.

What happens is the following: in order to compute x_c for a track candidate, the kick plane algorithm uses the candidate's intersection with the kick surface. Sometimes the intersection points falls outside the acceptance, where the kick plane parameters are not known. In those cases the value of $xPull$ is artificially set to 1000, thus falling outside our window. Most of the track candidates showing this problem are fakes and thus a high $xPull$ makes sense. However, it is also possible that real particles show this behavior in which case they are lost. The reason why that happens more for the lower momenta is that in such regime we start to have tracks which curl in the magnet field and are not well described by the kick plane model.

This knowledge allows to introduce a correction to the correction factor, W, being computed (Eq. 6.5). If we denote as ϵ_{20} the efficiency for a $xPull$ cut of 20. Then, the corrected factor is

$$W' = \frac{W}{1 - \epsilon_{20}}$$

It is also educative to look directly to the $xPull$ distribution for real tracks. This was already done in previous chapters, but it was assumed that both of the inner chambers were available. The situation is different in NOV01 for one of the sectors, where only one inner chamber is functional. What is seen is that for that sector $xPull$ presents very large tails. That invalidates the whole method. We need separate regions dominated by signal and noise, otherwise the fitting algorithm cannot distinguish between both components.

For the same reason the so called "correlated noise" (see section 5.5.1) cannot be distinguished from the signal. An estimation based on simulations is needed.

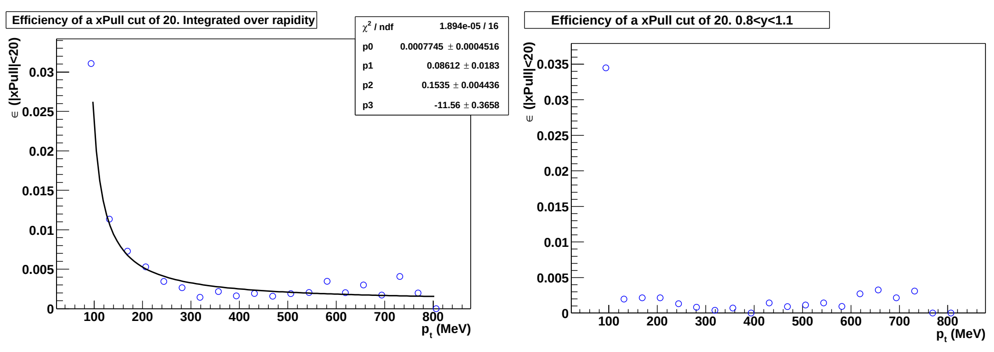


Figure 6.15: Efficiency for a cut in $xPull$ of 20. The leftmost figure is integrated over the full rapidity range while the rightmost corresponds to rapidities between 0.8 and 1.1. The leftmost figure is fitted to an phenomenological model $\epsilon = P_0 + \frac{P_1}{P_2 p_t + P_3}$. The increase toward the lower momenta is an edge effect (see text).

6.5.5 Results on $^{12}C + ^{12}C$ simulation at 2.0AGeV

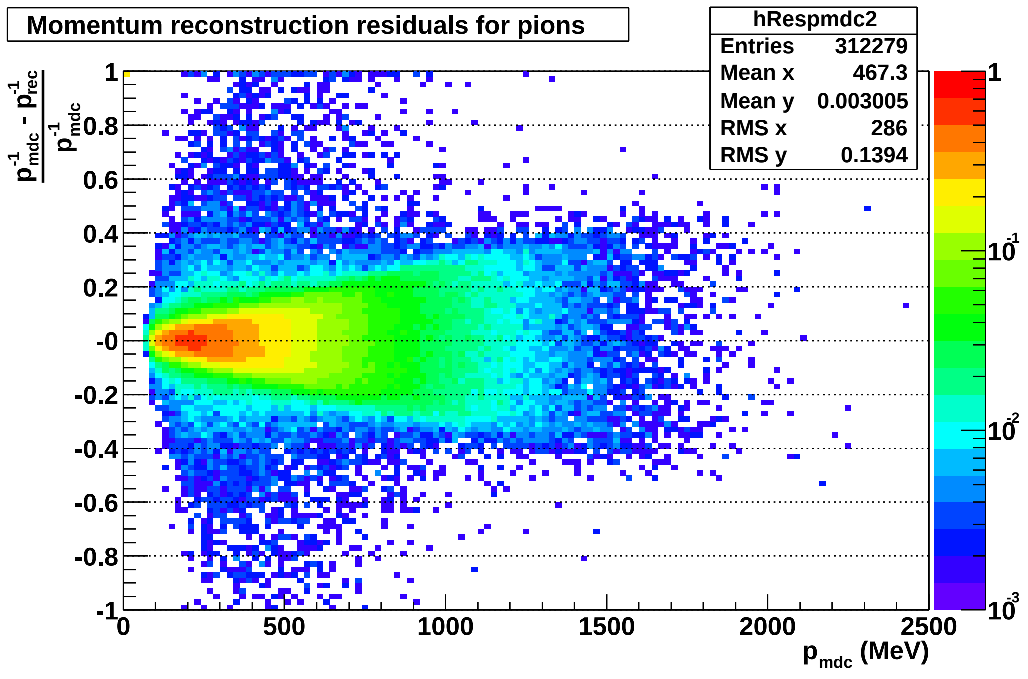
In simulation we have the advantage that the original momentum is known. We also have perfect particle identification and fake rejection capabilities, given by HGeant. Using those capabilities it is possible to see what kind of spectra are expected. We can run the simulation data on the same analysis chain as the real ones, thus giving us an idea of how that analysis chain modifies the results. That way it becomes easier to interpret the results on real data

First we should check if there are systematic errors in the momentum reconstruction for the simulations. As shown in Fig. 6.16 the only deviation comes from the energy loss effect. Fig. 6.17 shows how the correction from section 6.5.2 solves the problem.

As to how well the transverse momentum spectrum is reconstructed, Fig. 6.18 shows how the spectrum looks like from the event generator, that is the simulation input, and what the result is after it goes through the whole reconstruction program and the corrections in this chapter are applied. We see how the main features are retained while the T parameter is increased by nearly 3 MeV.

6.5.6 Real data fit to a thermal model

The resulting transverse momentum spectrum of real data is shown in Fig. 6.19 with fits to equation 6.2 in two different momentum ranges. In par-



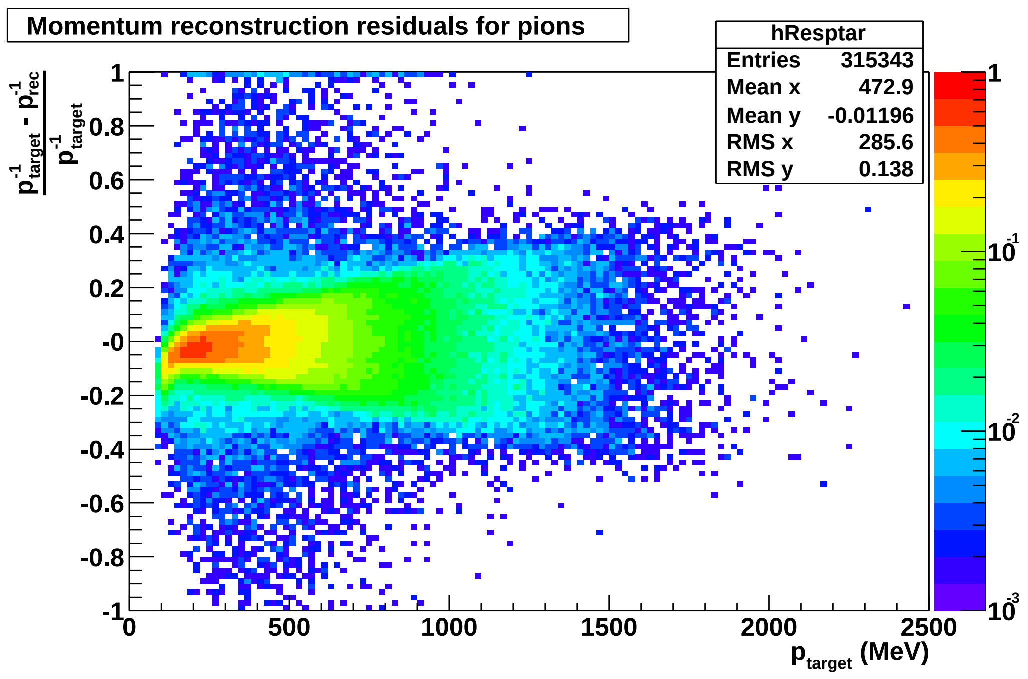


Figure 6.16: Residuals in the reconstructed momentum. On the left side the reconstructed momentum is compared with the particle momentum when entering the field. On the right side it is compared to the momentum when leaving the target. The difference is due to energy loss.

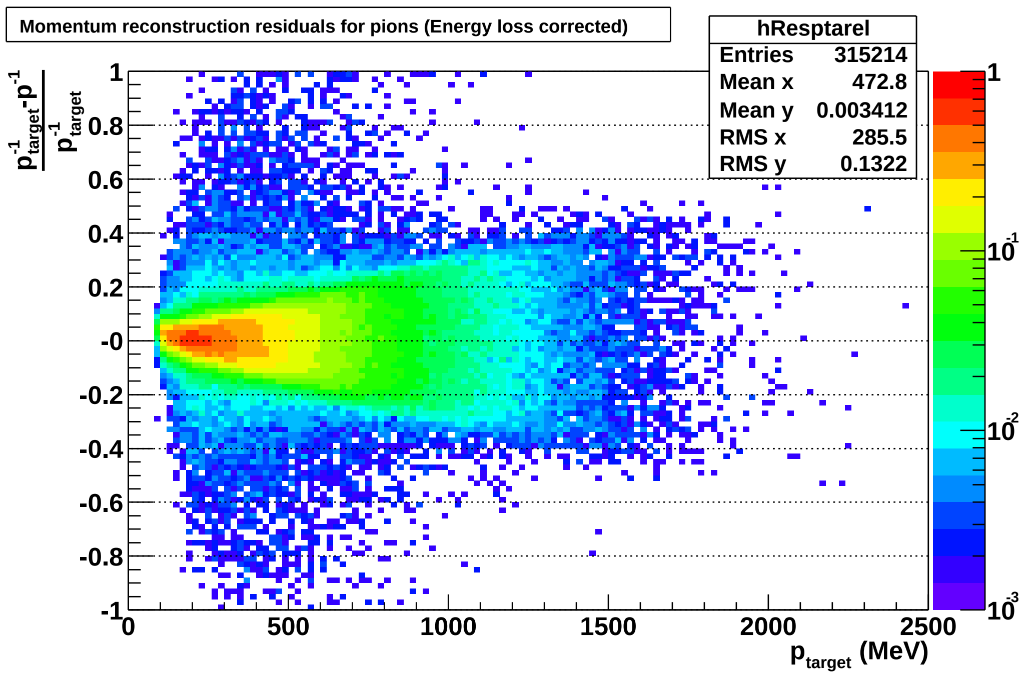
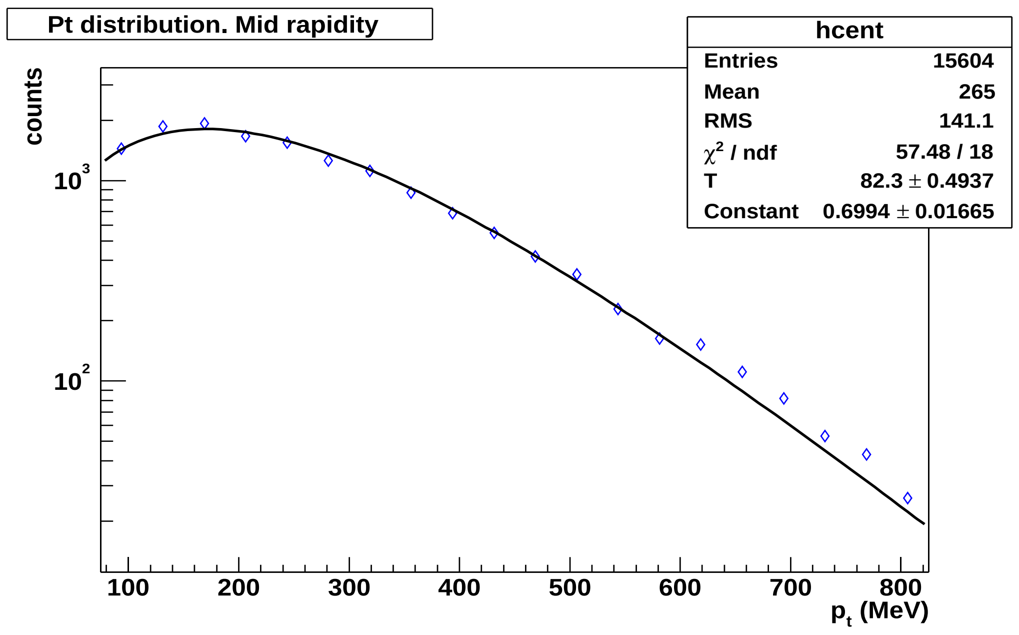
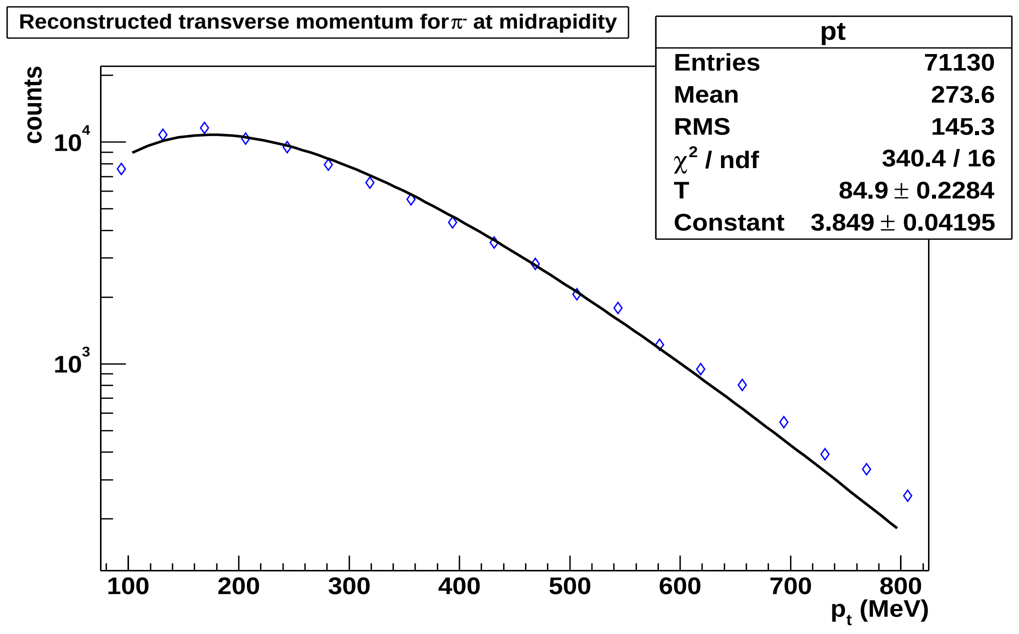


Figure 6.17: Residuals in momentum. The reconstructed momentum is energy loss corrected before comparing it to the particle momentum at the target.



(a) Ideal distribution from the event generator (BUU)



(b) Reconstructed distribution

Figure 6.18: Transverse momentum spectrum around mid-rapidity ($0.8 < y < 1.1$). The number of processed events is different in each figure.

ticular, for comparison with the data on table 6.4 it is needed to fit on the same momentum range as measured by the KaoS experiment. From Ref. (6) this range was estimated to be between 200 and 600 MeV at mid-rapidity. In that range the two temperatures obtained by us are compatible with the ones measured by KaoS. The fit to the full range shows that there is a deviation from the model both at the lowest and the highest momenta.

Deviations from the thermal model

For the deviation at the highest momenta the explanation is that the so called correlated noise (see 5.5.1) is larger in that region. In that section it was shown that a π^- could decay in flight between the inner MDCs and META through the channel $\pi^- \to \mu + \nu_\mu$. The muon inherits the direction of the pion because of the Lorentz boost and thus the hit it produces in META is mismatched with the hits produced by the pion in the MDCs. The effect is larger the larger the momentum, because the Lorentz boost is more significant.

In the lower momenta range there are two effects. On one side that p_t is the closer to the spectrometer acceptance limits. In the present analysis the ideal positions of the detector has been used to build the acceptance matrix, however it is known that the detectors are not exactly in their ideal positions. Then the acceptance matrix may be slightly different in the edges. Besides that, it has already been shown that the noise level steeply increases toward the lower momenta. In the method presented above the noise level is estimated in each bin as an average, that approximation is worse the larger the slope of the curve. In other words, the low momentum region would benefit from smaller binning, which requires higher statistics.

Comparison with previous data

As already mentioned, the two inverse slope parameters depend on the measured momentum range. If we are to compare with the data from table 6.4 we need to know what was the momentum range. The data in the table for C+C for 2.0 AGeV was taken by the KaoS collaboration. From Ref. (6) the accepted transverse momentum range around mid-rapidity has been estimated to go from 200 to 600 MeV. Those are the limits in the fit shown in the second part of Fig. 6.19, yielding the “temperatures”:

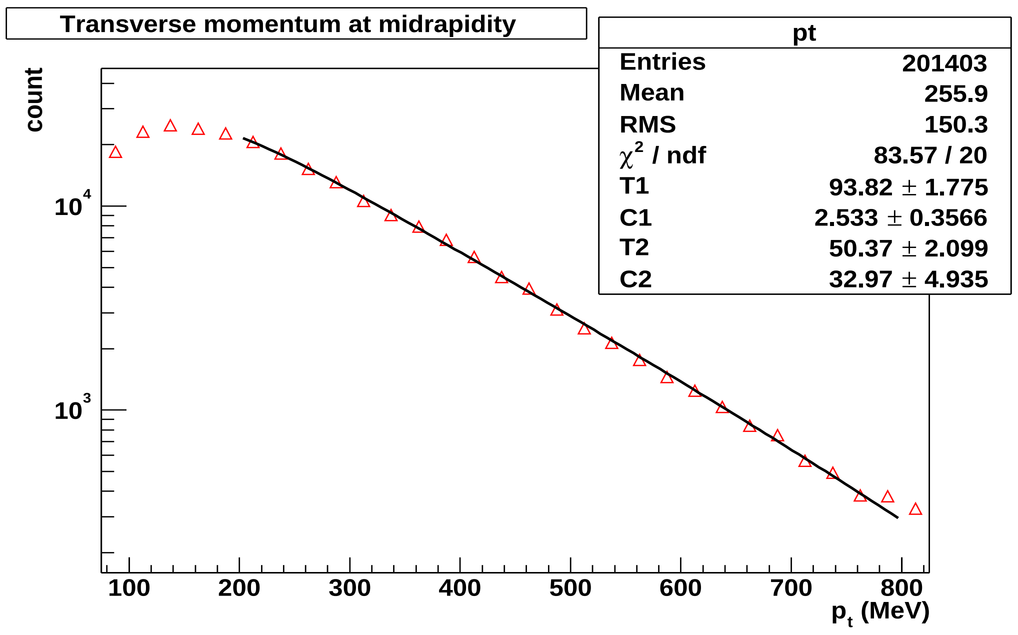
$$T_1 = 41 \pm 3 MeV, \;\; T_2 = 86 \pm 2$$

which are compatible, within the errors, with the measurements from KaoS. The errors are those from the fitting procedure.

6.6 Medium resolution data

As already mentioned one of the sectors had three active MDC chambers during the NOV01 data taking period. In this section we will focus on the improvement in resolution due to that fact. This resolution increase is illustrated in Fig. 6.20 where p is plotted versus β showing how the particle identification capabilities have improved.

On more quantitative terms, we can compare the resolution in the reconstructed proton inverse mass for a set of tracks using the low resolution and medium resolution kick plane algorithms. The results are presented in Fig. 6.21.



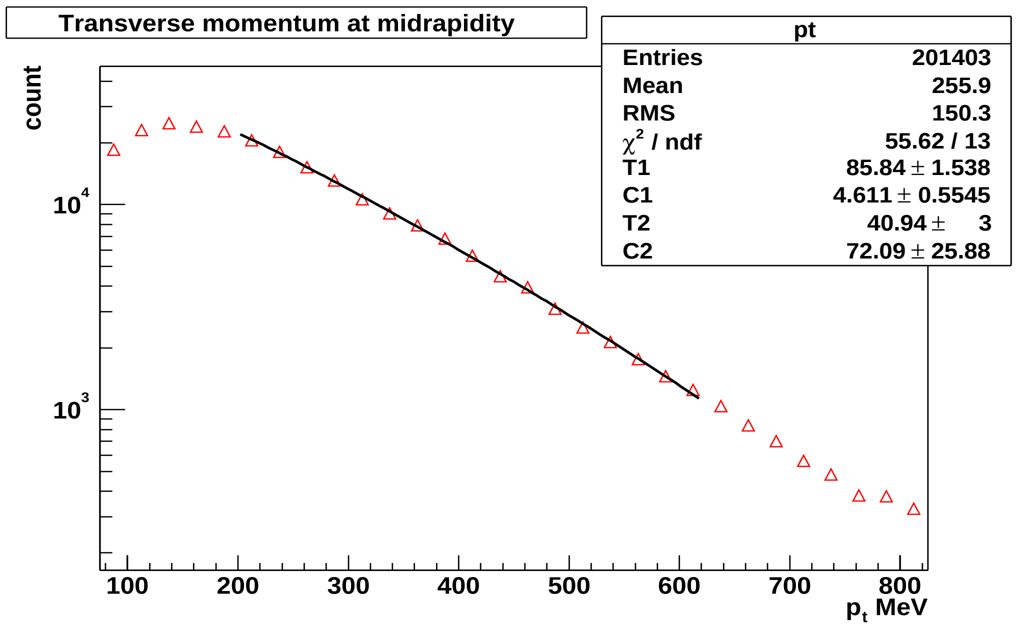


Figure 6.19: Transverse momentum spectrum of pions produced in C+C at 1.9 AGeV. The enhancement toward high p_t can be attributed to "correlated noise". At the lowest momenta the noise level increases sharply making the noise estimation inaccurate. The lower figure shows the same data as the upper, but he fitted region corresponds to the measurement range of the KaoS experiment for comparison with table 6.4

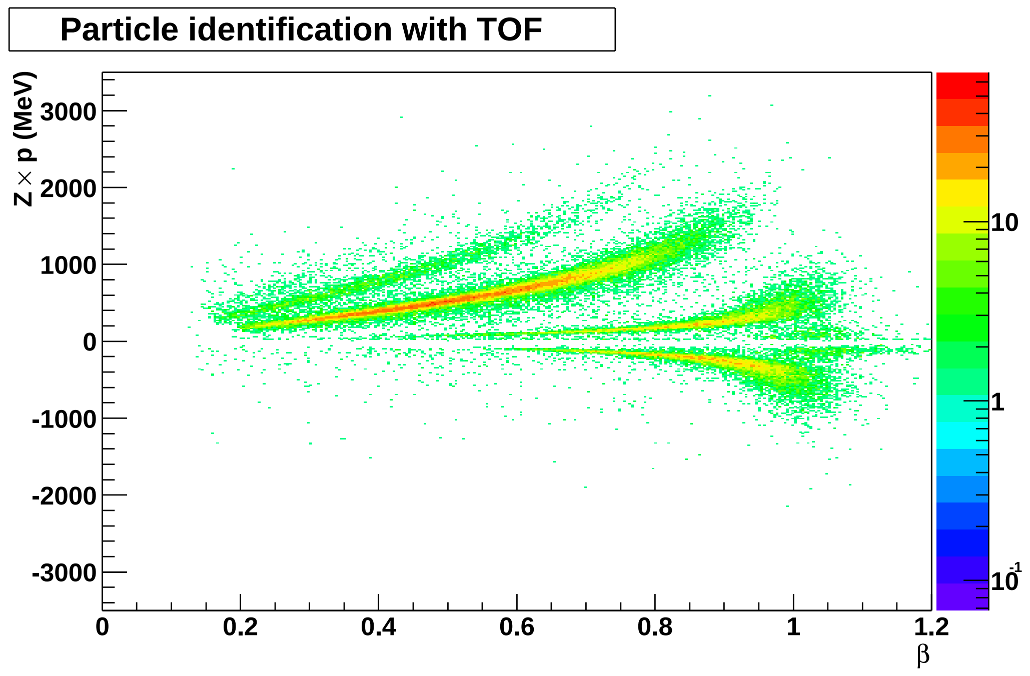
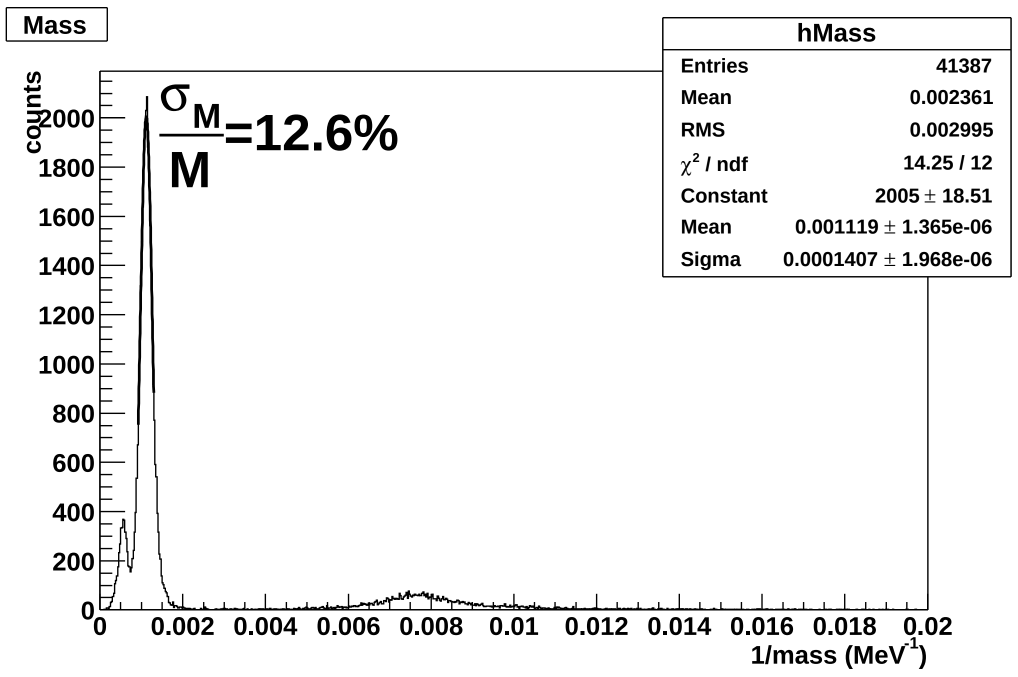


Figure 6.20: Particle identification using the TOF detector and the medium resolution setup (with MDC3). This figure is to be compared with Fig. 6.2 which corresponds to the low resolution setup (no outer MDC). The improved resolution allows the clear distinction between protons and deuterons. Tritium starts to be apparent also.



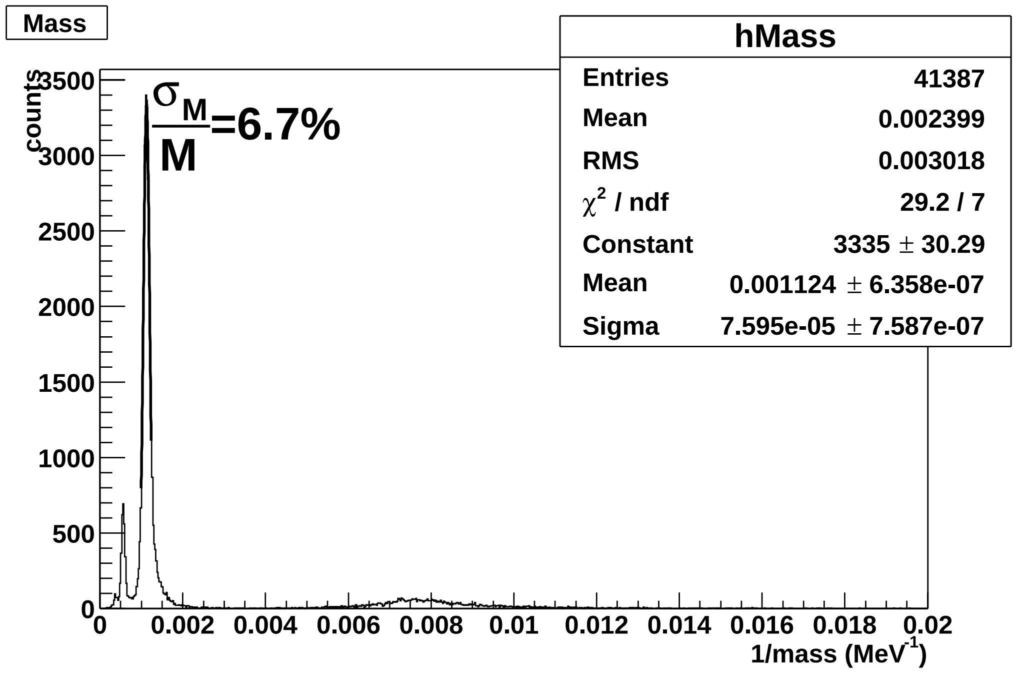


Figure 6.21: Comparison of mass resolution achieved with the low and medium resolution setups. The upper figure corresponds to the low resolution, the lower one is for medium resolution.

Bibliography

[1] Groom et al, Particle Data Group. *Review of Particle Physics.* European Physics Journal C(15):1-878, 2000.

[2] R.K. Bock and W.K Rischer. *The data analysis briefbook.* Springer,

[3] R.Stock. Physics Reports 135:259, 1986

[4] R. Brockmann et al. Physics review Letters 53:2012, 1984

[5] M. Ardid. *Particle Production at the GSI-Darmstadt Pion Beam Facility.* PhD thesis from Universitat de Valencia, 2002

[6] F. Laue et al. (KaoS Collaboration). *Production of Charged Pions, Kaons and Antikaons in Relativistic C+C and C+Au Collisions.* European Physics Journal A(9):397-410, 2000

[7] R. Averbeck et al. *Production of π^0 and η mesons in carbon-induced relativistic heavy-ion collisions.* Zeitschrift für Physik A(359): 65-73, 1997

[8] A. Förster. *Pionenproduktion und thermische Konzepte in relativistischen Schwerionenreaktionen.* Diploma thesis, Technische Universität Darmstadt, 1998

Chapter 7

Conclusions

In this work, momentum reconstruction algorithms have been implemented and applied to the analysis of pion production data.

The software framework where the different algorithms exist has been presented. The choice of an object oriented approach is justified by the reduction in code and management complexity that it affords. The user requirements for the reconstruction software have been analyzed and a design satisfying those requirements has been presented.

A vertex reconstruction algorithm has been proposed for the HADES experiment, in particular, it addresses the problem of outliers. The limitations on the vertex resolution imposed by the spectrometer design and the effect of multiple scattering have been explained as well as the sources for systematic errors. Finally, the performance of the method applied to events from the NOV01 data set has been shown.

Two algorithms have been presented for momentum reconstruction: the Kick Plane approach, that is less memory intensive, and the Reference Trajectories method. Their introduction is justified by the need for data analysis through the different stages of the spectrometer construction. Adaptations of the Kick Plane approach have been presented for the different possible setups. Their relative resolutions have been shown to range from the 3% to the 12% on the low resolution setup. It has been argued that even such a low resolution is comparable to that of the precursor experiment DLS.

The Reference Trajectories algorithm has been implemented for the complete setup, where the redundancy of the experimental system favours that kind of approach. A suggestion to improve the Kick Plane resolution in the complete setup using that redundant information has been made. The resolution achieved in this case is around 1% in simulations, matching the design goals of the HADES experiments.

Besides the methods for momentum reconstruction and track fitting, two methods have been implemented for the merging of track pieces before and after the magnet. Like before, both methods address different realities in the experiment. One method has been presented for the case in which no outer MDC chambers were available. In that case, the matching is performed comparing the track azimuthal deflection along the field with the prediction of the Kick Plane model. The addition of the rear MDC chambers opens up the possibility of using more information for the matching. That has lead to the implementation of a specific matching algorithm for that case. Both methods efficiencies and noise levels have been reported.

Finally, the momentum reconstruction methods have been applied to the analysis of the NOV01 data, focusing on the study of properties relative to pion production. The production ratios between the positive and negative charged pions have been shown to be in agreement with expectations. Agreement is also achieved for pion to proton ratio. These measurements exercise the particle identification capabilities of HADES applied to hadrons. In an attempt to check for possible systematic errors in the momentum reconstruction, the pion mass was derived from the data and found to be in good agreement with the known value. On a later stage, efficiency, acceptance and energy loss corrections have been derived from the data in order to obtain an spectrum of pion production as a function of transverse momentum. The methods developed to implement the corrections have been presented and the resulting spectrum compared to previous measurements from the KaoS collaboration. The spectrum has been fitted to a double Boltzmann distribution, with two “temperature” parameters, finding a good agreement with measurements from KaoS.

Appendix A

UML notation

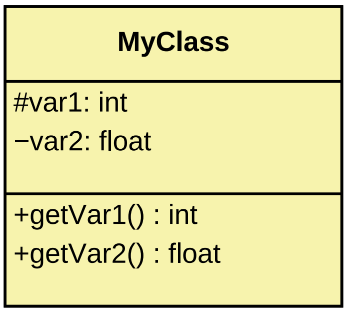
UML notation is a graphical representation for the modelling of software projects based on an Object Oriented Design. UML stands for "Unified Modelling Language", which makes reference to the fact that it is a language and not only a notation. UML tries to present the objects in a system and their relationships through graphical diagrams in a standardized way. The ideal is to deliver for software development what circuit diagrams are for electronics development.

The notation defines several types of diagrams, showing different aspects of a given software piece. From the static class structure to the physical code distribution going through use cases or time diagrams showing the interactions between objects necessary to accomplish a given task.

The full description of the notation is too large a topic to be covered here (see (1)), instead we will concentrate on those aspects of the notation which are necessary to understand the diagrams presented in this document. In particular we will only treat the most common constructions with class diagrams

A diagram is essentially a 2D structure formed by icons, lines (or arrows) and text. The most common elements populating a class diagram are:

Class A rectangle like

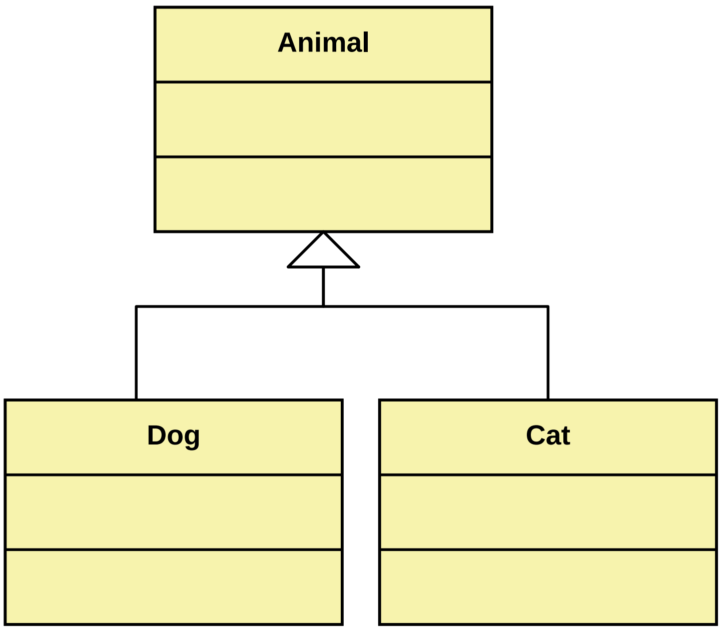


represents a class with name "Klass". The rectangle representing the class is divided in three parts, in the uppermost the class name is shown, the second part contains the class data members. Each data member is represented by a string with the syntax "name: type" identifying the variable name and its type[1]. The string is preceded by a symbol indicating the type of access to the variable: '+' if the member is public, '#' if it is protected and '-' if it is private[2].
The third part is another list of strings with the class methods. Those methods define the class interface with the outside. Each method is represented by a string in the form "name(argument): return_type"; where "name" is the method's name, "return_type" is the type returned by the function and "arguments" is a list of variable strings like those used to list the data members. Methods are also qualified for its visibility in the same way as data members.

Inheritance Is represented by an arrow finished by a hollow triangle. The arrow goes from the daughter class to the parent one, like in the following figure

[1]In the Pascal fashion

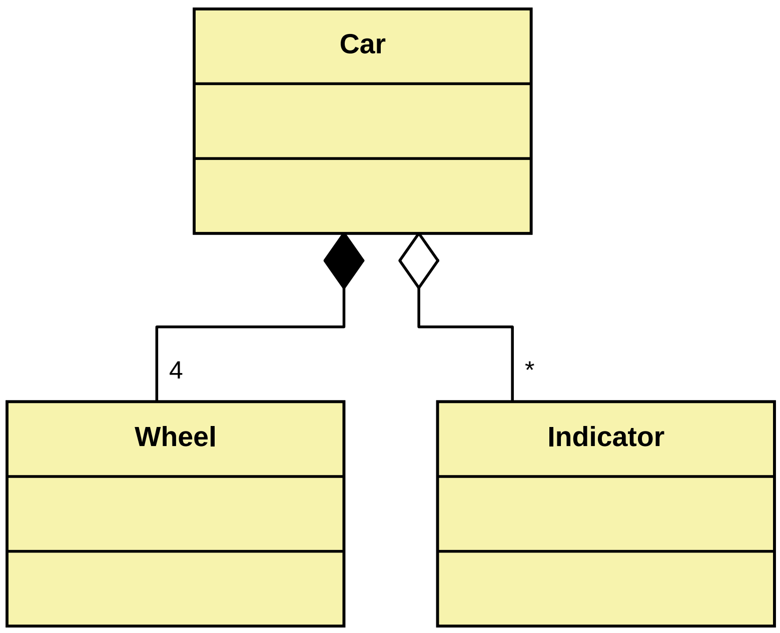
[2]A public data member is accessible from anywhere outside the class, a protected one is only accessible for the class itself and its daugthers. While a private one is only accessible to the class itself and friends



where the class “dog” is derived from the “animal” class.

Association Is represented by a solid line between the two related classes and it expresses a generic relationship between them. If an arrow is shown it indicates the direction of the association. This construct is typically used to express the case where a class has a poiner to another one, but without the second being a part of the first one conceptually. If the line is dashed it expresses a usage relationship, i.e. a class uses the services of another. In any of the extremes the multiplicity or the relation name can be indicated

Aggregation It is a kind of relation where a class is part of the other. This is expressed by a solid line joining the two classes, in the extreme of the container class a diamond is placed. The diamond can be solid or hollow, in the second case indicating composition by reference and composition by value in the first case. The following figure shows a class “car” containing 4 “wheels” by value and an arbitrary number of “indicator” objects by reference



The fundamental difference between agregation by value and reference is that in the first case the contained object is destroyed at the same time as the container one. While in the second case that does not need to be the case

Bibliography

[1] Uml notation guide. http://www.rational.com/uml/resources/documentation