Universidad de Cantabria

Facultad
de
Ciencias

Ensamblado y operación de un telescopio de protones con sensores de tipo Low Gain Avalanche Diode

(Assembly and operation of a proton telescope with Low Gain Avalanche Diode sensors)

Trabajo de Fin de Grado
para acceder al

GRADO EN FÍSICA

Autor: Raúl Penagos Solórzano

Director: Pablo Martínez Ruiz del Árbol

Junio, 2025

Agradezco al grupo CMS del Instituto de Física de Cantabria y en especial a mi director, Pablo Martínez, por su confianza, buen humor y orientación a lo largo de estos meses trabajando juntos.

Gracias a Pedro J. Valle por su ayuda para comprender el sistema óptico.

Asimismo, mis agradecimientos a cada uno de los profesores del grado en Física por sus lecciones, consejos y puntos de vista.

Doy gracias por los compañeros y buenos amigos con que he contado a lo largo de esta etapa.

"Cuando el camino se llena de curvas, lo mejor es tumbar y tocar rodilla."

Resumen

Los sensores Low Gain Avalanche Diode (LGAD), desarrollados originalmente para su uso en el CERN, se caracterizan por su excepcional resolución temporal y su alta resistencia a la radiación. Estas propiedades los hacen especialmente adecuados para aplicaciones en Radiofísica, como su integración en un telescopio de protones destinado a la obtención de imágenes tomográficas. En este trabajo se han llevado a cabo tareas de instrumentación y computación orientadas al ensamblaje de módulos LGAD y a la operación de una simulación del telescopio.

Por un lado, se ha implementado un sistema óptico en un robot para el montaje automático de sensores LGAD, así como un método de calibración programado específicamente para este fin. El proceso de calibración, que ha sido simulado, permitirá al robot determinar la posición de objetos con una precisión en el plano horizontal de $\sigma(x, y)_C = 40$ μm.

Por otro lado, se ha empleado un algoritmo de aprendizaje automático, concretamente una red U-Net, para aplicar técnicas de *denoising* a imágenes generadas mediante la simulación del telescopio de protones. Una vez entrenado, este modelo ha sido capaz de reconstruir imágenes simples con un error cuadrático medio de $MSE = 0.0033 \pm 0.0008$ por píxel, en un rango de valores de 0 a 255.

Palabras clave:
PROTECT, tomografía protónica, ETL, HL-LHC, LGAD, SCARA, procesado de imagen, instrumentación, función de verosimilitud, calibración, U-Net.

Abstract

Low Gain Avalanche Diode (LGAD) sensors, originally developed for use at CERN, are characterized by their exceptional time resolution and high radiation tolerance. These properties make them particularly well-suited for applications in Medical Physics, such as their integration into a proton telescope designed for tomographic imaging. In this work, instrumentation and computing tasks have been carried out to support the assembly of LGAD modules and the operation of a simulation of the telescope.

On one hand, an optical system has been implemented on a robot for the automated assembly of LGAD sensors, along with a specifically programmed calibration method. The calibration process, which has been simulated, will allow the robot to determine object positions with a horizontal plane precision of $\sigma(x, y)_C = 40$ μm.

On the other hand, a machine learning algorithm—specifically a U-Net neural network—has been used to apply denoising techniques to images generated by the simulated proton telescope. Once trained, this model has been able to reconstruct simple images with a mean squared error of $MSE = 0.0033 \pm 0.0008$ per pixel, within a value range from 0 to 255.

Key Words:
PROTECT, proton tomography, ETL, HL-LHC, LGAD, SCARA, image processing, instrumentation, Likelihood, calibration, U-Net.

Índice

1 Introducción y objetivos

Este trabajo se ha desarrollado en el contexto de la instrumentación en Física de Partículas, y la aplicación de sus innovaciones en Radiofísica para obtener un novedoso sistema de tomografía protónica. El elemento clave sobre el cual gira el proyecto son los detectores LGAD (*Low Gain Avalanche Diode*), desarrollados para el detector CMS del CERN, pero que prometen mejorar la planificación y la calidad de vida de los pacientes en tratamientos de protonterapia.

En el Instituto de Física de Cantabria, el grupo CMS [1] trabaja actualmente en la producción en masa y prueba de módulos detectores LGAD para su utilización en el novedoso MTD [2] (*Mips Timing Detector*) para el detector CMS, que entrará en funcionamiento con la actualización a alta luminosidad del Gran Colisionador de Hadrones (HL-LHC) [3]. Estos detectores están especializados en medir con gran precisión el tiempo de paso y posición de partículas cargadas, de modo que son ideales para estudiar su velocidad y energía, siendo además muy resistentes a altos flujos de radiación.

Al mismo tiempo, en el Hospital Universitario Marqués de Valdecilla, se está avanzando en la instalación de un equipo de protonterapia fabricado por la empresa *Varian*. Los tratamientos mediante esta tecnología presentan numerosas ventajas frente a los clásicos basados en RX (radioterapia). No obstante, antes de cada tratamiento, es necesario caracterizar a fondo al paciente con el fin de predecir el comportamiento del haz en su interior, y en la actualidad estos estudios se realizan empleando tomografía axial computarizada TAC, con radiación RX. Esto introduce imprecisiones que podrían ser corregidas si se emplea radiación de protones con el mismo fin.

De esta forma surge PROTECT, un proyecto donde se explora la implementación de tomografía protónica empleando sensores LGAD y los propios equipos de terapia como generadores del haz de protones. Sustituyendo los análisis TAC en pacientes por estudios realizados con un telescopio de protones, se mejorará la definición de los áreas tumorales a irradiar en los tratamientos, lo que tendrá un impacto directo en la calidad del tratamiento y reducirá los posibles efectos adversos por irradiar regiones sanas en el tratamiento.

Objetivos:

Los objetivos de este trabajo se dividen en dos partes: *hardware* y *software*, donde se aplicarán los conocimientos adquiridos a lo largo del grado en Física para contribuir a PROTECT. La parte de *hardware* se centrará en el montaje de una cámara industrial que dotará de visión a un robot utilizado en el ensamblado automatizado de módulos LGAD, al igual que la creación de algoritmos para su operación y calibración. Por otro lado, la parte de *software* se centrará en el desarrollo de un algoritmo basado en técnicas de *machine learning* para la reconstrucción de imágenes obtenidas tras simular el sistema PROTECT.

Esta memoria:

En sus capítulos, se presenta el trabajo y resultados obtenidos acorde a los objetivos descritos. A continuación, se introducirá el estado del arte en Física de Partículas y protonterapia (Cap. 2), justificando así la necesidad del MTD y el interés del proyecto PROTECT. Luego, se describirá la labor realizada en la actualización del robot de montaje automatizado (Cap. 3). Seguidamente, se expondrá el trabajo llevado a cabo en reconstrucción de imagen simulando PROTECT (Cap. 4). Para finalmente, presentar unas conclusiones generales del trabajo (Cap. 5). Adicionalmente, los capítulos se apoyarán en un anexo, incluyendo ilustraciones y la descripción de resultados accesorios a los objetivos principales.

2 Marco teórico y estado del arte

2.1. Estado del arte de la Física de Partículas: LGADs y el HL-LHC

2.1.1. CERN, LHC y CMS

El Gran Colisionador de Hadrones (LHC) es el mayor experimento científico, hasta la fecha, que se ha logrado diseñar, construir y operar con resultados. Situado a 100 metros bajo el Centro Europeo para la Investigación Nuclear (CERN), en la ciudad de Ginebra, este acelerador cuenta con cuatro grandes detectores que permiten el estudio en diferentes campos de la Física de Partículas. Como se muestra en la Fig. (2.1), estos son LHCb (estudio de la materia-antimateria y asimetría en el universo), ATLAS (el mayor detector multipropósito), ALICE (estudia colisiones de iones pesados) y CMS (detector multipropósito compacto). Este trabajo estará centrado en una de las tecnologías desarrolladas para el detector CMS (Compact Muon Solenoid).

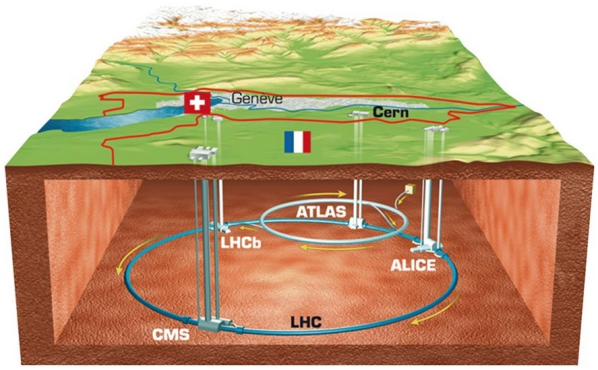

Figura 2.1: *Anillo principal del acelerador LHC y posición de sus cuatro grandes detectores. Imagen tomada de [4].*

Históricamente, el acelerador trabaja en periodos denominados *runs*, entre los cuales todos sus detectores se actualizan y sustituyen, mejorando a sus predecesores en precisión y resistencia a la radiación. Actualmente, (Julio 2022-2025) se está trabajando en el Run 3, gracias al cual se está recolectando más datos que en los dos períodos previos juntos, operando el acelerador a energías de E = 13.6 TeV. El futuro del LHC para el próximo *Run 4*, viene dado por la ambiciosa actualización a Alta Luminosidad (*High Luminosity*, HL-LHC) [3].

El Solenoide Compacto de Muones CMS

El detector CMS (*Compact Muon Solenoid*) [5], es uno de los dos detectores multipropósito del LHC junto a ATLAS. Si bien ambos tienen campos de estudio similares, la tecnología y técnicas empleadas en ambos son distintas. La principal característica de CMS es su tamaño compacto de 21 m de largo y 15 m de diámetro, cuando ATLAS tiene unas dimensiones de 46 m de largo y 25 m de diámetro. Y también, el enorme solenoide con que cuenta para desviar las trayectorias de las partículas generadas en las colisiones, capaz de generar un campo magnético de $B = 4.4$ T [6].
El detector CMS se divide en varias capas, cada una con una función bien determinada y que juntas se combinan para lograr reconstruir los eventos, es decir, los sucesos posteriores a una colisión protón-protón. Todas las capas, centradas en torno al punto de colisión, tienen forma cilíndrica, con secciones longitudinales (barril o *barrel*) y placas circulares en sus extremos (tapas o *endcaps*). En la actualidad, a grandes rasgos, sus capas son [6]:

- **Detector de trazas (*tracker*):** dividido en el *Inner Tracker* (IT) y *Outter Tracker* (OT), consiste en un gran número de detectores de píxeles y tiras de silicio. Estos tienen la función de reconstruir con precisión la trayectoria 3D de partículas cargadas en las proximidades al punto de colisión.
- **Calorímetro electromagnético (ECAL):** formado por cristales de centelleo de $PbWO_4$ que miden la energía de electrones y fotones.
- **Calorímetro hadrónico (HCAL):** mide la energía de hadrones (protones, piones,...) cuando estos generan cascadas hadrónicas en detectores de centelleo.
- **Solenoide:** un gran imán superconductor capaz de generar un campo magnético de hasta $B = 4.4$ Teslas, curvando la trayectoria de las partículas cargadas, separándolas y permitiendo la medida de su momento.
- **Detectores de muones:** las únicas partículas que alcanzan la capa más externa del detector CMS son los muones. Se reconstruye su trayectoria y momento mediante cámaras de deriva y otros detectores especializados.

Además de todas las capas de detectores, para el correcto funcionamiento del CMS son fundamentales el sistema de *trigger* y adquisición de datos, los cuales filtran los eventos más interesantes dentro de los millones que se producen por segundo.

Para su diseño y construcción, ha sido necesaria la colaboración internacional de más de 50 países, con institutos y universidades contribuyendo, en la medida de sus posibilidades, en el diseño, fabricación y pruebas de los componentes. Entre estos está el grupo de altas energías del Instituto de Física de Cantabria (IFCA).

2.1.2. *High Luminosity* - LHC

Para poder estudiar fenómenos en el LHC, se generan colisiones entre pares de paquetes de protones, llamados cada uno *bunch*, que viajan en sentidos opuestos conteniendo unos $n = 10^{11}$ p^+. A pesar del alto número de protones, en la actualidad solo se ven unos 37 eventos (colisiones protón-protón) por cada cruce de dos *bunch*. Sin embargo, en el HL-LHC el número de eventos ascenderá a unos 200, permitiendo obtener una cantidad de datos nunca vista antes.

Para lograr esto, la energía de operación del HL-LHC apenas aumentará respecto a su predecesor, pero se elevará su luminosidad en un factor 10. La luminosidad, Ec. (2.1), mide el número de colisiones que se pueden generar en un acelerador por unidad de superficie y tiempo. L viene dada en función del número de partículas en cada *bunch* N_1 y N_2, la sección de colisión efectiva $S_{eff} = 4\pi \cdot \sigma^2$ siendo σ la sección transversal estimada del *bunch*, y t el período de tiempo entre cruces de paquetes sucesivos. La tasa de colisiones $R = \mathcal{L}\sigma_p$ es proporcional a la luminosidad $\mathcal{L}$ y a la sección eficaz σ del proceso de colisión en concreto [7].

$$L = \frac{N_1 \cdot N_2}{S_{eff} \cdot t} \tag{2.1}$$

Para aumentar esta magnitud, se va a mejorar la calidad del haz, empleando un novedoso sistema de cuadrupolos magnéticos superconductores de Nb_3Sn que, generando un campo mucho mayor que el empleado actualmente, permitirá enfocar mejor el haz y reducir su sección σ en el punto de colisión.

Así, se dará un mayor poder estadístico al experimento, incrementando la probabilidad de observar fenómenos raros y la precisión de las medidas. No obstante, el mayor número de eventos tiene una clara contrapartida: será más complicado discernir los datos de los diversos eventos, ya que los eventos podrán solapar. Esto se conoce como el problema de *pile-up* (acumulación).

En cada evento se producen multitud de partículas y es necesario diferenciarlas para poder reconstruir sus trayectorias. En las anteriores etapas (con unas 40 colisiones), era posible distinguir partículas entre sí por su localización en z, Fig. (2.2, A), siendo esta la coordenada en la dirección del movimiento de los protones; en el futuro esto resultará insuficiente. Como se ve en la Fig. (2.2, B) para discernir los eventos se podría tener en cuenta el momento de tiempo en que ocurren además de su coordenada z.

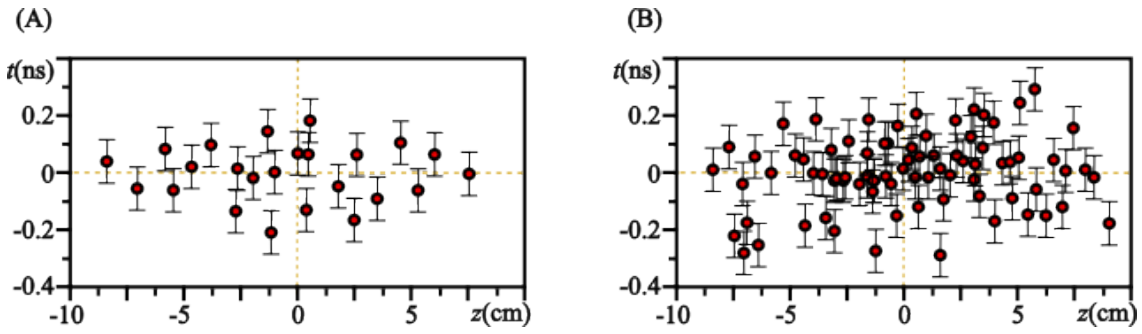

Figura 2.2: *Representación de las colisiones detectadas con respecto al tiempo y posición del cruce de los dos paquetes de protones.* ***(A)*** *en la situación actual, con unos 40 eventos distinguibles por* z.***(B)*** *en alta luminosidad, con cerca de 200, se da el problema de pile-up.*

CMS en HL-LHC

El HL-LHC, alcanzando una luminosidad de hasta $\mathcal{L} = 3000$ fb^{-1}, ofrece una gran oportunidad para la obtención de nuevos datos y estudio de fenómenos raros. No obstante, también supondrá un gran reto técnico para su diseño y construcción, dado el duro ambiente al que se someterán los componentes. La dosis de radiación recibida por los componentes del detector en "flujo equivalente

a neutrones de 1 MeV.ªumentará en casi 10 ordenes de magnitud, hasta $\phi = 1.6 \cdot 10^{15}$ n_{eq}/cm^2 en las partes más expuestas.

Con la nueva era, se hacen necesarias una serie de actualizaciones en todas las partes enumeradas en la introducción. En este apartado, se comentaran los cambios afectando al *tracker*, y cómo se atacará el problema de *pile-up* añadiendo una nueva capa de detección.

El *inner tracker* y *outter tracker* son dos detectores formados principalmente por píxeles de silicio que permiten determinar el origen, momento y carga de las partículas generadas en las proximidades de la colisión. Esta información, para una partícula, se denomina traza o *track*. Para afrontar el HL-LHC, ambos han sido rediseñados para ser más rápidos, precisos y resistentes a la radicación. Con este fin, contarán con más capas de detección y una mayor cobertura angular, mayor granularidad (resolución), menos material (reduciendo el scattering) y un sistema de enfriamiento más eficiente.

Una de las grandes innovaciones para el HL-LHC viene con el MIPs *Timing Detector* (MTD), la solución para el problema de *pile-up*. Este nuevo detector dará a CMS una nueva habilidad: medir de forma precisa el tiempo de paso (ToF, *Time of Flight*) de las partículas mínimamente ionizantes (MIPs). El MTD, permitirá una reconstrucción 4D de las 140-200 trazas acumuladas en cada cruce de paquetes de protones, reduciendo el número de trazas asignadas incorrectamente, y resolviendo así el problema de *pile-up*.

2.1.3. *MIPs Timing Detector* (MTD)

Para lograr su objetivo, el MTD se ha ideado como otra capa cilíndrica de detectores que envuelve al OT, dividida en dos tapas circulares: *endcap timing layer* (ETL) y el barril cilíndrico: *barrel timing layer* (BTL). Para cumplir su función, los detectores deberán satisfacer ciertos estándares [8]:

- Los detectores deben contar con una gran tolerancia a campos magnéticos. Además, los módulos deberán ser robustos mecánicamente, facilitando su instalación y permitiendo que superen la duración completa de la fase HL-LHC.
- Deben entrar en el espacio entre el OT y el ECAL. Esto es un espacio radial de $\Delta r = 40$ mm para el BTL y de $\Delta z = 45$ mm para el ETL. Igualmente, su geometría debe minimizar el espacio muerto (no cubierto por los detectores).
- Su resolución temporal deberá ser de $\delta(t) = 30 - 40$ ps, y podrá degradarse debido a la radiación hasta $\delta(t) = 50 - 60$ ps para el final del funcionamiento del HL-LHC. Esto asegura que la solución contra el problema de $pile - up$ sea efectiva durante toda su vida útil.
- Deben resistir dosis de radiación de hasta $\phi_{BTL} = 3 \cdot 10^{14}$ n_{eq}/cm^2 y $\phi_{ETL} = 3 \cdot 10^{15}$ n_{eq}/cm^2 en el barril y tapas respectivamente.
- Por último, su temperatura de funcionamiento deberá adaptarse a las condiciones entre el OT y ECAL a $T = -30°C$. Además, para mitigar los efectos de la radiación los detectores también se someterán a etapas de *annealing* (recocido) a temperatura ambiente o superior, las cuales deberá tolerar.

Dadas estas condiciones, y para el ETL, se comprobó que los sensores que mejor rendimiento ofrecían eran los LGADs. La razón para no implementarlos en el BTL también, es su alto coste y que en este la dosis de radiación recibida no es tan alta, por lo que se emplea una tecnología distinta, basada en cristales centelladores.

2.1.4. *Low Gain Avalanche Diodes*

Los detectores basados en materiales semiconductores, o detectores de estado sólido, son un estándar en la detección de partículas. En esta sección se introducirán las nociones básicas para comprender su funcionamiento y se describirá la tecnología LGAD.

Para introducir el tema, se explicará el funcionamiento de la cámara de ionización, una tecnología más rudimentaria con la misma función. Esta consta de dos electrodos cargados en los extremos de una cámara de gas a baja presión. Cuando algún tipo de radiación atraviesa el gas, libera iones y electrones que generan una corriente entre los electrodos, detectando mediante un pulso eléctrico la radiación incidente. En el caso de un detector de estado sólido, también se producen pulsos de corriente cuando una partícula atraviesa el dispositivo. Sin embargo, la energía de las partículas detectadas puede ser muy inferior. Esto es debido a que el mecanismo subyacente se basa en la generación de pares electrón-hueco, y no en la ionización de un gas.

Los detectores de estado sólido se puede construir combinando diferentes materiales semiconductores, generalmente se emplea silicio con impurezas dopantes en su red cristalina. Las impurezas, átomos con diferente número de electrones de valencia, pueden ser aceptoras si el átomo introducido tiene electrones de valencia de menos (huecos), o donoras si los posee en exceso. Los semiconductores dopados con tales átomos se denominan P y N respectivamente. Combinando ambos tipos, por ejemplo con una unión PN, es posible construir componentes electrónicos, como los diodos o los fotodiodos, que se emplean en detección.

Al poner un semiconductor P y uno N en contacto, se reordenan los electrones e iones generando un extremo de carga positiva y otro de carga negativa. En la región intermedia, no existen portadores libres de carga y se genera un campo eléctrico por la diferencia de carga entre los extremos, esta región se llama zona de depleción o agotamiento. Entonces, si se hace incidir radiación en la región de depleción de un fotodiodo, se generan pares electrón-hueco por la excitación de los fotones con energía mayor a la de la energía de gap de la red cristalina. Estos dispositivos trabajan en polarización inversa, aplicando una tensión inversa que ensancha la zona de agotamiento. En general no conducen corriente eléctrica, excepto cuando una partícula los atraviesa y se percibe la generación de pares electrón-hueco.

Para mejorar su respuesta y aumentar la eficiencia de detección, se han desarrollado dispositivos más complejos, como el diodo *pin*. Este se caracteriza por su estructura de tres capas: una capa P, una capa I (intrínseca[1]) y una capa N. En esta configuración, la capa intrínseca juega el papel de separar las regiones P y N. Físicamente, al aplicar un voltaje inverso al diodo, es decir un potencial positivo en la capa N, se crea una región de carga espacial en la capa intrínseca. Los portadores de carga se recombinan y forman una región de agotamiento más amplia que en una unión PN. Los fotones o partículas cargadas que atraviesan esta región pueden generan pares electrón-hueco, estos se desplazan por el campo eléctrico interno hacia el cátodo y ánodo del diodo. De este modo, se genera una corriente proporcional a la cantidad de energía depositada por la partícula.

Los diodos de avalancha tiene un funcionamiento similar a los *pin* y permiten ampliar la señal en un factor llamado ganancia interna M. Estos aprovechan un campo eléctrico intenso para acelerar los portadores de carga primarios, generados por los fotones o partículas incidentes. Así, al adquirir suficiente energía cinética, estos portadores provocan cascadas de ionizaciones, generando un núme-

[1] Semiconductor intrínseco: compuesto por un solo átomo, cuenta con igual número de portadores de carga n y p.

ro de portadores secundarios M veces mayor que el inicial. Este proceso incrementa la corriente medida, facilitando la distinción del pulso frente al ruido electrónico. Además, incluso partículas con baja energía pueden producir una señal significativa, mejorando la eficiencia de detección de eventos débiles.

Los *Low Gain Avalanche Diodes* son una evolución de estos dispositivos, diseñados específicamente para la detección de partículas cargadas con una excelente resolución temporal. Alcanzan valores de hasta $\Delta t = 3{\cdot}10^{-11}$ s, lo que los hace ideales para medidas precisas del tiempo de paso de partículas.

Para lograr una buena resolución temporal, es crucial que la señal eléctrica tenga un frente de subida abrupto, mientras que la caída puede ser más lenta. Esto se consigue con una zona de depleción estrecha, que permite una rápida recogida de los portadores de carga. Sin embargo, si esta zona es demasiado delgada, la señal será rápida pero débil, y podrá confundirse con el ruido de fondo. Por el contrario, una zona de depleción más ancha genera una señal más intensa, pero con una pendiente de subida más suave, degradando la precisión temporal.

Los diodos LGAD ofrecen una solución innovadora gracias a su estructura interna, que presenta modificaciones con respecto a la del diodo *pin*. Justo debajo de la región N del cátodo, incorporan una capa complementaria de material semiconductor tipo $P+$, con menos impurezas que la región $P++$ del ánodo. Esta, genera un campo eléctrico muy intenso, ver Fig. (2.3), que acelera los portadores en la región previa al cátodo, lo que permite una velocidad sin precedentes en la señal. Para evitar problemas de ruido y conseguir una señal rápida y de gran amplitud, la ganancia de los LGAD es moderada $M = (10 - 30)$.

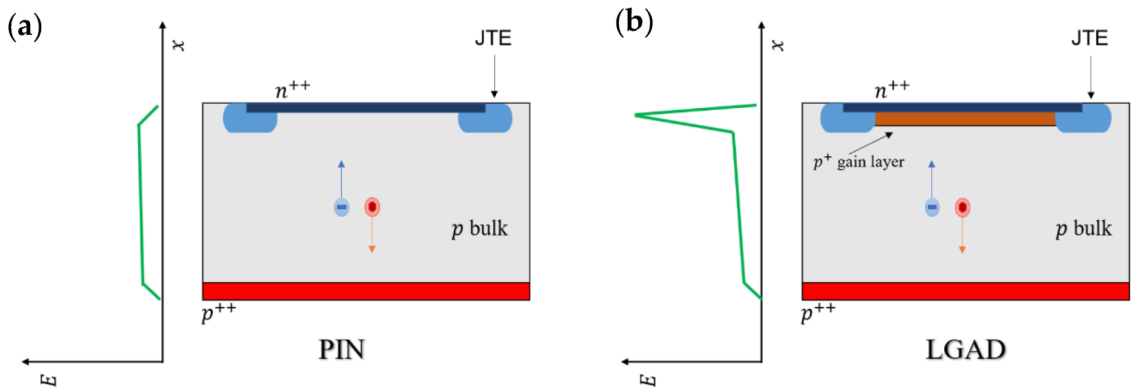

Figura 2.3: *Comparación de la estructura interna y campo eléctrico en un diodo pin* ***(a)*** *y un diodo LGAD* ***(b)****. Este ejemplo utiliza un sustrato p dopado levemente (de alta resistividad), que en la práctica resulta como un material intrínseco. Se ve como el campo* $\vec{E}$ *aumenta justo antes del cátodo en el diodo LGAD. Imagen tomada de [9].*

Módulos LGAD

Para su utilización en el ETL, los sensores de Si LGAD se montan en módulos independientes con chips para su alimentación y la extracción de datos, que facilitan su manipulación e instalación [8].

Los módulos están compuestos por sensores LGAD de 21×21 mm^2, cada uno es una matriz de 16×16 píxeles de 1.3×1.3 mm^2 y 300 μm de espesor. Cada sensor está conectado a dos chips de

lectura ETROC (*Endcap Timing ReadOut Chip*), que digitalizan la señal y extraen su información de tiempo de llegada (ToA) y tiempo sobre umbral (ToT).

Cada módulo cuenta con cuatro sensores que se ensamblan sobre una base de nitruro de aluminio (AlN), que proporciona una buena conductividad térmica y compatibilidad con la expansión térmica del silicio. Esta base se conecta con circuitos flexibles, que transmiten señales y alimentación desde las tarjetas de servicio, que también se encargan de la conversión de voltaje, distribución de reloj y comunicación con el sistema de adquisición de datos.

Cada módulo está encapsulado con una cubierta protectora de AlN y diseñado para ser mecánicamente robusto y térmicamente eficiente. Para su producción en masa, se recurre a estrategias como la que se introducirá en el Cap. 3.

2.2. Física de Partículas de libro

Cuando se estudia la interacción radiación-materia se diferencia a grandes rasgos entre partículas con carga (protones, electrones, alfa,...) y neutras (gamma y neutrones). En esta sección se introducirán las bases para entender el comportamiento al atravesar materia de los fotones y protones.

Los fotones interactúan con la materia mediante tres procesos: fotoeléctrico, efecto Compton y la producción de pares e^-e^+, si bien los dos últimos son muy improbables para energías iguales o inferiores a las de los rayos X. Al atravesar un medio, los fotones sufren una atenuación exponencial acorde a la Ley de Lambert-Beer, Ec. (2.2). Esta, describe la intensidad de un haz de fotones $I(x)$, al atravesar un espesor x de un medio con coeficiente de atenuación lineal μ, si su intensidad inicial es I_0.

$$I(x) = I_0 \cdot e^{-\mu \cdot x} \tag{2.2}$$

Los protones en cambio, partículas cargadas pesadas, interactúan principalmente de tres formas con la materia para energías superiores a $E_P = 10$ MeV. En primer lugar, ceden energía a los electrones del medio atravesado mediante ionización, si los expulsan, o excitación si solo hacen que el electrón cambie a un nivel superior de energía. En segundo lugar, sufren scattering elástico múltiple, por el cual se desvían de su trayectoria sin ceder energía. Y por último, y de forma más relevante a partir de $E_P = 100$ MeV, experimentan reacciones nucleares, que podrán ser elásticas causando una gran desviación en la trayectoria, o inelásticas si hay un cambio en la naturaleza y trayectoria de las partículas.

Para describir su pérdida de energía se define el poder de frenado $S = -\langle dE/dX \rangle$, la pérdida diferencial de energía por distancia recorrida. Esta magnitud depende de la densidad electrónica n_e y del potencial medio de ionización del medio I, es decir la cantidad de energía que se debe transferir para expulsar un electrón. Tomando la hipótesis de electrones libres, se puede llegar a la Ec. de Bethe-Bloch que describe el *stopping power* de una partícula con carga z y velocidad v en un medio, Ec. (2.3), donde m_e es la masa del electrón, c la velocidad de la luz, $\beta = v/c$, e es la carga del electrón y ε_0 es la permitividad del vacío.

$$-\left\langle \frac{dE}{dx} \right\rangle = \frac{4\pi}{m_e c^2} \cdot \frac{n_e z^2}{\beta^2} \cdot \left(\frac{e^2}{4\pi\varepsilon_0} \right)^2 \cdot \left[ln\left(\frac{2m_e c^2 \beta^2}{I \cdot (1-\beta^2)} \right) - \beta^2 \right] \tag{2.3}$$

Entre otras, la principal conlcusión de la Ec. (2.3) es que la pérdida de energía no depende de la masa de las partículas, sino solo de su velocidad $S \propto 1/v^2$ y carga.

Para analizar el alcance de la radiación en el medio, se utiliza la curva de Bragg, una representación de la tasa de pérdida de energía para partículas cargadas al atravesar un medio.

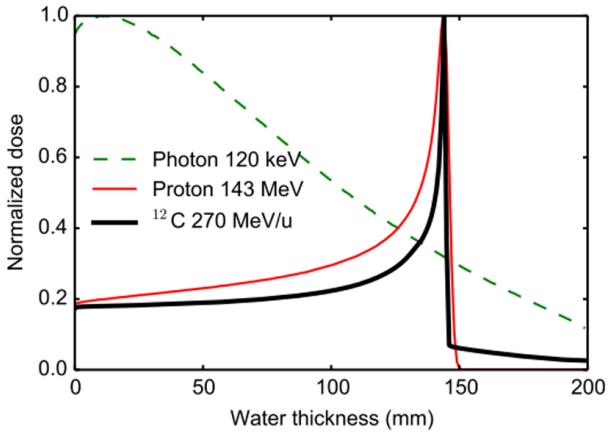

Figura 2.4: *Comparación normalizada de la dosis depositada en agua de un fotón, un protón y un ión de carbono. Figura tomada de Catherine T. Quiñones [10].*

Para un protón, en la Fig. (2.4) se ve como inicialmente la pérdida de energía es aproximadamente constante y, según la partícula profundiza perdiendo energía y velocidad, crece hasta un máximo abrupto llamado pico de Bragg. Es a esta profundidad donde se depositará una mayor dosis de energía. Por el contrario, los fotones depositarán una dosis que disminuye exponencialmente a lo largo de todo su recorrido por el medio, y es máxima en la superficie del medio.

2.3. Protonterapia

El estándar en países desarrollados para el tratamiento con radiación externa de tumores es la radioterapia de intensidad modulada con fotones. No obstante, en las últimas décadas se están incorporando nuevos equipos de protonterapia por sus ventajas.

Como se ha visto en la Fig. (2.4), la curva de deposición de energía de los protones tiene una distribución de dosis mucho más selectiva, al contar con un marcado pico de Bragg a cierta profundidad. En cambio, los fotones de RX, con un comportamiento exponencial, depositan una dosis importante en todo el tejido atravesado hasta alcanzar la profundidad objetivo del tratamiento.

Variando la energía de un haz de protones, normalmente $E = 70-200$ MeV, y combinando múltiples haces, se logra una acotación aún mejor de la zona de máxima dosis. En la práctica, esto permite aplicar dosis en las zonas tumorales de una forma más precisa e inocua para los tejidos sanos que con terapia RX, acorde al principio básico de la radioterapia "Dosis máxima en la zona tumoral y mínima en el tejido sano". Por esto, resulta una alternativa ideal cuando se trabaja en las proximidades de órganos críticos.

Los equipos de protonterapia constan de dos partes principales: el generador de protones (ciclotrón o sincrotrón), y el puerto de administración. En un ciclotrón, los protones se generan a partir de la ionización de gas hidrógeno y se aceleran con un fuerte campo eléctrico mientras se hacen girar con un campo magnético. Una vez alcanzada la energía de funcionamiento, se puede modular el haz a

valores más bajos utilizando atenuadores de grafito. Esta atenuación es muy efectiva, pero conlleva la generación de radiación de neutrones de muy alta energía debido a reacciones nucleares inelásticas. Para contrarrestar este problema los equipos se construyen en bunkers con paredes de hasta tres metros de espesor de hormigón baritado. Fuera de este, se encuentra la sala de tratamiento, a donde el haz llega con la energía específica para el tratamiento a través de guías ópticas [11].

Los puestos de administración, análogos a los de radioterapia de fotones, cuentan con un portal con filtros que permiten colimar el haz sobre el paciente. El portal o *gantry* puede ser fijo o permitir la rotación del eje principal del haz, lo que facilita considerablemente la aplicación de haces múltiples.

En España ya existen dos equipos operativos y se espera que el número supere los 10 en los próximos años, incluyendo uno en el Hospital Universitario Marqués de Valdecilla, en Cantabria. En este, ya se ha avanzado en la construcción del búnker de hormigón para la instalación de un ciclotrón y *gantry* orientanle en $\psi = 360°$ para terapia con protones, de la marca americana *Varian*. Este es distinto de los equipos de la marca IBA que se instalarán por el resto de la península, que solo rotan $\psi = 270°$.

2.3.1. Tomografía protónica: PROTECT

La terapia protónica ofrece la oportunidad de centrar la energía de radiación en la zona tumoral de una forma muy precisa, lo que supone una ventaja frente a la radioterapia convencional. Para su aplicación es necesario un conocimiento preciso del volumen, densidad y *stopping power* del tumor y los tejidos circundantes. De este modo, los profesionales pueden planificar el tratamiento en función de la energía incidente para que el pico de Bragg coincida exactamente con el volumen del tumor y la dosis total sea suficiente para que el tratamiento sea efectivo.

Normalmente, este estudio se realiza mediante tomografía axial computarizada TAC, que emplea radiación RX. Sin embargo, como se ha explicado, los fotones interaccionan de una forma distinta a la de los protones. Por lo que se requieren correcciones, que no siempre son acertadas.

Por ello, resulta ideal desarrollar técnicas de tomografía que, utilizando un haz de protones, permitan describir con precisión el *stopping power* del tejido tal y como lo experimentan los hadrones. Así, surge el proyecto PROTECT [2], donde se utilizarán detectores LGAD de vanguardia para construir un dispositivo de tomografía protónica, utilizando como base equipos de tratamiento *Varian*. La elección de estos sensores responde a las mismas ventajas que los hacen idóneos para el ETL: su alta resistencia a la radiación, excelente resolución temporal, reducido espesor... además de contar con la ventaja añadida de estar siendo fabricados en España.

La tomografía[3] es una técnica de imagen que permite reconstruir imágenes tridimensionales, planos o capas de estructuras no apreciables inicialmente, mediante la superposición de múltiples imágenes de atenuación bidimensionales [12]. Se basa en la reconstrucción de los índices de atenuación en secciones de volumen. Por tanto, es necesario contar con una fuente y un detector de radiación (fotones RX, o protones), para medir la atenuación del haz al atravesar un objeto, para posteriormente reconstruirla mediante algoritmos matemáticos [12]. Para que la reconstrucción sea posible, es necesario tomar imágenes desde diferentes ángulos, cuantas más sean, mayor será el detalle de

[2] *Cutting-Edge Semiconductor detectors for Advanced Proton Tomography in Oncological Proton Therapy* (PROTECT).

[3] Tomografía: del latín 'tomus-' sección.

la información sobre el objeto. Por esta razón, resulta clave que el equipo que se instalará en el Hospital de Valdecilla cuenta con un *gantry* de orientación $\psi = 360°$.

2.3.2. El dispositivo PROTECT

Antes de poder aplicar algoritmos de reconstrucción tomográfica, como el *back-filtered projection*, es necesario poder obtener imágenes planas y ajustar los parámetros del detector mediante simulaciones software para que estas sean de calidad.

El *gantry* de PROTECT ha sido maquetado en Geant4, un software escrito en C++ y utilizado para la simulación de detectores e interacción de partículas y materia. En este programa, se definen de forma fidedigna las características del haz y los planos con LGADs.

El *gantry* consta de dos detectores, el haz atraviesa primero el "detector 1" y, tras incidir en el paciente, llega al "detector 2". Ambos están formados por 4 planos, cada uno de los cuales es una matriz cuadrada de 16×16 módulos LGAD de ganancia $G = 10$ cada uno. Los planos están colocados sobre una fina capa de $e_C = 1$ mm de fibra de carbono que sirve como soporte, para los módulos completos: con su *ETROC* y sensor de silicio de $e_{Si} = 300$ μm.

La disposición de los planos con respecto al centro del *gantry* se muestra en la Fig. (2.5). En el centro, se colocará el hipotético paciente o los *phantoms*[4] para realizar pruebas.

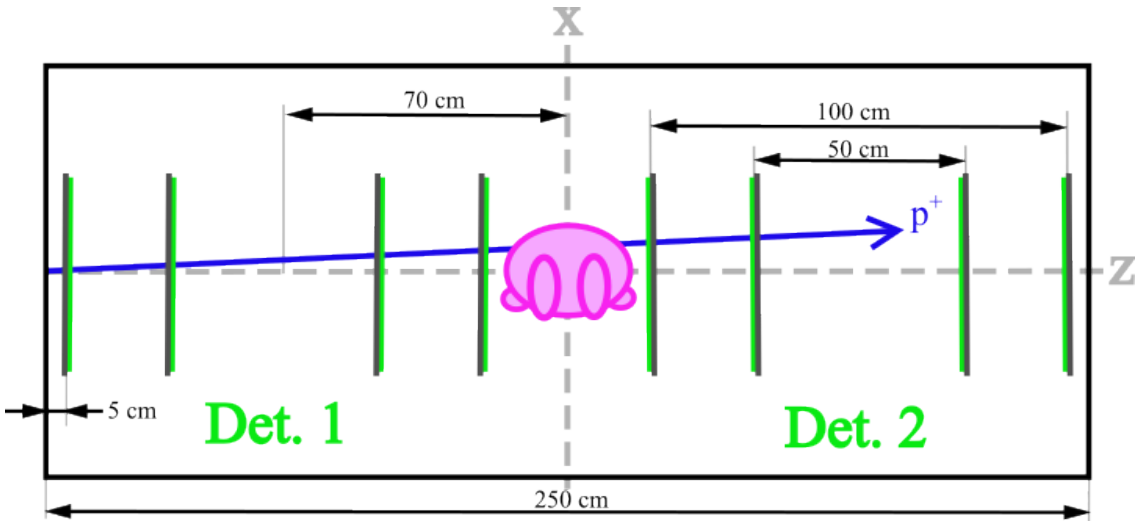

Figura 2.5: *Dibujo a escala de los planos de detección de PROTECT. El dispositivo es simétrico con respecto al puntos central* $(0, 0, 0)$*, donde se coloca al paciente (rosa). El haz de protones (azul) entra por detrás del primer plano del detector 1.*

El *beam* de protones, utilizado como sonda de estudio, ha sido simulado con distribuciones Gaussianas en sus parámetros [13]. En el programa, se genera un haz con una sección de $\sigma_{x,y} = 0.1 \times 0.1$ cm^2. Además, para cumplir su función y cubrir todo el área central del *gantry*, está programado para realizar un barrido en el plano xy de $N = 400$ pasos. Igualmente, el haz se puede modificar en cuanto al número de protones y su energía, que toma valores en el rango de trabajo de los equipos *Varian*, $E_P = 170 - 250$ MeV.

[4]Fantomas: modelos que simulan las características del tejido humano.

3 Montaje automático de módulos

En este capítulo, se describirá brevemente el sistema robotizado para montaje automático de módulos LGAD y se profundizará en el trabajo realizado para mejorar su operación con un sistema óptico.

3.1. Robots SCARA

El montaje de módulos por parte de los grupos de investigación es una tarea repetitiva y que requiere gran precisión y capacidad de replicar resultados. Por esta razón, se recurre a la utilización de sistemas automatizados. Más aún cuando el número de detectores a producir ronda el millar, con un objetivo de precisión en el montaje de $\delta(x, y) = 50$ μm.[1]

El Instituto de Física de Cantabria cuenta con un Robot Industrial SCARA, (selective compliance assembly robot arm), este tipo de brazo robotizado resulta ideal cuando se necesita buena repetitividad y ciclos rápidos de trabajo. Esta clase de robot está muy extendido por toda la industria y existe cierto convenio para describir su posición y movimiento. Los SCARA cuentan con cuatro grados de libertad, dos rotaciones axiales: $J_1 : [-\pi, \pi]$ y $J_2 : (-\pi, \pi)$ que dan movilidad a las articulaciones de ambos brazos, y, en el émbolo del extremo del robot, un control de altura z y rotación de su eje J_z. El ángulo J_1 se define en el primer brazo con respecto a la base fija, el ángulo J_2 es el formado por el segundo brazo con el primero, el control de altura z da la posición del émbolo relativo a la altura de los brazos, y por último, la rotación J_z es la rotación del émbolo respecto al extremo del segundo brazo.

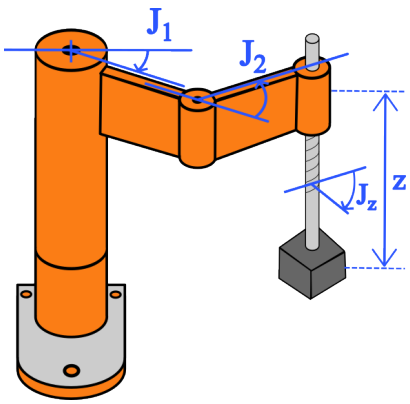

Figura 3.1: *Esquema general de un robot SCARA (naranja) con 3 ejes verticales de rotación, sobre su base fija (gris claro) y el émbolo (gris oscuro) se acotan sus 4 coordenadas propias* (J_1, J_2, z, J_z).

[1] El grupo CMS montará 900 módulos LGAD para el ETL en los próximos meses.

De esta manera, su rango de trabajo resulta equivalente aproximadamente al de un robot cartesiano (tipo impresora 3D), pero el montaje sobre una sola base y su menor número de partes lo hace mucho más cómodo para algunas aplicaciones. Entonces, se pueden definir dos sistemas de coordenadas: cartesianas (x, y, z, J_z) y el sistema propio del robot que se definirá *inner* (J_1, J_2, z, J_z) equivalentes entre sí. Sin embargo es necesario cierto formalismo que se tratará en las siguientes páginas para lograr la univocidad y correcta correspondencia entre los dos sistemas.

3.1.1. El robot

El robot utilizado está fabricado por la empresa tecnológica suiza Stäubli, se trata de un modelo TS2-60 SCARA [2]. Sus dimensiones se señalan en la Fig. (3.2). Su capacidad de carga es de $m_{max} = 8.4$ kg y cuenta con un alcance (ambos brazos alineado) de $l_1 + l_2 = 60$ cm, al mismo tiempo tiene una repetibilidad en las direcciones cartesianas x e y de $\sigma_{xy} = \pm 0,01$ mm y en z de $\sigma_z = \pm 0,004$ mm.

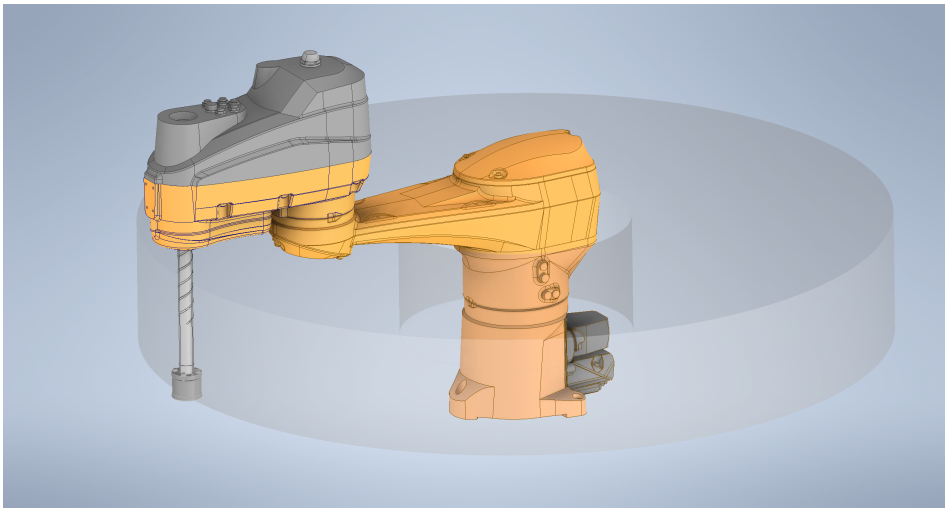

Figura 3.2: *Imagen CAD del robot utilizado. Se sombrea el rango de trabajo del robot. La longitud de los brazos es* $l_1 = 38$ *cm y* $l_2 = 24$ *cm respectivamente, y su altura es* $z_R = 60.77$ *cm. Imagen cedida por David Moya Martín.*

Para su operación en el montaje de módulos, se fija sobre una mesa estabilizada calibrada con una matriz de orificios de métrica 8 ($\varnothing = 8$ mm), sobre la cual se definen zonas con las diferentes piezas a montar en posiciones conocidas y zonas con adhesivo para el pegado de las misma. Con la correcta programación y calibración, el robot toma las piezas, aplica el adhesivo y las reposiciona pegándolas.

Por ejemplo, actualmente el robot se emplea para el montaje de módulos del *Inner Tracker*. Se parte de tres componentes: una pieza formada por el chip y sensor de Si, los cuales se reciben ya pegados en otro centro de investigación, un chip de conexiones HDI y un protector de espuma llamado *Airex*. El operador sitúa y esquina las piezas en unos soportes plantilla de posiciones conocidas para el robot, o *jigs*, y prepara una capa homogénea de adhesivo en otra región de la mesa. Entonces, el robot manipula las piezas, gracias a una serie de cabezales con líneas de vacío y electroimanes, montando hasta 6 módulos al mismo tiempo.

[2] Stäubli Mod. TS2-60-HB-FL-200-L-WS-FLA.

El trabajo realizado a lo largo de este TFG ha contribuido para que, en los próximos meses, el robot se pueda utilizar en el montaje de módulos LGAD para el ETL y para el proyecto PROTECT. En concreto, se ha montado y configurado un sistema óptico para dotar de visión al robot, y se han generado algoritmos para su operación y calibración. Así pues, estas labores se han centrado en instrumentación óptica, programación avanzada orientada a objetos y optimización geométrica. De este modo, se podrá realizar un montaje dentro del margen de repetibilidad necesario para el montaje de módulos LGAD $\delta x, \delta y = 50$ μm.

3.2. Sistema óptico

Como se ha explicado, en el montaje para módulos del IT, el robot supone que la colocación de las piezas sobre la mesa es perfecta y se centra en moverlas entre posiciones definidas con gran precisión. Para los módulos LGAD del ETL se ha optado por utilizar un sistema automatizado y más preciso, el robot contará con un sistema de visión: un microscopio con cámara que le permitirá deducir la posición de las piezas mediante la toma de imágenes de sus marcas de referencia o fiduciales, Fig. (3.5).

El sistema óptico adquirido consiste en los siguiente elementos:

- Montura C: Es la principal pieza del microscopio, la que se montará en el robot y a la cual se acoplan el resto de componentes gracias a sus roscas estándar. Está diseñada para objetivos infinitos de microscopio, para lo que cuenta con una lente de $f = 200$ mm que forma la imagen en el plano donde se coloca la cámara. Además, cuenta con un codo para una fibra óptica que asista en la toma de imágenes. En su interior hay un *beam spliter* o divisor de haz que hace incidir la iluminación de la fibra en la dirección del objetivo.
- Objetivo: se trata de un Mitutoyo M-Plan Apo 5x/0.14 f=200 (∞/0). Es un objetivo infinito [3]. Combinado con una lente de focal $f = 200$ mm se obtiene su mejor calidad de imagen y un aumento $M = 5\times$. Su apertura numérica (capacidad para captar luz) es $A.N. = 0.14$, y la distancia focal del objetivo es de $f' = 40$ mm.
- Cámara industrial IDS[4]: puede ser conectada a un ordenador utilizando un cable USB 3.0. El tamaño de su sensor CMOS (2064 $\times$ 1544 px) es $d_s = 1/1.18''$. Este estándar de sensor mide $x_s, y_s = 7.2, 5.4$ mm.
- Fuente de luz: Se utiliza una Fiber-Lite MI-152 de luz halógena, que fue sustituida por una luz led posteriormente. La luz se transmite utilizando un cable de fibras ópticas de $s = 1/4$".

Uno de los primeros pasos para probar las funciones de la cámara sin llegar a montarla en el robot, fue el diseño en un programa CAD[5], de dos acoples para la fibra. Los acoples, impresos en 3D, unían la fibra al codo del microscopio por un lado, y la fibra a la fuente de luz. En el segundo caso, fue necesario utilizar material de impresión más resistente a las altas temperaturas, ya que la bombilla halógena fundía las piezas de PLA (el estándar en impresoras 3D), por lo que se empleó plástico ABS y Onyx. Más adelante el problema se resolvió sustituyendo la bombilla por una de tipo LED.

[3]Objetivo infinito: el plano imagen, donde se forma la imagen del objeto, está en el ∞.
[4]Cámara: IDS Mod. U3-3270 CP Rev. 2.2.
[5]CAD: *(Computer-Aided Design)*, programa informático de diseño asistido.

3.2.1. Montaje del sistema óptico

El sistema óptico se montó para realizar pruebas y conocer las características del enfoque y campo de visión antes de incorporarlo al robot. En el Anexo .1 se incluye una ilustración detallada con las partes principales.

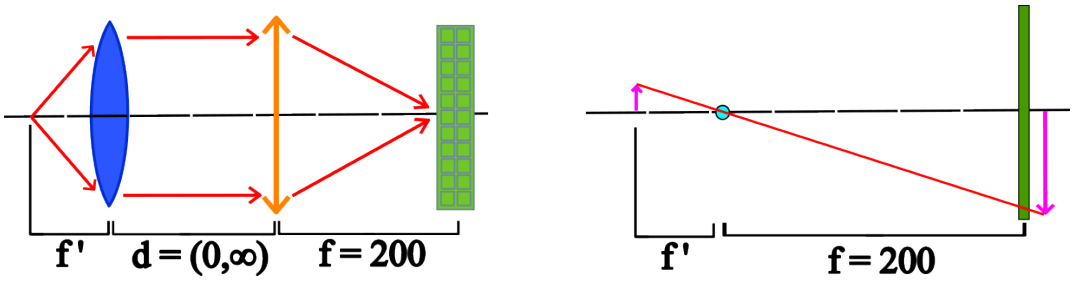

Figura 3.3: ***Izq.*** *Esquema del sistema óptico, lente objetivo (azul), lente correctora (naranja), y plano imagen en la cámara (verde).* ***Dcha.*** *Esquema de la cámara pinhole equivalente.*

El sistema óptico montado consiste en un objetivo que enfoca objetos a la distancia focal del objetivo f' y forma su imagen en ∞, y la lente auxiliar de la montura que recoge la imagen y la forma sobre el detector de la cámara a una distancia $f = 200$ mm. Este sistema se puede modelizar como una cámara *pinhole*, en la Fig. (3.3, Dcha.), con las dos distancias focales que dan un aumento $M = 200/40 = 5\times$. Este modelo hace que los rayos tengan que atravesar un orificio para incidir en el plano imagen. De esta forma, conocer la proyección de imágenes a través del sistema óptico resulta trivial.

Dadas las características de la cámara y el sistema óptico $M = 5\times$, se puede calcular su resolución espacial y estimar su campo de visión. Si la pupila de entrada[6] del sistema es el sensor CMOS, el campo de visión será $fov_{xy} = (7.2 \text{ mm}, 5.4 \text{ mm})/5 = (1.44 \text{ mm}, 1.08 \text{ mm})$. Además, la distancia de trabajo del objetivo, a la que tienen que estar los objetos que enfoca, estará en torno a $f' = 40$ mm.

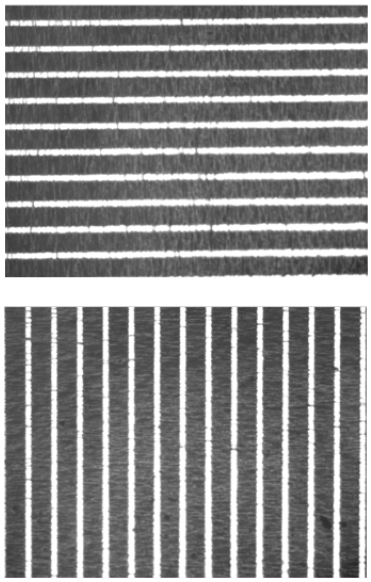

Figura 3.4: *Imágenes de una regla óptica orientada en los dos ejes cartesianos del sensor de la cámara. Las líneas brillantes están espaciadas* $\Delta x = 100\ \mu m$.

Una vez montado, y configurando la cámara para fotografiar empleando el sensor completo, se comprobó que la pupila de entrada del sistema es el propio sensor CMOS, ya que se veía una imagen sin bordes. Además, se midió que la distancia de trabajo del microscopio es de unos $l_w = 36$ mm. Asimismo, fotografiando una regla óptica, Fig. (3.4), y contando líneas, se pudo confirmar que el campo de visión es de $fov_{xy} = (1.4 \text{ mm}, 1.1 \text{ mm})$, como se había

[6]Pupila de entrada: elemento más restrictivo que corta la imagen formada en un sistema óptico.

estimado. Por último, se concluyó que la profundidad de campo[7] del sistema óptico es muy reducida, menor a $DOF << 0.5$ mm, por lo que todas la fotografías se realizarán a un altura $l_w = 36$ mm sobre los objetos.

3.2.2. Operación de la cámara

Una parte adicional al montaje del sistema óptico es su puesta a punto y el aprendizaje del uso de su software de control. Siendo la primera vez que se empelaba una cámara IDS, fue necesario adentrarse en la extensiva documentación de la marca e instalar el diferente *firmware*, librerías disponibles y la *suite* de software para controlar dispositivos de la marca IDS.

Una vez conectada por USB, la cámara se puede operar desde el software *IDS-peak Cockpit* para tomar imágenes o vídeo, con total libertad para ajustar parámetros de exposición, velocidad de captura, ISO, región activa del sensor (ROI), *downsampling*... Pero también, se puede operar desde *scripts* de código Python, Java y C++.

En lenguaje Python, se aprendió a controlar en detalle la cámara: buscar y abrir el dispositivo, iniciar su memoria *buffer*, establecer los parámetros de captura y realizar fotografías. Con este fin, se profundizó en la documentación de IDS y se preparó un *script* para operar la cámara y analizar las imágenes tomadas. Un resumen del trabajo realizado en este contexto se recoge en el Anexo .2.

3.3. Calibración del sistema óptico

3.3.1. Introducción

La función principal del microscopio es dar las coordenadas $(x, y, z)_{3D}$ de puntos que se fotografíen, para lo cual es fundamental conocer su posición. Calibrar la cámara consiste en determinar exactamente su posición y orientación con respecto al robot. En la Fig. (3.7) se puede ver el posicionamiento de la cámara con respecto al robot, que se describirá con los parámetros $(x, y, z, \psi, \theta, \phi)^0_C$, acorde a su posición y orientación respecto al extremo del émbolo. Así, el sistema podrá ser utilizado para identificar la posición y orientación de las diferentes piezas de los módulos al montarlos. Los puntos que se fotografiarán de ls componentes serán las marcas fiduciales en sus esquinas, Fig. (3.5).

Con el fin de calibrar el sistema óptico, se fotografiarán puntos con posición bien conocida sobre la mesa, en cada uno de los cuales se conocerá su posición, las coordenadas propias del robot, y las coordenadas de proyección del punto sobre el sensor CMOS de la cámara. Comparando los resultados reales y los obtenidos con una estimación de los parámetros de la cámara $(x, y, z, \psi, \theta, \phi)^{est.}_C$ simulada, se espera poder obtener información acerca del valor real de los mismos.
Ahora bien, este problema es geométrica y analíticamente complejo, razón por la que se debe acudir a la utilización de herramientas avanzadas de estimación paramétrica. Se ha optado por definir una 'función de *likelihood*', o función de verosimilitud, y aplicar una estimación de máxima verosimilitud.

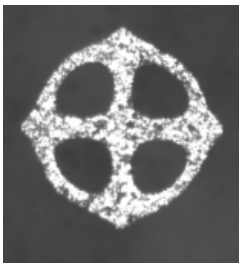

Figura 3.5: *Marca fiducial en un LGAD.*

[7] DOF: región del espacio imagen para el que se forma una imagen suficientemente nítida a través de un sistema óptico.

Likelihood

La función de verosimilitud $\mathcal{L}$ es una herramienta estadística que permite realizar estimaciones de los parámetros θ_j de un modelo probabilístico $y = f(x;\theta_j)$ partiendo de un conjunto de observaciones $\{y_i, x_i\}_{i=1,\dots,N}$ dadas por una distribución estadística [14]. En este sentido, es una generalización de la regresión por mínimos cuadrados, donde se asume que la distribución de las medidas es constante. Así pues, se va a utilizar el ejemplo de una regresión lineal $y = a + bx$ para entender cómo funciona este nuevo instrumento matemático.

Se parte de unos resultados y obtenidos en función de un modelo estadístico, $y = f(x, \theta_j)$, que dependen de unas variables x bien conocidas y de los parámetros del modelo θ_j. Tanto las variables x como los parámetros θ_j están fijos en la realidad del problema. Por ejemplo, si $y = a + bx$ mide la posición de un objeto con velocidad constante, está claro que su velocidad b, a y las medidas exógenas[8] de tiempo x son independientes y están fijos, mientras que las medidas endógenas[9] y dependerán de los tres. En este caso, la distribución que da las medidas podría ser una distribución Gaussiana de desviación estándar constante, dada por los errores aleatorios y de medida.

La función de verosimilitud da la vuelta al problema, e informa sobre la probabilidad de que los parámetros del modelo sean θ_j, habiendo medido unos determinados $\{y_i, x_i\}_{i=1,\dots,N}$. En problemas de variable continua y para una serie de medidas independientes, se define el *likelihood* según la Ec. (3.1):

$$\mathcal{L}(\theta_j) = \mathcal{L}(\theta_j\,; x_i, y_i) \;=\; \prod_{i=1}^{N} P(y_i \mid f(x_i, \theta_j)) \tag{3.1}$$

Así pues, la probabilidad de que los parámetros reales del modelo sean θ_j es igual al productorio, al ser medidas independientes, de las probabilidades de haber obtenido cada una de las medidas y_i a partir de x_i si los parámetro son θ_j.

Si se maximiza $\mathcal{L}$, se obtendrá la mejor estimación posible de los parámetros θ_j. Para el ejemplo del ajuste lineal, asumiendo un error Gaussiano en las medidas:

$$\mathcal{L}(a, b) \;= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2}\left(\frac{y_i - a x_i - b}{\sigma}\right)^2} \tag{3.2}$$

En la práctica, en vez de maximizar la función como aparece en la Ec. (3.2), se aprovecha que la función logaritmo es monótona y por tanto $max(f(x)) = max[ln(f(x))]$ para simplificar el problema. Y por convenio, se define y minimiza una nueva magnitud, q, el *log-likelihood*, Ec. (3.3):

$$q = -2\, ln(\mathcal{L}) \tag{3.3}$$

De este modo, en el caso particular lineal de la Ec. (3.2):

$$q = -2\, ln\, \mathcal{L} = -2 \sum_{i}^{N} ln \left[\frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2}\left(\frac{y_i - a x_i - b}{\sigma}\right)^2} \right] = -2N\, ln \left(\frac{1}{\sqrt{2\pi\sigma}} \right) + \sum_{i}^{N} \left(\frac{y_i - a x_i - b}{\sigma} \right)^2 \tag{3.4}$$

[8] Exógeno: de origen externo.
[9] Endógeno: originado por causas internas.

El primer término es constante, por lo que la minimización se reduce al segundo, el sumatorio, que se identificará como χ^2. Así pues, χ^2 es una medida que cuantifica la discrepancia entre los datos observados y_i y las predicciones del modelo $f(x_i, \theta)$, normalizada por la desviación estándar σ.

Por tanto, maximizar la verosimilitud (o minimizar el *log-likelihood* negativo) es equivalente a minimizar χ^2 cuando los errores son normales e independientes. Además, el valor final de χ^2 se puede utilizar como una medida de la calidad de los parámetros obtenidos.

3.3.2. Método experimental

Software control

El software que se ha utilizado y ampliado está implementado en lenguaje Python y consta de varios *scripts* que describen el sistema robot-cámara-mesa y su dinámica. El programa se ha ejecutado desde una interfaz CLI de Linux, para lo cual se ha instalado WSL (*Windows Subsystem for Linux*) en un equipo Windows, de esta forma se ha podido utilizar un intérprete de comandos *shell* sin necesidad de instalar el OS Linux.

Se partió de un código con clases para describir las principales partes del sistema y su dinámica. Estas son la mesa, el robot y la cámara.

En primer lugar, la mesa se simula como un plano sobre el cual se pueden definir multitud de puntos de referencia y sobre la que se fija el robot.

En segundo lugar, el sistema de visión se define con un modelo simple de cámara *pinhole*, por el cual los rayos de luz que llegan del objeto observado pasan por un punto focal y se proyectan sobre un plano, situado a una distancia igual a la focal del sistema. La cámara se posiciona con respecto al émbolo del robot según sus coordenadas iniciales $(x, y, z)_C$ y se orienta acorde a sus ángulos de Euler $(\psi, \theta, \phi)_C$, que definen su orientación unívocamente.

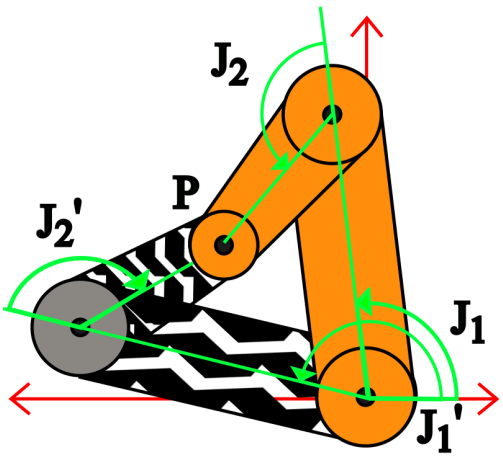

Figura 3.6: *Representación de cómo dos configuraciones* (J_1, J_2) *y* (J_1', J_2') *hacen que el robot apunte al punto* P.

El robot se simula acorde al modelo simplificado de la Fig. (3.1) e incluye funciones para mover sus brazos, moverse hasta puntos cartesianos del plano, realizar cambios de coordenadas entre el sistema cartesiano y propio, proyectar puntos sobre la cámara... El robot tiene asignada una mesa, sobre la cual está fijado, y una cámara que se moverá solidariamente con el mismo. Además, el programa incluye los valores reales de su precisión y repetibilidad.

El código se depuró y amplió incluyendo funciones para facilitar la operación de la cámara, informando sobre a qué posición de la mesa apunta en todo momento y permitiendo apuntar con la cámara a puntos sobre la mesa. Además, en la segunda función, el usuario tiene la libertad de seleccionar con qué ángulo J_z quiere 'aproximarse' al punto, ofreciendo de esta forma infinitas po-

sibilidades para observar un mismo punto.

Igualmente, ha sido necesario definir correctamente el movimiento del robot. Ya que, dado el sistema interno de un robot SCARA, se puede alcanzar un mismo punto del plano de la mesa utilizando dos configuraciones de ángulos J_1 y J_2, ver Fig. (3.6). Una vez analizados los diferentes posibles movimientos que puede realizar el robot, se llegó a la conclusión de que como condición de univocidad, cuando haya más de una posible configuración, se tomará la solución con menor $|J_1|_{min}$ siempre. Esta, además, contribuye a la estabilidad del robot, al reducir el movimiento del eje J_1, que afecta a los dos brazos y cuyas rotaciones tendrán un mayor momento angular que las del eje J_2, que solo mueve el segundo brazo.

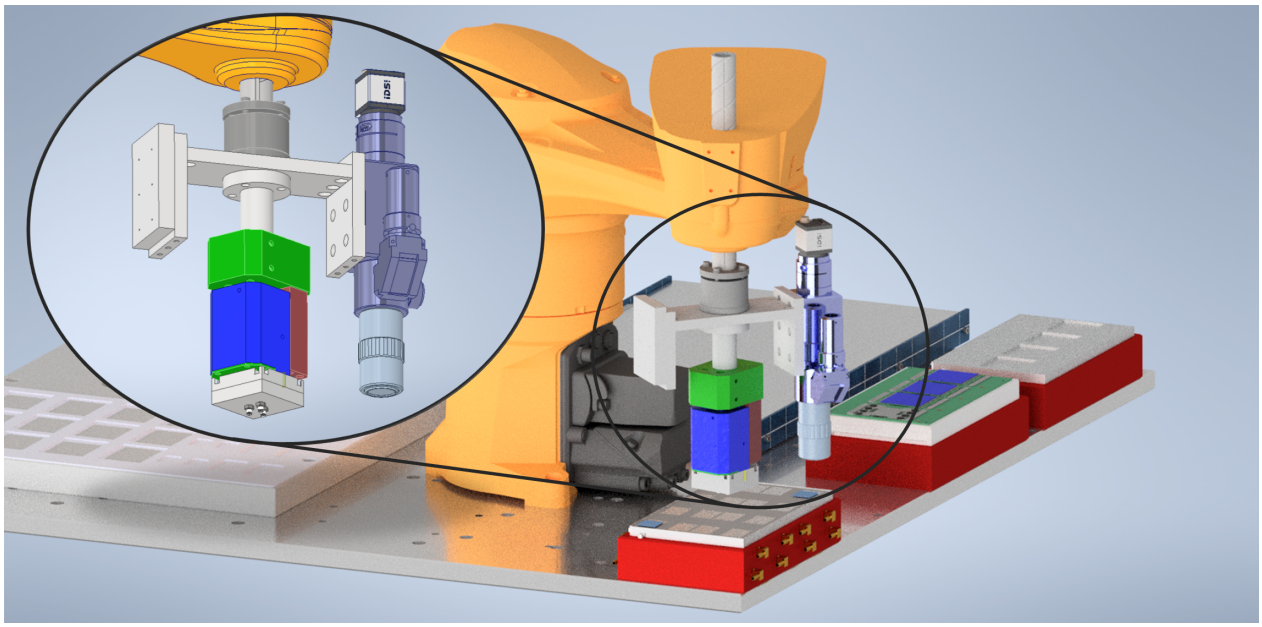

Figura 3.7: *Imagen CAD renderizada del robot SCARA y la mesa de trabajo. Detalle mostrando como el sistema óptico con cámara se acoplará al émbolo del robot (tal y como se ha simulado en el software). Imágenes cedidas por David Moya Martín.*

Likelihood en software

Para aplicar la función de verosimilitud se recurrió a un modelo general *GenericLikelihoodModel* de la librería de Python *statsmodels* [15].

El modelo funciona con dos listas, Ec. (3.5), una de magnitudes exógenas: las coordenadas de los puntos medidos, y otra con las magnitudes endógenas, las medidas del robot real para cada punto: sus coordenadas internas y la proyección del punto sobre el detector de la cámara. Además, el modelo tiene acceso a un robot estimación (con una cámara con parámetros estimados $(x, y, z, \psi, \theta, \phi)^{est.}_{C}$) con el que puede simular el método de medida.

$$\begin{aligned} exog &= [x_1, y_1, z_1, x_2, y_2, z_2, ...] \\ endog &= [(J_1, J_2, z, J_z)_1, (X, Y)_1, (J_1, J_2, z, J_z)_2, (X, Y)_2, ...] \end{aligned} \tag{3.5}$$

Resulta importante determinar cuales de los parámetros de la cámara $\theta_j = (x, y, z, \psi, \theta, \phi)^{0}_{C}$ se van a calibrar. Se realizaron múltiples funciones: una que optimiza todos a la vez, otra que solo optimi-

za las posiciones cartesianas, otra los ángulos, otra las coordenadas cartesianas (x, y), y múltiples combinaciones entre coordenadas y posiciones. La estructura general de las funciones que se han escrito corresponden con la del Cód. (1), en el Anexo .3.

Una vez definido el modelo, se aplica el método de minimización *Powell* a la función de *log-likliehood*. Con unas serie de evaluaciones a la función, se obtiene un nuevo set de parámetros $\theta_j^{est.2}$ que se aproximan más a los reales θ_j^0 que θ_j^{est}. Iterando el proceso para asegurar localizar el mínimo global de la función, se obtendrá la mejor estimación posible de los parámetros.

Método Calibración

Entonces, se propone un protocolo para calibrar la posición y orientación de la cámara. Este consiste en una toma de medidas con el robot real, seguida de una simulación donde se buscará replicar la toma de medidas real. Partiendo de una estimación de los parámetros de la cámara $(x, y, z, \psi, \theta, \phi)_C^{est.}$, estos se irán optimizando hasta ajustarse a los reales. Para mayor claridad, la calibración consistirá en los siguiente pasos:

1. Operando el robot SCARA real:
   - *a*) Se opera el robot para moverlo a las posiciones aproximadas desde las cuales se pueden fotografiar los puntos de calibración de la mesa. Estos puntos son marcas circulares de diámetro $\varnothing = 0.9$ mm, con coordenadas $(x, y, z)^{3D}$ bien conocidas sobre la mesa.
   - *b*) Se "accederá"a cada uno de los puntos de calibración con diferentes configuraciones $(J_1, J_2, z, J_z)_i$, utilizando múltiples $(n = 5)$ valores de J_z en cada caso, y se realizará una fotografía de su posición para cada configuración. Se registran $(J_1, J_2, z, J_z)_i$ y la proyección del punto sobre el sensor de la cámara $(X, Y)_i$, además de las coordenadas reales del punto $(x, y, z)_i^{3D}$.
2. Calibración mediante un SCARA simulado:
   - *a*) Se genera un robot y mesa de igual geometría y se le añade una cámara. Esta cámara se colocará en una posición y orientación inicial $(x, y, z, \psi, \theta, \phi)_C^{est.}$, que será la estimación inicial de las coordenadas de la cámara real.
   - *b*) Se hace que el robot virtual se mueva exactamente a cada una de las configuraciones $(J_1, J_2, z, J_z)_i$ que se guardaron previamente.
   - *c*) En esa posición, se calcula la proyección del punto $(x, y, z)_i^{3D}$ en el sensor de la cámara $(X, Y)_i'$.
   - *d*) Al depender de los parámetros de orientación de la cámara $(x, y, z, \psi, \theta, \phi)_{camera}^{est.}$, la proyección no coincidirá.
   - *e*) Se calcula la diferencia entre las proyecciones real $(X, Y)_i$ y simulada $(X, Y)_i'$, y a partir de estas se recalcula una mejor estimación de los parámetros de la cámara $(x, y, z, \psi, \theta, \phi)_C^{est.2}$.
3. Se itera múltiples veces el proceso (2) con el robot simulado, mejorando la estimación de los parámetros hasta alcanzar un mínimo del *log-likelihood*.

Antes de implementar este método para calibrar el robot del laboratorio, se ha validado con una simulación completa. Es decir, se ha simulado el robot real $robot_1$ (pasos 1) y se ha realizado una toma de medidas inicial, y a continuación se ha calibrado acorde a (pasos 2) partiendo de otro robot con una estimación de la orientación de la cámara $robot_2$. Este procedimiento tiene la ventaja de

que permite contrastar la validez de este modelo al comparar los resultados con los valores que se han asignado al $robot_1$.

Los pasos (2) están englobados dentro de la función *likelihood*, esta recalcula en cada iteración unos nuevos parámetros para la cámara buscando minimizar el valor del *log-likelihood*.

Para realizar un análisis cuantitativo de la validez de la calibración, partiendo de un $robot_1$ real se optimizará el $robot_2$ con un set de puntos de la mesa de calibración. Luego, con un set de puntos de prueba distintos se hará que ambos robots se muevan con exactamente la mismas coordenadas internas $(J_1, J_2, z, J_z)_i$ sobre cada uno de los puntos y se comparará la proyección del punto en sus sensores $(X, Y)'_i$ y $(X, Y)'_i$. Calculando las diferencias $\delta x_{CMOS} = (X - X')$ y $\delta y_{CMOS} = (Y - Y')$ y proyectándolo inversamente desde la cámara sobre el plano mesa, se podrá estimar la precisión y repetibilidad alcanzada al calibrar el robot.

3.3.3. Estimaciones iniciales: resultados y análisis

Tras simular la toma de medidas con una cámara real ligeramente inclinada y colocada con un desplazamiento respecto al robot, se trató de optimizar los parámetros de una segunda cámara para lograr conocer los parámetros $(x, y, z, \psi, \theta, \phi)_C$ de la real. En función de qué parámetros se trataron de estimar se llegó a resultados bien diferenciados, los cuales llevaron a unas conclusiones iniciales que se presentan a continuación.

Se probó a calibrar diferentes combinaciones de los parámetros de la cámara $(x, y, z, \psi, \theta, \phi)_C$. En la Tab. (3.1) se recogen varios ejemplos de optimización. Se incluyen los parámetros $(x, y, z, \psi, \theta, \phi)$ del robot real $robot_1$, los parámetros para el robot estimación inicial $robot_2$ al comenzar la optimización, se especifica que parámetros se han calibrado y su valor tras la calibración. Asimismo, se incluye el valor total, no normalizado al número de puntos de calibración, de χ^2.

Estas calibraciones se realizaron fotografiando 80 puntos de calibración de la mesa, con 5 orientaciones J_z del robot en cada uno (un total de 400).

Tabla 3.1: *Ejemplos de resultados para calibraciones de diferentes combinaciones de parámetros de posición y orientación de la cámara por separado, etiquetadas con N. En la primera columna se recogen los parámetros $(x, y, z, \psi, \theta, \phi)$ del robot real y en la segunda la estimación inicial para la optimización. La columna "Var. Cal." indica las variables que se optimizan y "Val. Cal." su resultado final tras la calibración. La última columna informa sobre el valor total de la función χ^2 al finalizar la minimización.*

	$(x, y, z, \psi, \theta, \phi)$/(cm, rad)		(cm, rad)		
N	$robot_1$	$robot_2$	*Var. Cal.*	*Val. Cal.*	χ^2
1	(5,0,3.6,0.03,0.02,-0.04)	(7,2,5,0,0,0)	Todas	(5.46, -0.25, 6.51, 0.02, -0.04, -1.09)	0.012
2	(5,0,3.6,-0.01,0.04,0.03)	(3,1,5,-0.01,0.04,0.03)	(x, y, z)	(5.00,0.002, 5.16)	0.017
3	(5,0,3.6,0.03,0.02,0)	(5,0,3.6,0,0,0)	(ψ, θ, ϕ)	(-0.041, 0.020, -0.003)	0.012
4	(5,0,3.6,0.2,0,0)	(5,0,3.6,0,0,0)	(x, y, ϕ)	(5.00, 0.00, 0.20)	0.013
5	(5,0,3.6,0.2,0.2,0)	(6,-1,3.6,0,0,0)	(x, y, ψ, θ)	(5.000, 0.014, 0.003, 0.001)	17

En primer lugar se trató de optimizar los seis parámetros de la cámara $(x, y, z, \psi, \theta, \phi)_C$, calibración de la Tab. (3.1, 1). No obstante, no se llegaba a los resultados esperados. Los parámetros alcanzados

no eran los del $robot_1$ si bien la función de *likelihood* sí se minimizó. Los valores de $(x, y, z, \psi, \theta, \phi)_C$ presentan desviaciones de entre el $\delta x_C = 23\,\%$ y $\delta\theta_C = 50\,\%$. Al ser los ángulos de inclinación reducidos, calibrar z_C resulta complicado por este método. Esta coordenada de la cámara se podrá conocer de una forma muy precisa enfocando objetos una vez esté montada la cámara en el robot, tomando ventaja de la reducida profundidad de foco de la cámara. Dada la repetibilidad del robot en el eje z, esta coordenada se podría calibrar con una precisión de hasta $\sigma(z)_C = 4\ \mu$m.

En la calibración Tab. (3.1, 2) se observa como partiendo de valores distantes en uno o más centímetros de los del robot real, resulta posible calibrar sus coordenadas $(x, y, z)_C$. Aún teniendo cierta rotación, hasta $\theta_C = 0.04$ rad $= 2.3°$, la función de Likelihood se logra minimizar a un valor total muy reducido.

Luego, se probó a calibrar los ángulos únicamente, suponiendo que la posición es $(x, y, z)_C$ conocida, Tab. (3.1, 3). Se vio que en general sus valores finales resultan erráticos, siendo θ_C el que menores diferencias con el real presentaba, que resulta compatible.

Buscando entender la dependencia de los ángulos y porqué resultaba complicado calibrar su valor, se probó a calibrar diferentes combinaciones de las coordenadas cartesianas de la cámara (x, y) con una o varias coordenadas angulares ψ, θ o ϕ, como en Tab. (3.1, 4 y 5).

En general se observó que es posible calibrar la posición aproximadamente, pero la orientación suele ser errática. En las calibración Tab. (3.1, 5), se llega al valor exacto de x, mientras que los ángulos ψ y θ toman valores incorrectos e y se desvía ligeramente del valor real esperado. Por último, fue especialmente notorio el valor de la calibración Tab. (3.1, 4) , donde obteniendo un $\chi^2 = 0.013$ bajo, el valor del ángulo ϕ calibrado es igual al del ángulo ψ del robot original $robot_1$.

3.3.4. Discusión de las estimaciones y resultados iniciales

Llegado este punto, se ha visto que en general los ángulos se calibran mal, mientras que las posiciones se acercan a los valores reales y se obtienen valores bajos de χ^2. Los patrones vistos en las tablas de resultados y otras muchas simulaciones ejecutadas llevaron a la formulación de dos hipótesis:

- **A)** Degeneración entre ángulos y posiciones. En consecuencia a realizar todas las fotografías a la misma distancia de focalización de los puntos, una traslación en la posición de la cámara puede compensar una rotación en la misma. Esto hace que la calibración de posiciones y ángulo sea errática, tal y como se ha visto.
- **B)** El sistema de ángulos empleado es problemático. Los ángulos de Euler empleados no identifican unívocamente la orientación de la cámara, dificultando el proceso de optimización para calibrar su orientación.

Se discute primero la hipótesis **B**. La simulación cuenta con un sistema de tres rotaciones para orientar los cuerpos (ψ, θ, ϕ), estas se aplican como las tres matrices de rotación de la Ec. (3.6).

$$B = \begin{pmatrix} cos(\psi) & sin(\psi) & 0 \\ -sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad , \quad C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & sin(\theta) \\ 0 & -sin(\theta) & cos(\theta) \end{pmatrix} \quad y \quad D = \begin{pmatrix} cos(\phi) & sin(\phi) & 0 \\ -sin(\phi) & cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.6}$$

Por lo consiguiente, el ángulo θ da una rotación $C(\theta)$ en torno al eje x cartesiano, mientras que ψ y ϕ se aplican como rotaciones $B(\psi)$ y $D(\phi)$ respectivamente, ambas respecto al eje z. Estas rotaciones se aplican a los cuerpos en el orden indicado por la Ec. (3.7), es decir, se ejecuta una rotación ϕ en el eje z del cuerpo, seguida de una rotación θ sobre el eje x para acabar con una rotación de ψ en el eje z.

$$rot = B_z(\psi) \cdot (C_x(\theta) \cdot D_z(\phi)) \tag{3.7}$$

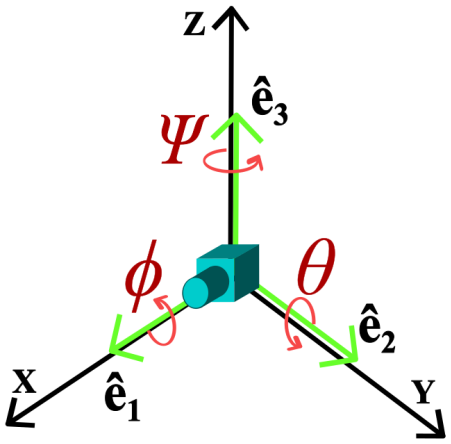

Figura 3.8: *Esquema de los ángulos de Tait-Bryan de roll (ψ), pitch (θ) y yaw (ϕ). En la situación real, el eje óptico de la cámara está orientado en el eje $\vec{e}_3$.*

El sistema clásico para orientar un sólido rígido, conocido como ángulos de Tait-Bryan, consiste en tres rotaciones en los diferentes ejes cartesianos del cuerpo $(e_1(x), e_2(y), e_3(z))$: *roll* (ψ), *pitch* (θ) y *yaw* (ϕ), ver Fig. (3.8). Las rotaciones se aplican en un determinado orden: $R(e_3), R(e_2)$ y $R(e_1)$. Si el ángulo de *pitch* es $\theta = \pm 90°$, los ejes de *roll* y *yaw* coincidirán. Esto se conoce como el problema de *Gimball-Lock* (bloqueo de cardán), en esta situación ψ y ϕ son equivalentes, el sistema está degenerado y habrá infinitas rotaciones compatibles con la misma orientación. No solo esto, el sistema ene general no es unívoco y siempre hay al menos dos soluciones [16].

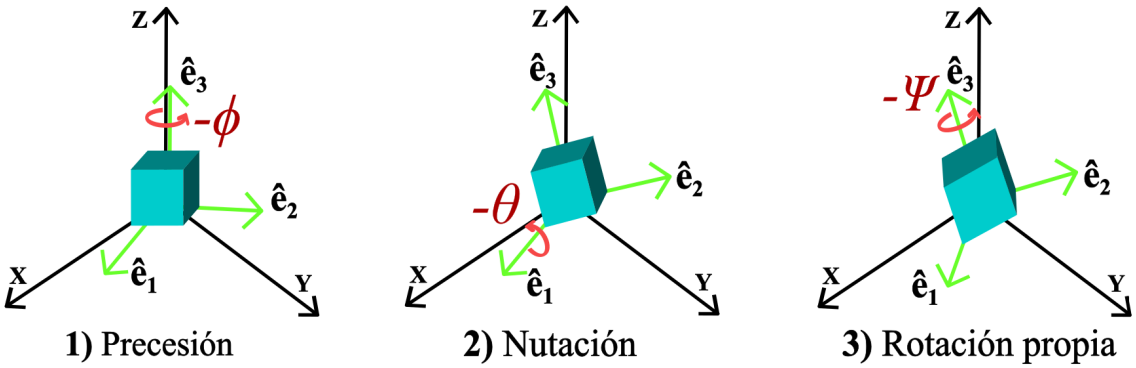

Figura 3.9: *Aplicación de los ángulos de Euler acorde a como se han definido en el problema:* 1° *precesión* ϕ, 2° *nutación* θ *y* 3° *rotación propia* ψ.

No obstante, en este trabajo se han empleado los ángulos de Euler, ampliamente utilizados en mecánica cuántica, que consisten en tres giros $R(e_3), R(e_2)$ y $R(e_1)$, con ángulos de precesión ϕ, nutación θ y rotación propia ψ. En la Fig. (3.9) se aplican los tres ángulos como se ha descrito en la Ec. (3.7). Este sistema, como se ha visto tratando de optimizar los ángulos, también tiene una casuística peculiar cuando el ángulo de nutación es nulo $\theta = 0$. En esta situación la solución es singular, los ángulos ψ y ϕ son equivalentes al ser sus ejes de rotación paralelos. Esto resulta un problema cuando el ángulo θ está por definición en torno a $\theta \approx 0°$ si la cámara está bien alineada, que es lo buscado. Por otro lado, tampoco es unívoco en general, al existir dos soluciones para $\theta = arccos(rot_{3,3})$.

Por tanto, sería conveniente definir restricciones al sistema de ángulos si se va a calibrar estos. De lo contrario, calibrar θ en torno a cero, lleva a la existencia de múltiples mínimos en la función de coste.

Se discute ahora la hipótesis **A**, esta surge a raíz de ver como en algunos casos, si bien no se llega a igualar los parámetros de la cámara real, el ajuste es aparentemente bueno al presentar un valor bajo de χ^2. Recordar que χ^2 cuantifica la discrepancia entre los datos observados y_i y las predicciones del modelo $f(x_i, \theta)$, por lo que si se consigue un $\chi^2 << 1$, o se ha llegado a los parámetros θ reales o a unos equivalentes.

Así es el caso de la calibración 2 de la Tab. (3.1), donde aún solo calibrando la posición de la cámara (que tiene cierta inclinación), se logra obtener valores tan bajos como $\chi^2/N = 4 \cdot 10^{-6}$ cm.

Se realizó un estudio en detalle del problema en dos dimensiones. Se llegó a la conclusión de que, si todas las medidas se realizan a la misma altura z_C, lo cual es un requisito dado que la distancia de la cámara es fija, será equivalente una inclinación de la cámara θ a un desplazamiento en y. Si bien ambas cámaras en la Fig. (3.10) tienen posiciones y orientación distinta, las medidas que pueden hacer son equivalentes para el robot con el correcto calibrado.

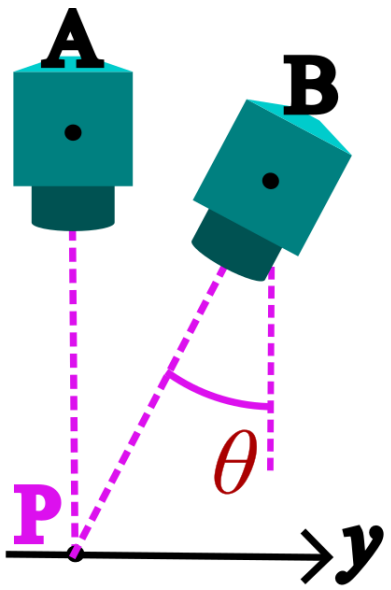

Figura 3.10: *Diagrama esquematizado demostrando como dos configuraciones de la cámara diferentes realizan medidas equivalentes sobre los puntos de la mesa, si siempre se trabaja a la misma altura.*

Esta relación ángulo-desplazamiento es fácilmente generalizable al espacio 3D. Si bien se puede realizar rotaciones de múltiples ejes (J_1, J_2, J_Z), dado que la cámara y el extremo del émbolo del robot (respecto al cual está definido $(x, y, z, \psi, \theta, \phi)_C$) se mueven solidariamente, el ángulo clásico de *roll* será equivalente a un desplazamiento en y_C y el de *pitch* a un desplazamiento en y_C. Así pues, para una misma altura z, sí hay degeneración entre ángulos θ y posición y. Para los ángulos utilizados en el software, la relación es directa entre θ y la coordenada y_C.

Una vez discutidas las hipótesis **A** y **B**, se decide cuales de los parámetros de la cámara se deben estudiar para obtener una buena calibración. Se opta por optimizar mediante software únicamente $(x, y)_C$, mientras que z_C podrá ser calibrada conociendo la distancia de enfoque del microscopio. De esta forma, se conocerá una posición de la cámara equivalente a la posición real, con inclinación $\psi, \theta, \phi = 0$. Independientemente de la inclinación del microscopio, se tendrá una función mediante la cual, dada la posición del robot (J_1, J_2, J_z) y la proyección de un punto en la cámara (X, Y), se conocerá la posición (x, y, z) exacta del punto observado.

3.3.5. Resultados y análisis

Tras decidir optimizar únicamente los parámetros $(x, y)_C$, se decidió definir varios escenarios realistas y exagerados para probar la validez de este método de calibración.

Se definieron diferentes escenarios iniciales, Tab. (3.2), con la cámara del $robot_1$ real mejor ($|\psi, \theta, \phi|$ bajos) y peor orientada ($|\psi, \theta, \phi|$ altos), y con valores de la estimación inicial del $robot_2$ más y menos próximos a los reales.

Tabla 3.2: *Calibraciones con 400 puntos (80 puntos de la mesa con 5 orientaciones J_z) de los parámetros $(x, y)_C$ de la cámara. En la primera columna se recogen los parámetros $(x, y, z, \psi, \theta, \phi)$ del robot real y en la segunda la estimación inicial para la optimización. La columna "Valores Calibración" indica el resultado final de $(x, y)_C$ tras la calibración. La última columna informa sobre el valor de la función de coste al finalizar la minimización. Los errores son calculados por el software al ejecutar la minimización Powell.*

	$(x, y, z, \psi, \theta, \phi)$/(cm)		(cm)	
N	$robot_1$	$robot_2$	Valores Calibración $(x \pm \delta x,\ y \pm \delta y)_C$	χ^2
1	$(5, 0, 10, 0, 0, 0)$	$(10, -9, 10, 0, 0, 0)$	$(5.0001 \pm 0.0001, -0.0001 \pm 0.0001)$	0.047
2	$(5, 0, 10, 1.5°, -2°, 0.7°)$	$(10, -7, 10, 0, 0, 0)$	$(5.000 \pm 0.011, 0.0626 \pm 0.0010)$	0.132
3	$(5, 0, 10, 10°, -7°, -8.5°)$	$(10, -5, 10, 0, 0, 0)$	$(4.968 \pm 0.007, -0.2169 \pm 0.0010)$	0.138
4	$(5, 0, 10, 1.5°, -2°, 0.7°)$	$(20, 10, 10, 0, 0, 0)$	$(5.001 \pm 0.009, -0.0628 \pm 0.0010)$	0.127
5	$(5, 0, 10, 10°, -7°, -8.5°)$	$(40, 40, 10, 0, 0, 0)$	$(4.967 \pm 0.011, -0.2170 \pm 0.0010)$	0.135
6	$(5, 0, 10, 0.5°, -1°, 0.2°)$	$(5.5, -0.4, 10, 0, 0, 0)$	$(5.000 \pm 0.009, 0.0315 \pm 0.0010)$	0.122

Además de los valores de χ^2, los datos de calibración se procesaron para obtener otras métricas de la calidad del ajuste. En concreto, tomando el robot real y el robot calibrado, se comparó la desviación de sus medidas δx e δy sobre la mesa. De este modo, se fotografió un set de puntos de *test* de la mesa (200 puntos distintos a los de calibración) con el $robot_1$ y se registró la posición interna del robot (J_1, J_2, J_Z, z). Luego se utilizó el $robot_2$ y se movió a las mismas posiciones, comparando las proyecciones de cada punto sobre la cámara de los dos robots, y proyectando su diferencia sobre la mesa.

A partir de estas medidas se obtuvieron dos histogramas para cada una de las seis calibraciones de la Tab.(3.2), ver las figuras Fig. (7 y 8) del Anexo .4. A cada uno de los histogramas, se le practicó un ajuste a una distribución Gaussiana, obteniendo su valor promedio μ y su dispersión en forma de la desviación estándar σ. Los valores menos favorables obtenidos se recogen en la Tab. (3.3).

Tabla 3.3: *Colección de los peores valores obtenidos, para el valor medio μ y la desviación estándar σ, de la desviación de las medidas de un robot con los parámetros del calibrado y el robot real con x_C e y_C. Se etiquetan con la calibración N de la Tab. (3.2) a la que corresponden.*

Valor menos favorable para:		$N_{cal.}$
$\mu(\delta x)$ / μm	-3	5
$\mu(\delta y)$ / μm	5	3
$\sigma(\delta x)$ / μm	26	4
$\sigma(\delta y)$ / μm	38	2

Después de obtener estos resultados, se decidió realizar una medida más realista, utilizando un número de puntos de calibración comparable al disponible en la mesa real. En concreto, se utilizaron 10 aleatoriamente entre los puntos de calibración de la mesa, y para la producción de los histogramas de dispersión se emplearon 400 puntos de prueba, buscando una alta estadística para el ajuste Gaussiano. En la Fig. (6) del Anexo .4, se puede ver la posición de los puntos de calibración y prueba sobre la mesa.

Se fijaron unos parámetros de la cámara y se dieron valores iniciales con un margen de error similar al que se tendrá en el robot real (tolerancias de las piezas respecto al diseño ideal CAD). La optimización se ejecutó múltiples veces partiendo de los mismos datos, llegando al mismo resultado,

recogido en la Tab. (3.4).

Tabla 3.4: *Resultados para varias calibraciones con 10 puntos de calibración (5 posiciones J_Z en cada uno). Se recogen los parámetros reales $robot_1$, la estimación inicial para la calibración $robot_2$ y los valores finales calibrados para x_C e y_C. Igualmente se incluye el valor final de χ^2.*

$robot_1(x, y, z, \psi, \theta, \phi) = (5.123\text{ cm}, 0.078\text{ cm}, 10\text{ cm}, 1.70°, -0.48°, 0.96°)$	
$robot_2(x, y, z, \psi, \theta, \phi) = (5\text{ cm}, 0\text{ cm}, 10\text{ cm}, 0°, 0°, 0°)$	
Valores Calibración $(x, y)_C$/cm	χ^2
$(5.126 \pm 0.011, -0.048 \pm 0.001)$	0.03

Se realizó un histograma de dispersión con esta calibración, Fig. (3.11).

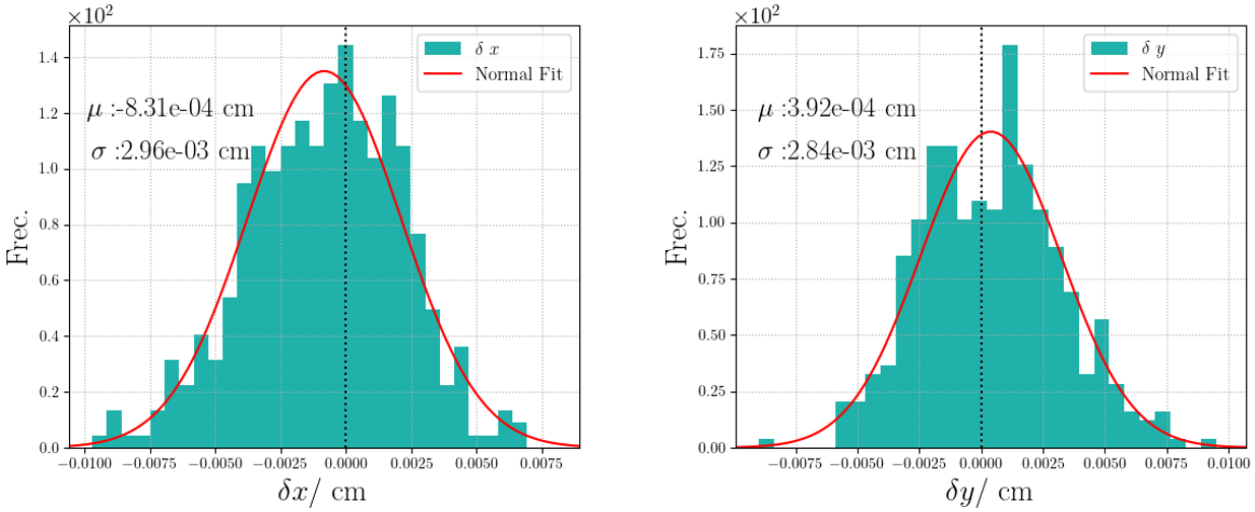

Figura 3.11: *Histograma de dispersión de la diferencia de las medidas realizadas con un robot real simulado con sus parámetros reales y los calibrados mediante el sistema desarrollado. Corresponde a la calibración de la Tab. (3.4), se representa la desviación en el eje $\vec{x}_C$ e $\vec{y}_C$ de la cámara. En cada uno, se representa el ajuste a una distribución Gaussiana, y se señala el valor medio (μ) y desviación estándar del ajuste σ.*

3.3.6. Discusión

Con el fin de poder extraer conclusiones de los resultados presentados en la anterior sección, es necesario compararlos y discutir su validez en base a lo esperado.

Las calibraciones de la Tab. (3.2), incluyen casos conservadores como el 1, donde la cámara real está perfectamente alineada y casos muy exagerados como el 5 donde los ángulos de inclinación de la cámara rondan los $\alpha = 10°$ y los valores iniciales de la posición son un orden de magnitud mayores a los reales. Analizando el valor final de χ^2/N, todos los ajustes parecen ser de calidad, sin importar si los ángulos son más o menos exagerados. Por esta razón, calibrar únicamente $(x, y)_C$ parece funcionar.

Si se analizan las figuras de los gráficos de dispersión del Anexo .4, Fig. (7 y 8), se puede comprobar visualmente como la dispersión se ajusta a una curva Gaussiana. Dado que el ajuste no es por

mínimos cuadrados, sino que se ajusta a una densidad de probabilidad, no se obtienen parámetros de índice de regresión. Como μ y σ ya se están empleando para caracterizar la desviación de x e y, tener sus errores no sería útil más allá que para verificar la calidad del ajuste a la distribución Gaussiana, que no es lo buscado.

Observando directamente los peores valores σ y μ de estas distribuciones, Tab.(3.2), se comprueba como los resultados para la desviación son inferiores o iguales siempre a $\mu_{max} = 5$ μm. La desviación llega a tomar un valor de $\sigma_{max} = 38$ μm, cuatro veces la repetibilidad del robot ($\delta(x,y)_{robot} = 10$ μm), pero que cumple con el estándar esperado para el montaje de módulos LGAD $\mathcal{O} = 50$ μm.

En el caso de las calibraciones con 10 puntos ($10 \times 5 J_Z$), se alcanza un valor $\chi^2/N = \chi^2/50 = 6 \cdot 10^{-4}$ cm. Para valorar mejor los resultados se han realizado los histogramas, cuya información es más descriptiva.
Como se puede ver en la Fig. (3.11), la desviación media entre ambas cámaras es de tan solo $\mu(\delta x) = 8$ μm y $\mu(\delta y) = 4$ μm. Su desviación estándar $\sigma(\delta x) = 30$ μm y $\sigma(\delta y) = 28$ μm, inferior a $\mathcal{O} = 50$ μm sigue cumpliendo la precisión necesaria para la producción de módulos. A pesar de contar con solo 10 puntos físicos de calibración, el resultado es satisfactorio.

Cabe destacar que no se aprecia un sesgo entre las coordenadas x e y, ambas parecen calibrarse con la misma calidad aproximadamente, ninguna está favorecida en general. Sin embargo, en el gráfico de la Fig. (7, Cal. 2) se puede ver como hay un valor anómalo en la distribución Gaussiana, con una desviación de $\delta y^1 \approx 400$ μm. De producirse este tipo de error aleatorio al estar montando módulos reales, el módulo en cuestión resultaría inutilizable en el detector real. No obstante, esto es algo que se espera y a lo que los equipos de producción de módulos están acostumbrados. Por esta razón, antes de soldar las conexiones en los mismos, se realizan medidas de los módulos una vez montados y se registra su desviación en una base de datos. En ese caso los módulos defectuosos pasarían a ser *dummies* y podrían emplearse para test de resistencia o con fines educativos, pero nunca en el ETL.

3.4. Conclusiones

En conclusión, una cámara acoplada en el émbolo extensible de una robot SCARA puede ser calibrada en función de tan solo sus dos coordenadas $(x,y)_C$ con respecto al extremo del émbolo, si todas las fotografías se van a realizar a la misma altura. De manera que se puede prescindir de considerar sus desviaciones angulares, si bien será conveniente reducirlas al máximo en el diseño del acople y montaje del miscroscopio en el robot.

Gracias a estas simulaciones, se ha comprobado que con tan solo 10 puntos de calibración sobre la mesa, tomando ventaja de realizar medidas con diferentes ángulos de aproximación J_Z, se puede hacer una buena calibración de los parámetros de la cámara mediante una minimización de *log-likelihood*. Si bien los parámetros que se estiman no son los reales de la cámara, pues se hace la asunción de que está perfectamente alineada, al ser equivalentes permiten realizar medidas de la posición de puntos con gran precisión. En concreto, se ha obtenido una precisión con desviación estándar no superior a $\sigma(x,y)_C = 40$ μm, entre el robot real y el hipotético robot con los parámetros calibrados. Para la coordenada z_C de la cámara, se podrá utilizar para su calibración el propio control de altura del robot y el enfoque de la cámara, con una precisión de hasta $\sigma(z)_C = 4$ μm, o la

profundidad de foco del sistema óptico. Esto se traduce parcialmente a la precisión en la capacidad de conocer las coordenadas de un punto sobre la mesa, que es la función del microscopio.

Finalmente, podría considerarse que $\sigma(x, y)_C = 40$ μm es un valor demasiado cercano al objetivo de tolerancia en los módulos $\mathcal{O} = 50$ μm. Sin embargo, hay otros factores a tener en cuenta más allá del funcionamiento de los LGAD individuales. Como se ha introducido, el detector CMS, con una longitud de $l = 21$ m, cuenta con una enorme solenoide, capaz de generar campos magnéticos de $B = 4.4$ T. Este campo es capaz de encoger el detector en $\Delta l = 5$ cm sobre su eje principal. El campo tiene un efecto también en la forma de los *end caps*, que se curvan, especialmente el ETL y otros componentes que se encuentran envueltos por el solenoide. Este efecto puede producir desviaciones de hasta $\delta = 5$ mm en la posición de los detectores, varios ordenes de magnitud superiores a $\mathcal{O}$. Y es que, dado el tamaño relativamente grande de los píxeles de los detectores LGAD, la mayor restricción para su montaje viene dada realmente por el hecho de que se deben realizar conexiones de precisión con aparatos de soldadura de cable. Como esta tarea se realiza de forma automatizada, en estaciones de *wire bonding*, resulta muy conveniente mantener constante y dentro de unos márgenes la distancia entre las distintas partes, que para estos fines puede rondar las $\mathcal{O} = 100$ μm.

Finalmente, esta simulación concluye que la calibración resulta apta para que el robot SCARA pueda producir módulos LGAD para el detector CMS.

El código utilizado para la calibración y la simulación del robot se puede encontrar en el repositorio de GitHub [17] y el código dedicado al control de la cámara IDS y tratado de imagen en el repositorio [18].

4 Algoritmos de reconstrucción

Esta parte del trabajo se ha centrado en la utilización de herramientas de análisis de datos y en la aplicación de un algoritmo basado en *machine learning*, con el fin de reconstruir imágenes obtenidas con una simulación del dispositivo PROTECT.

Se ha partido de un programa en Geant4, facilitado por el director, donde se simula la física y geometría del dispositivo de tomografía protónica PROTECT. Después de ejecutar el programa con multitud de ejemplos y diferentes parámetros para entender su funcionamiento, y una vez detectadas una cantidad considerable de trazas de protones ($n >> 100$), se ha programado e implementado un algoritmo para generar imágenes a partir de las trazas y un programa para generar imágenes de los objetos colocado en el centro del *gantry*. Con estos programas y con los datos obtenidos utilizando la simulación de Geant4, se ha entrenado una red convolucional, en particular una U-Net.

El objetivo principal es crear un programa de entrenamiento para una U-Net, de forma que, la red sea capaz de recibir como argumento la imagen calculada a partir de las trazas detectadas con sensores LGAD de un haz de protones y producir una imagen de atenuación del cuerpo que ha atravesado el haz. En vez de el cuerpo de un paciente, se ha trabajado con fantomas.

4.1. Introducción:

Una red neuronal convolucional U-Net es una herramienta de *machine learning* especializada en el tratamiento de imágenes. Para entender su funcionamiento, conviene explicar primero otra herramienta de aprendizaje automático, una de las redes neuronales más simples: el perceptrón multicapa.

4.1.1. Redes neuronales: *multilayer perceptron*

Una red neuronal de tipo *multilayer perceptron*, recibe unos parámetros de entrada $\vec{x}$ conteniendo características o parámetros de un objeto y, tras ejecutarse, da unos parámetros de salida $\vec{\mathcal{O}}$ inferidos mediante cálculos matriciales sobre $\vec{x}$.

Su estructura interna se describe como una serie de capas, cada una con un determinado número N de neuronas. Cada neurona recibe como *input* un vector $\vec{x} = [x_1, x_2, ..., x_N, 1]$, conteniendo los *outputs* de las neuronas de la capa anterior y un parámetro constante distinto de cero. La neurona cuenta con un vector de pesos $\vec{w}$ correspondientes a cada una de las neuronas de la capa previa y un peso adicional llamado *bias* w_0, propio de cada neurona, que se multiplican por los x_i y la constante 1 del vector $\vec{x}$ respectivamente. Después de realizar este producto escalar entre vectores, se pasa el resultado a una función de activación $\sigma(x)$. En general, esta es una función no lineal y con una derivada rápida de calcular (suele utilizarse la función sigmoide con $d\sigma/dx = \sigma(1-\sigma)$), esta función permite al sistema de capas adaptarse también a problemas no lineales. El producto descrito, para una capa, se calcula como la operación matricial Ec. (4.1), donde W_N es la matriz de pesos w_{ij} (el peso j de la neurona i), y $\vec{x}$ es el *input* de la red o el resultado de la capa anterior.

$$W_N \cdot \vec{x} = \begin{pmatrix} w_{11} & w_{12} & \dots & w_1^0 \\ w_{21} & w_{22} & \dots & w_2^0 \\ \vdots & \vdots & \ddots & \ddots \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \end{pmatrix} = \begin{pmatrix} \vec{w}_1 \cdot \vec{x} \\ \vec{w}_2 \cdot \vec{x} \\ \vdots \end{pmatrix} \tag{4.1}$$

Por tanto, el *output* de las neuronas de la capa N, $\vec{\mathcal{O}}_N$, se puede expresar en función de la matriz de pesos de sus neuronas W_N y el *input* recibido $\vec{\mathcal{O}}_{N-1}$, Ec. (4.2).

$$\vec{\mathcal{O}}_N = \sigma(W_N \cdot \vec{\mathcal{O}}_{N-1}) = \sigma(W_N \cdot \sigma(W_{N-1} \cdot \vec{\mathcal{O}}_{N-2})) = \sigma(W_N \cdot \sigma(W_{N-1} \cdot \sigma(W_{N-2} \cdot \vec{\mathcal{O}}_{N-3}))) \tag{4.2}$$

Esta estructura recursiva consiste en un producto de matrices con funciones de activación σ intercaladas, aplicado sobre el vector *input* de la red. Esto hace que la derivada de todo el sistema se pueda calcular de una forma eficiente aplicando la regla de la cadena.

Al finalizar todas las capas, el valor o vector de valores *output* será la cantidad que se esperaba inferir a partir del *input* $\vec{x}$.

Entrenar una red neuronal

Ahora bien, esta estructura, tiene que aprender a predecir el *output* deseado. Inicialmente no es más que una serie de matrices con un gran número de parámetros que facilita un *output*. Así pues, es necesario entrenar la red.

Este entrenamiento se resume en "prueba y error" mientras que se ajustan los parámetros de la red (los pesos W de las neuronas). Para poder realizar el adiestramiento, es necesario contar con un gran volumen de casos ejemplo con parámetros iniciales $\{x_i\}$ para cada uno de los cuales se conoce el resultado esperado $\{\vec{o}_i\}$ (*ground truth*). Este set de datos se va a llamar *training data set*, o muestra de entrenamiento. Se ejecutará la red para $\vec{x}$ y comparará el resultado con $\vec{\mathcal{O}^{GT}}$.

Para compararlos, se define una función de coste (o *Loss*), que mida la distancia o disparidad entre las estimaciones $\vec{\mathcal{O}}$ y el *ground truth* $\vec{\mathcal{O}^{GT}}$. Por ejemplo, la Ec. (4.3):

$$\mathcal{L}(\vec{x}, \vec{\mathcal{O}}^{GT}, W) = \sum_j ||\vec{\mathcal{O}}_j - \vec{\mathcal{O}}_j^{GT}|| \tag{4.3}$$

Su valor escalar dependerá de la muestra de parámetros aportada $\vec{x}$, los resultados reales $\vec{\mathcal{O}}^{GT}$ y los pesos de la red W. El objetivo del entrenamiento será encontrar los parámetros de la red que mejor se ajusten a la muestra de entrenamiento, Ec. (4.4).

$$\mathcal{W}_e \quad t.q. \quad \mathcal{L}(\vec{x}_j, \vec{\mathcal{O}}_j^{GT}, \mathcal{W}_e) = min[\mathcal{L}(\vec{x}_j, \vec{\mathcal{O}}_j^{GT}, \mathcal{W})] \tag{4.4}$$

Como se ha introducido, la red se puede derivar fácilmente empleando la regla de la cadena. Por consiguiente, para obtener el mínimo de la función de coste, se puede emplear el descenso de gradiente.

El enfoque más directo para buscar mínimos de una función de tipo campo escalar, $f : \mathbb{R}^n \to \mathbb{R}$ como $\mathcal{L}$, es la utilización del gradiente $\nabla f(x_1, x_2, ..., x_n)$. Esta magnitud informa de la dirección de máximo crecimiento o decrecimiento $-\nabla f(x_1, x_2, ..., x_n)$, dado un punto $(x_1, x_2, ..., x_n)$ de la función. El gradiente es el vector de las derivadas parciales de la función con respecto a cada una de sus variables, para la función de coste, la derivada de $\mathcal{L}$ con respecto a cada uno de los pesos de las neuronas w_i, Ec. (4.5).

$$\nabla\mathcal{L}(\vec{x}_j,\ \vec{\mathcal{O}}_j^{GT},\ \mathcal{W}) = \left(\frac{\partial\mathcal{L}(\vec{x}_j,\ \vec{\mathcal{O}}_j^{GT},\ \mathcal{W})}{\partial w_1}, \frac{\partial\mathcal{L}(\vec{x}_j,\ \vec{\mathcal{O}}_j^{GT},\ \mathcal{W})}{\partial w_2}, ..., \frac{\partial\mathcal{L}(\vec{x}_j,\ \vec{\mathcal{O}}_j^{GT},\ \mathcal{W})}{\partial w_n}\right) \tag{4.5}$$

Para encontrar el mínimo global de la función, bastará modificar los pesos de la red una cierta cantidad en la dirección del menos gradiente, que en el contexto de aprendizaje automático se denomina *learning rate*, y recalcular su valor en el nuevo punto. El desplazamiento no será siempre en la dirección del mínimo, pero siguiendo la trayectoria de mayor descenso se asegura alcanzar este valor rápidamente. Esta explicación simplista no considera la existencia de mínimos locales, sin embargo, estos pueden ser atajados combinando diferentes tamaños de paso y otras estrategias.

En conclusión, el entrenamiento se divide en una pasada hacia adelante (*fordward pass*) en la que se calcula la salida de la red para la muestra de entrenamiento, y una propagación hacia atrás (**backward pass**) en la que, dado el valor de la función de coste, se calculan los gradientes para actualizar los pesos de la red. Este proceso se itera múltiples veces sobre la muestra de entrenamiento en lo que se denominan épocas (*epochs*).

Si el entrenamiento es correcto, la red será capaz de predecir el *output* para los x_i de la muestra de entrenamiento y, además, si la muestra es representativa y de calidad, generalizará para otros x_i distintos.

Los posibles problemas

Una vez entrenada la red con la muestra de entrenamiento, se puede poner a prueba en una serie de datos adicional (para los cuales también se conoce el *ground truth*), llamado *test data set* o datos de prueba. Llegado este punto, la red puede realizar predicciones aproximadamente correctas a partir de los parámetros de entrada, o puede haber aparecido un de dos problemas:

a. ***Bias:*** Cuando se intenta procesar un problema muy complejo para los pocos parámetros que se están utilizando. Los parámetros no logran captar la complejidad de los datos y la función de coste proporciona valores muy altos, incluso en su mínimo, fallando en la muestra de entrenamiento.

b. ***Overfitting:*** Ocurre si se utiliza un modelo mucho más complejo que el problema que se estudia, por ejemplo si tiene demasiados parámetros. La función de coste se minimizará en la muestra de entrenamiento, pero con una muestra de prueba dará valores anómalos o "alucinaciones".

Ejemplificando con unos ajustes simples: Resulta imposible ajustar unos datos exponenciales empleando un modelo lineal, lo que supone un problema de *bias*; un modelo polinómico tiene demasiados parámetros y sobreajusta los datos, problema de *overfitting*.

4.1.2. Redes convolucionales: U-Net

Las redes convolucionales se especializan en manipular imágenes. En vez de realizar productos escalares sobre un vector *input*, aplican convoluciones a la matriz de una imagen *input*. Las convoluciones son operaciones con diferentes *kernel* sobre la matriz original, de tal forma que se puede reducir o aumentar las dimensiones de la fotografía.

La U-Net, un tipo de red convolucional, es además simétrica, en el sentido de que la imagen generada es del mismo tamaño que la imagen *input*. También, en cuanto al tipo de convoluciones y convoluciones inversas que se aplican, que son simétricas respecto al cuello de la red, ver en Fig. (4.1).
La imagen original X atraviesa el codificador (red convolucional $\mathcal{F}$), donde se reduce su dimensión mediante convoluciones y *max pools*[1], hasta llegar al cuello de la U-Net, donde la imagen se encuentra en su estado comprimido $\mathcal{F}(X)$. Este estado se denomina espacio latente, los datos originales se encuentran comprimidos en un espacio de dimensiones más bajas. A continuación, la imagen atraviesa el decodificador (red deconvolucional $\mathcal{G}$), hasta llegar a la imagen reconstruida $X' = \mathcal{G} \cdot \mathcal{F}(X))$.

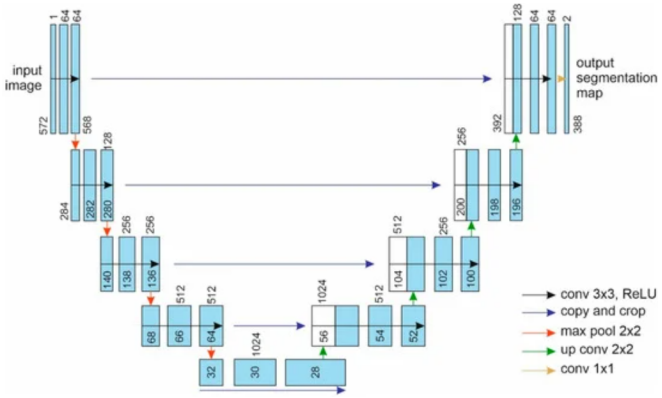

Figura 4.1: *Diagrama de las dimensiones de una imagen a medida que es procesada por una red convolucional. En la primera parte, el codificador $\mathcal{F}$ aplica reducciones con convoluciones y max pools a la imagen hasta reducirla al máximo en cuello, $\mathcal{F}(X)$. Luego el decodificador $\mathcal{G}$ la devuelve a su dimensión original siguiendo el proceso inverso, $X' = \mathcal{G} \cdot \mathcal{F}(X)$. Imagen tomada de [19].*

La imagen de entrada y salida es de las misma dimensiones, por lo que para entrenar una red U-Net es fundamental utilizar imágenes *input* y *ground truth* de las mismas dimensiones. Normalmente se utilizan un número de píxeles (base y altura) potencia de 2, como $N^2 = 128, 256, 512, 1024, ...$, por simplificar la compresión de la imagen en la red, que se reduce a la mitad en cada paso de codificación. Al mismo tiempo, las matrices de dimensión N^2 facilitan el trabajo al *hardware*, siendo su manipulación muy eficiente.
En función del entrenamiento que se les aplique, las redes convolucionales puede ejercer dos usos principales:

a. ***Segmentación.*** Su origen se encuentra en aplicaciones médicas, pero se emplea en campos tan distintos como la conducción autónoma. Consiste en que la red aprenda a reconocer patrones y siluetas de objetos, como pueden ser tumores, coches o figuras geométricas. Para entrenar la red, se utilizan como *ground truth* imágenes iguales a las del *input* con "máscaras"de diferentes colores sobre las clases de objetos que hay en la imagen y se quiere aprender a reconocer.

[1] *Max pooling*: reducción de dimensiones de una fotografía seleccionando el valor máximo en una matriz de regiones.

b. ***Denoising.*** Corrección de errores y artefactos en imágenes con ruido. Para entrenar la red, se utiliza un *input* de imágenes con ruido, defectos y artefactos simulados artificialmente y se pasa como *ground truth* las imágenes perfectas. Si la red se entrena correctamente, será capaz de realizar las misma correcciones en imágenes similares.

En el caso de estudio, la U-Net se va a entrenar para realizar *denoising*, transformando imágenes de las trazas de los protones, que contienen ruido debido al bajo número de protones por píxel y ciertos artefactos por el scattering de los protones con los chips de los detectores y la curvatura de su trayectoria, en imágenes de opacidad de los objetos que se han atravesado. Es decir mapas con la silueta de los objetos y con escala de gris informando de su opacidad.

4.2. Procedimiento para el entrenamiento de la U-Net

Para conseguir una U-Net funcional se han realizado cuatro pasos principales. Primero, se ha configurado y ejecutado la simulación del detector en Geant4. Segundo, se han generado imágenes a partir de las trazas de los protones, empleando un algoritmo de transmitancia. Tercero, se han creado imágenes de opacidad realistas y a escala de los objetos simulados, el *ground truth*. Y cuarto, se ha preparado y entrenado una red U-Net, para finalizar evaluando su funcionamiento con varias muestras de prueba.

***Hardware* empleado:**
Para acometer cada una de estas tareas se ha utilizado el *hardware* y *software* adecuados, trabajando en general en el sistema operativo Linux. Las simulaciones en Geant4 se han ejecutado en un ordenador remoto, conectándose a este mediante conexión *secure shell*[2]. Las imágenes de trazas y el *ground truth* también han sido generadas en este equipo ejecutando códigos Python desde su CPU.

El entrenamiento de la red, como se ha introducido, es un proceso exigente, donde es necesario realizar multitud de operaciones matriciales y calcular gradientes con respecto a un gran número de parámetros. Para esta operación se han utilizado los recursos computacionales del IFCA. La conexión a su servidor se ha realizado también con el protocolo SSH. En este caso, para su uso se ha trabajado en un *User Interface* donde los diferentes usuarios puede solicitar la ejecución de *jobs* mediante el sistema SLURM[3].

Así, el entrenamiento de la U-Net, se ha ejecutado en ocho GPUs Ampere A30[4]. Logrando de esta forma entrenar la red en periodos de tiempo muy reducidos, unos cuatro segundos por época, gracias a la capacidad de las GPUs para realizar cálculos simultáneos.

4.2.1. Simulación en Geant4

El sistema PROTECT está simulado en Geant4, como se ha introducido en el Cap. 2. Para su utilización, en primer lugar se instalaron los prerequisitos y compiló el *software* Geant4 en Linux y Windows, lo cual no resultó trivial. Fue necesario utilizar las funcionalidades básicas de *cmake*[5] y del lenguaje de programación C++.

[2]SSH: *Secure Shell*, protocolo seguro para conexión remota mediante línea de comandos entre ordenadores.
[3]SLURM (*Simple Linux Utility for Resource Management*): gestor de carga de trabajo para compartir los recursos de un servidor de alto rendimiento de manera eficiente.
[4]GPU Ampere A30: Tarjeta gráfica con la arquitectura NVIDIA Ampere. Con 3584 núcleos y 24 GB de RAM.
[5]*cmake:* software para construir, probar y empaquetar programas informáticos.

Tipo de Tejido	ρ/ g/cm^3
Pulmón	1.05
Tejido cerebral	1.03
Tejido adiposo	0.92
Tejido óseo	1.85

Tabla 4.1: *Valores de densidad para los cuatro tipos de tejido estudiados, datos extraídos de la documentación de Geant4 [20].*

Una vez aprendidos los básicos de utilización de Geant4, se instaló y compiló el programa PROTECT y la herramienta de tratamiento de datos ROOT.

Para entrenar la red, se simularon escenarios con *phantoms*, modelos que simulan el tejido humano y sus propiedades, posicionados en el centro de los dos detectores del *gantry* de PROTECT. Se generaron cuatro fantomas con forma de monedas, uno por cada cuadrante del plano $z = 0$ con diferentes tipos de tejido: graso, pulmonar, óseo y cerebral; cuyas densidades se recogen de la Tab. (4.1).
Los *phantoms* generados tienen un grosor $z_{size} = 0.5$ cm y radio de $r = 3$ cm. Su distribución en el plano $z = 0$ es en las posiciones $(x, y)_{pulmon} = (3, 3)$, $(x, y)_{cerebro} = (3, -3)$, $(x, y)_{grasa} = (-3, 3)$ y $(x, y)_{hueso} = (-3, -3)$, ver Fig. (4.2).

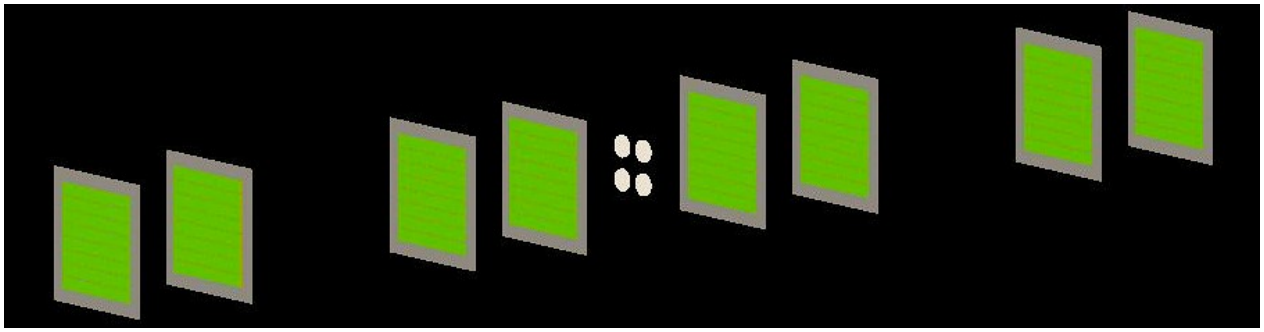

Figura 4.2: *Imagen de cuatro fantomas circulares de espesor $z_{size} = 0.5$ cm y radio $r = 3$ cm en el centro de los dos detectores (cada uno compuesto de 4 planos de LGADs). Imagen de la simulación en Geant4 de PROTECT.*

Para generar la muestra de entrenamiento para la Unet, se generaron archivos de configuración *json*[6] modificando ligeramente la posición y radio de los *phantoms*, como se verá en las ilustraciones Fig. (4.4). En total se definieron $N = 499$ escenarios, en los que se ejecutó la simulación con un haz de $n_{p+} = 120 \cdot 10^3$ protones de energía $E_{p+} = 250$ MeV. El número n_{p+} elegido es un múltiplo de los pasos del barrido del haz utilizado. Igualmente, se ejecutaron otros $N = 499$ escenarios sin fantomas.

Para cada simulación, se obtuvo un archivo ROOT con los datos de generación de los protones y las medidas registradas por los detectores LGAD. Sobre estos datos de detección, se ejecutó un algoritmo de reconstrucción de trazas. El cual calcula las posiciones $(x, y, z)_1$ y $(x, y, z)_2$ y los vectores velocidad $(v_x, v_y, v_z)_1$ y $(v_x, v_y, v_z)_2$ de los protones en los detectores 1 y 2. La reconstrucción solo resulta exitosa si los protones son detectados en los 8 planos de sensores LGAD y su desviación

[6] *json:* formato de archivo ligero para guardar parámetros o información.

angular es compatible con un evento de dispersión.

4.2.2. Algoritmos de generación de imágenes a partir de trazas

Para poder entrenar la U-Net, la información de las trazas de los protones debe condensarse en una imagen que la red pueda procesar.

La primera idea para esto fue utilizar el algoritmo de POCA, ver Anexo .5 para una explicación y desarrollo matemático detallado. El algoritmo no ofreció buenos resultados a pesar de trabajar en estadísticas altas con haces de $n_{p+} = 2 \cdot 10^6$ protones. La reconstrucción POCA resultaba en un número muy reducido de protones dispersados dentro de los *phantoms*, y las imágenes informaban más de la transmitancia de los protones que de su scattering. Esto hizo que se descartara el uso de imágenes generadas con el algoritmo de POCA y se optase por estudiar la transmitancia.

Se comprobó que al atravesar los planos de detección y los *phantoms* el descenso en el número de protones era considerable. Como segunda idea, para obtener imágenes de transmitancia de los protones, se comparó el número de p^+ detectados en el det. 2 del *gantry* cuando había *phantoms* y en vacío.

Utilizando las trazas de protones registrados en el detector 2, se reconstruyó su posición en el plano $z = 0$, para las simulaciones con y sin *phantoms*. Las 499 simulaciones sin objetos se unificaron en un archivo de alta estadística que se utilizó como fondo. Una vez obtenidas las coordenadas de los protones $(x, y)_P$ en el plano $z = 0$ se generaron histogramas con unas dimensiones y *binning* determinados, uno por cada escenario de fantomas $img_i^{Phantoms}$ y otro para el fondo img^{Fondo}. La imagen de transmitancia se calculó acorde a la Ec. (4.6), donde se normaliza el fondo de alta estadística. Luego se normalizó al valor máximo de $img_i^{Transmitancia}$ y multiplicó por 255, obteniendo una imagen en escala de grises de 8 bits.

$$img_i^{Transmitancia} = \frac{img_i^{Phantoms}}{img^{Fondo}/499} \tag{4.6}$$

Las imágenes se generaron en el rango $h \times w = 15 \times 15$ cm^2 sobre el plano $z = 0$, asegurando una imagen representativa del área ocupado por los *phantoms*. Por otro lado, se decidió generar las imágenes con resolución 256×256 px^2, calidad suficiente pero tamaño reducido para no lastrar la ejecución de la red. Finalmente, resultó clave la elección del *binning*, este se ajustó en función de la estadística de las simulaciones con fantomas. Se ha utilizado un número de $bins = 40$ en ambos ejes. Si se utiliza un valor más elevado aparece un gran número de *bins* negros fuera de la silueta de los *phantoms*, lo que se consideró que no favorecería el desempeño de la U-Net al aportar una gran cantidad de información sesgada por su baja estadística. No obstante, este *binning* podría resultar en que objetos de tamaño inferior a $\eta = 15$ cm$/40 = 0.375$ cm serán difícilmente discernibles.

Tanto el algoritmo de POCA como el de transmitancia, se programaron en dos *scripts* ejecutables con opciones, para indicar la dirección del directorio con los archivos de trazas con fantomas, el archivo de fondo, y el directorio destino para las imágenes generadas. Se programaron en Python utilizando la librería *uproot* para analizar la información de los archivos ROOT, *numpy* para generar los histogramas y la clase *Image* de la librería *Pillow* para generar imágenes ajustadas a las características necesarias.

4.2.3. Generación del *ground truth*

Para simular las "imágenes reales" de los *phantoms*, se leyeron los archivos de configuración *json* de cada uno de los 499 escenarios simulados. En cada uno, se extrajo la posición, material, radio y espesor de los cuatro fantomas. Con esta información, se generaron imágenes de la proyección de los *phantoms* sobre el plano $z = 0$ en escala de gris en función de su opacidad. La opacidad O se define como el producto de espesor z y densidad ρ de un objeto, Ec. (4.7). Esta se normalizó al máximo valor y multiplicó por 255 para obtener imágenes en escala de grises de 8 bits.

$$O = \rho \cdot z \tag{4.7}$$

De nuevo, se preparó un ejecutable con opciones para especificar los directorios con archivos *json* de configuración y el de salida para las imágenes generadas. Igualmente, el *ground truth* se generó a la misma escala con rango 15×15 cm^2 sobre el plano $z = 0$ y la misma resolución 256×256 px^2.

4.2.4. La U-Net y preparación para su entrenamiento

En vez de programar una red convolucional desde cero, se ha aplicado la técnica de aprendizaje transferido o *transfered learning*. Esta consiste en utilizar un modelo pre-entrenado y especializarlo con un entrenamiento personalizado para una tarea específica. Resulta práctico ya que entrenar un modelo nuevo puede resultar extremadamente costoso, dado que la cantidad de datos, potencia computacional y tiempo a dedicar es mucho mayor. Así, la red aprenderá a realizar una tarea específica: reconstruir los fantomas a partir de su patrón de su transmitancia, pero partiendo de una base sólida sabiendo distinguir bordes, figuras geométricas... al haber sido expuesta a un entrenamiento extensivo con un alto número de fotografías generales.

Se ha utilizado un modelo consistente de cinco convoluciones en el *encoder* combinadas con *max-pools*, y que utiliza la función ReLU[7] como función de activación. Esta U-Net se ha reutilizado del repositorio GitHub [21]. La red está implementada en Python utilizando las funcionalidades de la librería especializada *pytorch* [22].

La red original está configurada para realizar segmentación en 6 canales (5 clases de objeto + fondo), por lo que el primer paso fue fijar el número de canales de salida a 1, correspondiente a la imagen en escala de grises. Igualmente, se adaptó el entrenamiento al problema introducido, para lo que se ajustó una serie de hiperparámetros, que controlan el proceso de aprendizaje de la red, estos se discuten en los siguientes párrafos.

En primer lugar, se utilizó un manipulador de *data sets*, para poder contar con tres conjuntos de datos: dos durante el aprendizaje de la red, uno de entrenamiento con 450 pares de imágenes y otro con 40 de validación, y un tercero de prueba para validar su funcionamiento tras entrenarla, con 10 pares de imágenes. De esta forma, durante cada una de las épocas, además de ajustar sus pesos con la muestra de entrenamiento, se podrá verificar con datos adicionales para evitar problemas de *overfitting*. Estos cargadores de datos se configuraron para pasar las parejas de imágenes (*ground truth* e *input*) de manera aleatoria a la red en cada iteración, mejorando su entrenamiento. Igualmente, en ambos *data set*, se estableció el *batch size* a 10. Este es el número de muestras en que se divide cada *data set* en una época de entrenamiento, es decir, es el número de imágenes que estudia la red antes de actualizar sus parámetros, lo que ocurre varias veces a lo largo de una

[7] ReLU (Unidad Lineal Rectificada): función de activación que retorna la entrada si esta es positiva, y cero si es negativa o cero.

época. Un *batch size* bajo (1) permite que la red aprenda rápido utilizando poca memoria GPU, sin embargo, el gradiente que se calcula es muy ruidoso. En contrapartida, el mayor *batch size* posible (= al tamaño del *data set*) ofrece un gradiente muy estable y un tiempo por época más reducido, al actualizar los parámetro una sola vez, con la contrapartida de utilizar mucha memoria. Por estas razones, se optó por un valor equilibrado pero reducido.

En segundo lugar, se ha elegido la función de coste para evaluar las predicciones de la red durante el entrenamiento. Se han considerado tres de las más utilizadas para entrenamiento de U-Nets: el Error Cuadrático Medio MSE, la *Binary Cross-Entropy* BCE y el *Dice-Loss*.

La función de pérdida **MSE** [23] mide la diferencia media cuadrada entre dos valores. Se define acorde a la Ec. (4.8), donde N es el número de píxeles de la imagen, y_i el valor real de los píxeles y $\hat{y}_i$ su predicción. Esta función de coste es la referencia al entrenar redes para *denoising*.

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{4.8}$$

Cuanto menor sea su valor, mayor será la similitud entre dos imágenes. Sin embargo, a la hora de interpretar su valor es necesario tener en cuenta el tipo de imagen de estudio. Por ejemplo, un mismo valor MSE tendrá un mayor impacto para una imagen de 4 bits frente a una de 8 bits. Al entrenar una red, asegura una correlación en cuanto a la intensidad, sin embargo, tiende a suavizar los bordes y no contribuye especialmente a la conservación de la forma estructural. Se implementó desde el módulo *nn* de la librería *pytorch* [22].

La *Binary Cross-Entropy* **BCE** [24] se suele emplear en clasificación binaria. Su definición, en términos del número de píxeles total N, sus valores reales y_i y las predicciones $\hat{y}_i$, viene dada por la Ec. (4.9). Se debe aplicar sobre imágenes normalizadas a 1.

$$\mathcal{L}_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)] \tag{4.9}$$

Frente a la función MSE, BCE presenta una mayor sensibilidad en regiones con valores extremos (próximos a 1 ó 0). Así, esta función de coste mejora la definición entre regiones con alto contraste.

El coeficiente ***Dice*** [25] se define para estudiar la superposición entre regiones. La variable y_i es el valor real de los píxeles y $\hat{y}_i$ es la predicción de la red. A partir del coeficiente, el valor de la función de coste se obtiene restándolo a 1, como aparece en la Ec. (4.10).

$$\mathcal{L}_{Dice} = 1 - \frac{2\sum y_i\hat{y}_i + \epsilon}{\sum y_i + \sum \hat{y}_i + \epsilon} \tag{4.10}$$

Esta resulta especialmente útil en segmentación, ya que penaliza fuertemente los errores estructurales y contribuye a conservar formas y bordes. Para su implementación, se definió como una función en Python.

A partir de estas tres funciones, se puede definir una función de coste total $\mathcal{L}_{Total}$ sumando algunas de las anteriores.

Se programó un entrenamiento con 50 épocas, durante las cuales el *learning rate* varió desde $lr = 0.001$ hasta $lr = 1 \cdot 10^{-5}$, gracias a la implementación de un *learning rate scheduler*. En cada época se evaluó la muestra de entrenamiento y validación para los *batch* y se registró el valor de la función de pérdida total para ambas.

Una vez entrenada, se verificó el funcionamiento de la red con la muestra de 10 imágenes de prueba. Para lo cual se calculó el valor de MSE, y el error absoluto entre las imágenes de *ground truth* y las predicciones. Para calcular el error absoluto, simplemente se restó píxel a píxel ambas imágenes y se tomó el valor absoluto, creando una nueva imagen con esta magnitud.

4.3. Resultados y discusión

La red se entrenó utilizando dos funciones de coste totales $\mathcal{L}_{Total}$ distintas. Para ambas, se llevó a cabo el proceso descrito y se testó el resultado con la misma muestra de prueba. En primer lugar, utilizando como $\mathcal{L}_{Total}$ el parámetro MSE, $\mathcal{L}_{Total} = \mathcal{L}_{MSE}$. y en segundo lugar una combinación de la BCE y *Dice loss*, $\mathcal{L}_{Total} = \mathcal{L}_{BCE} + \mathcal{L}_{Dice}$.

Se observó como, en ambos casos, el valor de la función de coste fue disminuyendo en cada época para las muestra de entrenamiento y validación, si bien su variación se hacía menor gradualmente. En parte porque el modelo se estaba entrenando, y también como efecto del *learning rate scheduler*. En la Fig. (4.3) se representa la variación del valor de la función de coste $\mathcal{L}_{Total} = \mathcal{L}_{BCE} + \mathcal{L}_{Dice}$ a lo largo de las 49 épocas.

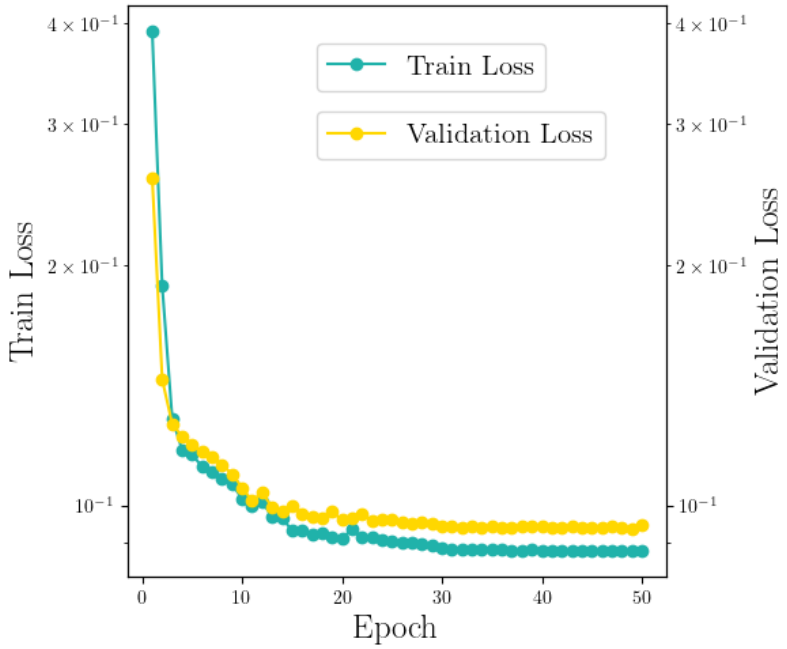

Figura 4.3: *Representación del valor de la función de coste en cada una de las 49 épocas de entrenamiento para la muestra de entrenamiento (azul) y validación (amarillo) en escala logarítmica, utilizando* $\mathcal{L}_{Total} = \mathcal{L}_{BCE} + \mathcal{L}_{Dice}$. *El valor de entrenamiento alcanza su mínimo con* $\mathcal{L}(train) = 0.088$, *mientras que el valor de validación se reduce hasta* $\mathcal{L}(val) = 0.094$.

En la figura Fig. (4.4, A) y Fig. (4.4, B), se recogen dos ejemplos representativos de la evaluación de la red tras el entrenamiento, utilizando las dos funciones de coste totales diferentes.

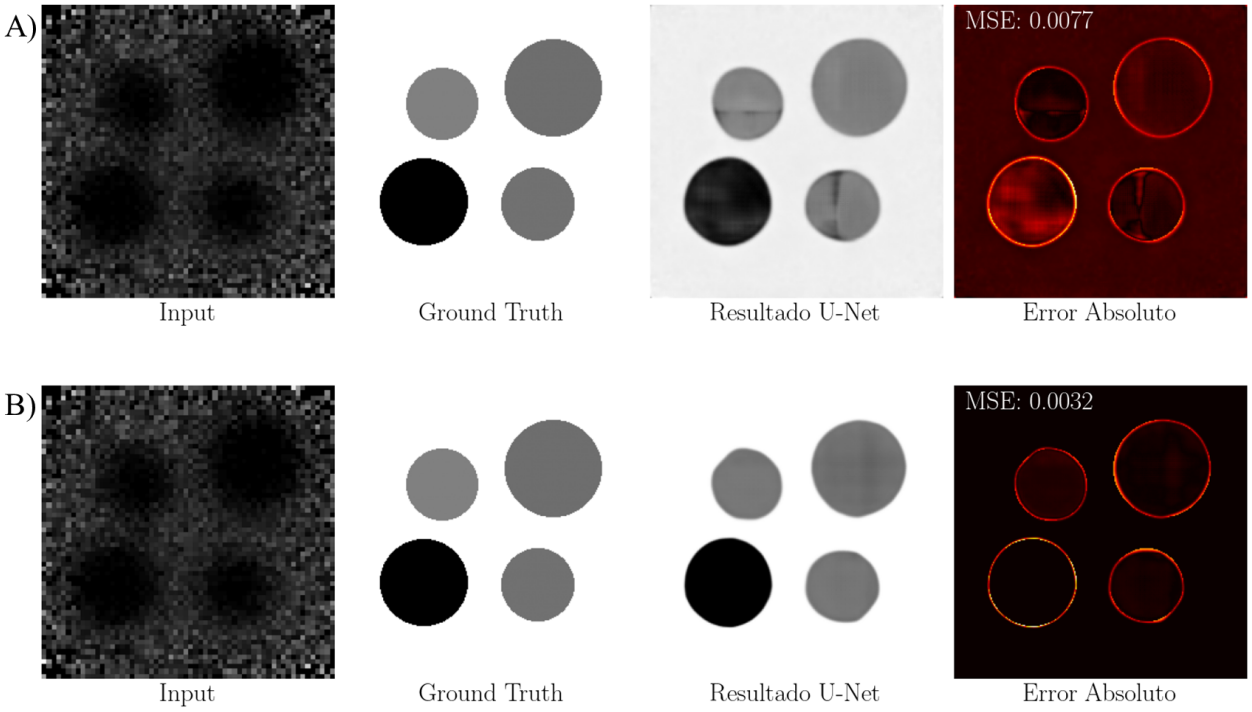

Figura 4.4: *De izquierda a derecha: Imagen de transmitancia utilizada como input de la red, ground truth, resultado de la red entrenada y representación del error absoluto entre la predicción y la realidad. En la fila* ***(A)*** *cuando se empleó en el entrenamiento* $\mathcal{L}_{Total} = \mathcal{L}_{MSE}$ *y en la fila* ***(B)*** *con* $\mathcal{L}_{Total} = \mathcal{L}_{BCE} + \mathcal{L}_{Dice}$*. Ambas representaciones del error absoluto se etiquetan con el valor de MSE entre el ground truth y la estimación.*

La Fig. (4.4) muestra cómo los resultados obtenidos con $\mathcal{L}_{Total} = \mathcal{L}_{BCE} + \mathcal{L}_{Dice}$ son cualitativa y cuantitativamente superiores a los obtenidos $\mathcal{L}_{Total} = \mathcal{L}_{MSE}$, si bien esta segunda es la función de coste indicada para realizar entrenamiento para *denoising*. Si se comparan las imágenes de error absoluto, se ve como la función de coste BCE + *Dice*, generalmente utilizada en segmentación, enseña a la red a reconocer mejor los bordes y la opacidad de los fantomas. Igualmente, diferencia el valor del fondo con superioridad a la red entrenada con la función MSE. De hecho, si se compara los valores de MSE en la imagen de Error Absoluto de la Fig. (4.4), se alcanza un valor inferior de MSE cuando minimizando BCE + *Dice* frente a cuando solo se minimiza la función MSE.

Tal y como se ha introducido, la minimización de MSE, se centra en corregir la intensidad media, por lo que no logra entrenar tan bien la red en calcular la opacidad de los *phantoms* ni en distinguir valores de máximo contraste $(0, 255)$. Para estas tareas, resulta más conveniente analizar la superposición de regiones, tarea que la función de *Dice* facilita a la red en su entrenamiento. La calidad de los resultados obtenidos con la U-Net se cuantifican en función del valor medio de MSE para las imágenes obtenidas en la muestra de prueba, Tab. (4.2). Para este tipo de imágenes, los resultados utilizando como función de coste BCE + *Dice* son superiores en un $\Delta(\text{MSE}) = 50\,\%$.

Por otro lado, se probó modificando $\mathcal{L}$ con factores de ponderación entre BCE y *Dice*, $\mathcal{L}_{Total} = A \cdot \mathcal{L}_{BCE} + B \cdot \mathcal{L}_{Dice}$. No obstante, los mejores resultados se obtuvieron con $A, B = 0.5$.

Entrenamiento U-Net	$\mu(\text{MSE}) \pm \sigma(\text{MSE})$
$\mathcal{L}_{Total} = \mathcal{L}_{\text{MSE}}$	0.0075 ± 0.0007
$\mathcal{L}_{Total} = \mathcal{L}_{\text{BCE}} + \mathcal{L}_{Dice}$	0.0033 ± 0.0008

Tabla 4.2: *Comparación del valor medio y desviación estándar de MSE para los resultados de la U-Net tras entrenarla utilizando dos funciones de coste diferentes.*

A continuación, para la red entrenada que mejores resultados ofrece, se observó la calidad de sus estimaciones en función del tamaño de los fantomas. En la Fig. (4.5), se presenta el resultado de la red para dos imágenes adicionales del *data set* de prueba.

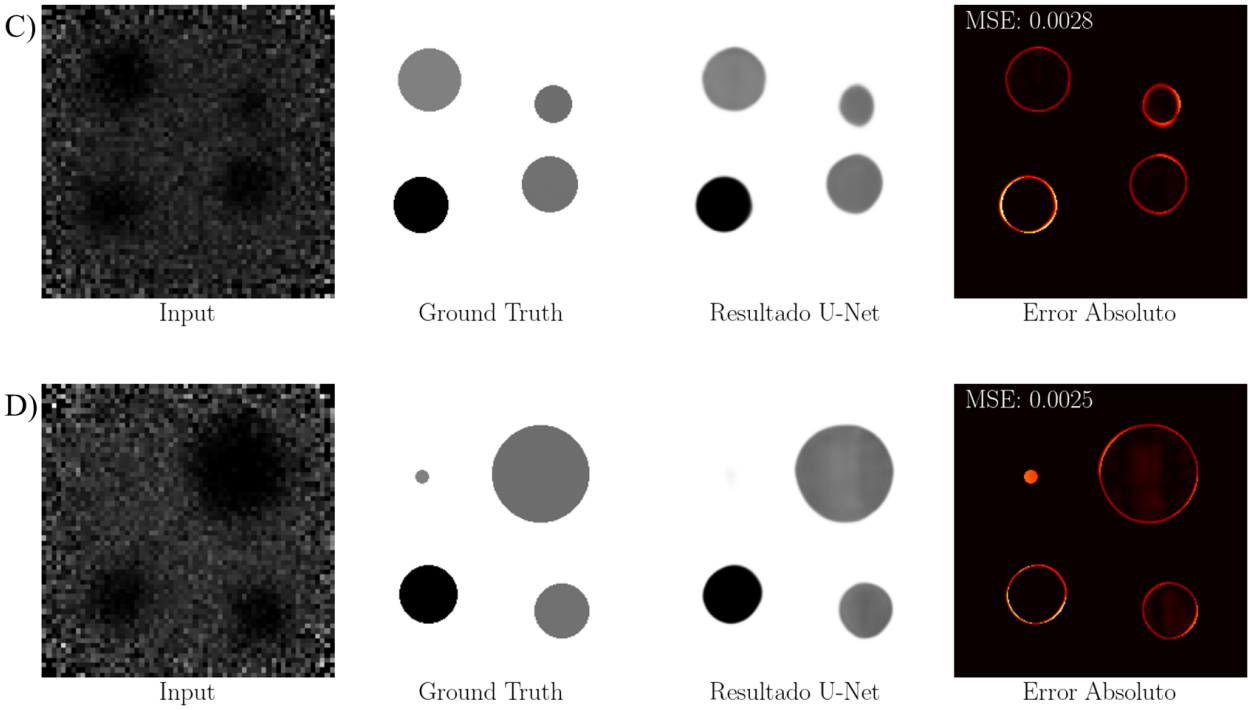

Figura 4.5: *Dos pruebas del funcionamiento de la U-Net al ser entrenada con la función de coste* $\mathcal{L}_{Total} = \mathcal{L}_{BCE} + \mathcal{L}_{Dice}$. *De izquierda a derecha: Imagen de transmitancia utilizada como input de la red, ground truth, resultado de la red entrenada y representación del error absoluto entre la predicción y la realidad. Se etiquetan con el valor de MSE entre el ground truth y la estimación.*

En la Fig. (4.5, C), la U-Net reconstruye exitosamente el fantoma de menor radio, $r_C = 0.93$ cm. Sin embargo, en la Fig. (4.5, D), fracasa en reproducir uno de radio $r_D = 0.33$ cm. Si bien en la imagen resultado de la U-Net hay una sombra en la posición esperada, esta no corresponde con el *phantom* tan bien como en otros casos. Para este mismo caso, $r_D = 0.33$ cm, la red entrenada con MSE no llega a mostrar la sombra.

En primera instancia, esto puede parecer una consecuencia del *binning* seleccionado. Para comprobarlo, se repitió el entrenamiento utilizando imágenes de transmitancia de protones con *binning* =

100, lo que debería ofrecer una resolución $\eta = 15$ cm$/100 = 0.15$ cm. No obstante, la red no fue capaz de reconstruir el fantoma de $r_D = 0.33$ cm, y la sombra que se ve en la Fig. (4.5, D) desapareció. Por lo tanto, la selección del *binning* ajustada a la estadística de los protones, buscando reducir los píxeles negros, resultó adecuada.

4.4. Conclusiones

El entrenamiento de la U-Net ha permitido preparar una herramienta que, aplicando *denoising* a una imagen de transmitancia de protones, reconstruye la proyección de fantomas simulados sobre el plano $z = 0$. Esta operación se realiza con un error cuadrático medio $MSE = 0.0033 \pm 0.0008$, cuando los píxeles de la imagen de 8 bits se encuentran en el rango [0, 255]. Así pues, el error medio por píxel es del $\delta(px) = 0.001\,\%$, un resultado verdaderamente bueno para una red entrenada con menos de 500 fotografías. Además, se ha comprobado que la red detecta la existencia de *phantoms* de hasta $r_D = 0.33$ cm de radio, si bien en este caso la reconstrucción de su silueta no es completa.

Una de las principales conclusiones adicionales que se extrae es que la utilización de funciones de coste propias de la segmentación, puede ofrecer mejores resultados que las funciones indicadas para *denoising*, como se ha visto en la Tab. (4.2). Este efecto se justifica al ser las imágenes de *ground truth* utilizadas casos geométricamente simples, con solo cuatro círculos con niveles discretos de gris sobre un fondo blanco uniforme. Por consiguiente, la red puede estar procesando las imágenes como una máscara con niveles de gris discretos, simplificando la tarea.

Dada la simpleza de los fantomas utilizados, la comparación no es del todo justa, pero se puede contrastar el radio de los *phantoms* detectados con la resolución en tomografía TAC de RX. La resolución típica en TAC [26] es de $\delta(x, y) = 1 - 0.5$ mm axialmente y $\delta(z) = 5 - 0.5$ mm longitudinalmente. El *fantoma* más pequeño parcialmente detectado tiene un radio (axial) de $r_D = 3$ mm y un espesor (longitudinal) de $z = 5$ mm. Por tanto la máxima resolución axial alcanzada en esta simulación es tan solo tres veces inferior, y la longitudinal es comparable a la de un equipo TAC profesional, un resultado prometedor para continuar desarrollando PROTECT.

Los detectores LGAD utilizados en la simulación han permitido una correcta medida de las trazas de protones del haz. Sin embargo, no se ha explotado todo su potencial, ya que las imágenes generadas solo han utilizado medidas de la posición y dirección de los protones. Implementando algoritmos de reconstrucción más complejos, que tengan en cuenta el tiempo de llegada y energía depositada, será posible mejorar la calidad y precisión del estudio.

El trabajo realizado sienta una base sólida para la aplicación de tecnología LGAD y redes U-Net para reconstrucción de imágenes utilizando radiación de protones, dejando un amplio margen de mejora gracias al potencial de los sensores. Con el objetivo de realizar reconstrucciones tomográficas, los siguientes pasos a tomar serán la combinación mediante el algoritmo de *Filtered Back Projection* de múltiples imágenes. Estas podrán utilizar la transmitancia, variación en dirección, tiempo de paso, y energía de los protones, y recrear un mapa del *stopping power* del paciente desde el punto de vista de un protón.

El código utilizado para el entrenamiento de la U-Net y las herramientas creadas para generar el *ground truth*, POCA e imágenes de atenuación, se puede acceder en el repositorio de GitHub [27].

5 Conclusiones

Este trabajo de fin de grado ha conseguido colaborar con dos grandes proyectos al nivel de un estudiante de Física. Por un lado, la actualización a alta luminosidad de CMS con su novedoso MTD, y por otro, el proyecto PROTECT. Ambos, aún teniendo fines diferentes, comparten la utilización de la tecnología LGAD.

En primer lugar, se ha comprendido el impacto del HL-LHC en el CMS, así como la necesidad de un detector con alta resolución temporal, el MTD. Para conseguir un montaje preciso de los módulos LGAD, utilizados en el ETL, se ha contribuido actualizando el *hardware* de un robot de montaje automático con un sistema óptico. Finalmente, los objetivos alcanzados en esta parte son el montaje del microscopio, el desarrollo de un método para su calibración y el desarrollo de códigos que permiten la toma y análisis de imágenes con su cámara. La calibración del sistema óptico permitirá al robot conocer la posición horizontal de objetos con una precisión $\sigma(x, y)_C = 40$ μm. Estos resultados, permitirán al grupo CMS afrontar la producción de módulos LGAD en los próximos meses.

En segundo lugar, con el proyecto PROTECT, se ha visto como las novedades en instrumentación de Física de Partículas pueden ser aplicadas a otros campos, como la Radiofísica. Utilizando una simulación de un dispositivo de tomografía protónica basado en la tecnología LGAD, se ha buscado reconstruir imágenes de fantomas. Analizando la transmitancia de protones y entrenando una red convolucional de tipo U-Net para un caso simple, se ha logrado obtener imágenes con un error medio relativo por píxel de tan solo $\delta(px) = 0.001\,\%$. El proceso para obtener estos resultados ha llevado al aprendizaje de nociones en *machine learning* y la utilización de diversas herramientas computacionales avanzadas.

Finalmente, en este trabajo se ha buscado aplicar la Física de una forma ambiciosa a problemas de interés tecnológico y científico. Se ha comprobado como, aún enfrentándose a pequeñas piezas de los grandes puzzles que son PROTECT y el HL-LHC, la cantidad de obstáculos que se pueden encontrar en el camino es incalculable. Aplicando el método científico, los conocimientos previos y los nuevos adquiridos, ha sido posible obtener resultados y extraer conclusiones dentro de la complejidad del campo de estudio.

Referencias

[1] Instituto de Física de Cantabria (IFCA). Física experimental de Altas Energías e instrumentación. https://ifca.unican.es/es-es/investigacion/fisica-de-altas-energias-e-instrumentacion/fisica-experimental-de-altas-energias-e-instrumentacion, 2025. Accedido el 10 junio 2025.

[2] K. Kaadze, J. N. Butler, and T. Tabarelli. New precision timing detector for the CMS HL-LHC upgrade. https://cms.cern/news/new-precision-timing-detector-cms-hl-lhc-upgrade, June 2018. Accedido el 10 junio 2025.

[3] CERN. High-Luminosity LHC. https://home.cern/resources/faqs/high-luminosity-lhc, 2025. Accedido el 10 junio 2025.

[4] Gaceta UNAM / Universidad Nacional Autónoma de México. Desarrollan en Física detector de partículas – imagen de los detectores ALICE. https://www.gaceta.unam.mx/desarrollan-en-fisica-detector-de-particulas/, October 2019. Imagen vista el 10 junio 2025.

[5] CMS Collaboration. Detector – CMS Experiment. https://cms.cern/detector, 2025. Accedido el 10 junio 2025.

[6] CIEMAT – Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas. El experimento CMS. https://divulgacion.ciemat.es/cms, 2025. Accedido el 10 de junio de 2025.

[7] Ramon Cid-Vidal. Luminosidad. https://www.lhc-closer.es/taking_a_closer_look_at_lhc/0.luminosity/idioma/es_ES, 2025. Accedido el 10 de junio de 2025.

[8] CMS Collaboration. A mip timing detector for the cms phase-2 upgrade. Technical Report CERN-LHCC-2019-003 / CMS-TDR-020, CERN, 2019. Technical Design Report, Revised 26 September 2019. Accedido: 2025-06-10.

[9] M. Carmen Jiménez-Ramos et al. Study of ionization charge density-induced gain suppression in lgads. *Sensors*, 22(3):1080, 2022.

[10] Catherine Therese Quiñones. *Proton Computed Tomography*. PhD thesis, Institut National des Sciences Appliquées de Lyon, Lyon, France, 2016. PhD thesis.

[11] Revista Nuclear. Revista Nuclear No. 316 – Sección relevante. `https://www.revistanuclear.es/wp-content/uploads/hemeroteca/316/NE316-06.pdf`, 2025. Accedido el 10 de junio de 2025.

[12] Salvador Galindo Uribarri. Principios matemáticos de la reconstrucción de imágenes tomográficas. *Ciencia ergo sum*, 10(3):271–281, 2003. Descargado el 10 de junio de 2025 desde Dialnet.

[13] Leire Etcheverry Martínez. Tomografía protónica de alta resolución basada en medidas del tiempo de vuelo. TFG, *Grado en Física*, Universidad del País Vasco / Euskal Herriko Unibertsitatea, September 2024. hdl:10902/34748.

[14] Ronald A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A*, 222:309–368, 1922.

[15] Skipper Seabold and Josef Perktold. Statsmodels: Econometric and statistical modeling with python. *Proceedings of the 9th Python in Science Conference*, 57:61, 2010.

[16] Gregory G. Slabaugh. Computing euler angles from a rotation matrix. Technical report, Queen Mary University of London, 1999. Accedido: 2025-06-10.

[17] Raúl Penagos. Cameracontrol. `https://github.com/RaulPenagos/CameraControl`, 2025. Accedido: 2025-06-11.

[18] Raúl Penagos. Ids_camera. `https://github.com/RaulPenagos/IDS_Camera`, 2025. Accedido: 2025-06-11.

[19] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *arXiv preprint arXiv:1505.04597*, 2015. Figure 1 referenced. Accedido: 2025-06-10.

[20] CERN. Geant4 material names — application developer guide. `https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/html/Appendix/materialNames.html`, 2025. Accedido: 2025-06-10.

[21] Mostafa Wael. U-net in pytorch. `https://github.com/Mostafa-wael/U-Net-in-PyTorch`, 2020. Accedido: 2025-06-10.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

[23] NFDI4Ing Consortium. Mean squared error — data quality metrics. `https://quality.nfdi4ing.de/en/latest/image_quality/MeanSquared_Error.html`, 2024. Accedido: 2025-06-10.

[24] Neeraj Nan. Binary cross entropy — machine learning. `https://medium.com/@neerajnan/binary-cross-entropy-machine-learning-1c5dee1f2d52`, 2023. Accedido: 2025-06-10.

[25] Jeremy Jordan. An overview of semantic image segmentation. `https://www.jeremyjordan.me/semantic-segmentation/`, 2018. Accedido: 2025-06-10.

[26] Radiopaedia.org. Spatial resolution in computed tomography. `https://radiopaedia.org/articles/spatial-resolution?lang=us`, 2025. Accedido en junio de 2025.

[27] Raúl Penagos. Myunet. `https://github.com/RaulPenagos/MyUnet`, 2025. Accedido: 2025-06-11.

[28] IDS Imaging Development Systems. U3-3270CP Rev.2.2 USB·3·Vision industrial camera. `https://en.ids-imaging.com/store/u3-3270cp-rev-2-2.html`, 2025. Accedido el 10 de junio de 2025.

[29] bestoneshop-korea. LENTE OBJETIVO MITUTOYO M Plan Apo 5x/0,14 f=200. `https://www.ebay.es/itm/285531128973`, 2025. Accedido el 10 de junio de 2025.

[30] IDS Imaging Development Systems GmbH. Ids peak sdk. `https://en.ids-imaging.com/manuals/ids-peak-sdk/`, 2023. Accessed: 2025-06-12.

[31] IDS Imaging Development Systems GmbH. Rapid prototyping with python — ids peak — tech tip, 2021. Accedido: 2025-06-11.

[32] Gary Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

Anexo

.1. Montaje del microscopio

En la Fig. (1), se ve la cámara, objetivo y montura-C. Esta tiene una rosca estándar de diámetro $\varnothing = 1$ pulgada para poder acoplar la cámara. Por el otro lado, la montura cuenta con una rosca para el objetivo infinito. El microscopio, proyecta la imagen formada en el objetivo sobre la cámara gracias a una lente de $f = 200$ mm dentro de la montura. Además, los 6 agujeros con métrica M4 permitirán su montaje en el robot, como se ve en la Fig. (3.7).
El codo de la montura, con un diafragma y una entrada para fibra óptica de $\varnothing = 1/4''$ permite introducir un haz de luz para asistir la captura de imágenes, sin necesitar ajustes en la exposición de la cámara.

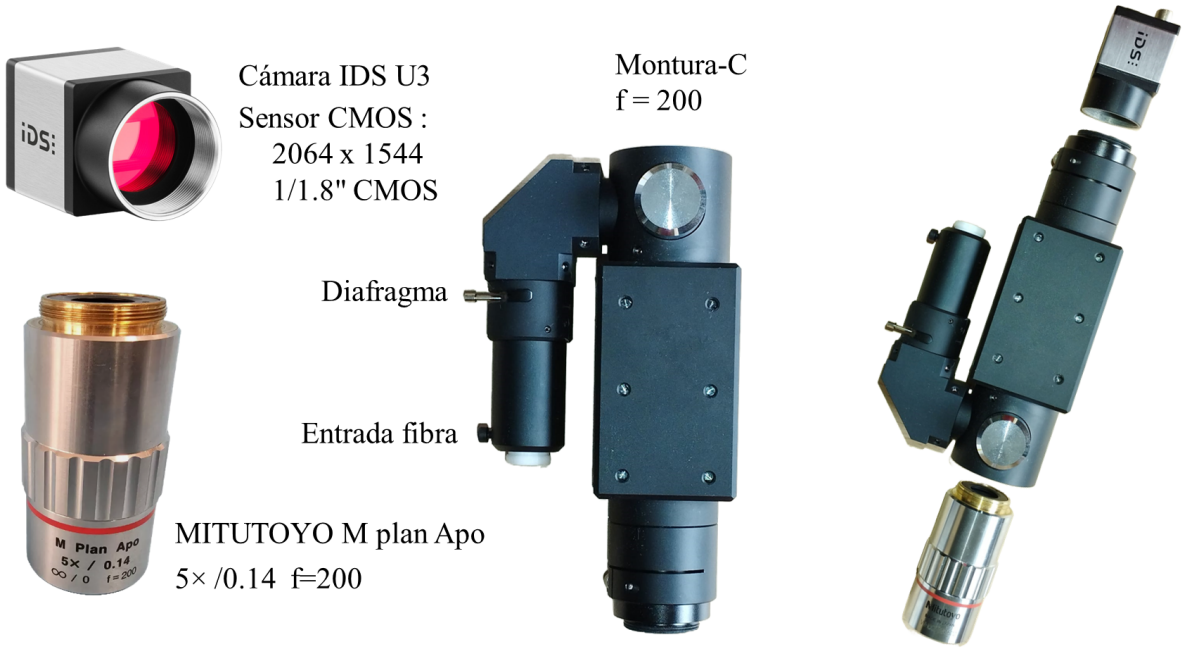

Figura 1: *Partes del sistema óptico y como se acoplan. Imágenes de cámara [28] y objetivo [29].*

.2. Software IDS

Para la utilización de la cámara, que inicialmente se empleará para medir la posición de puntos y marcas fiduciales, como la Fig. (3.5), fue necesaria la implementación de un *software* orientado a objetos. En primer lugar, se desea tomar fotografías y manipular todos los parámetros de la cámara desde códigos de Python, para lo que se creó la clase *IDS_Camera*. En segundo lugar, las fotografías deben permitir obtener el punto central de marcas fiduciales, y de los puntos de calibración (en la práctica orificios escariados[1] de 1 mm), para lo que fueron necesarios algoritmos básicos de edición de imagen, para facilitar este trabajo se generó la clase *Image*.

.2.1. Clase *IDS_Camera*

Para ejecutar la cámara correctamente, es necesarias la instalación de ciertas librerías gratuitas. En primer lugar se recomienda instalar la *suite software* desde la página de IDS utilizando el buscador para el modelo específico de la cámara[2]. Se seleccionará entre las opciones el instalador de software *IDS-peak standard setup* y se extraerá en el equipo. Así, se habrá instalado también el *IDS cockpit*, para manipular la cámara con *GUI*[3]. A continuación, es necesario crear una variable de entorno apuntando al *path* del archivo *cti* correspondiente (U3 y 64 bits) en la carpeta del software instalado. La variable se llamará:

```
GENICAM_GENTL64_PATH
```

Por último, para su utilización en Python bastará instalar la librería de la marca [30] con *pip*. Esta por si sola no permitiría la conexión a la cámara, razón por la que se han indicado los pasos anteriores.

```
pip install ids_peak
```

Para usar la cámara se debe inicializar la librería de *ids_peak* y crear una instancia del controlador de dispositivos [31]. Al actualizarlo, se puede detectar la cámara (si no está siendo utilizada por otra aplicación). A continuación, se abre la cámara y se establece el tipo de disparo (*software trigger*). Seguidamente, se puede iniciar la adquisición, período durante el cual se pueden tomar fotografías. Es necesario para esto definir el *buffer* y activar la adquisición a la espera de *triggers*. Se puede establecer el tiempo de exposición y a continuación tomar una imagen, ejecutando el trigger, extrayéndola del *buffer* y transformándola a formato color rgb. Finalmente, se puede guardar o procesar.

Al acabar, es necesario asegurar que los dispositivos estén cerrados. Para lo cual se cierra la librería *ids_peak*.

```
ids_peak.Library.Close()
```

Los anteriores pasos se englobaron en una clase *IDS_Camera*, en la cual se simplifica el uso de la cámara a:

a. Conectar la cámara al ordenador con USB.

b. Crear una instancia de la clase *IDS_Camera*.

[1] Técnica de mecanizado para refinar el diámetro final de orificios perfectamente cilíndricos y lisos.
[2] Camera uEye. Interface: USB3.0, Family: CP Rev2.2, Model: U3-3270CP-M-GL Rev 2.2.
[3] GUI: interfaz gráfica de usuario.

c. Ejecutar la función *start_camera()*, que inicializa la cámara desde cero hasta la adquisición.

d. Tomar fotografías con las funciones desarrolladas *get_image()* o *auto_exposure_get_image()*. La segunda de las cuales realiza varias fotografías para calibrar su exposición a las condiciones externas automáticamente.

e. Finalmente, ejecutar *close_device()*.

.2.2. Clase *Image*

Con el fin de editar las imágenes tomadas con la cámara, estas se crean como instancias de la clase *Image*. En esta, se almacena una copia original de la matriz de la imagen y se ofrecen múltiples métodos para procesarla. *Image* utiliza la librería Python *cv2* [32], que se puede instalar así:

```
pip install opencv-python
```

- **save()** Guarda la imagen en el *path* indicado. Las fotografía se identifican acorde a la fecha y hora de creación.
- **display()** Muestra la imagen en su último estado de edición por pantalla.
- **binarize()** Binariza la imagen, originalmente en escala de grises [0,255]. Utiliza como *treshold* la mitad (o el especificado) del máximo valor encontrado en la imagen y devuelve una fotogrfía en blanco (255) y negro (0)
- **soften()** Dado que la calidad de la binarización depende de lo bien que se haya ajustado la exposición de la cámara, esta función permite corregir las imperfecciones en la silueta de los objetos fotografiados (un fiducial por ejemplo). Realiza un promedio en la fotografía tomando un *kernel* del tamaño especificado por el usuario.
- **serch_cm()** Busca el "centro de masa" de un fiducial, u otra silueta, en una imagen binarizada que se entre en el campo de visión. Obteniendo de esta forma el centro de su posición de una forma precisa. Realiza un promedio al número de píxeles blancos en ambos ejes.
- **search_border()** Una vez binarizada la imagen, se puede realizar una búsqueda de bordes. Con este fin se buscan los píxeles frontera que estén rodeados por otros de distinto color. Estos se retornan como dos listas (x, y) y se muestra una imagen en pantalla de los bordes y la imagen.
- **circle_treatment()** Es fundamental para poder calibrar el robot en la situación real con orificios de $\varnothing = 0.9$ mm. Dados los bordes de una imagen binarizada con un circulo (agujero negro) sobre fondo blanco (el metal refleja la luz), se utiliza esta función para obtener el centro del circulo aún cuando este no se vea al completo. Se hace uso de la clase auxiliar *CircleFit*. Esta crea un modelo estadístico a partir de los puntos (x, y) del borde y obtiene mediante una minimización la posición del centro (x_c, y_c) y el radio r del círculo que mejor se ajusta a los datos.

.2.3. Ejemplos de uso

De esta forma, utilizar la cámara para hacer una fotografía puede ser tan sencillo como ejecutar:

```
my_camera = IDS\_Camera()
my_camera.start_camera()
my_camera.auto\_exposure\_get\_image()
fig = my_camera.image
# Edit and save the image
my_camera.close_device()
```

Además, con el postprocesado utilizando la clase *Image*, se puede obtener resultados como los que se presentan a continuación. Se incluyen las lineas de código ejecutadas y el resultado obtenido en las diferentes etapas.

```
fig.soften(16).display(True)
fig.binarize().display(True)
fig.find_cm()
fig.display(True)
```

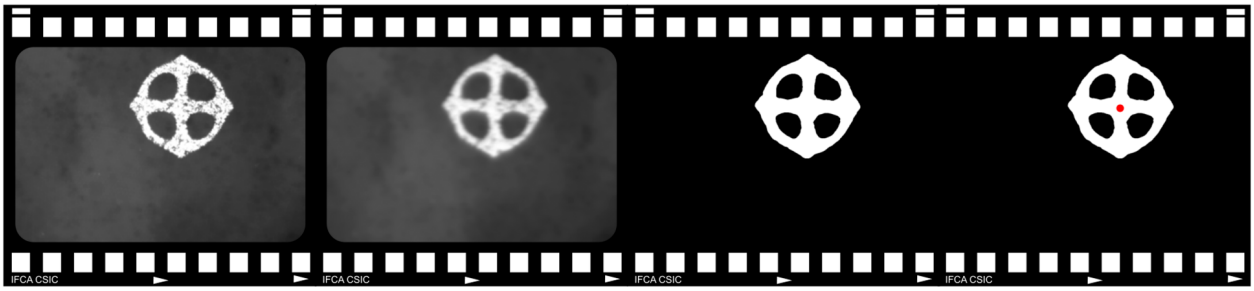

Figura 2: *Cálculo del centro de un fiducial fotografiado con la cámara IDS aplicando un filtro y binarización, ejecutando el anterior código. El programa retorna las coordenadas* (x, y) *del punto central numéricamente, además de como un punto rojo sobre la imagen.*

```
fig.binarize().display(True)
fig.find_cm()
fig.display(True)
```

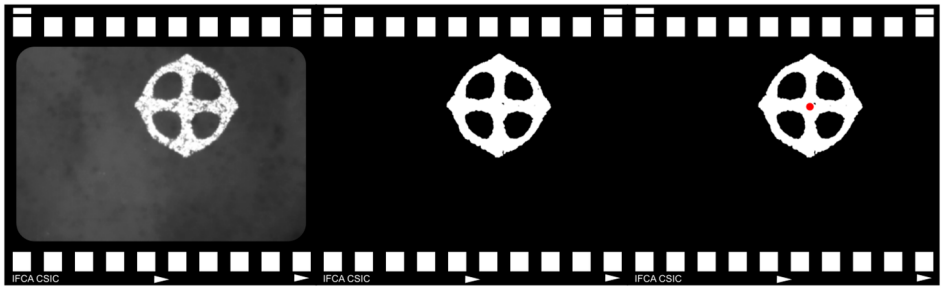

Figura 3: *Cálculo del centro de un fiducial fotografiado aplicando únicamente una binarización, en el código sobre la imagen. En este caso, comparando con la Fig. (2), como la imagen ha sido tomada con la exposición ideal, la binarización funciona bien y no es necesario utilizar un filtro.*

```
fig.soften(2).binarize(1.5).display(True)
fig.search_border(show=True, save=True)
fig.circle_treatment(show=True, save=True)
```

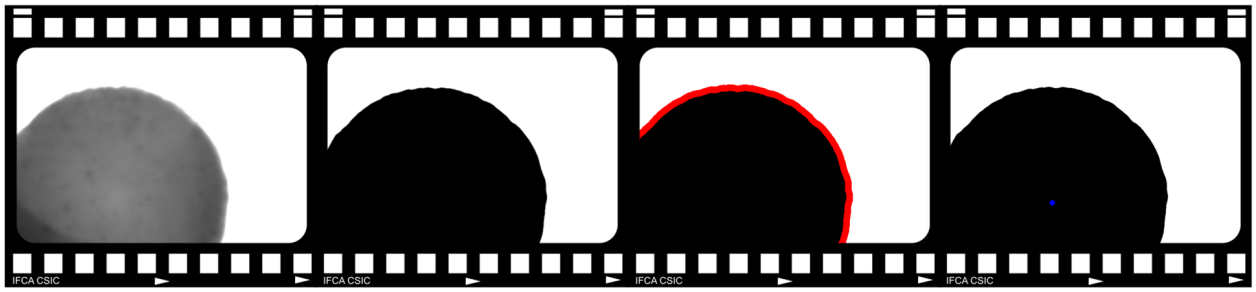

Figura 4: *Cálculo del centro de un orificio escariado de calibración, la imagen se ha quemado intencionadamente (foto sobreexpuesta) para que las imperfecciones de la superficie metálica fuesen menos apreciables. Se aplica un filtro soften para que el color de la imagen sea más uniforme antes de binarizar. En la binarización, se personaliza el treshold de tal modo que el círculo sea negro completamente. Finalmente, se ejecuta el buscador de bordes y se ajustan los puntos para obtener el centro del círculo de calibración, señalado en color azul sobre la última imagen. Ver código sobre la imagen.*

```
fig = Image(cv2.imread('./path_to_image.png'))

fig.soften(8).binarize(2).display(True)
fig.soften(16).binarize(2.5).display(True)
fig.search_border(show=True, save=True)
fig.circle_treatment(show=True, save=True)
```

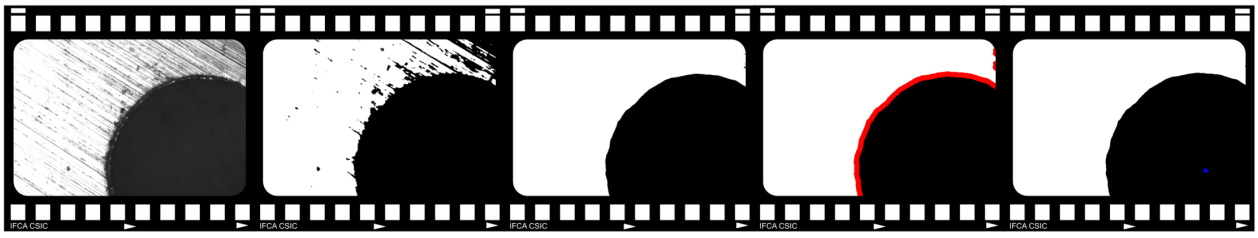

Figura 5: *Como se ve en el código sobre la figura, se abre una imagen bien expuesta de un punto de calibración y se utilizan las mismas operaciones que en la Fig. (4). En este caso se ejemplifica como el filtro soften y la binarización con treshold personalizable permiten obtener imágenes de calidad sobre las que poder buscar bordes (tomas 2 y 3). Igualmente, en la cuarta toma se ve como algunos puntos de borde yacen fuera del círculo, debido a las marcas sobre el material que no se logran corregir totalmente con el filtro. Sin embargo, mientras el número de estos puntos sea mucho menor al de puntos sobre el borde del círculo, el algoritmo de optimización, basado en un Likelihood, obtendrá buenos resultados. El centro del circulo se señala en azul sobre la quinta toma.*

El código para controlar la cámara y procesar imágenes se puede encontrar en el repositorio GitHub [18].

.3. Implementación del *Likelihood*

Estructura general de la función de *Likelihood* que se ha adaptado el problema, basada en las funcionalidades del módulo de Python *statsmodels.base.model* [15]. La estructura general de la clase se muestra en 1.

Para su utilización es necesario importar las siguientes librerías, fácilmente instalables con *pip*.

```
from statsmodels.base.model import GenericLikelihoodModel
import numpy as np
```

Una de las ventajas de la función utilizada es poder utilizar diferentes algoritmos numéricos-iterativos de minimización dada una función $log - likelihood$. Estos métodos incluyen *bfgs, newton, powell, minimize, ...* y también otros con una gran personalización como *basinhopping*, que facilita la optimización al poder seleccionar el usuario diferentes incrementos para ir modificando cada variable a optimizar y también un control T que marcará lo probable que será que el modelo abandone una mínimo del $log - likelihood$ en busca de mejores soluciones.

Si bien unos se centran en la aproximación de la matriz Hessiana[4] del modelo y otros realizan iteraciones aleatorias, la idea general de estos algoritmos es la de un descenso de gradiente muy eficiente. Dada una evaluación de la función de $log - likelihood$ con unos parámetros θ_j^1, si se calcula el gradiente $\nabla \vec{f}(\theta_j, x)$ de la misma, se podrá obtener un nuevo set de parámetros θ_j^2 que se aproximen más a la realidad variando los parámetros en la dirección de $-\nabla \vec{f}(\theta_j, x)$.

De entre todos los métodos disponibles para minimización, se recomienda el método *Powell*, al no ser la matriz Hessiana un requisito para su implementación y por la experiencia con su uso a lo largo de la calibración realizada. Se vio como alcanza mínimos de forma más rápida y reduciendo de manera efectiva el valor de χ^2 frente a otros de los métodos disponibles.

[4]Hessiana: matriz cuadrada con las derivadas parciales de segundo orden de una función escalar con respecto a sus variables independientes.

```python
class MyLikelihood(GenericLikelihoodModel):

    def __init__(self, endog, exog, args, **kwds):
        """
        Generador de la clase.
            exog    Su valor es externo al modelo, ej: tiempo)
            endog   Su valor depende de otros parámetros del modelo, ej: posicion)
            args    Otros argumentos, p.e. el robot para hacer medidas
            **kwds  Parametros iniciales (~ reales) a partir de los cuales minimizo
        """
        self.n = len(endog)
        self.endog = np.asarray(endog)
        self.exog = np.asarray(exog)

        super(MyLikelihood, self).__init__(self.endog, self.exog, **kwds)

    def loglike(self, params):
        """
        Calcula y retorna el valor del loglikelihood dados los parametros del modelo
        """
        # Actualizar los valores de los parametros
        self.a = params[0]
        self.b = params[1]

        chi2 = 0.0

        for i in range(0, self.n):
            # Se comparan todas las medidas y calcula el valor de la función de coste
            chi2 += ...

        return -chi2

    def fit(self, start_params = None, method='powell', maxiter = 10000, **kwargs):
        """
        Dada la función de loglike y unos parámetros iniciales realiza la optimización.
        Se define un máximo de iteraciones para el método escogido:
        methods = bfgs, lbfgs, nm, newton, powell, cg, ncg, basinhopping, minimize, ...
        """
        if start_params is None:
            # Valores iniciales razonables para los parámetros
            start_params = [..., ...]

        return super(MyLikelihood, self).fit(start_params=start_params, method=method,
                                             maxiter=maxiter, **kwargs)
```

Listing 1: *Función general de Likelihood en Python.*

.4. Histogramas y puntos de calibración

En este anexo, se incluye una imagen con la distribución de los puntos de calibración sobre la mesa del robot, Fig. (6). Es importante destacar el papel que el tamaño de estos orificios ha tenido en el proceso de calibración.

Los orificios tal y como aparecen posicionados en la Fig. (6) corresponden con las posiciones de los orificios de la mesa real del SCARA, estos son de métrica M8 ($\varnothing = 8$ mm). El campo de visión de la cámara es de tan solo $fov_{xy} = (1.4 \text{ mm}, 1.1 \text{ mm})$, por lo que resulta imposible trabajar con estos orificios. No obstante, esta fue la motivación inicial para crear la función introducida en el Anexo .2 *circle_treatment()*, de la clase *Image*.

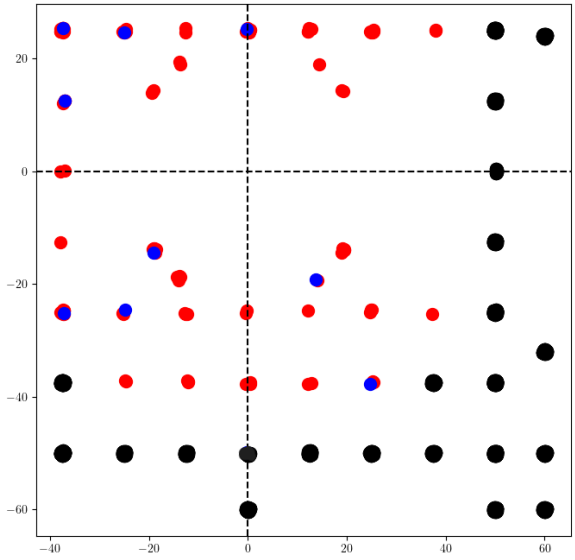

Figura 6: *Esquema de los puntos de referencia sobre la mesa utilizados en una de las calibraciones de la cámara mesa. En color azul aparecen los 10 (2 solapan) utilizados para la calibración, en rojo los 400 puntos de test, y en color negro los no accesibles para el robot. La cruz punteada marca el centro de la mesa, donde está el eje J_1 del SCARA.*

Para poder calibrar el robot, los expertos en mecanizado del grupo CMS han diseñado placas con métricas precisas que se pueden montar con pasadores de tolerancia baja en los orificios M8 de la mesa. Estas placas, de aluminio, tendrán una matriz de orificios con coordenadas conocidas con una alta precisión. Su diámetro es de $\varnothing = 0.9$ mm y para su perforación se combina una broca y un escariador, de forma que resultan círculos perfectos.

Así, los puntos que se utilizarán para la calibración real del robot serán como el que aparece en la primera imagen de la Fig. (5). Entonces, la función *circle_treatment()* sí resultará útil, al poder determinar el centro de los orificios de calibración aún cuando estos no se fotografían por completo.

Ahora bien, en la simulación, para conseguir más de 500 puntos sobre la mesa, se añadió una matriz circular de radio $r = 4$ mm con puntos alrededor de cada uno de los centros de los orificios M8. Por lo que cada mancha en la Fig. (6) está compuesta de varios puntos.

También se recogen, gráficos con la diferencia en medida entre los robots calibrado y real para los test de calibración de la Tab. (3.2). Estos aparecen en las Fig. (7 y 8) de las próximas dos páginas.

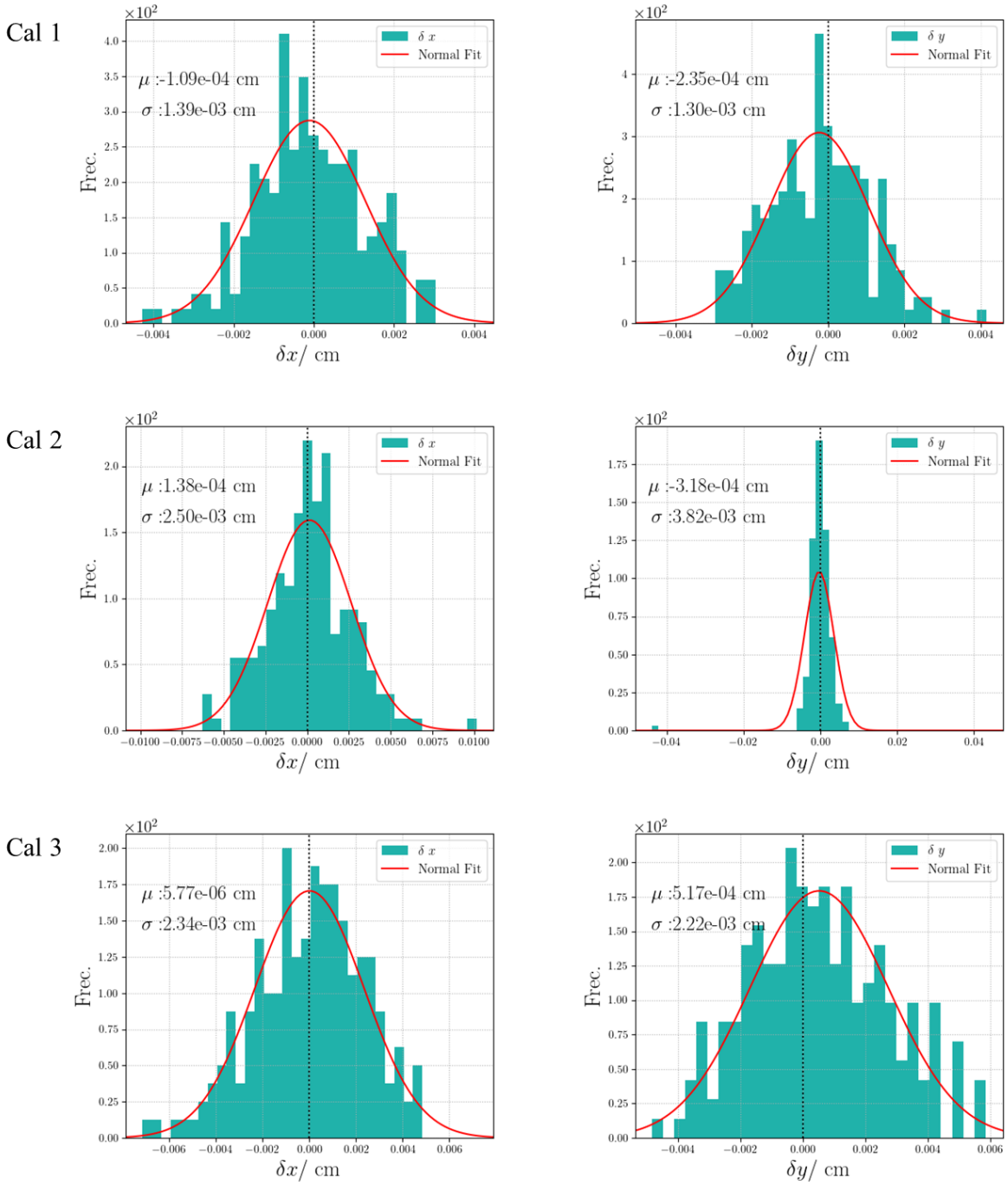

Figura 7: *Histogramas de dispersión de la diferencia de las medidas realizadas con un robot real simulado con sus parámetros reales y los calibrados mediante el sistema desarrollado. Corresponden a las calibraciones 1 a 3 de la Tab. (3.2), en cada caso se representa la desviación en el eje $\vec{x}_C$ e $\vec{y}_C$ de la cámara. En cada uno, se representa el ajuste a una distribución de probabilidad Gaussiana, y se señala el valor medio (μ) y desviación estándar del ajuste σ.*

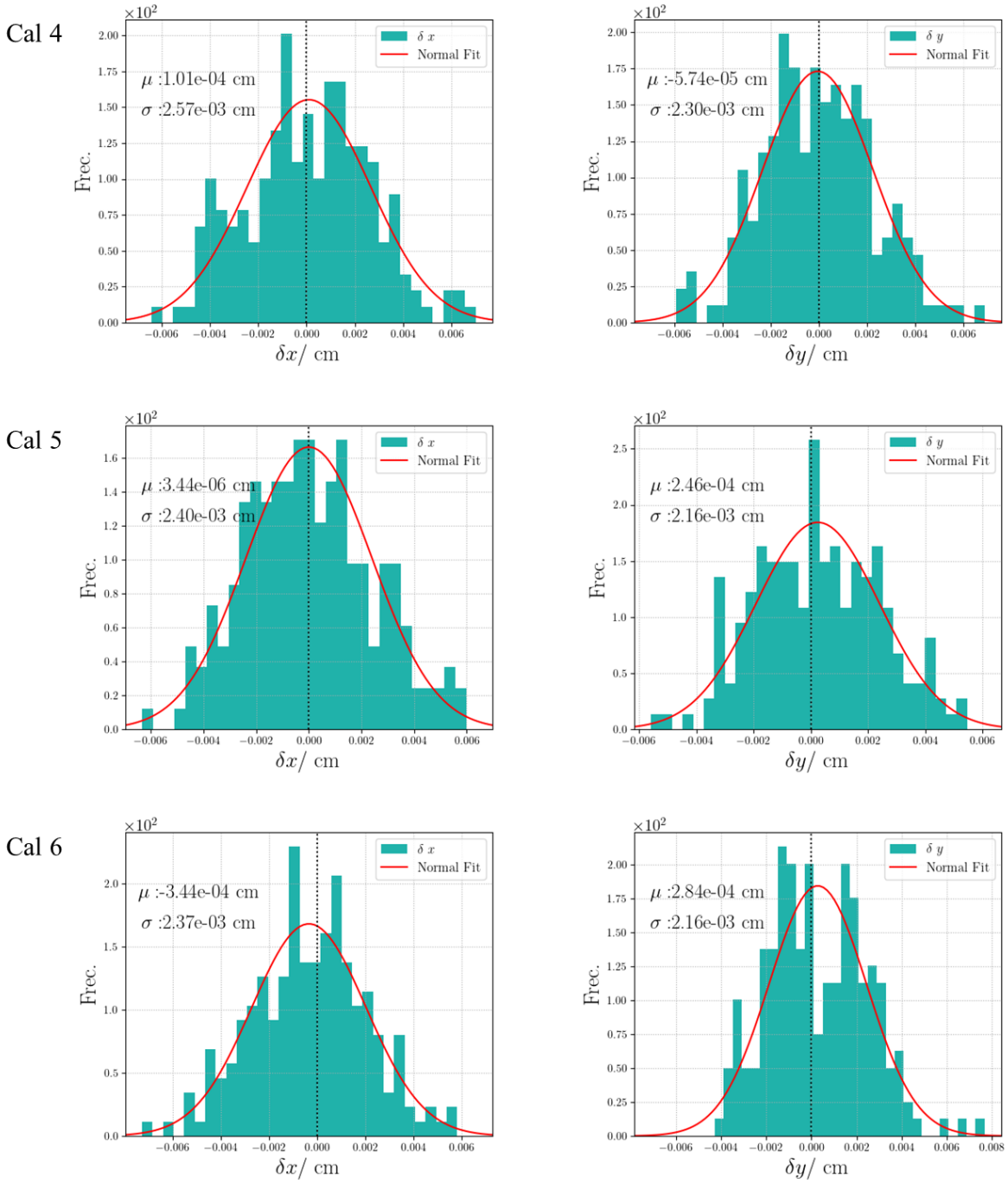

Figura 8: *Histogramas de dispersión de la diferencia de las medidas realizadas con un robot real simulado con sus parámetros reales y los calibrados mediante el sistema desarrollado. Corresponden a las calibraciones 4 a 6 de la Tab. (3.2), en cada caso se representa la desviación en el eje $\vec{x}_C$ e $\vec{y}_C$ de la cámara. En cada uno, se representa el ajuste a una distribución de probabilidad Gaussiana, y se señala el valor medio (μ) y desviación estándar del ajuste σ.*

.5. Algoritmo de POCA

El algoritmo de POCA, *Point of Closest Approach*, es ampliamente utilizado en tomografía muónica, cuyos principios básicos son similares a la protónica. Este algoritmo estima, a partir de la trayectoria de un partícula cargada antes y después de atravesar un cuerpo, el lugar donde la partícula ha interactuado dentro del volumen escaneado. Para lo que se aproxima que la partícula solo ha sufrido scattering en ese punto.

El algoritmo intenta determinar el punto más probable donde ha ocurrido la dispersión, para lo cual se considera que las trayectorias de entrada y salida de la partícula son rectas. El algoritmo busca el punto de máximo acercamiento entre ambas rectas, el cual se aproxima por el punto donde ha ocurrido la mayor dispersión.

El modelo cuenta con ciertas limitaciones al simplificar el problema geométricamente, considera que la dispersión solo ocurre en un punto cuando puede haber varias. En contrapartida, se trata de un algoritmo muy rápido y computacionalmente eficiente, permitiendo realizar reconstrucción tridimensional en problemas complejos. Su máximo potencial se aprovecha combinándolo con otros métodos de reconstrucción más complejos, como el *Filtered Back Projection*.

Su resolución matemática es simple y se puede atajar de diferentes formas. En este caso, se ha expresado dos trayectorias $\vec{a}$ y $\vec{b}$, que serán la trayectoria de entrada y salida y se ha minimizado la distancia entre sus puntos.

$$\begin{cases} a: \ \vec{x}_a + \alpha \ \vec{v}_a \\ b: \ \vec{x}_b + \alpha \ \vec{v}_b \end{cases} \tag{1}$$

Se expresa el vector $\vec{\delta}(\alpha, \beta)$ conectando dos puntos cualquiera, Ec. (2), y se toma $\vec{d} = (\vec{x}_a - \vec{x}_b)$:

$$\vec{\delta}(\alpha, \beta) = (\vec{x}_a - \vec{x}_b) + (\alpha \ \vec{v}_a - \beta \ \vec{v}_b) \tag{2}$$

La distancia entre dos puntos cualesquiera de las trayectorias será la norma de este vector, como el objetivo es minimizar la distancia, se simplifica tomando la norma al cuadrado $||\vec{v}||^2 = \vec{v} \cdot \vec{v}$:

$$D^2(\alpha, \beta) = ||\delta(\alpha, \beta)||^2 = \vec{d} \cdot \vec{d} \ + \ 2\alpha \ \vec{d} \cdot \vec{v}_a \ - \ 2\beta \ \vec{d} \cdot \vec{v}_b \ + \ \alpha^2 \ ||\vec{v}_a^2|| \ + \ \beta^2 \ ||\vec{v}_b^2|| \ - \ 2\alpha\beta \ \vec{v}_a \cdot \vec{v}_b \tag{3}$$

Para obtener la mínima distancia, se deriva con respecto a los parámetros α y β e iguala a cero:

$$\begin{array}{l} \frac{\partial D^2(\alpha,\beta)}{\partial \alpha} = 2\vec{d} \cdot \vec{v}_a \ + \ 2\alpha ||\vec{v}_a||^2 \ - \ 2\beta \vec{v}_a \cdot \vec{v}_b = 0 \\ \frac{\partial D^2(\alpha,\beta)}{\partial \beta} = -2\vec{d} \cdot \vec{v}_b \ + \ 2\beta ||\vec{v}_b||^2 \ - \ 2\alpha \vec{v}_a \cdot \vec{v}_b = 0 \end{array} \tag{4}$$

Se despeja y llega a que:

$$\alpha = \frac{-\vec{d} \cdot \vec{v}_a + \beta \vec{v}_a \cdot \vec{v}_b}{||\vec{v}_a||^2} \qquad\qquad \beta = \frac{\vec{d} \cdot \vec{v}_b + \alpha \vec{v}_a \cdot \vec{v}_b}{||\vec{v}_b||^2} \tag{5}$$

Entonces, se definen los términos escalares: $A, B, C, ...$ para mayor simpleza:

$$A = \vec{d} \cdot \vec{v}_a \quad , \quad B = ||\vec{v}_a^2|| \quad , \quad C = \vec{v}_a \cdot \vec{v}_b \quad , \quad D = \vec{v}_b \cdot \vec{d} \quad , \quad E = ||\vec{v}_b^2|| \quad \& \quad F = C$$

Llegando a un sistema:

$$\begin{cases} A + \alpha B - \beta C = 0 \\ -D + \beta E - \alpha F = 0 \end{cases} \tag{6}$$

A continuación, este sistema se resuelve por reducción llegando a expresiones en función de los escalares definidos para β y α:

$$\alpha = \frac{-AF + DB}{EB - FC} \qquad\qquad \beta = \frac{DC - AE}{BE - FC} \tag{7}$$

De esta forma, sustituyendo α y β en la Ec.(1), se puede obtener un punto $(x, y, z)_1$ y $(x, y, z)_2$ sobre cada una de las dos trayectorias, que son los más próximos de ambas rectas entre sí. Finalmente, se define el POCA como el punto medio entre ambos:

$$\text{POCA} = \frac{(x, y, z)_1 + (x, y, z)_2}{2} \tag{8}$$

Este algoritmo se implementó en un ejecutable de Python con opciones. Sin embargo, finalmente no se trabajó con imágenes de POCA, pues estás informaban más sobre la transmitancia que sobre el scattering en los fantomas, como se ve en la Fig. (9). Idealmente, se habría esperado un alto número de trazas de protones sufriendo scattering en los *phantoms.*

Figura 9: *Ejemplo de imagen de POCA, obtenida con las misma trazas que las imágenes de atenuación de la Fig. (4.4).*

El algoritmo es ampliamente utilizado con muones. Son partículas fundamentales, leptones con masa m_μ =106 MeV/c^2, que únicamente interaccionan débil y electromagnéticamente con el medio que atraviesan. Estas partículas atraviesan fácilmente edificios o volcanes gracias a su alta capacidad de penetración.

Los protones en cambio, son hadrones compuestos, con una masa m_μ =938 MeV/c^2. Estas partículas interaccionan además fuertemente con los núcleos, sufriendo dispersión elástica e inelástica. Su penetración es muy inferior a la de los muones. Esta es la razón por la que se cree que el POCA no ha sido tan efectivo como se esperaba.

El código desarrollado para la reconstrucción POCA se puede encontrar en el repositorio GitHub [27].