Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Desarrollo de aplicaciones basadas en aprendizaje automático en sistemas empotrados STM32

Autor: Juan Manuel Cuerva Gutiérrez

Tutor: Antonio Luque Estepa

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Desarrollo de aplicaciones basadas en aprendizaje automático en sistemas empotrados STM32

Autor:
Juan Manuel Cuerva Gutiérrez

Tutor:
Antonio Luque Estepa
Profesor Titular

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado: Desarrollo de aplicaciones basadas en aprendizaje automático en sistemas empotrados STM32

Autor: Juan Manuel Cuerva Gutiérrez
Tutor: Antonio Luque Estepa

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor, el Doctor Antonio Luque Estepa, por aportar su experiencia, su visión y sus inestimables consejos durante todo el proceso de elaboración de este proyecto. Ojalá más profesores como usted.

A mis padres, por darme la oportunidad de estudiar esta carrera en el ámbito de la Robótica, que tanto me apasiona desde pequeño.

A mi novia, Elena, por su incombustible apoyo en los momentos más duros a lo largo de estos años.

Y, por último, agradecer a la World Wide Web y a todos aquellos que brindan su ayuda y comparten su conocimiento de forma de libre.

Juan Manuel Cuerva Gutiérrez
Sevilla, 2021

Índice Abreviado

Índice

1 Introducción Y Objetivos

El desarrollo del hombre depende fundamentalmente de la invención. Es el producto más importante de su cerebro creativo. Su objetivo final es el dominio completo de la mente sobre el mundo material y el aprovechamiento de las fuerzas de la naturaleza a favor de las necesidades humanas

- Nikola Tesla -

Este proyecto trata dos tecnologías con un pasado distinto, los sistemas embebidos y el aprendizaje automático, pero que, juntas, suponen el siguiente paso en la evolución de la Inteligencia Artificial.

1.1 Introducción

El futuro del aprendizaje automático es pequeño. Actualmente, existen más de 250 billones de microcontroladores en el mundo [17] y sus usos y aplicaciones son casi ilimitadas; vehículos, equipos industriales y comerciales, electrodomésticos, electrónica de consumo... Según IC Insights [11], aunque en el año 2020 las ventas mundiales de MCU cayeron un 8 % hasta los 14,9 mil billones de dólares debido al Covid-19, se pronostica que para año 2023 los ingresos de MCU alcanzarán un nivel récord de 18.8 billones de dolares. Además, la encuesta de la consultoría *Silent Intelligence* refuerza el pronóstico anterior y es que el *TinyML*, término en inglés que se refiere a la integración de técnicas de aprendizaje automático en pequeños dispositivos basados en microcontroladores de apenas unos kilobytes de memoria, puede alcanzar más de 70 mil millones de dólares en valor económico en los próximos cinco años [31].

Estas cifras no pasan desapercibidas y por ello el *TinyML* es un área que está experimentando un gran desarrollo y ganando cada vez más importancia. Tanto es así, que la universidad de Harvard lanzó en agosto del año pasado un certificado profesional en esta temática [32].

1.1.1 Evolución del TinyML: De la nube a los microcontroladores

En los inicios de la Inteligencia Artificial, los modelos de machine learning eran entrenados y alojados en la nube haciendo uso de CPUs y GPUs de alta gama para entrenar los modelos y realizar allí las inferencias debido a la enorme capacidad de cómputo necesario, por lo que cada aplicación necesitaba estar conectada a la nube (Figura 1.2). Los microcontroladores en estos momentos estaban relegados a la gestión de los sensores y actuadores.

Sin embargo, esto introducía un gran problema: la latencia. El retraso involucrado en la petición de inferencia y su respuesta era inasumible para una gran cantidad de aplicaciones, principalmente pertencecientes al sector industrial, médico y vehículos inteligentes. De esta manera, surgió el concepto de *Edge Computing*, un paradigma que acerca los servicios de computación a los dispositivos

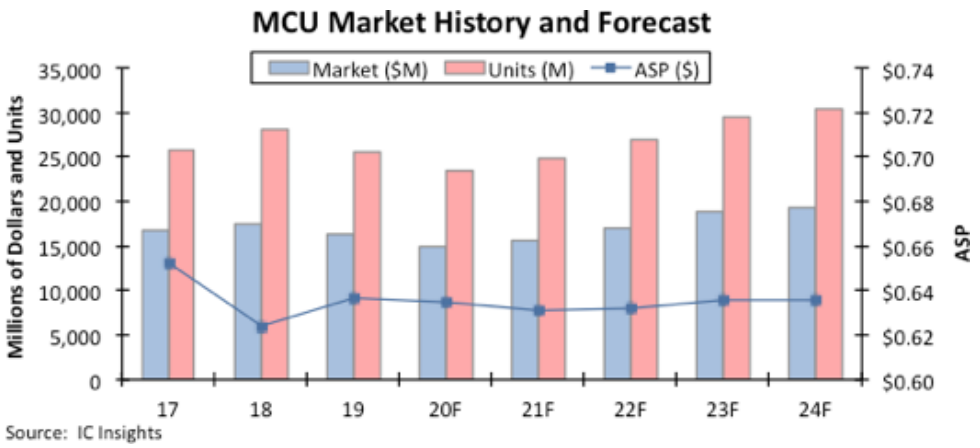

Figura 1.1 Historial y pronóstico del mercado de microcontrolares. Fuente: IC Insights.

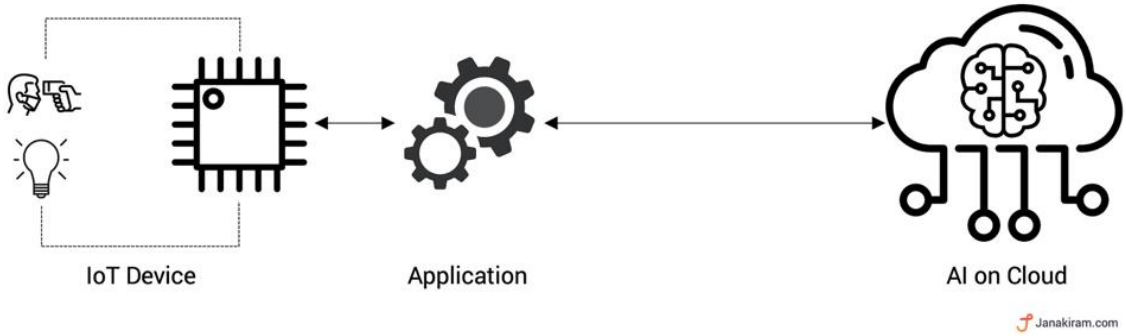

Figura 1.2 Paradigma de Inteligencia Artificial en la nube. Fuente: Janakiram MSV (extraída de Forbes).

finales y sensores generadores de datos, mejorando los tiempos de respuesta y ahorrando ancho de banda [8] (Figura 1.3). No obstante, aunque los modelos entrenados podían ser alojados *at the edge*, el entrenamiento debía realizarse en la nube, debido a la limitación de recursos de estos dispositivos.

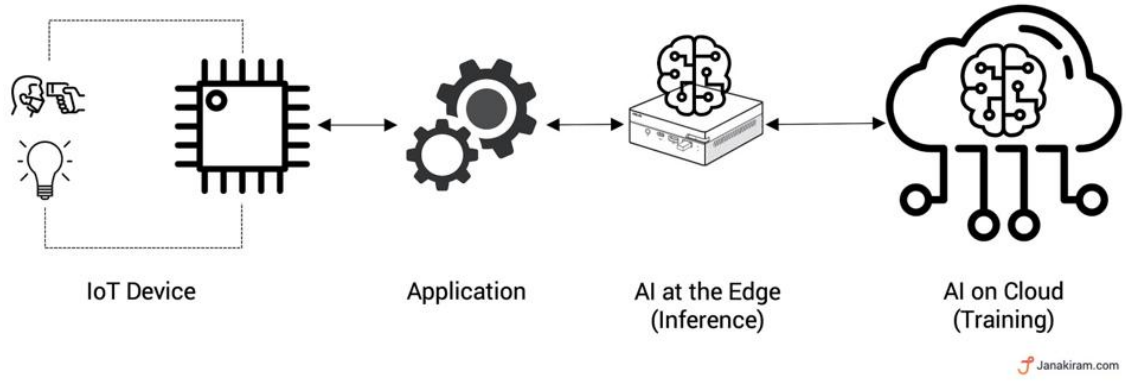

Figura 1.3 Paradigma de Inteligencia Artificial *at the Edge*. Fuente: Janakiram MSV (extraída de Forbes).

Si bien ejecutar inteligencia artificial *at the edge* puede ser una buena solución en gran cantidad de casos, siguen existiendo escenarios donde implementar este paradigma resulta inviable: por ejemplo, en la electrónica de consumo, la implementación de este modelo lleva a un aumento del costo total de los dispositivos así como de los costes de soporte para los proveedores. Otro ejemplo

son los entornos industriales, donde incorporar modelos de aprendizaje automático para detectar anomalías en tiempo real y proporcionar mantenimiento predictivo, puede llegar a ahorrar millones de euros en costes de mantenimiento. Por ello, aplicar inteligencia artificial sobre microcontrolares resulta clave para ciertos escenarios.

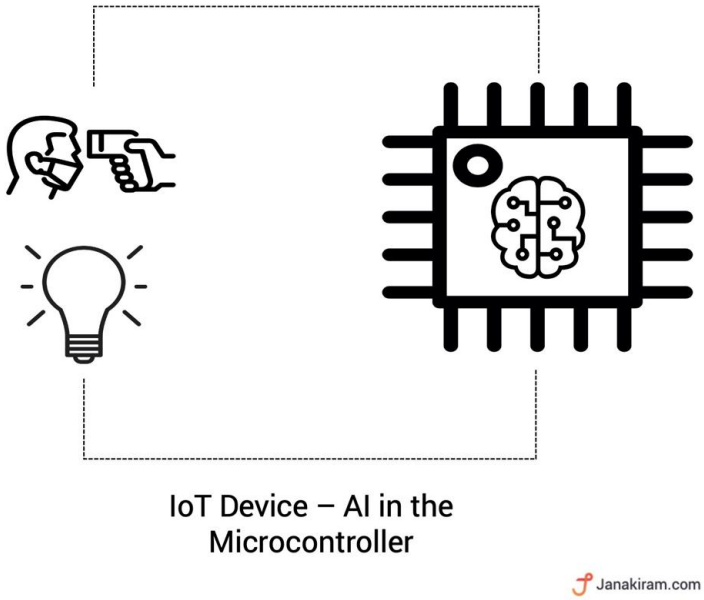

Figura 1.4 Paradigma de Inteligencia Artificial sobre el propio microcontrolador. Fuente: Janakiram MSV (extraída de Forbes).

1.1.2 Importancia del TinyML

La utilidad de aplicar técnicas de deep learning sobre microcontroladores es de gran relevancia por varios motivos. En primer lugar, los microcontroladores hacen uso de muy poca cantidad de energía (del orden de mW) comparados con los dispositivos que necesitan estar conectados permanentemente a la red o cambios frecuentes de baterías con lo que, un microcontrolador, podría ejecutar aplicaciones de inteligencia artificial durante meses con una sola pila de botón. En segundo lugar, las aplicaciones son independientes de paradigmas como *edge computing* o cloud, haciendo posible el uso de inteligencia artificial en cualquier parte del mundo, sin necesidad de acceso a internet, lo que abre caminos inexplorados a aplicaciones hasta ahora inalcanzables. Por ende al no tener la necesidad de enviar datos a la nube, los cuales pueden contener información privada de carácter personal, los usuarios no deben preocuparse por la seguridad de los mismos, ya que éstos nunca abandonan el dispositivo. A todo ello se debe sumar la reducción de costes que supone el uso de microcontrolares (desde unos pocos euros) frente al uso de una estación de trabajo de aprendizaje automático de alto rendimiento, que puede alcanzar cotas del orden de miles de euros.

Sin embargo, son sus principales ventajas las que hacen del TinyML todo un reto para la ingeniería. Por ejemplo, la memoria de un microcontrolador resulta 2 o 3 órdenes de magnitud menor que la de un teléfono móvil, manejándose en el rango unos pocos de kilobytes. Por ello, resulta de vital importancia desarrollar algoritmos que hagan un uso más eficiente del espacio, ejecutando menos instrucciones y operaciones y permitiendo un mejor aprovechamiento de la energía.

Con este trasfondo, resultará de interés el estudio realizado en este trabajo, en el que se presenta el desarrollo de aplicaciones basadas en aprendizaje automático sobre sistemas empotrados y en concreto sobre un microcontrolador de la familia STM32, el cual se catalogaría como un equipo

de bajas prestaciones en comparación a los usados habitualmente con esta tecnología. Además, se engloba también el desarrollo de los drivers para los sensores que se usarán en este proyecto.

1.2 Objetivos

En este proyecto se pretende, partiendo de herramientas ya existentes, realizar el desarrollo de un framework que permita aplicar técnicas de aprendizaje automático en un sistema embebido de capacidades limitas así como crear un driver propio para la interacción de la placa de desarrollo con periféricos que habiliten la posibilidad de capturar datos del entorno. Además se estudiarán las dificultades que ésto presenta.

Como punto de partida, se utilizará la placa de desarrollo *Núcleo-64 STM32F401RE* [15] y el módulo ambiental *X-NUCLEO-IKS01A1* [33] y se creará un driver para poder hacer uso del acelerómetro integrado. Una vez se puedan obtener medidas, se empleará una red neuronal pre-entrenada que, mediante la herramienta *STM32CubeMX* [19], se transformará para poder ser interpretada por la placa de desarrollo y, con *STM32CubeIDE* [18], se creará un framkework que permita al microcontrolador realizar inferencias así como hacer uso de una o varias redes neuronales al mismo tiempo. La aplicación de ejemplo utilizará una red neuronal proporcionada por *STMicroelectronic* que permite la clasificación de actividad humana. Para concluir, se modificará dicha red para reconocer un movimiento adicional y se presentarán las posibles líneas futuras de desarrollo del proyecto.

2 Desarrollos existentes

El placer más noble es el júbilo de comprender

- Leonardo da Vinci -

En este capítulo se mostrará el panorama actual de las tecnologías existentes para la implementación de aplicaciones de inteligencia artificial en dispositivos embebidos de bajo coste y se presentarán las herramientas elegidas para el desarrollo de este proyecto.

2.1 TensorFlow

TensorFlow es la apuesta clave de Google para crear un ecosistema de Machine Learning end-to-end que ayude a potenciar la revolución del Deep Learning. Según su propia página, TensorFlow [5] es una plataforma de código abierto de extremo a extremo para el aprendizaje automático que cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos que permite tanto a investigadores como desarrolladores compilar e implementar con facilidad aplicaciones basadas en redes neuronales.

TensorFlow cuenta, a parte de su biblioteca principal, con *TensorFlow.js*, una biblioteca de JavaScript para entrenar e implementar modelos en el navegador y en Node.js, *TensorFlow Lite* para implementar modelos en dispositivos integrados y móviles y *TensorFlow Extended*, una plataforma de extremo a extremo para la preparación de datos, el entrenamiento, la validación e implementación de modelos en entornos de gran producción.

Figura 2.1 Logo de TensorFlow.

Una de las virtudes de Tensorflow es su arquitectura simple, que permite la capacidad de compilar y entrenar modelos de apredizaje automático de forma fácil haciéndo uso de APIs intuitivas y de

alto nivel como Keras [12], con ejecución inmediata, rápido prototipado e iteración de modelos inmediata y con depuración sencilla. Además, cabe destacar su flexibilidad, ya que permite entrenar e implementar con facilidad modelos en la nube, locales, en el navegador o en el propio dispositivo empleando lenguajes como Python, JavaScript o Swift.

En sus tres años de historia ha acumulando más de 41 millones de descargas y es el proyecto con mayor número de contribuciones en Github año tras año, con más de 1.800 contribuciones [16] y con decenas de ejemplos de uso en distintas plataformas.

2.2 Edge Impulse

Edge Impulse [10] es una plataforma de desarrollo que permite la aplicación de machine learning en sistemas embebidos. Fundada en 2019, ha conseguido posicionarse como una de las líderes del sector en la actualidad, con más de 5,4 millones de dólares en financiación [6] y usada por más de 1000 empresas a través de más de 11000 proyectos de ML (Machine Learning) en todo el mundo. Se especializa en aplicaciones industriales y profesionales que incluyen mantenimiento predictivo, seguimiento y monitoreo de activos y detección de actividades en humanos y animales.

La gran ventaja de Edge Impulse es que proporciona herramientas para dar soporte al usuario en todo el proceso de desarrollo de ML (Figura 2.2), acelerando el tiempo de desarrollo de aplicaciones de años a semanas, habilitando modelos altamente optimizados que se pueden implementar en una amplia gama de hardware, desde MCUs hasta CPUs.

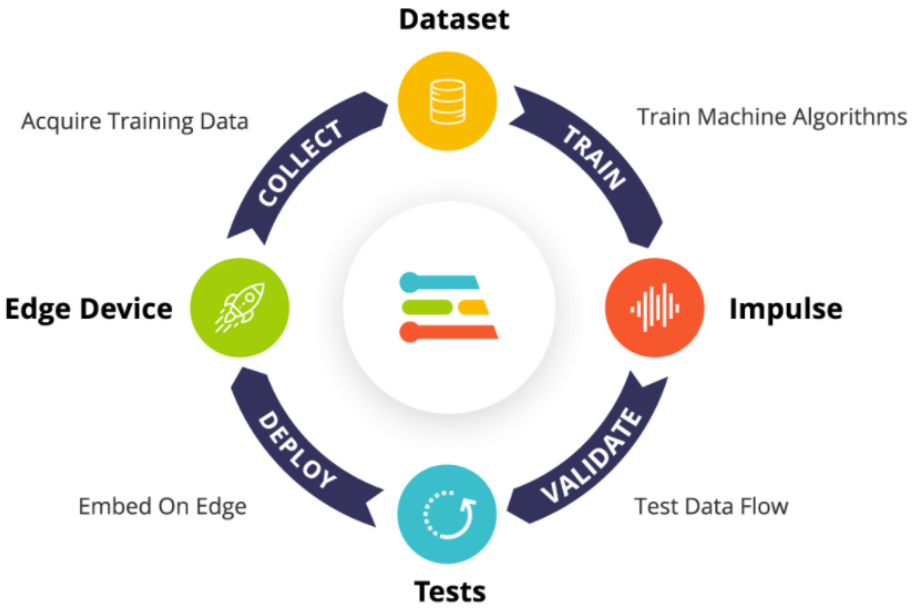

Figura 2.2 Ciclo de trabajo en Edge Impulse.

Para ello, gran parte de sus herramientas se centran en la manipulación de los conjuntos de datos: adquisición de datos (Figura 2.3), visualización (Figura 2.4), análisis (Figura 2.5), procesamiento y filtrado y creación de *datasets* entre otros. También posee una biblioteca de algoritmos extensibles y personalizables y un proceso de aplicación de datos a dichos algoritmos de forma automatizada. Esto incluye una potente infraestructura para la creación y entrenamiento de redes neuronales (Figura 2.6) que permite al usuario tener el control de cada parte del proceso y visualizar que está

sucediendo en cada momento. Su implementación se centra principalmente en microcontroladores y CPUs de gama baja.

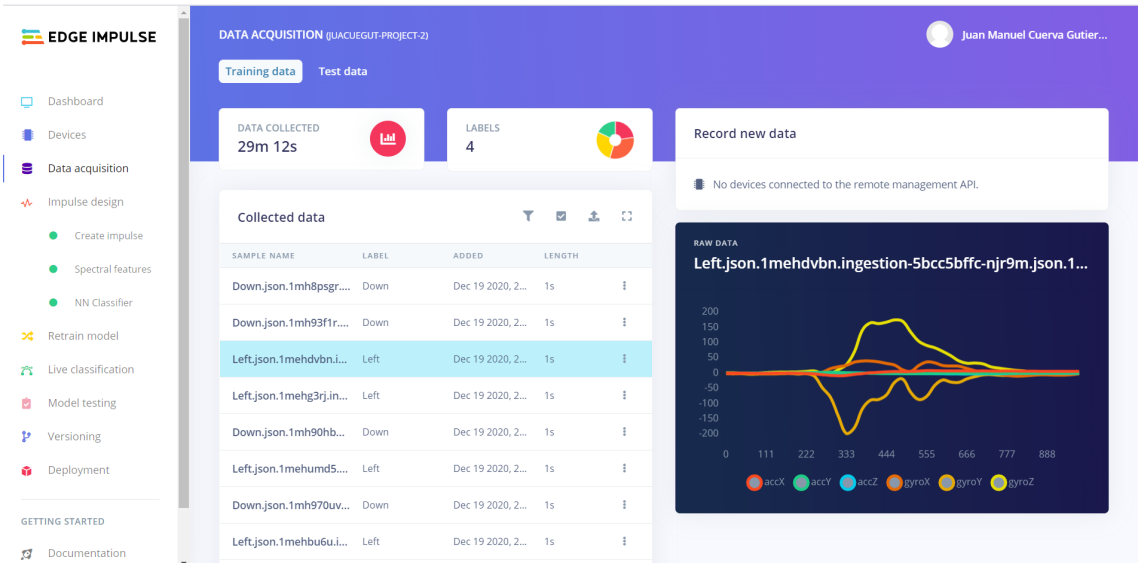

Figura 2.3 Adquisición de datos con Edge Impulse.

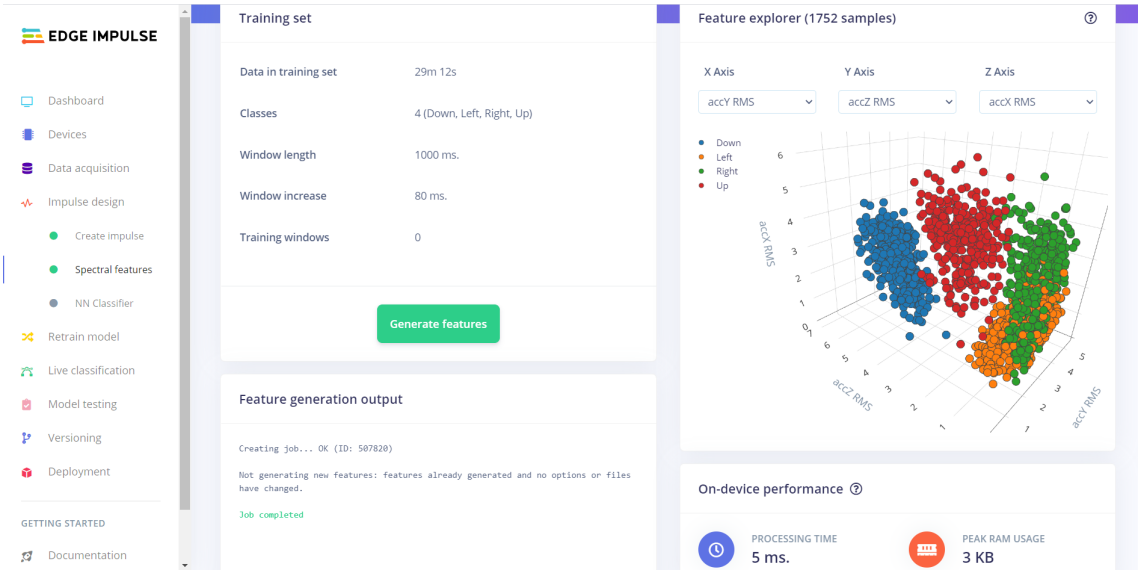

Figura 2.4 Visualización de los conjutos de datos con Edge Impulse.

Con Edge Impulse, se puede recopilar datos, crear una red, entrenarla y realizar el despligue de la aplicación en cualquier dispositivo. Para ello, permite crear librerías en C++ sin dependencias externas, librerías para Arduino con ejemplos que pueden ejecutarse en la gran mayoría de placas basadas en Arm, crear un CMSIS-PACK como una librería de C++, el cual se puede instalar en la mayoría de MCUs de ST haciendo uso de STM32CUBE.AI y crear un paquete de *WebAssembly*, permitiendo la creación directa de firmware para una gran variedad de placas de desarrollo.

Edge Impulse opera con un modelo de negocio típico de "Software como Servicio" (SaaS). Posee un nivel gratuito para ingenieros individuales que permite realizar todo el flujo de desarrollo (desde la captura y procesamiento de datos hasta el entrenamiento y salida a producción) y ofrece servicios *premium* a empresas mediante una suscripción, que incluye funciones para la colaboración entre

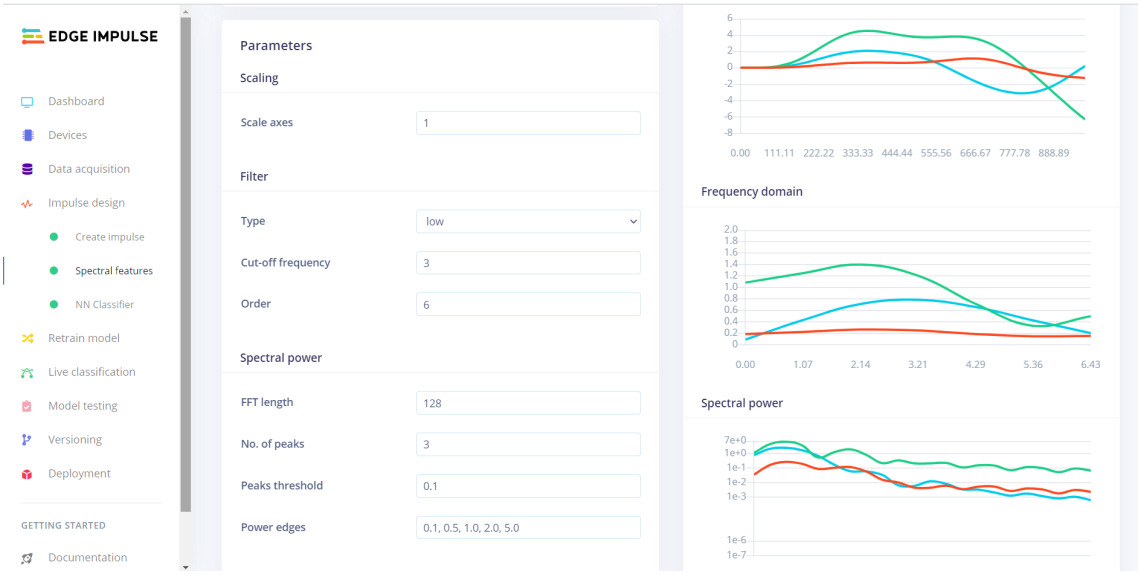

Figura 2.5 Análisis de los datos con Edge Impulse.

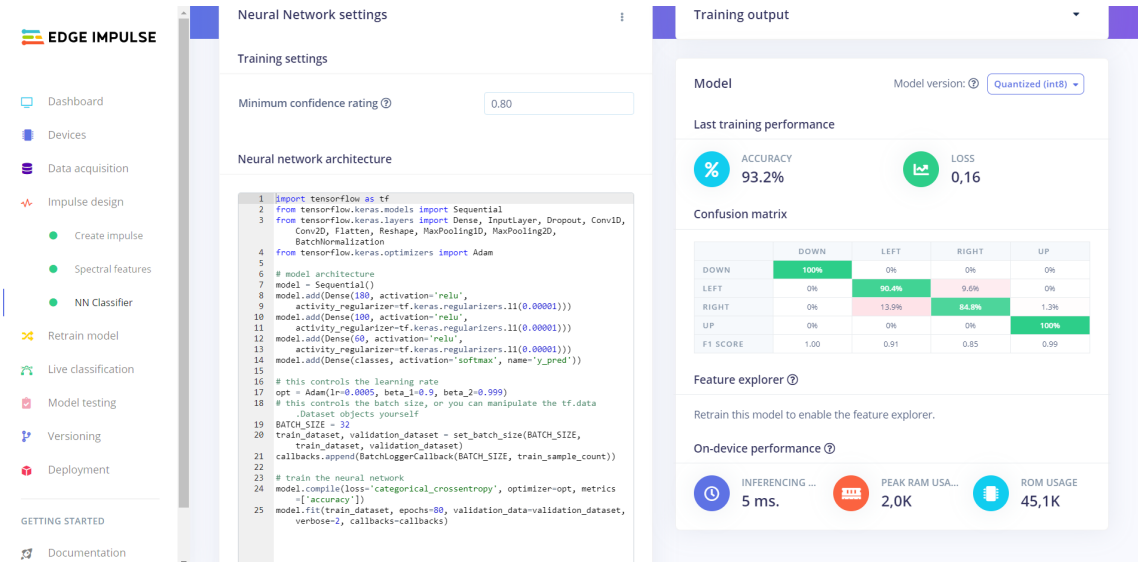

Figura 2.6 Creación de una red neuronal con Edge Impulse.

equipos de ingenieros, trabajar con conjuntos de datos más grandes, control de versiones y mayores opciones de seguridad.

2.3 MCUNet

Cabe destacar en este apartado MCUNet, un sistema capaz de diseñar redes neuronales compactas que se adaptan a las necesidades de los dispositivos integrados a la red IoT, ofreciendo máxima velocidad y precisión a pesar de las limitaciones en cuanto a memoria o potencia de procesamiento que suelen presentar estos dispositivos. Este sistema fue diseñado por Ji Lin como autor principal y Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan y Song Han como co-autores, del Instituto Tecnológico de Massachusett, la Universidad Nacional de Taiwan y el laboratorio de AI MIT-IBM Watson [13] y fue presentado en la "Conferencia Anual sobre Sistemas de Procesamiento de Información Neuronal" en diciembre de 2020.

MCNet consiste en el uso combinado de dos tecnologías desarrolladas por el MIT; TinyNAS y TinyEngine (Figura 2.8). TinyNAS es un algoritmo de búsqueda de arquitectura neuronal capaz de manejar diversas restricciones automáticamente (tipo de dispositivo, latencia, energía y memoria) y crear redes de tamaño personalizado para cada tipo de microprocesador (Figura 2.7). TinyEngine sirve como motor de inferencia encargado de dirigir la gestión de los recursos a través de un código esencial, que incorpora los requisitos básicos necesarios para su funcionamiento, adaptando la programación de la memoria de acuerdo con la topología general de la red en lugar de la optimización por capas, lo que reduce el tiempo de compilación, reduce el uso de memoria en 3.4x y acelera la inferencia en 1.7-3.3x en comparación con el uso de TF-Lite Micro y CMSIS-NN (Figura 2.9).

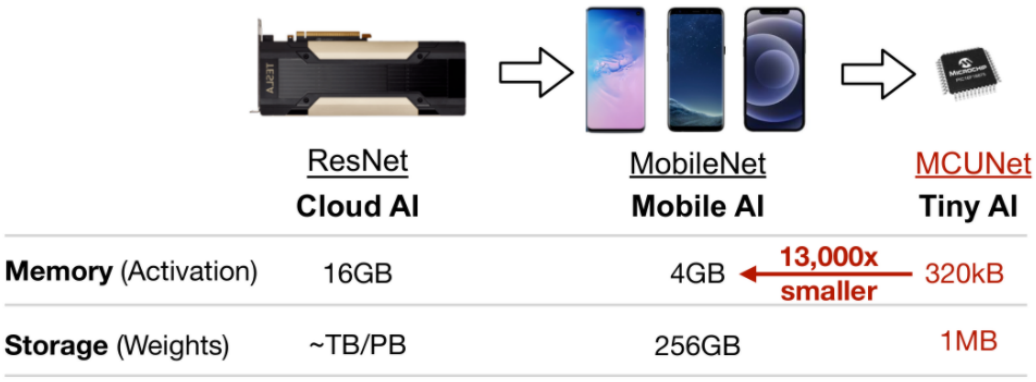

Figura 2.7 Comparación del uso de la memoria con MCUNet. Imagen extraida de [13].

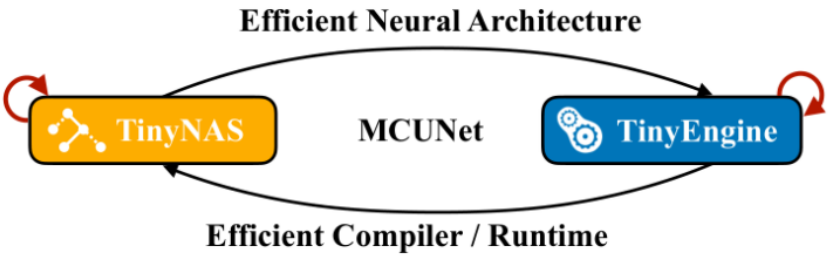

Figura 2.8 Sistema de trabajo del algortimo de MCUNet. Imagen extraida de [13].

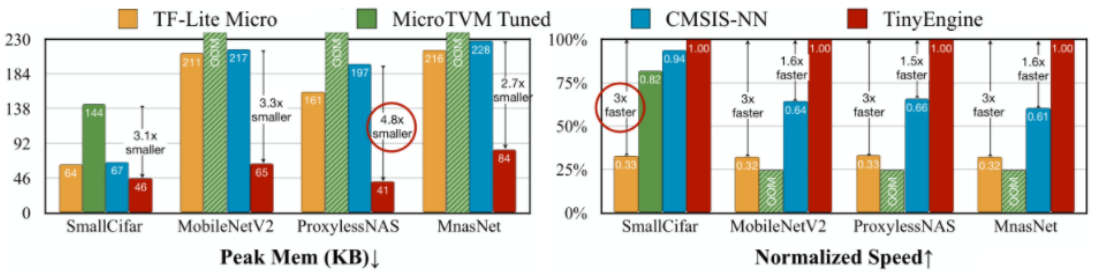

Figura 2.9 Comparación del uso de la memoria y el tiempo de inferencia entre otros frameworks actuales y MCUNet. Imagen extraida de [13].

2.4 Herramientas de STMicroelectonics

De forma cada vez más frecuente, los fabricantes de microcontroladores ofrecen como parte de su software un IDE que permite, de forma sencilla, programar y hacer debug de sus microcontroladores. En concreto, la empresa *STMicroelectronics* proporciona un ecosistema de desarrollo conocido como *STM32Cube ecosystem*, el cual ofrece una solución software completa para microcontroladores y microprocesadores de STM32.

STM32Cube es una combinación de herramientas software y librerías integradas que cubren todas las necesidades de un ciclo completo de desarrollo de proyectos.

Como se comentará más adelante en la sección 2.6, al hacer uso de una placa de ST, éste será el ecosistema de desarrollo que se utilizará en este proyecto.

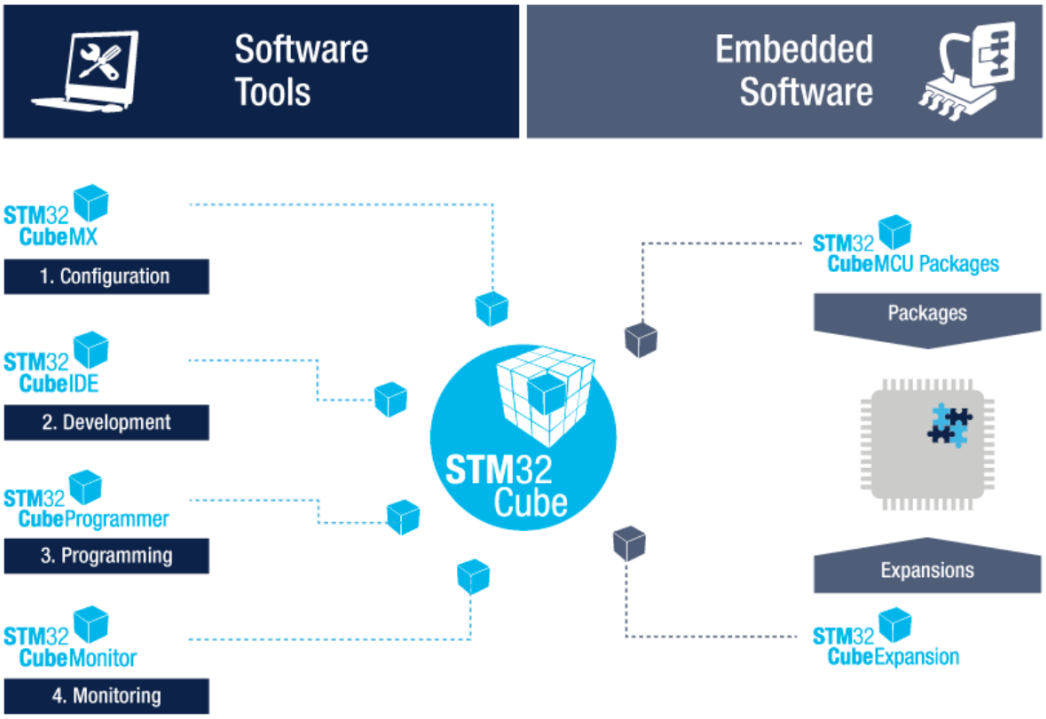

Figura 2.10 Ecosistema de desarrollo de software STM32Cube.

Principalmente, se van a usar dos herramientas de este ecosistema: el programa de configuración *STM32CubeMX* y el entorno de desarrollo *STM32CubeIDE*.

2.4.1 STM32CubeMX

STM32CubeMX es una herramienta gráfica que permite una fácil configuración de los microcontroladores y microprocesadores de STM32, así como la generación del código C necesario para la inicialización de éstos en los núcleos Arm® Cortex®-M y para la generación del código fuente para árboles de dispositivos en Linux en los núcleos Cortex-A, a través de procesos paso a paso. Algunas de las características que hacen de *STM32CubeMX* un software gran utilidad para trabajar con microcontroladores de *STMicroelectronics* son las siguientes:

- Fácil selección de microcontroladores, cubriendo toda la cartera de STM32.
- Selección de placas de una lista de placas de STMicroelectronics.

- Configuración sencilla de microntroladores (pines, reloj, periféricos, middleware) y generación de su correspondiente código de inicialización en lenguaje C.
- Generación de informes de configuración
- Generación de proyectos listos para importar a un IDE para una selección de de entornos de desarrollos integrados. Todos los proyectos de STM32CubeMX incluyen código C de inicialización, drivers HAL para STM32, middleware para la configuración del usuario y todos los archivos necesarios para abrir y construir el proyecto en el IDE seleccionado.
- Cálculo del consumo de energía para aplicaciones definidas por el usuario.
- Auto-actualizaciones que permiten al usuario mantener actualizado el STM32CubeMX.

El ciclo de configuración dentro de *STM32CubeMX* se puede observar en la Figura 2.12.

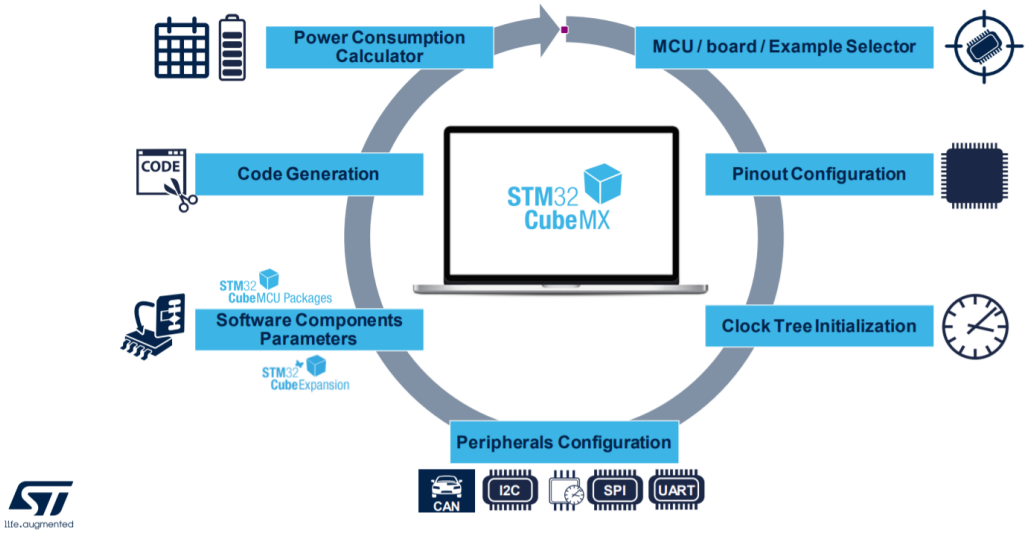

Figura 2.11 Ciclo de uso de la herramienta STM32CubeMX.

2.4.2 STM32CubeIDE

STM32CubeIDE es una avanzada plataforma de desarrollo para C/C++ que incluye configuración de periféricos, generación de código, compilación y debug para los microcontroladores y microprocesadores de la familia STM32. Está basado en el *framework* Eclipse®/CDT y GCC como toolchain, y GDB para realizar debugs. Por ello, permite la integración con los cientos de plugins existentes que completan las características del IDE de Eclipse®.

Una de las principales ventajas de STM32CubeIDE es que permite integrar la configuración de STM32 y las funcionalidades de creación de proyectos de STM32CubeMX, lo que ofrece una experiencia de herramienta todo en uno y permite ahorrar tiempo de instalación y desarrollo.

2.5 Red neuronal de ST

En este apartado se presentará la red neuronal de ST que se va a utilizar como base en este proyecto. Mas adelante, en la sección 4.1.2, se comentarán los cambios realizados a dicha red.

Esta red puede ser descargada desde el *Github de STMicroelectronics* [1], directamente en formato "*.h5*" lista para su implementación con *STM32CubeMX*. Para realizar modificaciones en dicha red y ver su estructura, se debe acceder a [20] donde se hace uso de la herramienta de Google Research

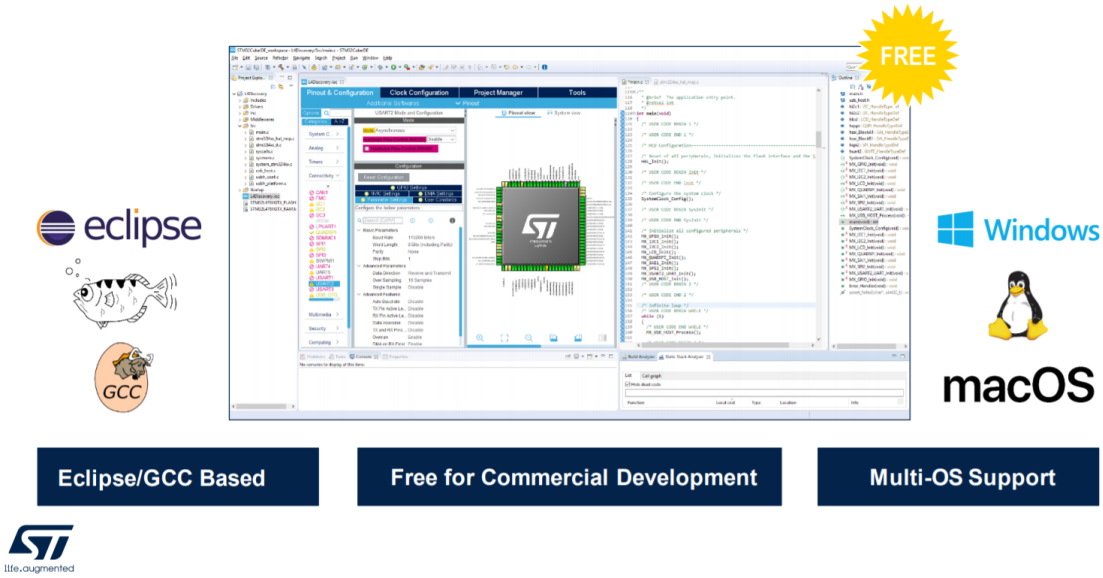

Figura 2.12 Plataforma de desarrollo STM32CubeIDE.

Colaboratory [2], un servicio cloud basado en notebooks de Jupyter, que permite el uso gratuito de las GPUs y TPUs de Google, no requiere configuración y permite compartir contenido fácilmente.

La red distingue entre tres movimientos; estacionario, caminando y corriendo. Para ello, hace uso de un dataset propio, el cual también se puede encontrar en el *Github de STMicroelectronics* [1]. La estructura de este dataset será tratada en detalle en el apartado 4.1.2. En la Figura 2.13 se pueden ver algunas gráficas de estos movimientos.

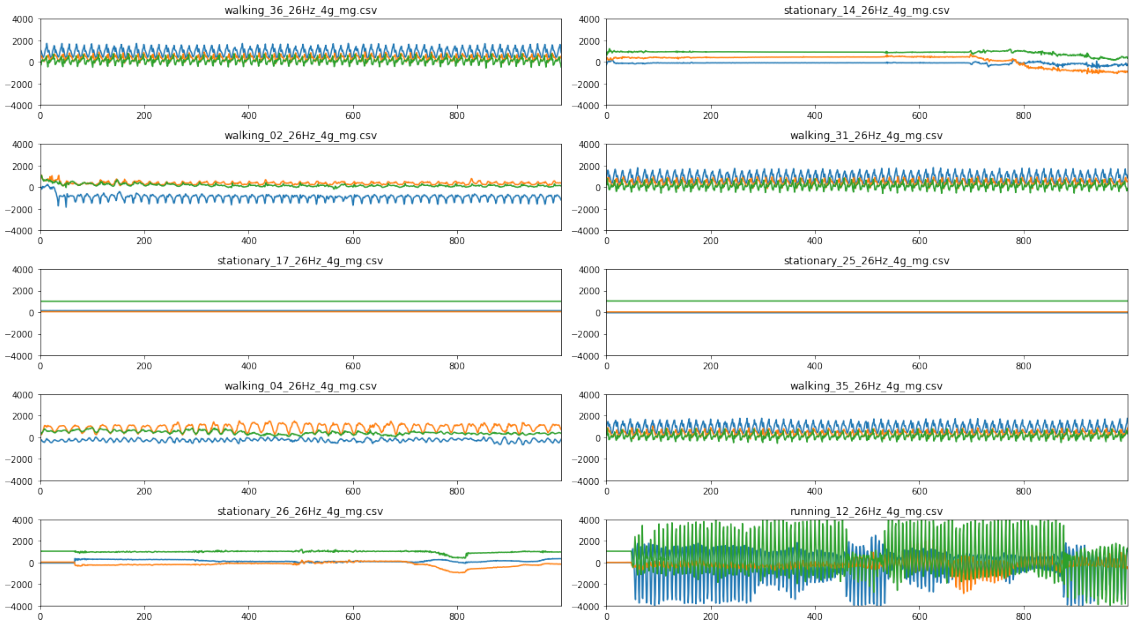

Figura 2.13 Gráficas de los movimientos contenidos en el dataset de la red neuronal de ST.

La red está construida sobre un modelo secuencial de keras (*tf.keras.models.Sequential*) donde se apilan seis capas, cada una de ellas con un tensor de entrada y un tensor de salida. En concreto, su estructura es la siguiente:

Código 2.1 Fragmento del archivo *main.c*.

```
## Conv1D based model
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=16, kernel_size=3, activation='relu',
      input_shape=(26, 3)),
  tf.keras.layers.Conv1D(filters=8, kernel_size=3, activation='relu'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(3, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=30)
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

print("Test loss:", test_loss)
print("Test acc:", test_acc)
model.summary()
```

Las dos primeras capas son de tipo *Conv1D*, es decir, crean un núcleo de convolución (*kernel*) que convolucionan con la entrada de la capa para producir un tensor de salida. Como función de activación utilizan una ReLu, función de tipo no lineal de las más utilizadas debido a que permite un aprendizaje muy rápido en las redes neuronales. Después existe una capa de *Dropout*, una técnica de regularización que tiene como objetivo reducir el sobreajuste (*overfitting*) omitiéndo aleatoriamente neuronas, tanto ocultas como visibles, durante el proceso de entrenamiento de la red. A continuación, la siguiente capa (*Flatten*) convierte los datos multidimensionales en un vector de características 1D. Dicho vector es la entrada de la siguiente capa, de tipo *Dense*, en la que cada neurona recibe información de todas las neuronas de su capa anterior. Por último, la capa de salida, también de tipo *Dense*, tiene 3 neuronas (tres posibles tipos de salida) y tiene como función de activación *Softmax*, la cual asigna probabilidades decimales a cada clase hasta sumar 1.0.

Cabe destacar que esta red hace uso de TensorFlow 1.15 ya que TensorFlow 2.2 no es soportado aún por X-CUBE-AI.

2.6 Hardware utilizado

El hardware del cual se ha hecho uso en este proyecto ha sido facilitado por el Dr. Antonio Luque Estepa y consite en una placa de desarrollo *Núcleo-64 STM32F401RE* y una tarjeta de expansión MEMS con sensores ambientales y de movimiento *X-NUCLEO-IKS01A1*.

2.6.1 Placa de desarrollo: Núcleo-64 STM32F401RE

La placa de desarrollo Núcleo-64 STM32F401RE proporciona una forma fácil y asequible de desarrollar nuevos conceptos y construir prototipos aprovechando las características de rendimiento

y consumo de energía proporcionadas por el microcontrolador STM32. Además, soporta la conectividad con ARDUINO® Uno V3, lo que le brinda acceso a multitud de módulos de expansión compatibles. Imágenes de esta placa de desarrollo se pueden ver en la Figura 2.14 y en la 2.15.

Características del microcontrolador:

- Microcontrolador STM32 con package LQFP64.
- 1 led para el usuario compartido con ARDUINO®.
- 1 boton pulsador para el usuario y otro de reset.
- Oscilador de cristal de 32.768 kHz.
- Diferentes opciones de fuentes de alimentación: ST-LINK, USB VBUS, o fuentes externas.
- Programador / depurador ST-LINK integrado con capacidad de reenumeración USB: almacenamiento masivo, puerto COM virtual y puerto de depuración.
- Gran disponibilidad de bibliotecas y ejemplos de software gratuitos con el paquete STM32Cube MCU.
- Soporte para una amplia variedad de entornos de desarrollo integrados (IDE), incluido IAR.
- Integrado con Workbench®, MDK-ARM, and STM32CubeIDE

Características especificas de la placa:

- SMPS externo para generar suministro lógico Vcore.
- HSE de 24 MHz.
- Conector dedicado externo para experimentación SMPS, conector Micro-AB/Mini-AB-USB para la depuración por ST-LINKMIPI®.
- Compatible con Arm® Mbed Enabled ™.

2.6.2 Tarjeta de expansión MEMS: X-NUCLEO-IKS01A1

La placa *X-NUCLEO-IKS01A1* es una tarjeta de expansión MEMS con sensores ambientales y de movimiento, compatible con Arduino UNO R3 y que interactúa con los microcontroladores STM32 a través de I²C. Imágenes de esta tarjeta se pueden ver en la Figura 2.16 y en la 2.17. Sus características son las siguientes:

- LSM6DS0: Acelerómetro MEMS 3D (±2/±4/±8 g) + giróscopo 3D (±245/±500/±2000 dps).
- LIS3MDL: Magnetómetro MEMS 3D (±4/ ±8/ ±12/ 16 gauss).
- LPS25HB: Sensor de presión MEMS, con barómetro de salida digital absoluta de 260-1260 hPa.
- HTS221: Sensor digital de humedad relativa y temperatura.
- Socket DIL de 24 pines disponible para adaptadores MEMS adicionales y otros sensores (UV index)
- Librería de desarrollo de firmware y ejemplos para todos los sensores compatibles con el firmware de STM32Cube.
- Compatible con placas Núcleo STM32.
- Equipada con conector Arduino UNO R3.
- Certificado RoHS de restricción de ciertas sustancias peligrosas en aparatos eléctricos y electrónicos.

Figura 2.14 Placa de desarrollo Núcleo-64 STM32F401RE (parte de arriba).

2.7 Conexión entre la placa Núcleo-64 STM32F401RE y la tarjeta X-NUCLEO-IKS01A1

La conexión entre la placa de desarrollo y la tarjeta ambiental es sencilla, ya que únicamente se debe montar la tarjeta X-NUCLEO-IKS01A1 encima de la placa Núcleo-64 STM32F401RE (ver Figura 2.19).

Todos los sensores están conectados en un solo bus I²C con lo que se debe configurar y realizar la selección de la dirección del sensor que se quiere utilizar en cada momento.

Figura 2.15 Placa de desarrollo Núcleo-64 STM32F401RE (parte de abajo).

Figura 2.16 Tarjeta de expansión X-NUCLEO-IKS01A1 (parte de arriba).

Figura 2.17 Tarjeta de expansión X-NUCLEO-IKS01A1 (parte de abajo).

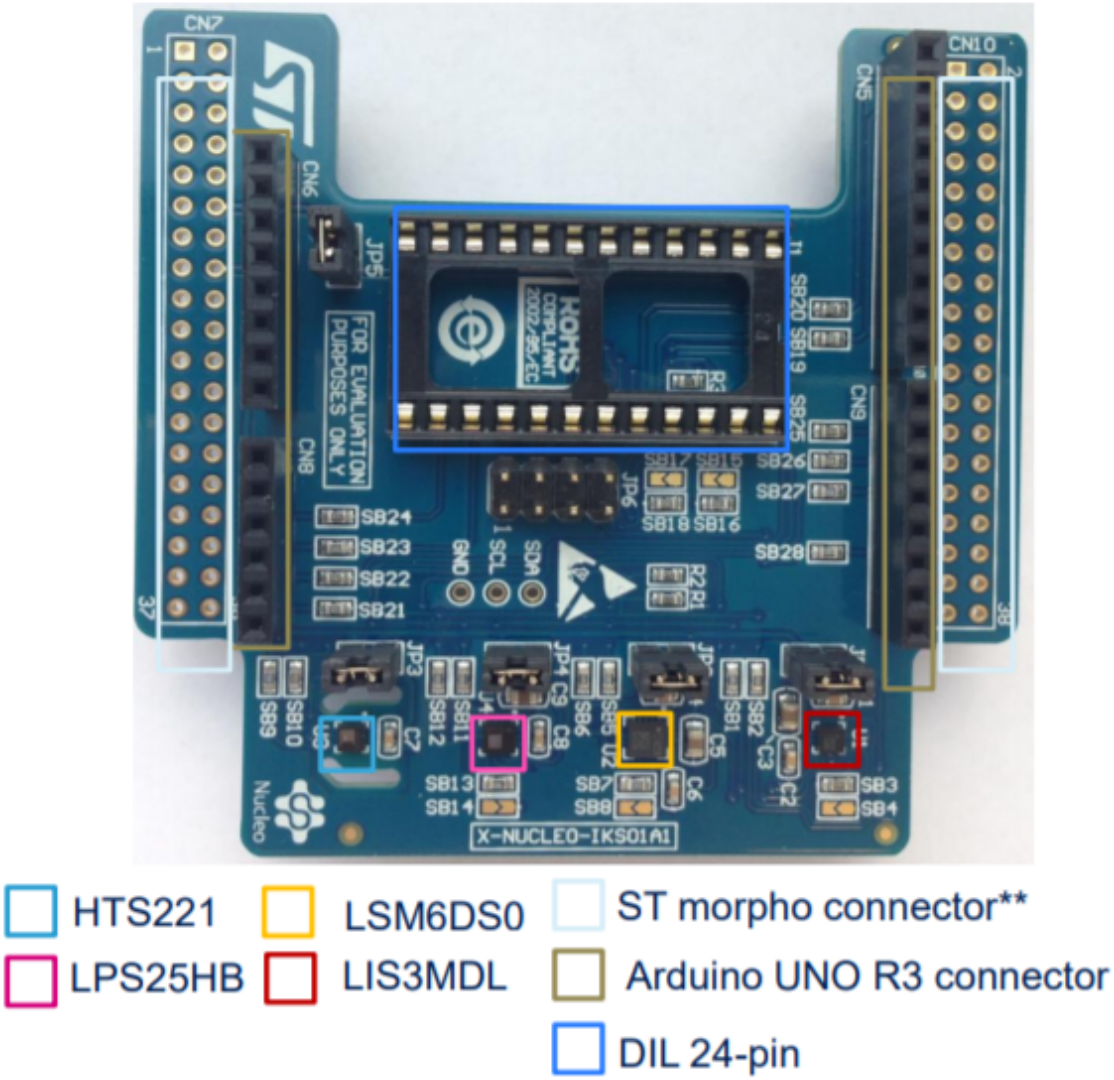

Figura 2.18 Posición de los sensores en la tarjeta de expansión X-NUCLEO-IKS01A1.

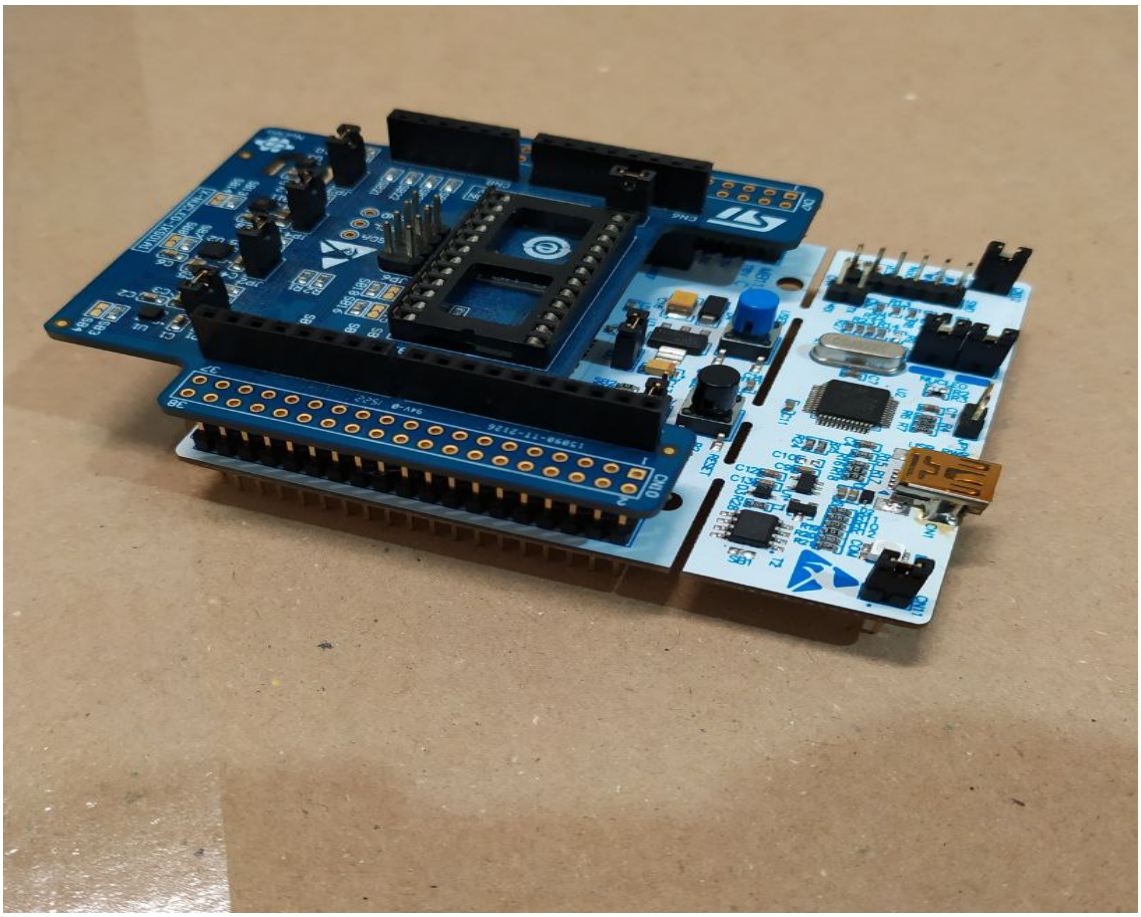

Figura 2.19 Conexión entre la placa Núcleo-64 STM32F401RE y la tarjeta X-NUCLEO-IKS01A1.

3 Diseño e implementación

Los científicos estudian el mundo tal como es; los ingenieros crean el mundo que nunca ha sido

- Theodore Von Karman -

En este capítulo se presentarán los diseños realizados, las implementaciones llevadas a cabo y se realizará un estudio de la problemática enfrentada como presentación de la necesidad de los desarrollos que se abarcan en este TFG.

3.1 Consideraciones previas

Para utilizar los diseños, primero se debe configurar correctamente el entorno. Ésto implica configurar el I^2C, la USART, instalar el paquete de expansión de Inteligencia Artificial *X-CUBE-AI* [25] y cargar una red-preentranada para generar los archivos sobre los cuales se apoya la programación. Si resulta de interés al lector, todo ello será detalladamente tratado en el anexo A.

3.2 Diseño del driver para el acelerómetro LSM6DS0

Si bien es cierto que STM32CubeMX ofrece un paquete de expansión que incluye drivers para reconocer sensores de temperatura, humedad, presión y movimiento (el paquete X-CUBE-MEMS1 [27]), éste solo ofrece drivers para las tarjetas X-NUCLEO-IKS01A2, X-NUCLEO-IKS01A3 y X-NUCLEO-IKS02A1, pero no para la tarjeta X-NUCLEO-IKS01A1. Si se busca dicha tarjeta en la página de ST [33], ésta aparece marcada como "NRND" (*Not Recommended for New Dessings*) y se indica explícitamente que se encuentra sin soporte.

Realizando una búsquedad dentro de la página de ST, se puede comprobar que la última versión del paquete X-CUBE-MEMS1 que soportaba la placa X-NUCLEO-IKS01A1 era la versión 4.4.1.

A la hora de trabajar en este TFG, la versión actual de STM32CubeMX es la 6.1.0, mientras que en el caso del paquete de expansión X-CUBE-MEMS1 la versión es la 8.2.0. Desde ST no se permite la instalación de software *legacy* por lo que, en STM32CubeMX, no se puede instalar ninguna versión anterior de ningún paquete de expansión ni se da acceso a descargas de versiones anteriores de dicho programa. Como consecuencia, hacer uso de los drivers propios de ST para la placa X-NUCLEO-IKS01A1 en las versiones actuales resulta imposible.

Como solución, se propuso la creación de proyectos para las tarjetas X-NUCLEO-IKS01A2, X-NUCLEO-IKS01A3 y X-NUCLEO-IKS02A1, con ligeras modificaciones, aunque no se obtuvieron resultados positivos ya que cada una de estas placas cuenta con un acelerómetro distinto[1], con

[1] LSM6DSL, LSM6DSO y LSM330DHCX, respectivamente

distintos registros y distintas funciones. Nótese aquí la importancia de distinguir entre los acelerómetros *LSM6DS0* (terminado en el número 0, el cual se utiliza en este proyecto) y el *LSM6DSO* (terminado en la letra O), pues poseen características diferentes y operan de forma distinta, lo que puede inducir a error de forma sencilla. De hecho, también se trató de utilizar de forma aislada el acelerómetro *LSM6DSO* (terminado en la letra O), el cual sí tiene soporte dentro de *STM32CubeMX*, tal y como se puede ver en la Figura 3.1. En este camino también se invirtió bastante tiempo y resultó igualmente infructuoso, pues se tratan de acelerómetros diferentes pero, en un principio, se pensó que este acelerómetro era el buscado y no otro distinto debido al gran parecido entre ambos nombres.

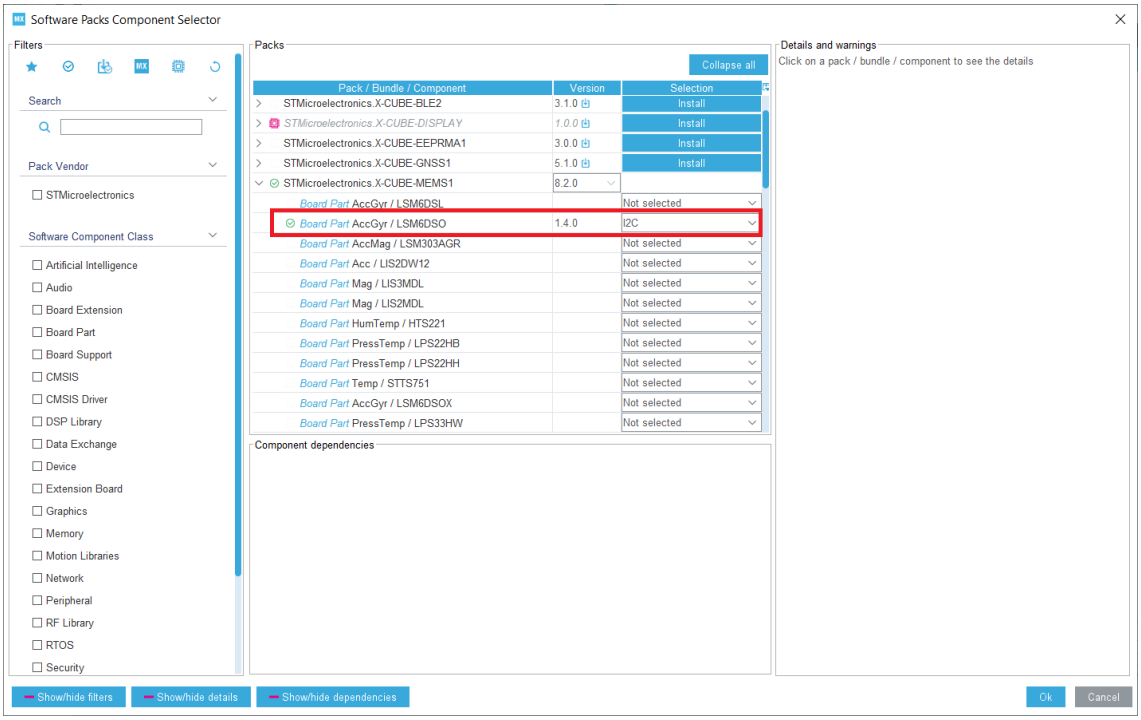

Figura 3.1 Selección del acelerómetro *LSM6DSO* (letra O), de forma aislada, tal y como aparece en *STM32CubeMX*.

Si bien es cierto, desde la propia página de ST se puede descargar el paquete X-CUBE-MEMS1-V4 [29], también NRND, específico para la tarjeta *X-NUCLEO-IKS01A1*. El paquete en cuestión contiene ejemplos de uso de dicha tarjeta pero, y aunque pueden importarse a *STM32CubeIDE*, no pueden modificarse ya que este paquete no puede ser instalado en *STM32CubeMX* y no existe por tanto, en el directorio del proyecto, el archivo *.io* necesario para instalar el paquete de *X-CUBE-AI*, cargar la red neuronal y poder realizar aplicaciones de Inteligencia Artificial.

Por otra parte, también se intentó trabajar desde un proyecto vacío e importar los archivos necesarios para hacer funcionar únicamente el acelerómetro lo cual tampoco acabo resultando factible principalmente por tres motivos:

1. La gran cantidad de dependencias entre los ficheros, las cuales hacían imposible compilar el proyecto.
2. Debido al bug controlado de *STM32CubeIDE* que impide importar dependencias al importar proyectos al *workbench*.
3. El cambio en el sistema de funciones *BSP* a partir de la versión 4.4.1. *BSP* es una capa de abstracción que se compone de controladores específicos para la placa de desarrollo y que, en su mayoría, contiene drivers para memorias externas (como SDRAM o Flash QSPI), soporte

para pantallas lcd, pantallas táctiles, acelerómetros, etc. Este cambio en el ecosistema de funciones aumentó la complejidad a la hora de trazar las similitudes entre versiones.

Por último y como opción desarrollada en este TFG, se pensó que la mejor forma de solventar el problema era creando directamente drivers propios que permitieran hacer uso de los sensores necesarios de la placa *X-NUCLEO-IKS01A1* de forma sencilla y que, una vez desarrollados, permitieran al usuario una utilización rápida y eficaz, sin invertir tiempo arreglando dependencias ni tocando valores de registros internos.

Por tanto, para abordar el diseño del driver, en primer lugar se creó un programa destinado a establecer una comunicación sencilla, con manjeador de errores, a través de I^2C entre el acelerómetro y el microcontrolador. Dicho código se presenta a continuación:

Código 3.1 Comunicación a través de I2C.

```
I2C_HandleTypeDef hi2c1;
HAL_StatusTypeDef ret;
uint8_t buf[13];

while(1){
buf[0]=REG_ACC;
ret = HAL_I2C_Master_Transmit(&hi2c1, LSM6DS0_ADDR, buf, 1,
    HAL_MAX_DELAY);
if(ret != HAL_OK){
  strcpy((char*)buf,"Error SCL!!\r\n");
} else {
ret = HAL_I2C_Master_Receive(&hi2c1, LSM6DS0_ADDR, buf, 2,
    HAL_MAX_DELAY);
if(ret != HAL_OK){
    strcpy((char*)buf,"Error SDA!!\r\n");
  } else {
   strcpy((char*)buf,"Todo OK!!\r\n");
   }

HAL_UART_Transmit(&huart2, buf,strlen((char *)buf), HAL_MAX_DELAY);
}
```

Donde se hace uso de la función:

Código 3.2 Función HAL_StatusTypeDef HAL_I2C_Master_Transmit.

```
HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c,
uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Timeout)
```

Donde:

- ***hi2c***: Puntero a una instancia de la estructura *I2C_HandleTypeDef*, definida en ***stm32f4xx_hal_i2c.h***. Contiene la información de configuración para el I^2C especificado.
- ***DevAddress***: Dirección del dispositivo esclavo. Tal y como se especifica en su datasheet [26], la dirección del acelerómetro es 0xD6.
- ***pData***: Puntero al array buffer que contiene la secuencia de bytes a transmitir.
- ***Size***: Cantidad de datos a enviar.

- ***Timeout***: Indica el tiempo máximo, en milisegundos, antes de finalizar la transmisión.

El resultado de esta función es de tipo *HAL_StatusTypeDef* y puede ser: *HAL_OK, HAL_ERROR, HAL_BUSY, HAL_TIMEOUT*[2]. Si el resultado es distinto de *HAL_OK*, se copia un mensaje de error en el buffer. En caso contrario, se realiza la llamada a la función:

Código 3.3 Función HAL_StatusTypeDef HAL_I2C_Master_Receive.

```
HAL_StatusTypeDef HAL_I2C_Master_Receiven(I2C_HandleTypeDef *hi2c,
    uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Timeout
    )
```

Donde:

- ***hi2c***: Puntero a una instancia de la estructura *I2C_HandleTypeDef*, definida en ***stm32f4xx_hal_i2c.h***. Contiene la información de configuración para el I^2C especificado.
- ***DevAddress***: Dirección del dispositivo. Tal y como se especifica en su datasheet [26], la dirección del acelerómetro es 0xD6.
- ***pData***: Puntero al array buffer que contiene la secuencia de bytes a transmitir.
- ***Size***: Cantidad de datos a enviar.
- ***Timeout***: Indica el tiempo máximo, en milisegundos, antes de finalizar la transmisión.

Nuevamente, si el resultado de la función *HAL_StatusTypeDef HAL_I2C_Master_Receive* es distinto de *HAL_OK*, se copia un mensaje de error al buffer. En caso contrario, se copia el mensaje *Todo OK!!* para indicar que la transmisión se ha realizado sin inconvenientes.

Una vez el resultado fue el esperado, se procedió a la lectura del acelerómetro. Debido a la escasa documentación y al hecho, ya comentado, de que el acelerómetro LSM6DS0 no tuviera soporte dentro de *STM32CubeMX* hicieron de esta parte una de las más complejas del proyecto.

En primer lugar, se debe acudir al datasheet del acelerómetro *LSM6DS0* [26] para conocer cómo se deben leer los datos. El módulo inercial LSM0 posee dos modos de operación: acelerómetro activo y giróscopo apagado y ambos, acelerómetro y giróscopo, encendidos operando al mismo ODR[3]. Cambiar de un modo a otro requiere una operación de escritura: para la operación en modo normal, en el que únicamente el acelerómetro está activo, se debe escribir sobre *CTRL_REG6_XL (20h)*, mientras que en el modo en el que ambos están activos, se debe escribir sobre *CTRL_REG1_G (10h)*.

En este proyecto no se utilizan las medidas del giróscopo, por lo que el modo de operación será el modo normal.

Una vez establecido el modo de operación, se debe buscar cómo realizar las lecturas del mismo. Según su datasheet, para realizar múltiples lecturas, se debe leer desde los registros *OUT_X_XL (28h - 29h)* hasta los registros *OUT_Z_XL (2Ch - 2Dh)* y, una vez leídos éstos, el sistema empieza nuevamente desde *OUT_X_XL (28h - 29h)*. Estas operaciones se pueden observar en la Figura 3.2, extraída del datasheet.

En primer lugar se debe configurar el ODR (Output Data Rate). Mirando el datasheet del acelerómetro, las únicas frecuencias posibles son 10 Hz, 50 Hz, 119 Hz, 238 Hz, 476 Hz y 952 Hz. Si el modo de operación configurado es el del acelerómetro funcionando y el giróscopo apagado (el modo de operación establecido por defecto) se debe modificar el registro *CTRL_REG6_XL*. En este primer código, el valor del ODR será el máximo, 952 Hz, aunque más adelante se realizarán modificaciones para que dicho valor pueda ser seleccionado por el usuario de entre los posibles.

[2] Definición de la estructura en *stm32f4xx_hal_def.h*

[3] Output Data Rate

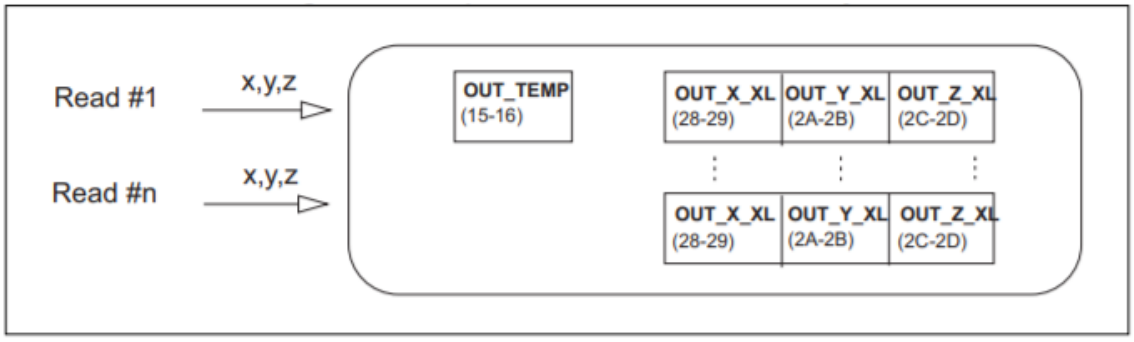

Figura 3.2 Múltiples lecturas del acelerómetro del módulo inercial LSM6DS0. Imagen extraída del datasheet.

En segundo lugar, se debe seleccionar la escala de trabajo del acelerómetro, a saber: ±2/±4/±8/±16 g. Dependiendo de la escala elegida, la variable *sensitivity* tomará diferentes valores de aquéllos definidos en *LSM6DS0_CUSTOM.h*.

Por último, se debe atender a la representación de los valores. Según el datasheet, el valor leído en estos registros es expresado en una palabra de 16-bits en complemento a dos. Esto es, para formar la palabra de 16-bits de *OUT_X_XL*, se deben leer los registros *OUT_X_L_XL* y *OUT_X_H_XL*, cada uno de 8-bits, y concatenarlos.

Se expone a continuación el código que lleva a cabo dichas operaciones:

Código 3.4 Configuracion del ODR y lectura de los ejes del acelerómetro LSM6DS0.

```
static char dataOut[256];
int16_t dataRaw[3];
float sensitivity = 0;
uint8_t regValue[6] = {0, 0, 0, 0, 0, 0};
uint8_t i, j, k;
uint8_t numberOfByteForDimension;
uint8_t value;

//Configuracion del ODR

HAL_I2C_Mem_Read(I2C_handler, LSM6DS0_ADDR ,
    LSM6DS0_ACC_GYRO_CTRL_REG6_XL, I2C_MEMADD_SIZE_8BIT, &value, 1,
    I2C_EXPBD_Timeout);

value &= ~LSM6DS0_ACC_GYRO_ODR_XL_MASK;
value |= LSM6DS0_ACC_GYRO_ODR_XL_952Hz;

HAL_I2C_Mem_Write(I2C_handler, LSM6DS0_ADDR ,
    LSM6DS0_ACC_GYRO_CTRL_REG6_XL, I2C_MEMADD_SIZE_8BIT, &value, 1,
    I2C_EXPBD_Timeout);

  //Lectura del acelerometro

numberOfByteForDimension = 6 / 3;
```

```
    k = 0;

    for (i = 0; i < 3; i++ )
    {
        for (j = 0; j < numberOfByteForDimension; j++ )
        {
          HAL_I2C_Mem_Read(I2C_handler, LSM6DS0_ADDR,
              LSM6DS0_ACC_GYRO_OUT_X_L_XL + k, I2C_MEMADD_SIZE_8BIT,
            &regValue[k], 1, I2C_EXPBD_Timeout);
            k++;
        }
    }

    dataRaw[0] = ((((int16_t)regValue[1]) << 8) + (int16_t)regValue[0]);
    dataRaw[1] = ((((int16_t)regValue[3]) << 8) + (int16_t)regValue[2]);
    dataRaw[2] = ((((int16_t)regValue[5]) << 8) + (int16_t)regValue[4]);

 /* Seleccionar Escala */
    float sensitivity = 0;

    switch( fullScale )
    {
      case FS_LOW:
        sensitivity = LSM6DS0_ACC_SENSITIVITY_FOR_FS_2G;
        break;
      case FS_MID:
        sensitivity = LSM6DS0_ACC_SENSITIVITY_FOR_FS_4G;
        break;
      case FS_HIGH:
        sensitivity = LSM6DS0_ACC_SENSITIVITY_FOR_FS_8G;
        break;
      case FS_EXTRA_HIGH:
        sensitivity = LSM6DS0_ACC_SENSITIVITY_FOR_FS_16G;
        break;
      default:
        sensitivity = -1.0f;
    }

/* Calculate the data. */
  acceleration.AXIS_X = ( int32_t )( dataRaw[0] * sensitivity );
  acceleration.AXIS_Y = ( int32_t )( dataRaw[1] * sensitivity );
  acceleration.AXIS_Z = ( int32_t )( dataRaw[2] * sensitivity );

 snprintf(dataOut, 256, "%d,%d,%d\r\n", (int)acceleration.AXIS_X, (int)
     acceleration.AXIS_Y, (int)acceleration.AXIS_Z);

HAL_UART_Transmit(&huart2, dataOut, strlen(dataOut), HAL_MAX_DELAY);
```

Cabe destacar el uso de la función *HAL_I2C_Mem_Read* en lugar de *HAL_I2C_Master_Receive*. Ésto es debido a que esta última función no soporta el tipo de transferencia I^2C requerido para leer

datos de una *sub-dirección*.

El término *sub-direccionamiento* (del inglés *sub-addressing*) se refiere al uso, por parte de los dispositivos basados en I2C, de direcciones internas que son independientes de la dirección del dispositivo, tal y como se define en la especificación del protocolo I^2C. Para admitir este esquema de direccionamiento interno, el fabricante del dispositivo define un protocolo de comunicaciones que utiliza *bytes* dentro del patrón de datos para definir estas direcciones.

En el caso del módulo inercial LSM6DS0, el protocolo I^2C implementado se comporta como un dispositivo esclavo, y se debe cumplir para la lectura de datos:

1. La transacción comienza cuando el dispositivo maestro envía una condición de inicio *ST* (START condition). Una *ST* es definida como una transición de nivel alto a bajo en la línea de datos SDA (Serial Data Line) mientras que la línea *SCL* (Serial Clock Line) se mantiene en nivel alto.
2. Después de la *ST*, el dispositivo maestro envía 8 bits. Los 7 primeros indican la dirección del esclavo, *SAD* (Slave Address), y el último se corresponde con un de bit de escritura, *W*.
3. El dispositivo esclavo confirma la recepción de los datos anteriormente enviados devolviendo un bit de reconocimiento, *SAK* (Slave Acknowledge).
4. El dispositivo maestro envía la sub-dirección al dispositivo esclavo *SUB* (Sub-address) .
5. El dispositivo esclavo confirma la recepción de estos datos con un *SAK*.
6. El dispositivo maestro envía una condición repetida de inicio, *RS* (Repeated Start).
7. El dispositivo esclavo responde con un *SAK*.
8. Después, el dispositivo esclavo transmite los datos. Cada transferencia de datos contiene 8 bits.
9. Si el dispositivo maestro espera más datos, éste responde con una confirmación de recepción de datos *MAK* (Master Acknowledge). Si por el caso contrario no espera ningún dato más, responde con un *NMAK* (No Master Acknowledge) indicando al dispositivo esclavo que detenga las comunicaciones y libere el bus.
10. Cuando la transferencia de datos finalice, el dispositivo maestro enviará una una condición de parada *SP* (STOP condition). La condición de parada es generada por una transición de nivel bajo a nivel alto en la línea *SDA*, mientras que la línea *SCL* se mantiene en nivel alto.

Información específica acerca de este protocolo puede ser consultada en el datasheet del acelerómetro LSM6DS0. Así mismo, en las Figuras 3.3 y 3.4 se puede observar de manera más gráfica este proceso para la lectura de un único byte y para la lectura de múltiples, respectivamente.

Master	ST	SAD + W		SUB		SR	SAD + R			NMAK	SP
Slave			SAK		SAK			SAK	DATA		

Figura 3.3 Proceso de transferencia cuando el dispositivo maestro está recibiendo (leyendo) un byte de datos del esclavo..

Una vez se pueden leer los datos del acelerómetro correctamente, lo siguiente fue crear dos archivos, *LSM6DS0_CUSTOM.h* y *LSM6DS0_CUSTOM.c*, con el objetivo de que, con sólo copiarlos en el proyecto en el que se esté trabajando, el usuario pudiera hacer uso de las funcionalidades del acelerómetro LSM6DS0 de forma rápida y sencilla.

Por una parte, el archivo *LSM6DS0_CUSTOM.h* contiene las direcciones de memoria, registros y definiciones de las estructuras necesarias para configurar correctamente el acceso al sensor.

Master	ST	SAD+W		SUB		SR	SAD+R			MAK		MAK		NMAK	SP
Slave			SAK		SAK			SAK	DATA		DAT A		DATA		

Figura 3.4 Proceso de transferencia cuando el dispositivo maestro está recibiendo (leyendo) múltiples bytes de datos del esclavo..

Por otra parte, el archivo *LSM6DS0_CUSTOM.c* contiene la definición de las funciones que puede utilizar el usuario. El objetivo que se ha seguido a la hora de diseñar las funciones que conforman este fichero ha sido que éstas se mantuvieran simples a la par que prácticas, brindándole al usuario la posibilidad de modificar ciertos parámetros propios del acelerómetro de forma ágil.Con este objetivo en mente, se han creado dos funciones; *Inicializar_LSM6DS0* y *Obtener_Ejes_LSM6DS0*.

La función *Inicializar_LSM6DS0* se debe usar, como su propio nombre indica, para la inicialización del LSM6DS0. Se encarga de configurar el modo de operación y el ODR. Su prototipo es el siguiente:

Código 3.5 Prototipo de la función Inicializar_LSM6DS0.

```
void Inicializar_LSM6DS0(I2C_HandleTypeDef *I2C_handler, float odr)
```

Donde:

1. ***I2C_handler***: Puntero a una instancia de una estructura de tipo *I2C_HandleTypeDef* , definida en *stm32f4xx_hal_i2c.h*. Contiene la información de configuración para el I^2C especificado.
2. ***odr***: Valor para el ODR. El valor introducido se aproximará a uno de los valores de ODR admisibles por el acelerómetro. Dichos valores aparecen en *LSM6DS0_CUSTOM.h* o pueden consultarse directamente en el datasheet [26].

El código completo de esta función es el que se muestra a continuación:

Código 3.6 Función Inicializar_LSM6DS0.

```
void Inicializar_LSM6DS0(I2C_HandleTypeDef *I2C_handler, float odr){

    LSM6DS0_ACC_GYRO_ODR_XL_t new_odr;
    uint8_t value;

    new_odr = ( odr <= 10.0f ) ? LSM6DS0_ACC_GYRO_ODR_XL_10Hz
              : ( odr <= 50.0f ) ? LSM6DS0_ACC_GYRO_ODR_XL_50Hz
              : ( odr <= 119.0f ) ? LSM6DS0_ACC_GYRO_ODR_XL_119Hz
              : ( odr <= 238.0f ) ? LSM6DS0_ACC_GYRO_ODR_XL_238Hz
              : ( odr <= 476.0f ) ? LSM6DS0_ACC_GYRO_ODR_XL_476Hz
              :                     LSM6DS0_ACC_GYRO_ODR_XL_952Hz;

      HAL_I2C_Mem_Read(I2C_handler, LSM6DS0_ADDR ,
          LSM6DS0_ACC_GYRO_CTRL_REG6_XL, I2C_MEMADD_SIZE_8BIT, &value, 1,
           I2C_EXPBD_Timeout);

    value &= ~LSM6DS0_ACC_GYRO_ODR_XL_MASK;
    value |= new_odr;
```

```
    HAL_I2C_Mem_Write(I2C_handler, LSM6DS0_ADDR ,
        LSM6DS0_ACC_GYRO_CTRL_REG6_XL, I2C_MEMADD_SIZE_8BIT, &value, 1,
        I2C_EXPBD_Timeout);

}
```

La función *Obtener_Ejes_LSM6DS0* es la que se utiliza para obtener los valores del acelerómetro. Su prototipo es el siguiente:

Código 3.7 Prototipo de la función Obtener_Ejes_LSM6DS0.

```
void Obtener_Ejes_LSM6DS0(I2C_HandleTypeDef *I2C_handler,
    SensorAxesFloat_t *acceleration, SensorFs_t fullScale);
```

Donde:

1. ***I2C_handler***: Puntero a una instancia de una estructura de tipo *I2C_HandleTypeDef*, definida en *stm32f4xx_hal_i2c.h*. Contiene la información de configuración para el I^2C especificado.
2. ***acceleration***: Puntero a una instancia de una estructura de tipo *SensorAxesFloat_t*, definida en *LSM6DS0_CUSTOM.h*. Almacenará los datos de lectura del acelerómetro cada ciclo.
3. ***fullScale***: Instancia de una estructura de tipo *SensorAxesFloat_t*, definida en *LSM6DS0_CUSTOM.h*. Indica cual es la escala del acelerómetro, pudiendo elegir entre *FS_LOW, FS_MID, FS_HIGH, FS_EXTRA_HIGH*, los cuales se corresponden con ±2g, ±4g, ±8g y ±16g, respectivamente.

El código completo de esta función queda de la siguiente forma:

Código 3.8 Función Obtener_Ejes_LSM6DS0.

```
void Obtener_Ejes_LSM6DS0(I2C_HandleTypeDef *I2C_handler,
    SensorAxesFloat_t *acceleration, SensorFs_t fullScale)
{
  HAL_StatusTypeDef ret;
  uint8_t buf[13];
  SensorAxes_t aux_acceleration;

    buf[0]=REG_ACC;
    ret = HAL_I2C_Master_Transmit(I2C_handler, LSM6DS0_ADDR, buf, 1,
        HAL_MAX_DELAY);
    if(ret != HAL_OK){
      printf("Error SCL!!\r\n");
    } else {
    ret = HAL_I2C_Master_Receive(I2C_handler, LSM6DS0_ADDR, buf, 2,
        HAL_MAX_DELAY);
    if(ret != HAL_OK){
        printf("Error SDA!!\r\n");
      } else {

    int16_t dataRaw[3];
```

```
  uint8_t regValue[6] = {0, 0, 0, 0, 0, 0};

  uint8_t i, j, k;
  uint8_t numberOfByteForDimension;

  numberOfByteForDimension = 6 / 3;

  k = 0;
  for (i = 0; i < 3; i++ )
  {
      for (j = 0; j < numberOfByteForDimension; j++ )
      {
        HAL_I2C_Mem_Read(I2C_handler, LSM6DS0_ADDR,
            LSM6DS0_ACC_GYRO_OUT_X_L_XL + k, I2C_MEMADD_SIZE_8BIT, &
            regValue[k], 1, I2C_EXPBD_Timeout);
        k++;
      }
  }

  dataRaw[0]= ((((int16_t)regValue[1]) <<8 ) + (int16_t )regValue[0]);
  dataRaw[1]= ((((int16_t)regValue[3]) << 8) + (int16_t )regValue[2]);
  dataRaw[2]=((((int16_t)regValue[5]) << 8 ) + (int16_t )regValue[4]);


/* Seleccionar Escala */
  float sensitivity = 0;

  switch( fullScale )
  {
    case FS_LOW:
      sensitivity = LSM6DS0_ACC_SENSITIVITY_FOR_FS_2G;
      break;
    case FS_MID:
      sensitivity = LSM6DS0_ACC_SENSITIVITY_FOR_FS_4G;
      break;
    case FS_HIGH:
      sensitivity = LSM6DS0_ACC_SENSITIVITY_FOR_FS_8G;
      break;
    case FS_EXTRA_HIGH:
      sensitivity = LSM6DS0_ACC_SENSITIVITY_FOR_FS_16G;
      break;
    default:
      sensitivity = -1.0f;
  }

/* Calcular valores de salida */

aux_acceleration.AXIS_X = ( int32_t )( dataRaw[0] * sensitivity ) ;
aux_acceleration.AXIS_Y = ( int32_t )( dataRaw[1] * sensitivity ) ;
aux_acceleration.AXIS_Z = ( int32_t )( dataRaw[2] * sensitivity ) ;
```

```
  acceleration->AXIS_X = ( float ) aux_acceleration.AXIS_X ;
  acceleration->AXIS_Y = ( float ) aux_acceleration.AXIS_Y ;
  acceleration->AXIS_Z = ( float ) aux_acceleration.AXIS_Z ;

    }

   }
}
```

3.3 Diseño del *framework* de la red neuronal

El paquete de expansión *X-CUBE-AI* de *STM32CubeMX* ofrece al usuario la conversión automática de redes neuronales pre-entrenadas y su posterior integración, mediante bibliotecas optimizadas, al espacio de trabajo del proyecto. Permite además, la posibilidad de validar modelos de redes neuronales tanto en PC como en placas STM32, así como medir el rendimiento directamente en los propios dispositivos. Para ello permite elegir entre tres "aplicaciones" para empezar a desarrollar proyectos de inteligencia artificial de forma ágil. Estas aplicaciones son:

1. *SystemPerformance*: Una aplicación de prueba de IA que se puede ejecutar de forma independiente sin necesidad de escribir código. Tiene como fin el análisis de rendimiento de redes neuronales.
2. *Validation*: Una aplicación de prueba de IA con fines de validación de redes. También se puede ejecutar de forma independiente.
3. *ApplicationTemplate*: Su función es actuar a modo de *template* como base de aplicaciones de AI.

En la Figura 3.5 se puede ver una imagen de la configuración de *X-CUBE-AI*.

El primer problema al seleccionar *ApplicationTemplate* consiste en que el propio código creado no imprime texto alguno por pantalla a través de la USART. Ésto es debido a que dicho template hace uso de *printf*, función que se encuentra definida de forma débil en el código de ST y por lo cual debe reescribirse. Los pasos a seguir para arreglar este problema pueden ser consultados en el anexo A.2.

Sin embargo, el principal problema consiste en que, como bien se avisa en el manual de usuario del paquete *X-CUBE-AI* [28], dicho proyecto generado no es realmente un template completo de una aplicación de ejemplo de uso. En realidad, este template únicamente proporciona la estructura para redes neronales con una única entrada y una única salida y si se desea modificar, lo cual es casi necesario para cualquier aplicación de IA, resulta confuso y se deben realizar numerosos cambios en código externo, no definido en *main.c*, haciendo el traspaso de datos complejo y perdiendo de vista el flujo del código.

Ante esta problemática, surgió la idea de crear un *framework* que, añadido en *main.c*, permitiera al usuario utilizar una o varias redes neuronales previamente cargadas con STM32CubeMX, de forma sencilla y con código 100 % controlado. Para ello no se necesita añadir ninguna aplicación, ya que ésta será desarrollada en este TFG y únicamente resulta necesario marcar el check de *Core* (Figura 3.5), para generar las librerías y los pesos de la red neuronal pre-entrenada.

Presentada la problemática, se pasan a comentar las modificaciones necesarias que se van a realizar sobre el el fichero *main.c*, con el objetivo de crear dicho *framework* donde se podrá implementar *cualquier red neuronal*[4] en la placa de desarrollo.

[4] siempre y cuando la red cumpla con las limitaciones de espacio del dispositivo.

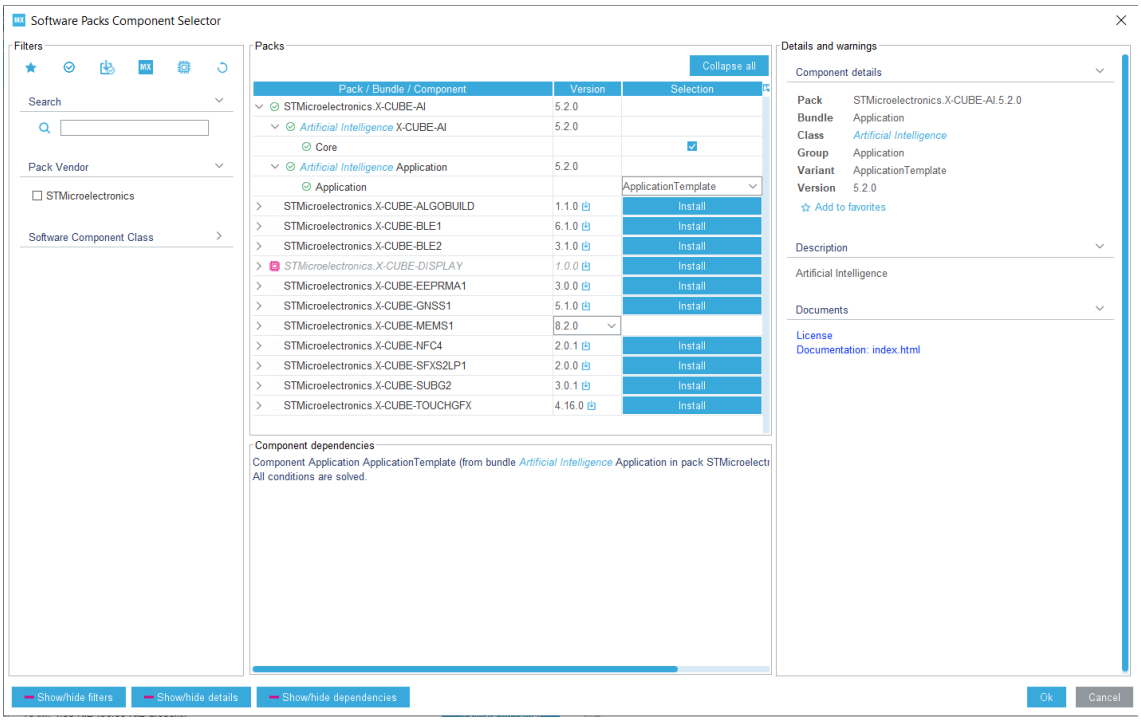

Figura 3.5 Aplicaciones posibles dentro del paquete X-CUBE-AI de STM32CubeMX.

3.3.1 Inclusión de archivos de cabecera

El primer paso será añadir los archivos *.h* creados en el anexo A. Es de importancia introducir dicho código en las regiones habilitadas para ello, en caso contrario, se perderá el código al modificar el fichero *.io*.

Código 3.9 Ficheros de cabecera para el framework de la red neuronal.

```
  /* Private includes
     ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "ai_platform.h"
#include "acc_neural_model.h"
#include "acc_neural_model_data.h"
/* USER CODE END Includes */
```

El fichero *ai_platform.h*, corresponde al *runtime* de *CUBE.AI*, mientras que los ficheros *acc_neural_model.h* y *acc_neural_model_data.h* son los específicos de cada red neuronal que se implemente. Los pasos para generar estos ficheros se comentan ampliamente en el anexo A.1.2.

3.3.2 Inclusión de buffers

El siguiente paso es añadir los búferes que usará la red neuronal.

Código 3.10 Búferes para el framework de la red neuronal.

```
/* USER CODE BEGIN 0 */
//Parametros Red Neuronal
ai_handle acc_neural_model2;
```

```
float aiInData[AI_ACC_NEURAL_MODEL_IN_1_SIZE];
float aiOutData[AI_ACC_NEURAL_MODEL_OUT_1_SIZE];
uint8_t activations2[AI_ACC_NEURAL_MODEL_DATA_ACTIVATIONS_SIZE];
const char* activities[AI_ACC_NEURAL_MODEL_OUT_1_SIZE] = {
  "Parado", "Caminando", "Corriendo"
};
/* USER CODE END 0 */
```

El búfer *activations* se utilizará en tiempo de ejecución por *CUBE.AI* para, durante la fase de inferencia, almacenar los resultados intermedios mientras que *aiInData* y *aiOutData* serán los búferes de entrara y salida de la red neuronal, respectivamente. El tamaño de éstos está especificado en el archivo *acc_neural_model.h*. En el caso de *AI_ACC_NEURAL_MODEL_IN_1_SIZE* su valor depende principalmente del vector de entrada (o *frames* del inglés), que está a su vez relacionado con el tiempo que el dispositivo está capturando datos para la inferencia, multiplicados por las dimensiones del mismo, en este caso, los 3 ejes cartesianos. En el caso de *AI_ACC_NEURAL_-MODEL_OUT_1_SIZE*, depende de los posibles valores de salida de la red neuronal, tres en este caso. Por último, el búfer *activities* contiene las etiquetas de salida. Estos literales deben ser modificados por el usuario en función del uso propio de la red neuronal.

3.3.3 Funciones principales

A continuación se añaden las funciones que compondrán el cuerpo del *framework*. Éstas deben ser agregadas en la región comprendida entre *USER CODE BEGIN 4* Y *USER CODE END 4*, al final del archivo *main.c*.

Función *AI_Init*

Esta función se encarga de crear una instancia del modelo así como de inicializarlo.

Código 3.11 Función AI_Init.

```
static void AI_Init(ai_handle w_addr, ai_handle act_addr)
{
  ai_error err;

 /* 1 - Create an instance of the model */
  err = ai_acc_neural_model_create(&acc_neural_model2,
      AI_ACC_NEURAL_MODEL_DATA_CONFIG);
  if (err.type != AI_ERROR_NONE) {
    printf("ai_network_create error - type=%d code=%d\r\n", err.type,
        err.code);
    Error_Handler();
  }

    /* 2 - Initialize the instance */
  const ai_network_params params = {
      AI_ACC_NEURAL_MODEL_DATA_WEIGHTS(w_addr),
      AI_ACC_NEURAL_MODEL_DATA_ACTIVATIONS(act_addr)
  };

  if (!ai_acc_neural_model_init(acc_neural_model2, &params)) {
    err = ai_acc_neural_model_get_error(acc_neural_model2);
```

```
    printf("ai_network_init error - type=%d code=%d\r\n", err.type, err.
        code);
    Error_Handler();
  }
}
```

Función *AI_Run*

Esta función crea manejadores para los búferes de la red con la definición predeterminada de *AI buffer* que aparece en el fichero *ai_platform.h* y que se utiliza en *acc_neural_model.h* según los datos generados para cada modelo. Después, estos búferes se actualizan con los datos facilitados por el usuario.

Código 3.12 Función AI_Run.

```
static void AI_Run(float *pIn, float *pOut)
{
  ai_i32 batch;
  ai_error err;

  /* 1 - Create the AI buffer IO handlers with the default definition */
  ai_buffer ai_input[AI_ACC_NEURAL_MODEL_IN_NUM] =
      AI_ACC_NEURAL_MODEL_IN;
  ai_buffer ai_output[AI_ACC_NEURAL_MODEL_OUT_NUM] =
      AI_ACC_NEURAL_MODEL_OUT;

  /* 2 - Update IO handlers with the data payload */
  ai_input[0].n_batches = 1;
  ai_input[0].data = AI_HANDLE_PTR(pIn);
  ai_output[0].n_batches = 1;
  ai_output[0].data = AI_HANDLE_PTR(pOut);

  batch = ai_acc_neural_model_run(acc_neural_model2, ai_input, ai_output
      );
  if (batch != 1) {
    err = ai_acc_neural_model_get_error(acc_neural_model2);
    printf("AI ai_network_run error - type=%d code=%d\r\n", err.type,
        err.code);
    Error_Handler();
  }
}
```

Función *argmax*

El objetivo de esta función es devolver la posición del valor más alto a la salida de la red neuronal.

Código 3.13 Función argmax.

```
static uint32_t argmax(const float * values, uint32_t len)
{
  float max_value = values[0];
```

```
  uint32_t max_index = 0;
  for (uint32_t i = 1; i < len; i++) {
    if (values[i] > max_value) {
      max_value = values[i];
      max_index = i;
    }
  }
  return max_index;
}
```

3.3.4 Cambios dentro de la función *main*

Lo primero es inicializar la red neuronal, llamando a la función *AI_Init* previamente definida:

Código 3.14 Llamada a la función AI_Init.

```
  /* USER CODE BEGIN 2 */

  AI_Init(ai_acc_neural_model_data_weights_get(), activations2);

  /* USER CODE END 2 */
```

Después, se crea el siguiente bucle while:

Código 3.15 Bucle while(1).

```
  SensorAxesFloat_t accelerationFloat;
  uint32_t write_index = 0;

  while (1)
   {
       Obtener_Ejes_LSM6DS0(&hi2c1,&accelerationFloat,FS_LOW);

           aiInData[write_index + 0] = (float) accelerationFloat.AXIS_X/
              2000.0f;
           aiInData[write_index + 1] = (float) accelerationFloat.AXIS_Y/
              2000.0f;
           aiInData[write_index + 2] = (float) accelerationFloat.AXIS_Z/
              2000.0f;

           printf("%.2f,%.2f,%.2f\r\n", aiInData[write_index + 0],
                      aiInData[write_index + 1], aiInData[write_index +
                         2]);

           write_index += 3;

           if (write_index == AI_ACC_NEURAL_MODEL_IN_1_SIZE) {
           write_index = 0;
```

```
          printf("Running inference\r\n");
          AI_Run(aiInData, aiOutData);

          /* Output results */
          for (uint32_t i = 0; i < AI_ACC_NEURAL_MODEL_OUT_1_SIZE; i++)
              {
            printf("%8.6f ", aiOutData[i]);
          }

          uint32_t class = argmax(aiOutData,
              AI_ACC_NEURAL_MODEL_OUT_1_SIZE);
          printf(": %d - %s\r\n", (int) class, activities[class]);

          printf("Esperando 4 segundos para nuevo movimiento...\r\n");
      HAL_Delay(4000);
        printf("Haz un movimiento!\r\n");
   }
}
```

La primera operación en este bucle es obtener las lecturas del acelerómetro mediante la llamada *Obtener_Ejes_LSM6DS0* y almacenar dicho valor en *accelerationFloat*.

A continuación, se guarda el valor de cada lectura en el búfer *aiInData*, dividiéndolo antes por la escala que hemos seleccionado, 2g en este caso, para obtener valores entre -1 y 1.

Después, se incrementa la variable usada como índice del búfer para la escritura de datos hasta que el valor alcanza *AI_ACC_NEURAL_MODEL_IN_1_SIZE*, momento en el que se resetea dicha variable y se llama a la función *AI_Run* para efectuar la inferencia a partir de los datos almacenados en el búfer *aiInData* y guardar el resultado en el búfer de salida *aiOutData*.

Por último, se recorre dicho búfer con la función *argmax* para obtener el índice del valor de salida más probable para dicha inferencia y se imprime el resultado por pantalla.

3.4 Sistema completo

En este apartado se incluye el código del sistema completo así como los desarrollos aún no comentados y necesarios para su implementación.

Utilizar el driver del acelerómetro es tan sencillo como incluir los ficheros *LSM6DS0_CUSTOM.h* y *LSM6DS0_CUSTOM.c* en *Core* → *Inc* y *Core* → *Src*, respectivamente. El código de estos ficheros se detalla a continuación:

Código 3.16 Fichero *LSM6DS0_CUSTOM.h*.

```
//INCLUIR LIBRERIAS ESTANDAR
#include <string.h>
#include <stdint.h>

//DEFINES ACELEROMETRO LSM6DS0
#define LSM6DS0_ADDR 0xD6
#define LSM6DS0_ACC_GYRO_OUT_X_L_XL 0X28
#define I2C_MEMADD_SIZE_8BIT 0x00000001U
#define I2C_EXPBD_Timeout 0x1000
#define LSM6DS0_ACC_GYRO_CTRL_REG6_XL 0X20
```

```
#define LSM6DS0_ACC_SENSITIVITY_FOR_FS_2G 0.061 /**< Sensitivity value
    for 2 g full scale [mg/LSB] */
#define LSM6DS0_ACC_SENSITIVITY_FOR_FS_4G 0.122 /**< Sensitivity value
    for 4 g full scale [mg/LSB] */
#define LSM6DS0_ACC_SENSITIVITY_FOR_FS_8G 0.244 /**< Sensitivity value
    for 8 g full scale [mg/LSB] */
#define LSM6DS0_ACC_SENSITIVITY_FOR_FS_16G 0.732 /**< Sensitivity value
    for 16 g full scale [mg/LSB] */
#define  LSM6DS0_ACC_GYRO_FS_XL_MASK 0x18


#define CONVERT_MG_TO_MS2 0.00980665f

//ESTRUCTURAS
typedef struct
{
  int32_t AXIS_X;
  int32_t AXIS_Y;
  int32_t AXIS_Z;
} SensorAxes_t;

typedef struct
{
  float AXIS_X;
  float AXIS_Y;
  float AXIS_Z;
} SensorAxesFloat_t;

typedef enum
{
  LSM6DS0_ACC_GYRO_FS_XL_2g   = 0x00,
  LSM6DS0_ACC_GYRO_FS_XL_16g  = 0x08,
  LSM6DS0_ACC_GYRO_FS_XL_4g   = 0x10,
  LSM6DS0_ACC_GYRO_FS_XL_8g   = 0x18,
} LSM6DS0_ACC_GYRO_FS_XL_t; //SENSIBILIDAD

typedef enum
{
  FS_LOW,
  FS_MID,
  FS_HIGH,
  FS_EXTRA_HIGH
} SensorFs_t;
```

Código 3.17 Fichero *LSM6DS0_CUSTOM.c*.

```
#include "LSM6DS0_CUSTOM.h"
#include "main.h"
```

```
#include "stm32f4xx_hal_i2c.h"

//DECLARACIONES DE FUNCIONES
void Obtener_Ejes_LSM6DS0(I2C_HandleTypeDef *I2C_handler,
    SensorAxesFloat_t *acceleration, SensorFs_t fullScale);


//FUNCIONES
/**
 * @brief Get the LSM6DS0 accelerometer sensor axes
 * @param I2C_handler the device handle
 * @param acceleration pointer where the values of the axes are written
 * @param fullScale set the full scale value
 */
void Obtener_Ejes_LSM6DS0(I2C_HandleTypeDef *I2C_handler,
    SensorAxesFloat_t *acceleration, SensorFs_t fullScale)
{
  HAL_StatusTypeDef ret;
  uint8_t buf[13];
  SensorAxes_t aux_acceleration;

    buf[0]=REG_ACC;
    ret = HAL_I2C_Master_Transmit(I2C_handler, LSM6DS0_ADDR, buf, 1,
        HAL_MAX_DELAY);
    if(ret != HAL_OK){
      printf("Error SCL!!\r\n");
    } else {
    ret = HAL_I2C_Master_Receive(I2C_handler, LSM6DS0_ADDR, buf, 2,
        HAL_MAX_DELAY);
    if(ret != HAL_OK){
        printf("Error SDA!!\r\n");
      } else {

    int16_t dataRaw[3];
    uint8_t regValue[6] = {0, 0, 0, 0, 0, 0};

    uint8_t i, j, k;
    uint8_t numberOfByteForDimension;

    numberOfByteForDimension = 6 / 3;

    k = 0;
    for (i = 0; i < 3; i++ )
    {
        for (j = 0; j < numberOfByteForDimension; j++ )
        {
          HAL_I2C_Mem_Read(I2C_handler, LSM6DS0_ADDR,
              LSM6DS0_ACC_GYRO_OUT_X_L_XL + k, I2C_MEMADD_SIZE_8BIT, &
              regValue[k], 1, I2C_EXPBD_Timeout);
            k++;
```

```
      }
    }

    dataRaw[0] = ((((int16_t)regValue[1]) << 8) + (int16_t)regValue[0]);
    dataRaw[1] = ((((int16_t)regValue[3]) << 8) + (int16_t)regValue[2]);
    dataRaw[2] = ((((int16_t)regValue[5]) << 8) + (int16_t)regValue[4]);


  /* Seleccionar Escala */
    float sensitivity = 0;

    switch( fullScale )
    {
      case FS_LOW:
        sensitivity = LSM6DSO_ACC_SENSITIVITY_FOR_FS_2G;
        break;
      case FS_MID:
        sensitivity = LSM6DSO_ACC_SENSITIVITY_FOR_FS_4G;
        break;
      case FS_HIGH:
        sensitivity = LSM6DSO_ACC_SENSITIVITY_FOR_FS_8G;
        break;
      case FS_EXTRA_HIGH:
        sensitivity = LSM6DSO_ACC_SENSITIVITY_FOR_FS_16G;
        break;
      default:
        sensitivity = -1.0f;
    }

  /* Calcular valores de salida */

  aux_acceleration.AXIS_X = ( int32_t )( dataRaw[0] * sensitivity ) ;
  aux_acceleration.AXIS_Y = ( int32_t )( dataRaw[1] * sensitivity ) ;
  aux_acceleration.AXIS_Z = ( int32_t )( dataRaw[2] * sensitivity ) ;

  acceleration->AXIS_X = ( float ) aux_acceleration.AXIS_X ;
  acceleration->AXIS_Y = ( float ) aux_acceleration.AXIS_Y ;
  acceleration->AXIS_Z = ( float ) aux_acceleration.AXIS_Z ;

     }
   }
}
```

Tal y como se menciona en apartados posteriores, en la sección 4.1.2, los datos se capturan a una frecuencia de 26Hz. Para asegurarlo, se hace uso de las características de la placa de desarrollo y se configura un timer, abriendo el fichero del proyecto *.io*.

Para elegir un timer, se debe hacer click sobre la pestaña *Timers* en el menu lateral izquierdo y seleccionar uno que no tenga el símbolo triangular amarillo, símbolo que indica que ese timer en concreto ha sido "parcialmente deshabilitado" debido a que alguno o todos sus canales han sido deshabilitados por entrar en conflicto con alguna configuración previa realizada. En concreto y para la

placa de desarrollo *STM32F401RE*, los timers parcialmente deshabilitados son el *TIM2,TIM4,TIM5* y *TIM9*, debido a que éstos entran en conflicto con la USART2 y el I2C1. Por ello, en este desarrollo se ha hecho uso del timer *TIM1*.

Para conocer las características del timer seleccionado, se debe acudir al datasheet de la placa STM32F401RE [24]. Según la figura 3.6, dicho timer posee una resolución de 16-bits, lo que permite contar hasta un valor máximo de 65535.

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary output	Max. interface clock (MHz)	Max. timer clock (MHz)
Advanced-control	TIM1	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	84	84

Figura 3.6 Características del timer *TIM1*.

Además resulta de utilidad consultar, en este mismo datasheet, el diagrama de bloques de la placa *STM32F401RE*, donde se puede observar a que preescalador está conectado dicho timer. En la Figura 3.7 se puede comprobar que *TIM1* está conectado al preescalador *APB2*.

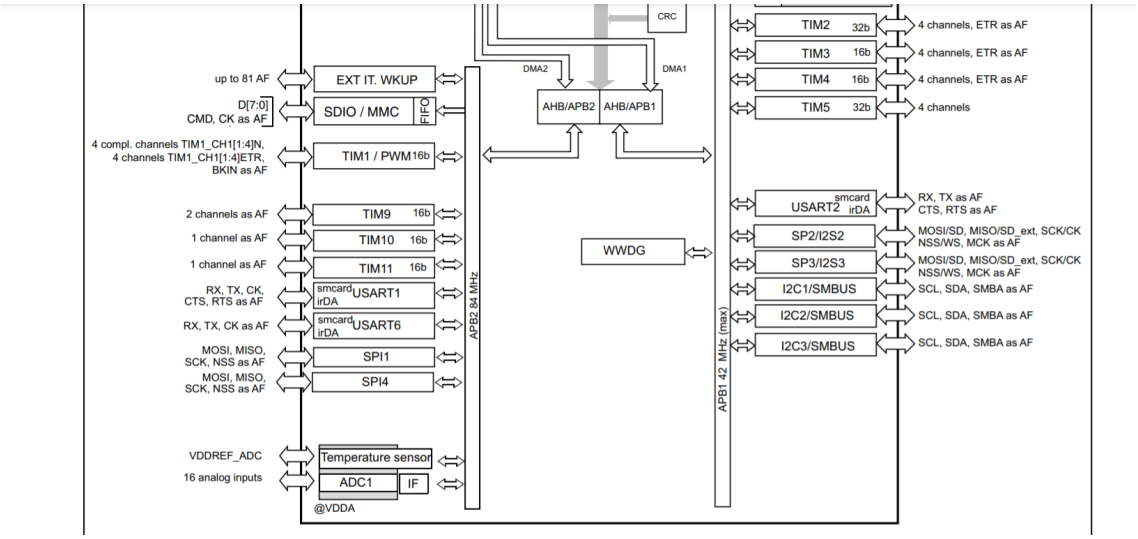

Figura 3.7 Fragmento del diagrama de bloques de la placa *STM32F401RE*.

Una vez conocidos estos datos, antes de proceder a la configuración del timer seleccionado, se debe examinar la configuración actual del reloj principal de la CPU, haciendo click sobre la pestaña *Clock Configuration* situada en la parte superior.

En la Figura 3.8 se puede observar que el reloj principal de la CPU, denominado *HCLK*, está operando a 84 MHz, su frecuencia máxima, y que el preescalador al cual está conectado *TIM1*, *APB2* está puesto a 1.

Ahora, se van a comentar los valores de configuración del timer. Puesto que se busca una frecuencia de muestreo de 26Hz, esto quiere decir que la subrutina de interrupción debe activarse cada 38,46 ms aproximadamente. Dado que el reloj principal está operando a 84 Mhz, si se configura el preescalador a 84 se puede obtener 1 Mhz en la frecuencia del timer. Hay que tener en cuenta que con estos valores, solo se podrán fijar períodos de hasta un máximo de 65,535 ms.

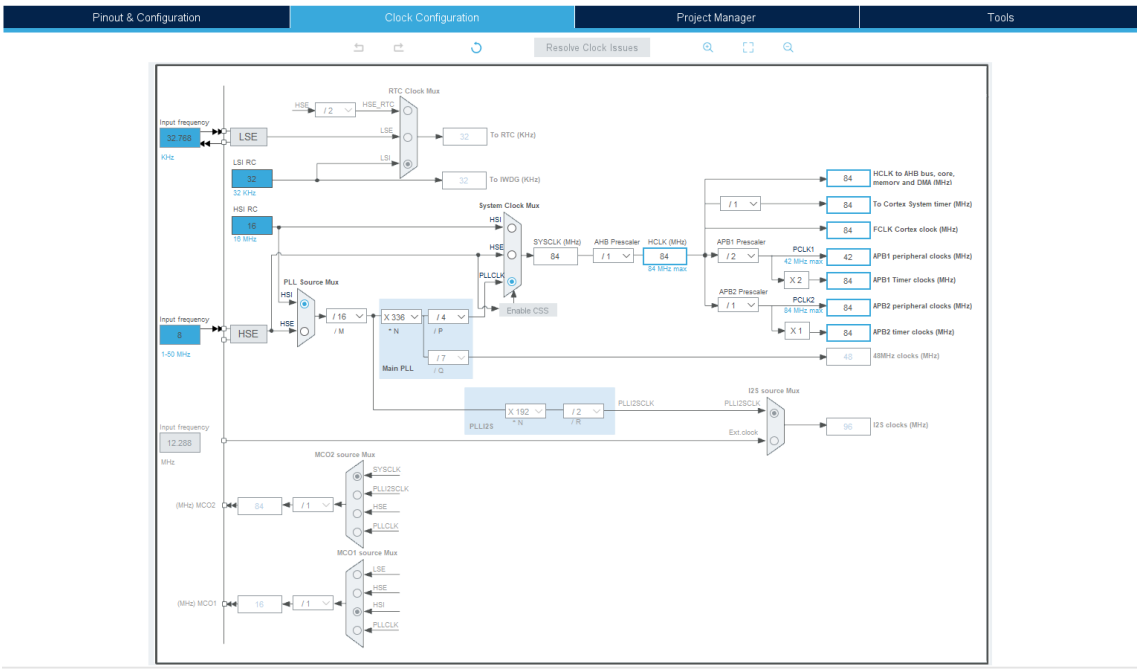

Figura 3.8 Configuración del reloj interno de la placa *STM32F401RE*.

$$\frac{84Mhz}{84} = 1Mhz \rightarrow \frac{1tick}{\mu s}$$

Donde el tiempo máximo según la Figura 3.6 es:

$$\text{Tiempo máximo: } 65535 \cdot 1\mu s = 65{,}535 \text{ ms}$$

Sin embargo, esto no supone problema alguno pues, como se ha comentado más arriba, únicamente se busca fijar el timer con un período de 38,46 ms. Para obtener el valor del contador, se debe realizar la siguiente operación:

$$\frac{1}{26} sec \cdot \frac{84Mhz}{84} \approx 38462 \text{ ticks}$$

Una vez obtenidos los valores, se pasa a introducir dichos parámetros en STM32CubeMX. Dentro de la ventana de configuración del timer *TIM1*, se debe seleccionar *Clock Source: Internal Clock* y dejar las demás opciones por defecto (canales deshabilitados).

Bajo la ventana *Configuration* → *Parameter Settings* → *Counter Settings* se deben introducir los parámetros previamente calculados con ciertos matices:

- *Prescaler (PSC - 16 bits value)*: 83. Se debe introducir 83 y no 84 debido a que si se introduce un valor de 0, el preescalador será configurado a 1, si se introduce un valor de 1, el preescalador será configurado a 2 y así sucesivamente.
- *Counter mode*: *Up*.
- *Counter Period (AutoReload Register - 16 bits value)* 38461. Nótese que se introduce un tick menos del calculado pues si se introdujera 38462, la interrupción saltaría en el tick 386463.

Estas configuraciones se pueden ver en la Figura 3.9

A continuación, se añade un fragmento del código del archivo *main.c*, con las modificaciones realizadas para incluir el timer.

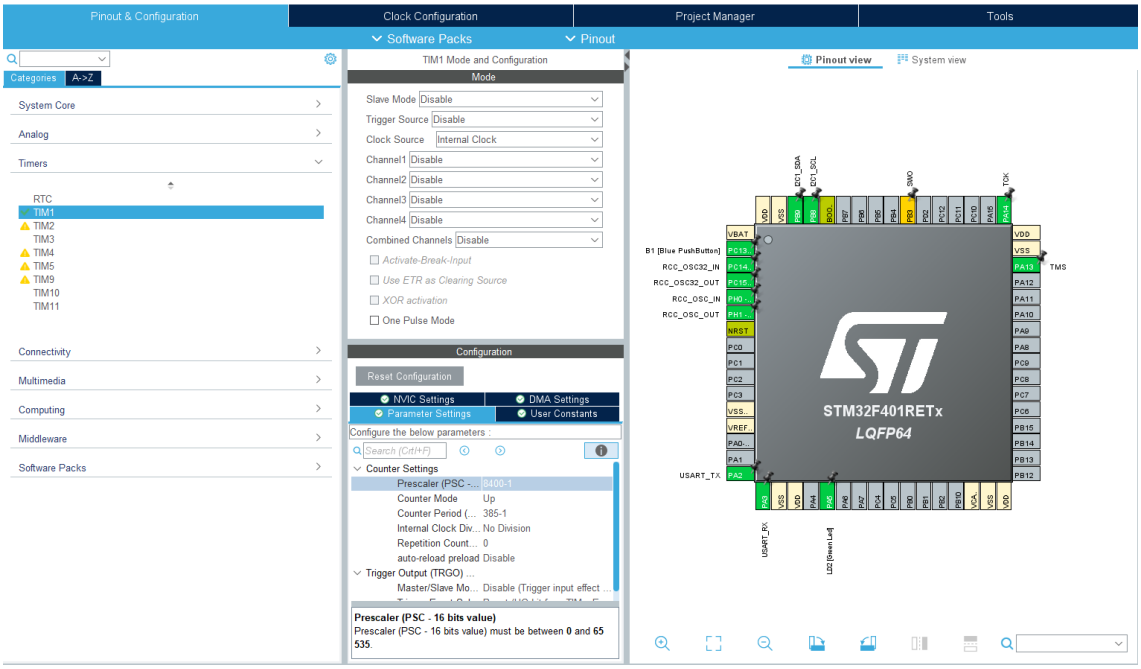

Figura 3.9 Configuración del timer *TIM1*.

Código 3.18 Fragmento del archivo *main.c*.

```
int main(void)
{
  /* MCU Configuration
      ----------------------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the
      Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_I2C1_Init();
  MX_CRC_Init();
  MX_TIM1_Init();

  /* USER CODE BEGIN 2 */
  RetargetInit(&huart2);

  printf("\r\n Iniciando Aplicacion de Reconocimiento...\r\n");

  AI_Init(ai_acc_neural_model_data_weights_get(), activations2);

  printf("Esperando 4 segundos para comenzar...\r\n");
  HAL_Delay(4000);
```

```
printf("Haz un movimiento!\r\n");
HAL_Delay(1000);

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */

//Iniciamos el timer
HAL_TIM_Base_Start_IT(&htim1);

while (1)
 {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  if(flagInterrupcion==1){

     flagInterrupcion=0;

     HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

     Obtener_Ejes_LSM6DS0(&hi2c1,&accelerationFloat,FS_MID);

         aiInData[write_index + 0] = (float) accelerationFloat.AXIS_X/
             4000.0f;
         aiInData[write_index + 1] = (float) accelerationFloat.AXIS_Y/
             4000.0f;
         aiInData[write_index + 2] = (float) accelerationFloat.AXIS_Z/
             4000.0f;

         printf("%.2f,%.2f,%.2f\r\n", aiInData[write_index + 0],
                    aiInData[write_index + 1], aiInData[write_index +
                        2]);

         write_index += 3;

         if (write_index == AI_ACC_NEURAL_MODEL_IN_1_SIZE) {
         write_index = 0;

         printf("Realizando inferencia...\r\n");
         AI_Run(aiInData, aiOutData);

         /* Output results */
         for (uint32_t i = 0; i < AI_ACC_NEURAL_MODEL_OUT_1_SIZE; i++)
             {
           printf("%8.6f ", aiOutData[i]);
         }
```

```
          uint32_t class = argmax(aiOutData,
              AI_ACC_NEURAL_MODEL_OUT_1_SIZE);
          printf(": %d - %s\r\n", (int) class, activities[class]);


      printf("Esperando 4 segundos para nuevo movimiento...\r\n");
      HAL_TIM_Base_Stop_IT(&htim1);
      HAL_Delay(4000);
      HAL_TIM_Base_Start_IT(&htim1);
      printf("Haz un movimiento!\r\n");
        }
    }
   }
  /* USER CODE END 3 */
}
```

Tal y como se aprecia en el código 3.18, la llamada *HAL_TIM_Base_Start_IT* inicia el timer. Como se puede leer en el manual de usuario del driver HAL para microcontroladores de la familia STM32F4 [21], se debe utilizar *HAL_TIM_Base_Start_IT* en lugar de *HAL_TIM_Base_Start* debido a que esta última función es la encargada de iniciar el timer en modo interrupción.

Cada vez que se genera una interrupción, se llama a la función *HAL_TIM_PeriodElapsedCallback*. Esta función aparece definida en el fichero *stm32f4xx_hal_tim.c*, bajo el tipo *__weak*, donde se informa de que para hacer uso de ella, debe ser implementada en los archivos de usuario. Por tanto, es definida en el fichero *main.c* y se enmarca entre las regiones */* USER CODE BEGIN 4 */* y */* USER CODE END 4 */*. Su código es el siguiente:

Código 3.19 Función *HAL_TIM_PeriodElapsedCallback*.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  //Verificamos que sea el timer 11 el que ha hecho saltar la
     interrupcion
  if(htim==&htim1){
    flagInterrupcion=1;
  }
```

Dicha función recibe como argumento un puntero a una estructura de tipo *TIM_HandleTypeDef*, la cual contiene la identificación del timer. Esta función puede ser llamada por cualquier timer que haya generado una interrupción por lo que, para crear un código más robusto con vistas a introducir un mayor número de redes neuronales y de timers, se pregunta qué timer ha hecho saltar la interrupción en cuestión. Si ese timer es el configurado anteriormente, *htim1*, se pone la variable *flagInterrupcion* a nivel alto. En el código 3.18, cada vez que la variable *flagInterrupcion* esté a nivel alto, se realiza la lectura del acelerómetro. El comportamiento del timer se puede observar de forma gráfica en la Figura 3.10.

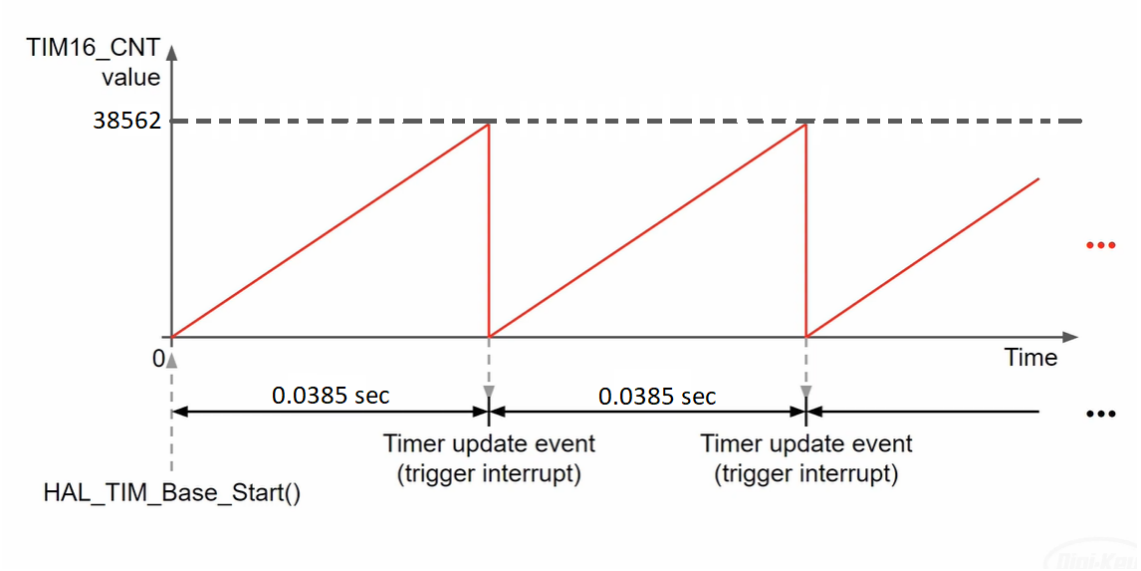

Figura 3.10 Funcionamiento del timer *TIM1*.

Cabe destacar que los timers entran en conflicto con la función *HAL_Delay*. Ésto es debido a que *HAL_Delay* usa la interrupción SysTick y, si la prioridad de SysTick es menor que la prioridad de la interrupción a la que se llama, se terminará en un bucle infinito, ya que SysTick nunca se invocará. Puesto que se quiere utilizar *HAL_Delay* con el propósito de que el usuario pueda ver por pantalla el resultado de la inferencia así como para introducir un tiempo de espera entre un movimiento y otro, en primer lugar, se para el timer con la función *HAL_TIM_Base_Stop_IT*, tal y como se puede ver en el código 3.18. Después, se realizan las llamadas *HAL_Delay* y por último se vuelve a iniciar el timer con *HAL_TIM_Base_Start_IT*. Ésto resuelve los conflictos y permite generar el comportamiento deseado.

4 Pruebas

No hay emoción más intensa para un inventor que ver una de sus creaciones funcionando. Esa emoción hace que uno se olvide de comer, de dormir, de todo.

- Nikola Tesla -

En este capítulo se presentarán las pruebas realizadas hasta llegar al sistema completo. Se incluirá además, al final de este capítulo, una demo del sistema completo funcionando.

4.1 Pruebas parciales

En este capítulo se presentarán las pruebas realizadas sobre la red neuronal de ST, se mostrarán los resultados y se expondrá la necesidad de adaptación de dicha red.

4.1.1 Pruebas con la red de ST

En primer lugar, la red de ST puede distinguir entre *parado*, *caminando* y *corriendo*. Al analizar la red con *STM32CubeMX* obtenemos las siguientes especificaciones:

- Complejidad: 24053 operaciones MACC.
- Ocupación de Flash: 47.17 KiB (de 512 KiB presentes).
- Ocupación de RAM: 1.88 KiB (de 96 KiB presentes).

En la Figura 4.1 se puede ver una gráfica del uso de memoria por cada capa de la red. En concreto, en esta figura se está mostrando, a la derecha, un detalle de la primera capa donde tal y como se puede observar se utilizan 8632 operaciones MACC con una ocupación de 1536 B.

Antes de comenzar con las pruebas, los movimientos entrenados con la red de ST son:

- *Parado*: La persona puede estar sentada, de pie sin moverse o de pie realizando pequeños movimientos con los brazos. Algunos datos han sido capturados con el dispositivo situado encima de un escritorio sin movimiento.
- *Caminando*: La persona está caminando en linea recta con algunos cambios en la dirección de forma ocasional.
- *Corriendo*: La persona está corriendo.

Al realizar pruebas con dicha red surgieron algunos inconvenientes como, por ejemplo, el hecho de que había que exagerar los movimientos, moviendo los brazos de una forma no acorde a los movimientos que se trataban de inferir, con objeto de detectar los diferentes estados. Por ejemplo,

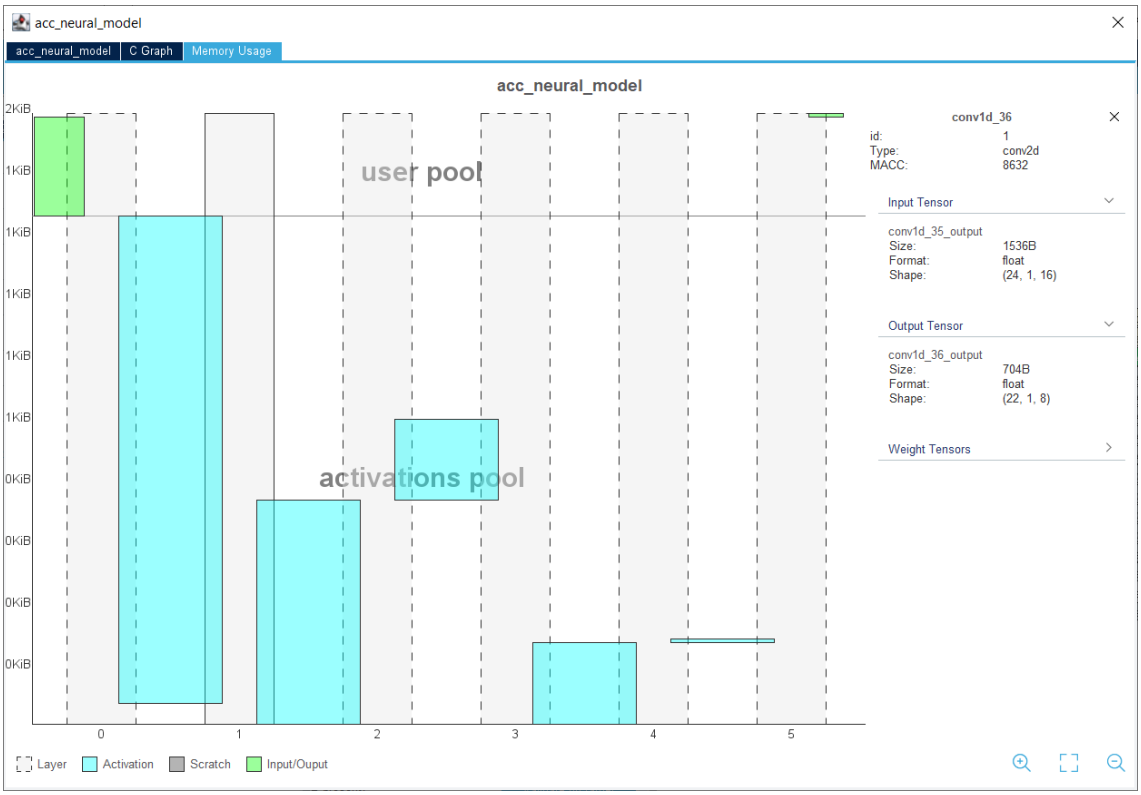

Figura 4.1 Gráfica del uso de memoria por cada capa de la red neuronal de ST.

la gran mayoría de las veces la red detectaba como natural el estado *caminando*. Únicamente al dejar la placa estable sobre la mesa se clasificaba como *parado*. Por último, para detectar el estado *corriendo* había que realizar movimientos muy rápidos con los brazos y no los propios de cuando se realiza esta actividad.

Después de analizar los resultados, se llegó a la conclusión de que dicho comportamiento podía estar motivado principalmente por tres factores:

1. Los datos utilizados para el entrenamiento estaban capturados con un acelerómetro distinto (LSM6DSL) y con un fondo de escala diferente (4g).
2. El acelerómetro de la placa X-NUCLEO-IKS01A1 estaba configurado a su máximo ODR (952 Hz), mientras que el acelerómetro LSM6DSL estaba configurado para tomar muestras con una frecuencia de 26 Hz.
3. Los datos del dataset habían sido tomados con el sensor colocado en diferentes posiciones (sobre el brazo o dentro del bolsillo) y no acoplado al reverso de la mano, tal y como se estaban realizando las pruebas.

Así pues, se proponen una serie de cambios sobre esta red para mejorar su comportamiento así como para introducir un nuevo movimiento, "*saltando*". Dichos cambios serán tratados en la siguiente sección.

4.1.2 Adaptación de la red

En este apartado se van a presentar los cambios realizados a la red neuronal de ST.

Para realizar estas modificaciones, se ha partido del código original ubicado en [20], y previamente comentado en la sección 2.5, y se ha creado un nuevo *notebook de Python*. El resultado se encuentra en:

Código 4.1 Notebook de Python con las modificaciones a la red neuronal de ST.

```
https://colab.research.google.com/drive/1C2LRmMLoVWJcEeUxh4Wok9m-
    O8x4YEUm?usp=sharing
```

Y puede ser consultado por todo aquel interesado. A continuación, se presentan de forma estructurada todos los cambios realizados.

Cambios en el dataset

En primer lugar, se descargó el dataset utilizado en la red del repositorio de Github de STMicroelectronics:

Código 4.2 Dataset de la red neuronal de ST.

```
https://github.com/STMicroelectronics/stm32ai/raw/master/AI_resources/
    HAR/dataset.zip
```

El dataset está organizado en tres carpetas: *stationary*, *waking* y *running*. Dentro de cada una de ellas, existen numerosos archivos en formato *.csv* y, en el interior de cada uno de éstos, están los datos *puros* capturados directamente del acelerómetro. Cada fichero contiene exactamente 1000 muestras. En la Figura 4.2 se puede observar un fragmento de uno de estos archivos.

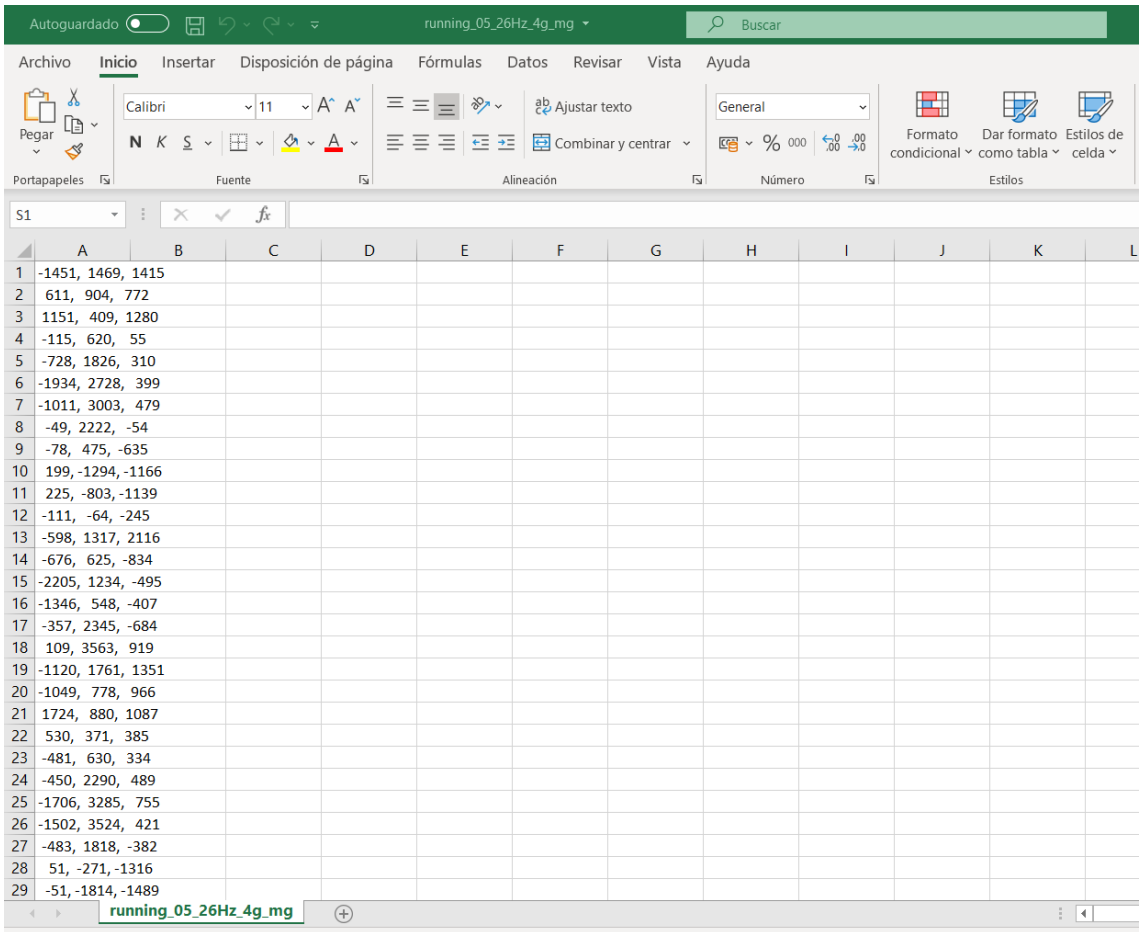

Figura 4.2 Fragmento del fichero *running_05_26Hz_4g_mg.csv*.

Representando algunos de ellos, podemos ver las gráficas de las Figuras 4.3, 4.4, 4.5 y 4.6.

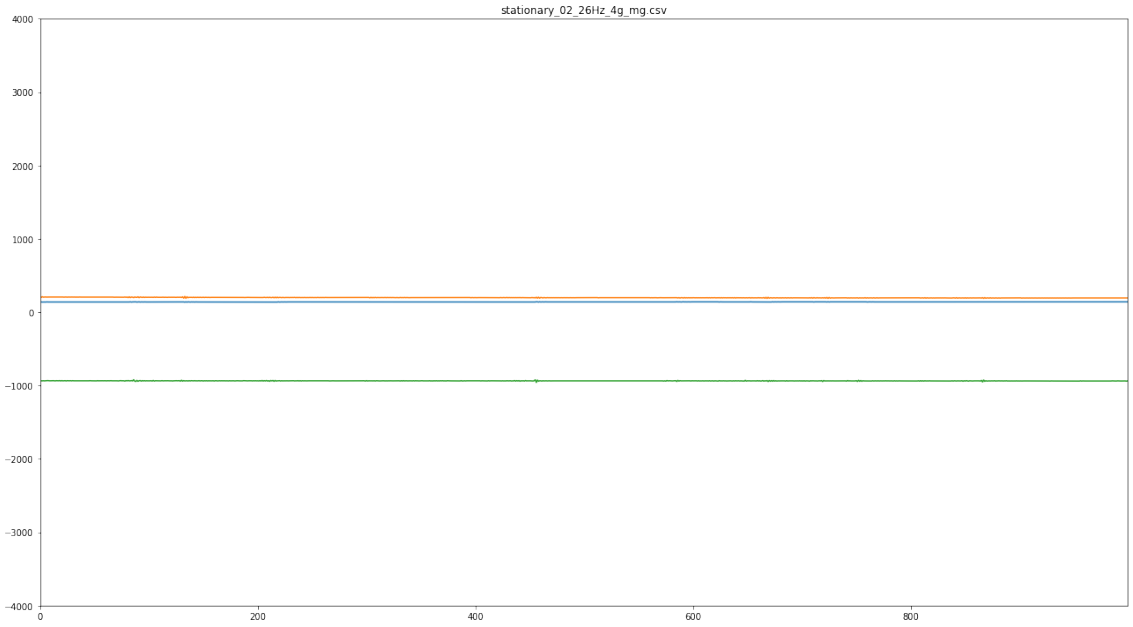

Figura 4.3 Gráfica del estado *estacionario*.

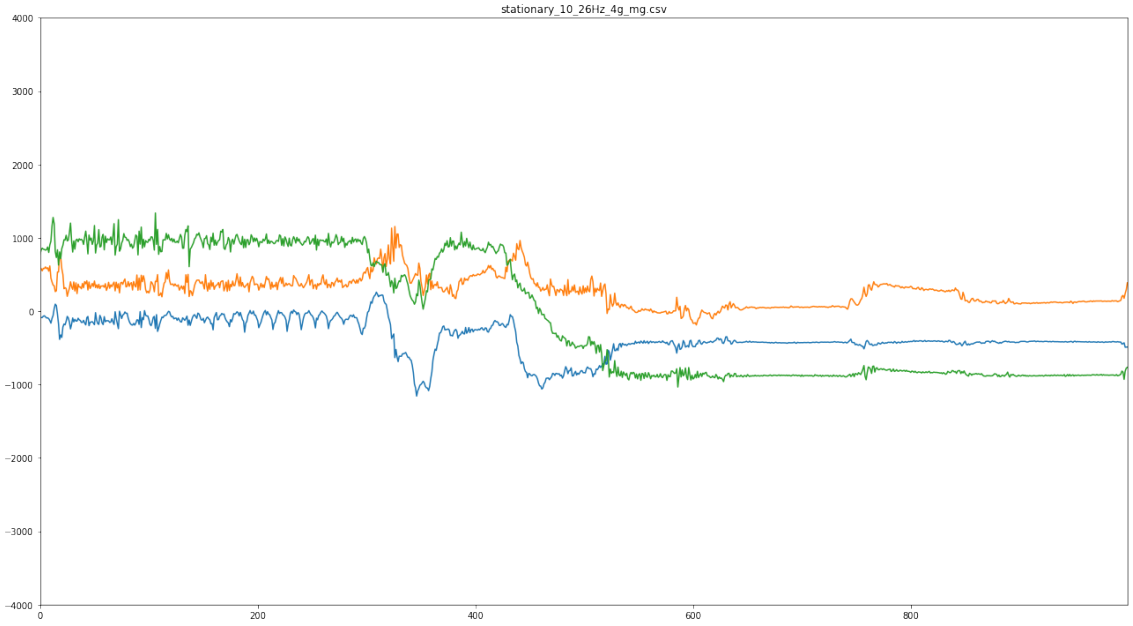

Figura 4.4 Gráfica del estado *estacionario* con pequeños movimientos de brazos en el sitio.

La placa utilizada para tomar estos datos ha sido la *STM32L4 Discovery kit IoT node (B-L475E-IOT01A)* [4], la cual hace uso del acelerómetro *LSM6DSL*.

Según se especifica desde el propio dataset, al tomar estos datos, el acelerometro se configuró a una tasa de muestreo de 26 Hz y un rango de [-4000 mg; 4000 mg]. Dicho esto, cada fichero *.csv* contiene unos 38.47 segundos de captura de datos: para *estacionario* existen más de 23 minutos, para *caminando* más de 25 minutos y para *corriendo* cerca de 12 minutos. Los datos fueron recopilados con la placa en diferentes posiciones: en la mano, en la muñeca, en el brazo y en el bolsillo.

Como modificación a la red se propuso incluir el estado *saltando*. Para mantener la estructura

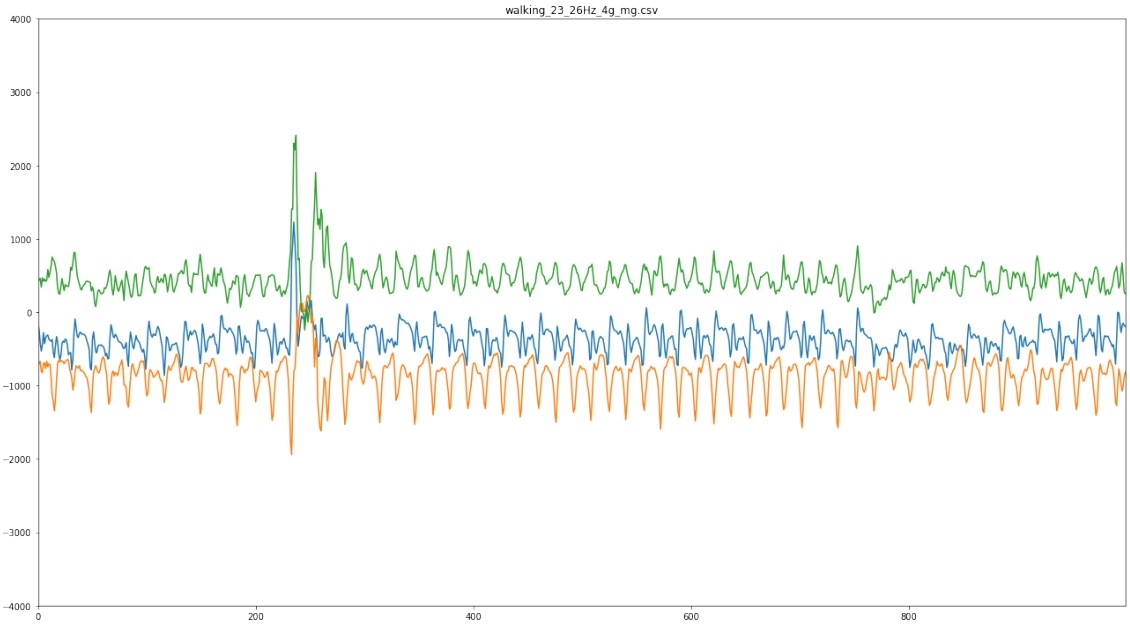

Figura 4.5 Gráfica del estado *caminando*.

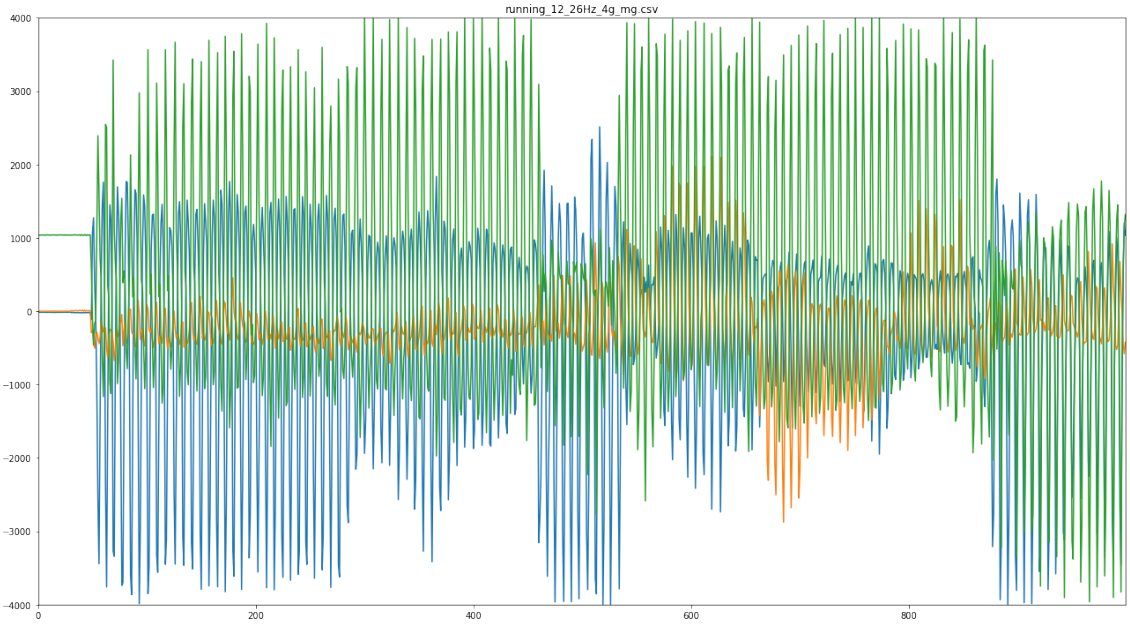

Figura 4.6 Gráfica del estado *corriendo*.

del dataset, se creó una nueva carpeta *jumping* y se añadieron 18 archivos *.csv*, cada uno de ellos con 1000 datos, tomados con una tasa de muestreo de 26Hz. Para ello, se creó un programa en *STM32CubeIDE* para capturar datos:

Código 4.3 Función *main* del programa de captura de datos.

```
int main(void)
{
  /* MCU Configuration
     ----------------------------------------------------------*/
```

```
/* Reset of all peripherals, Initializes the Flash interface and the
   Systick. */
HAL_Init();

/* Configure the system clock */
SystemClock_Config();


/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
MX_I2C1_Init();
MX_CRC_Init();
MX_TIM1_Init();

/* USER CODE BEGIN 2 */
RetargetInit(&huart2);

printf("\r\n Iniciando programa de captura de datos...\r\n");
/* USER CODE END 2 */

/* Infinite loop */

/* USER CODE BEGIN WHILE */
int mode=1;

  if(mode==1){

  HAL_TIM_Base_Start_IT(&htim1);

  while(1){

    if(flagInterrupcion==1){

    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

    Obtener_Ejes_LSM6DSO(&hi2c1,&accelerationFloat,FS_MID);

    accelerationAux.AXIS_X = (float) accelerationFloat.AXIS_X/4000.0f;
    accelerationAux.AXIS_Y = (float) accelerationFloat.AXIS_Y/4000.0f;
    accelerationAux.AXIS_Z = (float) accelerationFloat.AXIS_Z/4000.0f;

    printf("%.2f,%.2f,%.2f\r\n", accelerationAux.AXIS_X,
        accelerationAux.AXIS_Y, accelerationAux.AXIS_Z);

    flagInterrupcion=0;
    }
  }
```

```
    }

    if(mode==2){

      int mediciones=0;

      printf("Esperando 6 segundos para comenzar");

      HAL_Delay(6000);

      HAL_TIM_Base_Start_IT(&htim1);

      while(mediciones<1000){

        if(flagInterrupcion==1){

        flagInterrupcion=0;

        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

        Obtener_Ejes_LSM6DS0(&hi2c1,&accelerationFloat,FS_MID);

      accelerationAux.AXIS_X = (float) accelerationFloat.AXIS_X;
      accelerationAux.AXIS_Y = (float) accelerationFloat.AXIS_Y;
      accelerationAux.AXIS_Z = (float) accelerationFloat.AXIS_Z;

        printf("%.2f,%.2f,%.2f\r\n", accelerationAux.AXIS_X,
            accelerationAux.AXIS_Y, accelerationAux.AXIS_Z);

        mediciones++;
        }
      }
     /* USER CODE END WHILE */
    }
}
```

Este programa captura datos de dos modos diferentes: si la variable *mode* está configurada a 1, el programa captura datos de forma continua y muestra por pantalla los valores capturados en un rango de -1 a 1 mientras que, si está configurada a 2, capturará datos *puros* hasta completar 1000 mediciones. Nótese como el acelerómetro *LSM6DS0* ha sido configurado a una escala de 4g mediante el uso del argumento *FS_MID*.

Los datos que se van a tomar están capturados a una frecuencia de muestreo de 26Hz, para lo cual se hace uso de un timer. Lo relativo a la configuración de dicho timer fue comentado previamente en el apartado 3.4. Además, la posición de la placa en este caso será en la mano derecha.

Para visualizar y guardar los datos se ha hecho uso del emulador de terminal *Putty* [30]. Para configurar el emulador para guardar la sesión, se debe pichar sobre *Session* → *Logging*, seleccionar *Printable Output* y elegir un nombre y ubicación para el archivo.

Un ejemplo de un archivo *saltando* se puede visualizar en la gráfica 4.7

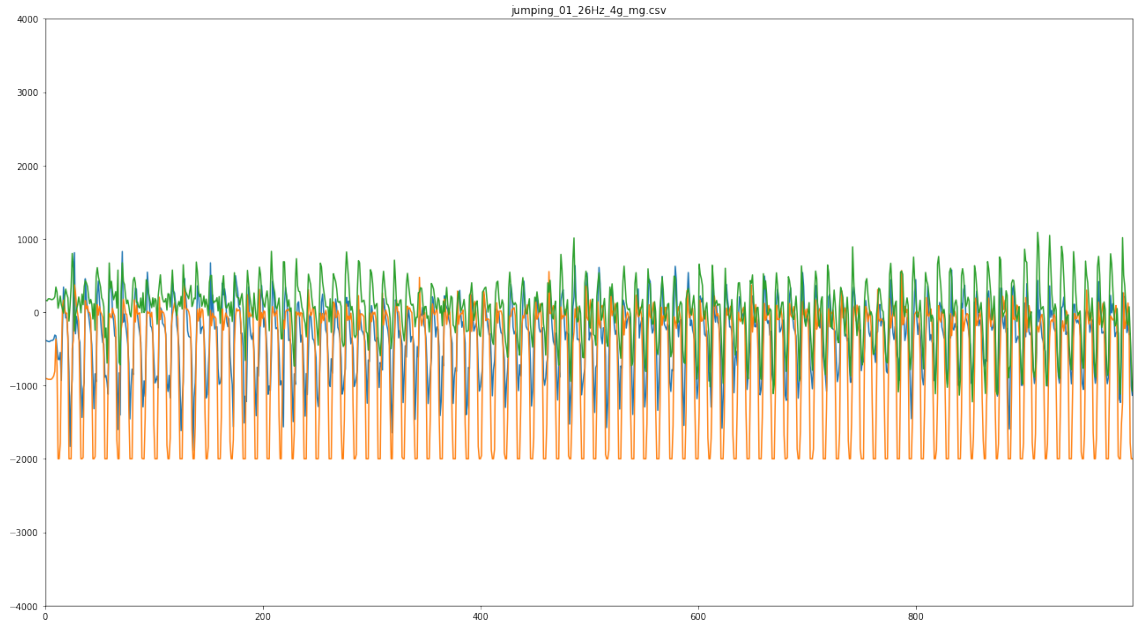

Figura 4.7 Gráfica del estado *saltando*.

Cambios en el código de la red

Con respecto al código, el primer paso es cargar la biblioteca de código abierto para aprendizaje automático *TensorFlow* [5]. Tal y como se avisa en [20], se debe usar la versión 1.15, ya que la versión 2.2 aún no está soportada en *X-CUBE-AI*. Para ello se ejecuta el siguiente comando:

Código 4.4 Biblioteca TensorFlow.

```
%tensorflow_version 1.x
import tensorflow as tf
```

En concreto, la versión utilizada para crear esta red neuronal fue la 1.15.2.

Después, se debe descargar el dataset que se va a utilizar, extraer los datos y cargarlos en memoria. El dataset a utilizar se encuentra subido en un *repositorio de github creado para este proyecto*[1] y es de acceso público para todo aquel que desee utilizarlo. Estos pasos se traducen en el siguiente código:

Código 4.5 Cargar el dataset en memoria.

```
#Descargar el dataset
!wget -nc https://github.com/juacuegut/TFG_Gesture_Recognition/raw/main/
    dataset.zip

#Descomprimirlo
!unzip -n dataset.zip

#Cargar los datos en memoria
import glob
import numpy as np
```

[1] https://github.com/juacuegut/TFG_Gesture_Recognition

```
# Load data into memory
labels = ['stationary', 'walking', 'running', 'jumping']
x = []
y = []
y_filenames = []
for i, label in enumerate(labels):
    filenames = glob.glob('dataset/' + label + '/*.csv')
    for filename in filenames:
        data = np.loadtxt(filename, delimiter=',')
        x.append(data)
        y.append(i)
        y_filenames.append(filename)

x2 = np.array(x).reshape(len(x), 1000, 3)
y2 = np.array(y)
```

Enventanado de los datos

Cada captura de datos se divide ahora en ventanas, cuyo tamaño será la dimensión de entrada de la red neuronal. En concreto, y tal y como se ha comentado anteriormente, se ha dividido el dataset en ventanas de 1 segundo (26 muestras a 26Hz). Se ha aplicado además un 50 % de overlap para aumentar la cantidad de puntos en nuestros datos para *training* y *test*.

Código 4.6 Enventanado de datos.

```
import numpy as np

def frame(x, frame_len, hop_len):

    assert(x.shape == (len(x), 3))
    assert(x.shape[0] >= frame_len)
    assert(hop_len >= 1)

    n_frames = 1 + (x.shape[0] - frame_len) // hop_len
    shape = (n_frames, frame_len, x.shape[1])
    strides = ((hop_len * x.strides[0],) + x.strides)
    return np.lib.stride_tricks.as_strided(x, shape=shape, strides=
        strides)

x3 = []
y3 = []
for i in range(x2.shape[0]):
    # frames = frame(x2[i], 26, 26) # no overlap
    frames = frame(x2[i], 26, 16) # 50% overlap
    x3.append(frames)
    y3.append(np.full(frames.shape[0], y[i]))

x3 = np.concatenate(x3)
y3 = np.concatenate(y3)
```

Preprocesamiento del dataset

Para esta red, se usará un simple escalado de los datos. Este escalado consiste en dividir los datos capturados entre el rango de la escala del sensor (4g) para obtener valores en un rango de entre -1 y 1:

Código 4.7 Escalado de los datos.

```
# Normalizar los datos de entrada entre [-1;1]
x4 = x3 / 4000
```

Después, se divide el dataset entre dos conjuntos: "entrenamiento" (*training*) y "pruebas" (*test*). Para ésto, los datos se mezclan y se reservan un 25 % de ellos para la evaluación del modelo y las pruebas del mismo. Con respecto al código, se hará uso en este caso de la librería *Scikit-learn* [3] y en concreto de la función *train_test_split* que lleva a cabo esta operación.

Código 4.8 División del dataset en *training* y *test*.

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x4, y3, test_size
    =0.25)

print("Trainning samples:", x_train.shape)
print("Testing samples:", x_test.shape)
```

El resultado son 5032 muestras para la fase de *training* y *1678* muestras para la fase de test.

Creación del modelo

Para la creación de la red neuronal se ha utilizado el siguiente código:

Código 4.9 Creación del modelo.

```
## Conv1D based model
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=16, kernel_size=4, activation='relu',
      input_shape=(26, 3)),
  tf.keras.layers.Conv1D(filters=8, kernel_size=4, activation='relu'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(4, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=30)
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

print("Test loss:", test_loss)
print("Test acc:", test_acc)
model.summary()
```

Para incluir el nuevo estado, se ha cambiado el tamaño del kernel, mediante la variable *kernel_size*, de 3 a 4. Además, se ha modificado la capa de salida de la red neuronal para contemplar 4 clases en lugar de 3, cambiando la capa en la que se aplica la función *softmax*, la cual asigna probabilidades decimales a cada clase sumando entre todas ellas 1.0.

Entrenamiento

A continuación, se presentan los *epochs* ejecutados durante la fase de "entrenamiento" utilizando las 5032 muestras seleccionadas para ello. Como se puede comprobar, durante esta fase se alcanzó un 98.93 % de precisión.

Código 4.10 Fase de entrenamiento.

```
Train on 5032 samples
Epoch 1/30
5032/5032 [==============================] - 1s 105us/sample - loss:
    1.0054 - acc: 0.5636
Epoch 2/30
5032/5032 [==============================] - 0s 90us/sample - loss:
    0.4769 - acc: 0.8168
Epoch 3/30
5032/5032 [==============================] - 0s 97us/sample - loss:
    0.2505 - acc: 0.9050
Epoch 4/30
5032/5032 [==============================] - 0s 85us/sample - loss:
    0.1666 - acc: 0.9412
Epoch 5/30
5032/5032 [==============================] - 0s 79us/sample - loss:
    0.1350 - acc: 0.9519
Epoch 6/30
5032/5032 [==============================] - 0s 91us/sample - loss:
    0.1131 - acc: 0.9595
Epoch 7/30
5032/5032 [==============================] - 0s 87us/sample - loss:
    0.1025 - acc: 0.9622
Epoch 8/30
5032/5032 [==============================] - 0s 86us/sample - loss:
    0.1019 - acc: 0.9652
Epoch 9/30
5032/5032 [==============================] - 0s 92us/sample - loss:
    0.0884 - acc: 0.9710
Epoch 10/30
5032/5032 [==============================] - 0s 88us/sample - loss:
    0.0973 - acc: 0.9660
Epoch 11/30
5032/5032 [==============================] - 0s 88us/sample - loss:
    0.0887 - acc: 0.9700
Epoch 12/30
5032/5032 [==============================] - 0s 90us/sample - loss:
    0.0842 - acc: 0.9694
Epoch 13/30
```

```
5032/5032 [==============================] - 0s 86us/sample - loss:
    0.0724 - acc: 0.9764
Epoch 14/30
5032/5032 [==============================] - 0s 88us/sample - loss:
    0.0674 - acc: 0.9756
Epoch 15/30
5032/5032 [==============================] - 0s 81us/sample - loss:
    0.0724 - acc: 0.9752
Epoch 16/30
5032/5032 [==============================] - 0s 88us/sample - loss:
    0.0627 - acc: 0.9783
Epoch 17/30
5032/5032 [==============================] - 0s 84us/sample - loss:
    0.0639 - acc: 0.9795
Epoch 18/30
5032/5032 [==============================] - 0s 86us/sample - loss:
    0.0586 - acc: 0.9785
Epoch 19/30
5032/5032 [==============================] - 0s 87us/sample - loss:
    0.0572 - acc: 0.9817
Epoch 20/30
5032/5032 [==============================] - 0s 88us/sample - loss:
    0.0582 - acc: 0.9799
Epoch 21/30
5032/5032 [==============================] - 0s 88us/sample - loss:
    0.0516 - acc: 0.9827
Epoch 22/30
5032/5032 [==============================] - 0s 90us/sample - loss:
    0.0580 - acc: 0.9791
Epoch 23/30
5032/5032 [==============================] - 0s 85us/sample - loss:
    0.0533 - acc: 0.9815
Epoch 24/30
5032/5032 [==============================] - 0s 86us/sample - loss:
    0.0474 - acc: 0.9849
Epoch 25/30
5032/5032 [==============================] - 0s 84us/sample - loss:
    0.0491 - acc: 0.9839
Epoch 26/30
5032/5032 [==============================] - 0s 83us/sample - loss:
    0.0508 - acc: 0.9819
Epoch 27/30
5032/5032 [==============================] - 0s 91us/sample - loss:
    0.0420 - acc: 0.9855
Epoch 28/30
5032/5032 [==============================] - 0s 88us/sample - loss:
    0.0457 - acc: 0.9849
Epoch 29/30
5032/5032 [==============================] - 0s 88us/sample - loss:
    0.0537 - acc: 0.9821
```

```
Epoch 30/30
5032/5032 [==============================] - 0s 89us/sample - loss:
   0.0428 - acc: 0.9863
1678/1678 - 0s - loss: 0.0284 - acc: 0.9893
Test loss: 0.028437232520959657
Test acc: 0.98927295
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_12 (Conv1D)           (None, 23, 16)            208
_________________________________________________________________
conv1d_13 (Conv1D)           (None, 20, 8)             520
_________________________________________________________________
dropout_6 (Dropout)          (None, 20, 8)             0
_________________________________________________________________
flatten_6 (Flatten)          (None, 160)               0
_________________________________________________________________
dense_12 (Dense)             (None, 64)                10304
_________________________________________________________________
dense_13 (Dense)             (None, 4)                 260
=================================================================
Total params: 11,292
Trainable params: 11,292
Non-trainable params: 0
```

Una vez finalizado el entrenamiento, podemos construir la matriz de confusión, la cual permitirá evaluar el modelo obteniendo de forma visual el desempeño del algoritmo empleado. Para ello, utilizamos el siguiente código:

Código 4.11 Obtención de la matriz de confusión.

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

y_pred = model.predict_classes(x_test)
confusion_matrix = tf.math.confusion_matrix(y_test, y_pred)
sess = tf.compat.v1.Session()

plt.figure()
sns.heatmap(sess.run(confusion_matrix),
            annot=True,
            xticklabels=labels,
            yticklabels=labels,
            cmap=plt.cm.Blues,
            fmt='d', cbar=False)

plt.tight_layout()
plt.ylabel('True label')
```

```
plt.xlabel('Predicted label')
plt.show()
```

El resultado de la ejecución de estos comandos se puede observar en la Figura 4.8:

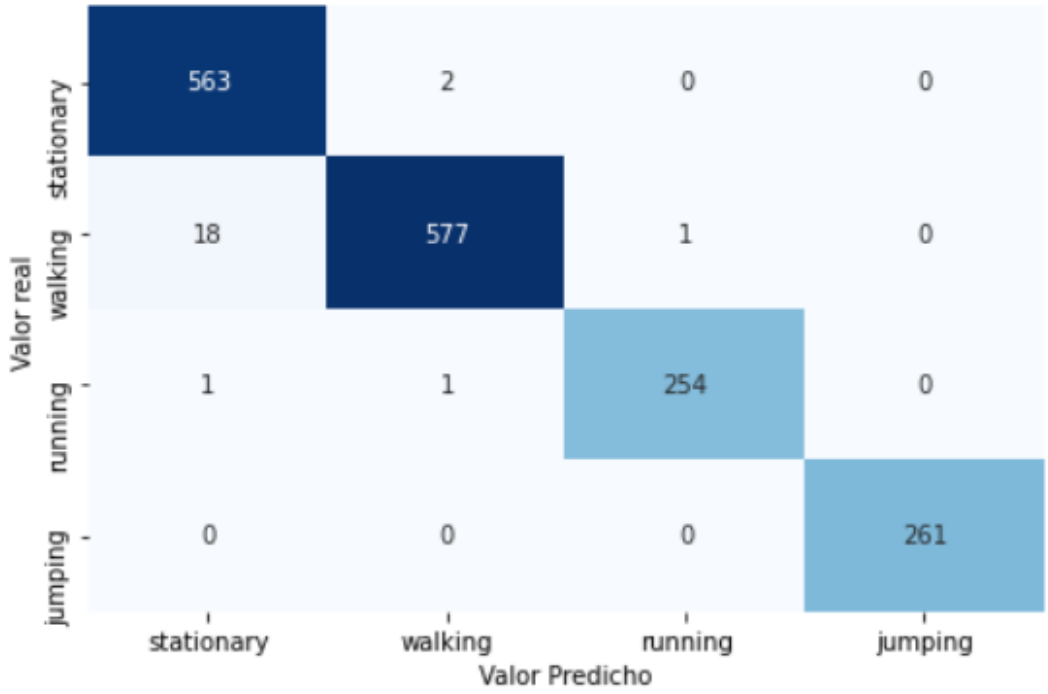

Figura 4.8 Matriz de confusión.

Guardado del modelo

Para poder utilizar el modelo en *STM32CubeMX*, se debe guardar en formato *.h5* y descargarlo del notebook. Esto se puede conseguir con el siguiente código:

Código 4.12 Guardado y descarga del modelo.

```
# Guardar el modelo en formato ".h5"
model.save('model_jumping_inc.h5')
# Descargar modelo
from google.colab import files
files.download('model_jumping_inc.h5')
```

En este caso, al nuevo modelo se le ha puesto del nombre de *model_jumping_inc.h5*, pero puede ser modificado por el usuario.

4.1.3 Mediciones

Adicionalmente, y aprovechando las capacidades de la placa de desarrollo, se ha programado un timer para medir el tiempo de inferencia de la red operando en el microcontrolador.

En concreto, se ha programado el timer *TIM3*, el cual no entra en conflicto con ningún otro componente del sistema (lo cual se debe verificar, tal y como se comenta en la sección 3.4).

Dado que el reloj principal está operando a 84 Mhz, si se configura el preescalador a 84 se puede obtener 1 Mhz en la frecuencia del timer. Nótese que, tal y como se ha comentado en la sección 3.4,

bajo estos valores solo se podrán fijar períodos de hasta un máximo de 65,535 ms. Sin embargo, puesto que se pretenden medir tiempos muy pequeños, debido a que la inferencia se realiza de forma muy rápida, este valor será más que suficiente. El resultado de la configuración del timer se muestra en la Figura 4.9.

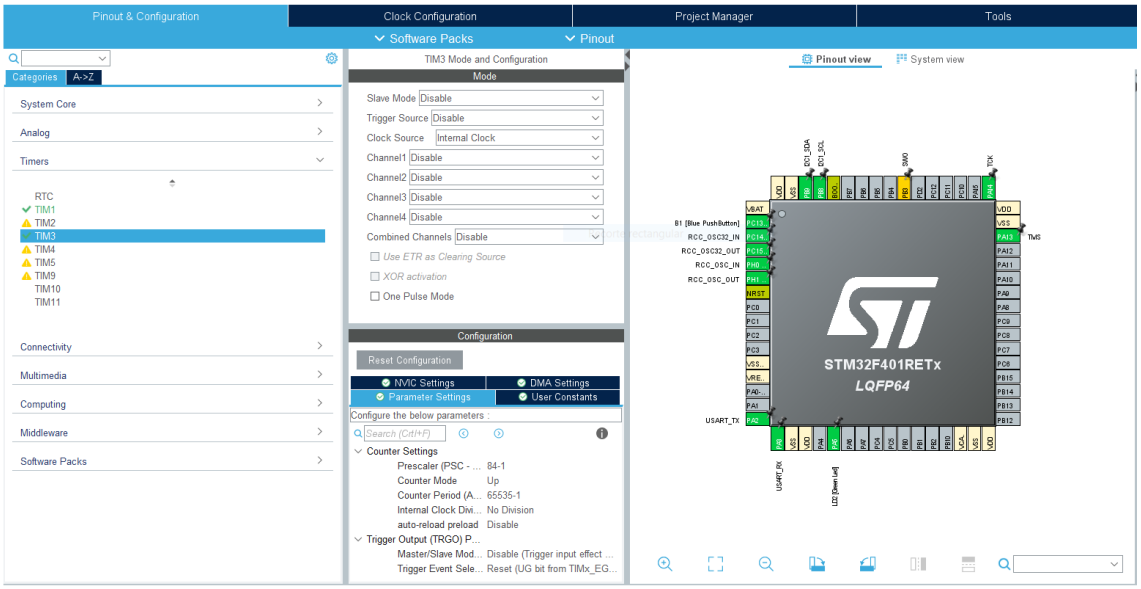

Figura 4.9 Configuración del timer para medir el tiempo de inferencia, *TIM3*.

Para utilizar el timer, se añadió el siguiente código:

Código 4.13 Código de inclusión del timer *TIM3*.

```
while (1)
 {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  if(flagInterrupcion==1){

     flagInterrupcion=0;

     HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

     Obtener_Ejes_LSM6DS0(&hi2c1,&accelerationFloat,FS_LOW);

         aiInData[write_index + 0] = (float) accelerationFloat.AXIS_X/
             4000.0f;
         aiInData[write_index + 1] = (float) accelerationFloat.AXIS_Y/
             4000.0f;
         aiInData[write_index + 2] = (float) accelerationFloat.AXIS_Z/
             4000.0f;

         write_index += 3;
```

```
        if (write_index == AI_ACC_NEURAL_MODEL_IN_1_SIZE) {
        write_index = 0;

        printf("Realizando inferencia...\r\n");
        HAL_TIM_Base_Start(&htim3);
        timer_value = __HAL_TIM_GET_COUNTER(&htim3);
        AI_Run(aiInData, aiOutData);
        timer_value = __HAL_TIM_GET_COUNTER(&htim3) - timer_value;

        printf("Tiempo realizando inferencia: %u us\r\n", timer_value)
            ;

         /* Output results */
        for (uint32_t i = 0; i < AI_ACC_NEURAL_MODEL_OUT_1_SIZE; i++)
            {
          printf("%8.6f ", aiOutData[i]);
        }

        uint32_t class = argmax(aiOutData,
            AI_ACC_NEURAL_MODEL_OUT_1_SIZE);
        printf(": %d - %s\r\n", (int) class, activities[class]);

    printf("\r\n");
    printf("Esperando 4 segundos para nuevo movimiento...\r\n");
    HAL_TIM_Base_Stop_IT(&htim1);
    HAL_Delay(4000);
    HAL_TIM_Base_Start_IT(&htim1);
    printf("Haz un movimiento!\r\n");

      }
  }
 }
```

Como se puede ver, antes de llamar a la función encargada de realizar la inferencia, *AI_Run*, se inicializa el timer *TIM3* mediante la llamada anteriormente explicada *HAL_TIM_Base_Start(htim3)*. Después, se llama a la función *__HAL_TIM_GET_COUNTER(htim3)* para obtener el valor del contador (del registro CNT) en el momento actual y se almacena en la variable *timer_value*. A continuación, se llama a la función para realizar la inferencia y por último, simplemente se restan los tiempos para obtener el período transcurrido. Los resultados han sido incluidos en la demo del sistema completo y se mostrarán en el siguiente apartado.

4.2 Demo del sistema completo

Una vez realizados los cambios, la red resultado presenta las siguientes características:

- Complejidad: 25828 operaciones MACC.
- Ocupación de Flash: 44.11 KiB (de 512 KiB presentes).
- Ocupación de RAM: 1.82 KiB (de 96 KiB presentes).

En la Figura 4.10 se puede ver una gráfica del uso de memoria por cada capa de la red. En concreto, en esta figura se está mostrando, a la derecha, un detalle de la primera capa donde tal y como puede observar se utilizan 10408 operaciones MACC con una ocupación de 1472 B.

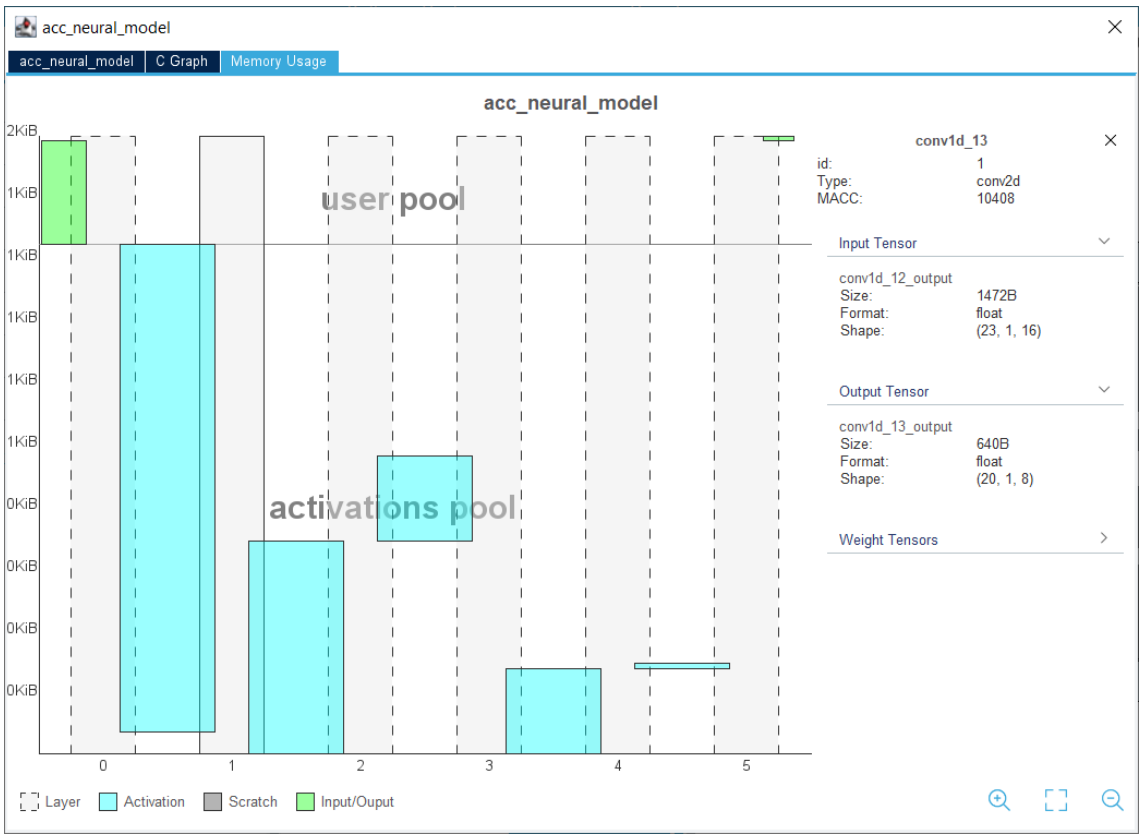

Figura 4.10 Gráfica del uso de memoria por cada capa de la red neuronal resultado.

Para realizar las pruebas y adquirir los datos, se utilizó un guante y se adaptó la placa de desarrollo para ser acoplada en éste. Las pruebas del sistema completo se realizaron en la mano derecha. Las Figuras 4.11, 4.12, 4.13 y 4.14 contienen imágenes del sistema completo.

Dada la naturaleza de este trabajo, no se puede plasmar el funcionamiento del sistema. Sin embargo, se adjunta una captura de las inferencias realizadas para diferentes movimientos (Figura 4.15).

Como se puede comprobar, el tiempo de inferencia es alrededor de 3,95 milisegundos. Además, también se muestra por pantalla la clasificación de cada movimiento en tanto por uno, siendo la mayor de éstas el resultado de la inferencia.

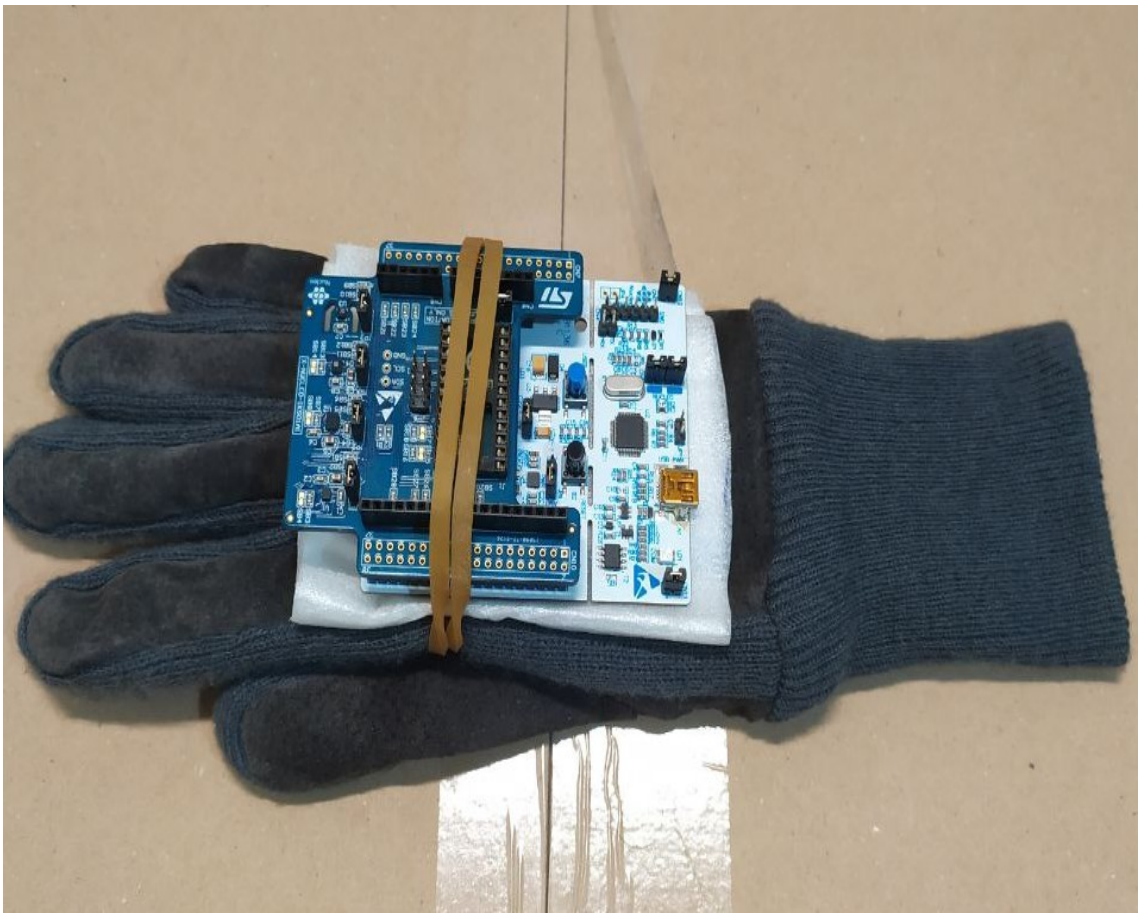

Figura 4.11 Imagen del sistema utilizado para adquirir datos y realizar pruebas.

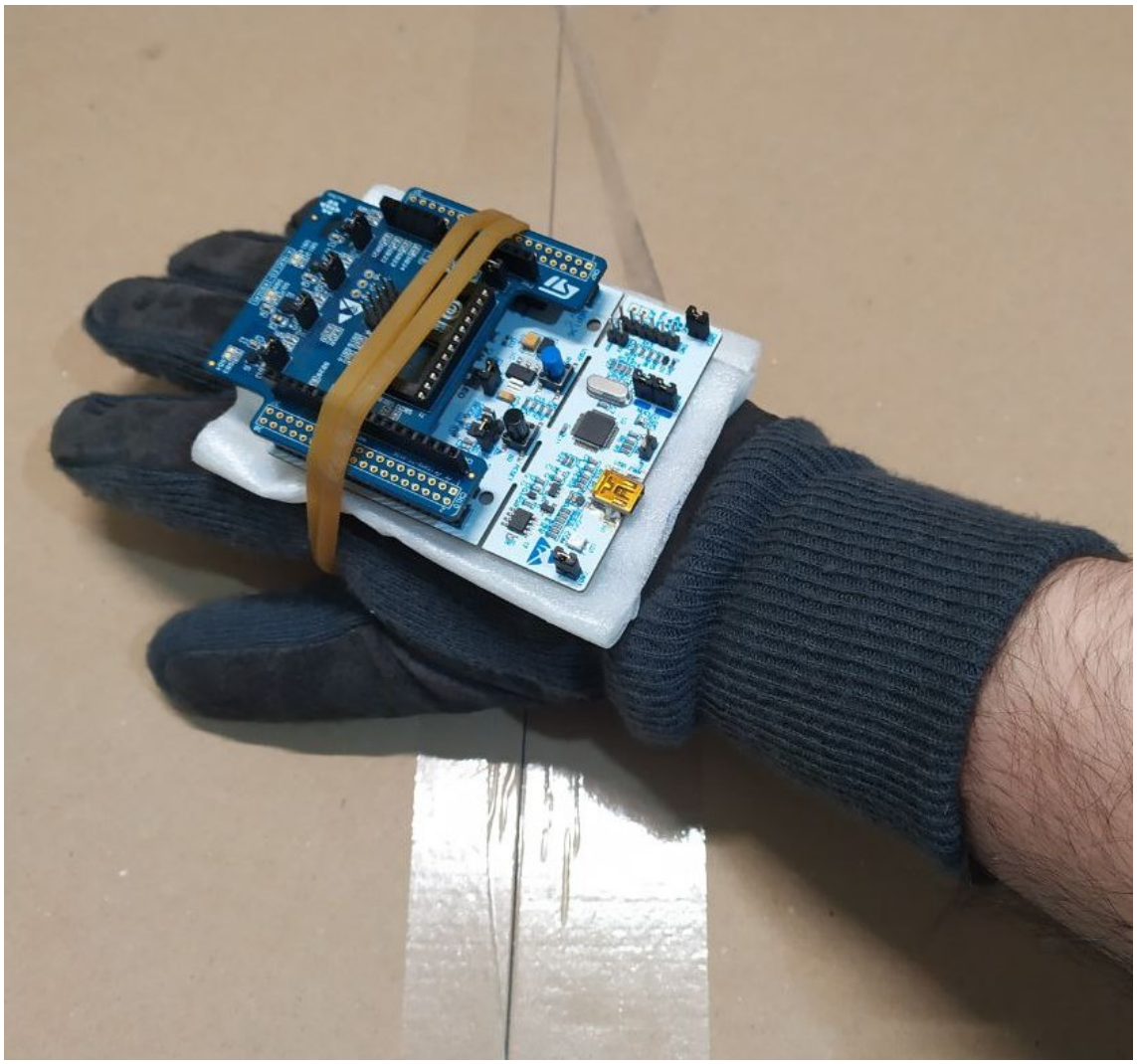

Figura 4.12 Imagen del sistema colocado sobre la mano derecha..

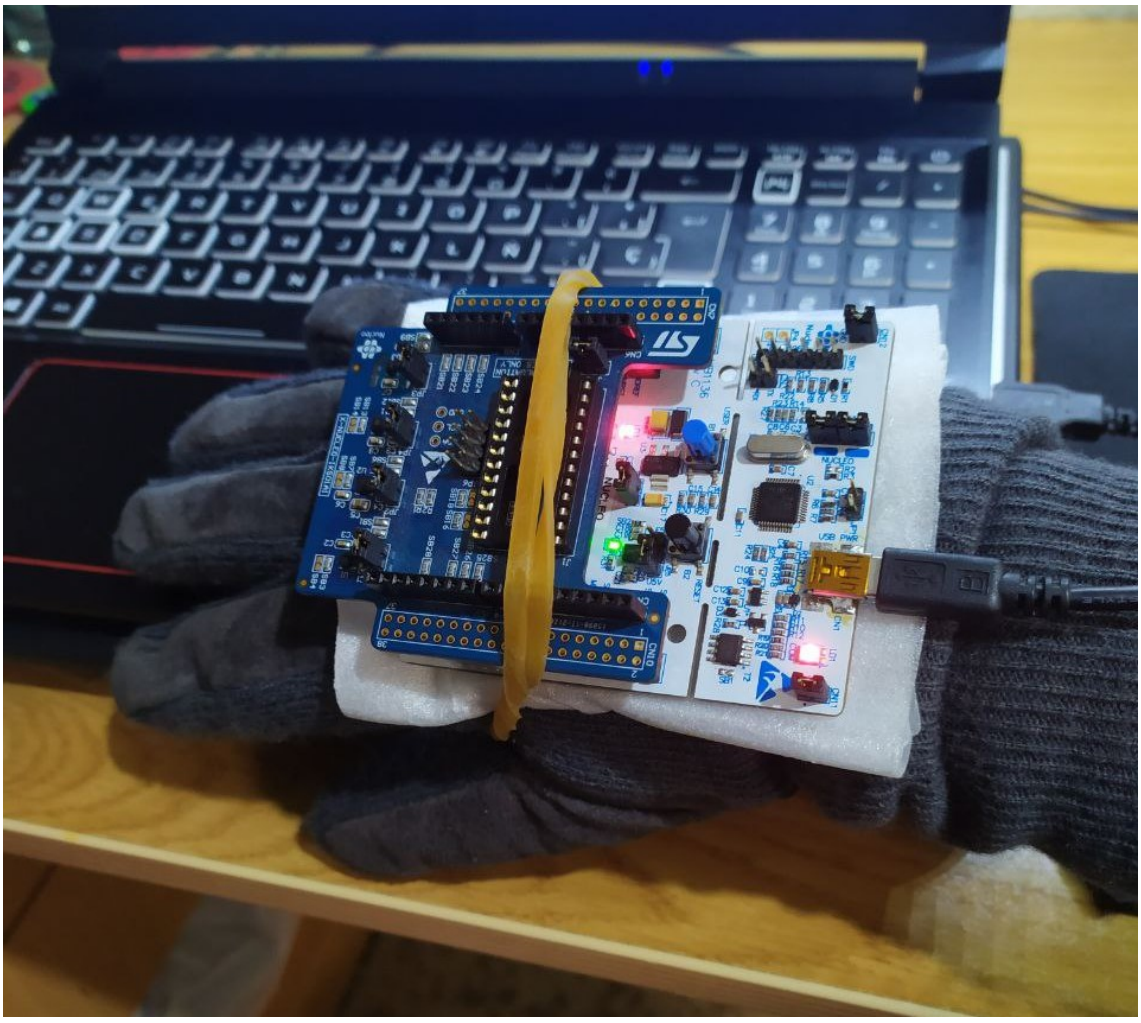

Figura 4.13 Detalle del sistema completo.

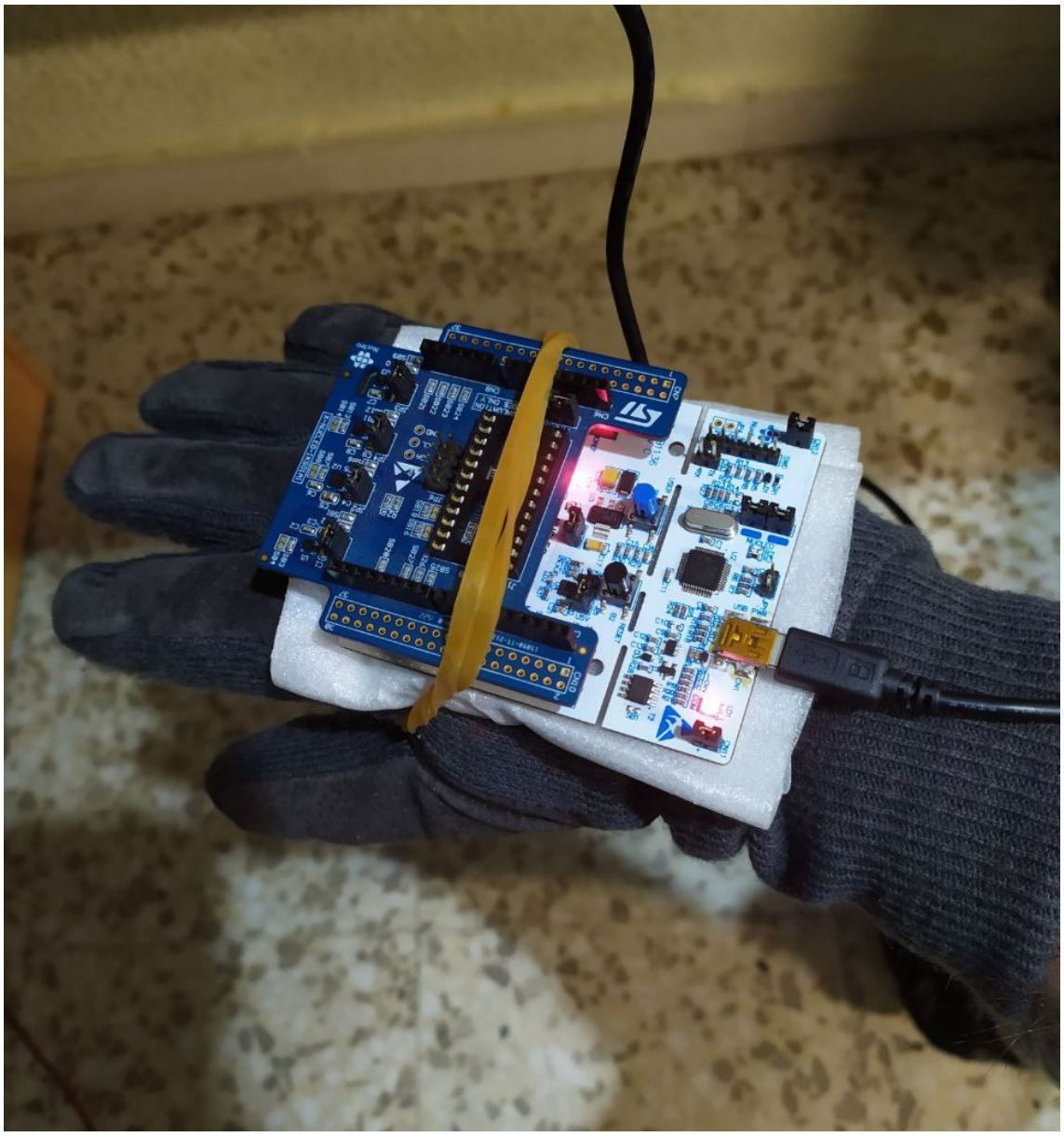

Figura 4.14 Imagen del sistema colocado a la mano derecha, listo para su uso.

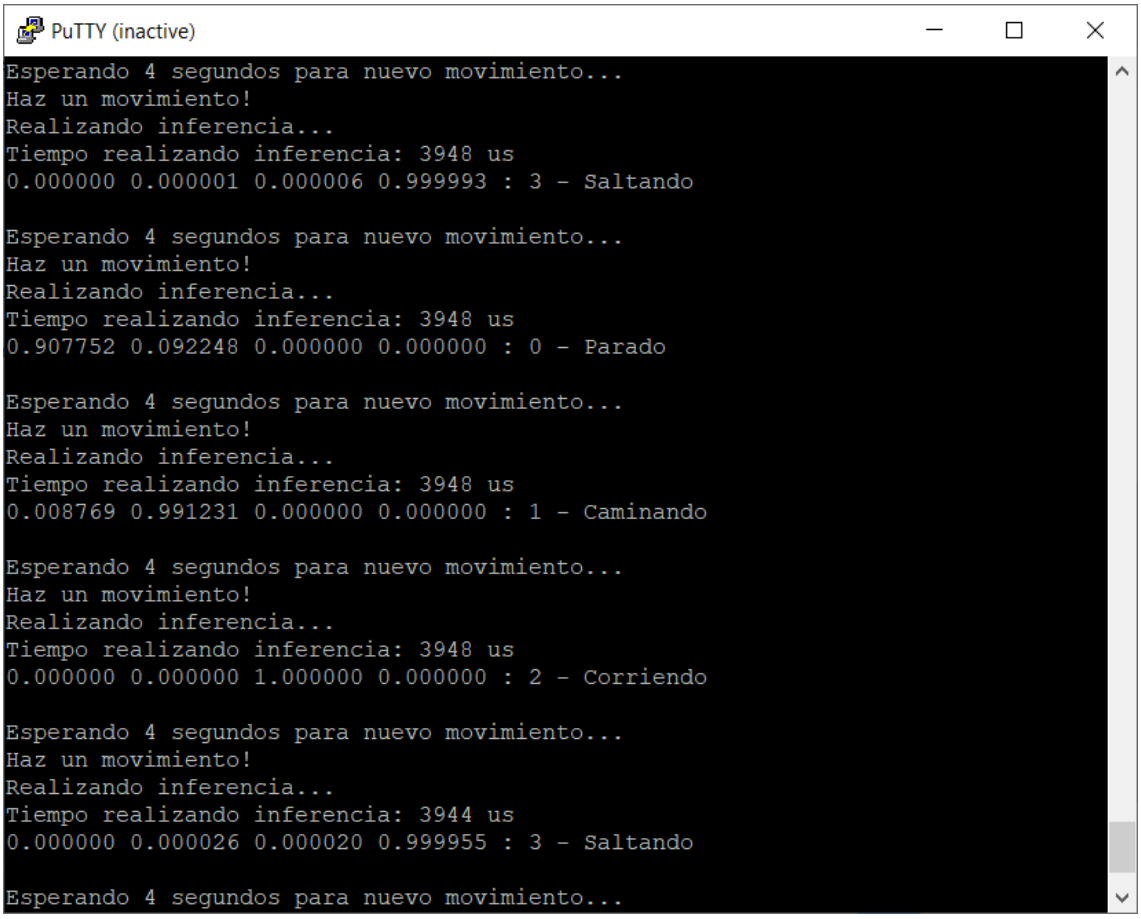

Figura 4.15 Captura del sistema completo realizando inferencias.

5 Conclusiones y desarrollos futuros

El futuro mostrará los resultados y juzgará a cada uno de acuerdo a sus logros.

- Nikola Tesla -

En este capítulo se presentarán las conclusiones obtenidas como fruto del desarrollo de este proyecto, así como los estudios realizados y las guías para una futura evolución del mismo.

5.1 Conclusión de objetivos

En este proyecto se ha demostrado como resulta posible introducir una red neuronal en un dispositivo embebido utilizando las herramientas adecuadas. Además, durante la realización de este proyecto, se han enfrentado problemáticas reales propias de un entorno laboral como es el hecho de tener que dar uso a un dispositivo obsoleto y que carece de soporte, ya sea por tiempo, por el ahorro de dinero que ésto supone o porque el dispositivo es nuevo y se debe crear un driver desde cero. Adicionalmente, se ha diseñado un framework que potencia las herramientas existentes y que cualquier interesado puede utilizar como base sobre la cual desarrollar cualquier tipo de aplicación que requiera del uso de una o varias redes neuronales. Por último, se introducido en el microcontrolador una red neuronal modificada y entrenada con un dataset propio haciendo uso del framework para exponer la versatilidad, escalabilidad y correcto funcionamiento del mismo. Por todo lo expuesto anteriormente, se puede afirmar que los objetivos de este tfg han sido cumplidos, abriéndo además con él la puerta a un desarrollo más ágil de aplicaciones basadas en aprendizaje automatico sobre dispositivos embebidos. No obstante, el proyecto admite mejoras y desarrollos futuros, los cuales serán comentados a continuación.

5.2 Desarrollos futuros

El *framework* desarrollado, aunque funcional, permite una serie de mejoras que sin duda incrementarían su valor potencial haciéndo más sencillo su uso. Las guias de desarrollo de este proyecto son variadas.

En primer lugar, para hacer uso del driver del acelerómetro, el usuario debe copiar manualmente los ficheros *LSM6DS0_CUSTOM.h* y *LSM6DS0_CUSTOM.c*. Adicionalmente, el usuario debe introducir código a mano en el fichero *main.c*, con el fin de poder incluir una o varias redes neuronales en el dispositivo embebido deseado. Aunque éstas son las recomendaciones de *STMicroelectronics*, como bien se presenta en la guía publicada po el mismo en Noviembre del año pasado [23], sería de gran interés generar dicho código de forma dinámica minimizando así la intervención del usuario.

Como bien es sabido, no es posible asignar un nombre dinámico a una variable en tiempo de ejecución, por lo que se exploraron diferentes posibilidades. Para comenzar, sería de interés crear un

template utilizando el motor *Apache FreeMarker™* [7], una biblioteca de Java para la generación de texto basada en plantillas y datos cambiantes, haciéndo uso del lenguaje FTL (FreeMarker Template Language), con el cual se pueden crear archivos (*.flt*) que pueden ser usados por *STM32CubeMX* para generar código específico.

Más interesante todavía, sería hacer uso de la herramienta *STM32 Pack Creator* [22], una herramienta desarrollada por *STMicroelectronics* cuyo propósito, como su nombre indica, es la creación de paquetes de software. Un paquete de software es una colección de archivos que se envían en formato *.zip* (después renombrados a *.pack*) y que cumplen con las especificaciones Arm® CMSIS-Pack [14], las cuales definen una forma estandarizada de creación de componentes software. Dentro de un paquete se incluyen, entre otros:

- Código fuente, archivos de encabezado y bibliotecas de software
- Plantillas de documentación y código fuente
- Proyectos de ejemplo
- Archivo en formato *.pdsc* que describe el contenido del paquete y contexto de uso para los archivos suministrados en su interior

Con esta herramienta se podría crear un archivo *.pack* instalable desde *STM32CubeMX* y que permitiera al usuario añadir el driver a su proyecto, así como elegir entre varios ejemplos de uso de la red neuronal, con una o varias redes en función de las necesidades del mismo. Esto brindaría sin duda más funcionalidad para el usuario.

Como una idea de inclusión de dos redes neuronales coexistentes en el microcontrolador, y haciéndo uso del driver creado para el acelerómetro, sería posible el desarrollo de una aplicación para detectar caídas. Por ejemplo se podría añadir una red que detectara si existe o no caída y otra que detectase la naturaleza de la misma (si se ha producido de forma lateral, frontal o hacia atrás), lo cual brindaría la capacidad de una primera evaluación de los posibles daños causados por el impacto y su posterior acción (por ejemplo, llamar a una ambulancia) lo que resultaría de utilidad sobre todo en el caso de personas de avanzada edad.

Desde un punto de vista más técnico, resultaría de interés explorar los límites de este software como, por ejemplo, comprobar cuántas redes se pueden cargar en el microcontrolador o cómo afecta a la *performance* la inclusión de una red adicional hasta alcalzar el límite de tamaño. Asímismo, también resultaría de interés realizar un estudio acerca de la relación entre la cantidad de movimientos que puede reconocer una red y el tamaño que ésta ocupa en el microcontrolador.

Apéndice A
Configuración del entorno y posibles problemas

Si le hubiera preguntado a la gente qué querían, habrían dicho caballos más rápidos.

- Henry Ford -

En esta sección se detallarán los pasos a seguir para configurar el entorno y se añadirá una explicación de los archivos y funciones generadas por *STM32CubeMX*.Estos pasos implican configurar las comunicaciones entre la placa de desarrollo y la tarjeta de expansión, las comunicaciones entre la placa y el PC y la instalación y configuración del paquete de Inteligencia Artificial *X-CUBE-AI*. Además, se incluye una sección de posibles problemas que se pueden encontrar al hacer uso de este software.

A.1 Configuración de STM32CubeMX

El primer paso, después de la instalación del programa, es empezar un nuevo proyecto (Figura A.1). En el caso concreto de este trabajo de fin de grado, se usará la placa *Núcleo-64 STM32F401RE* presentada en la sección 2.6.1, por lo que se puede elegir la opción *ACCESS TO BOARD SELECTOR*. Ésto abrirá una nueva pestaña donde, una vez encontrada la placa, aparecerá información sobre la misma tal y como se muestra en la Figura A.2.

Al hacer click sobre *START PROJECT*, se abrirá una ventana preguntando al usuario si desea inicializar los pines (Figura A.3).

Después, el usuario se encuentra en la ventana de configuración del microcontrolador (Figura A.4).

A.1.1 I2C y USART

En primer lugar, después de seleccionar la placa de desarrollo *STM32F401RE* en *STM32CubeMX*, algunos pines aparecen configurados por defecto. Para este proyecto, se deben marcar además los pines *PB8* y *PB9* como *I2C1_SCL* y *I2C1_SDA*, respectivamente.

Una vez realizado este paso, en la pestaña **Conectivity** se debe marcar I2C1 y configurarlo tal y como se ve en la Figura A.5.

En la parte derecha de la Figura A.5 se pueden ver el conjunto de pines configurados.

Cabe prestar especial atención a los pines *PA2* y *PA3* que aparecen configurados automáticamente como *USART_TX* y *USART_RX*, los cuales se utilizarán en la configuración del puerto serie implementando el protocolo **USART**.

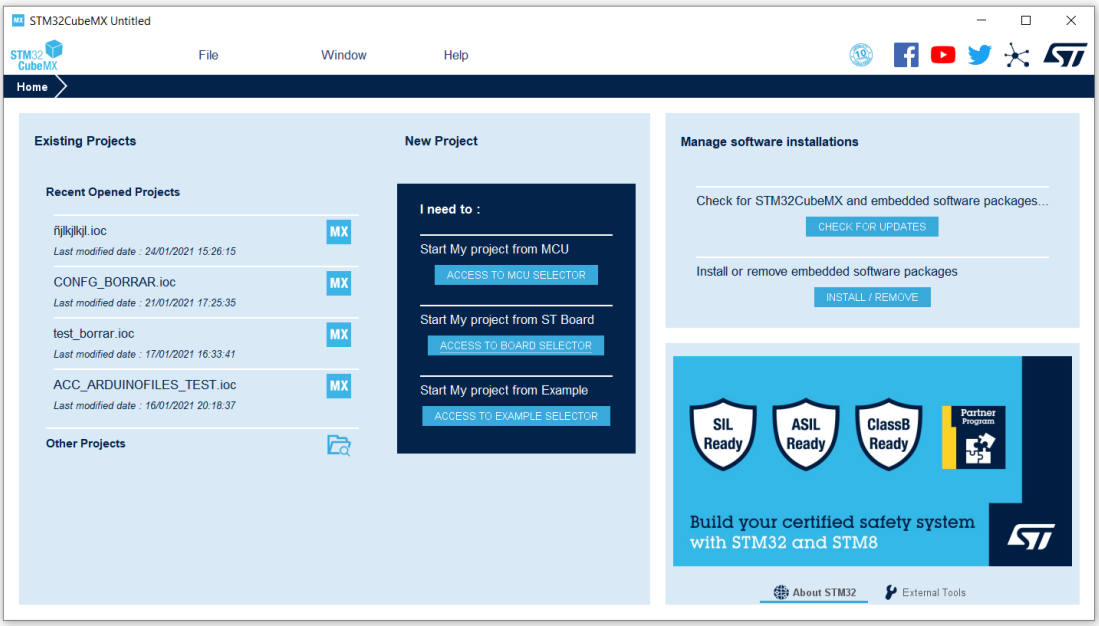

Figura A.1 Página principal de STM32CubeMX.

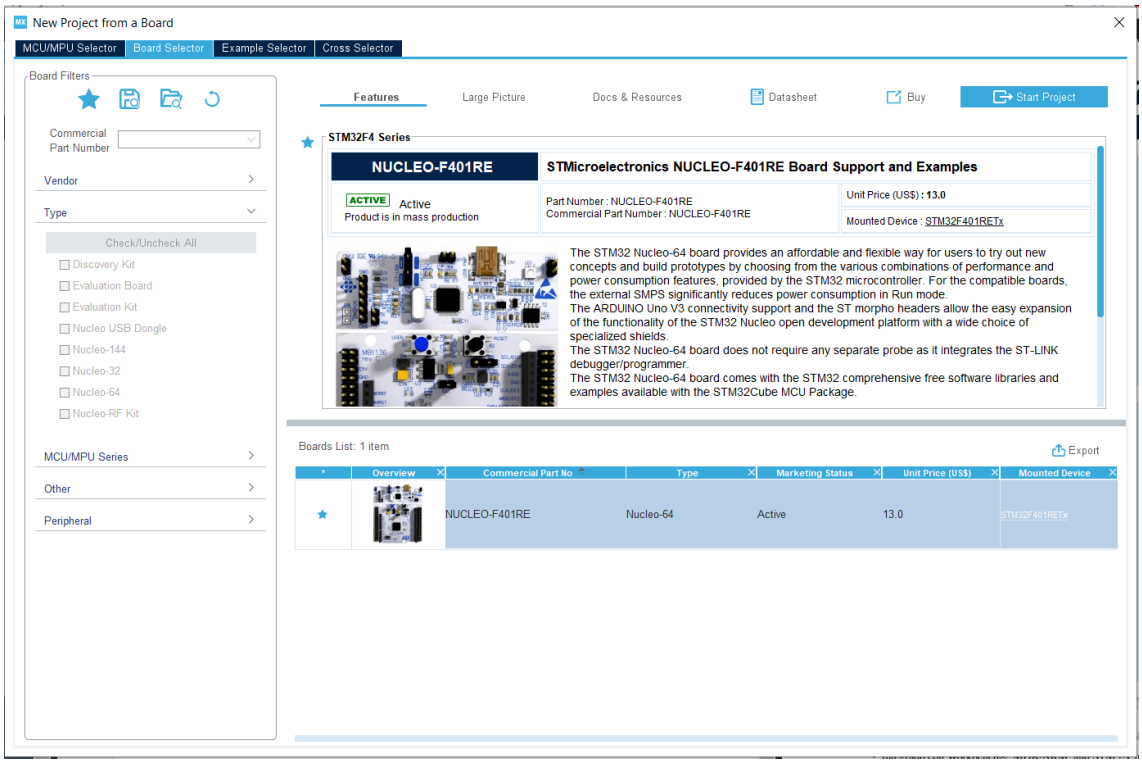

Figura A.2 Nuevo proyecto y selección de la tarjeta Núcleo-F401RE en STM32CubeMX.

Todos estos pines deben ser consultados en el datasheet de la placa de desarrollo [24].

Una vez realizadas las configuraciones, elegido un nombre apropiado para el proyecto y seleccionado como IDE el software *STM32CubeIDE*, se debe clickar sobre *GENERATE CODE*, lo cual permitirá importar el proyecto al espacio de trabajo dentro de *STM32CubeIDE*.

Hecho esto, se crearán de forma automática los ficheros que contienen las configuraciones indicadas anteriormente en *STM32CubeMX*.

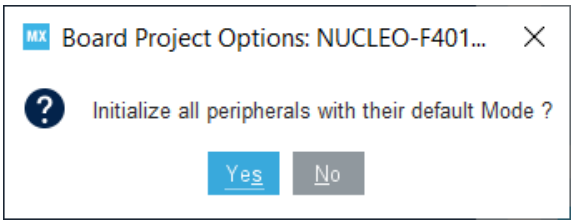

Figura A.3 Inicialización de los periféricos en modo por defecto en STM32CubeMX.

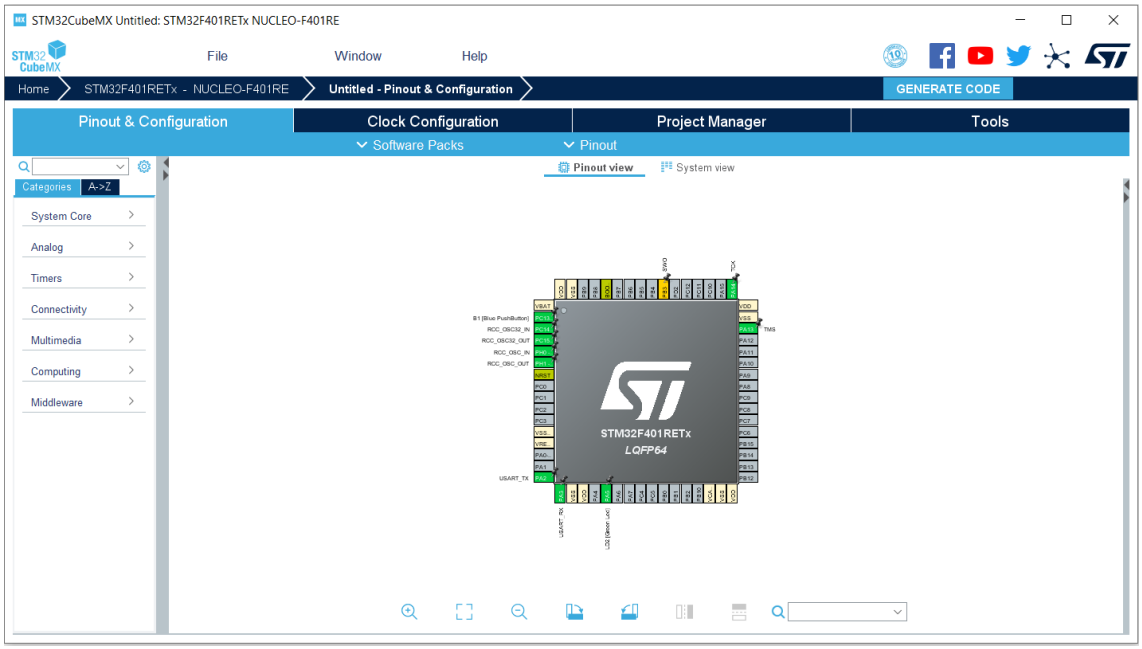

Figura A.4 Ventana de configuración del proyecto creado en STM32CubeMX.

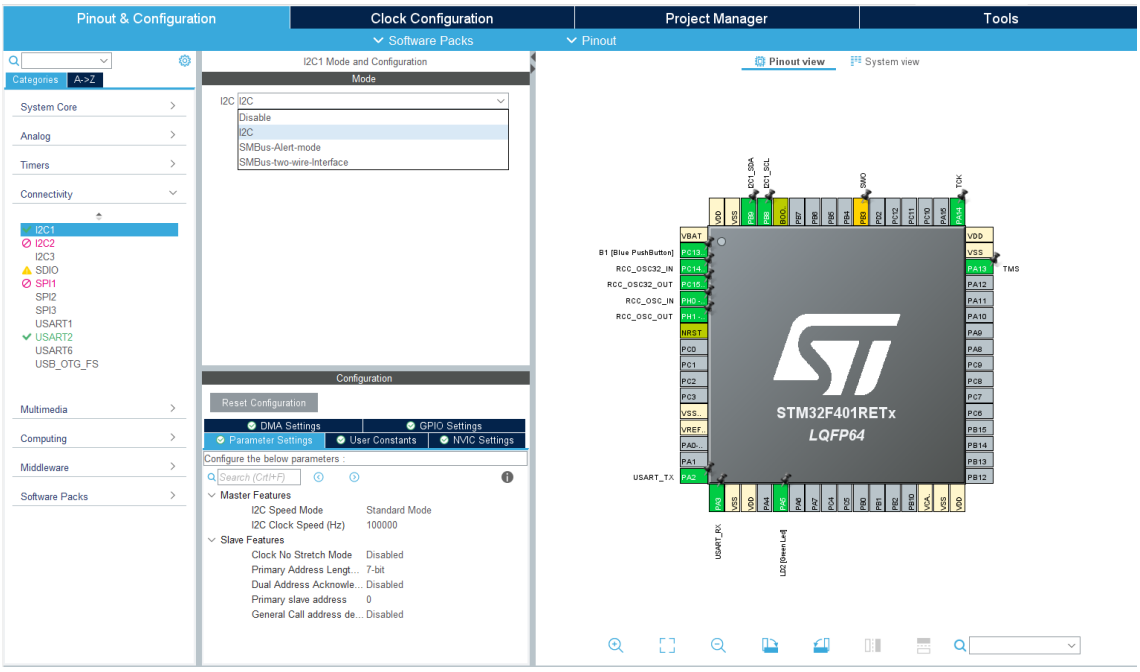

Figura A.5 Configuración del I^2C para el driver del acelerómetro LSM6DS0.

En concreto, el fichero *main.c* es el destinado a ser modificado por parte del usuario. Este fichero será generado cada vez que se realicen modificaciones sobre el archivo *".io"* asociado al proyecto por lo que el usuario deberá escribir dentro de las regiones encapsuladas para ello, es decir, aquéllas enmarcas entre *USER CODE BEGIN* y *USER CODE END*. Todo código de usuario no escrito entre estas regiones se perderá en caso de modificación del archivo ".io" y no podrá ser recuperado.

En la figura A.6 se puede observar a la izquierda, parte de los archivos generados por *STM32CubeMX* y, a la derecha, la plantilla del fichero *main.c* a modificar.

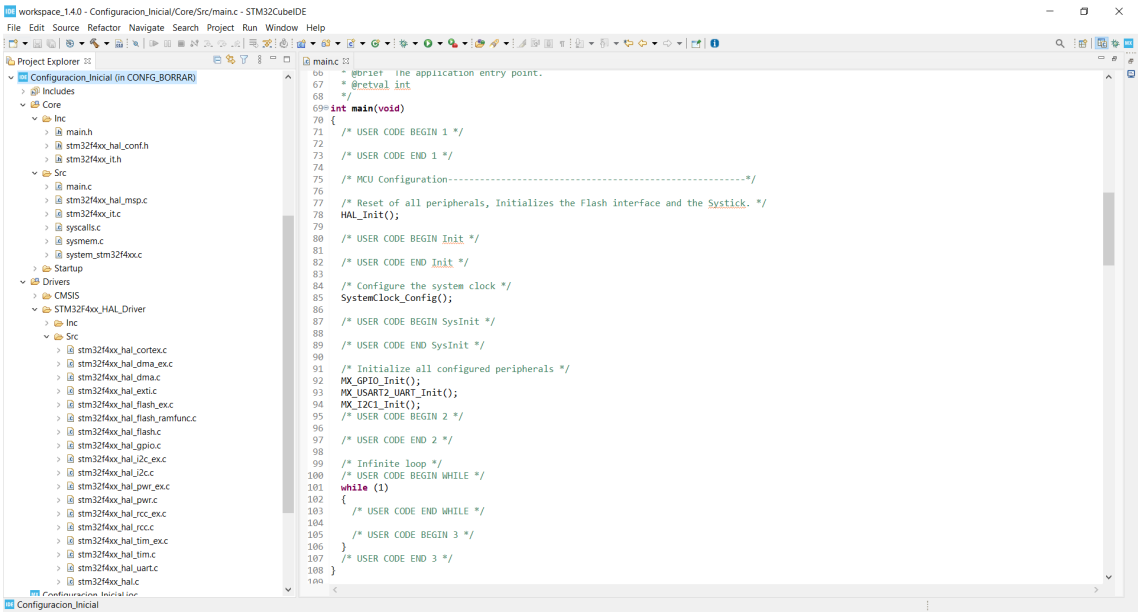

Figura A.6 Sistema de ficheros generados por *STM32CubeMX* y platilla del fichero *main.c*.

Resulta de particular interés indicar que en este fichero ya existen ciertas funciones creadas con el fin de inicializar el I^2C y la UART.

La primera llamada es *HAL_Init()*. Esta función está definida en *stm32f4xx_hal.c* y es la encargada de inicializar la librería HAL, a saber: resetear los periféricos, inicializar la interfaz de la Flash y configurar el temporizador de 24 bits de cuenta descendente *Systick* para generar interrupciones cada milisegundo. Cabe destacar que en este punto, el reloj no está aún configurado con lo que se utiliza como fuente de reloj el oscilador RC interno de 16MHz, HSI (High Speed Internal Oscillator).

Código A.1 Función HAL_Init().

```
HAL_StatusTypeDef HAL_Init(void)
{
  /* Configure Flash prefetch, Instruction cache, Data cache */
#if (INSTRUCTION_CACHE_ENABLE != 0U)
  __HAL_FLASH_INSTRUCTION_CACHE_ENABLE();
#endif /* INSTRUCTION_CACHE_ENABLE */

#if (DATA_CACHE_ENABLE != 0U)
  __HAL_FLASH_DATA_CACHE_ENABLE();
#endif /* DATA_CACHE_ENABLE */

#if (PREFETCH_ENABLE != 0U)
  __HAL_FLASH_PREFETCH_BUFFER_ENABLE();
```

```
#endif /* PREFETCH_ENABLE */

  /* Set Interrupt Group Priority */
  HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);

  /* Use systick as time base source and configure 1ms tick (default
      clock after Reset is HSI) */
  HAL_InitTick(TICK_INT_PRIORITY);

  /* Init the low level hardware */
  HAL_MspInit();

  /* Return function status */
  return HAL_OK;
}
```

La siguiente función es *SystemClock_Config()*, encargada de realizar, como su propio nombre indica, la configuración del reloj en base a los datos introducidos por el usuario en *STM32CubeMX*. Los datos de configuración del reloj se pueden observar en la Figura A.7 mientras que el código generado se puede consultar en el cuadro A.2

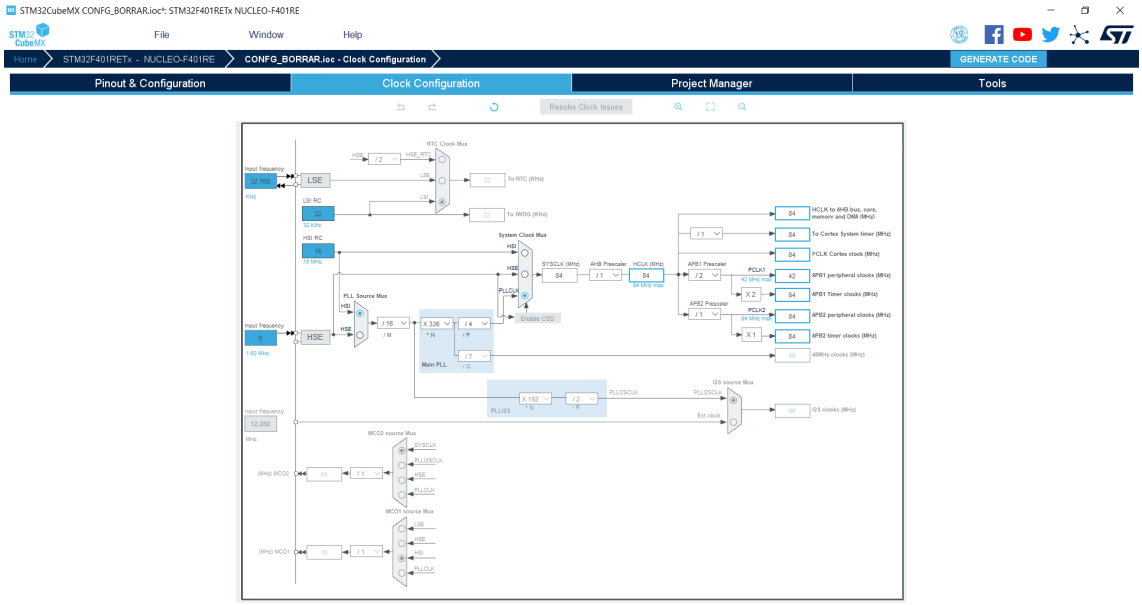

Figura A.7 Configuración del Reloj desde la herramienta *STM32CubeMX*.

Código A.2 Función SystemClock_Config().

```
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
```

```
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
  /** Initializes the RCC Oscillators according to the specified
     parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK
     )
  {
    Error_Handler();
  }
}
```

Después, se encuentra la función de inicialización de la USART mediante la llamada *MX_USART2_UART_Init()*, y la correspondiente a la inicialización del I^2C mediante la función *MX_I2C1_Init()*, tal y como se muestra en los códigos A.3 y A.4, respectivamente.

Estos códigos, al igual que el correspondiente a la configuración del reloj, también han sido generados por la herramienta de configuración gráfica *STM32CubeMX* en base a los parámetros introducidos por el usuario.

Código A.3 Función MX_USART2_UART_Init.

```
static void MX_USART2_UART_Init(void)
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
```

```
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Código A.4 Función MX_I2C1_Init().

```
static void MX_I2C1_Init(void)
{
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK
  {
    Error_Handler();
  }
}
```

A.1.2 Red neuronal

Para introducir la red neuronal, desde *STM32CubeMX*, se debe seleccionar "Software Packs → Select Components" o bien pulsar las teclas "Alt+o", tal y como se ve en la Figura A.8

Seguidamente, se abrirá la ventana *"Software Packs Component Selector"* donde el usuario tiene la posibilidad de seleccionar software adicional que puede instalar en su proyecto (Figura A.9). Para el desarrollo de este trabajo se ha instalado el pack *STMicroelectronics.X-CUBE-AI*.

Una vez instalado, desde esta misma ventana, se deberá hacer click sobre el paquete anterior, luego sobre *X-CUBE-AI* y después, marcar la casilla *Core*. El desplegable de *Application* se dejará tal y como aparece por defecto a *"Not selected"*. Estas configuraciones aparecen en la Figura A.9.

Después, en el menú de la izquierda de la ventana del proyecto se debe seleccionar *"Software Packs"* y aparecerá un desplegable en el cual se debe hacer click sobre ***STMicroelectronics.X-CUBE-AI.5.2.0***[1]. Aparecerá ahora a la derecha, bajo la pestaña ***Mode***, la casilla *Artificial Intelligence X-CUBE-AI*, la cual deberá ser seleccionada, apareciendo ahora bajo ***Configuration***, toda la configuración relativa a la red neuronal, tal y como se muestra en la Figura A.10

En este punto, el usuario debe dar *un nombre a la red*[2], seleccionar el tipo de fichero y cargar el modelo del cual se quiere hacer uso.

[1] La versión puede haber cambiado en función de cuando esté el lector revisando este documento.
[2] En este proyecto, el nombre dado a la red es *acc_neural_model*

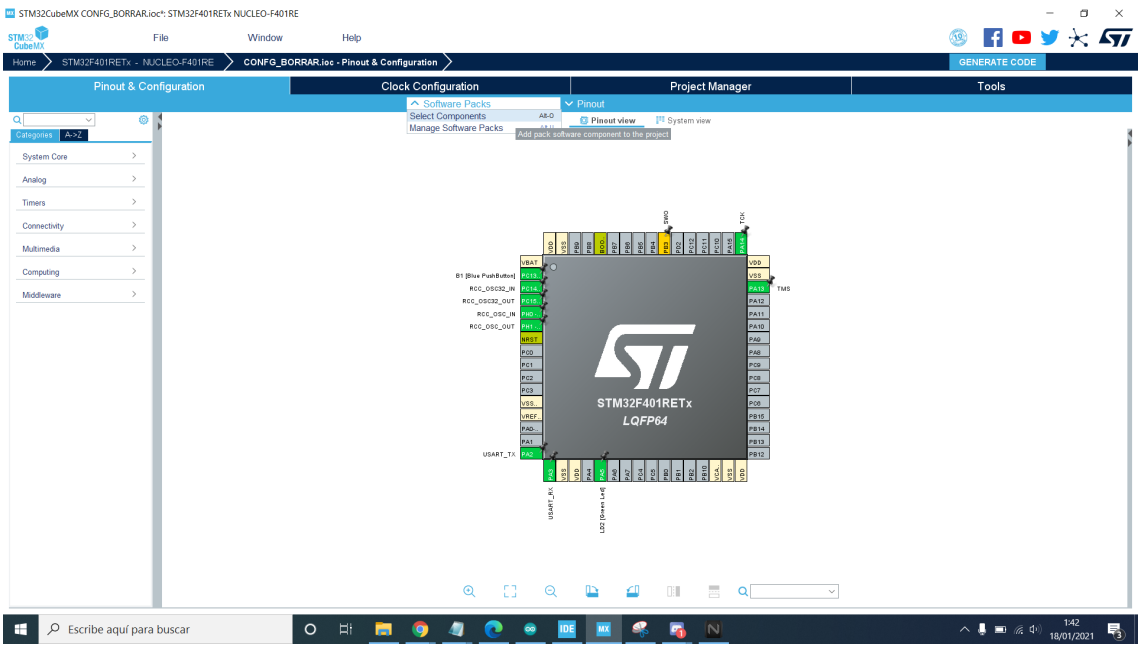

Figura A.8 Selección de componentes en *STM32CubeIDE*.

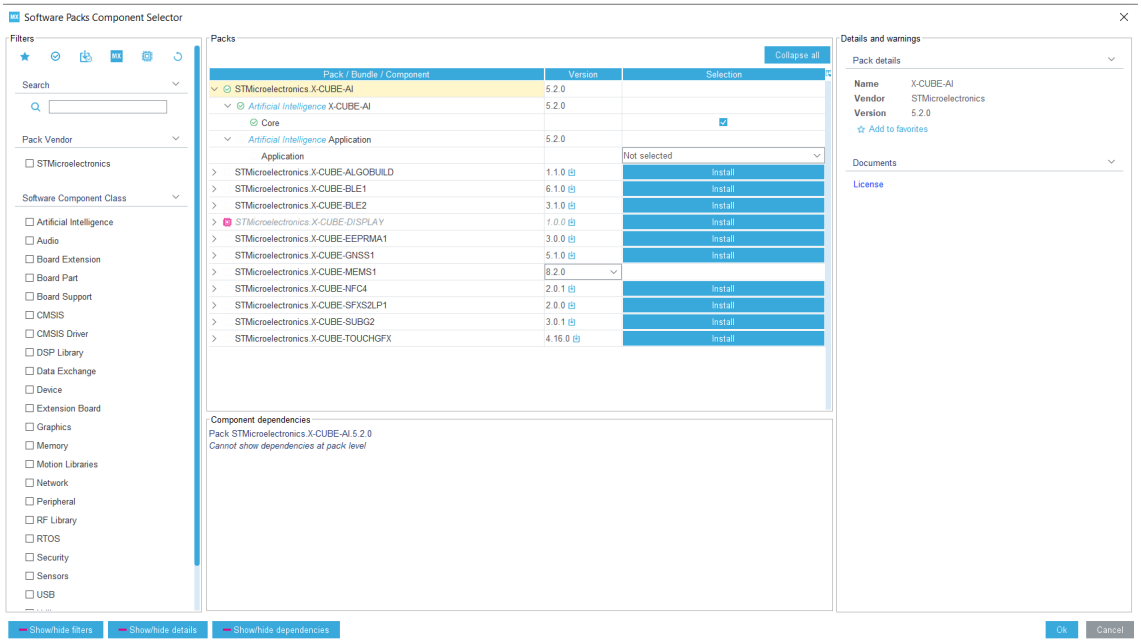

Figura A.9 Instalación del software *X-CUBE-AI* en *STM32CubeMX*.

Como se ha presentado durante el desarrollo de este trabajo de fin de grado, en primer lugar se hará uso del fichero *model.h5* [23] facilitado por STMicroelectronics, que contiene la estructura y pesos de un modelo entrenado para reconocer actividades humanas.

Una vez cargado el modelo, resulta de interés hacer click sobre ***Analyze***, lo cual brinda al usuario una visión general del modelo de la red neuronal (Figura A.11). De esta forma, se puede saber el espacio que ésta ocupa en el dispositivo, así como la cantidad de operaciones de multiplicar-acumular[3] empleadas.

[3] Más conocidas como multiply-and-accumulate operations (MAC)

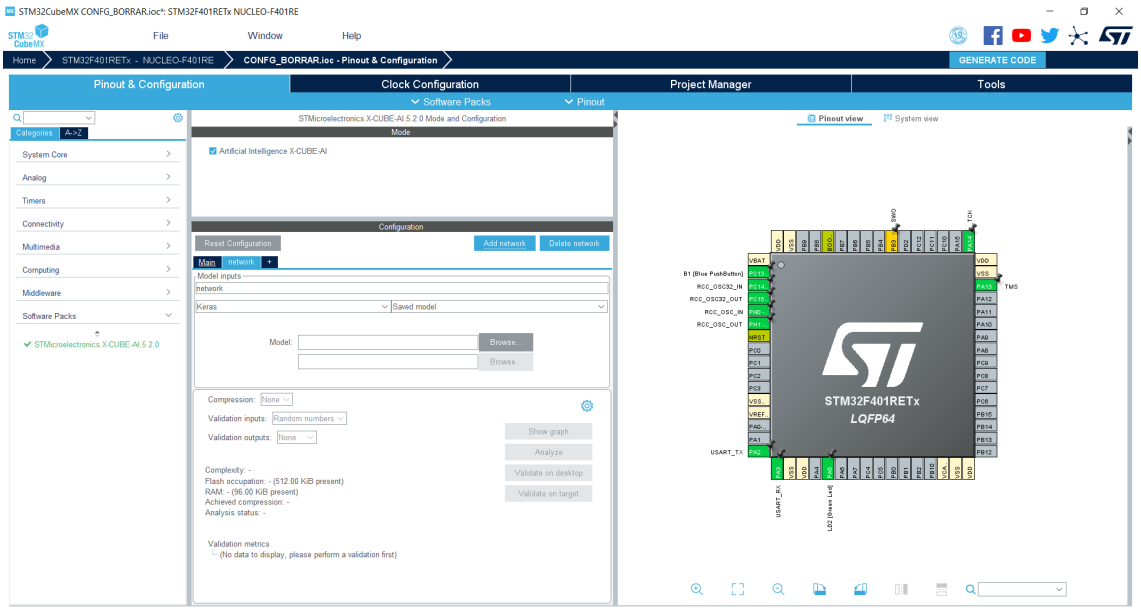

Figura A.10 Configuración del paquete *STMicroelectronics.X-CUBE-AI* en *STM32CubeMX*.

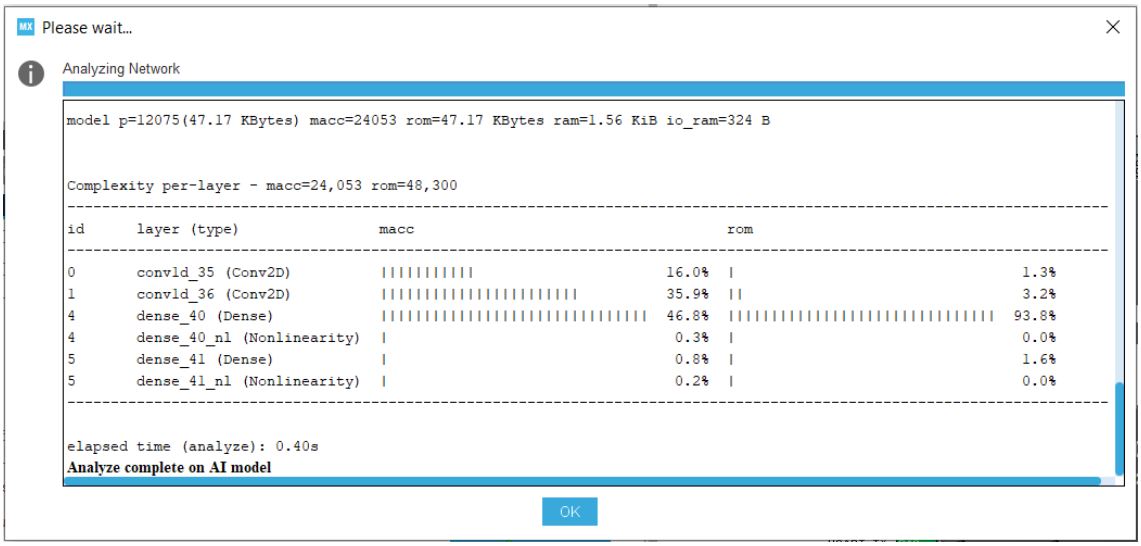

Figura A.11 Análisis del modelo de test de red neuronal empleado.

En concreto, tal y como se puede observar en la Figura A.12, el modelo utilizado emplea 24053 operaciones MACC y ocupa **poco más del 9 % de la memoria Flash** y **menos del 2 % de la memoria RAM**.

Adicionalmente, se puede utilizar la funcionalidad ***Validate on Desktop***, la cual ejecuta una cierta cantidad de tests sobre el modelo para comprobar el funcionamiento de la inferencia (Figura A.14).

Al pulsar sobre *GENERATE CODE*, *SMT32CubeMX* creará el código de configuración de la red en base al modelo proporcionado y preguntará al usuario si desea importar el código a *STM32CubeIDE*. Desde el editor, se puede ver cómo, en el menú de la izquierda (***Proyect Explorer***), ha aparecido una nueva carpeta asociada al proyecto en cuestión, ***X-CUBE-AI***. Dentro de ésta y en el interior a su vez de ***App***, se encuentran los ficheros de configuración generados. En concreto, cabe destacar el fichero ***network_data.h***[4], que contiene los pesos de la red.

[4] El nombre *network* es el generado por defecto en *STM32CubeMX* al introducir una red. Tanto el nombre de éste como el de los demás archivos generados cambiarán si el usuario lo modifica desde la ventana de configuración

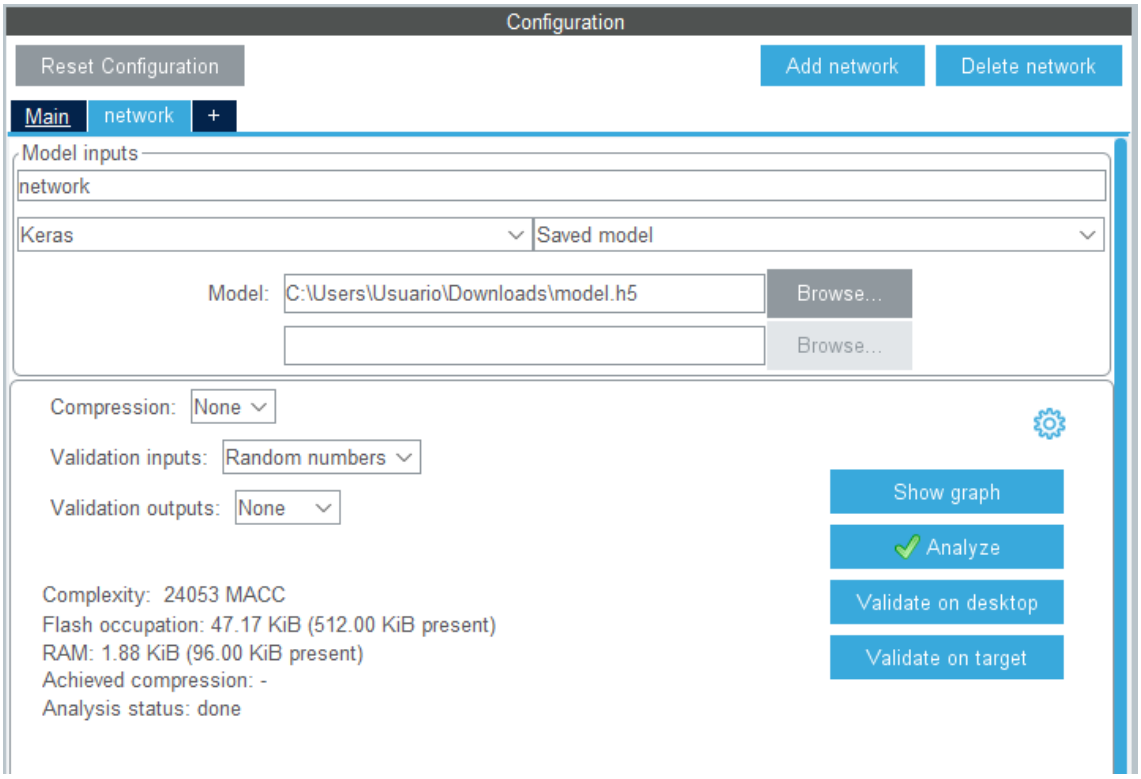

Figura A.12 Visión general del modelo de test de red neuronal empleado.

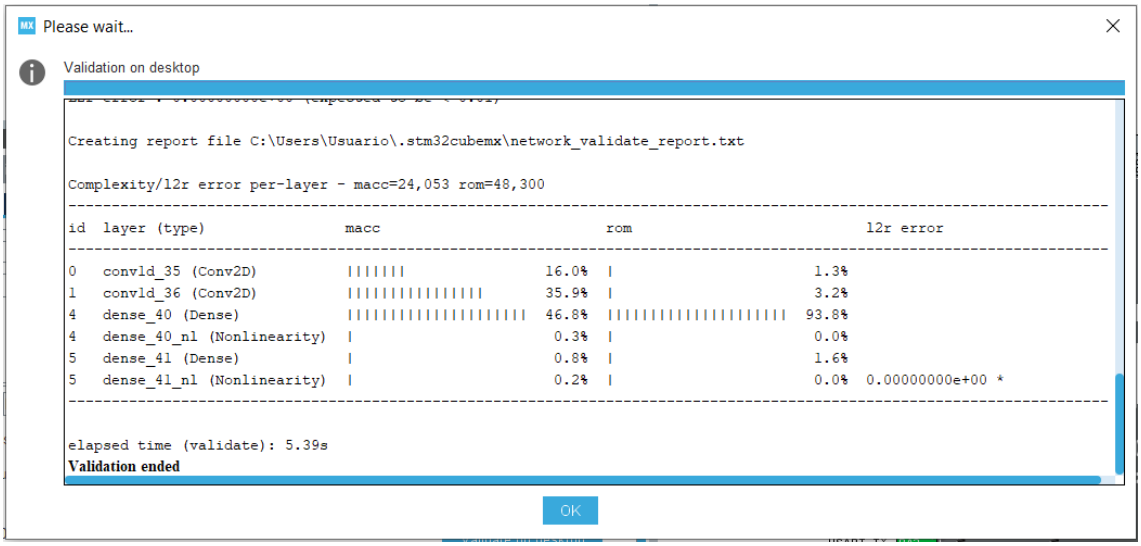

Figura A.13 Validación en Escritorio del modelo de test de red neuronal empleado.

Al no haber añadido ninguna aplicación (Figura A.9), no se creará código nuevo en el archivo ***main.c***, lo cual tiene sus ventajas, ya que se permite de esta forma la creación de código 100 % controlado y se otorga una mayor funcionalidad.

Una vez realizada la configuración inicial del entorno y presentadas las funciones generadas, se puede proceder a la utilización del *framework*.

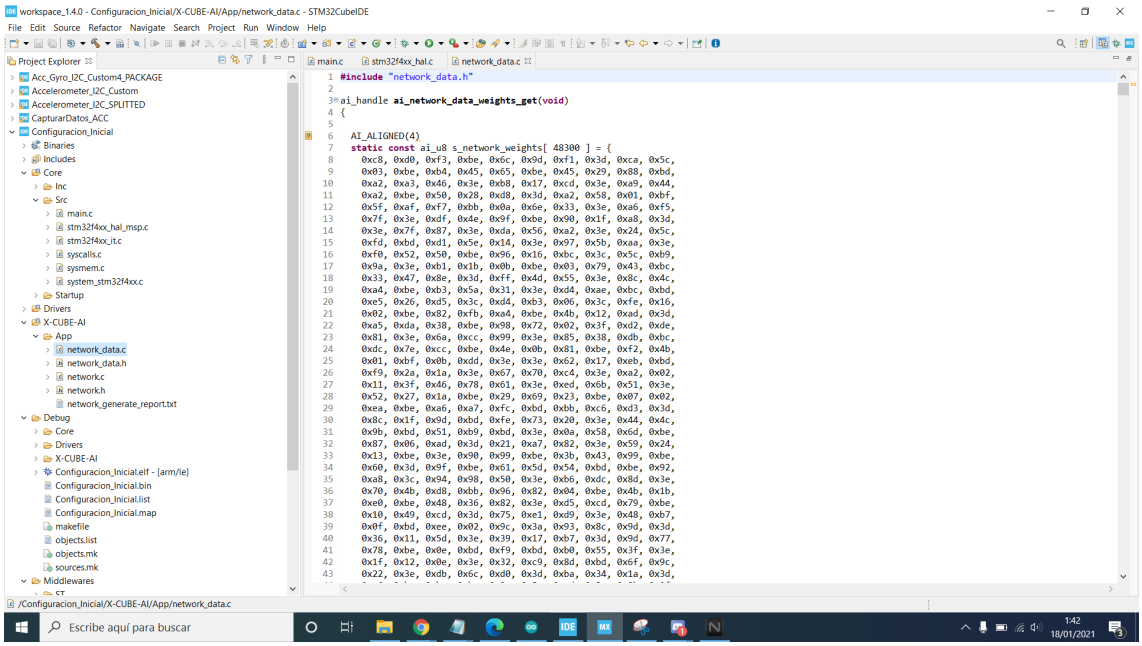

Figura A.14 Ficheros de configuración de la red neuronal y archivo *network_data.c*.

A.2 Posibles problemas

En este apartado se comentarán algunos de los problemas encontrados utilizando el software del ecocistema STM32Cube y se expondrá la solución a los mismos.

A.2.1 No se muestran valores de tipo *float*

Al usar GCC y Newlib-nano la implementación de entrada/salida de números en punto flotante está definida como *weak*. Si se desea usar "%f" se debe introducir el comando **"-u _printf_float"** en las banderas de GCC Linker. Ésto se puede realizar desde la configuración del proyecto, mediante los siguientes pasos:

1. Abrir las propiedades del proyecto haciendo click derecho sobre la carpeta del proyecto, **Project** → **Properties**.
2. Expandir la opción **C/C++ Build** y desplazarse hasta **Settings**
3. Hacer click en la pestaña **Settings**, expandir **MCU GCC Linker** y seleccionar **Miscellaneous**.
4. En **Other flags** añadir **-u _printf_float**.
5. Hacer click en **Apply and close**.

Una imagen con el resultado se muestra en la Figura A.15.

A.2.2 No se muestran por pantalla las llamadas *printf*

Por defecto, muchos microcontroladores no poseen el concepto de consola, aunque resulta de gran utilidad para poder realizar debuggs y mostrar información por pantalla. Si se intenta cargar alguna aplicación de ejemplo propia de *STM32CubeMX*, éstas hacen uso de *printf* aunque, al ejecutar los ejemplos no se muestra nada por pantalla porque dicha función no está bien definida. Para arreglar este inconveniente se ha adoptado la solución presentada en [9], cuyos pasos son los siguientes:

En primer lugar, deshabilitar el fichero *syscalls.c* ubicado en **Core** → **Src**.

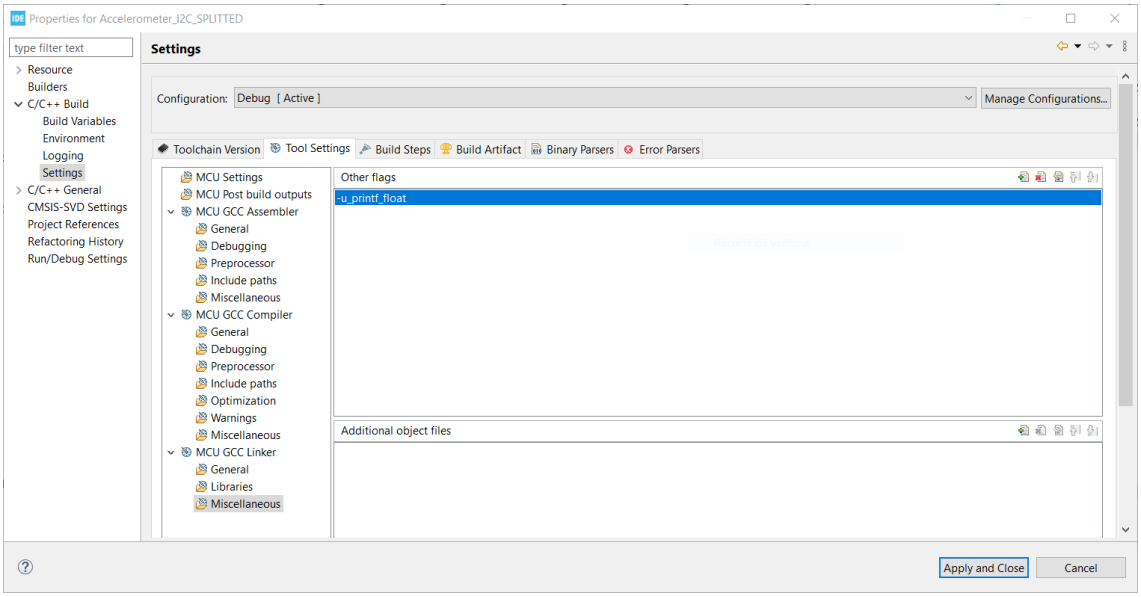

Figura A.15 Configuración del tipo *float* en STM32CubeIDE.

Esto es necesario porque los ficheros que se van a añadir ya definen muchas de las funciones existentes (aunque éstas no estén bien definidas) y, si no se deshabilita dicho fichero, se obtendrá el error de "múltiples definiciones" tal y como se puede ver en la Figura A.16.

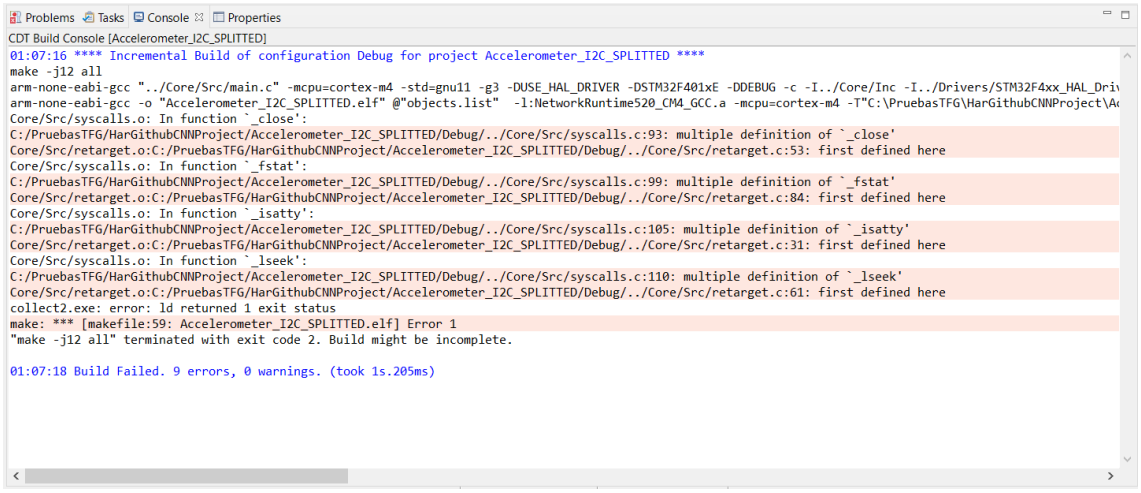

Figura A.16 Error de múltiples definiciones si no se deshabilita el fichero *syscalls.c*.

Para ello, se debe hacer click derecho sobre *syscalls.c*, seleccionar **Properties** y bajo **C/C++ Build** → **Settings** marcar la casilla **Exclude resource from build**. Estos pasos se muestran en la Figura A.17

Después, se deben añadir los ficheros *retarget.h* y *retarget.c* en **Core** → **Inc** y **Core** → **Src**, respectivamente. Por último, se debe añadir el include de *retarget.c* en el fichero *main.c*. Los códigos de *retarget.h* y *retarget.c* se pueden consultar a continuación.

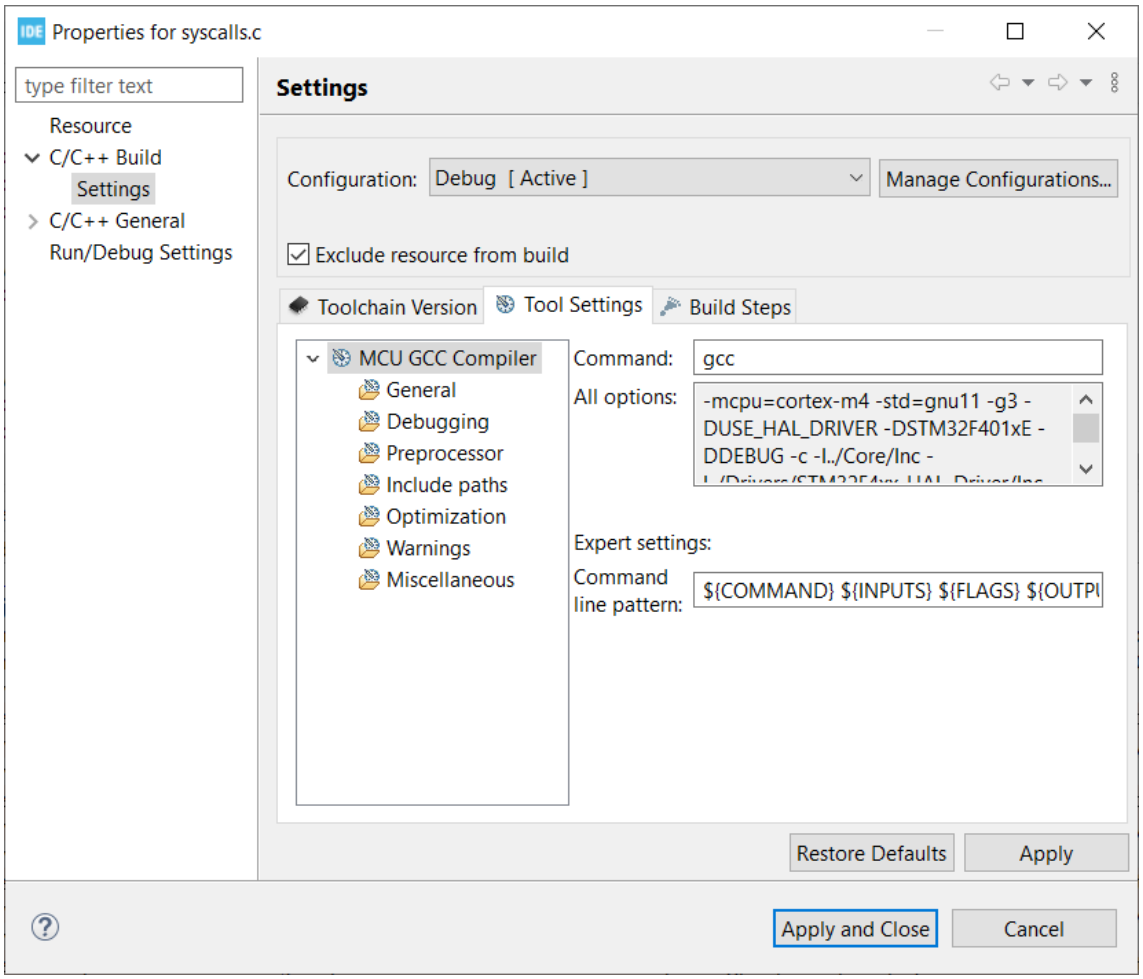

Figura A.17 Deshabilitar el fichero *syscalls.c*.

Código A.5 Archivo *retarget.h*.

```
// All credit to Carmine Noviello for this code
// https://github.com/cnoviello/mastering-stm32/blob/master/nucleo-
    f030R8/system/include/retarget/retarget.h

#ifndef _RETARGET_H__
#define _RETARGET_H__

#include "stm32f4xx_hal.h"
#include <sys/stat.h>

void RetargetInit(UART_HandleTypeDef *huart);
int _isatty(int fd);
int _write(int fd, char* ptr, int len);
int _close(int fd);
int _lseek(int fd, int ptr, int dir);
int _read(int fd, char* ptr, int len);
int _fstat(int fd, struct stat* st);

#endif //#ifndef _RETARGET_H__
```

Código A.6 Archivo *retarget.c*.

```
// All credit to Carmine Noviello for this code
// https://github.com/cnoviello/mastering-stm32/blob/master/nucleo-
    f030R8/system/src/retarget/retarget.c

#include <_ansi.h>
#include <_syslist.h>
#include <errno.h>
#include <sys/time.h>
#include <sys/times.h>
#include <limits.h>
#include <signal.h>
#include <../Inc/retarget.h>
#include <stdint.h>
#include <stdio.h>

#if !defined(OS_USE_SEMIHOSTING)

#define STDIN_FILENO 0
#define STDOUT_FILENO 1
#define STDERR_FILENO 2

UART_HandleTypeDef *gHuart;

void RetargetInit(UART_HandleTypeDef *huart) {
  gHuart = huart;

  /* Disable I/O buffering for STDOUT stream, so that
   * chars are sent out as soon as they are printed. */
  setvbuf(stdout, NULL, _IONBF, 0);
}

int _isatty(int fd) {
  if (fd >= STDIN_FILENO && fd <= STDERR_FILENO)
    return 1;

  errno = EBADF;
  return 0;
}

int _write(int fd, char* ptr, int len) {
  HAL_StatusTypeDef hstatus;

  if (fd == STDOUT_FILENO || fd == STDERR_FILENO) {
    hstatus = HAL_UART_Transmit(gHuart, (uint8_t *) ptr, len,
        HAL_MAX_DELAY);
    if (hstatus == HAL_OK)
```

```
      return len;
    else
      return EIO;
  }
  errno = EBADF;
  return -1;
}

int _close(int fd) {
  if (fd >= STDIN_FILENO && fd <= STDERR_FILENO)
    return 0;

  errno = EBADF;
  return -1;
}

int _lseek(int fd, int ptr, int dir) {
  (void) fd;
  (void) ptr;
  (void) dir;

  errno = EBADF;
  return -1;
}

int _read(int fd, char* ptr, int len) {
  HAL_StatusTypeDef hstatus;

  if (fd == STDIN_FILENO) {
    hstatus = HAL_UART_Receive(gHuart, (uint8_t *) ptr, 1, HAL_MAX_DELAY
        );
    if (hstatus == HAL_OK)
      return 1;
    else
      return EIO;
  }
  errno = EBADF;
  return -1;
}

int _fstat(int fd, struct stat* st) {
  if (fd >= STDIN_FILENO && fd <= STDERR_FILENO) {
    st->st_mode = S_IFCHR;
    return 0;
  }
  errno = EBADF;
  return 0;
}

#endif //#if !defined(OS_USE_SEMIHOSTING)
```

Apéndice B
Códigos principales

En este apéndice, se añade el código completo de los programas principales desarrollados en este proyecto y no comentados en apartados anteriores

B.1 main.c

Código B.1 Código completo del archivo main.c.

```
/* Includes
   ------------------------------------------------------------------*/
#include "main.h"

/* Private includes
   ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */


//Para la red neuronal //SI SE INVIERTE DE LUGAR CON LSM6DSO_CUSTOM.h
   FALLA
#include "ai_platform.h"
#include "acc_neural_model.h"
#include "acc_neural_model_data.h"

//Para el accelerometro
#include "LSM6DSO_CUSTOM.h"


//Includes Varios
#include "retarget.h" //To use Printf
#include<stdio.h>

/* USER CODE END Includes */

/* Private typedef
   -----------------------------------------------------------*/
```

```
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define
   ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro
   ------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables
   ------------------------------------------------------------*/
CRC_HandleTypeDef hcrc;

I2C_HandleTypeDef hi2c1;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim3;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes
   -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2C1_Init(void);
static void MX_CRC_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM3_Init(void);
/* USER CODE BEGIN PFP */
void Inicializar_LSM6DS0(I2C_HandleTypeDef *I2C_handler, float odr);
void Obtener_Ejes_LSM6DS0(I2C_HandleTypeDef *I2C_handler,
   SensorAxesFloat_t *acceleration, SensorFs_t fullScale);
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);

static void AI_Init(ai_handle w_addr, ai_handle act_addr);
static void AI_Run(float *pIn, float *pOut);
static uint32_t argmax(const float * values, uint32_t len);
/* USER CODE END PFP */
```

```
/* Private user code
   ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

//Parametros Red Neuronal
ai_handle acc_neural_model2;
float aiInData[AI_ACC_NEURAL_MODEL_IN_1_SIZE];
float aiOutData[AI_ACC_NEURAL_MODEL_OUT_1_SIZE];
uint8_t activations2[AI_ACC_NEURAL_MODEL_DATA_ACTIVATIONS_SIZE];
const char* activities[AI_ACC_NEURAL_MODEL_OUT_1_SIZE] = {
  "Parado", "Caminando", "Corriendo", "Saltando"
};

int flagInterrupcion=0;

/* USER CODE END 0 */

/**
  * @brief The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  uint32_t write_index = 0;
  uint16_t timer_value = 0;
  SensorAxesFloat_t accelerationFloat;

  /* USER CODE END 1 */

  /* MCU Configuration
     ---------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
     Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
```

```
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_I2C1_Init();
  MX_CRC_Init();
  MX_TIM1_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
  Inicializar_LSM6DS0(&hi2c1,952);

  RetargetInit(&huart2);

  printf("\r\n Iniciando Aplicacion de Reconocimiento...\r\n");

  AI_Init(ai_acc_neural_model_data_weights_get(), activations2);

  printf("Esperando 4 segundos para comenzar...\r\n");
  HAL_Delay(4000);
  printf("Haz un movimiento!\r\n");
  HAL_Delay(1000);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */

  //Iniciamos el timer
  HAL_TIM_Base_Start_IT(&htim1);

  while (1)
   {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    if(flagInterrupcion==1){

       flagInterrupcion=0;

       HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

       Obtener_Ejes_LSM6DS0(&hi2c1,&accelerationFloat,FS_MID);

           aiInData[write_index + 0] = (float) accelerationFloat.AXIS_X/
               4000.0f;
           aiInData[write_index + 1] = (float) accelerationFloat.AXIS_Y/
               4000.0f;
           aiInData[write_index + 2] = (float) accelerationFloat.AXIS_Z/
               4000.0f;

           /* Para imprimir el los datos del acelerometro */
//          printf("%.2f,%.2f,%.2f\r\n", aiInData[write_index + 0],
```

```
//                      aiInData[write_index + 1], aiInData[write_index +
    2]);


          write_index += 3;

          if (write_index == AI_ACC_NEURAL_MODEL_IN_1_SIZE) {
          write_index = 0;

          printf("Realizando inferencia...\r\n");
          HAL_TIM_Base_Start(&htim3);
          timer_value = __HAL_TIM_GET_COUNTER(&htim3);
          AI_Run(aiInData, aiOutData);
          timer_value = __HAL_TIM_GET_COUNTER(&htim3) - timer_value;

          printf("Tiempo realizando inferencia: %u us\r\n", timer_value)
              ;

           /* Output results */
          for (uint32_t i = 0; i < AI_ACC_NEURAL_MODEL_OUT_1_SIZE; i++)
              {
            printf("%8.6f ", aiOutData[i]);
          }

          uint32_t class = argmax(aiOutData,
              AI_ACC_NEURAL_MODEL_OUT_1_SIZE);
          printf(": %d - %s\r\n", (int) class, activities[class]);

      printf("\r\n");
      printf("Esperando 4 segundos para nuevo movimiento...\r\n");
      HAL_TIM_Base_Stop_IT(&htim1);
      HAL_Delay(4000);
      HAL_TIM_Base_Start_IT(&htim1);
      printf("Haz un movimiento!\r\n");

        }
    }
   }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
```

```
  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
  /** Initializes the RCC Oscillators according to the specified
     parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK
     )
  {
    Error_Handler();
  }
}

/**
  * @brief CRC Initialization Function
  * @param None
  * @retval None
  */
static void MX_CRC_Init(void)
{

  /* USER CODE BEGIN CRC_Init 0 */

  /* USER CODE END CRC_Init 0 */

  /* USER CODE BEGIN CRC_Init 1 */
```

```
  /* USER CODE END CRC_Init 1 */
  hcrc.Instance = CRC;
  if (HAL_CRC_Init(&hcrc) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN CRC_Init 2 */

  /* USER CODE END CRC_Init 2 */

}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief TIM1 Initialization Function
  * @param None
```

```
  * @retval None
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 84-1;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 38462-1;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) !=
      HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */

}

/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
```

```
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 84-1;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 65535-1;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) !=
      HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */

  /* USER CODE END TIM3_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */
```

```
  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
```

```
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */
static uint32_t argmax(const float * values, uint32_t len)
{
  float max_value = values[0];
  uint32_t max_index = 0;
  for (uint32_t i = 1; i < len; i++) {
    if (values[i] > max_value) {
      max_value = values[i];
      max_index = i;
    }
  }
  return max_index;
}

static void AI_Init(ai_handle w_addr, ai_handle act_addr)
{
  ai_error err;

  /* 1 - Create an instance of the model */
  err = ai_acc_neural_model_create(&acc_neural_model2,
      AI_ACC_NEURAL_MODEL_DATA_CONFIG);
  if (err.type != AI_ERROR_NONE) {
    printf("ai_network_create error - type=%d code=%d\r\n", err.type,
        err.code);
    Error_Handler();
  }

  /* 2 - Initialize the instance */
  const ai_network_params params = {
      AI_ACC_NEURAL_MODEL_DATA_WEIGHTS(w_addr),
      AI_ACC_NEURAL_MODEL_DATA_ACTIVATIONS(act_addr)
  };

  if (!ai_acc_neural_model_init(acc_neural_model2, &params)) {
    err = ai_acc_neural_model_get_error(acc_neural_model2);
    printf("ai_network_init error - type=%d code=%d\r\n", err.type, err.
        code);
    Error_Handler();
  }
}

static void AI_Run(float *pIn, float *pOut)
{
  ai_i32 batch;
```

```
  ai_error err;

  /* 1 - Create the AI buffer IO handlers with the default definition */
  ai_buffer ai_input[AI_ACC_NEURAL_MODEL_IN_NUM] =
      AI_ACC_NEURAL_MODEL_IN;
  ai_buffer ai_output[AI_ACC_NEURAL_MODEL_OUT_NUM] =
      AI_ACC_NEURAL_MODEL_OUT;

  /* 2 - Update IO handlers with the data payload */
  ai_input[0].n_batches = 1;
  ai_input[0].data = AI_HANDLE_PTR(pIn);
  ai_output[0].n_batches = 1;
  ai_output[0].data = AI_HANDLE_PTR(pOut);

  batch = ai_acc_neural_model_run(acc_neural_model2, ai_input, ai_output
      );
  if (batch != 1) {
    err = ai_acc_neural_model_get_error(acc_neural_model2);
    printf("AI ai_network_run error - type=%d code=%d\r\n", err.type,
        err.code);
    Error_Handler();
  }
}

//Funcion Callback Timer
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  //Verificamos que sea el timer 11 el que ha hecho saltar la
      interrupcion
  if(htim==&htim1){
    flagInterrupcion=1;
  }
}



/* USER CODE END 4 */

/**
  * @brief This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
      state */
  __disable_irq();
  while (1)
  {
  }
```

```
  /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT
/**
  * @brief Reports the name of the source file and the source line
     number
  *        where the assert_param error has occurred.
  * @param file: pointer to the source file name
  * @param line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
     line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
        line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

B.2 LSM6DS0_CUSTOM.h

Código B.2 Código completo del archivo LSM6DS0_CUSTOM.h.

```
/**
******************************************************************************
  * @file          : LSM6DS0_CUSTOM.h
  * @brief         : Header for LSM6DS0_CUSTOM.c file.
  *                  This file contains the definitions and values needed
  *                  to use the accelerometer LSM6DS0.
  *
  ******************************************************************************
**/

//INCLUIR LIBRERIAS ESTANDAR
#include <string.h>
#include <stdint.h>

//DEFINES PARA DEBUG DE ERRORES
#define REG_ACC 0x00 //buffer

//DEFINES ACELEROMETRO LSM6DS0
#define LSM6DS0_ADDR 0xD6
#define LSM6DS0_ACC_GYRO_OUT_X_L_XL 0X28
#define I2C_MEMADD_SIZE_8BIT 0x00000001U
```

```
#define I2C_EXPBD_Timeout 0x1000
#define LSM6DS0_ACC_GYRO_CTRL_REG6_XL 0X20
#define LSM6DS0_ACC_SENSITIVITY_FOR_FS_2G 0.061 /**< Sensitivity value
    for 2 g full scale [mg/LSB] */
#define LSM6DS0_ACC_SENSITIVITY_FOR_FS_4G 0.122 /**< Sensitivity value
    for 4 g full scale [mg/LSB] */
#define LSM6DS0_ACC_SENSITIVITY_FOR_FS_8G 0.244 /**< Sensitivity value
    for 8 g full scale [mg/LSB] */
#define LSM6DS0_ACC_SENSITIVITY_FOR_FS_16G 0.732 /**< Sensitivity value
    for 16 g full scale [mg/LSB] */
#define  LSM6DS0_ACC_GYRO_FS_XL_MASK 0x18

#define LSM6DS0_ACC_GYRO_IF_ADD_INC_ENABLE 0x04
#define LSM6DS0_ACC_GYRO_CTRL_REG8 0X22
#define  LSM6DS0_ACC_GYRO_IF_ADD_INC_MASK 0x04

#define CONVERT_MG_TO_MS2 0.00980665f

//Mascara para el ODR
#define LSM6DS0_ACC_GYRO_ODR_XL_MASK 0xE0

//ESTRUCTURAS
typedef struct
{
  int32_t AXIS_X;
  int32_t AXIS_Y;
  int32_t AXIS_Z;
} SensorAxes_t;

typedef struct
{
  float AXIS_X;
  float AXIS_Y;
  float AXIS_Z;
} SensorAxesFloat_t;

typedef enum
{
  LSM6DS0_ACC_GYRO_FS_XL_2g   = 0x00,
  LSM6DS0_ACC_GYRO_FS_XL_16g  = 0x08,
  LSM6DS0_ACC_GYRO_FS_XL_4g   = 0x10,
  LSM6DS0_ACC_GYRO_FS_XL_8g   = 0x18,
} LSM6DS0_ACC_GYRO_FS_XL_t; //SENSIBILIDAD

typedef enum
{
  FS_LOW,
  FS_MID,
  FS_HIGH,
  FS_EXTRA_HIGH
```

```
} SensorFs_t;

typedef enum
{
  LSM6DS0_ACC_GYRO_ODR_XL_10Hz  = 0x20,
  LSM6DS0_ACC_GYRO_ODR_XL_50Hz  = 0x40,
  LSM6DS0_ACC_GYRO_ODR_XL_119Hz   = 0x60,
  LSM6DS0_ACC_GYRO_ODR_XL_238Hz   = 0x80,
  LSM6DS0_ACC_GYRO_ODR_XL_476Hz   = 0xA0,
  LSM6DS0_ACC_GYRO_ODR_XL_952Hz   = 0xC0,
} LSM6DS0_ACC_GYRO_ODR_XL_t;
```

B.3 LSM6DS0_CUSTOM.c

Código B.3 Código completo del archivo LSM6DS0_CUSTOM.c.

```
/**
******************************************************************************
  * @file          : LSM6DS0_CUSTOM.c
  * @brief         : Driver for the accelerometer LSM6DS0.
  *                  This file contains the functions to use
  *                  the accelerometer LSM6DS0.
  ******************************************************************************
**/


#include "LSM6DS0_CUSTOM.h"
#include "main.h"

#include "stm32f4xx_hal_i2c.h"

//DECLARACIONES DE FUNCIONES
void Obtener_Ejes_LSM6DS0(I2C_HandleTypeDef *I2C_handler,
    SensorAxesFloat_t *acceleration, SensorFs_t fullScale);
void Inicializar_LSM6DS0(I2C_HandleTypeDef *I2C_handler, float odr);

//FUNCIONES
/**
 * @brief Get the LSM6DS0 accelerometer sensor axes
 * @param I2C_handler the device handle
 * @param acceleration pointer where the values of the axes are written
 * @param fullScale set the full scale value
 */
void Obtener_Ejes_LSM6DS0(I2C_HandleTypeDef *I2C_handler,
    SensorAxesFloat_t *acceleration, SensorFs_t fullScale)
{
  HAL_StatusTypeDef ret;
  uint8_t buf[13];
```

```
SensorAxes_t aux_acceleration;

  buf[0]=REG_ACC;
  ret = HAL_I2C_Master_Transmit(I2C_handler, LSM6DS0_ADDR, buf, 1,
      HAL_MAX_DELAY);
  if(ret != HAL_OK){
    printf("Error SCL!!\r\n");
  } else {
  ret = HAL_I2C_Master_Receive(I2C_handler, LSM6DS0_ADDR, buf, 2,
      HAL_MAX_DELAY);
  if(ret != HAL_OK){
      printf("Error SDA!!\r\n");
    } else {

  int16_t dataRaw[3];
  uint8_t regValue[6] = {0, 0, 0, 0, 0, 0};

  uint8_t i, j, k;
  uint8_t numberOfByteForDimension;

  numberOfByteForDimension = 6 / 3;

  k = 0;
  for (i = 0; i < 3; i++ )
  {
      for (j = 0; j < numberOfByteForDimension; j++ )
      {
        HAL_I2C_Mem_Read(I2C_handler, LSM6DS0_ADDR,
            LSM6DS0_ACC_GYRO_OUT_X_L_XL + k, I2C_MEMADD_SIZE_8BIT, &
            regValue[k], 1, I2C_EXPBD_Timeout);
         k++;
      }
  }

  dataRaw[0] = ((((int16_t)regValue[1]) << 8) + (int16_t)regValue[0]);
  dataRaw[1] = ((((int16_t)regValue[3]) << 8) + (int16_t)regValue[2]);
  dataRaw[2] = ((((int16_t)regValue[5]) << 8) + (int16_t)regValue[4]);


/* Seleccionar Escala */
  float sensitivity = 0;

  switch( fullScale )
  {
    case FS_LOW:
      sensitivity = LSM6DS0_ACC_SENSITIVITY_FOR_FS_2G;
      break;
    case FS_MID:
      sensitivity = LSM6DS0_ACC_SENSITIVITY_FOR_FS_4G;
      break;
```

```
      case FS_HIGH:
        sensitivity = LSM6DS0_ACC_SENSITIVITY_FOR_FS_8G;
        break;
      case FS_EXTRA_HIGH:
        sensitivity = LSM6DS0_ACC_SENSITIVITY_FOR_FS_16G;
        break;
      default:
        sensitivity = -1.0f;
    }

  /* Calcular valores de salida */

  aux_acceleration.AXIS_X = ( int32_t )( dataRaw[0] * sensitivity ) ;
  aux_acceleration.AXIS_Y = ( int32_t )( dataRaw[1] * sensitivity ) ;
  aux_acceleration.AXIS_Z = ( int32_t )( dataRaw[2] * sensitivity ) ;

  acceleration->AXIS_X = ( float ) aux_acceleration.AXIS_X ;
  acceleration->AXIS_Y = ( float ) aux_acceleration.AXIS_Y ;
  acceleration->AXIS_Z = ( float ) aux_acceleration.AXIS_Z ;

     }
   }
}


/**
 * @brief Initialize LSM6DS0 accelerometer sensor axes
 * @param I2C_handler the device handle
 * @param odr Number of the output data rate to be configured
 */
void Inicializar_LSM6DS0(I2C_HandleTypeDef *I2C_handler, float odr){

    LSM6DS0_ACC_GYRO_ODR_XL_t new_odr;
    uint8_t value;

    new_odr = ( odr <= 10.0f ) ? LSM6DS0_ACC_GYRO_ODR_XL_10Hz
              : ( odr <= 50.0f ) ? LSM6DS0_ACC_GYRO_ODR_XL_50Hz
              : ( odr <= 119.0f ) ? LSM6DS0_ACC_GYRO_ODR_XL_119Hz
              : ( odr <= 238.0f ) ? LSM6DS0_ACC_GYRO_ODR_XL_238Hz
              : ( odr <= 476.0f ) ? LSM6DS0_ACC_GYRO_ODR_XL_476Hz
              :                     LSM6DS0_ACC_GYRO_ODR_XL_952Hz;

      HAL_I2C_Mem_Read(I2C_handler, LSM6DS0_ADDR ,
          LSM6DS0_ACC_GYRO_CTRL_REG6_XL, I2C_MEMADD_SIZE_8BIT, &value, 1,
           I2C_EXPBD_Timeout);

    value &= ~LSM6DS0_ACC_GYRO_ODR_XL_MASK;
    value |= new_odr;
```

```
    HAL_I2C_Mem_Write(I2C_handler, LSM6DSO_ADDR ,
        LSM6DSO_ACC_GYRO_CTRL_REG6_XL, I2C_MEMADD_SIZE_8BIT, &value, 1,
        I2C_EXPBD_Timeout);

}
```

B.4 CapturarDatos.c

Código B.4 Código completo del archivo CapturarDatos.c.

```
/* Includes
   ------------------------------------------------------------------*/
#include "main.h"

/* Private includes
   ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

//Para el accelerometro
#include "LSM6DSO_CUSTOM.h"


//Includes Varios
#include "retarget.h" //To use Printf
#include<stdio.h> //To use FILE

/* USER CODE END Includes */

/* Private typedef
   -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define
   ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro
   -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables
   ---------------------------------------------------------*/
CRC_HandleTypeDef hcrc;
```

```
I2C_HandleTypeDef hi2c1;

TIM_HandleTypeDef htim1;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes
   -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2C1_Init(void);
static void MX_CRC_Init(void);
static void MX_TIM1_Init(void);
/* USER CODE BEGIN PFP */
void Obtener_Ejes_LSM6DS0(I2C_HandleTypeDef *I2C_handler,
   SensorAxesFloat_t *acceleration, SensorFs_t fullScale);
void Inicializar_LSM6DS0(I2C_HandleTypeDef *I2C_handler, float odr);
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);
/* USER CODE END PFP */

/* Private user code
   ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */


int flagInterrupcion=0;

SensorAxesFloat_t accelerationFloat;
SensorAxesFloat_t accelerationAux;

/* USER CODE END 0 */

/**
  * @brief The application entry point.
  * @retval int
  */
int main (void) {
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration
     --------------------------------------------------------*/
```

```
/* Reset of all peripherals, Initializes the Flash interface and the
   Systick. */
HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
MX_I2C1_Init();
MX_CRC_Init();
MX_TIM1_Init();
/* USER CODE BEGIN 2 */
Inicializar_LSM6DSO(&hi2c1,952);
RetargetInit(&huart2);

printf("\r\n Iniciando programa de captura de datos...\r\n");
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
int mode=1;

  if(mode==1){
  HAL_TIM_Base_Start_IT(&htim1);
  while(1){
    if(flagInterrupcion==1){
    Obtener_Ejes_LSM6DSO(&hi2c1,&accelerationFloat,FS_LOW);
    accelerationAux.AXIS_X = (float) accelerationFloat.AXIS_X/4000.0f;
    accelerationAux.AXIS_Y = (float) accelerationFloat.AXIS_Y/4000.0f;
    accelerationAux.AXIS_Z = (float) accelerationFloat.AXIS_Z/4000.0f;
    printf("%.2f,%.2f,%.2f\r\n", accelerationAux.AXIS_X,
        accelerationAux.AXIS_Y, accelerationAux.AXIS_Z);
    flagInterrupcion=0;
    }
  }
  }
  if(mode==2){
    int mediciones=0;
    printf("Esperando 6 segundos para comenzar");
    HAL_Delay(6000);
```

```
      HAL_TIM_Base_Start_IT(&htim1);
      while(mediciones<1000){
      if(flagInterrupcion==1){
      flagInterrupcion=0;
      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
      Obtener_Ejes_LSM6DS0(&hi2c1,&accelerationFloat,FS_LOW);
      accelerationAux.AXIS_X = (float) accelerationFloat.AXIS_X;///
         4000.0f;
    accelerationAux.AXIS_Y = (float) accelerationFloat.AXIS_Y;/// 4000.0
       f;
      accelerationAux.AXIS_Z = (float) accelerationFloat.AXIS_Z;///
         4000.0f;
      printf("%.2f,%.2f,%.2f\r\n", accelerationAux.AXIS_X,
    accelerationAux.AXIS_Y, accelerationAux.AXIS_Z);
      mediciones++;
        }
     }
  }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

  /* USER CODE END 3 */

}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
  /** Initializes the RCC Oscillators according to the specified
     parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
```

```
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK
      )
  {
    Error_Handler();
  }
}

/**
  * @brief CRC Initialization Function
  * @param None
  * @retval None
  */
static void MX_CRC_Init(void)
{

  /* USER CODE BEGIN CRC_Init 0 */

  /* USER CODE END CRC_Init 0 */

  /* USER CODE BEGIN CRC_Init 1 */

  /* USER CODE END CRC_Init 1 */
  hcrc.Instance = CRC;
  if (HAL_CRC_Init(&hcrc) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN CRC_Init 2 */

  /* USER CODE END CRC_Init 2 */

}

/**
  * @brief I2C1 Initialization Function
```

```
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
```

```
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 84-1;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 38462-1;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) !=
      HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
```

```
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */
//Funcion Callback Timer

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
```

```
  //Verificamos que sea el timer 11 el que ha hecho saltar la
      interrupcion
  if(htim==&htim1){
    flagInterrupcion=1;
  }
}



/* USER CODE END 4 */

/**
  * @brief This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
      state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT
/**
  * @brief Reports the name of the source file and the source line
      number
  *        where the assert_param error has occurred.
  * @param file: pointer to the source file name
  * @param line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
      line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
         line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Índice de Figuras

Índice de Códigos

Bibliografía

[1] *Github stmicroelectronics*, *https://github.com/STMicroelectronics/stm32ai/raw/master/AI_resources/HAR*, Diciembre 2020.

[2] *Google colaboratory*, *https://colab.research.google.com/notebooks/intro.ipynb*, Diciembre 2020.

[3] *Scikit-learn: Machine learning in python*, *https://scikit-learn.org/stable/*, Diciembre 2020.

[4] *Stm32l4 discovery kit iot node (b-l475eiot01a)*, *https://www.st.com/en/evaluation-tools/b-l475e-iot01a.html*, Diciembre 2020.

[5] *Tensorflow: An end-to-end open source machine learning platform*, *https://www.tensorflow.org/*, Octubre 2020.

[6] Crunchbase, *Discover innovative companies and the people behind them*, *https://www.crunchbase.com/organization/edge-impulse/company_financials*, Enero 2021.

[7] The Apache Software Foundation, *Apache freemarker*, *https://freemarker.apache.org/*, Enero 2021.

[8] Eric Hamilton, *What is edge computing: The network edge explained*, *https://www.cloudwards.net/what-is-edge-computing/*, 2018.

[9] Shawn Hymel, *How to use printf on stm32*, *https://shawnhymel.com/1873/how-to-use-printf-on-stm32/*, Noviembre 2020.

[10] Edge Impulse, *Edge impulse: Making things smarter*, *https://www.edgeimpulse.com/*, Noviembre 2020.

[11] IC Insights Inc., *Mcus expected to make modest comeback after 2020 drop*, *https://www.icinsights.com/news/bulletins/MCUs-Expected-To-Make-Modest-Comeback-After-2020-Drop--/*, 2020.

[12] Keras, *The python deep learning api*, *https://keras.io/*, Diciembre 2020.

[13] Ji Lin, Wei-Ming Chen, John Cohn, Chuang Gan, and Song Han, *Mcunet: Tiny deep learning on iot devices*, Annual Conference on Neural Information Processing Systems (NeurIPS), 2020.

[14] Arm Ltd., *Cmsis-pack documentation*, *https://www.keil.com/pack/doc/CMSIS/Pack/html/index.html*, Enero 2021.

[15] NUCLEO-F401RE, *Stm32 nucleo-64 development board with stm32f401re mcu*, https://www.st.com/en/evaluation-tools/nucleo-f401re.html, Octubre 2020.

[16] José María Rodríguez, *El ecosistema de tensorflow para programadores principiantes y expertos en machine learning: cursos, lenguajes y edge computing*, https://www.genbeta.com/desarrollo/ecosistema-tensorflow-para-programadores-principiantes-expertos-machine-learning-cursos-lenguajes-edge-computing, Enero 2021.

[17] VentureBeat SF, *Why tinyml is a giant opportunity*, https://venturebeat.com/2020/01/11/why-tinyml-is-a-giant-opportunity/, 2021.

[18] Software STM32CubeIDE, *Integrated development environment for stm32*, https://www.st.com/en/development-tools/stm32cubeide.html, Octubre 2020.

[19] Software STM32CubeMX, *Stm32cube initialization code generator*, https://www.st.com/en/development-tools/stm32cubemx.html, Octubre 2020.

[20] STMicroelectronics, *Human activity recognition*, https://colab.research.google.com/github/STMicroelectronics/stm32ai/blob/master/AI_resources/HAR/Human_Activity_Recognition.ipynb, Diciembre 2020.

[21] _____, *User manual um1725 - description of stm32f4 hal and low-layer drivers*, https://www.st.com/resource/en/user_manual/dm00105879-description-of-stm32f4-hal-and-ll-drivers-stmicroelectronics.pdf, Diciembre 2020.

[22] _____, *User manual um2739 - how to create a software pack enhanced for stm32cubemx using stm32 pack creator tool*, https://www.st.com/resource/en/user_manual/dm00716911-how-to-create-a-software-pack-enhanced-for-stm32cubemx-using-stm32-pack-creator-tool-stmicroelectronics.pdf, Enero 2021.

[23] _____, *Getting started with stm32cube.ai and motion sensing on the stm32l4 iot node*, https://wiki.st.com/stm32mcu/wiki/Getting_started_with_STM32Cube.AI_and_motion_sensing_on_the_STM32L4_IoT_node, Noviembre 2020.

[24] _____, *Stm32f401xd stm32f401xe arm® cortex®-m4 32b mcu+fpu, 105 dmips, 512kb flash/96kb ram, 11 tims, 1 adc, 11 comm. interfaces*, https://www.st.com/resource/en/datasheet/stm32f401re.pdf, Noviembre 2020.

[25] _____, *Ai expansion pack for stm32cubemx*, https://www.st.com/en/embedded-software/x-cube-ai.html, Octubre 2020.

[26] _____, *Lsm6ds0 - inemo inertial module: 3d accelerometer and 3d gyroscope*, https://www.st.com/resource/en/datasheet/lsm6ds0.pdf, Octubre 2020.

[27] _____, *Sensor and motion algorithm software expansion for stm32cube*, https://www.st.com/en/embedded-software/x-cube-mems1.html, Octubre 2020.

[28] _____, *User manual um2526 - getting started with x-cube-ai expansion package for artificial intelligence (ai)*, https://www.st.com/resource/en/user_manual/dm00570145-getting-started-with-xcubeai-expansion-package-for-artificial-intelligence-ai-stmicroelectronics.pdf, Octubre 2020.

[29] _____, *Version 4 legacy sensor and motion algorithm software expansion for stm32cube*, https://www.st.com/en/embedded-software/x-cube-mems1-v4.html, Octubre 2020.

[30] Simon Tatham, *Putty software*, *https://www.putty.org/*, Octubre 2020.

[31] tinyML® Asia, *Big opportunities for tinyml applications: Everywhere and always-on*, *https://www.tinyml.org/asia2020/slides/tinyMLAsia2020d1p3-Gousev.pdf*, 2020.

[32] Harvard University, *Tiny machine learning (tinyml)*, *https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning*, 2020.

[33] X-NUCLEO-IKS01A1, *Motion mems and environmental sensor expansion board for stm32 nucleo*, *https://www.st.com/en/ecosystems/x-nucleo-iks01a1.html*, Octubre 2020.