Revista ELECTRO, Vol. 46, 2024, pp. 37 - 42
https://itchihuahua.mx/revista_electro
ISSN Electrónico: 3061-774X

# IDENTIFICACIÓN DE RADIOISOTOPOS EN TIEMPO REAL APLICANDO MÍNIMOS CUADRADOS USANDO C++ PARA UNA ARQUITECTURA EMBEBIDA

Pacheco, Alberto; Rivera Mariano*; Chacón Raúl; Robledo, Isidro
*TecNM campus Chihuahua, Av. Tec. 2909, Chihuahua, Chih. 31310*
{alberto.pg, raul.cb, isidro.rv}@chihuahua.tecnm.mx
**Centro de Investigacion en Matematicas AC, Guanajuato, Gto. 36160*
*mrivera@cimat.mx

## RESUMEN.

**Se presenta una implementación eficiente de un método de análisis espectral de rayos gamma de baja resolución aplicando mínimos cuadrados no-negativos para identificar radioisótopos en tiempo real en una plataforma embebida por medio de una librería de C++ altamente optimizada para procesadores ARM64 utilizando el entorno de desarrollo Xcode 15 para iOS 17 mezclando códigos escritos en lenguaje C++ 17 dentro del lenguaje nativo Swift, explo–tando las capacidades propias de la arquitectura de compilación LLVM. Se alcanzaron tiempos de ejecución del algoritmo RIID inferiores a los reportados en la literatura. Se discuten posibles mejoras tanto del algoritmo como del conjunto de datos de prueba.**

**Palabras Clave: Identificación de radioisótopos, análisis espectral de rayos gamma, mínimos cuadrados, sistema embebido.**

## ABSTRACT.

**A very fast embedded software implementation of a low-resolution gamma ray spectral analysis method using non-negative least squares for real-time radioisotope identification is presented. This implementation was carried out using a highly optimized C++ library for ARM64 processors using the Xcode 15 development environment under iOS 17, mixing C++ 17 code with native Swift language, exploiting the capabilities offered by LLVM compilation architecture. Average execution times of the RIID algorithm were lower than those reported in the literature. Possible improvements to both the algorithm and the testing datasets are discussed.**

**Keywords: radioisotope identification (RIID), non-negative least squares, gamma ray spectral analysis, embedded system.**

## 1. PROBLEMÁTICA

La identificación de radioisótopos (*radioisotope identification*, RIID) es importante en una amplia variedad de aplicaciones, tales como la prevención, vigilancia y asistencia en desastres, transporte de material ilícito, contaminación radioactiva, manejo de radioisótopos dentro de la medicina, materiales de construcción, procesos industriales, yacimientos minerales, energía nuclear, entre otros [1-4].

El objetivo del trabajo consiste en implementar un método eficiente y apto para arquitecturas embebidas capaz de identificar radioisótopos a partir de espectrogramas de rayos gamma (γ) de baja resolución provenientes de un detector tipo NaI.

## 2. HERRAMIENTAS Y MARCO TEÓRICO

### 2.1. Herramientas para software embebido.

El compilador Clang, compatible con GCC (Fig. 1), introducido en 2007 como parte del proyecto LLVM para soportar lenguaje C, C++, Objective-C. LLVM soporta a su vez Swift, Rust, Python, R, D, Go, Lua, CUDA, Julia, Haskell, entre otros. Clang ha destacado en la generación de código nativo de alta calidad y su velocidad de compilación es, en muchos casos, superior a GCC. LLVM y Clang son conocidos por ofrecer mensajes de error muy detallados y claros, lo que facilita la depuración del código, ofreciendo una sólida integración con los modernos entornos de desarrollo (IDE), como Xcode.

```
> gcc --version
Apple clang version 15.0.0 (clang-1500.0.40.1)
Target: arm64-apple-darwin23.5.0
```

Figura 1. Compilador C/C++ disponible en macOS.

LLVM (*low level virtual machine*) es un ecosistema modular de compiladores y herramientas (*toolchain technologies*), diseñadas para optimizar y generar código a partir de múltiples lenguajes de programación, sirviendo como *toolkit* para el desarrollo de compiladores avanzados mediante una sola representación intermedia de código

(*intermediate representation*, IR), como se ilustra en la Fig. 2.

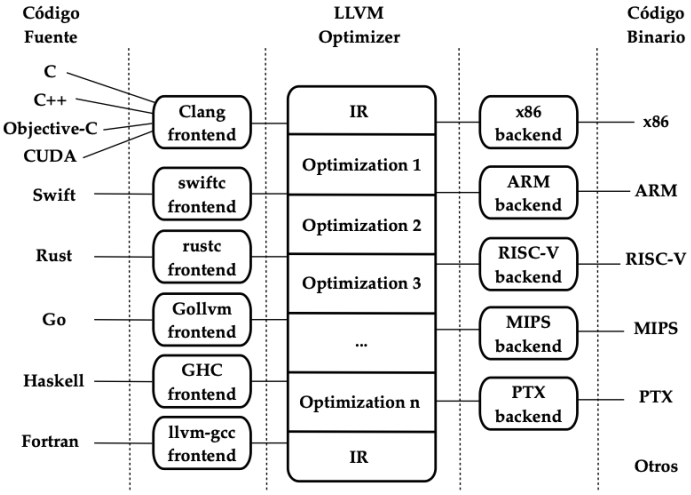


Figura 2. Arquitectura LLVM soportando diversos lenguajes y plataformas vía IR optimizado [6].

LLVM fue iniciado por Chris Lattner en 2000 en la Universidad de Illinois, en sus tesis de maestría y doctorado [5, 6]. En 2005 fue contratado por Apple para mejorar sus herramientas de desarrollo, suplantando en 2008 el compilador GCC por el compilador Clang más moderno, eficiente y modular, hasta culminar en 2014 en la introducción del lenguaje Swift, reemplazando a Objective-C como el lenguaje oficial en las plataformas de Apple macOS, iOS, iPadOS, watchOS, tvOS y visionOS. Si bien desde 2014 en Xcode 6 existía la capacidad de integrar código C++ en proyectos escritos en Swift, es a partir de Xcode 15 (2023) donde es posible mezclar código de Swift y C++ dentro de un mismo proyecto, es decir, llamar de forma más simple a funciones de C++ desde Swift y viceversa (Swift/C++ *interoperability*) [7].

A partir de 2005 se incorpora LLVM *embedded toolchain*, convirtiéndose en una característica clave, especialmente dada la popularidad de los dispositivos móviles y embebidos que utilizan procesadores ARM, ofreciendo optimizaciones y mejoras de rendimiento, tales como: elegir instrucciones con menor consumo de energía, minimizar accesos a memoria mediante el reúso de registros y alineación de datos, reducir el tamaño de código con instrucciones de 32 bits a sus equivalentes en 16 bits (ARM *thumb mode*), planificar flujo eficiente de instrucciones (*pipeline scheduling*), insertar código en línea, eliminar código no usado y vectorizar código.

### 2.2. Librería Eigen.

La librería Eigen de código abierto [8] esta escrita en lenguaje C++ para realizar operaciones algebraicas sobre matrices y vectores con alto rendimiento, versatilidad y facilidad de uso. Eigen es ampliamente usada en áreas como álgebra lineal, optimización, simulaciones científicas, motores de juego, procesamiento de señales y aprendizaje automático. Entre los proyectos que utilizan Eigen destacan: OpenCL, Dlib, Qt, TensorFlow, OpenCV, Ceres Solver, Bullet Physics, ROS (*robot operating system*) y Unreal Engine.

### 2.3. Detectores de rayos gamma.

Para el análisis espectral de un radioisótopo (RI) basado en rayos gamma (γ) es posible usar detectores de baja resolución basados en cristales de yoduro de sodio (NaI) [9]. Al interactuar los rayos γ con los átomos del cristal NaI pueden ocurrir los siguientes efectos: 1) el efecto fotoeléctrico: donde el rayo γ transfiere toda su energía a un electrón en el cristal, lo que excita al electrón y lo expulsa del átomo; 2) la dispersión de Compton: el rayo γ transfiere parte de su energía a un electrón, cambiando su dirección y energía; 3) la producción de pares: al presentarse rayos γ de alta energía (arriba de 1.022 MeV), al incidir en el núcleo del átomo crean pares electrón-positrón.

El efecto de dispersión Compton ocurre cuando un rayo γ de alta energía incide sobre un electrón libre o débilmente ligado en un átomo (Fig. 4). A diferencia del efecto fotoeléctrico, donde la energía del rayo γ es absorbida completamente por el electrón, en el efecto Compton, parte de la energía del rayo γ es transferida al electrón, y el resto es dispersado en forma de un rayo γ de menor energía. Este efecto es crucial para la

física de partículas y el diagnóstico de imagen médica, así como para la astrofísica y otras áreas de la ciencia.

La ecuación de Compton relaciona el cambio en la longitud de onda del rayo γ con el ángulo de dispersión $\theta$ del rayo γ. La ecuación es la siguiente:

$$\Delta\lambda = \lambda' - \lambda = \frac{h}{m_e c}(1 - \cos\theta)$$

donde:

- $\Delta\lambda$ es el cambio de longitud de onda del rayo γ,
- $\lambda$ es la longitud de onda inicial del rayo γ,
- $\lambda'$ es la longitud de onda después de la dispersión,
- $h$ es la constante de Planck,
- $m_e$ es la masa del electrón.
- $c$ es la velocidad de la luz,
- $\theta$ es el ángulo de dispersión del rayo γ.

Acorde al principio de conservación de energía y del momento, parte de la energía del rayo γ se convierte en energía cinética del electrón, originando un cambio de ángulo y un aumento en la longitud de onda del rayo γ dispersado (*scattered photon*) debida a la disminución de energía del rayo γ incidente (Fig. 3).

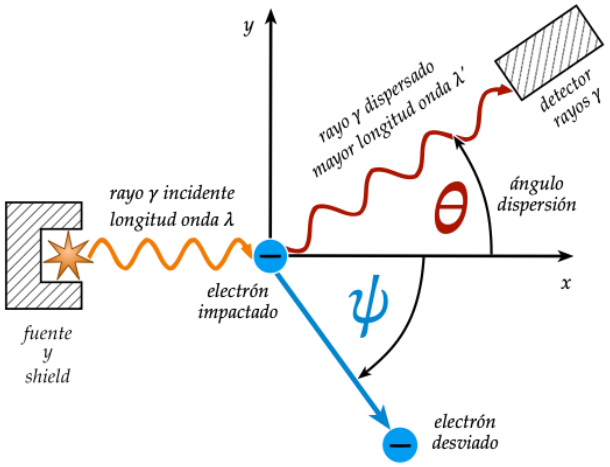


Figura 3. Efecto de dispersión Compton.

Los efectos fotoeléctricos, dispersión de Compton y producción de pares serán captados por el detector de rayos γ para producir un espectro de energía (Fig. 4), donde aparecen fotópicos (*photopeaks*) cuando la energía completa del rayo γ incidente es transferida a un electrón por el efecto fotoeléctrico, ocasionando una cascada de fotones de luz que son detectados y convertidos a una señal eléctrica proporcional a dicha energía. Por otro lado, los fotones dispersos del efecto Compton, pueden contribuir a la formación de un espectro continuo de energía inferior al fotópico (*Compton continuum*). Ocasionalmente, la producción de pares puede originar un pico de aniquilación (0.511 MeV) o un pico de escape sencillo o escape doble. El espectro mostrará además otras fuentes de ruido: ruido electrónico, del sensor, de absorción, dispersión múltiple y si no cuenta con un escudo (*shield*), aparecerá un ruido o radiación de fondo [9, 10].

Identificar un radioisótopo (RI) a partir de un espectro gamma proveniente de un detector NaI (NaI γ *spectra*), implica analizar los patrones del espectro y compararlos con espectros RI conocidos [10], ya sea aplicando técnicas simples como la identificación de picos hasta el uso de una red neuronal (ANN), siendo los primeros poco precisos, y un modelo ANN es opaco, pesados y complejo pocas veces susceptible de implementar en un sistema embebido de bajo costo [9-17]. Un espectro gamma (Fig. 4) consiste en un histograma de cuentas por segundo (eje *y*) para cada nivel de energía (eje *x*). Un radioisótopo tiene como propiedad principal la presencia de uno o más picos en regiones muy específicas de energía. Por ejemplo, el Cesio-137 tiene un pico a 662 keV y su región de Compton (fotones dispersos) aparece debajo de los 662 keV (Fig. 4). El Cobalto-60 en cambio, tiene dos picos (1.17 MeV y 1.33 MeV).

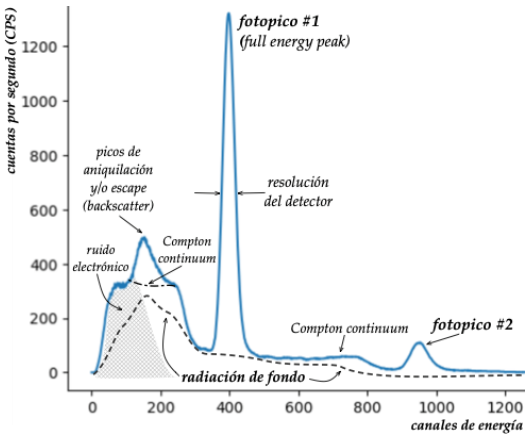

Figura 4. Espectro de energía para rayos gamma.

### 2.4. Análisis espectral.

El método de mínimos cuadrados no-negativos (*non-negative least squares*, NNLS por sus siglas en inglés) es una técnica matemática y de optimización numérica utilizada para resolver problemas de mínimos cuadrados con restricciones de no negatividad en las variables. Este método de optimización es de utilidad para encontrar la mejor aproximación a un conjunto de datos minimizando la suma de los cuadrados de las diferencias entre los valores observados y los valores estimados.

En espectrometría, los espectros observados pueden ser el resultado de la convolución de los espectros originales con la función de respuesta del instrumento de medición. La deconvolución puede ser utilizada para obtener un espectro más preciso, revelando detalles finos que podrían estar ocultos.

El problema específico de detección de radioisótopos (RIID) a partir de NaI γ *spectra* puede formularse como sigue:

$$signal \approx coeff \cdot \theta$$

donde:

- *signal* es el vector de observaciones (señal),
- *coeff* es el vector de coeficientes (candidatos),
- $\theta$ es una matriz de materiales puros.

Para estimar los posibles candidatos (*coeff*), sujeto a que todo $coeff_i \geq 0$, el método NNLS minimiza la función de error cuadrático:

$$min_{coeff} \|coeff \cdot \theta - signal\|_2^2$$

Es decir, mediante NNLS se busca minimizar la diferencia cuadrática entre la señal observada y la señal reconstruida bajo la restricción de que las magnitudes del vector *coeff* no sean negativas.

Por su parte, para implementar el método NNLS, la librería de C++ Eigen [8] ofrece diversos *solvers* de mínimos cuadrados para poder abordar diferentes tipos de problemas de ajuste y optimización.

## 3. METODOLOGÍA

Para identificar un radioisótopo (RIID) por medio del análisis espectral de rayos gamma aplicando el método de mínimos cuadrados no-negativos se consideró la siguiente metodología (algoritmo):

- Recopilación de datos: adquisición de los datos espectrales de cada material (*endmembers* RI).
- Selección de *endmembers* RI**:** determinar las firmas espectrales de los *endmembers*.
- Aplicación del método NNLS: resolver el problema de mínimos cuadrados (*solver*).
- Reconstrucción y análisis: una vez obtenido el vector de coeficientes *a*, se puede analizar por orden de precisión los *endmembers* RI identificados.
- Visualización e interpretación: en caso de éxito, mostrar el resultado obtenido (mejor candidato RI).

## 4. DESARROLLO

Se utilizó la librería de C++ Eigen v3.4 [8] dentro de un proyecto Xcode 15. Se revisó el código fuente de librería Eigen para eliminar incompatibilidades y buscar la versión soportada (C++17), para configurar Clang (Fig. 5).

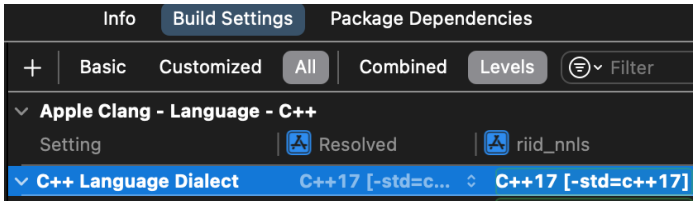


Figura 5. Configuración de Clang para Eigen/C++.

Para el conjunto de datos de *endmembers* se consideraron 5 variantes diferentes de 3 radioisótopos (en total: 15 firmas de 256 canales) y se incorporaron como recursos tipo archivo en proyecto Xcode. Al inicio del programa, un código en Swift lee archivos y en un módulo en C++ se define la matriz `Phi` de tipo `Eigen::MatrixXd` y los vectores `Signal` y `Coeff` de tipo `Eigen::VectorXf` (Figs. 6 y 7).

```
static  MatrixXd  Phi(SPEC_SIZE, PHI_ROWS);
static  VectorXf  Signal(SPEC_SIZE);
static  VectorXf  Coeffs(PHI_ROWS);
```

Figura 6. Declaración de variables en Eigen/C++.

A partir de una base de datos de prueba de 100 muestras normalizadas, una rutina en Swift selecciona

una muestra aleatoria como señal de entrada (`Signal`) para luego ejecutar el *solver* NNLS.

Para ello fue necesario convertir la señal de entrada como vector de 256 enteros en Swift a un vector de flotantes en C++ (inicio de la secuencia en la Fig. 8). Posteriormente, usando dicho vector de entrada `Signal` y la matriz de isótopos puros `Phi`, se obtiene el vector de candidatos `Coeffs` invocando desde C++ al *solver* NNLS `Eigen::FullPivHouseholderQR`, tal y como se muestra en la última secuencia de la Fig. 7, quedando como sigue:

```
Coeffs=Phi.colPivHouseholderQr().solve(Signal)
```

Dicho solver NNLS de Eigen usa descomposición QR (matriz ortogonal y triangular superior) para resolver sistemas de ecuaciones y problemas de mínimos cuadrados con pivoteo para mayor robustez y precisión.

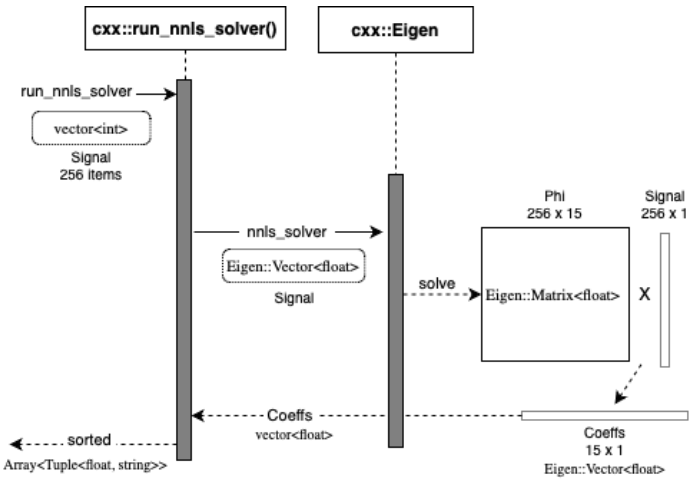


Figura 7. Secuencia UML para solver NNLS Eigen/C++.

De acuerdo con la metodología establecida y tal como se muestra en la parte final de la secuencia de la Fig. 7, se procede a elegir el mejor candidato obtenido en el vector de coeficientes, siempre y cuando dicho coeficiente sea superior a un umbral mínimo.

Finalmente, mediante SwiftUI se visualizan en la pantalla del dispositivo móvil basado en iOS, los espectrogramas gamma correspondientes, así como el resultado obtenido del análisis espectral para identificar el radioisótopo a partir del espectro de entrada presente en el vector `Signal` analizado (Fig. 8).

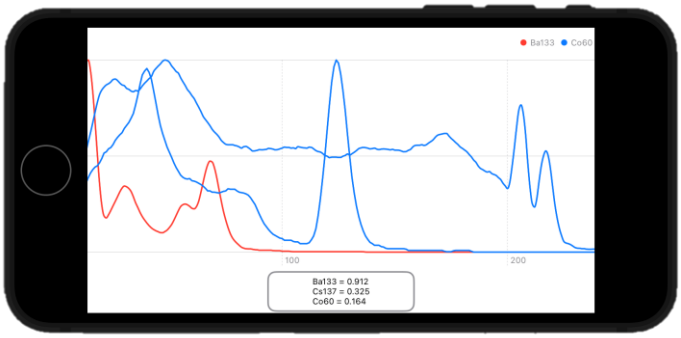


Figura 8. Visualización de resultados en la pantalla del dispositivo móvil.

## 5. RESULTADOS

Para determinar el rendimiento del procedimiento implementado, se incorporó un ciclo para invocar 100 veces el *solver* NNLS de Eigen/C++ con muestras RI aleatorias de entrada. Para ello se utilizó la siguiente función de la librería estándar `<chrono>` de C++:

```
chrono::steady_clock::now()
```

Tras ejecutar el programa 20 ocasiones, invocando a su vez 100 veces el *solver* NNLS en cada ocasión, se obtuvo un tiempo de corrida promedio de 2.9 ms ±0.25 ms, resultando hasta un orden de magnitud inferior a los reportados en la literatura [9, 10], logrando identificar con éxito el total de las muestras de prueba con un coeficiente promedio de 85% ±5%.

## 6. CONCLUSIONES Y COMENTARIO FINAL

A pesar de las dificultades encontradas por la ausencia de documentación, códigos muestra y numerosos detalles de configuración para la reciente capacidad de Xcode/LLVM para mezclar código C++ en aplicaciones nativas de Swift (*C++ interoperatibility*) se logró el objetivo de implementar un código más eficiente y robusto en una plataforma embebida (iOS) basada en un procesador ARM64, gracias a la incorporación de la librería Eigen para álgebra lineal optimizada en C++, que permitió

implementar, de manera muy eficiente y novedosa, un método de análisis espectral de rayos gamma de baja resolución mediante mínimos cuadrados no-negativos (NNLS) para identificar radioisótopos (RIID) en tiempo real.

Para lograr un sistema RIID más confiable será necesario incrementar a futuro el tipo y número de muestras RI, considerando variaciones en las condiciones de lectura, efecto Compton, calibración de detectores NaI y ruido. Por otra parte, el alto desempeño alcanzado posibilita un refinamiento iterativo del algoritmo RIID, mismo que está siendo explorado para incorporar la detección de mezclas RI usando descomposición espectral.

LLVM continua evolucionando, y muy recientemente (2024), se anunció la posibilidad de desarrollar sistemas embebidos utilizando un dialecto de lenguaje Swift [18, 19], por lo que sin duda, vale la pena seguir explorando las tecnologías más recientes de compilación disponibles para desarrollar software embebido por medio de C++ (Clang), Swift y LLVM.

## Agradecimientos.

Este trabajo formó parte del proyecto "*Ajuste de mo–delos basados en ciencia de datos y procesamiento de señales para algoritmos de identificación de radioisótopos dentro del espectro Gamma*" (17458.23-P) auspiciado por el Tecnológico Nacional de México y el proyecto de Conahcyt CB-A1-43858 (MR).

## Referencias.

[1] U. Yoshinori et al, Surveillance of radioactive cesium in domestic foods on the Japanese market, J Food Soc Jpn, 56(2):49-56, 2015.
[2] D. Rangaswamvi et al, Estimation of radiological dose from radon, thoron and their progeny levels in the dwellings of Shivamogga district, Proc IARP, 2018.
[3] S. Miyuki et al, New method for visualizing the dose rate distribution around the Fukushima Daiichi nuclear power plant using ANNs, Sci Rep 11, 1857, 2021.
[4] S. Huseyin, Gamma Spectroscopy by Artificial Neural Network Coupled with MCNP, PhD thesis, 2017.
[5] C. Lattner, "LLVM: An Infrastructure for Multi-Stage Optimization," M.S. thesis, Dept. of Computer Science, Univ. of Illinois, USA, 2002.
[6] C. Lattner, "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation," Ph.D. thesis, Dept. of Computer Science, Univ. of Illinois, USA, 2005.
[7] Apple, Mixing Swift and C++, Swift.org Blog, Junio 2023, disponible en: https://www.swift.org/documentation/cxx-interop/
[8] B. Jacob, Eigen C++ template library, sitio oficial, recuperado: Julio 2024, disponible en: http://eigen.tuxfamily.org
[9] T. Burr & M. Hamada, Radio-Isotope Identification Algorithms for NaI γ Spectra, Algorithms, 2(1): 339-360, 2009.
[10] B. Tom & H. Michael, Radio-isotope identification algorithms for NaI g spectra, Algorithms, 339-360, 2009.
[11] M. Alamaniotis et al, Fuzzy-Logic Radioisotope Identifier for Gamma Spectroscopy in Source Search, IEEE Trans. Nucl. Sci, 60(4): 3014-3024, 2013.
[12] B. David et al, Principal component analysis of gamma-ray spectra for radiation portal monitors, IEEE Trans. Nucl. Sci., 59(1):154-160, 2012.
[13] S. Galib et al, A comparative study of machine learning methods for automated identification of radioisotopes using NaI gamma-ray spectra, Nucl. Eng. Tech., 53: 4072-4079, 2021.
[14] S. Huseyin, Gamma Spectroscopy by Artificial Neural Network Coupled with MCNP, PhD thesis, 2017.
[15] A. Miltiadis et al, Application of Fireworks Algorithm in Gamma-Ray Spectrum Fitting for Radioisotope Identification, Int. J. Swarm Intel. Res. (IJSIR), 6(6):102-125, 2015.
[16] A. Miltiadis et al, Fuzzy-logic radioisotope identifier for gamma spectroscopy in source search, IEEE Trans. Nucl. Sci., 60(4):3014-3024, 2013.
[17] R. Runkle et al, Analysis of spectroscopic radiation portal monitor data using principal components analysis, IEEE Trans. Nucl. Sci., 53(3):1418-1423, 2016.
[18] Apple, A Vision for Embedded Swift, Swift GitHub repository, Junio 2024, disponible en: http://github.com/swiftlang/swift-evolution/blob/main/visions/embedded-swift.md
[19] K. Mracek, Get Started with Embedded Swift on ARM and RISC-V Microcontrollers, Swift Blog, Abril 2024, disponible en: https://www.swift.org/blog/embedded-swift-examples